NTNU

Norwegian University of
Science and Technology

HEEE - Handler for Exceptionally Exceptional Exceptions

Author(s)

Vegard Solheim
Olafur Johan M. Trollebø
Lars Walter Westby
Aleksander Steen

Bachelor in Software Engineering
20 ECTS
Department of Computer Science and Media Technology
Norwegian University of Science and Technology,

18.05.2016

Supervisor(s) Ivar Farup

Sammendrag av Bacheloroppgaven

Tittel:	**HEEE - Handler for Exceptionally Exceptional Exceptions**
Dato:	18.05.2016
Deltakere:	Vegard Solheim Olafur Johan M. Trollebø Lars Walter Westby Aleksander Steen
Veiledere:	Ivar Farup
Oppdragsgiver:	Innit AS
Kontaktperson:	Joakim Jøreng, joakim@innit.no
Nøkkelord:	Feilhåndtering, Javascript, Laravel, PHP, AngularJS
Antall sider:	189
Antall vedlegg:	8
Tilgjengelighet:	Åpen

Sammendrag:

All programvare har feil, hvorav mange programmeringsspråk uttrykker de som unntak. Innit AS drifter programvare for deres kunder, og foreløpig håndterer alle feilrapporter fra deres systemer manuelt. De ønsker derfor å få utviklet et system for å motta, prosesserer og organisere disse feilrapportene. Vårt system, H3E, vil utføre de oppgavene, og fra dette datasettet tilby en utviklerorientert grafisk oversikt over alle feilrapporter, så de kan fokusere på å rette opp feil.

Summary of Graduate Project

Title:	**HEEE - Handler for Exceptionally Exceptional Exceptions**
Date:	18.05.2016
Authors:	Vegard Solheim Olafur Johan M. Trollebø Lars Walter Westby Aleksander Steen
Supervisor:	Ivar Farup
Employer:	Innit AS
Contact Person:	Joakim Jøreng, joakim@innit.no
Keywords:	Exceptions, Javascript, Laravel, PHP, AngularJS
Pages:	189
Attachments:	8
Availability:	Open

Abstract: All software has errors, which in many programming languages are expressed as exceptions. Innit AS is hosting software for its customers, and currently manually has to handle exception reports from its systems. They therefore wished to have developed a system to receive, processes and organise those exception reports from a multitude of externally hosted systems. Our system, H3E, will handle those tasks, and from that data provide a developer-centric graphical overview of all reports received for developers to focus on fixing issues

Preface

We would like to thank all the people that has helped us develop this software. Ivar Farup for providing excellent critique in the writing of this thesis report plus assistance in any development related matter we had. Joakim Jøreng for being available to answer all our questions and thoughts at any time of the day and giving us the opportunity to develop this software for his company. Eivind Johansen for providing extremely helpful feedback in regard to our graphical presentation. Øivind Kolloen and Mariusz Nowostawski providing valuable design pointers, guidelines and best practices in API creation plus technical assistance whenever we had issues with AngularJS. Rune Hjelsvold for giving us in-depth knowledge of database design and critical help during the development of our database triggers and events. Last but not least, all the exceptional and high quality teachers we've had over the last three years.

Contents

List of Figures

1 Introduction

Software does not run flawlessly. Problems happen, and in many programming languages these problems cause exceptions. Such exceptions can be caught in the code, handling the problem and allowing the software to keep running, or they can go uncaught and cause further trouble for the system. Exceptions can be caused by tiny inconsequential information notifications or connection failures – events that are expected to happen – to potentially devastating system errors and information state failures. Thus exceptions are both expected and a large part of modern software development, and reports about exceptions can tell a lot about how a software module works, or does not work.

The company Innit AS is a medium-size software development and hosting company with 21 employees[1] as of spring 2016. As Innit hosts software for their customers, they are responsible for keeping software running as intended. When exceptions happen that are not caught and handled in the software itself, Innit receives reports from the software about what went wrong, what caused it, which parts of the software or modules were affected, any data input just prior to the exception, and more. This information is gathered, stored, and analysed in order to maintain and improve the affected software.

Formerly Innit did this work all manually. All their systems were sending their exception report to an IRC-channel that Innit hosted, where they had to manually parse all the information and attempt to gleam any potential trends from the large amounts of information reported. Even though they only had a few dozen exceptions occurring in a typical day, each report is easily be several hundred words of densely packed information.

Because reading this information and in particular attempting to find any trends manually is both quite difficult and very time consuming, Innit have requested the development of an automated system to receive and manage these exception reports.

1.1 Project Inception

Already before the bachelor project descriptions were released, our group, which had worked together for almost a year at that time, was contacted by Joakim, our main contact person at Innit. He offered to let us have this project, as they required a stable and competent team to develop a system that could be used reliably during daily operations. We were quite interested, and upon receiving the project description a few days later, readily agreed it was a good project for us.

During our contact with the employer we quickly found the project was, while concrete enough to quickly get a feeling for what to do, flexible enough to give us real freedom of choice for development tools, methodology, and scope. Innit knew very well that this project would require us to learn new frameworks and tools, and were quite willing to assist in this process. This is also why the recommended AngularJS and the PHP-framework Laravel were chosen, since they have in-house experience with those and could provide guidance and assistance if needed, see Section 4.1.

1.2 Project Description

The system had the working title of HEEE, short for Handler for Exceptionally Exceptional Exceptions. The system's main functionality is to receive, parse and organise exception reports, and provide a user friendly GUI and a multitude of graphical statistics about the exceptions occurring.

The systems functionality is to:

- Receive, parse and store exception reports in a fitting format.
- Display exception statistics in a graphical format.
- Potential extra features:
  - Automatically propose new issues for adding to the JIRA issue tracking system.
  - A hybrid mobile application for Android and iOS, for receiving exception report notifications and check the exception statistics.
  - End-of-the-month-report, giving a summary of how the past month has been.

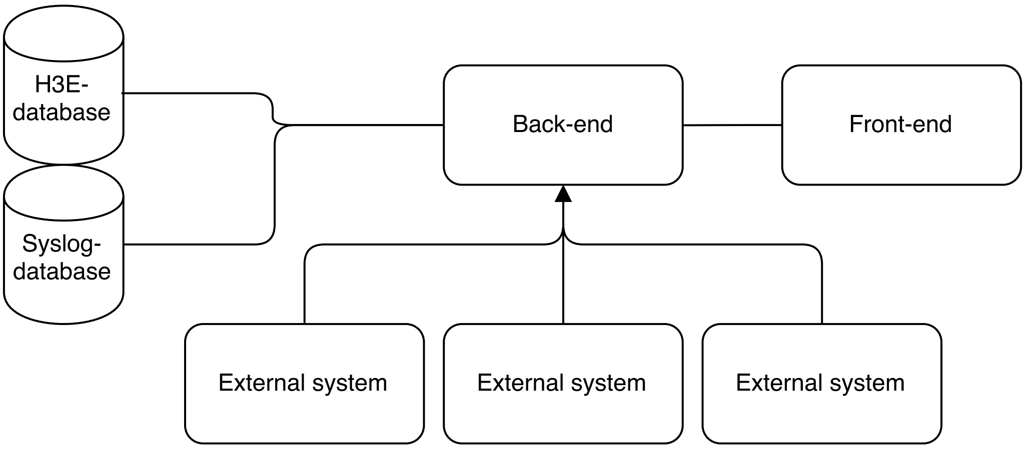


Figure 1: Overview of the system.

To elaborate on the features mentioned. The system's back-end will receive exception reports from hosted systems running on Innit's servers. Our system then parses and organise the reports in a local database. The back-end supplies an API for statistics about the exception reports and meta-information about them.

The system's front-end is a separate system that requests information through the back-end's API, and through integration with a third-party tool creates graphical diagrams and displays them. The system will be running on a local Debian server at Innit, and only requires communication between the front-end and the back-end, both being on the local network.

Regarding the extra features; they are not within the explicit scope of the project, but the employer has expressed interest in those features if we have sufficient time to develop them. However they are not mandatory features the system needs before delivery. The end-of-the-month-report is a report that, as the name implies, will be dynamically created and show a summary of the past month with several categories of statistics, and perhaps small trivia about the system and developers; e.g. who fixed the most reports.

The name HEEE – pronounced H-triple-E – is the short version for the working name

Handler for Exceptionally Exceptional Exceptions. In lack of a better name, we chose it for the humour value. While we intended to only use H3E as a working title and come up with a proper name during the development process, both we and Innit became so familiar with that it simply stuck. In the end we decided to keep it, to no objection from neither Innit nor our supervisor.

1.2.1 Purpose

The purpose of the project is to relieve the developers at Innit of having to manually parse through the exception reports that are constantly coming in, thus increases the effectiveness of receiving and managing exception reports. The system is intended to make it far easier for them to see longer-term trends and potential correlations that would be very difficult to spot by manually reading through all the reports. In addition the operations staff at Innit will get a subsystem for viewing all syslog-related information.

1.2.2 Product Objectives

With this project, we wish to develop a system that:

- Can receive, parse, and store exception reports in a suitable data model.
- Can display exception statistics in a variety of ways, increasing understanding of potential problems.
- Allow Innit operations staff to browse and filter syslog-formatted log files in an efficient manner.

1.3 Learning Objectives

The intention of the Bachelor Thesis is to get experience working on a larger project over a longer period of time, where we organise, plan, develop and deliver on our own. While we are delegating tasks and thus not learning equally, all team members are expected to learn at least some of all the points in the list below.

- The Javascript- and PHP-based frameworks AngularJS and Laravel, see Section 4.1.
- Use of professional development tools like task runners, code analysis- and project management tools, see Section 6.1.
- Integrate with existing tools for graphical representation of statistics.
- Develop a completely modular system, where modules can easily be replaced.
- Use a Kanban-based continuous development process.
- Work for a real employer with other expectations than in an academic setting.
- Create a product that will be used in an commercial setting.
- Handle exception reports and report data in a professional manner.
- Work with a operations-defined standard format for data storage, such as syslog, see Section 4.5 [2].

In addition to this, the team members must learn and understand how to work in a systematic process and to create a scientific assessment of this work, together with the capability of self-reflection based on the work that is done. This work must be documented and presented in an scientifically sound way, with consideration of scientific ethics and the impact such work might have on individuals and society as a whole [3].

1.4 Audience

Target Audience

The target audience here is mainly the developers and operations staff at Innit AS. The system will be running locally on a server at Innit, and the information displayed statically on screens in the development and operations offices. Thus we do not have to take into account that non-technical audience will be viewing or using the software. It is also important to keep in mind this system will not be much interacted with, but mainly available to give a fast overview what is currently happening.

Report Audience

The main audience for this project report are those already related to the project in any way, that being staff of the employer, our supervisor, the examiners, and potentially any future employer. As this is very specific development for one employer's particular use cases, we see it as less likely that a larger audience will be interested in viewing the development process.

This report assumes of the reader a general knowledge of web development, software development processes, and typical IT-terminology such as front-end and back-end, HTTP-protocol, APIs, and more, though particular abbreviations and terminologies can be gleamed from the glossary, see Section 1.8. In addition some knowledge about software development tools is expected, though in-depth experience is not necessary. The main tools we have used are listed in Section 6.1. Parts of the report will be easier to understand given some basic knowledge of AngularJS and Laravel and their structure, though the report is fully understandable without such knowledge.

1.5 Field of Study

This project will introduce the team to several fields and concepts within software development, some of which we have had cursory introductions to over the past few years of study, and some of which will be entirely new to us. The main areas we will be are as follows, but not limited to:

- RESTful API-design
- Application development for Debian
- Relational database design and use of an ORM-tool
- Web application front-end and back-end frameworks
- Web application development

Scope Limitations

Within the broad areas of study mentioned in Section 1.5, some limitations are set in order to make the scope of the project manageable.

- The system will only be running on Debian, no development has to be done for Windows or Macintosh. If we have sufficient time, we might create a mobile application. See the project description in Section 1.2.
- The creation and display of statistic graphics will be done through integration of an existing third-party tool/package.
- Since the application will only be running locally, authentication is not necessary.

Both the reporting interface and the back-end's API will only be available over local network.

- We will only have to supply a format for data from external systems. Innit will take care of rewriting the reporting in their systems.
- The employer supplies read-only access to a syslog database for any infrastructure reports.

1.6 Project Organisation

Project leader – Vegard Solheim As the project leader Vegard carries responsibility for paperwork like meeting notes, writing and formatting of plans and the thesis report, etc. In addition he will be mainly working on back-end development and database design.

Design/Technical staff – Aleksander Steen As technical staff Aleksander carries responsibility for the database system, syslog-related tasks, the ShareLatex-stack where we write this thesis, code analysis tools, commit hooks, and more. In addition he is working on design and development of the front-end, and syslog-related functionality on the back-end.

Back-end – Olafur Johan Trollebøe Olafur has the main responsibility for back-end development, and handling the database models and Eloquent.

Front-end/technical staff – Lars Walter Westby Lars has main responsibility for front-end development and design, and has the responsibility for configuring tools such as task runners, code analysis, and such.

Supervisor – Ivar Farup As our supervisor, Ivar is the point of contact for any issues or information we require regarding the project as a whole, and will as an external third-party give feedback and discuss our ideas, suggestions, and the format and content of the thesis.

Employer – Innit, represented by Joakim Jøreng The employer defines the system requirements and provides technical considerations and feedback on the development process. They are represented by Joakim, which is our main contact at Innit, together with other staff of the employer that are relevant for a meeting.

1.7 Academic Background

All four team members have been studying Bachelor in Software Engineering at Norwegian University of Science and Technology. While we have all followed the same study program, some choices of optional courses have been different, and we have differed greatly in preferred topics to focus on. Thus we have arrived here after two and a half year with similar knowledge and experience, but quite different areas of expertise – a fact that displays itself in the areas of responsibility in the project organisation, see Section 1.6

Lars, and to some degree Aleksander, have by far the most experience designing and building user-facing functionality. Aleksander has had extra courses in systems management, and has long experience setting up and running several different server systems.

Olafur and Vegard have both focused more on the systems development, and have had additional coursework in software security and secure development.

Relevant to the project, we all have some experience with PHP, Javascript and mobile development as well as HTML and CSS, briefly using some frameworks in the aforementioned languages. A general knowledge of and experience with relational models and databases is in place as well. None of us have had optional courses that are particularly relevant to the project, though the knowledge and experience from mostly unrelated courses can still be partially transferred. We all have some experience with projects and project management, though Vegard is the most proficient in this area. We have never worked on a project as large as this before, but we are all confident it will not provide any insurmountable problems.

1.8 Glossary

Throughout the report there will be some acronyms and references to people and concepts. In this section we will list what we hope are all the potentially ambiguous or otherwise unexplained pieces of the text.

HEEE. Handler for Exceptionally Exceptions. Working title of the system.

Product or System. Context dependent, but generally refers to the entire HEEE system.

Employer. Innit AS, represented by Joakim Jøreng, see Section 1.6

Supervisor. Ivar Farup, see Section 1.6

Front-end. The part of the system handling display, graphics and user interaction. Written in Javascript/AngularJS.

Back-end. The part of the system receiving exception reports, communicating with the database, and supplying an API to the front-end. Written in PHP/Laravel.

API. Generally refers to the RESTful API the back-end supplies to the front-end to request information.

Toaster. Refers to a little pop-up in the lower right part of the screen, giving the user information about events or similarly.

Route Refers to a path in an URL, and generally refers to a path in the back-end's API. For instance, in 'test.com/v1/statistics/application/:applicationID', the '/statistics/application/' part is a route.

Main Route The same as a regular route, but is the first part of the path is the main route. For instance, in 'api.test.com/v1/statistics/application/:applicationID', the '/statistics/' part alone is the main route.

1.9 Document Structure

This thesis is structured in large part after sections in the Software Development Life Cycle, even though the development itself is fully agile and not remotely similar to the traditional waterfall development process. The report is structured as follows:

Introduction This chapter presents the project and scope to the reader, introduces the team, employer and supervisor, and provides other general information about the project.

Project Management This chapter introduces the reader to the development process and major choices made early in the project

Requirements This chapter provides more detailed information about the project's requirements.

Technical Design This chapter provides an overview of our choice of languages and frameworks, conventions to use, how we interpret the requirements, and architectural and database design.

Implementation This chapter contains the result of our architectural and design decisions and implementation choices.

Development Process This chapter contains the larger part of information regarding our tools and development process.

Testing and Quality Assurance This chapter provides in-depth information about how we did testing, retrieved feedback, and ascertained quality, performance and maintainability.

Discussion This chapter contains discussion about the project as a whole, comparing objectives with the resulting product, team self-assessment, and self-critique and alternative choices.

Conclusion This chapter concludes the report in a brief conclusion of the project and report.

2 Project Management

2.1 Development Methodology

Within the team we already had some experience with continuous development, i.e. Kanban, along with some Scrum. The group composition, internal dynamics, knowledge and motivation all favoured an agile methodology, and we have found e.g. a sprint-based methodology to be unnecessary for forcing progress. Because the employer was uncertain about the required amount of work to develop this system, they had some additional functionality that could be added in case there was too little work for a full bachelor thesis. This unknown factor greatly favoured an agile methodology, together with our negligible experience with this kind of system to estimate sufficiently well for Scrum sprints, thus rendering the sprints quite impractical.

However Kanban could potentially provide too few restrictions and guidelines, and cause us to not have sufficiently good information feedback loops and cause a lacklustre review process on our work. This can be at least partially mitigated by strict rules on version control and code reviews, see Section 6.1 about version control. In addition, since it is not limited to sprints or similarly, we might get too few deadlines to work by, and thus not get work done in time. This can be mitigated by planning with set deadlines and demo-dates. In that way we can get the benefits of a sprint-esque work flow, while having the flexibility of Kanban, though without the benefit of a set backlog per sprint. Lastly, without actual sprints it could become more difficult to learn from previous work tasks and mis-estimation. This can be partially mitigated by paying particular attention to previous estimates during review sessions, both internally to the group and during meetings with our supervisor.

Despite the possible drawbacks, for the reasons mentioned above we unanimously decided to use Kanban as described by Henrik Kniberg and Mattias Skarin as our development methodology [4]. We incorporated some features from Scrum in an adapted form, most notably weekly and bi-weekly meetings with respectively the supervisor and the employer, see Section 2.3. Neither our supervisor nor the employer had any objections to our choice of methodology, given that we set some deadlines for when particular demos should be ready and displayed.

The team readily agreed soon after receiving the initial requirements description that we would use an agile development methodology. While the outline of the system was sufficiently clear and did not warrant any particular methodology, the employer expressed a wish to avoid any kind of waterfall-esque development process. In addition the employer explicitly wanted us to have as full freedom as practically feasible to test several different approaches.

2.2 Task Board

The task board is central to a Kanban process, being one of the three main points highlighted by Kniberg [4]. While a physical task board might be preferable for a small team like ours, we did not have a single room available for us to use during the entire semester.

For task tracking we started out with Trello. It is a simple and much used board-style task tracker that we have some experience with from earlier projects. Its reliable, easy to use, free, and with only four people on the team we reckoned that a more advanced tracker was not needed.

When the Professional Programming course began, which two of the four team members of us are following, it was recommended for reasons of learning benefits to use the Atlassian stack, with Bitbucket for repository and NTNU-hosted JIRA and Confluence for task tracking and documentation respectively. See Section 6.1 about the tool usage. The Atlassian stack was recommended due to it being much used in enterprise development, and using such a rather advanced tool would be advantageous in learning real professional development.

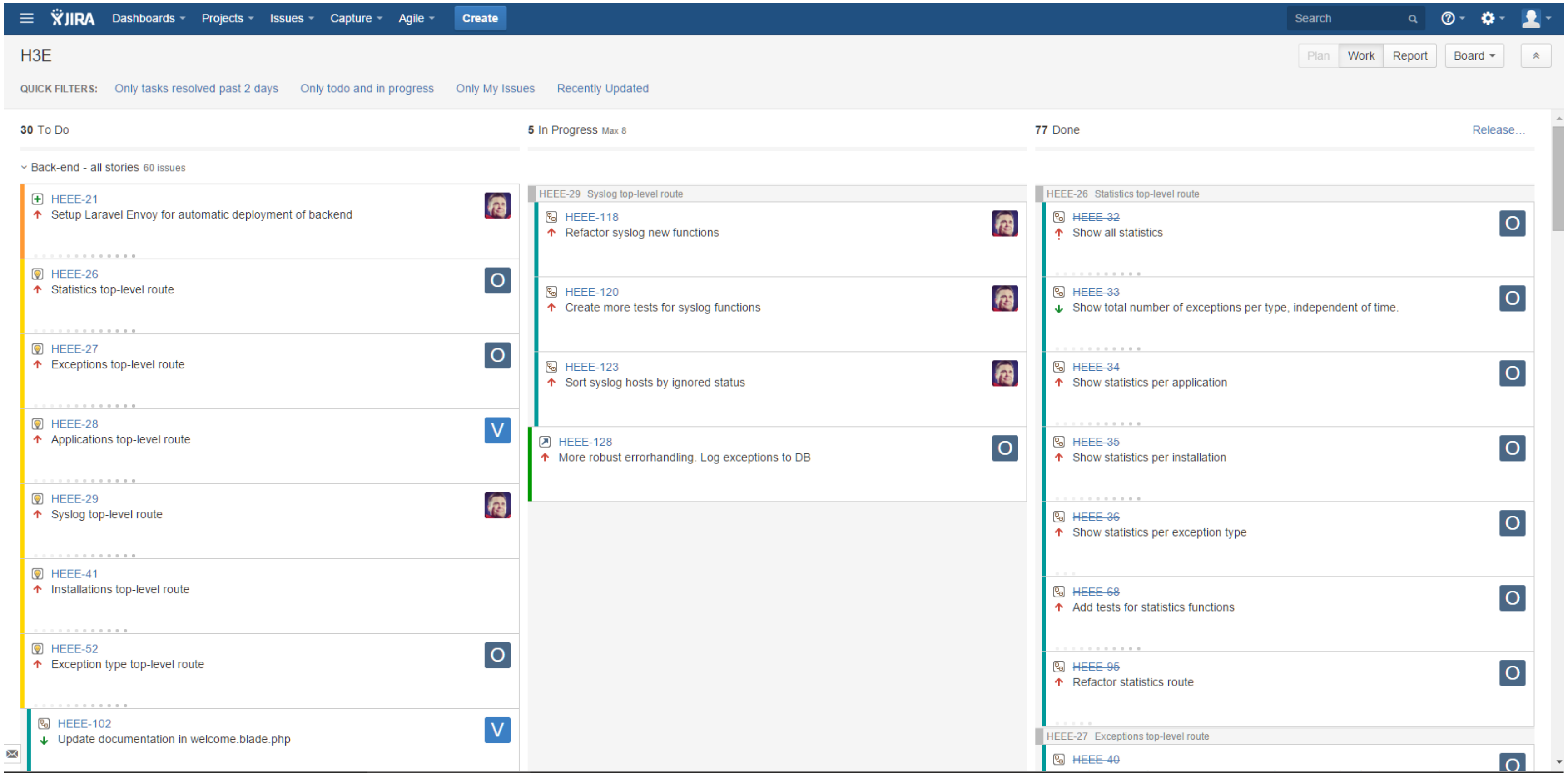
Swim lanes are a common feature in Scrum and Kanban boards where one can, for instance, have all the tasks of one person to always be shown in a single horizontal lane, separated from other people's tasks. For instance all tasks assigned to Alice are gathered in one swim lane at the top, separated horizontally from all Bob's tasks, which are all gathered in between Alice's and Charlie's horizontal swim lanes. We wished to have swim lanes per person and sorted hierarchically, i.e. first separated by front-end, back-end, or mobile system, then separated by user stories. However combining the per-person swim lanes with a such a hierarchical setup was not supported in JIRA's Kanban board, and we had to manage without the per-person swim lanes.

Figure 2 shows a screen shot of the JIRA task board. We used epics, essentially a hierarchical top-level item underneath which one can group tasks for better visibility, to delimit tasks relating to the front-end, back-end, mobile client, documentation, and the informational website required by the university. For subdividing work within each epic we used user stories. The user stories represented larger chunks of each system, modules if you like. Most, if not all, regular tasks that actually represents work are created as sub-tasks to each user story, in order to delimit. Thus we could have one person work on tasks within the Exception-story in the back-end epic, while another person worked on the Application-story in the back-end epic, and they would both know exactly what the other person is working on, and could easily avoid messing up each others files and code.

This epic to user story to sub-task structure effectively created a three-layer hierarchy for regular planned work and features. Other types of work however, e.g. bug fixes and certain improvements, were often made as regular tasks side by side with the user stories, and not as sub-tasks to a user story.

In the to-do column can be seen stories and tasks specifically for the back-end epic. The front-end and the mobile application epics both have a similar structure, while the documentation and informational website epics follow a more straightforward structure without as much subdividing; mostly because the tasks are more interconnected, and there is less work in total.

As can be seen in Figure 2, we went with only the three default columns: 'to-do', 'in progress' and 'done'. Because we had explicit rules that another team member do a code review before a task is completed and code is pushed to the master branch, we did not have a need for a 'testing' column; the branch one worked on sent a pull request to



H3E

30 To Do

Back-end - all stories 60 issues

HEEE-21
Setup Laravel Envoy for automatic deployment of backend

HEEE-26
Statistics top-level route

HEEE-27
Exceptions top-level route

HEEE-28
Applications top-level route

HEEE-29
Syslog top-level route

HEEE-41
Installations top-level route

HEEE-52
Exception type top-level route

HEEE-102
Update documentation in welcome.blade.php

5 In Progress Max 8

HEEE-29 Syslog top-level route

HEEE-118
Refactor syslog new functions

HEEE-120
Create more tests for syslog functions

HEEE-123
Sort syslog hosts by ignored status

HEEE-128
More robust errorhandling. Log exceptions to DB

77 Done

HEEE-26 Statistics top-level route

~~HEEE-32~~
Show all statistics

~~HEEE-33~~
Show total number of exceptions per type, independent of time.

~~HEEE-34~~
Show statistics per application

~~HEEE-35~~
Show statistics per installation

~~HEEE-36~~
Show statistics per exception type

~~HEEE-68~~
Add tests for statistics functions

~~HEEE-95~~
Refactor statistics route

HEEE-27 Exceptions top-level route

~~HEEE-40~~

Figure 2: Screenshot of the JIRA task board, taken February 15th.

master, requesting somebody to do a code review and approve the pull request.

We had some issues with JIRA not allowing one to archive completed stories and tasks, bar fully deleting them. This caused the Kanban board to expand very quickly, with the completed-column taking up a great lot of space vertically. We never found a good solution to this, and are amongst the reasons we – while using JIRA throughout most of the project – are not very satisfied with it.

2.3 Meetings

2.3.1 Meetings with Employer

We scheduled to have meetings with Innit every other Wednesday, though this did not happen in reality. Due to the substantial travel distance, we instead had two meetings over Skype, and several text-only 'meetings' where we simply sent our contact at Innit some questions and received answers later on, at times with some discussion around alternatives and asking advice. We had a total of 5 physical meetings with Innit; the first in December to discuss the project and initial requirements, two later in January to elaborate on the system requirements and discuss proposals of ours, one in March to demonstrate the at the time completed features of the front-end and back-end. The last meeting was at the end of April, where we showcased the feature completed product – feature completed being that no more features or functionality will be added, the only tasks left being refactoring and other under-the-hood changes that does not change the functionality of the product.

Typically for the meetings we had a list of questions which we sequentially discussed our way through. The list was usually sent a few days before the meeting so Innit could prepare. In addition to those questions, we had a retrospective discussion of the progress, any particular problems we met that they could provide assistance on, and the backlog for the next few weeks. The employer did not explicitly control the backlog, but offered advice on what to prioritise, how to do some tasks, and some tools that could be useful.

The group leader was responsible for taking notes and writing a meeting summary afterwards. These summaries can be seen in Appendix C

2.3.2 Meetings with Supervisor

Meetings with our supervisor was scheduled for every Tuesday morning. With the meetings scheduled every week it was far easier to cancel a meeting we felt we did not need, than having to set up additional ones. Before the first few meetings we sent the newest draft for the project plan / thesis report, so that he could read through and comment on it. After the first few meetings we simply made an account for Ivar on the ShareLatex stack we were writing the thesis on, so that he could read and comment directly in the document itself. This greatly improved the feedback loop for both parties, as he could check the document at a moments notice if needed, did not depend on us to send the most recent report – which if sent early could be quite outdated by the time of the meeting, or if sent late would give him little time to prepare –, and he could himself choose how to read it, whether in the LaTeXsource code or the generated PDF-document.

Usually we spent the entire hour set off for the meeting, having a partial retrospective and partial planning meeting, with the supervisor asking critical questions to ascertain our continued progress. Several times we displayed very work-in-progress demos in order to receive constructive criticism and discuss potential issues with a given design decision.

2.3.3 Group Meetings

Group meetings usually were not planned beforehand; they were quite ad hoc whenever we realised a group decision had to be made. This was not a conscious decision of ours; rather it was part of utilising agile flexibility to the fullest, combined with us doing most work sitting together on the campus.

We had a core working time of 09.00 to 15.00, generally to 16.00 on all days, and even if a person was not available in person, all group members were available in a common Facebook-chat we keep running continuously.

3 Requirements

3.1 Use-case overview

In this section we will display the functionality requirements of the system in the form of several use case diagram. The first diagram, Figure 3, is a fully generalised use case diagram displaying the required main functionality in a user-oriented way. The two secondary use case diagrams, Figures 4 and 5 are more technically oriented and displays initial ideas for separating parts of the system into modules.

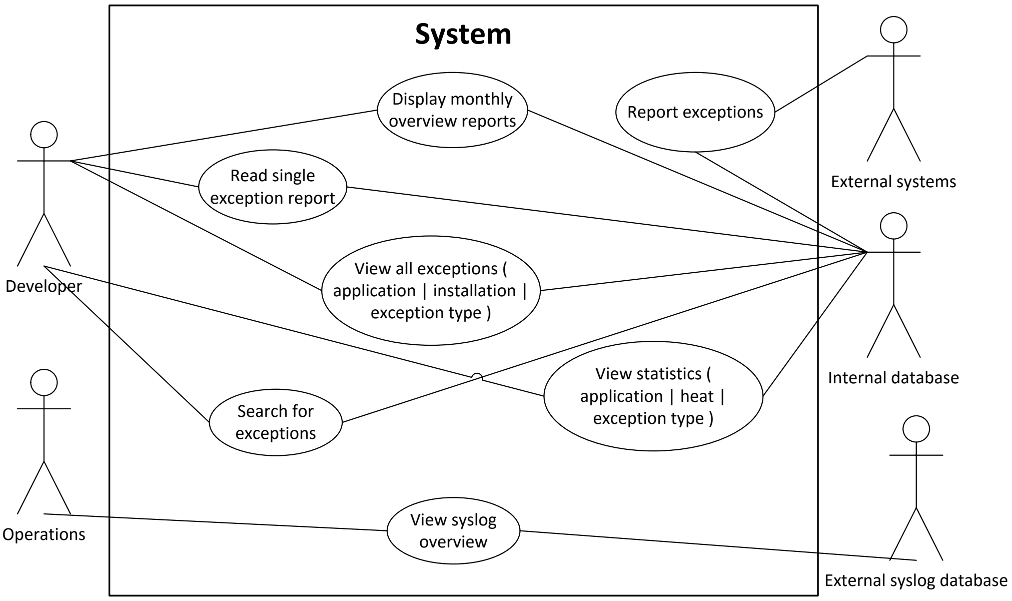

System
Display monthly overview reports
Report exceptions
Read single exception report
View all exceptions (application | installation | exception type)
View statistics (application | heat | exception type)
Search for exceptions
View syslog overview
Developer
Operations
External systems
Internal database
External syslog database

Figure 3: General use case diagram of the required functionality.

Figure 3 displays the functionality from the users perspective, without any indication of how it is to be implemented or modularised. The functionality is focused on overviews, single views of exceptions and syslog reports, and filtering options. The single use case of displaying a monthly overview report is a task that was later added when it became obvious we worked far too fast for the initial requirements to be enough work for the entire thesis period. It is a functionality to create, perhaps automatically, a monthly overview report with aggregated statistics about the month's exceptions. It can also be used before a month is done, to see how the month is going so far.

While it is not explicitly functionality as seen by the user, we have added a use case where external systems deliver the reports to the systems internal database. This to show how and where information enters the system. Note that the syslog database will not receive data through or in any way related to our system, and how it receives information is thus not shown in the diagram.

It has not been specified in the use case diagrams whether users will use desktop computers, laptops, tablets or mobile phones, but this is noted and defined as a part of the design requirements, see Section 3.2.

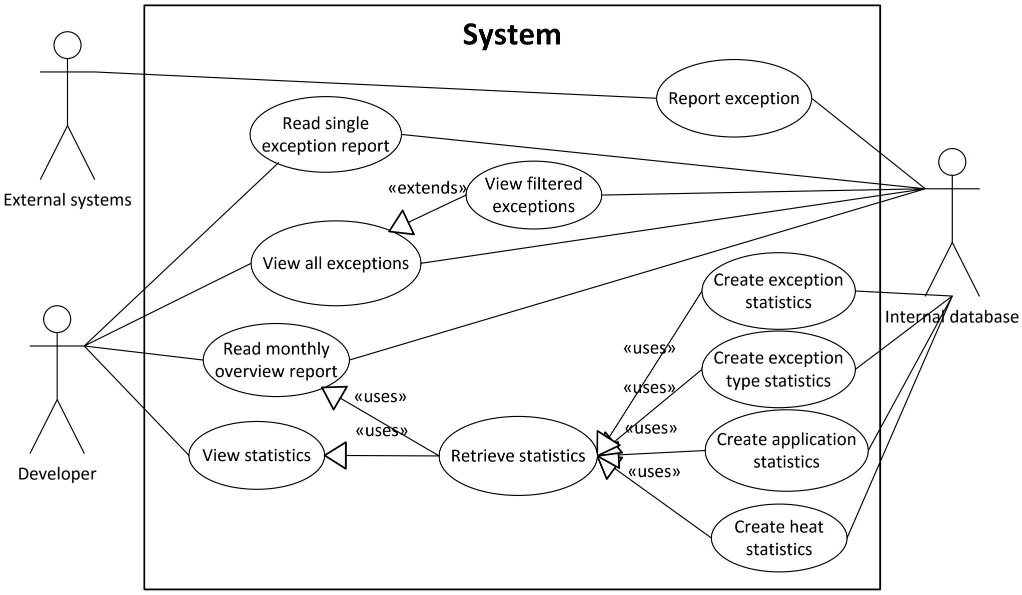


Figure 4: Use case overview of the system's exception-oriented functionality.

Figures 4 and Figure 5 show more technically oriented use case diagrams, with some indications of where one can start to divide architecture. As can be seen from the figures, as well as in the general functionality overview in Figure 3, developers and operations personnel will generally want to see completely different data. Developers wish to see a lot of statistics about the system, filtered on e.g. applications, installations, exception types and more, as well as sometimes reading single exception reports. Operations generally only care about syslog-reports, and this will be requested from a separate read-only syslog database that will be run separately by the employer.

As developers per the requirements stated by Innit will check the syslog-data far more often than operations personnel might check the exception data, only the first have been added as a use case, but since the system will not have any internal separation between the developer-data and operations-data, both parties can easily see each other's data and statistics.

The syslog-specific use case overview in Figure 5 is very simple, giving an indication of how much of the syslog part of the system is meant to be running with minimal or no user interaction. The front-end will mainly be continuously displaying current information about systems, while the back-end will generally only respond to incoming exception reports and API-requests. Only the front-end part running per individual user and the mobile application will be receiving any particular input. See more about the front-end design in Section 3.2

3.2 Design Requirements

The most requested feature of the system is what has come to be called the heat map. The heat map is simply a full-screen view dedicated to showing a transparently coloured

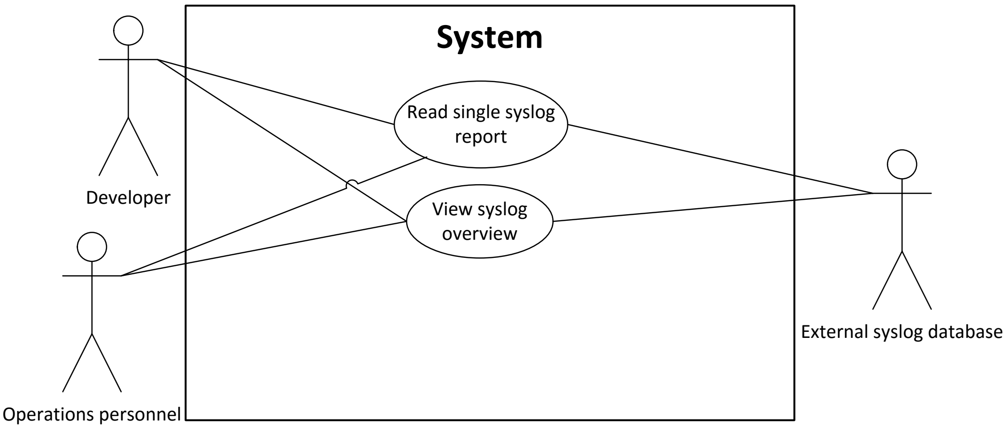


Figure 5: Use case overview of the system's syslog-related functionality.

graph; the colour depending on how 'hot' the situation currently is, i.e. how quickly are the exception reports currently coming in, multiplied with the severity/priority of the exceptions; that is whether the exceptions occurring are only informational and barely worth noticing, or potentially critical and must be handled immediately, and several stages

The graphical overview will be one or more types of diagrams that will display more elaborated information about what the heat map is an immediate indication of; how fast are exception reports coming in, and what severity are they. These graphical representations are intended to be highly configurable and can be filtered on multiple conditions, e.g. applications, specific installations or applications, types of exceptions, severity, time and date, amount, etc.

In Figure 6 can be seen one of the early sketches for the overview design. The heat map is the shaded area in the top right. In the top left corner of is a list of the most recent exception reports, below that is a graph of the heat during the past 30 days. At the right of that, in the middle right, is a list of each application's heat. In the bottom left is a diagram for heat or exception data, and in the bottom right is a bar graph of heat and exception data. Most of these were intended to be customisable.

At this early stage we expected Innit wanted a fair bit of information available on the overview screen since the system was mainly intended to be on constant display without interaction. Later they specified that the heat map could encompass most of the screen, with additional information left to the remaining parts of the application which developers would access and check manually.

Figure 7 contains some further information about how the site is laid out and how it is traversed. It is a somewhat later sketch, from when Innit had stated they wanted less information on the main page, rather favouring several sub-pages for additional information. The entire site is laid out as a single-page site, where the container in the middle of the picture is the main container for the site. From there, the main content rectangle is swapped out as partials in order to change content. The partials themselves can then be overlaid with modals to display other content temporarily without removing the partial from display memory.

For instance the main container in Figure 7 has a fixed navigation bar on the top. In

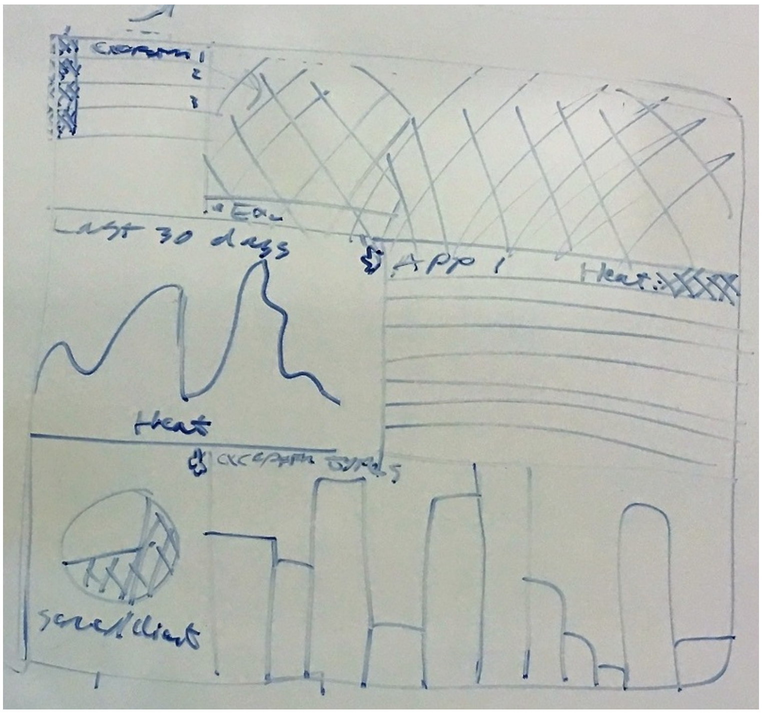


Figure 6: An early sketch of the overview screen. Heat map is the shaded area in the top right.

this sketch it also has a fixed sidebar on the left, but that was later moved into the partials so it would not be displayed unnecessarily in the overview. The partial to the right for instance only has a large graph at the top and as of the picture, undecided content below. The left partial however contains a smaller graph in the top left corner, and a list of the applications or installations that most recently received exception reports, in the top right corner. Upon selecting one, the user is taken to a modal with further information about the installation or exception. Below this is a content container intended for general exception data, though still undecided as to the specific content at the time.

3.3 Data Requirements

During January, February and early March, the amount of data required to be stored increased a fair bit, reflected in the discussion about database design, see Section 4.6. Since the employer both wanted to have statistical information presented in a graphical way, filtered by several different variables such as application, installation, exception types, and so forth, in addition to naturally being able to check specific exceptions in order to figure our what happened, why it happened, and how to fix it. We discussed some back and forth what information was required, and used the five W's as a way to figure out what information the employer needed to understand and fix any causes for exceptions [5].

Who – which customer was affected? This requires some kind of customer-identifying information, the specific installation name for instance.

What – what exception occurred? This requires the exception name, and potentially

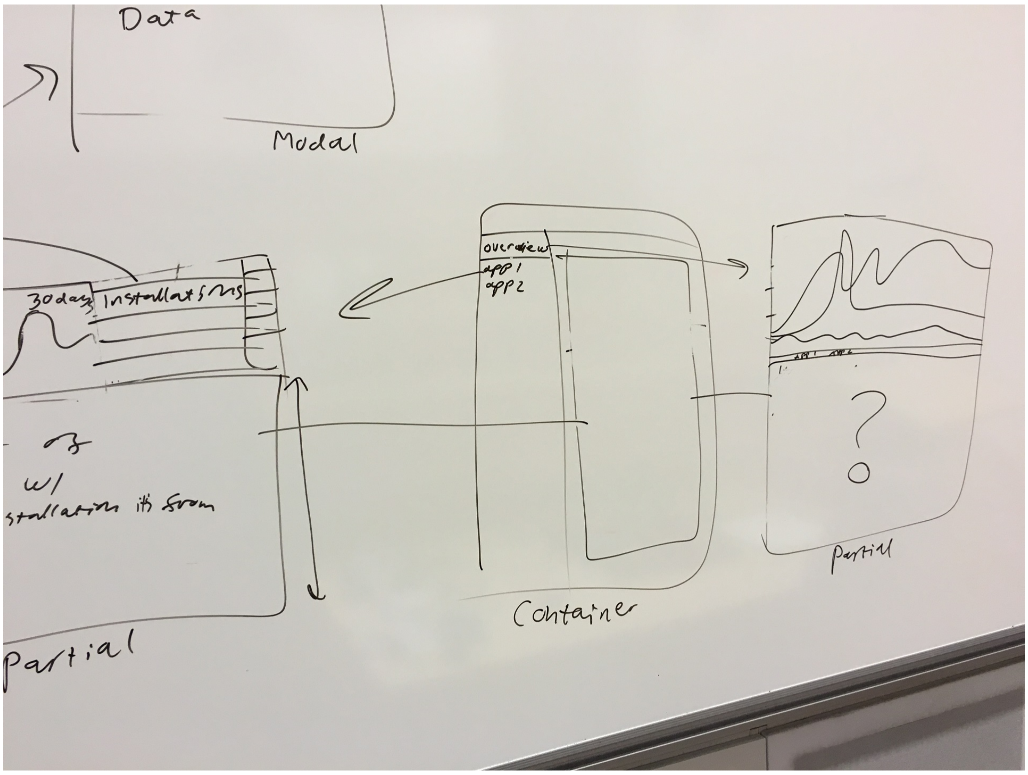


Figure 7: An early sketch of the overview screen with information on site traversal.

other identifying information of the exception type and exception message.

Where – where did the exception occur? This requires an application name, file name, and the line number where the exception occurred.

When – when did the exception occur? This simply requires a time stamp.

Why – why did the exception occur? This requires at a minimum a stack trace, and perhaps some other state information.

We partially used this technique of asking ourselves – with some sample exception reports from Innit available – what kind of information we ourselves would need to find and repair an issue in a program, and what other information would be – while not strictly necessary – practical for checking a single exception or reading the aggregated statistics. This self-interrogation based on the five W's quickly gave us a good outline of the information we needed to store. The information and the exception reports from Innit culminated in a draft of the data required, displayed in Figure 8. More fields were later added as their need became obvious, though the initial draft did cover a good part of the data required.

3.4 General Requirements

In addition to the specific requirements covering the design and data, there are some general requirements for the entire system and development process.

```
language: ‘php’     // language in which exception occurred
location: ‘server’  // client or server side
created: timestamp  // when exception occurred

application:
    name: ‘’        // name of application
    baseurl: ‘’     // name/url of the installation

client:             //if exception occurred at client
    username:‘’
    name:‘’
    browser:‘’

action:
    method: ‘get’,  // GET, POST, PUT, PATCH, DELETE
    data: []        // for instance POST-data
    sqlquery: [query-string]   // when DB-related exceptions

exception:          // json encoded dump
    type: ‘’        // type of exception
    file: ‘’        // full file path
    line: ‘’        // line number that caused exception
    message: ‘’     // some kind of message
    stacktrace: ‘’
```

Figure 8: Initial data model requirements.

- While not explicitly required it is highly recommended to build the front-end in AngularJS and the back-end in Laravel, because Innit have both experience and expertise with these frameworks. See more about AngularJS and Laravel in Section 4.1.
- The system must be able to integrate information from a separate read-only syslog-database that the operations department are running. See more about syslog in Section 4.5.
- The system should be as modular as practically possible to make it. In particular the front-end and back-end should be entirely separated and be replaceable without any rework of the other.
- We are to create a format in which the exception reports will be received, and the employer will take care of re-writing their reporting modules to fit it. For the format, see Section 3.3.
- No authentication is required. All systems will be running on the local network.
- Initially we were to assume only large screens. With us taking on a mobile app displaying a web view, this requirement has been changed to the site needing to scale for any screen size.
- It is preferable to do configuration through options in the system itself, or with configuration files.

3.5 User Types

While this is not a system intended for wide-range private deployment as mentioned in Section 1.4, there are two rather distinct types of users.

Developers. The developers are the main audience for the system, with the default overview and the application/installation/exception type overviews being in response to the their requirements. As Innit have stated, it is the developers whom need this system the most to replace their current manual parsing of exception reports.

The main overview, as well as all the information regarding exceptions, exception types, applications and for the most part installations are thus designed with developers in mind. They wish to get a quick and useful overview in order to prioritise time better, before diving into the single reports.

Operations. Operations' requirements are thus, while no less important, a smaller part of the development process and the finished product. This is also noticeable in that the syslog part of the system requested by operations is almost entirely separated from the remainder of the system data-wise, as can be seen in the data models, see Section 4.6.

Operations mainly require overview of the network and server hardware equipment, and will for the most part not have any need for all the exception data.

3.6 Architectural Draft

After the first meeting with Innit, we created an initial architectural draft based on the information we had and current understanding at the time. This draft evolved somewhat over the course of the next few weeks until the end of January. This work resulted in Figure 9, which is a very high-level model of the system and its different parts.

Figure 9 shows the relevant system borders and points of interaction. The external systems – represented here only by a single subsystem-box – will be sending information into the system via an externally available API. These reports are then processed in any way we find to be necessary, though the processing part might end up changing roles or disappearing altogether. A database module that will contain all database management logic will receive both the exception reports to be stored and all requests from the main data request API. It is also the only module allowed to have direct contact with the two databases, which can be read about in further detail in Section 4.6. The syslog database is controlled by Innit and our product will have read-only access to the data stored there. More about syslog can be read in Section 4.5. Any and all storage of information will be done in our products own database. The API will be a simple RESTful interface providing data in JSON-format.

In the front-end, the data service will be a typical AngularJS service that utilise the back-end's API and retrieve data requested by other modules. Unlike the back-end, which generally only responds to and processes input and requests, the front-end will have a controller that is continuously running, requesting data from the back-end and updating the views. As the system will be running on a display in the developers' office continuously with no or minimal user input, it will have to be running on its own accord. The

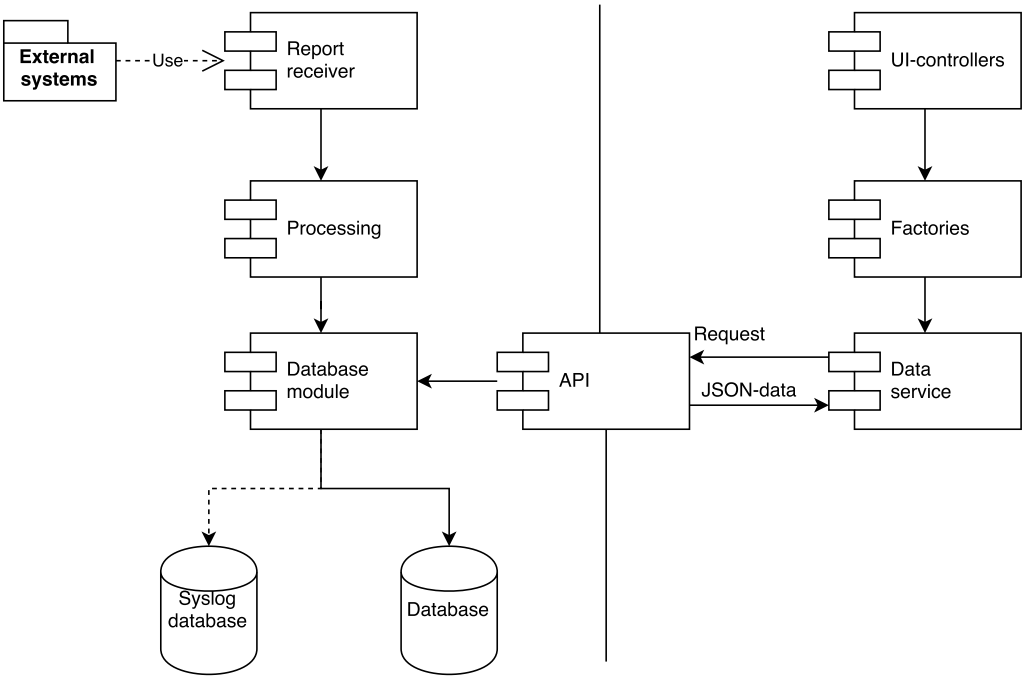


Figure 9: Module diagram with back-end on the left and front-end on the right.

graphics segment will generally handle the regular AngularJS views, as well as contain the package for drawing graphics.

It is worth noting that we expected modules and subsystem borders to change a fair bit during development. However the draft provided an important starting point for discussing access points, system borders, database management, and more. Given this information, much of which can be read in the meeting summaries in Appendix C. We also created several detailed architectural diagrams which can be seen in back-end architecture in Section 4.3, front-end architecture in Section 4.4 and API-specification in Section 5.3.

4 Technical Design

4.1 Languages and Frameworks

As was mentioned in requirements in Section 3.4 it was highly recommended to develop the back-end in Laravel and the front-end in AngularJS; amongst others because Innit had quite a bit of experience and expertise with the frameworks, for Laravel see Section 4.1 and for AngularJS see Section 4.1. We did some research on the two, in addition to researching alternatives; mainly plain PHP for the back-end, and Javascript, React.js, Ember and jQuery for the front-end. Ultimately we did follow the employer's recommendations and chose Laravel and AngularJS, where for the latter we chose version 1, as v2 was still in beta at the time we started. While stability was unlikely to be an issue, there were far more learning and helpful resources available for version 1, which was an important factor as none of us had used AngularJS before.

For design we did not have any particular recommendations to go by from Innit, and researched Bootstrap, PureCSS, Foundation, Less and Sass. We ultimately chose Bootstrap for design, and Less as CSS pre-processor which helps out with CSS syntax. We had a fair bit of experience with Bootstrap from earlier projects, and found it both simple and nice enough to use, considering none of us are particularly well versed in design. The Bootstrap-version is an Angular-friendly version that does not require jQuery as a dependency, and has a bit of Angular-friendly features and usability. We chose Less partially because we found it nice and simple to use, and both HotTowel and Bootstrap uses it. Thus we found it widely enough accepted and presumably practical enough for our use.

Such a mix of languages and frameworks, having from zero knowledge to some previous experience, gave us a good combination of building on knowledge and learning how to learn a new framework. This creates an interesting environment of building on experience and exploring entirely new technologies and approaches.

Laravel

Laravel is a PHP-based framework inspired by Java. It focuses on doing common tasks such as authentication, routing, sessions and such within the framework itself, thereby reducing the amount of boilerplate code the developer need to write [6]. Since we for the back-end did not find any alternatives that we deemed better, we simply defaulted to the employers recommendation of Laravel.

Laravel is what we later came to call a ‘magic’ framework, meaning that many tasks are done implicitly, and there are no real clues as to where something is called and why something is done. This follows from Laravel utilising dependency injection as one of its main design patterns. This focus upon hidden functionality and general ‘magic’ makes its code very easily readable, at the cost of call-trees that could be difficult to follow, and worse yet to debug.

We did in several cases find it to do too much ‘magic’ for our preference, especially in cases of input validation where we had to override the framework, or simply make it stop processing input in the way it is supposed to do. In addition it could at times be too influenced by what could be called ‘niche functionality’ that interfered with the general development process, or simply lacking in areas which standard PHP has already covered. One particular issue we encountered during development was lack of memory when fetching large amounts of data from the database. When retrieving and sending a few megabytes of data, the back-end simply crashed because Laravel ran out of memory. We never actually managed to fix this, and merely had to sidestep it by chunking data in smaller chunks and limiting the amount of data a single API request could contain.

Eloquent

Eloquent is an Object-Relational Mapping tool, implemented using the ActiveRecord pattern [7]. Object-Relational Mapping is a way of mapping objects in program logic to the relational structure that a database uses for organising data, and managing the data stored in a relational format using object-oriented code. ORM is intended to hide away at least part of the database communication, and optimise it through optimised queries and caching, in addition to making database communication more portable.

Eloquent is the default ORM tool for Laravel, and it is close to built into the Laravel framework, requiring no setup other than the URL and login information of the database to be used. This made it highly practical as a starting point when we had to begin learning and using Laravel for the back-end. However it did cause us several major issues, see Section 6.2.

Database

The data the system is intended to store is simple organised textual data in a set format, the kind of data well suited for a relational database. Since we had the most experience with relational databases, and the employer had no particular preference, we decided to go with MariaDB. The choice of MariaDB instead of MySQL is due to further optimisation in the former, and that it better supports some types of triggers that we later found we did need.

Since both of the databases have the exact same interface it did not matter to the rest of the system which we chose. In fact that equal interface became quite important later, as we – to our great relief – found that Rsyslog worked just as fine with MariaDB even though it is developed for use with only MySQL.

AngularJS

AngularJS is a Javascript framework that follows the model-view-controller (MVC) pattern. It is one of the most popular Javascript frameworks, and is sponsored and maintained by Google and a community of both individuals and corporations. Its main goal is to simplify both development and testing of single-page applications, and to provide a lightweight framework for dynamic web pages in general.

We used it for the front-end, to make a single-page application containing several views with heat map, graphs, incoming data, viewing single exceptions, and a lot more.

After a initial learning curve, angular was a breeze to work with. We originally had some issues learning the framework, and how to test our application properly but these issues fixed themselves as we got more fluent and experienced with the framework.

Bootstrap

Bootstrap is an open-source front-end web framework that contains a large amount of design templates for a huge lot of CSS-elements for all from typography and buttons to entire navigation bars. It provides a simple 'mobile-first' framework which handles much of the dynamic scaling to make it work on a multitude of devices, leaving the developer able to focus on the content.

Because the original Bootstrap depends on jQuery, and thus handles the Document Object Model (DOM) in a radically different way from how AngularJS does it, there were some onsiderations to be made. However since we at the time did not know of Angular Bootstrap – also called AngularStrap – we are using the UI Bootstrap project, which is a Bootstrap-implementation based on AngularJS, made by the AngularUI Team [8].

Less

Less is one of the more popular CSS pre-processors available, and it extends the default CSS by adding concepts from regular programming languages such as variables and functions (mixins), in order to make CSS more maintainable and extendable. Since CSS by default is a declarative language, lacking many of the concepts from programming languages, it can be a lot of work to update parts of the design or structure since all values are hard-coded. Less adds such concepts, and compiles into regular CSS before it is put into production. Running Less before pushing is one of the tasks automated with the use of the taskrunner.

4.2 Coding Conventions

This section will elaborate briefly on the coding style and conventions we have adopted in this project. Note that due to some differences in industry conventions for Laravel and AngularJS there are also some differences between conventions in the front-end and back-end code, as can be gleamed from Figure 10 and Figure 11, though they are not elaborate samples.

- Laravel – https://github.com/laravel/docs[9]
- AngularJS – https://github.com/johnpapa/angular-styleguide [10]
- HTML/CSS Google conventions - https://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml[11]
- API Best Practices - http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api[12]

The list above describes where we have found the initial conventions and style guides for the project, and though we have adapted them somewhat – within reasonable limits – to fit our preferences, we have taken care to not make the project too unfamiliar or difficult to read for anyone only used to other styles. As seen in the project plan, see Appendix 9, we initially chose Google's AngularJS style guide, though we soon after finishing the project plan changed this to John Papa's AngularJS style guide, noted in the aforementioned list. No particular reason caused this change, it was simply that Lars, being responsible for front-end development, preferred the latter style guide, and that John Papa's guide is endorsed by the Angular team.

In general we have opted for a similar coding style in both the front-end and the back-end. For instance we use space before a starting curly bracket, and space behind 'if'-branches, in addition to being generous with whitespace in general. For the back-end we have opted to follow the PHPDoc-format, while the front-end follows the JSDoc-format.

Since we had no previous knowledge of API design and development, we attempted to use the API Best Practices as a guidebook. While the API did not become perfect, it does follow a good hierarchical and consistent structure, see Section 5.3.

```
/**
 * For easier lookup we create a hashmap on an object's property
 * resolving to itself.
 *
 * @param {Array} array Array of Objects to index.
 * @param {String} property Name of property to index hashmap on.
 * @returns {Object} Hashmap of original array of objects.
 */
function createHashMap(array, property) {
    var map = {};
    for (var i = 0; i < array.length; i++) {
        map[array[i][property]] = array[i];
    }

    return map;
}
```

Figure 10: Sample code for displaying coding conventions of front-end

Figure 10 displays a sample of code from the front-end following out coding conventions. All functions require a JSDoc-comment with both a comment explaining in natural language what the function does, and other details deemed worth mentioning. All parameters and the return value are added with an explanation and note of which type is expected. Function and variable names also follow the camel-case convention. The function calls that are snake-case as the PHP standard functions, which are implemented with snake-case.

The function itself follows a standard of assigning temporary variables first, often with an inline comment if deemed necessary – which it is not here due to the function being short and having a good JSDoc-comment. Curly braces start on the same line and are ended on the line after the last line of code inside it. This is a matter of style preference rather than official conventions, but we have adopted it in both the front-end and back-end. We have adopted a medium amount of whitespace, as a compromise between the differing preferences of the team members.

The code example is not a large elaborate sample, but briefly gives an idea of how the code is written and structured.

Figure 11 shows a sample of code following the back-end's coding conventions. As with the front-end code, all functions require a PHPDoc-comment with both a comment explaining in natural language what the function does, and other details worth explaining. All arguments and the return value have an explanation of it and the expected type as per the PHPDoc standard.

```
/**
 * Query Function that allows retrieval of exceptions with given
 * exception context IDs passed as a string.
 *
 * @param Builder $query
 * @param string|array $ids String containing the IDs of the exception
 * contexts we want.
 * @return Builder $query
 */
public function scopeWhereInExceptionContextIDs($query, $ids) {
    if ($ids != null) {
        // If $ids is not an array, we need to explode it,
        // otherwise we just let it pass, as the only other
        // outcome would be an array.
        if (!is_array($ids)) {
            // Explode the values on comma. If there is no comma there,
            // it will just return the normal string as array.
            $ids = explode(',', $ids);
        }
        $query->whereIn($this->table . '.exception_context_id', $ids);
    }
    return $query;
}
```

Figure 11: Sample code for displaying coding conventions of back-end

In the code itself we have the curly braces starting on the same line as the declaration it belongs to, and the ending curly brace on the line after the last code inside it. We generally do not use very much whitespace inside parentheses, but whitespace is added before and after operators, as with the dot in the 'whereIn'-function call argument. Vertical whitespace is added at necessity, and is often addressed with ending curly braces on their own lines. It is worth mentioning that in the back-end we elected to use '$this->table' instead of hard-coding the table names, to make it change-proof.

Note that some line wraps in the code sample are not part of the actual code, and are merely artefacts of the slimmer LaTeXarticle documents.

4.3 Back-end Architecture

This section will give an high-level architectural overview of the back-end. Based on the requirements in Chapter 3 there were a few details important for the back-end architecture that we had to consider. In particular, the back-end needed an externally available API for communication with the front-end, and to receive the exception reports from the systems Innit are hosting. Innit did not state any requirements relevant for the internal data flow of the back-end, other than the languages and frameworks mentioned in Section 4.1, leaving us quite free to create an internal architecture of our preference.

We consider the systems that are sending exception reports 'Hosted Systems', such as in Figure 12. The 'Hosted Systems' is indicating the multitude of systems Innit is hosting for other customers, which will be sending exception reports. The arrow from it to the API simply indicate they are sending data to the controllers through the API.

The API-interface is not an actual interface-type class as in languages such as Java, but rather a specification for what is available in the routes-file. The routes-file is a file containing all the URLs the system will accept and which class and function to route the request to. The arrow going from the API to the controllers simply indicate communication, and not any particular type of communication. While not strictly necessary, we built in versioning of the APIs so software can use older versions of the API, and not cause errors the instant someone changes the interface. The API is further elaborated upon in Section 5.4.

All the controller-classes in the figure are actual controllers, extending the main Controller-class built into the Laravel framework. They contain all the main logic in the system, receiving requests, storing or retrieving data, and performing any other processing. In Laravel, controllers can be shared between several routes, but we have opted to have dedicated controllers per top level route; e.g. the routes '/reports/monthly' and '/reports/yearly' are both in the top-level route '/report' and use the ReportController along with all other routes that start with '/report', while the routes '/statistics/total' and '/statistics/application/:id' are both in the '/statistics' top-level route, using the StatisticsController.

All the models mentioned are Eloquent object-relational models, converting data between the objects in the program logic and the relational tables in the database. This is done using the object-relational mapper (ORM) Eloquent, which is supplied with and used as default by Laravel. Read more about the Eloquent ORM in Section 4.1. The models are thus void of any real program logic and act only as translators between the real logic in the controllers and the database.

Figure 12 thus gives a full, if simplified, picture of how the back-end is built up. Requests to the API are routed to the relevant controller, which contains any necessary logic for retrieving data through the models and creating a response. Any necessary logic for handling data is done in the controllers. The connections between the controllers, models, and the database have been simplified by grouping the modules into a single container. How the controllers interact with the modules can be quite dynamic, and detailing all that through regular connections would make the diagram unreadable. Therefore we have opted to have simple connections from controllers to modules, indicating communication, with the same from modules to the database.

Note that some controllers and models have for the purpose of simplicity been combined.

4.4 Front-end Architectural Overview

This section will give an high-level architectural overview of the front-end. As with Section 4.3 it will be a simplified high level architectural overview, based on the requirements stated in Chapter 3. While Innit did not state any particular requirements about the architecture of the front-end, their recommendation of AngularJS, along with some of the design requirements such as continuously running graphs and heat map, put a fair bit of constraints on the system flow and state handling of the system.

Initially the only differentiation we had was what AngularJS put constraints on; whether data should be stored locally on the client or retrieved every time it was needed, where logic should be placed, and a basis for where to divide the front-end into modules.

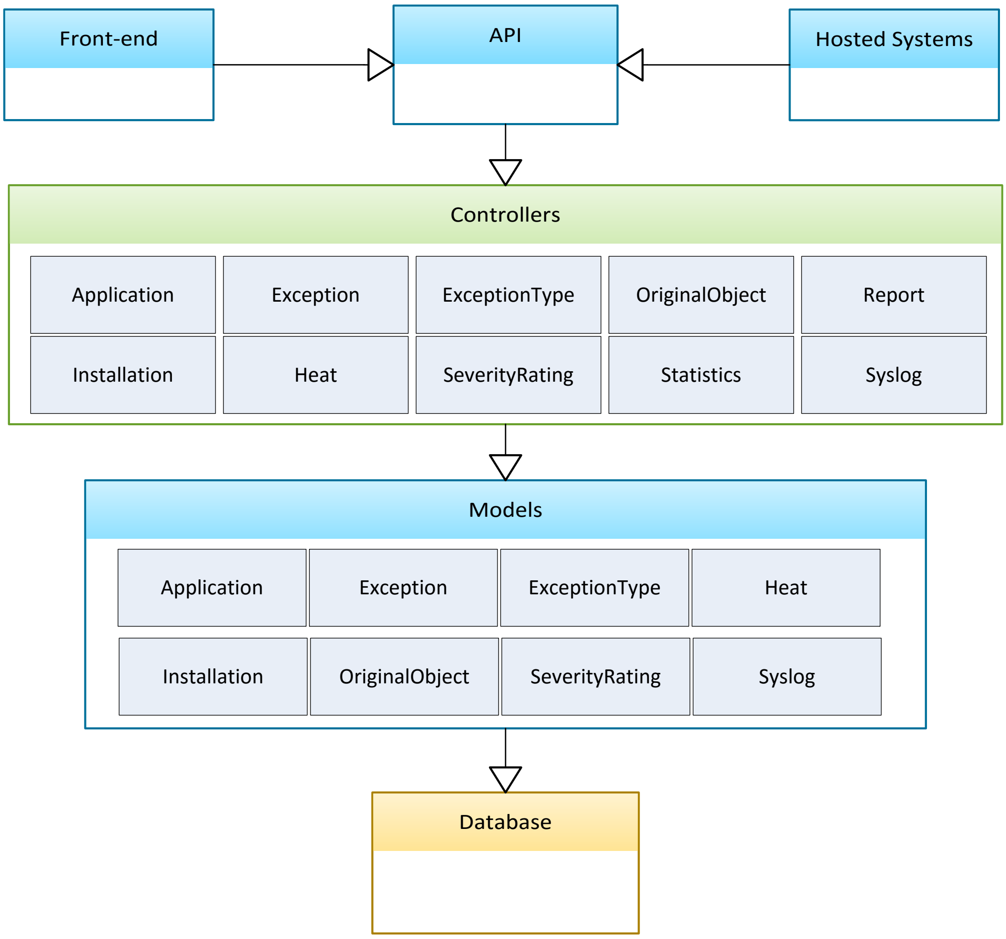


Figure 12: High-level back-end architectural overview.

This was used as a basis for the AngularJS modules, factories and services.

In Figure 13 the rounded box ‘H3E’ represents the entire system, with all of its configuration files and other dependencies. The first step after this was elaborating how the front-end would be separated in terms of modules. These modules contain the main logic of the system, being the ‘controllers’ in the Model-View-Controller pattern on which AngularJS is built. The models thus contain the programmatic logic, and communicates with the view that the user is seeing and interacting with, and they are completely standalone so that modules with their views can easily be removed without potentially ruining or removing system state other modules depend on.

The actual state of the system is contained in the factories, whose classes are singletons processing the data from the back-end. Thus they are the ‘models’ in MVC. In this way, containing the actual data and state. The services retrieve the requested data through the back-end’s API, and deliver the data to the factories for processing. The services are only called by factories, while the factories are only called by the standalone modules. In this way it is similar to a 3-layered architecture, though the similarity is not complete as each layer does not keep its own state and all necessary logic, and assumes there is a layer above it that will use its functionality.

Be aware that the overview in Figure 13 only displays the main modules and conceptual handlers involved, and is not a proper representation of the internal runtime flow. This structure follows the typical AngularJS structure of modules using factories, directives, services, etc. It is assumed the reader has some knowledge of how AngularJS' works, but it is by far not a prerequisite to understand the overview. For an introduction to what AngularJS is, see Section 4.1

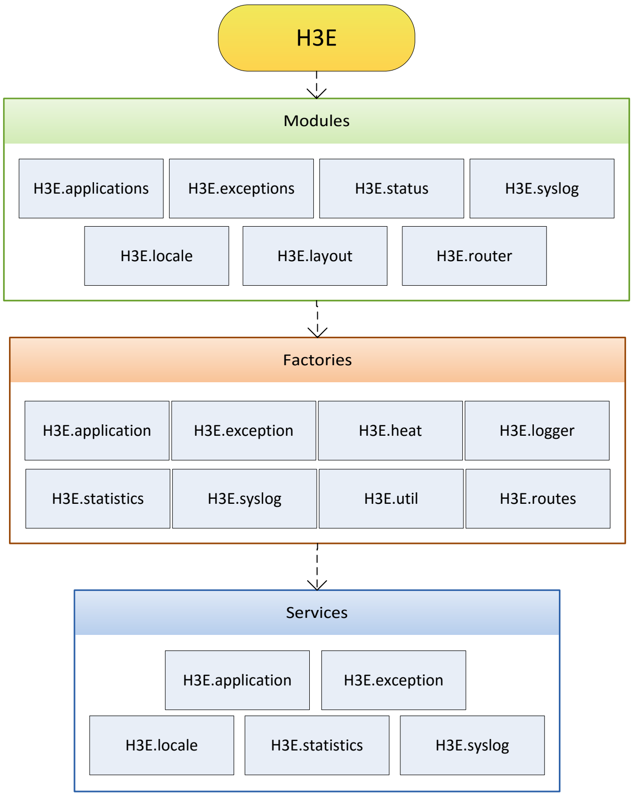


Figure 13: Front-end architectural overview.

4.5 Syslog

Syslog is a method for communicating information from devices, networking equipment and other units to a central logging server. It is a collection of standards defining logging format, required information and such.

The purpose of syslog is to provide a standardised logging format for most system and networking-related events such as login events, software errors, hardware state changes, and far more, providing the user with information, analysis and debug messages from the system. The systems can be all from enterprise grade routers and other networking hardware, to the Linux kernel on a home server using Rsyslog for reporting. All syslog-messages use a specific format that must adhere to the syslog standard for the daemon to be able to interpret it correctly. The message contains amongst other things, the following:

- Identifying information such as IP-address, MAC-address, or equivalent.

- Timestamps.
- Facility information, being an integer between 0 and 23, including endpoints, defining what kind of unit or device sent the message.
- Severity label, being an integer between 0 and 7, including endpoints, representing a application-defined severity from emergency (0) to debug information (7).
- The message itself, containing things like error messages and stack traces.

In the simplest form syslog consists of three systems; the log reporting software, a service daemon collecting the logs at a central server, and lastly a system that stores and analyses the aforementioned logs [2]. The idea with syslog is to collect and store logs generated by software and hardware from all systems in a network, whether it is VoIP-phones, printers, switches, routers computers or software. When the logs are stored they can be automatically analysed, or simply stored for manual review.

Because syslog was merely a de facto standard since the 80s – IETF documenting the status quo as-is in 2001 and creating a standard for it in 2009 – it evolved into a plethora of different implementations which are not all compatible with each other.

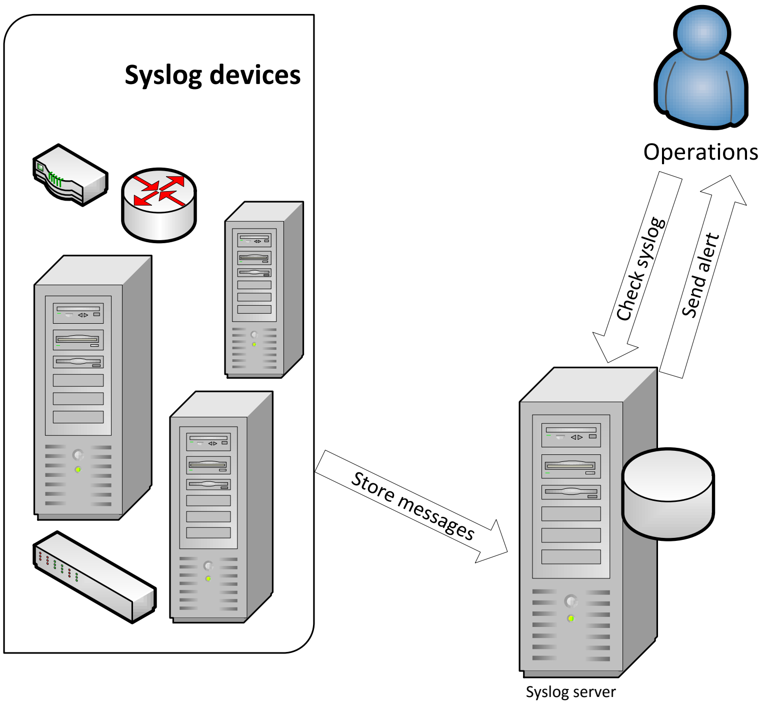


Figure 14: Illustration of syslog.

Innit did not at the time have a syslog database running, but wanted one and let us take the choice of which syslog standard and application to use. We opted to use Rsyslog, which is an high-performance open-source syslog daemon for forwarding log messages over an TCP/IP-network, and which is included in Debian-based Linux distributions. At its basis the Rsyslog daemon service only collects information about the kernel health and root login events through SSH, and stores them in rotating text files on the local host, and can either keep storing it or send it to some other system for more permanent storage.

We decided upon Rsyslog due to it being in wide spread use, is built for high performance, it having a stable community, and because using an established software for parsing incoming syslog messages is far safer than attempting to write parsing software on our own.

Since we are choosing an established application for parsing the syslog messages and inserting the data into the database we have to rely on Rsyslog's own database structure, and cannot change anything in the database structure, which is less than ideal for our purposes, see Section 4.6.

4.6 Database Design

Because so much of the product's functionality is based around storage and retrieval of the exception reports, the database design is central to creating a good product. From Innit we early on received some example exception reports, containing the general pieces of information we would need to store. Based on these reports we created the initial database models. There was not very much work required in creating this first version of the database model as we both had some ideas from the meeting before New Years as to what information was required, and while the reports better specified the information, they did not necessitate any reconsideration of the initial model. Thus after building the initial data model in the first week, it incrementally evolved alongside creating the exception report format throughout January, with the central parts of the model remaining constant throughout. However as development proceeded, the changes to the data model over time was substantial. To read how we got to this resulting data model, see Section 6.2. In this section we will describe the resulting data model in detail.

Figure 15 shows the product's main storage database, and is for the most part a fully normalised model, designed with efficiency in mind while avoiding duplicating information. Each application has been separated from its installations to achieve full normalisation. The same goes for exceptions by separating it from its exception types, and similarly for several other tables. Full normalisation – hence avoiding data duplication and inconsistency – was arguably not as much a requirement from Innit as it was a requirement from ourselves. While some amount of data queries could become more complex to create, going away from normalising the data model was never a real option for us, bar the syslog table for reasons further described later in this section.

The core parts of the database table are the exceptions, applications, heat and syslog tables, given a definition of core tables as those which not only exist for purpose of normalisation, contain sub-elements belonging to a single other element, or in other ways are made to simplify queries and behaviour.

The exceptions table contains the actual exception data that have been sent to the system. It is backed up by the original_objects table containing the raw unprocessed exception report that was submitted, which the employer wanted in order to retain a full copy of the report for archival reasons. To avoid duplication of data, the exception types of each exception was relocated to a table of its own, the exception_types table. The exception_contexts table contains the context of each occurring exception, i.e. the file name and line number where an exception occurred. The context was separated out into a table on its own both to de-duplicate data, and to making queries exceptions from a single context far simpler and faster to execute.

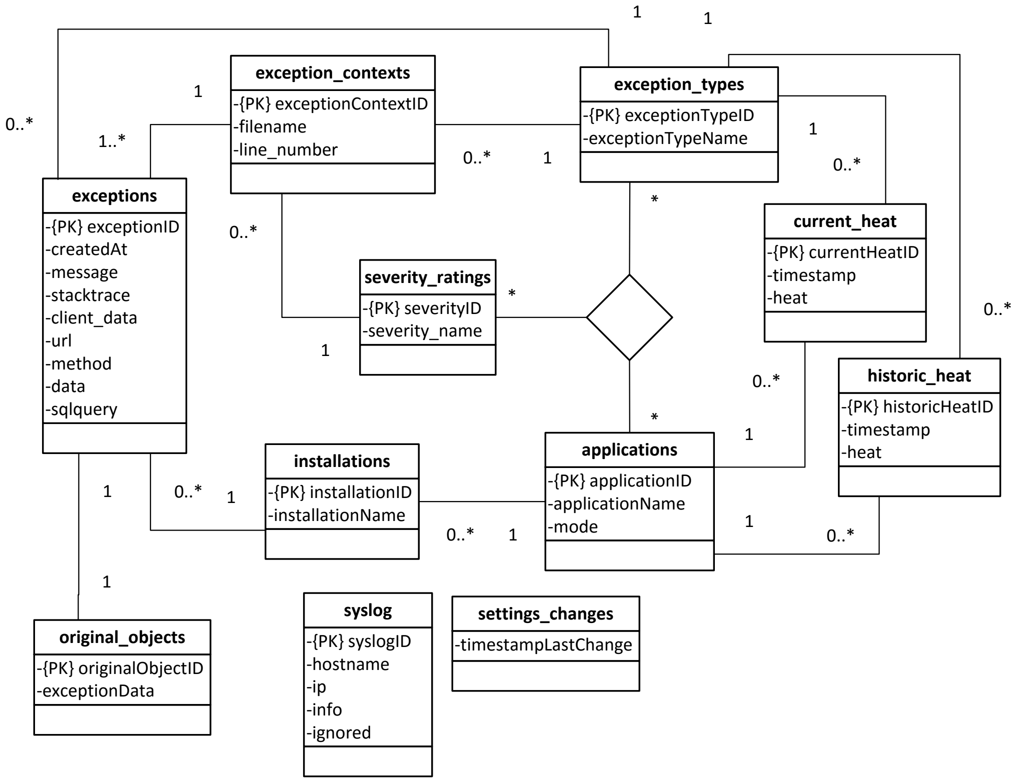


Figure 15: Model of the main database.

The severity_ratings table is a measure to de-duplicate information from the exception_contexts table, in addition to also simplifying some queries for all exception contexts and – by implication – exceptions with a particular severity rating. This measure is important, since the three-way relation between exception_types, severity_ratings and applications would otherwise be far more difficult to write, and potentially magnitudes more computationally intensive to retrieve.

The applications table naturally contains a list of all the applications that have thus far sent exception reports to the database, and the installations table is merely the subtable containing all installations of each application. The applications table could be read the other way around, being a de-duplication measure from the installations table, but because the application table is so important for the three-way relation and the heat-storage we consider it the core table of the two. Applications is thus used for retrieving heat – heat being a product of a particular exceptions' severity values, put together in an more easily discernible format for graphing – per combination of application and exception type, and to retrieve severity ratings for each combination of exception type and applications.

The heat storage tables – heat being a product of a particular exceptions' severity values, put together in an more easily discernible format for graphing –, current_heat and historic_heat contains the current heat per application and exception type combination, and the historic heat data per application and exception type combination, respectively. The heat tables are somewhat of a special case scenario, because they have exactly the same columns, data types and relations. We decided however that for simpler queries and management that the current and historic heat data were best set in separate tables. For more information on heat and heat management, see Section 5.5.

The settings_changes table is also a somewhat special case, only being used for storing the timestamp for when somebody last updated an exception context's severity rating or the severity rating of an application and exception type combination, the latter being the three-way relation.

Note that tables such as applications, installations, exception types and such tables are not preemptively set, but entries are added dynamically as new such units send exception reports with new exception types.

Syslog Database

As have been mentioned in the use case section, the syslog-functionality is a mostly standalone feature required by the operations department. This is reflected in the database model where the syslog-table is entirely isolated from the other tables. We have considered this thoroughly, and have not found any reason to have a relation between the syslog-table and the other tables.

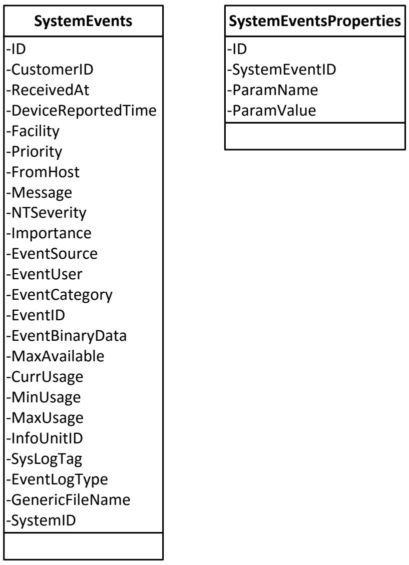


Figure 16: Model of the read-only syslog database.

The fact that we could not change the syslog database seen in Figure 16 at all, due to it being defined and set by Rsyslog, necessitated some unfortunate duplication of information. The syslog database is large, potentially containing hundreds of millions of records that are not normalised. Without indexing on particular columns that we would need, we could not use that database for continuous retrieval of data. Thus the syslog table in the main database is essentially a compromise between our preferred database design and a predefined database structure set by Rsyslog. This compromise means duplicating some columns from the syslog database into our product's own database, in order to have them normalised and indexed so we can retrieve data from it continuously, without creating a huge processing overhead in retrieving only unique entries.

5 Implementation

5.1 Graphical Design Considerations

Designing a front-end that was compliant with standards for universal design was perhaps a more important thing for us than for Innit. At least they did not mention it in the earlier stages, though they greatly appreciated that we had taken the initiative on universal design.

We wanted to take advantage of the site's capabilities to create a responsive, modern and fluent design that follows modern web development standards and would be usable in a multitude of situations, from very close up to a good distance away from the screen such as with the overview screen in Section 1.4. In addition, while it is outside the scope of this project, it is possible that a mobile application version of this system will be created in the near future, in which case having the front-end compliant with universal design and using responsive design could greatly reduce the amount of work necessary to make a reasonable mobile application.

One major consideration was the design of the heat map, see Figure 17, in particular since it was the main feature Innit wanted with the entire system. During our GUI feedback meeting with an expert at campus, see Section 7.7, the use of colours as indicators was pointed out. Initially we had a fixed colour for the heat map, displayed as a background to the heat status graph on the status page. Using colour alone to display heat could be very ambiguous for colour blind users. We therefore re-designed the heat map with less focus on the colour usage, and more focus on easy to read text and unambiguous symbol usage instead.

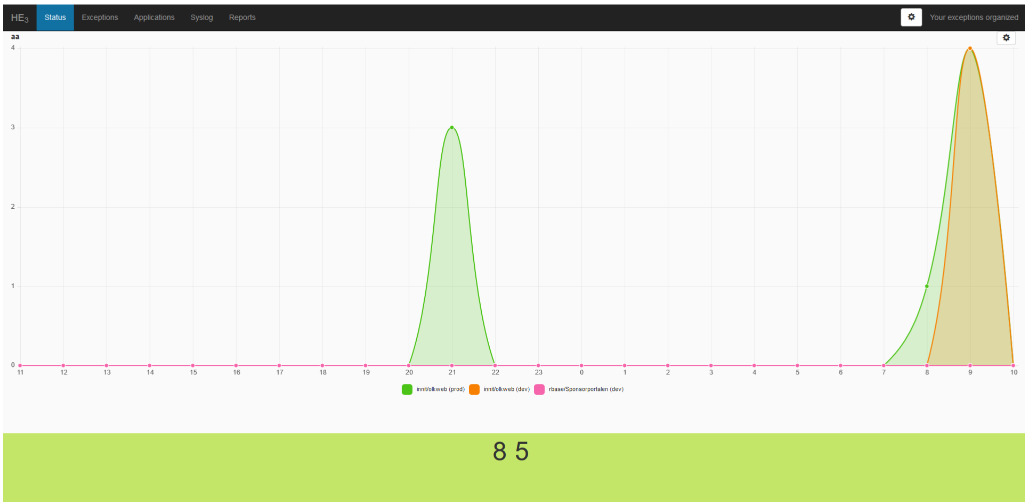
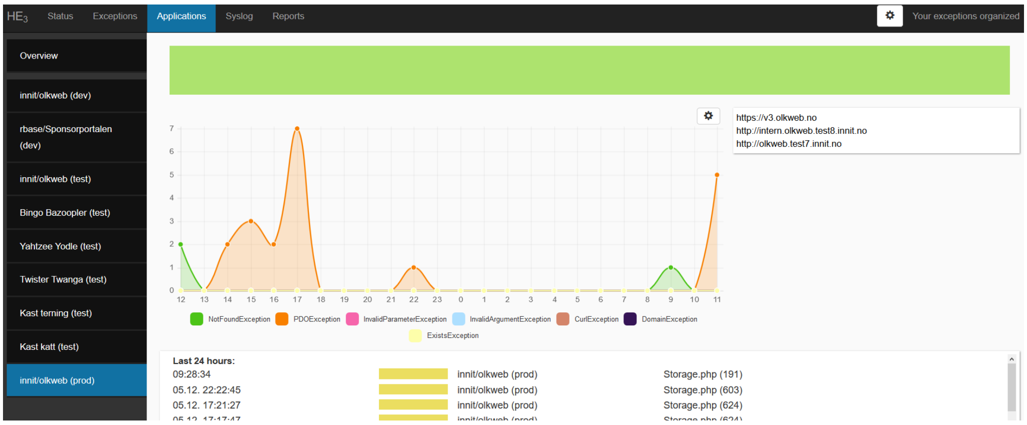


Figure 17: Front-end heat map design. The graph shows the heat over time for the currently selected applications. The bottom green bar is the heat map showing current heat for this application, with a number for the heat in absolute value.

Site navigation is done in a very standard way, with a main navigation bar at the top of the page, and sub-navigation bars on the left side of the screen, see Figure 18. For outlining we used a standardised way of filling the chosen menu item with the colour blue, and in the main navigation bar the text also changes from grey to white. Since the colour is only used to outline it compared to the other menu items, the use of colours is perfectly acceptable in this case.



https://v3.olkweb.no
http://intern.olkweb.test8.innit.no
http://olkweb.test7.innit.no

Last 24 hours:

09:28:34	innit/olkweb (prod)	Storage.php (191)
05.12. 22:22:45	innit/olkweb (prod)	Storage.php (603)
05.12. 17:21:27	innit/olkweb (prod)	Storage.php (624)
05.12. 17:17:47	innit/olkweb (prod)	Storage.php (624)

Figure 18: Front-end application overview design. The green bar is a heat map for this application. The graph shows number of exceptions of each type for this application. The right side shows all installation URLs for this application, and the bottom – which continues further down the page – shows all the single exceptions for this application in descending order by time and date.

Many buttons such as for configuration, inserting data, deletion, etc., use symbols instead of text. We have opted to use the Glyphicon Halflings icons through Bootstrap as they are compliant with universal design and responsive design, are not bound by strict copyright rules, and the icons are well known to give an accurate relation between icon and the function it represents, e.g. a cogwheel for settings or an 'X' for deletion or closing a window. Note that the Glyphicon Halflings set are normally not available for free, but the creator has made them available for use with Bootstrap at no cost [13]. This usage can be seen in Figure 18, where the button for opening the settings window uses the almost globally recognised cogwheel.

Fully responsive design has not been a major focus for this project. It has always been a consideration, but we have not made extensive testing on smartphone-sized screens, as the functionality and design preferences of Innit has been focused on typical TV-, laptop- and desktop sized screens.

For images of the front-end that was not displayed here, see Appendix F

5.2 Exception Report Format

Creating the exception report which our back-end system were to receive format was an important task that had to be done early, along with the data models for database storage. We were tasked with creating this format, then Innit would re-write their reporting modules to send reports in that format. We needed to build a format that would not require an excess of information, but would still be sufficiently future-proof in the sense that no (major) overhauls or changes were required once the system is deployed due to lack of information.

To achieve this Innit gave us several example exception reports with the type of information they were sending in their old system. With those as a basis, and while building the initial back-end architecture we built an initial format proposal which we presented to Innit at a meeting in January. They were quite satisfied with it and had no particular details to append. Throughout January and early February the initial format evolved somewhat as we built working demos of the back-end, and go to testing with actual semi-real data, though no major structural changes were needed; we only added or renamed a few of the fields as we saw it needed, and when Innit in late February had a few proposed columns to be added before the testing with real data would begin.

```
Method: POST
URL: /exceptions/
Data:
{
  ‘language’:‘php’,
  ‘location’:‘server’,              // ‘server’ or ‘client’
  ‘created’: ‘2016-01-01 12:34:56’, // YYYY-MM-DD hh:mm:ss
  ‘application’: {
    ‘name’: ‘Test Application’
    ‘baseurl’: ‘http://www.test.com’,
    ‘mode’: ‘production’        // ‘developer’, ‘test’ or ‘production’
  },
  ‘client’: {
    ‘username’: ‘User Name’,
    ‘name’: ‘Real Name’,
    ‘browser’: ‘Mozilla Firefox 5.0’
  },
  ‘action’: {
    ‘url’: ‘http://www.test.com/’,  // Where the exception occurred.
    ‘method’: ‘get’,                // get, post, put, patch, delete
    ‘data’: {},                     // POST-data
    ‘sqlquery’: ‘[sqlstring]’
  },
  ‘exception’: {
    ‘type’: ‘PDOException’,
    ‘file’: ‘index.php’,
    ‘line’: 5,
    ‘message’: ‘What went wrong!’,
    ‘stacktrace’: ‘[Stacktrace as String]’
  }
}
```

Figure 19: Format for adding a new exception report

While elements such as the API in Figure 21 kept changing and evolving throughout the entire development process, the exception report format did not need any changes after we started receiving real data. The only adjustments made were how strict the input validation had to be. Due to unforeseen circumstances, on Innit's request we ended up removing close to all input validation. We did not wish to do so, though due to how the employer's systems sent data there were too many pitfalls in what kind of data was sent in. Normally this would have been highly insecure, a fact we expressed to the employer, but they assured us that since the system would only run on the local network and only fed data from their reporting module, the risk was acceptable. Thus there is only the most rudimentary input validation currently in working order, in essence only checking whether some of the fields are the correct type.

5.3 API-specification

A preliminary API-specification was prioritised early along with the reporting format, see Section 5.2, with the intention of both agreeing with the employer on which functionality is central, and having it slowly evolve as development and testing showed what worked best. While we had ideas and rough sketches of the API already before new years and the official thesis start, it evolving steadily as we realised new use cases and found better ways of implementation.

The API intentionally follow the principles of a stateless RESTful API; i.e. it uses the regular HTTP-request methods GET, POST, PUT, PATCH and DELETE, and operates on uniform resource identifiers (URIs) [14]. This means the API treats every request as an independent transaction, and all communication only consist of independent request-response pairs. Thus one cannot refer to or assume knowledge of previous API-requests, giving some overhead with information that must be added in every single request, e.g. an authentication token. However being stateless gives a simpler, more reliable and more scalable API. Simpler because it requires no logic for sharing or moving state, sessions or anything of the kind; more reliable since it does not require allocation or memory for a session or complex state handling; more scalable as one can simply add new servers without any intricacies of sharing or moving state, handling sessions, dynamic allocation of memory for session data, etc. For the server it is essentially respond and forget, with no state to clean up for the server in case the client dies during the transaction. Some transactions might be simpler with a stateful session, but we had no such occurrences.

We did not find the API to lose any particularly advantageous features for our use by being stateless, and could thus save quite a bit of development time. Since Laravel has built-in features for creating RESTful APIs, the development time for setting up the API-interface itself was almost negligible; only the actual logic that is called to retrieve the requested information takes some time to develop, and even that is considerably easier, being stateless and completely decoupled from other parts of the API.

A major advantage with a simple decoupled API like this is that the entire interface can be re-used for both the interactive desktop and mobile clients, in addition to the originally intended minimally-interactive front-end.

The initial stages of the API was concocted in early January, almost immediately after receiving the first indications and examples from Innit about what information a user would require. After that, the API evolved as we worked throughout January and February, having better clarified what the employer wanted, and us finding better paths of implementation. Over the two months the API changed quite dramatically, most parts of the interface either receiving major overhauls and redesigns, or being replaced altogether by better designs.

As an example, the first draft of the API structure looked like in Figure 20. This was the absolutely first idea, partly based on ideas bred during the Christmas holiday, and partly from the first meeting in January. Needless to say, it would not have been sufficient for all

```
/exception
/all
    /id/:id
     /date/:date
    /from/:id
/application
    /all
    /:name
/statistics
    /all
    /total
     /:id
        /all
        /total
        /last/:last\_entries
        /data/:from/:to
```

Figure 20: Initial draft of API structure.

the requirements. It proved a good starting point however and it quite quickly evolved to better suit our needs, resulting in the API structure seen in Figures 21A and 21B. This figure only shows the URIs and GET-arguments available. The full API structure with explanatory comments can be seen in Appendix D.

It is worth nothing that Figures 21A and 21B are the result of a long evolving process, and several of both the routes and sub-routes came to be as a result of the development process, and not the initial few requirements meetings with the employer. This has been in accord with our agile development methodology, and through keeping in constant communication with the employer throughout the entire process. For instance the /severityRatings, /exceptionContexts and /reports routes were added as late as in early- and mid-March, after feedback during the demonstration of the system in early March.

The /graph route was actually added as late as the end of April, during a major bout of refactoring. It is the result of moving much logic from the front-end to the back-end, logic concerning transforming data into the format the graph package requires to create the graphs. There was a major discussion whether this would be acceptable to do, as it arguably introduces some coupling between the front-end and the back-end which we were to avoid. However by making simplifying the communication and data handling the main focus, we managed to avoid explicit coupling while significantly reducing complexity and lines of code. For more detail on this improvement, see Section 6.3.1.

Figure 21: Final API structure

```
All routes are configured to use these return values:

* Request that yields data: Content and status code 200 (OK).
* Request that yields no data: Empty array and status code 200 (OK).
* Invalid ID of a specific item:
"Resource not found" and status code 404 (Not Found).
* Invalid URL (Non-existent route):
"Resource not found" and status code 404 (Not Found).
* Invalid query (Failed validation):
Array with error messages and status code 422 (Unprocessable Entity).

GET-arguments starting with *

/exceptions
        * after=:exceptionID
        * from=:timestamp
        * exception_type_ids=:exceptionTypeID, comma-separated list
        * installation_ids=:installationID, comma-separated list
        * application_ids=:applicationID, comma-separated list
        * search=:string
    /exceptions/:exceptionID

/exceptionTypes
    /exceptionTypes/:exceptionTypeID

/exceptionContexts
    /exceptionContexts/:exceptionContextID
        * application_id=:applicationID
        * context_ids=:contextID, comma-separated list

/severityRatings
    /severityRatings/:severityRatingID
    /severityRatings/application/:applicationID

/applications
    /applications/:applicationID

/installations
    /installations/:installationID
        * application_ids=:applicationID, comma-separated list

/reports
    /reports/monthly
    /reports/monthly/:year-month
    /reports/yearly
    /reports/yearly/:year
        * from=:date
        * to=:date
```

21A First part of the final API structure.

```
/statistics
    /statistics/total
    /statistics/application/:applicationID
    /statistics/application/:applicationID/total
    /statistics/installation/:installationID
    /statistics/installation/:installationID/total
    /statistics/exceptionType/:exceptionTypeID
        All non-‘/total’-routes support the following:
        * from=:date
        * to=:date
        * modes=:modes, comma-separated list

/syslog
    /syslog/:syslogID
    /syslog/hosts
    /syslog/hosts/:syslogID
        * ignored=:true/:false
        * gt=:number
        * lt=:number
        * host=:name

/graphs
    /graphs/application/:applicationID
    /graphs/exceptionType/:exceptionTypeID
    /graphs/mixed
    /graphs/total
        * mode=:mode, ‘heat’ or ‘exceptioncount’
        * timespan=:timespan
        * application_ids=:application_IDs, comma-separated list,
        only for /exceptionType and /mixed
        * exception_type_ids=:exception_type_IDs, comma-separated
        list, only for /application and /mixed

Global arguments
    All endpoints support chunking using these arguments
    * limit=:limit
    * offset=:offset
    * orderBy=:fieldName
```

21B Second part of the final API structure

5.4 System Administration and Configuration

Front-end

The front-end system is configurable using the file appConfig in the root folder. appConfig keeps track of all configuration parameters necessary for the system to function correctly, e.g. such as toaster-arguments – where a toaster is the small information pop-up in the lower right corner of the screen – , locale settings, and more. It also contains the variables necessary for locating the back-end's API end point, such as the example in Figure 22

The reason for choosing configuration files is that Innit during the initial requirements phase expressed that they prefer text files for initial setup and configuration. In

```
‘RESTAPI’: ‘https://ex.ankazhi.info’,
‘API_VERSION’: ‘/v1’,
‘ROUTE_SYSLOG’: ‘/syslog’,
‘ROUTE_EXCEPTIONS’: ‘/exceptions’,
```

Figure 22: Example of content in front-end configuration file

addition the intent is that one can e.g. simply change the few relevant lines in order to change which back-end is queried; one can easily change to a back-end that is hosted somewhere else entirely, or use another version of the API. With multiple versions of the API available, one can more easily upgrade the back-end API without breaking the front-end's dependency on the API functionality, and the front-end can be upgraded to the new API while the old version is still using the old version. Naturally the main benefit is avoiding any hard-coded endpoints or other information of the kind, and once some information is set in a configuration file, it would not make sense to keep other parts of the configuration in another type of storage.

Back-end

In much the same way as the front-end, back-end configuration is done through a dedicated configuration file. In this case it is a default Laravel-specific environment file called ‘.env’, which contains all the data required for operation such as system mode, database endpoint, credentials, and API-end point. Since our system uses both the dedicated internal database for storage, and relies on an separate syslog-database external to the product to retrieve syslog data, the configuration file contains both the database connections as in the example in Figure 23.

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=SomeRandomString

DB_HOST=localhost
DB_DATABASE=dbName
DB_USERNAME=someUsername
DB_PASSWORD=superSecretPassword

DB_HOST2=localhost
DB_DATABASE2=dbName2
DB_USERNAME2=someUsername
DB_PASSWORD2=superSecretPassword

API_VERSION=v1
```

Figure 23: Example data from .env-file

5.5 Heat

Some kind of feature that could at a glance immediately tell developers and operations staff how the systems are coping was a greatly requested feature by the employer. This request resulted in what is called heat maps. Heat maps are a graphical representation of how a system is running, that instead of showing absolute values – thus requiring the onlooker to know what absolute value is fine, cause for concern, or critical – shows the values are percentages or relative values, relative to either some previous maximum or an pre-set absolute maximum. Heat is then, as mentioned in Section 4.6, a product of a particular exceptions' severity values, put together in an more easily discernible format to make it easier to use and graph.

Heat Calculation

There are two sources of heat values. One is a constant heat value based on which exception type a given exception belongs to. The other is a heat value taken from the context an application occurred in; in which application the exception occurred, and where in that application it occurred. This latter factor is intended to adjust for the fact that some exception types can be almost irrelevant in one application or part of the code, while the same types can be absolutely critical in other applications, or even simply in another part of the same application. For instance, if a ObjectNotFoundException were to happen sometime when displaying a little used sub-page of an application, that might be trivial, while another ObjectNotFoundException could be critical if the missing object is a central part of the login process.

Calculating the heat from exceptions is kept simple, and done as follows each time a new exception is input:

1. A new exception comes in.
2. The system retrieves its exception type and context severity values.
3. The retrieved values are added together, and then added to the current heat value for that combination of exception type and application.
4. The database is updated with the new current heat value.

Note that this updating is done with the use of database triggers and events. Points 2, 3 and 4 are done by a trigger that is activated when a new exception is entered into the database. In addition, there are two events running at regular time intervals; one event handles the heat degradation, see Section 5.5.1, and another event handles moving the current_heat values to the historic_heat table every hour, see Section 4.6

For displaying heat the front-end can request the historic heat data, of which is taken a snapshot every set time unit – usually less often than heat degradation -, and request the current heat data as often as the user wishes to. As a consequence, the historic heat data will be less fine-grained than if one keeps the front-end up and running, continuously polling the back-end for new heat data. However since it is the current heat that is most important, this is not seen as a major issue. In addition the employer can change how often historic data is stored, though with some more work than the other configuration options, since it requires change to a trigger in the database.

5.5.1 Heat Degradation

A major consideration was how to trigger the reduction function. Initially we considered running it every time heat was added to the specific database entry, and degrading the value by a corresponding amount of time since the last time it degraded, but we quickly reasoned this could cause hours or even days where no degradation would happen. In general we wished to have all functionality in the back-end only run in response to input,

```
<?php
$n = 0;

for ($i = 0; $i < 300; $i++) {
    $n = $n + rand(10, 40);
    $n = $n - 30;

    print $n . ’<br/>’;
}
?>
```

Figure 24: Code for generating linear degradation graph data

not have any timers that would trigger functions. However we found no other sufficiently good alternative to using a timer to trigger heat degradation reliably.

While heat increases are calculated each time a new exception comes in, heat degradation is run automatically approximately every configurable interval of time. Retrieving heat values is done through the API, which simply retrieves the value from the database, no calculation required at all. We reason that this setup will scale rather well, albeit that is not a requirement. The database handles transactions and thus concurrency, and there should not be any issues with lost exceptions or failed degradation updates until well after some other part of the system is likely to be the bottleneck.

In addition to when to trigger the degradation, the way of degrading the heat value over time is quite important. The employer did not specify any particular way this should be done, and it was left to our judgement to find a good technique for handling degradation. In the following sections is testing of the alternatives, with simulated data, the code used to simulate the data, and the resulting heat values graphed.

Linear Increase and Linear Degradation

One possibility was linear increase and degradation. Each incoming exception report would add an amount of heat based on its severity, and at each regular time interval a fixed amount of heat would be removed. An example is shown in Figure 25. As can be seen, the heat is very unstable, and quickly either the incoming heat overwhelm the degradation and the heat simply keep increasing, or the degradation overwhelm the input and go into negative values. The middle case in the figure is almost stable, though does go somewhat down to negative values. However even with the middle case, Innit would have to continuously adjust the degradation in line with the amount of incoming exception reports to keep it stable, and we would have to take care to avoid the heat becoming negative.

Adjusting the degradation value would be very difficult since it would be a global adjustment of degradation, while the different exception types and combinations would have wildly differing heat increases. It took several attempts of trial and error with the numbers to get them low enough to graph without either the top of bottom degradation values completely overshadowing the middle ground, and for such unknown quantities of incoming heat, linear increase and degradation would not be sufficiently good. The code for generating the graph data can be seen in Figure 24

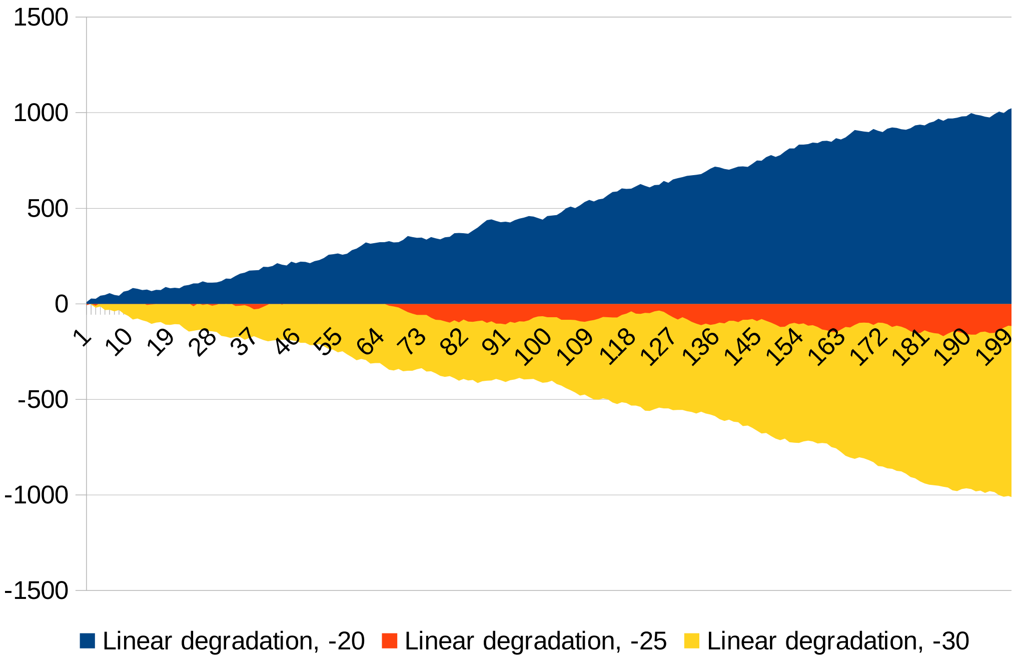


Figure 25: Using linear input and linear degradation, degradation value in the legend

Exponential Increase and Exponential Degradation

Another possibility is the case of an exponential function for both increase and decrease; the more heat is available the larger the increase in heat would be, which would fit the use case that an application producing a lot of heat in a short amount of time is extra critical, see Figure 27. However the increase would depend on the initial data, so that if an application for instance ran for days without any incoming exceptions, the degradation could become low enough that even a large amount of exceptions would make a barely noticeable change, or if enough exceptions came in at once it could produce runaway heat that could take a long time to subside. Both of these cases are displayed in Figure 27, where random numbers generated with the code in Figure 26 were used to simulate input. In addition there would always have to be some heat from the start, else multiplying with zero would produce the unfortunate amount of zero heat. Both issues could be fixed by adding both a fixed value and an exponential for all exceptions, but that still leaves the risk of runaway heat, where a major case could potentially cause an integer overflow.

It is worth mentioning that even finding values for increase and degradation where the values would not immediately go to zero – due to the floor()-function, which was added to avoid getting numbers with potentially tens of digits after the decimal point – or into the millions took a significant amount of attempts. We found it to be highly unstable and thus a very poor alternative.

Linear Increase and Exponential Degradation

A third alternative was linear input and exponential degradation. In comparison to the two previous alternatives, having linear increase and exponential degradation proved to make the degradation inherently stable; i.e. it will always go towards an equilibrium, see

```
<?php
$n = 100;

for ($i = 0; $i < 100; $i++) {
    $n = $n * (rand(11, 15) / 10);
    $n = floor($n*0.8);

    print $n . '<br/>';
}
?>
```

Figure 26: Code for generating exponential input and degradation graph data

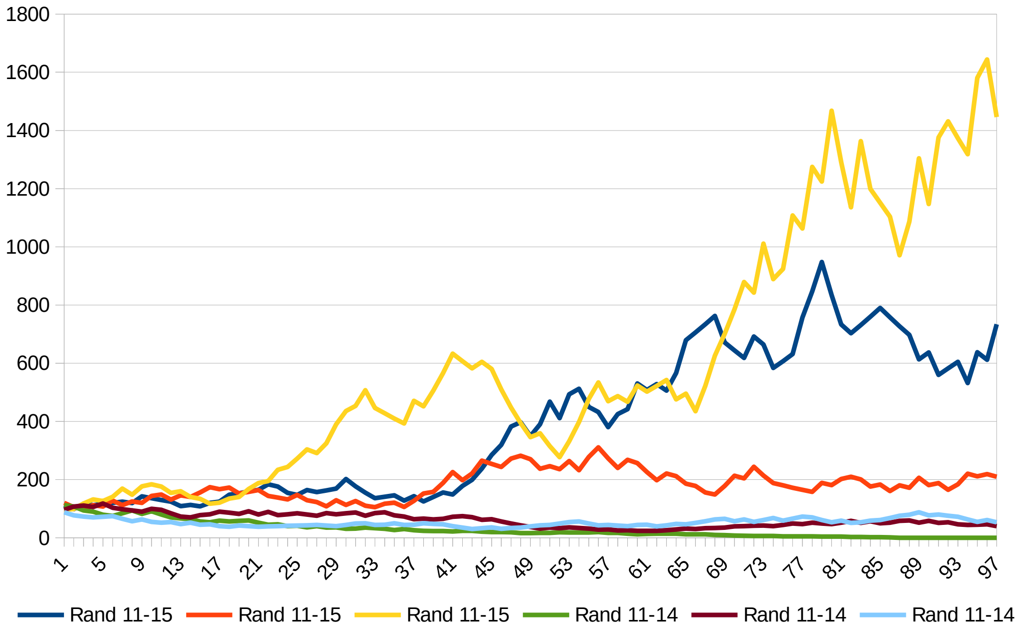


Figure 27: Using exponential input and exponential degradation. Top and bottom random values for input multiplier in the legend

```
<?php
$n = 0;

for ($i = 0; $i < 300; $i++) {
    $n = $n + rand(10, 40);
    $n = floor($n*0.8);

    print $n . '<br/>';
}
?>
```

Figure 28: Code for generating linear input and exponential degradation graph data

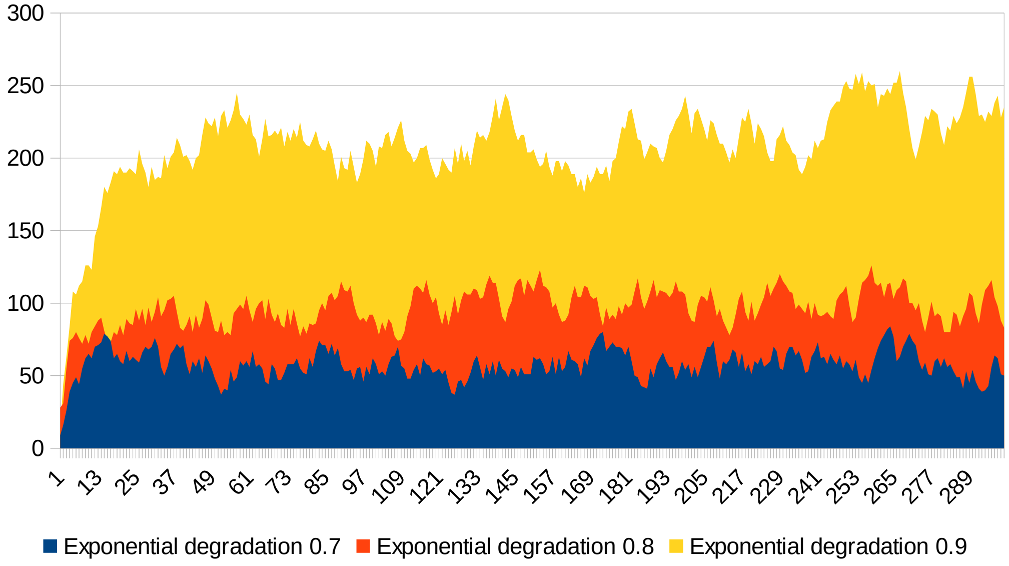


Figure 29: Uses linear input and exponential degradation. Multiplier in the legend

Figure 29, based on data generated with the code in Figure 28. That is because with a large heat value, any single added value will be an increasingly smaller percentage of the total value, while a fixed percentage of the total value is removed at each degradation event. Thus with small heat values the degradation will get increasingly smaller in absolute value, while any added heat value will be a larger percentage of the total heat value.

To provide a concrete example on why linear increase and exponential degradation is more stable; with a heat value of 1000, adding ten heat would only add one percent of the initial value, while degrading by 5% would remove 50 heat. In the case of a small heat value like 100, adding ten heat would be adding 10% to the heat value, while degrading by 5% would only remove five heat. Thus the value would by itself go towards an equilibrium between increase and degradation.

To further ascertain that the exponential degradation is fully stable, the code in Figure 28 was adjusted slightly to add a major amount of heat at long intervals, as seen in Figure 30. At the start, and every 30 intervals the code will add a fixed amount of 400 heat to the system, to see how well it copes with the rapid and sudden increase.

```
<?php
$n = 0;

for ($i = 1; $i < 300; $i++) {
    $n = $n + rand(10, 40);
    $n = floor($n*0.8);

    if (($i % 30) == 0) {
        $n = $n + 400;
    }

    print $n . ’<br/>’;
}
?>
```

Figure 30: Code for generating linear input and exponential degradation graph data

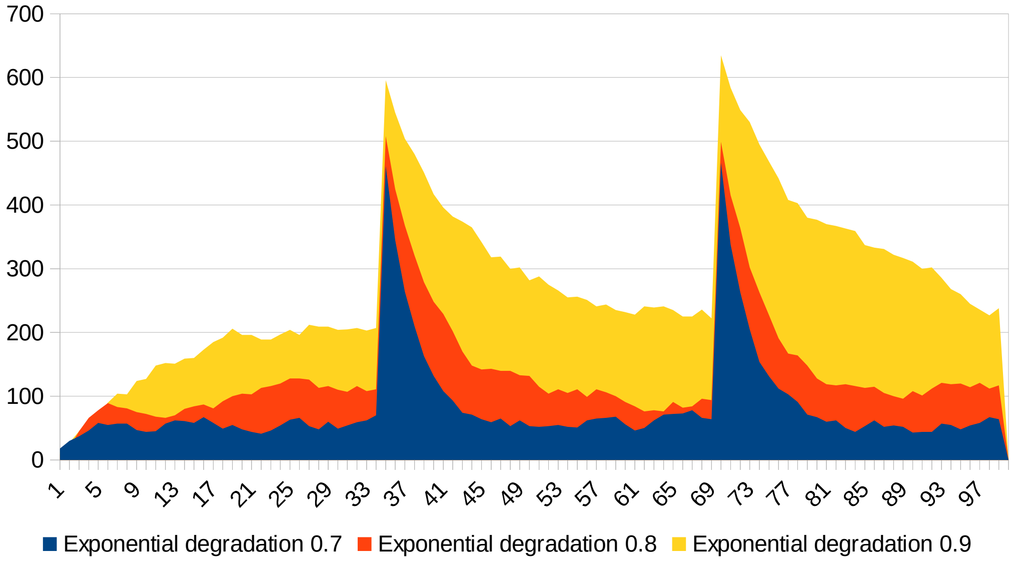


Figure 31: Uses linear input and exponential degradation. Multiplier in the legend

As can be seen in Figure 31, the exponential degradation handles such sudden peaks quite gracefully. The increase is large compared to the typical heat in the system, but it quickly subsides to ‘normal’ levels, since the size of the chunk – in absolute value – scales with the amount of heat. After it reaches ‘normal’ levels it tapers off, at just the values seen in Figure 29 where the heat input – while using a random function – is quite stable.

Concluding on Alternative

With the previous three cases in mind, we decided upon using linear value increments and exponential degradation due to its stability as Figure 29 displays nicely, thus requiring the least amount of care during operation since it is stable no matter the size of the exponent, only changing at what approximate value it reaches the equilibrium. If the heat goes very high, it will quite quickly subside so that it does not overshadow other applications for too long, and the implementation above cannot make current heat negative unless the employer configures the system with a negative multiplier.

A fair advantage, though it was not a requirement at any point, is that the formula for exponential degradation is very simple both to understand and to calculate. The formula can be seen in Equation (5.1).

$$n = n \cdot x \tag{5.1}$$

Where n is the current value, and x is the percentage of the original value to keep, in the range $0 < x < 1$.

5.5.2 Heat Degradation Recalculation

A topic of some debate was whether we were going to recalculate heat and its degradation backwards in time if an exception type severity rating or exception context severity rating was changed. Naturally it would only be recalculated for the relevant exception type and context combination, but how long to recalculate for, if recalculating at all, was a subject of some debate that was important for the general heat degradation. A linear degradation would have been far easier to recalculate, as one could simply add or remove the difference in heat value for that exception type and context combination. Since we went ahead with exponential degradation, the only practical way to recalculate heat and degradation would be to go back to a given point in time, take the heat at that time, then recalculate for all exceptions and degradation intervals in order of occurrence since that starting point in time.

Recalculating since a point in time in that way seems computationally intensive, and that was also the case one could make for not recalculating anything, simply going with the new severity rating onwards from the time of the change. While whether to recalculate will be up to Innit's choosing, some back-of-the-envelope calculations shows that the recalculation is far from as computationally heavy as one would immediately expect. Even with a worst case scenario of sequentially recalculating 100 000 exceptions having arrived over a week, with degradation every 30 seconds, would still only be about 300 000 multiplications and additions for the exceptions, and approximately 20 000 degradation events.

Because we do not need to update the database with the intermediate results for each exception or degradation, the calculation part can arguably be completed within 800 000 processor operations, potentially meaning far less than 800 000 clock cycles due to instruction level parallelism in the CPU. Even using 800 000 clock cycles, it would still only take in the order of two to three hundred microseconds. Because intermediate results and much of the input data can be cached on the CPU, much of the input data will already be present on the CPU. The only computationally intensive task that could cause issues is if hundreds of requests to the database have to be done repeatedly, causing far higher delays than the computation itself. Even with this, calculation times should even in the worst of cases stay in the order of a few tens of milliseconds.

By far the largest factor in how long this calculation would take is the time spent physically moving data. Moving data internally between the CPU and memory would not cause a significant overhead, as accessing internal RAM takes between 20 and 150 nanoseconds, depending on type and quality of RAM and motherboard. Much of the data could also be prefetched and cached on the CPU itself. It is data not in RAM that could cause issues, e.g. if several seeks would have to be done with a hard drive – each seek typically being in the order of 10 milliseconds – the transfer alone could cross a hundred milliseconds. The absolute worst case would be fetching data from a hard drive that has to spin up, something that can take several seconds. However we should generally be able to make the code fully latency safe, it is the processing time that was potentially problematic.

Despite the non-zero likelihood of significant data fetching latency; on the basis of this back-of-the-envelope calculation we agreed that processing capability would not be a cause for limiting any backwards calculation. For the calculation above, it is worth noting that 100 000 exceptions are at least one magnitude above what is expected even in bad cases, and likely two to three magnitudes more than a typical week.

In the end Innit decided, based on our advice and because they did not expect to use it much, that the recalculation gave far less benefits than the required work and potential pitfalls was worth and the feature was put on indefinite hold. Any implementation thus never got any further than tentative concept-code and was never pushed to the repository. For the concept-code, see Figure 1 in Appendix F.

5.5.3 Heat Map Data Transfer

The back-end offers an API route to retrieve all the current heat values, which the front-end then rearranges into the format is required for its current task. Because the current heat values degrade, and new exceptions can come in at any time, the front-end needs to retrieve all the data again ever so often. Retrieving everything each time is not a very scalable solution due to the potential size of the response, but some more back-of-the-envelope calculations show that for the scale of use Innit intends the response has a margin of two magnitudes in size before retrieving all the data becomes problematic. Unfortunately we cannot avoid having to retrieve all the data, as much of it is needed very often on the main overview page that also has to be running independent of input.

Innit expects in the area of 2050 applications, including the different modes (developer, testing, production), each having about 3040 exception types. With the higher numbers, this gives a total of 1200 combinations of applications and exception types. Each entry is 1020 bytes large in the response, so with the larger of the numbers, this becomes ≈ 39.06 kilobytes. This is also without any compression, which is shown in Section 7.5 to decrease the size of typically transferred data by a significant percentage.

Retrieving all the data every time is obviously not ideal, but the amount of data is almost negligible and is highly unlikely to increase significantly. In addition all the transfers will be over Innit's local network, where transfer speeds typically are far greater and the latency far less than transferring the same data over the open Internet.

There are some ways to reduce the data transferred; most simple is to periodically check whether the current heat has been degraded or any new exceptions have arrived since the last time data was retrieved. A second way is to only retrieve the data that has changed since last time, and a third would be to only retrieve the needed data just as it is required. However the latter two alternatives was put away after a lengthy discussion because it would be a fair bit of rework, and most of the data is almost always needed. The information from Innit about how many applications and exception types – information they have available already through their manual handling of all exception reports – showed that the size of the response was highly unlikely to ever get noticeably large.

5.5.4 Heat Map Data Conversion

Heat Map Data Conversion In order for the heat map to show a proper colour to indicate how much heat has accumulated, some conversion of data has to happen. Note that the heat map can show heat for all systems, any single one application, any one exception type, and any combination of exception types and applications, including one or more of all the previous.

Initially the front-end receives the heat data mentioned in Section 5.5.3, and also receives the averaged historic heat for the last period of time, of which the default is a month. The front-end then converts the current heat for the chosen total/type/application/combination to a percentage of the average value of the previous time period, using Equation (5.2). Depending on how the system is configured, the average heat will typically have a position 2 and 5 in the array of colours, where green is 1 and red is 10.

$$((n \times 100)/m)/x \tag{5.2}$$

Where n is current heat from the chosen source, m is the average heat for the chosen source, and x is an arbitrary value for reducing the result to an integer, in the range $0 < x < 10$.

The part $(n \cdot 100)/m$ gives how large a percentage the current heat is compared to the average heat value. E.g. with 126 as current heat, and average being 75, it gives $(126 \cdot 100)/75 = 168$, or 168% more than average heat. To get this value simply into a heat map colour, we have an array with 10 colours, ranging from green to red via yellow. Then one can divide the current heat percentage with x, and then round down to get a position in the array – and thus which colour – the percentage has. The value x is entirely arbitrary, and is added in this way to allow easy configuring of what colour the average value should have, and by implication which colours the current heat values will get.

As an example, if one does $\texttt{floor}(168/30) = 5$ with the heat from the previous calculation, the heat will give exactly yellow, and the reddest position will be at 300% of average heat or above. To change where yellow and red is, one can simply increase the divisor. Having a divisor of 40 gives $\texttt{floor}(168/40) = 4$, a slightly greener colour than position 5, and the reddest position is at 400 or higher percentage of average heat. Note that the function floor() is used only for readability.

6 Development Process

6.1 Tools

Version Control

We decided early on to use Git for version control. This was barely a decision as much as nobody even considering any other option. We already had quite a lot of experience with Git. It is by now the de facto industry standard for version control, and it has been greatly recommended both during our education and by friends and colleagues. Both the employer and our supervisor supported our choice.

We started out with GitHub for hosting the repositories, but due to its integration with Confluence and JIRA – which we used due to the Professional Programming course – we moved all the repositories to Bitbucket in late January. The system uses two repositories to separate the front-end and back-end systems. Neither of the repositories will be publicly available during the development process. Whether they are open after having been handed over the employer will be at their discretion.

IDE

We had early on decided we would use the IDE PHPStorm, from JetBrains. The IDE, while not important to the product's completed functionality, is central to a good development environment for the team. We had during the year before received warm recommendations about Webstorm/PHPStorm from friends and colleagues, and the team has a fair bit of experience with the related Java IDE IntelliJ, also from JetBrains. Webstorm is made for Javascript, and PHPstorm is for both PHP and Javascript; i.e. Webstorm's functionality is a subset of PHPStorm's.

In the past our team has used both Eclipse and Notepad++ for web development, but the latter is unsuitable for anything but tiny projects, and we did not have particularly positive experience with Eclipse mainly due to lacking features and missing functionality, and general discomfort with its workspace. In addition, Jetbrains supplies all students with free access to their entire suite of IDEs. In light of these factors, there was no real discussion about which IDE to use; we had informally but unanimously decided on PHPStorm even before the project began.

Task Tracker

For task tracking we started out with Trello, but switched to the Atlassian stack a month later; i.e. Bitbucket, Confluence, and the issue tracker JIRA, whereas the latter two are hosted by the university. See Section 2.2

A particular feature we were incentivised to use were the smart commits feature of the integration between JIRA and Bitbucket. It allows one to add some particular tags in the commit messages, which Bitbucket will pick up on and send to JIRA, which then updates the actual task or issue with the new state, comment, time logged, comment, or anything else added in the message. In theory this should lessen overhead since we did not have to manually update tasks on the task board. In practice however, during the first two months we always had the task board up to find the tag of the particular task, and afterwards had to check JIRA to make sure the smart commit actually worked, which it quite often did not to begin with. As we became more comfortable with the work flow and we learned to avoid any quirks, it became reliable enough that we did start to save some overhead, though the time saved was never significant. For more information about how we used the JIRA task board, see Section 2.2.

There were some inefficiencies with using a large enterprise tool however, in large part due to some trouble getting all the integrations up and working properly. Even after that, we had issues configuring JIRA due to inherent limitations in how it handled swim lanes on the Kanban board.

We did have some issues with JIRA. Amongst other things, we never found the option to archive completed tasks and stories, and thus the Kanban-board quickly filled up with months old tasks. This is not an issue had we been using Scrum, as each completed sprint is effectively its own archive. However JIRA does not seem to have very good support for Kanban, as quite a bit of complaints we have happened upon while searching for solutions also state quite explicitly. We managed to partially avoid this nuisance by adding custom filters – filters that decided which tasks were visible in the task board – though it was not an ideal solution.

Another quirk was inconsistent design between JIRA and Confluence; while they mostly had the same theme, colour scheme and such, small things like placement of buttons, implementation of contextual menus – or lack thereof – and in particular the options pages created confusion and cost us a not insignificant amount of time during the first month after adopting the Atlassian stack. JIRA is incredibly click-heavy. Everything required clicking somewhere, with a partial or, more often, a full page-refresh as the result. While not being a major issue per se, it could be fatal to concentration and pull one out of 'the flow'.

As the project progressed, we did manage to handle or avoid some of the issues, learning the peculiarities of JIRA, though we never quite managed to find it as simple and useful as Trello. It is worth noting that JIRA is an enterprise-size task tracker, while Trello generally targets small to medium sized development teams, and thus has entirely different target audiences. Had it not been for the Professional Programming course, we likely would not have started using JIRA in the first place, and might even have moved back to Trello as JIRA and Confluence was somewhat unnecessary for our purposes. We we do however not regret learning the stack as we will very likely use it again after completing the education.

Documentation Tools

As mentioned in Section 6.1, we used the Atlassian stack for much of the work. This included using Confluence as a wiki-like tool for documenting the project. We did initially document many of our thoughts and ideas in a regular LaTeXdocument, mainly because we had started to draft some ideas well before Christmas, as soon as we learned of the project and its purpose. This kept going to some degree far into January, before we got access to Confluence and got it set up properly for our use.

In addition we had some API- and format-specific documentation duplicated to a page that was returned when one called the API's root URL. This provided any user with the most necessary of documentation on how to use the API and which formats were used for input and responses without having to dive into Confluence. There was also some diagrams and general information that was only in the project report; some of which later got duplicated to Confluence.

In early February we started getting some proper documentation going on Confluence, though there were still some diagrams and documentation that were solely in the LaTeXinformation document, or that were still duplicated and not always properly updated. By the February-March month switch, all major documentation had been moved to Confluence, and was kept updated regularly there.

Confluence was somewhat unnecessary though, as the system did not really require very much documentation. Much of the documentation was due to the Professional Programming course, which effectively had us create more documentation than strictly necessary with the purpose of learning professional documentation practices. It was a good learning experience, though due to what information was required in the thesis report and what needed to be on Confluence, we never managed to avoid a significant amount of duplication, which would in most enterprise settings be considered bad practice.

Task Runner

Task runners are tools that can be set up to run groups of tasks in various situations. Typical uses are for running several other tools and/or commands in succession, often periodically or in automatic response to some other situation. Using task runners in this way both simplifies development as one can get much done with a few commands, once the task runner is set up, and it assures that the tasks are done in the exact same way every time, removing potential human errors.

We are using the task runner Gulp on the front-end to simplify and streamline the development process. Gulp is set up to automate tasks such as serving the code, wiring together dependencies, linting, running tests, and building and deploying code. For long the go-to default task runner has been Grunt, which works mostly using configuration files. Gulp is a newer task runner that has gained a major foothold, and it uses a more code-like configuration style compared to Grunt. We tested both tools, and landed on Gulp– mostly due to personal preferences in the team, rather than one being objectively better than the other. We prefer the configuration style of Gulp, and it being the newer of the tools did not count negatively. There were lots of pre-made configuration files for both, so that did not affect our choice.

Text Editor

In earlier reports we had often used Google Docs, but we have found it to be highly impractical for longer reports with larger bibliographies and cross-referencing the text. We did briefly consider Google Docs and Office365, but the preference for LaTeXwas far greater than any advantage WYSIWYG – What You See Is What You Get – text editors could surpass. LaTeXis superior in terms of writing long academic reports, handling multi-file reports and bibliographies, and the resulting output is automatically very well formatted. With that in mind we had some strong preferences for writing in LaTeX, and we would prefer to have an real-time online collaborating tool. We also considered using local LaTeXinstallations, and sharing the documents via Git, though with excellent alternatives for real-time online collaboration, collaboration via Git was passed over.

There are two major alternatives for online collaborating when writing in LaTeX, Overleaf and ShareLatex. They are both real-time online collaborating tools for writing in LaTeX, and we tested both of them for reports in the previous semester. Overleaf had an advantage in that their free community subscription allowed several people to collaborate, and only limited storage space, while ShareLatex limited on the amount of collaborators. However we found that ShareLatex's stack was open source, with in-depth documentation on how to set it up locally. Because of that we chose ShareLatex, and set up the stack on Aleksander's server.

Only later did we find that NTNU has a premium subscription for ShareLatex, and never actually needed to set up the stack on our own. Since we had already set it up however we decided to run with the self-hosted stack. The self-hosted stack does not provide the kind of assured redundant storage that the ShareLatex site provides, but Aleksander has some reasonable redundancy and backup set up, and with us regularly downloading the full source for additional separate backup, we reasoned it was more than safe enough for us. As a bonus, the private server likely provides us with far more processing power for faster compilation than the hosting sites provide.

Graph Package

As was one of our initial learning objectives described in Section 1.3, we wished to find an existing third-party package to create and display the graphs. It would have to be reasonably lightweight, provide good looking graphs, and have indications of fair activity and update schedule. Given these rather generic requirements, we discovered several packages which we started comparing. While there were good alternatives, based on our requirements we decided upon the Angular Chart package [15].

Angular Chart provides reactive graphs, meaning that it updates automatically if the input data is changed. We find it simple to implement and use, it is very lightweight, and it has a reasonably active update schedule. It is used on the status page containing the heap map, and twice on the overview page for the two graphs showing heat and number of exceptions.

Time Tracking

For tracking time spent we started out manually tracking our time spent on the project in a spreadsheet. This worked out reasonably well, and we have seen many other previous groups have done this. When the course Professional Programming started we were recommended to use a proper tracking tool, like Toggl. Toggl was recommended and it seemed quite good during our brief preliminary testing, and as such we moved all our tracked time to Toggl in mid-January and used it from there on. All the time logs can be seen in Appendix G.2.

There are some not entirely insignificant differences in how each team member tracked time spent. While Vegard and Olafur incorporated or tracked time spent for eating or other non-project activities, mostly as ‘no project’ or ‘other’, Aleksander skipped tracking such activities, and Lars tracked only time explicitly spent on specific issues in the issue tracker. This causes some noticeable differences in total time tracked, though it should be noted that all team members have spent approximately the same amount of time working on the project.

The differences in time tracking mainly stems from different types of work and the practicality of separating out tracked tasks. For instance, report-work does not use the issue tracking that development does, making it less practical to track it in regards to issues, or any similar handle.

Also note that the time logged is somewhat incomplete because work with the thesis presentation and potential deployment-related work for Innit takes place after thesis delivery. It is uncertain how much this will amount to, but it will likely be in the proximity of 200 hours in total for the team.

6.2 Data Model Development

This section will be describing how the data model described in Section 4.6 came to be. It was a long and far more dynamic process than we expected. We had from earlier courses been instructed to – as far as possible – understand the data that would be required and design a data model for it during the initial stages of a project. Of course we have had mishaps in former projects that have required updates to the data model, but none of it had led us to anticipate the scale of changes the data model would undergo during the projects development phase. While the data model seen in Figure 15 is a little expansive for a relatively small project, it originally consisted of the exceptions, exception_types, installations, applications, and syslog tables. Note though that we did not entirely expect the data model to stay fixed; we knew some kind of severity rating and perhaps some storage of historic heat would be necessary. However going from five to eleven tables was in the least an unexpected result, though looking back it was arguably inevitable.

The first addition was the severity_table in mid-February, containing the severity rating of each exception type. This was a employer wish in order to sort by severity. In addition it was necessary for calculating the heat from each application or installation, for the front-page overview graphs and heat map.

The second and far larger addition was after the demonstration of the system in early March. Requirements of how severity is set and stored changed a fair bit to the side of significant rework and additional work was required. We initially got the impression that severity was a simple value based on which type of exception it was, expanded to making heat value be a product of two separate severity values, where the first is a severity value for each exception type for each application, and the second is a far more dynamically set severity value based on the exception context, i.e. where in the application code it occurred. For additional discussion of how the heat and severity values are handled, see Section 5.5. This addition thus required – after quite a lot of group discussion of how to do it – the exception_context, current_heat, historic_heat, settings_changes and the original_objects tables which were all added over the course of two weeks.

The three-way relation necessary between exception_types, applications, and the recently created severity_ratings tables proved to be a large issue, as our ORM Eloquent does not support three-way relations, and it was stated by the author that such support will not be implemented. To circumvent this issue a solution that only partially used Eloquent functionality had to be improvised, where the remainder of the query used raw SQL-statements. For instance the relation could not be set up using Eloquent, and demanded some less than ideal solutions, since querying the relation cannot be done too well with the standard Eloquent query-builder.

Amongst the details discussed with Innit was whether an application should be able to have different states, i.e. development, testing and production. Innit greatly wanted this, and in order to enforce the difference we had to implement composite/partial primary keys for the table in question. However Eloquent does not support composite keys out-of-the-box [16], causing some further issues with setting up the table with the composite keys, and querying them properly. We were thus left with three options; create unique keys for the pairs of keys in relation tables, extend the Eloquent framework ourselves to support composite keys, or scrap Eloquent altogether.

Because scrapping Eloquent at this point would be quite problematic, requiring quite a lot of rework of our database interaction, in addition to Eloquent being the default ORM and almost built into Laravel, the option was never really considered a real alternative. Extending Eloquent ourselves or, more likely, adding code that had already been developed by others that have had the same issues with Eloquent, was a considerably better alternative, though at the time it seemed as creating unique keys for each of the pairs of keys would require the least work, or in the least break the lest amount of functionality that would be hard to fix.

With the knowledge of experience, in retrospect extending Eloquent seems it would have been the best option. Creating unique keys seemed simple and the least error-prone, but it proved not to be so simple, since an internal MySQL-/MariaDB-limit on key prefix length limits the length of unique keys to 767 bytes, or 255 characters (a limit in InnoDB). The exception contexts table required unique keys, but because one of the columns that was to be used was set to a maximum of 2048 characters, we could not set it unique. In the end we decided to reduce the column to 255 characters, and accept any potential issues it could create. Thus if a system submits an exception report with a file path longer than 255 characters, it will cut off at 255 and potentially not be unique. If this happens we might be forced to abandon the unique check for this table altogether. Cutting off like that is naturally not an ideal solution, as cutting off the remainder of a path in itself can cause it to be equal to another entry's path, and thus fail the unique requirement. See Section 8.3.2 for additional thoughts on the matter.

6.3 Issues and Bugs Encountered

6.3.1 Front-end

Static Data Storage

During the project's initial stages, it seemed like a good idea to store much of the static data, such as exception types and application names in the browser's local storage. Saving data we rely to be up-to-date on the initial page load, however seems like a questionable solution in hindsight. Since every exception the system receives can be from a new system or be of a new type, storing these definitions hold no real benefit. The intention was to reduce the amount of requests sent to the back-end by avoiding asking for data that had already been retrieved. Despite the good intentions, keeping the data up-to-date proved to be nothing but unnecessary overhead. The locally stored data can never be trusted to be up-to-date, so every new exception has to check if the data required is available, and request it if necessary. Since those parts of the data will likely always be cached in server memory, the only savings would be a tiny – on the order of a few kilobytes at most – every minute or so.

Usually these issues are easily circumvented by simply checking and/or retrieving any data that has arrived since the last check/retrieval. However we found during testing that the checking would not reduce the data transfer significantly enough to warrant the increase in code complexity. In fact we found retrieving such static data every time to be the simplest solution, causing the least amount of delay for the user.

Editing Severity Ratings

One of the features the employer wanted was a the ability to edit the severity values of both exception contexts and exception types through the front-end. An issue that surfaced here – and is mentioned in the Heat-Section 5.5 – is how to handle other front-end users when a value has been updated, in particular because the system is stateless. The obvious fix is making the front-end poll the back-end continuously for any changes to severity values. Exactly how to handle such polling however was not without its discussions, as checking each and every timestamp on the severity ratings entries would have been far more work for the database than we deemed acceptable. Thus we avoided the issue by having an entirely new database table, the settings_changes table, whose single purpose is to contain the latest timestamp for when any severity rating has been changed, see Section 4.6. Creating single tables for such a use case is generally frowned upon, but we could come up with no solution that would be as fast and simple, while not requiring significant changes to the data model.

The front-end polls the back-end continuously, and whenever a client recognises that its data is outdated, it prompts the user to refresh the page. This refresh is also done automatically if no response is given within 30 seconds, for instances of the front-end such as the TV overview screen mentioned in the Project Description in Section 1.2. The user can cancel a refresh and even disable the polling feature completely for the duration of the session, if working with out-of-date data is acceptable.

Front-end Graph Data Format

Formatting the statistical data from the back-end into a format that the third party graph package demands required a large amount of highly complex code. In fact, the grapher.controller-file that contained the logic for converting the graph data into the correct format was by far the largest file in the project based on lines of code. It was also contained an unreasonable amount of the most complex code. This reformatting is also mentioned in Appendix E.1.1

Upon reaching the stage of feature completion at the end of April, the graph controller underwent thorough consideration about how it could be improved. We realised that the complexity of the controller was in large part due to several differing formats on the returned data from the back-end, much of which required its own very similar code to handle.

To exemplify, combinations of application and exception type statistics we needed to display.

- Selected exception types for one application
- Selected exception types for two or more applications
- Selected exception types for all applications
- One exception type for all applications

- All exception types for one application

While some may seem overlapping, they had to be handled differently because they were to display the data in different ways; e.g. aggregated number of exceptions per selected application, or number of exceptions per selected application and exception type combination. While it did not give risky edge cases, the different cases were just different enough that the code could not be generalised and combined without making it completely unmaintainable.

After some debate in the team, we found that the logic could be significantly simplified by having the back-end serve the data in a manner consistent with how the graph package requires the data.

To interject, this kind of adhering to a package's required format is generally grossly frowned upon as it potentially introduces major coupling between the back-end and the front-end, something both Innit and we wished to avoid. However, in this case we deemed it worth the partial coupling because of the greatly simplified code and increased maintainability.

We added a /graph main route for retrieving the graph data. See Section 5.3 for more information on the API. The functionality was separated out as the dedicated /graph-route in order to reduce the coupling to a minimum, and because we did not have time to change all the formats in all the routes. This also gave us the opportunity to test whether such a unified format would actually work without breaking large parts of the front-end relying on the current API. We found that having a unified format – while not the most practical format in all situations – in total would prove beneficial, and we would have changed all routes to use the same format given sufficient time, thus avoiding coupling altogether.

The change is thus not only negative due to introducing coupling between the systems, but is arguably positive as a proof-of-concept ready for a potential later change that would be beneficial to the whole system, regardless of the graph package.

As concrete benefits the graph.controller-file and related logic in other files was shrunk by far in excess of a thousand lines of code, much of which was the most complicated and difficult to maintain in the entire front-end, and replaced with simple back-end API calls. The back-end did have a significant increase in lines of code, in the order of 600 lines of code, though far less than the decrease in the front-end. The added code in the back-end is also far less complex since the routes are completely separated from each other, and most of the added code are merely handling of edge cases.

Added benefits of this change includes a noticeably shorter delay for the user in displaying the graph, though we do not have exact numbers on this. Creating the format on the back-end when it is first retrieved is far more efficient than first retrieving it on the back-end, then going through code with a processing time scaling of O^2 or O^3 – depending on the data to show – in order to transform it to the graph format. While the amount of data was small and is likely to stay that way, thus not being important that the transformation scaled so poorly, it was still fully avoidable and preferable to do so.

Figure 32 shows the format of the /graph main route. Note that in any one response it will only have either 'application', 'exceptiontype', 'mixed' or 'total' on the second level in the response-array, not mixed in as they are in the figure. We have simply added them

```
{
    "graphs": {
        "application": {
            "url": "application\/{id}",
            "description": "Gives graph data for a single
            application. Exception type IDs must be supplied."
        },
        "exceptiontype": {
            "url": "exceptiontype\/{id}",
            "description": "Gives graph data for a single
            application. Exception type IDs must be supplied."
        },
        "mixed": {
            "url": "mixed\/",
            "description": "Gives graph data for mixed types.
            Exception type IDs and Application IDs must be supplied."
        },
        "total": {
            "url": "total\/",
            "description": "Gives graph data for all exceptions
            (Total). Given as a single data entry"
        }
    }
}
```

Figure 32: Format the /graph route returns.

together in order to show what data it can contain. Figure 33 shows the format with a single type of data, as the response looks like in reality.

6.3.2 Back-end

Heat Degradation Trigger

While working on the heat degradation functionality in the database mentioned in Section 5.5.1, we encountered an issue with the heat degradation trigger. It was set to run every minute, passing over every record in the current_heat table. This event in turn would fire off a procedure that degraded the heat in each record, with the degradation based on how long it had been since the last degradation event for that record – we reasoned we had to check time since last degradation, since concurrent events could cause a table to be locked, making a degradation event late.

As we tested the main event procedure, we kept getting a result where the database said the rows were affected – aka. a successful operation – but the table was left in a state where one could not update any values. Debugging the issue took several hours before we found the cause of the issue. Because we expected internal loops to have their own local scope, an iterator name was reused. However we found that the variables were all in a global scope, where the iterator would be reset to 0 each time the internal loop degraded a single record in current_heat table, whereas the outer loop would never reach the end, and thus remain in an undefined state when all the entries had been passed over. Giving the internal iterator another name fixed the issue.

```
{
    "graphs": {
        "application": {
            "url": "application\/{id}",
            "description": "Gives graph data for a single
            application. Exception type IDs must be supplied."
        },
        "application": {
            "url": "application\/{id}",
            "description": "Gives graph data for a single
            application. Exception type IDs must be supplied."
        },
        "application": {
            "url": "application\/{id}",
            "description": "Gives graph data for a single
            application. Exception type IDs must be supplied."
        }
    }
}
```

Figure 33: Format the /graph route returns for applications.

Note that because the typical amount of exceptions incoming per day turned out to be far lower than we initially planned for, degradation was later changed to every half hour making a few seconds margin of error was fully acceptable, in turn allowing us to remove the checking of the number of seconds since last degradation.

6.4 Working with Live Data

Our system have been receiving live data from Innit since late March, and this bears with it the potential for receiving potentially sensitive person and business data. This was noted as a risk in the project plan, see Appendix 9, and to alleviate this risk we decided early together with the employer that we would not build the system with all the requirements for handling personal data. Despite this, we did assume it could happen and set all reporting traffic to only go over HTTPS. This turned out to be a reasonable assumption, as we several times throughout the spring received sensitive data.

The issue lies in that the data we received was live production data, exceptions generated by real world users and potentially containing confidential or otherwise sensitive data. Because of this, the first order of business for Olafur and Aleksander every day, including weekends, was to manually check all exception reports that had arrived the past 24 hours and look for data like social security numbers, passwords, uniquely identifying information such as full addresses and phone numbers, and other similar types of data. If we found them, we would write them down, purge the database, and notify Innit, sending them the data so they could find where it originated and modify their reporting module to remove such data before sending the exception reports.

Receiving real world data has been a great tool during development for testing our solutions, and asserting full real world functionality of all system features. However it also did cost us some time in finding, logging and reporting all such potential pieces of data. In addition, every time we received anything highly sensitive, like social security numbers, we purged the back-end's database. This was somewhat impractical as in particular testing the graphing functionality often required several days worth of data to fully test. The full purging was because the API was publicly available – though not searchable by search engines – since receiving live data required that Innit had access to the API in order to POST data, and the API was without any authentication.

Note that authentication was never a part of the requirements, as this will not be an issue when the system is fully deployed at Innit's offices, inside their local network, thus with no external access to the system.

7 Testing and Quality Assurance

7.1 Code Review

From the first stages we have had a strict rule that all committed code must be reviewed by another team member. This has been implemented through always working on a repository branch instead of master, and that a pull request into master must be reviewed and accepted by at least one other team member.

Naturally the size of the pull request somewhat limits how strict the rule is. Often we have been fine with a single other team member reviewing the code, with the precondition that all unit tests and linting passed without issues.

We have also had some unofficial norms on the size of the pull requests in order to make reviewing them simpler; it was expected that one attempted to limit the size of each pull request to a maximum of a few hundred lines. This was on large part because Bitbucket did not have code highlighting in the pull request view, making reading larger amounts of code quite taxing. Of course avoiding large pull requests was not always possible, and there were at several occasions requests of up to a thousand lines of code. The most important was however to keep commits at a reasonable size, preferably not more than 200 lines of code per commit. Most commits were far smaller than this, since as a general rule a commit should only cover a single issue.

Limiting the size in this way, and having strict rules on code review helped ascertain that committed code was sufficiently following our rules and guidelines for code structure and readability, see Section 4.2 about coding standards. There were often comments added on pull requests that certain parts should be clarified with more or better comments, and some parts should be restructured somewhat. Pull requests were therefore declined on a reasonably regular basis, an action often considered to be an important hallmark of whether such code review rules actually work and are accepted by the team.

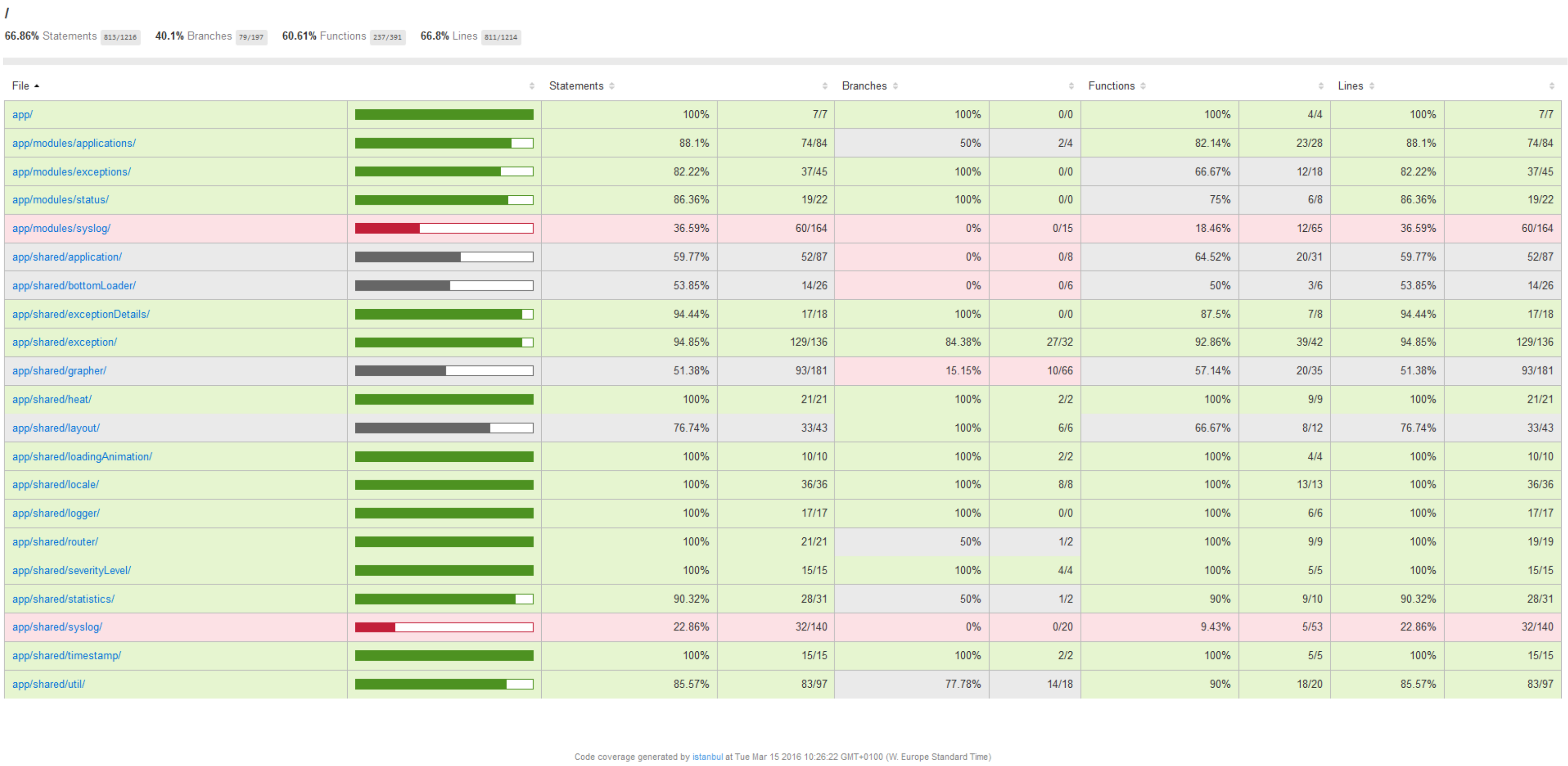
7.2 Unit Testing Tools

7.2.1 Front-end

Thanks to AngularJS' focus on testability, writing unit tests for the front-end application was made relatively easy through the use of Jasmine and Karma, which work well with AngularJS and has a lot of official documentation available.

Karma is a task runner for tests, much like how Gulp is a general task runner; it sets up a temporary web server and runs all the Jasmine tests. Jasmine is a 'Behavior Driven Development' testing framework for Javascript which does not rely on browsers. Jasmine sets up and mocks the environments that are required for each test, and then runs through the testing code, see Section 7.2.1 for how mocking dependencies is done. One simply describe an easily readable condition for software – i.e. what values and formats to expect, not 'it should look like this' –, a function to run, and what Jasmine should expect.

Having automated tasks run the tests make unit testing an semi-automatic feature of



/

66.86% Statements 813/1216 **40.1%** Branches 79/197 **60.61%** Functions 237/391 **66.8%** Lines 811/1214

File ▲		Statements		Branches		Functions		Lines	
app/		100%	7/7	100%	0/0	100%	4/4	100%	7/7
app/modules/applications/		88.1%	74/84	50%	2/4	82.14%	23/28	88.1%	74/84
app/modules/exceptions/		82.22%	37/45	100%	0/0	66.67%	12/18	82.22%	37/45
app/modules/status/		86.36%	19/22	100%	0/0	75%	6/8	86.36%	19/22
app/modules/syslog/		36.59%	60/164	0%	0/15	18.46%	12/65	36.59%	60/164
app/shared/application/		59.77%	52/87	0%	0/8	64.52%	20/31	59.77%	52/87
app/shared/bottomLoader/		53.85%	14/26	0%	0/6	50%	3/6	53.85%	14/26
app/shared/exceptionDetails/		94.44%	17/18	100%	0/0	87.5%	7/8	94.44%	17/18
app/shared/exception/		94.85%	129/136	84.38%	27/32	92.86%	39/42	94.85%	129/136
app/shared/grapher/		51.38%	93/181	15.15%	10/66	57.14%	20/35	51.38%	93/181
app/shared/heat/		100%	21/21	100%	2/2	100%	9/9	100%	21/21
app/shared/layout/		76.74%	33/43	100%	6/6	66.67%	8/12	76.74%	33/43
app/shared/loadingAnimation/		100%	10/10	100%	2/2	100%	4/4	100%	10/10
app/shared/locale/		100%	36/36	100%	8/8	100%	13/13	100%	36/36
app/shared/logger/		100%	17/17	100%	0/0	100%	6/6	100%	17/17
app/shared/router/		100%	21/21	50%	1/2	100%	9/9	100%	19/19
app/shared/severityLevel/		100%	15/15	100%	4/4	100%	5/5	100%	15/15
app/shared/statistics/		90.32%	28/31	50%	1/2	90%	9/10	90.32%	28/31
app/shared/syslog/		22.86%	32/140	0%	0/20	9.43%	5/53	22.86%	32/140
app/shared/timestamp/		100%	15/15	100%	2/2	100%	5/5	100%	15/15
app/shared/util/		85.57%	83/97	77.78%	14/18	90%	18/20	85.57%	83/97

Code coverage generated by istanbul at Tue Mar 15 2016 10:26:22 GMT+0100 (W. Europe Standard Time)

Figure 34: Screenshot of the Karma coverage dashboard from March, mid-way in development. Each row is a module in the front-end.

the setup. For instance, each time we run Gulp for setting up the web server it runs the unit tests, and also does linting of the code using JShint and JSCS, to spot inconsistencies and errors. The most common way to run the tests is simply using 'gulp test'. We could have the unit tests run automatically whenever the code is updated or the file is saved, but because the unit tests takes some time to complete, we have opted not to do that. In addition fully automatic testing tended to be a nuisance during development. Hence we manually run the unit tests through Gulp, though it is recommended to do so quite often.

Jasmine and Karma also provides us with excellent coverage reports, thorough code examination, detailed code review and complexity reports, as seen in Figure 34. The figure shows an example of the resulting Karma dashboard after all the tests have been run. The first row is a high level overview of the test coverage of the entire system, i.e. the /app root folder. All the rows after that are a breakdown of the test coverage per folder – effectively being per module, as a folder in AngularJS most often is equivalent to a module – showing percentage of statements, conditional branches, functions and lines of code that are covered by tests.

In addition we have Plato, which is a static code analysis tool that provides linting, along with statistics about lines of code, complexity and maintainability values, and estimates of errors in the code. If run often it also gives a good historic record of the development and changes in e.g. complexity, maintainability and estimated errors.

Front-end Mocking

As is typical in unit tests, every module in the front-end is set up individually and all the functions are tested in isolation. When setting up the environment for the module, the module will assume that all the other parts of the program has been created and will be available for use. These other parts have to be mocked to make the testing environment robust and completely reproducible; all the dependencies are replaced with static mocked objects which offer the same functionality the module or function being tested expects is available.

For instance the factory that handles exceptions expects that a service is available for requesting exception data from the back-end. The code in Figure 35 is a sample of how such a mocked service is created with the functionality expected, and injected into the exceptionFactory. The real service's functions are also tested, but in their own separate unit tests.

In Figure 35 the mocked service is created with two functions, search and getExceptionTypes. When one of the tests run code that asks the service for one of these functions, instead of requesting data from the back-end, the mocked object simply returns static data. AngularJS handles the details of how this is handled, but essentially the factory uses dependency injection to get whatever dependencies it needs, which makes it easy to just give it a fake dependency. The function being tested remains blissfully ignorant of where the dependencies came from and how they are implemented, as long as they provide the functionality the function requires.

Since AngularJS is built with dependency injection and mocking in mind, there are libraries available for mocking typical cases such as HTTP-calls, for when we are testing the services themselves. The HTTP-mocking module hijacks all HTTP-requests and

```
module(’H3E.exceptionFactory’, function($provide) {
    exceptionService = {
        getExceptionTypes: function() {
            var q = $q.defer();
            q.resolve([
                {id: 1, name: ’type 1’},
                {id: 2, name: ’type 2’},
                {id: 3, name: ’type 3’}
            ]);
            return q.promise;
        },
        search: function() {
            var q = $q.defer();
            q.resolve([
                {id: 1, exception_type_id: 1},
                {id: 2, exception_type_id: 3}
            ]);
            return q.promise;
        }
    };

    $provide.value(’exceptionService’, exceptionService);
});
```

Figure 35: Sample code for mocking a service

returns a static set of data, just like the regular mocking of objects.

7.2.2 Back-end

Laravel is shipped with and configured for using PHPUnit as the default system for running unit tests. Unlike the elaborate set up on the front-end with several different tools and automation, the back-end stayed so small and simple that we initially intended to have the tests run automatically quite often. However, because we run the unit tests in total isolation, all the local database tables are set up and inserted data into before each single test, and then dropped again after each test. This makes unit testing in the back-end a rather slow endeavour, typically taking a minute each time.

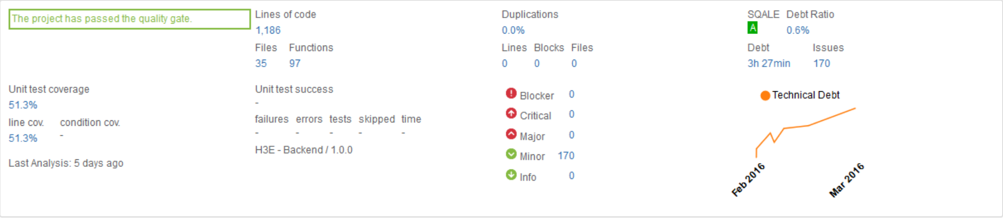
While developing we also run the code continuously against the local database, testing that it affects the tables correctly. If the unit tests were run continuously, they would have to drop and re-create the tables for each test to avoid duplicate unique keys, and this would greatly interfere with the manual testing while developing.

For these two reasons in particular – that the unit tests are slow, and manual testing while developing – we opted not to have unit tests run automatically on the back-end.

7.3 Static Analysis

As mentioned in Section 7.2 we used Sonar and SonarQube for static analysis of the code base – both front-end and back-end – due to former experience using the tools, and knowing they are quite useful and reliable. For the back-end, Sonar was coupled with the Xdebug PHPUnit module for code review.

Sonar was hosted on Aleksander's server, where we pulled down all code pushed to the respective master branches, and both run the code through analysis and deploy the system for integration testing, see Section 7.4.



The project has passed the quality gate.

Unit test coverage
51.3%

line cov.	condition cov.
51.3%	-

Last Analysis: 5 days ago

Lines of code
1,186

Files	Functions
35	97

Unit test success
-

failures	errors	tests	skipped	time
-	-	-	-	-

H3E - Backend / 1.0.0

Duplications
0.0%

Lines	Blocks	Files
0	0	0

Blocker	0
Critical	0
Major	0
Minor	170
Info	0

SQALE	Debt Ratio
A	0.6%

Debt	Issues
3h 27min	170

Figure 36: Screenshot of the Sonar results after an analysis

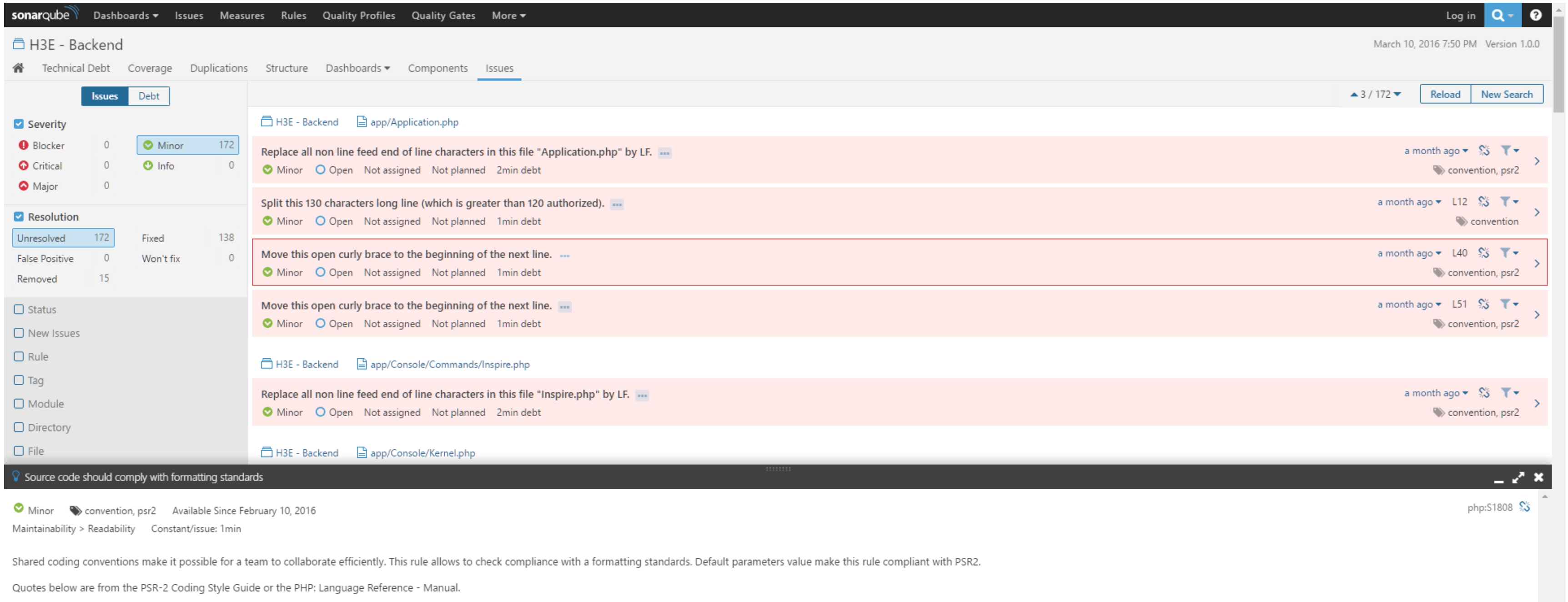
Figure 36 displays the Sonar Dashboard's main overview of the back-end, with the most important pieces of information. However, since Laravel is not directly supported in Sonar, there is an error causing it to show that no tests has been run in the second column from the left; the line 'Unit Test Success' and the line below it. Equal to how Karma shows the test coverage on the front-end in Figure 34, the Sonar dashboard shows test coverage of the back-end in the leftmost column, along with the number of files, functions and lines of code in the second column from the left.

The rightmost column shows the estimated percentage of technical debt – code that should be cleaned up and generalised – along with an estimate of how much time it would take to clean up all the technical debt, and a graph of the technical debt over time. It should be mentioned that the technical debt estimated time is far from accurate on a small scale, and we have earlier found it to be off by several hundred percent in either direction. However on the large scale it might be a good approximation, assuming the very wrong estimations at least partially cancel each other out.

At the top of the third line from the left, the percentage of code that has been duplicated and potentially can be merged and generalised is shown. The perhaps most important part of Sonar are the five indicators below that; the amount of each type of issue displayed in descending order of importance. Clicking on the number brings one to another display where one can browse the reason for every issue, their exact position in the code, and often a suggested way to fix the issue.

Figure 37 shows how the issue overview looks. Since there are no blockers, critical or major issues, it is set to display minor issues, containing issues such as code not following coding conventions. Each pink line is an issue containing some information about it. When one clicks on one of them, a larger window appears as seen in the bottom of the screenshot, containing reason for marking the issue, more information about it, and often some information or suggestion as to how it can be solved.

Note that some issues in Figure 37 are in the Laravel files, not our files. For instance, the app/Console/Commands/Inspire.php and the app/Console/Kernel.php files are Laravel's own files. Thus the default Laravel files does impact our issue rating somewhat since we had not at the time excluded all the default files, but the impact was not very large.



H3E - Backend app/Application.php

Replace all non line feed end of line characters in this file "Application.php" by LF.
Minor Open Not assigned Not planned 2min debt — a month ago ▾ — convention, psr2

Split this 130 characters long line (which is greater than 120 authorized).
Minor Open Not assigned Not planned 1min debt — a month ago ▾ L12 — convention

Move this open curly brace to the beginning of the next line.
Minor Open Not assigned Not planned 1min debt — a month ago ▾ L40 — convention, psr2

Move this open curly brace to the beginning of the next line.
Minor Open Not assigned Not planned 1min debt — a month ago ▾ L51 — convention, psr2

H3E - Backend app/Console/Commands/Inspire.php

Replace all non line feed end of line characters in this file "Inspire.php" by LF.
Minor Open Not assigned Not planned 2min debt — a month ago ▾ — convention, psr2

H3E - Backend app/Console/Kernel.php

Source code should comply with formatting standards

Minor convention, psr2 Available Since February 10, 2016 — php:S1808
Maintainability > Readability Constant/issue: 1min

Shared coding conventions make it possible for a team to collaborate efficiently. This rule allows to check compliance with a formatting standards. Default parameters value make this rule compliant with PSR2.

Quotes below are from the PSR-2 Coding Style Guide or the PHP: Language Reference - Manual.

Figure 37: Screenshot of Sonar's issue overview

7.4 Integration Testing

During the development we set up an external system which we used for system and deployment testing for testing our system. This was a self-hosted solution hosting a functional front – and back-end on Aleksander's private home server. The server were configured to automatically pull/fetch any updates merged into the the git repositories' respective master branches, and the setup is deployed automatically but the testing/feature-run is done manually.

The solution was used by Innit to test-run their newly re-designed exception modules, and already in late February we started receiving exception reports from their reporting modules. The test-system was set up with HTTPS as the information reported could be of a compromising factor. This proved to be worth it, since we several times received usernames and passwords together, national identification numbers, full names, date of birth, address, e-mails, and more.

We also received an enterprise grade level L2/L3 switch from Innit which supported syslog, that Aleksander used and incorporated into his local network to retrieve the syslog formats used and test the syslog-related functionality of our solution.

Upon actual deployment of the system, our own 'HEEE' infrastructure will be dismantled and deleted. Everything will be hosted locally at Innit's offices, on their local network behind their firewall, and thus requiring less strict security. Thus the added security currently implemented will not be necessary once the system is finished and deployed.

7.5 Performance

Creating a lightweight back-end which potentially has to serve thousands of requests in parallel – it is highly unlikely to reach those numbers, but we have to assume it can happen – without noticeable service degradation is both important and quite difficult. In order to test as close to real-world performance as we could we chose to use the popular tool iperf. It is a specialised network performance testing tool, capable of generating a large amount of requests to web servers in order to ascertain their performance, scalability and how well the service degrades under massive load. Our with iperf testing revealed that the back-end can quite easily handle 10 000 queries per second, given that the bandwidth is not a bottleneck.

In addition we ran Google's PageSpeed analysis, which uncovered that Gzip was not enabled – despite us believing it was – and could, if enabled, save approximately 80% of bandwidth usage between the back-end and any software using the API. Potentially enabling such compression could negatively affect the back-end's performance, and since the front-ends can be quite a lot weaker, the decompression could be noticeable. However after enabling Gzip we did not notice any increase in server CPU usage or I/O, other than the fact that the amount of data sent and received was significantly smaller. In fact the performance hit was so small as to be unmeasurable within the margin of error. Thus Gzip will be enabled by default in the back-end.

We tested our infrastructure using a VM on Aleksander's server, which has a maximum network bandwidth of 100 Mbit downstream and 10 Mbit upstream. This is respectively one and two magnitudes lower than the server on which the back-end will be hosted when in use, which has a 1000/1000 local connection at Innit's premises. Since we had a limited uplink in the testing environment we needed to make an efficient back-end from the start that was able to receive exception reports from Innit while we were developing and running tests.

The VM was allocated 4 vCPU and 2GB of RAM plus a 100GB vDisk running the operating system Xbuntu 14.04 LTS (Ubuntu with XFCE). We tested first placing the DB and OS on a current generation SSD which gave us expected high results when we queried the DB continuously. Later on we tested the system by storing it on a regular hard drive, which has much slower IOPS than a SSD. This worked out quite well, even when the database was hammered with queries.

Considering that the back-end will be only used for in-house purposes, and based on those results, Innit should have no performance issues in their local environment within two magnitudes of the anticipated usage pattern.

Heat Recalculation Performance

One potential performance bottleneck is the case where several days of heat recalculation had to be done when a severity rating was changed. We had to consider this carefully, both how to retrieve the data for the recalculation and how to do the computation itself. However after a short back-of-the-envelope calculation in Section 5.5.2, we realise this calculation will not be anywhere near the bottleneck we feared it could be.

Since the heat recalculation has been put on indefinite hold, any concerns about heat recalculation performance issues are no longer as important.

7.6 Demonstrations

Per the original schedule we were going to have three demonstrations of the system as it neared completion: early march, late march, and a final full system demonstration and acceptance test in late April. Due to our unexpectedly fast development pace we had a full-system demonstration already on March 7th, which also served as an early acceptance test of the features and design. Neither the front-end or back-end were complete, and not all features were ready to use yet, but most was completed sufficiently that the design theme and layout, data retrieval, much of the processing, and all major architectural decisions had already been completed.

We found the demo in total to be quite beneficial to us, as we in the weeks before had less communication with the employer than usual due to heavy workload on their side, and thus had made some design decisions we could not be entirely certain the employer would like. In addition the employer had not seen prototypes earlier that were anywhere near as complete, and they could have decided we had went off in the wrong direction. However all our worries were rendered moot as he employer was greatly satisfied with what we had done at the time. We got some very good feedback for improvements, in addition to some requested changes to the heat map / severity requirements which required a bit of rework.

We also had a meeting in late April to showcase our near final product. They were very satisfied with out work and was amazed at how well the application had formed. We received several pointers on some small graphical improvements they would like so we could improve it further design-wise before the deadline.

The last demonstration was done over a Skype conference call May 13th before final delivery in May, where we showcased the ultimately final product to them with the proposed changes from our last meeting in late April.

7.7 GUI Feedback Demonstration

As with most projects that have a GUI, it is both useful and often necessary to get a third-party to review the design decisions. Because our team consists of only developers having little knowledge and experience with user interaction and UI design, we took the opportunity that arise from working at a university campus; to have an expert in the field of UX- and UI-design, assistant professor Eivind Arnstein Johansen, review our system.

In mid-April we had a meeting, using him as an complete outsider with no previous knowledge of the project and thus completely lacking any bias towards our design decision. We were rather surprised and satisfied that he had nothing in particular to note on the overall setup and design, which using the Bootstrap-framework – see Section 4.1 – used responsive design and was generally mobile-ready at the time. In addition to this we received valuable input on details we had overlooked, such as how to make the lists more readable, colour usage and colour contrast, size and spacing of elements, and other such small but very important details. As an added bonus he gave some tips regarding what content we could add on the application overview site, a page that had been mostly empty at the time.

8 Discussion

8.1 Target Achievements

8.1.1 Achieved Learning Objectives

A central objective for our team is whether we have fulfilled the learning objectives stated in Section 1.3, and how we have incorporated the fields of study described in Section 1.5 into our work.

Foremost bears mentioning the frameworks we have used; while the division of workload have significantly affected how fluent each team members can be said to be in each of the frameworks, we consider this objective fulfilled. Olafur has had the main responsibility for the back-end development and thus Laravel, and Lars has had the main responsibility for the front-end and thus AngularJS. Despite the responsibilities, all four team members have gained notable insight and experience with the frameworks, both with general development and in a professional development situation. Similarly all team members have gained significant experience with the Kanban development methodology, described in Section 2.1, and the use of professional development tools such as: JIRA, described in Section 2.2 and Section 6.1, Confluence, described in Section 6.1, task runners, described in Section 6.1, and code analysis tools described in Section 7.2 and Section 7.3.

Integration of a third-party tool for graphical representation has been fulfilled through the integration of a Javascript graph package, see Section 6.1. A module-based system has been achieved on the front-end through the module structure that is built into AngularJS, where the file and folder structure represents both modules and in large part their dependencies. On the back-end it has been achieved through how main routes are separated as an inherent side effect of the RESTful API-structure. Using a operations defined standard has been done by using the IEEE-defined syslog-standard, see Section 4.5.

Working for an commercial employer to create a product to be used in an commercial setting has been a major learning experience for the team. While the employer took care to support the development as an academic project, we have gained valuable insight from having worked in the intersection of academic and commercial settings. In particular the team had to develop the entire system and handle live data in a professional manner, where the live data could potentially contain sensitive person- and business information, see Section 6.4.

Through achieving these learning objectives, we have visited and developed our understanding of the designated fields of study, see Section 1.5. RESTful API-design has been used to great extent in the back-end's API, and both the front-end and the back-end have been structured in large part from the stateless RESTful design. We have also through incorporating RESTful design into our patterns of thought, gained noticeable benefits in areas such as how to simplify object interfaces and manage modularisation in a system. As the database design not only was designed initially but had to change greatly throughout the development, we have gained far deeper understanding in relational database design and implementation. Similarly, while we only had academic knowledge of ORM-tools formerly, using Eloquent for database communication have changed this academic knowledge to personal experience. In particular having to circumvent issues and limitations of Eloquent and some SQL-query issues, gave benefits in deeper understanding of relational databases and ORM-tools in general.

The only field of study we have not achieved much further knowledge in is developing for Debian. Developing in high-level languages, the only considerations for the platform has been which supporting software tools to choose, e.g. Rsyslog described in Section 4.5.

8.1.2 Achieved Task Objectives

Where previous courses have always focused on a particular part of the software development life cycle, we have through this project gained experience in project management and most of the software development life cycle. From the project inception and aggregating requirements to system hand-off, we have had to handle on our own any issues occurring, in the process gaining understanding of how such problems occur and how to prepare for and avoid them. Systematic and professional software development can be taught, but it arguably cannot be properly understood without the first-person experience of such a project.

As a thesis project we not only had to develop the system, but also learn to and handle professional and, to a certain degree, scientific tasks such as proper documentation of process and product, self-reflection upon ones personal work, and seeking criticism with the purpose of self-improvement and improving the project. In the case of heat generation and degradation, we experimented and documented our results in-depth, see Section 5.5.1. In addition, working with live data potentially containing sensitive personal- and business information demanded rigorous scientific ethics in regards to our handling and proper removal of such data, and notifying the employer for them to improve their reporting routine, see Section 6.4.

8.1.3 Achieved Product Objectives

With the project description in Section 1.2 in mind, we have completed all the main features, arguably in excess. Receiving, parsing and storing reports in a suitable data model, being the main back-end feature, was in large part completed in late March and has functioned with live data ever since. The main front-end feature were the heat map and the graph pieces, and as its completed design and functionality is thoroughly described in Section 5.1.

The initial expectations and requirements of the employer were relatively low as they originally intended a far smaller and less thoroughly designed product. For this reason they added suggestions on extendable features. As the employer later noted in the first demo meeting, the product we had created far exceeded their expectations; the main features were far more well designed and thought through than they anticipated.

Despite so thorough a focus on the main features, we achieved to implement an additional feature; the monthly report feature, which also concluded with an increased scope and was integrated into the front-end. The syslog-functionality too was merely an afterthought, and assumed to be so small that it was not set up as a main feature, though added as a product objective, while in reality it received a good amount of consideration and development time.

We are unfortunately not able to definitely state that the purposes described in Section 1.2.1 are fulfilled before the system is fully deployed and integrated into Innit's work process. However the fact that it has worked very well in testing since Innit started sending live data in early March is a notable achievement in its own, and bodes well for full deployment.

8.2 Team and Process Discussion

8.2.1 Division of Workload

The intended roles and areas of work are listed in Section 1.6. The workload on the front-end proved to be higher than expected causing Lars to work full-time on it from the start. Aleksander started out with infrastructure, tools and some development on both back-end and front-end, but later switched to a majority of front-end development. Olafur did some ad hoc work on the front-end later in the spring, but generally kept to the back-end throughout the entire project. He also had to struggle with some issues regarding Eloquent and the database management, as some features which we required were unsatisfactorily or not at all implemented, see Section 6.2. Vegard started out on the back-end, but moved over to working almost full-time on the report, with some back-end and database-related work, and some refactoring on the front-end.

Despite the assigned work, all team members got some hands-on experience with the database design, back-end and front-end, as we always worked together on campus. By working together, we often had group-wide discussions and ad hoc meetings regarding general design and implementation issues we encountered, keeping all members up to date on progress and knowledge about the tools and frameworks.

The team members' skill sets very well complemented each other; we barely ever had any situations where somebody did not have anything to do or had to wait for others to complete their tasks before work could be continued. Due to the ability to fluidly delegate work upon necessity, and that all team members were generally aware what others were working on, the workload was split very evenly amongst the four team members. In addition we prioritised working on the report from the project inception, causing some reduction in available development resources, but left us with far more time to improve and perfect the product when nearing delivery. The workload was very difficult to ascertain before beginning development, though in total we found that the roles were well assigned and stayed reasonably fixed.

8.2.2 Time Schedule Assessment

Here we will compare the initial time schedule we created during the project plan phase with a resulting time schedule containing the time actually spent and the actual dates for demos, both displayed as Gantt-diagrams; Figure 38 is the original estimate, and Figure 39 is the resulting diagram of actual time spent and dates of completion.

During the pre-project phase there were initially some differences, namely the testing and deciding on tools took a few days less than expected. Apart from that, the length of time and dates are as we originally planned; the research and learning new frameworks being an intermittent task we did when we had available time.

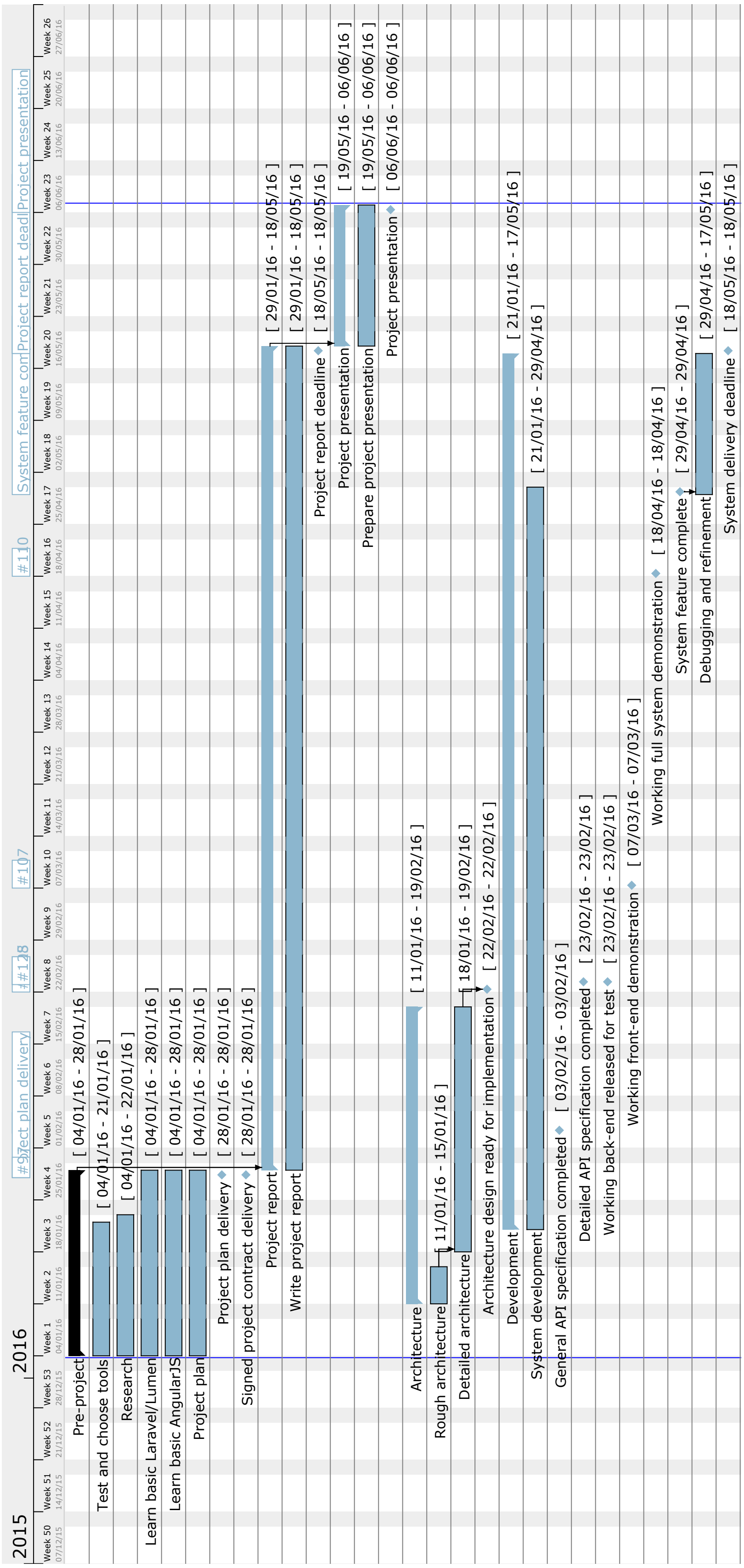


Figure 38: Original Gantt diagram from the planning stage

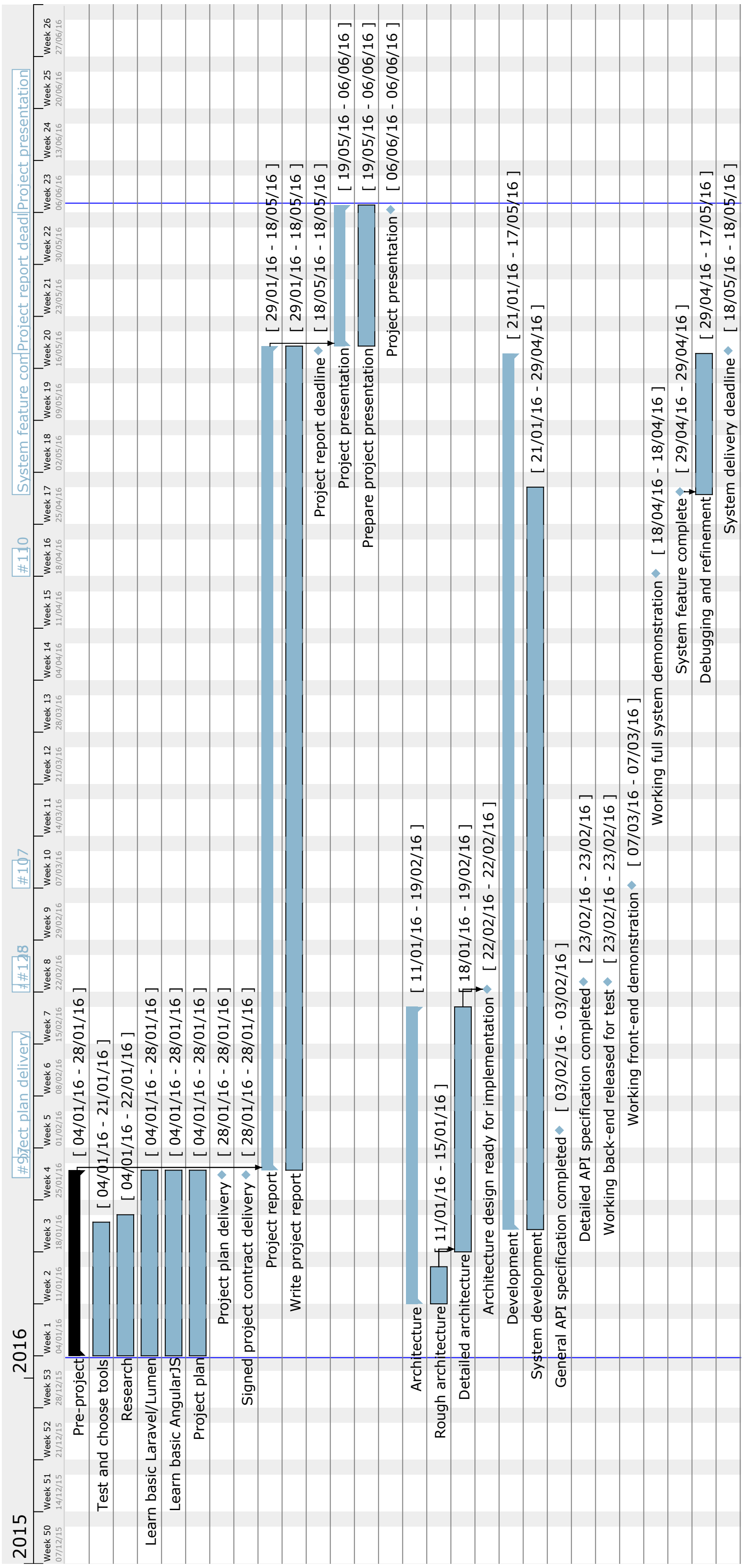


Figure 39: Resulting Gantt diagram of the actual time schedule

The most striking differences are in the architecture and development phases. The general architecture was completed on time, at the same time as the project plan delivery. In practice the detailed architecture was a more continuous work in progress that was not deemed mostly complete until mid-February, though some minor but noticeable changes did occur, e.g. in the amount of views on the front-end, and the division of routes on the back-end.

While the official start of development was not until we delivered the project plan, the early testing of the frameworks and potential architectural decisions led to code that was re-used and built upon, thus in practice starting the development phase a week and a half early. Granted, it was not entirely unexpected as we had been thinking about and considering the project for several weeks before New Years, already having several ideas brooding upon starting planning in early January.

The general API completion was rather unexpectedly on schedule; we deemed it completed two days before estimated. However all the deadlines after that the time estimates ended up significantly off. A detailed API format for posting exception reports was not finished until the day when we also released the mostly completed back-end for testing, a task that itself was done over two weeks earlier than expected. The API for retrieving data however underwent several modest additions and configuration as the project continued.

We did a front-end demo a little over two weeks before we initially anticipated. It is worth noting that both the front-end and back-end both had far more finished functionality than anticipated at that stage. This is despite the front-end proving to be far more work than expected, not only because it stopped being a simple display for the information requested from the back-end, but also because the employer changed their mind upon a few occasions, requiring significant rework, see Section 6.2.

The last demonstration was nicely on time, with the demonstration on April 16. and the deadline being at April 20. Innit never really cared much about the deadlines, mostly focused on allowing us the freedom of a good development process and end result. From this full-system demonstration onwards only a few small features were implemented, before calling it feature complete.

Notable pieces of development that cost us the most time has been lack of knowledge of- and experience with Javascript/AngularJS and PHP/Laravel, and noteworthy issues relating to the frameworks that are described in Section 4.6 and Section 4.1. Since we anticipated and prepared for this, we argue that we have completed more work with better quality in a shorter amount of time than planned for, generally recognised as a successful project.

8.3 Critique and Alternative Approaches

8.3.1 Process

There is a saying that goes ‘nothing is as permanent as a temporary solution’. There is a major reason for this, because most commonly the limitations of a project is not in amount of work or quality, but rather in time and resources. While we have had somewhat more time and resources available than often is during a bachelor thesis, we certainly have not been able to avoid the issues that plague a project.

In this section we will be listing up points of self-critique, unfortunate or outright bad decisions, and in general parts of the project we could or should have done better.

Documenting and Duplication

Documenting decisions was at times left off for too long, at times forgetting it before we suddenly needed it, or Vegard allocated time for himself to do it. Since Vegard often did documentation for others, it sometimes meant the documentation could become too abstract – documenting the intention rather than the implementation – and potentially crucial details could be, and likely have been, overlooked or forgotten. There was also an issue when for example new features were implemented, but never documented - resulting in the other team members not being aware of functionality that was necessary or would have helped greatly. This was partially of remedied by always working together on campus, everybody being available for assistance. Since this was an repeated issue it would have been better to enforce a strict documentation policy from the beginning of this thesis.

In addition to being late at documenting, we tended to duplicate some documentation between the index page for the back-end, information documents on our ShareLatex-stack, and Confluence. While we intended to keep everything in Confluence, only having the interface duplicated to the index page for the back-end, we only properly gathered the documentation in one place by mid-February. Confluence was rarely used since only two of the groups members had the professional programming course which required it, and the fact that the index page of the API was faster and easier to access.

Versioning

Throughout development there were some less than ideal situations where versions of the front-end and back-end did not line up very well, breaking front-end code depending on the API. While we did have some versioning set up for the back-end API, see Section 5.4, it was mostly intended for use in production, though it would certainly have worked during development. Regardless of how it would have been implemented, with versioning some major – albeit mostly insignificant – nuisances could have been prevented, avoiding some front-end situations where updates to API usage had to be completed in the same commit as another issue, breaking our commit rules of only adding code belonging to one issue per commit.

For instance, the back-end API version – or API level – 0.7.1 would work with front-end 0.7.1. In a case where the back-end updated to 0.8.0, changing some part of the API would still allow front-end to use API version 0.7.1. Simultaneously another branch of the front-end could be updated to use API version 0.8.0.

Had we started over again, we would have been strict on versioning from the start, with corresponding versions on the back-end and front-end to handle changes in the API versions. As a secondary benefit it would make the static code analysis statistics over time for more useful, since we could far more easily see the impact various versions made, rather than track down dates for particular commits.

Front-end for Innit

We should have set up and made available earlier a front-end for Innit to play around with and test the features. This would not have been much work, as we did set one up eventually, at little time expenditure. This would have let Innit test the front-end earlier and better than only at the demo meetings. However it is not guaranteed that it would have improved the feedback loop as all our contacts at Innit became very busy as the spring progressed. It never really affected us much, as we never had to redo something due to working in the wrong direction, though we find it somewhat unlikely that Innit would have had more time for testing and feedback than they allocated for meetings.

8.3.2 Product

Use of Laravel

As was partly elaborated on in Section 4.1, we were not entirely appreciative of Laravel due to some quirks and lack of functionality in the framework. A potential risk in using Laravel is that there is only a single person, Taylor Otwell, doing the absolute majority of development, to the scale of 95-98% of both number of commits and lines of code committed [17]. This makes the entire project quite dependant on the single person being active and managing the framework development well. Note that there is a larger large and active community, indicating it might very well take over development of Laravel should Mr. Otwell ever stop developing it. Since the employer warmly recommended Laravel, we considered it a fully acceptable risk.

However as mentioned in Section 4.1, we had some issues with memory limitations, together with some other annoying, but otherwise inconsequential issues. In retrospect we could have remedied some of the memory issues and assured high performance by using e.g. C++ for some parts or even the whole back-end. While less used in web development, C++ can provide considerably better performance than PHP for simple serving of API. There are major libraries available for such usage, but we have not done sufficient research on the subject to state for a fact whether C++ certainly would have been a better alternative for us. C++ also demands more development work and has less built-in security than the ‘magic’ provided by Laravel. One major noteworthy point is that quirks about Laravel and its built-in ORM have perhaps cost us more development time than regular back-end development has, and it is thus not certain that Laravel has cost less time than a back-end in full C++ would have.

Use of Eloquent

Eloquent, being a sub-project under the Laravel framework project, suffers from the same dependency upon the same single person doing the majority of development [18]. While there is, as with Laravel, a reasonably large and active community that likely would be ready to take over the project if necessary, the dependency upon a single person would have made a decision to use another ORM tool more likely.

We had several issues with Eloquent during the development process, most of which are enumerated in Section 6.2, with the main issues being lack of support for three-way relations and lack of support for composite/partial keys, the latter for which the main developer said that he would not add support [16]. Considering the issues we have had with Eloquent, and that so commonly used features as composite/partial keys would not be implemented, we would in retrospect very likely have chosen another ORM tool for the development. In particular if we had chosen something else than Laravel to develop the back-end, another ORM would have been an obvious choice, although we have not done sufficient research on the area to say which ORM tools would have been the most likely.

Composite Keys Issue

The issues we had with composite keys in Eloquent, thoroughly examined in Sections 6.2, and mentioned in Section 8.3.2, cost us a significant amount of time and effort. In retrospect, for the composite keys issue we should likely have used existing code to extend Eloquent, and even after having cut down the length of the file path column and set up the code to use that, it might have been better for maintainability and avoiding potential issues. It would have been preferable not to deliver a system with a data model restricting path lengths to 255 characters, considering that path lengths often reach numbers in the mid hundreds as it is. While about a hundred characters is a fair margin, additional margin is always preferable.

It is also not unlikely that somebody else have completed extending Eloquent in the way we required, though we did not find any references to such completed code, only bits and pieces of code that could partly alleviate the issue. Extending a framework would have been a good learning experience, something we know from some minor changes we made elsewhere in the Laravel framework. Regardless, it has been a significant learning experience having to repair and work around issues that arise.

Unified Format

As was noted in Section 6.3.1, despite the good intentions from the start, the API response format from the main routes ended up varying quite a bit, and it was only around feature completion we managed to come up with a fully unified response format. However that was too late to change all the routes on the back-end and the data handling on the front-end to use the new format, and thus it was only the /graph route that used it, functioning as a reasonably good test for the format.

We should have made such a unified format earlier, or at least attempted to unify it. It would have been very difficult to come up with a properly working format in the beginning, considering how much the data model changed over time, see Sections 4.6 and 6.2. However considering how finished the data model was in late March, we should at that time have attempted the rework, although it is not unlikely that such an attempt even then would have been premature or simply would not have worked out.

It is worth noting that everything the team knows and has learnt about creating APIs and using RESTful communication has been during this project period, with a few snippets of information picked up at random in other settings. Because we knew so little, and the API had to be specified very early to allow development, we gave ourselves too little time to learn, with the risk of creating a bad structure. As this section notes this did to some degree happen, with the lack of a single unified format, although the formats in use are not bad. We will argue that the resulting implementation is reasonably well adjusted to the typical use, and the lack of this unified format is not a major issue for maintainability and further work.

Data Abstraction Layer

We found only late in the project that Laravel is built to support an abstracted interface between the logic of the system and the data models. This could be used to provide an unchanging interface for retrieving data that would work regardless of how the data is stored. Currently the back-end's logic uses the data models of Eloquent as a semi-abstraction to retrieve data from the database, but the logic still depends on the overall database design, and can break upon changing the data model.

For instance one could call for 'all exception types from a given application' in both the current implementation, and with a data abstraction layer. The current version would likely have issues if the relation between exceptions and applications changed; e.g. if we added a new table as a relation between installations and applications. The data abstraction layer however, if implemented well, would let the logic remain unaffected even if we re-implemented the data model with a new design.

Thus the current implementation allows the logic to be affected by changes to the data model. This is less than ideal, though we assumed using the models in this semi-abstracted way was the preferred implementation until we in early May found that Laravel supported a fully abstracted data layer. Had we known this from the design stages, a not insignificant fraction of data model changes and issues could have had their impact greatly reduced.

8.4 Further Work and Extendable Features

This section will lists potential further work and extendable features of the system. Each feature will be briefly described, with some ideas for implementation and assessment of the idea's potential and likely workload. Several of the mentioned features are unlikely to even be considered due to inherent complexity, difficulty and amount of work required. This in particular covers the features that would require machine learning.

Unify format for all routes. As noted in Section 8.3.2 one could rework all the back-end routes to use only the unified format as in the /graph main route. While the workload would be far from insurmountable, the development and testing would be significant.

Integrate with JIRA. An integration with JIRA could filter out the most important exception reports and automatically or semi-automatically post issues on JIRA for the developers. This was initially an idea for this thesis project as additional work, see Section 1.2. It was later scrapped by Innit, see Section 8.1.3. It is most certainly a possibility for the future, though it would likely cost significant time and resources to design and implement what would likely have to be a machine learning algorithm, that would post only the most important issues.

Mobile Application. Since the front-end is designed as a web-view, it should not prove too difficult to complete the remaining adaptations for it to be usable on a mobile phone. However, as mentioned in Section 8.1.3, we are quite uncertain how well the very graphing system which makes the front-end good for large screens, would work on a small screen. It could also be used as a notification system if certain heat-thresholds were met, very similar e.g. Facebook's push-notifications, a feature which presumably would not require very much work to implement. It could also be implemented as a dedicated application, in which case adapting the web-view would not be necessary.

Notification system. Some simple notification system could be added without a mobile application, e.g. sending messages to a mobile phone or using e-mail. The latter would be very simple, while the former would likely not be much harder, by using some third-party system for sending phone messages that handles payment.

User Authentication. Authentication could be implemented on both the front-end and back-end so it could be available outside the local network. While adding authentication late in development is generally highly insecure, we have implemented the system with a fair bit of consideration for potentially later adding authentication. In particular the back-end, with Laravel handling most authentication, would require very few changes.

Widget-based user interface. The front-end could allow drag & drop of elements to rearrange them to a personal preference. This type of interface could allow adding new features simply by importing new widgets, or similarly, resulting in a fully personalised and unique interface design.

Translation of the front-end. Currently the front-end is only available in English. We have added angular-translate to the project, and set up all text to be retrieved from string-files. Thus the only requirement would be to translate all the strings to the new language.

Creating own syslog parser. Currently Rsyslog has some drawbacks, in particular regarding the database tables as mentioned in Section 4.6. While a new syslog-parser would alleviate these issues, it would also be a rather large project on its own. Using Rsyslog for sending messages, and only using a new or customised parser for receiving and adding to the database would be a significantly simpler project, it would likely still require a great deal of work.

Implement machine learning. Machine learning could be used to automatically find which exceptions are bad and which are acceptable, after an initial learning period. Currently the user has to set the severity for for all exception types manually, which is not ideal. Such an machine learning project would be rather difficult however, and likely well outside the scope of what could be achieved in a bachelor thesis.

Desktop Application. The front-end could be implemented as a desktop application instead of a web-view relying on a browser. This would also depend on whether a mobile application would use the web-view or a dedicated application. It is difficult to estimate how much work this would require, though it would likely involve slightly more work than building the front-end in the first place required.

Data Abstraction Layer. One could change retrieving data on the back-end from having the logic directly interact with the data model, to interact with a data layer in between the logic and the data model, so that any changes to the database model would be invisible to the logic; the data interface would be unchanged. See Section 8.3.2. This would be a fair bit of work, but changing one main route at a time would allow the work to be split over a longer period of time.

Integration with development tools. Some integration with development tools could allow links in the front-end to lead directly to the relevant task, issue or commit, much like how the Atlassian stack has issue links between Bitbucket, Confluence and JIRA.

9 Conclusion

We have reached this point after several months of developing a system for Innit in a professional setting. The process has been invaluable for us, both in terms of learning new frameworks and tools, but perhaps the most in experiencing and managing a full software development project from start to finish in a close to real world situation. We have greatly benefited from working full days for months on end on a single project, rather than splitting time between a multitude of smaller projects. We have developed the product we aimed for, a product we can be proud of.

We have in this project managed to work together as a professional team for a long period of time, see Section 8.2. We have set and completed a list of results, effects and learning goals that we wished to accomplish by the end of this thesis, and we will argue that we have completed them all in a more than satisfactory fashion, see achieved learning objectives in Section 8.1.1, and achieved task objectives in Section 8.1.2.

We have gained valuable insight and experience with modern agile development methodologies, software development professionalism, and perhaps most importantly, working together as a team in a project that is important for the customer.

In this project we have developed an application that is highly extensible, maintainable, and which matches both our personal and Innit's requirements, see product objectives in Section 8.1.3. In short time the product will be officially delivered to our employer, and Innit will from that point on-wards hopefully be noticeably more efficiently in discovering issues in and improving their software.

Bibliography

1 Innit AS. Historie (online). 2015. URL: http://innit.no/om-selskapet/ (Visited 2016.01.05).

2 Gerhards, R. Request for Comments: 5424. The Syslog protocol (online). 2009. URL: https://tools.ietf.org/html/rfc5424 (Visited 2016.02.23).

3 NTNU Gjøvik. Bachelor's Thesis (online). 2016. URL: http://english.hig.no/course_catalogue/student_handbook/2015_2016/courses/avdeling_for_informatikk_og_medieteknikk/imt3912_bachelor_s_thesis (Visited 2016.05.03).

4 Kniberg, H. & Skarin, M. 2010. *Kanban and Scrum making the most of both*. InfoQ, C4Media Inc.

5 Wikipedia Community. Five Ws (online). 2016. URL: https://en.wikipedia.org/w/index.php?title=Five_Ws&oldid=708846467 (Visited 2016.03.09).

6 Otwell, T. & et. al. A PHP-Framework for Web Artisans (online). 2016. URL: https://github.com/laravel/laravel (Visited 2016.4.11).

7 Otwell, T. Eloquent: Getting Started (online). 2016. URL: https://laravel.com/docs/5.2/eloquent (Visited 2016.4.13).

8 AngularUI Team. UI Bootstrap (online). 2016. URL: https://angular-ui.github.io/bootstrap/ (Visited 2016.05.03).

9 Otwell, T. & et. al. Laravel Docs (online). 2016. URL: https://github.com/laravel/docs (Visited 2016.01.25).

10 Papa, J. & et. al. Angular Style Guide (online). 2016. URL: https://github.com/johnpapa/angular-styleguide (Visited 2016.01.25).

11 Google. Google HTML/CSS Style Guide (online). 2016. URL: https://google.github.io/styleguide/htmlcssguide.xml (Visited 2016.01.25).

12 Sahni, V. Best Practices for Designing a Pragmatic RESTful API (online). 2016. URL: http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api (Visited 2016.01.25).

13 Kovarik, J. License (online). 2016. URL: https://glyphicons.com/license/ (Visited 2016.05.03).

14 Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. CHAPTER 5 Representational State Transfer (REST) (online). 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (Visited 2016.03.02).

15 Touffe-Blin, J. Angular Chart (online). 2016. URL: https://jtblin.github.io/angular-chart.js/ (Visited 2016.05.09).

16 Aebersold, S. & Otwell, T. Eloquent should support composite keys (online). 2016. URL: https://github.com/laravel/framework/issues/5355 (Visited 2016.4.12).

17 Otwell, T. & et. al. Contributors to Laravel (online). 2016. URL: https://github.com/laravel/laravel/graphs/contributors (Visited 2016.4.13).

18 Otwell, T. & et. al. Contributors to Eloquent (online). 2016. URL: https://github.com/illuminate/database/graphs/contributors (Visited 2016.4.13).

A Project Plan

Project Plan
Group H3E - Handler for Exceptionally Exceptional Exceptions
Bachelor's thesis IMT3912

Vegard Solheim, Olafur Johan Trollebø,
Lars Walter Westby, Aleksander Steen

Spring 2016

Contents

1 Introduction

1.1 Background

Software does not run flawlessly. Problems happen, and often these problems are caught in the form of exceptions, handling the problem and keeping the software running while the problem itself is reported. Exceptions can be tiny inconsequential information notifications or connection failures - events that are expected to happen - to potentially devastating bugs and information state failures. Thus exceptions are both expected and a large part of modern software development, and reports from exceptions can tell a lot about how a software module works, or doesn't work.

The company Innit AS is a medium-size software development and hosting company[1]. As Innit develops and hosts software for customers, they are responsible for keeping software running as intended. When exceptions happen, Innit receives reports from the software about what went wrong, what caused it, if it worked out, and parts of the software or modules were affected. This information can and is gathered, stored, and analysed in order to maintain and improve the affected software.

Innit have requested the development of a system for automated receiving and managing exception reports, and displaying a statistical overview of the exceptions.

1.2 Learning Objectives

- Learn the Javascript- and PHP-based frameworks AngularJS and Laravel, respectively.
- Use of professional development tools like taskrunners, code analysis and project management tools.
- Integrate with existing tools for graphical representation of statistics, i.e. diagrams.
- Develop a completely modular system, where modules can easily be replaced.
- Use a Kanban-based continuous development process.
- Work with a real customer having high expectations, creating a product that is likely to actually be used.
- Handle exceptions and exception reports in a professional manner.

1.3 Performance Objectives

With this project, we wish to develop a system that:

- Increases the effectiveness of receiving and managing exception reports.

- Can display exception statistics in a variety of ways, increasing understanding of potential problems.
- Makes it easier for Innit to identify potential problems in their software.
- Allow Innit operations staff to browse and filter Syslog log files in a more efficient manner than before.

1.4 Target Audience

The target audience here is mainly the developers and operations staff at Innit AS. The system will be running locally on a server at Innit, and the information displayed statically on screens in the development and operations offices. Thus we do not have to take into account that non-technical audience will be viewing or using the software. It is also important to keep in mind this is not a system that will be worked very actively with, but viewed at a distance every once in a while.

2 Project scope

2.1 Field of Study

- RESTful API-design
- Debian-based application development
- Relational database design
- Web application front-end and back-end frameworks
- Web application development

2.2 Scope Limitations

- The system will only be running on Debian, no development has to be done for Windows or Macintosh. If we have sufficient time, we might create a mobile application. See the project description 2.3.
- The creation and display of statistic graphics will be done through integration of an existing third-party tool/package.
- Since the application will only be running locally, authentication is not necessary. Both the reporting interface and the statistics API will only be available over local network.
- We will only have to supply a format for data from external systems. Innit will take care of rewriting the reporting in existing systems.

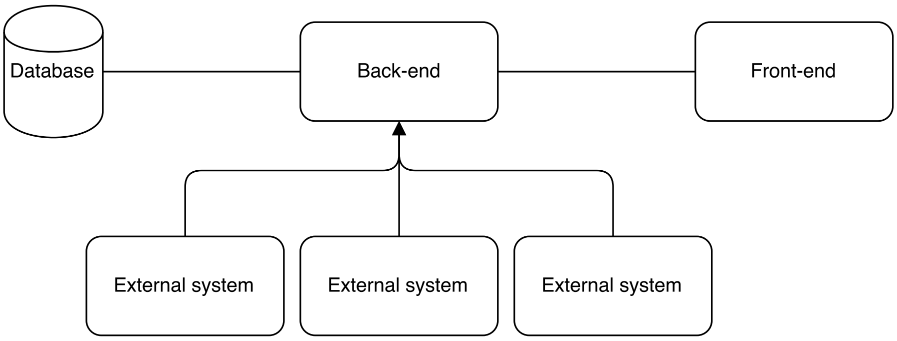


Figure 1: Overview of the system

2.3 Project Description

The system we are to develop has the working title of H3E, short for Handler for Exceptionally Exceptional Exceptions. The systems main functionality is to receive and organise exception reports, and display statistics about the reports.

The systems functionality is to:

- Receive, store and organise exception reports in a preset format, i.e. Syslog.
- Display exception statistics in a graphical format.
- Automatically propose new issues for adding to the JIRA issue tracking system.
- Extra features:
  - A hybrid mobile application for Android and iOS, for receiving exception report notifications.
  - End-of-the-month-report, giving a summary of how the past month has been.

To elaborate on the bullet points mentioned above. The system's back-end will receive exception reports from hosted systems running on Innit's servers. The system will then parse and organise the reports in a local database, and potentially automatically propose new issues that can be added to the issue tracking system. The back-end will supply an API for statistics about the exception reports and meta-information about them.

The systems front-end will be a separate system that will request information from the back-end through the back-end's API, and through integration with a third-party tool will create graphical diagrams and display them on a screen.

The system will be running on a local Debian server at Innit, and only requires communication between the front-end and the back-end, both being on the local network.

Regarding the added points about extra features. They are not within the explicit scope of the project, but the customer have expressed interest in having those features if we have the time leftover to develop them. However they should not be seen as mandatory features to be done before system delivery. The hybrid mobile application will be an application displaying a web view of the statistics site, and able to subscribe to categories like applications, installations, types of bugs, etc., and receive notifications about those subscribed items. The end-of-the-month-report is a report that, as the name implies, will be automatically created and sent out via. e-mail. It will contain a summary of the past month with several categories of statistics, and perhaps small trivia about the system and developers; e.g. who fixed the most reports.

3 Project Organisation

3.1 Roles and Responsibility

Project Leader - Vegard Solheim
Responsible for meeting notes, writing and formatting the thesis, updating time schedules. Working on back-end development.

Design/Technical staff - Aleksander Steen
Responsible for tools working correctly, i.e. ShareLatex, code analysis tools, commit hooks, etc and the design and development of the front-end.

Back-end - Olafur Johan Trollebøe
Responsible of the back-end development.

Front-end/backend/technical staff - Lars Walter Westby
Responsible for automated taskrunner tasks, front-end architecture and general Angular/Laravel technical support.

Supervisor - Ivar Farup
As our supervisor he is the point of contact for any issues or information we require regarding the project as a whole, and will as an external third-party give feedback and discuss our ideas, suggestions and the format and content of the thesis.

Customer - Innit v/ Joakim Jøreng
The customer defines the system requirements, and provides technical considerations, expertise, and feedback on the development process.

3.2 Project Rules

- If there are any disagreement, a vote will be called to solve the matter. In case of a tie, the supervisor will decide.

- Vegard Solheim will be our group leader, and will ensure the group is functioning and keeping the project on track. All issues will be voted upon in a democratic manner.
- If a member does not contribute to the project, a warning will be given. If no improvement in a reasonable time, a vote will be called to decide further action.
- All members of the group are to be considered full members, and can sign on behalf of the entire group if two or more members are present.

For the signed document, see appendix A.

3.3 Meetings and routines

For meetings we have decided to go with planning more meetings than are likely to be necessary, since it is easier to cancel a meeting when it is not needed, rather than scheduling new meetings whenever necessary.

In particular for our supervisor and Innit it is easier to schedule far into the future. Our group is far more adaptable, as we only have one subject in addition to the thesis - everyone having the same subject - and can thus adapt and plan depending on when our supervisor and Innit are available.

We have thus planned to have a meeting with our supervisor at 09.15 every Tuesday. Meetings with the customer is scheduled be on Wednesdays every other week.

3.4 Resource responsibilities

Innit will be supplying the project group with any required or relevant hardware and tools that are not available for free through NTNU or free student licenses. In addition the customer will be responsible for creating formats, but where the project group desires to do so, we have a large freedom to create our own formats where preferable.

Innit is also responsible for always having a dedicated contact person, irrespective of individuals being sick or otherwise indisposed.

The project group will be responsible for planning meetings and supplying the customer with sufficient information for having productive meetings, informing the customer about progress and creating and displaying demos. The project group will also be responsible for handling issues related to the university and our time schedules.

4 Planning and development process

4.1 Development process

The outline of the system is quite clear, and does not warrant any particular development process. However the customer has expressed explicitly that they wish to give us close to full freedom to test different approaches. In addition there are additional modules they wish to have developed, in case workload is insufficient for a bachelor thesis. This, while not specifically demanding it, does make agile development preferable.

Internally in the group we have some experience with continuous development, i.e. Kanban, and have found it to work quite well for us. The group composition, internal dynamics, knowledge and motivation for work have proved to make e.g. a Sprint-based framework unnecessary for forcing progress and results. In addition we agreed that we do not have sufficient experience with this type of system to estimate sufficiently well for Scrum sprints, thus rendering sprints somewhat impractical.

For these reasons we have unanimously decided we wish to go forth with Kanban as described by Henrik Kniberg and Mattias Skarin for our development process[2]. It will incorporate some features from Scrum in an adapted form, most notably a weekly or bi-weekly retrospective meeting that will be held in conjunction with meetings with the supervisor.

We have conferred with our supervisor whom does not have any objections to our choice. Neither does the customer, which has no particular preference and expresses a wish for us to have close to full freedom of how we wish to arrange development. They merely require that we have demos ready at some points during development, but leave us the freedom to choose when, where, and how much should be ready.

As mentioned in subsection 3.3 we will incorporate weekly meetings with our supervisor, and bi-weekly meetings with the customer.

4.2 Framework usage

We have decided to use the frameworks AngularJS and Laravel for the front-end and back-end, respectively. This was strongly recommended by the customer as they have quite a lot of experience with both frameworks, and would thus be far more able to provide support for our development. After learning the basics and trying them out, we agreed with the customer that it is a good recommendation.

For AngularJS it is also a choice whether to use Angular v1 or v2. The customer did not have any particular opinion or recommendation on that matter, and left it to us to decide. We have decided to go with Angular v1, as v2 only recently became available and is still in beta. The customer supports this decision. There are far more learning and helpful resources available for version 1, which is important as none of us have much experience using AngularJS. We also discussed the use of ReactJS which also is a good alternative, but we opted for not using it.

As for graphical presentation and elements, we decided to use Bootstrap due to some prior knowledge and experience, and Less, a CSS pre-processor, both because of having seen the use of such a framework in previous projects, as well as recommendations through best practice guides and frequent use in Yeoman packages and similar project skeleton generators. What AngularJS module we will end up using for graphs is still undecided, but several have been tested and had their pros and cons discussed. Having a mix of frameworks we have never used before and some we have prior experience with creates a nice environment we feel we can grasp, while it also lets us explore new technology and approaches.

Thanks to AngularJS' focus on testability, continuously writing unit tests for the front-end application is made easy through the use of Jasmine and Karma, which work well with AngularJS and has a lot of official documentation available. Automatic tasks also make actually executing the tests easy, once our task-runner is configured. Automated linting in JShint and JSCS for front-end code is also set up to warn for code inconsistencies and errors through out taskrunner, Gulp. The back-end will primarily use PHPUnit for unit tests (which Laravel is primarily configured for) while using Sonar coupled with PHPmetrics for code review.

4.3 Version control

We have decided to use Git for version control, and use BitBucket for hosting the repositories. The system uses two repositories to separate the front-end and back-end systems. Neither of the repositories will be publicly available during the development process. Whether they are open after having been handed over to Innit will be up to them.

The decision to go with Git is due to it having been recommended greatly during our education, has become more or less an industry standard, and we have a fair lot of experience using it. Both the customer and supervisor supports our choice.

We have decided to go with BitBucket as the host due to the integration with Confluence and JIRA, both of which we will use in a subject parallel to the thesis.

5 Development tools

JetBrains PHPstorm

PHPStorm has been strongly recommended by friends and colleagues - and all larger frameworks for those two languages. The team has a lot of experience with the related IDE IntelliJ, which uses the same IDE kernel as PHPStorm for Java development. In addition, JetBrains supplies all students with free access to their entire suite of IDEs. In light of that, we have decided to use PHPStorm for both front-end and back-end development. We have used Eclipse and Notepad++ for web development in the past, but with PHPstorm's native Laravel support and AngularJS plugins, we unanimously decided to skip the other two candidates.

ShareLatex
ShareLatex is a online real-time collaboration tool for writing LaTeX. We have good experience with using LaTeXto write reports, and writing the report in it was a given from the beginning. We tested both OverLeaf and ShareLatex beforehand, and decided to go with ShareLatex on the background of recommendations, as well as its compilation being far faster than OverLeaf's. A ShareLatex premium licence is available through NTNU. However we had already set up the freely available ShareLatex stack on a group members servers and decided to use that. It does not provide the kind of assured redundant storage that hosting the report on the ShareLatex site would provide, though we consider it to be sufficiently safe storage for our purpose. For some added safety all group members will be downloading the source files whenever changes have been done. As a bonus, the private server provides us with far more computational power for faster compilation than the hosting sites provides.

We did briefly consider using Google Docs and Office365, though we quickly decided to write in LaTeX, since the latter is superior in terms of writing long academic reports and handling multi-file reports and bibliographies. While Google Docs is free, and Office365 is available for free through NTNU, that did not put them anywhere close to the available tools and options of LaTeX.

Sonar/PHPmetrics With experience from a previous Java-project where we used Sonar in combination with Maven for automatic static code analysis, we have decided to keep using Sonar for the static analysis. Due to Maven being a Java-specific tool, we opted to use PHPMetrics as replacement for it.

Gulp
To simplify and streamline the development process we use a taskrunner tool to run automated tasks such as serving the code, wiring dependencies, linting, testing and building. The go-to standard taskrunner has been Grunt for many years, which works heavily based on configuration files. A newer taskrunner that has gained a lot of heat is Gulp, which works in more of a code-like fashion for its configuration setup, as opposed to Grunt. After looking at both tools and weighing them against each other, we decided to go for Gulp, for a number of reasons. The biggest reason would be the way the configuration files are written, which worked in favour of our group's preferences. The fact that gulp seems to be considered the new and more modern tool also helped us choose. Many preconfigured taskrunner files exist for both tools, so the availability of pre-written tasks did not matter as much to us when choosing.

Trello
Trello is a well known and much used work planning tool which all members of the group have some experience using. We are of the opinion it has the best user interface and usability, after having tested a wide variety of Scrum or Scrum-related work planning tools in a project the preceding autumn.

JIRA

Due to several of the group members having the course Professional Programming, where we will be working with and reviewing the bachelor project development process and code, we have decided to create a JIRA-space in addition to the Trello-board. We have been and will be using Trello until the JIRA-space is set up and ready to use. When that happens, we will make a choice whether to continue with Trello, completely switch over to JIRA, or use Trello for the work tracking and JIRA for issue tracking.

Confluence

As a part of Professional Programming course we will be using Confluence for Wiki-esque purposes like documentation and document management. This will be in addition to, but not overlapping with, the ShareLatex stack where we write the project report.

MariaDB/PHPMyadmin

We quickly decided to go with a relational database, both due to we having far more experience with relational models, and because we only have to store simple organised textual data in a set format. We did consider NoSQL databases, but we found no good arguments in favour of NoSQL for the kind of report data we are to store.

Having chosen relational database, we had some discussion whether to choose MySQL or MariaDB, but eventually landed on the latter due to external recommendations. MariaDB is said to be faster/better optimised and generally better in use, while the APIs and interfaces are essentially equal to MySQL.

The customer expressed no particular preference regarding choice of database system, but essentially assumed we would use MySQL or MariaDB.

5.1 Documentation and standards

All the code should naturally follow the standards described below. All code should be documented as it is written, with PHPDoc or equivalent written as every function is written, and inline comments during or immediately after writing the code. One can optionally run Sonar on the code before merging a branch with master. Upon merging or pushing code to master, our server will automatically pull down master and run it through Sonar and PHPMetrics. All standards, protocols and similar should be documented on Confluence.

Unit-tests should either be written before or alongside writing the functionality. Any non-trivial component that can reasonably be tested must have unit-tests. Naturally functionality that can't easily be isolated for unit testing does not require tests, but one should strive to test as much as possible.

The front-end application is continuously linted for code inconsistencies, following configuration files we have set up and can easily change. This ensures each member's individual coding style is less obvious, and the project follows a standard as a whole.

For standards we have decided to simply follow the best practice for each of the frameworks and languages, as listed below.

- Laravel - https://github.com/laravel/docs[3]
- AngularJS - https://google.github.io/styleguide/angularjs-google-style.html[4]
- HTML/CSS Google conventions - https://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml[5]
- API Best practices - http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api[6]

5.2 Risk assessment

In the table 1 we have listed all risks to the system and development process that we have been able to identify. All risk having a risk equal to or higher than medium (i.e. very low * very high, low * high and medium * medium) have been listed with precautions reducing likelihood and/or impact in table 2

Table 1: Table of potential risks

No.	Issue	Probability	Impact
1	Unable to finish the project	Medium	High
2	A team member is long-term sick	Medium	Medium
3	ShareLatex is down or loss of ShareLatex source	Low	High
4	BitBucket is down	Very low	Low
5	Total source code corruption	Very low	Very high
6	Contracting entity no longer wants the project done	Low	Medium
7	Group disputes	Low	Medium
8	Inaccurate time estimation	High	Low
9	Hardware malfunction resulting in loss of code, e.g loss of laptop	Low	Low
10	Customer contact person is rendered indisposed for an extended period of time.	Low	High
11	There are privacy issues with the system	Medium	High

5.3 Choice of language

The customer have specified that the front-end should be made ready for an indefinite number of language. English and Norwegian are required languages

Table 2: Table of solving aforementioned risks.

No.	Preventative Action	Incident Resolving
1	Make use of agile methods and methodology to keep project on tracks	Hand in whatever has been completed
2	Make sure at least two group members know each part of the code	If a member is ill over a extended period of time, the project leader will decide further action and will be responsible for keeping the ill member updated on the project
3	Regular backups. Download source after editing.	Switch over to NTNU-license ShareLatex instead of self-hosted version
5	Regularly commit to an online source version control repository	Restore project from older working version
8	Accept some mis-estimates	Re-evaluate backlog. Learn from previous estimates and take on less work at a time
10	Innit will be responsible for always having a contact person available	A new contact person from Innit will be appointed
11	Innit has stated they'll take responsibility for avoiding sending any privacy-related information to the system	No action. Innit has stated they'll take care of it.

and will be support from the start. The customer does not have any requirements for back-end language, and thus it will be made to use English by default without any built-in internationalisation.

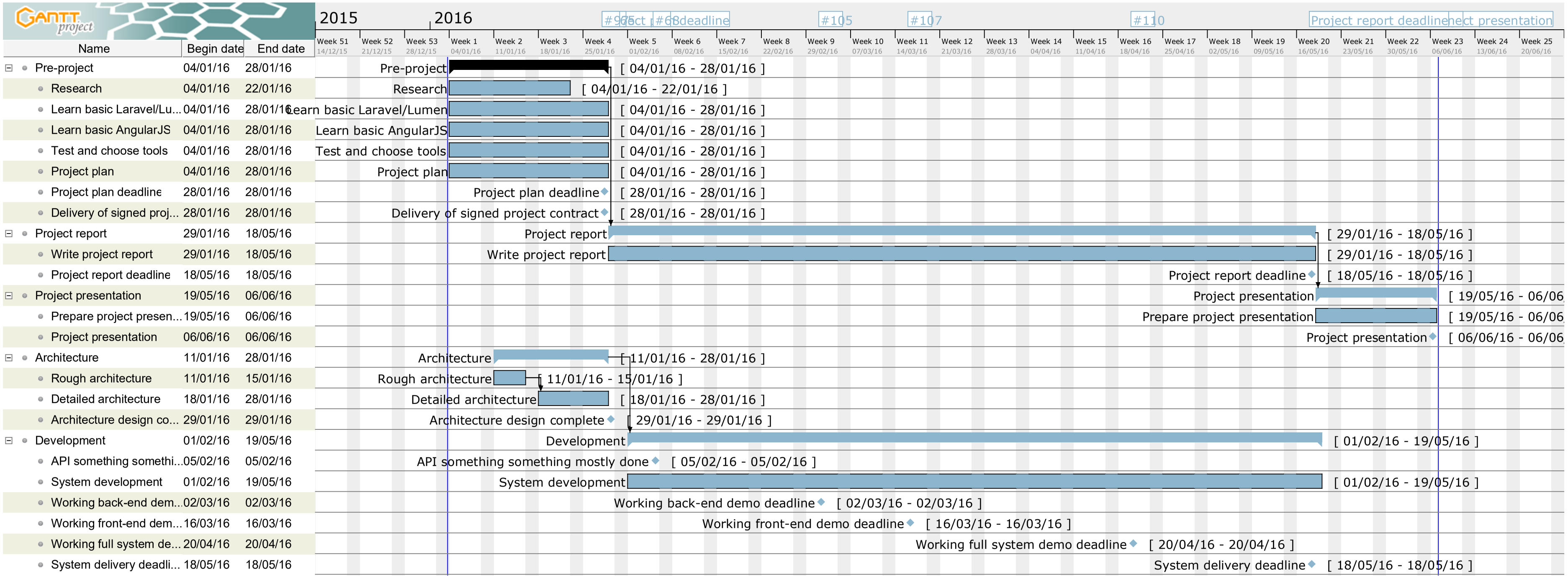
5.4 Legal

We have not identified any potential legal issues with the system, with the caveat that Innit handles all potential privacy issues as promised. All code developed will be released with an MIT-license.

6 Resource planning

6.1 Gantt schema

Gantt-schema created in the open-source freeware GanttProject[7].



Name	Begin date	End date
⊟ Pre-project	04/01/16	28/01/16
Research	04/01/16	22/01/16
Learn basic Laravel/Lu...	04/01/16	28/01/16
Learn basic AngularJS	04/01/16	28/01/16
Test and choose tools	04/01/16	28/01/16
Project plan	04/01/16	28/01/16
Project plan deadline	28/01/16	28/01/16
Delivery of signed proj...	28/01/16	28/01/16
⊟ Project report	29/01/16	18/05/16
Write project report	29/01/16	18/05/16
Project report deadline	18/05/16	18/05/16
⊟ Project presentation	19/05/16	06/06/16
Prepare project presen...	19/05/16	06/06/16
Project presentation	06/06/16	06/06/16
⊟ Architecture	11/01/16	28/01/16
Rough architecture	11/01/16	15/01/16
Detailed architecture	18/01/16	28/01/16
Architecture design co...	29/01/16	29/01/16
⊟ Development	01/02/16	19/05/16
API something somethi...	05/02/16	05/02/16
System development	01/02/16	19/05/16
Working back-end dem...	02/03/16	02/03/16
Working front-end dem...	16/03/16	16/03/16
Working full system de...	20/04/16	20/04/16
System delivery deadli...	18/05/16	18/05/16

GANTT project
2015
2016
#96 #75ct #68 #68deadline
#105
#107
#110
Project report deadline
Project presentation
Week 51 14/12/15
Week 52 21/12/15
Week 53 28/12/15
Week 1 04/01/16
Week 2 11/01/16
Week 3 18/01/16
Week 4 25/01/16
Week 5 01/02/16
Week 6 08/02/16
Week 7 15/02/16
Week 8 22/02/16
Week 9 29/02/16
Week 10 07/03/16
Week 11 14/03/16
Week 12 21/03/16
Week 13 28/03/16
Week 14 04/04/16
Week 15 11/04/16
Week 16 18/04/16
Week 17 25/04/16
Week 18 02/05/16
Week 19 09/05/16
Week 20 16/05/16
Week 21 23/05/16
Week 22 30/05/16
Week 23 06/06/16
Week 24 13/06/16
Week 25 20/06/16
Pre-project [04/01/16 - 28/01/16]
Research [04/01/16 - 22/01/16]
Learn basic Laravel/Lumen [04/01/16 - 28/01/16]
Learn basic AngularJS [04/01/16 - 28/01/16]
Test and choose tools [04/01/16 - 28/01/16]
Project plan [04/01/16 - 28/01/16]
Project plan deadline [28/01/16 - 28/01/16]
Delivery of signed project contract [28/01/16 - 28/01/16]
Project report [29/01/16 - 18/05/16]
Write project report [29/01/16 - 18/05/16]
Project report deadline [18/05/16 - 18/05/16]
Project presentation [19/05/16 - 06/06
Prepare project presentation [19/05/16 - 06/06
Project presentation [06/06/16 - 06/06
Architecture [11/01/16 - 28/01/16]
Rough architecture [11/01/16 - 15/01/16]
Detailed architecture [18/01/16 - 28/01/16]
Architecture design complete [29/01/16 - 29/01/16]
Development [01/02/16 - 19/05/16]
API something something mostly done [05/02/16 - 05/02/16]
System development [01/02/16 - 19/05/16]
Working back-end demo deadline [02/03/16 - 02/03/16]
Working front-end demo deadline [16/03/16 - 16/03/16]
Working full system demo deadline [20/04/16 - 20/04/16]
System delivery deadline [18/05/16 - 18/05/16]

6.2 Work Breakdown Structure

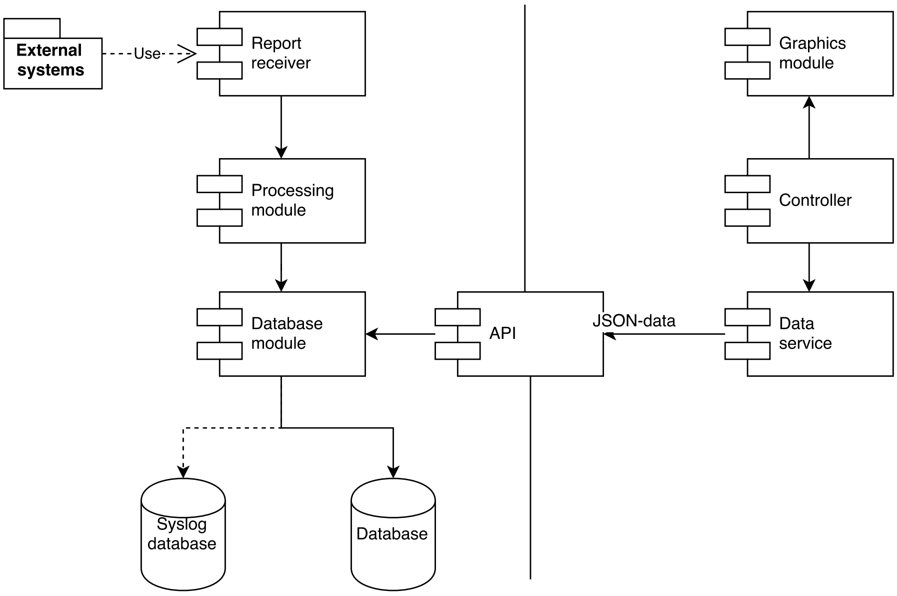


Figure 2: Module diagram of the system

Figure 2 shows the separated modules of the system. The line indicates the separation between the back-end on the left, and the front-end on the right. The front-end, by using AngularJS, is implicitly built as a Model-View-Controller, where the three modules represent each part.

- The external systems as the systems the customer is hosting for customers, which will be sending the exception reports.
- The report receiver will be providing the API and receiving and handling the reports.
- The processing module will be handling general system management and configuration, though it might end up being removed.
- The database module is, as the name implies, a module for handling all connections to the database, so that neither the receiving or processing module, nor the API-module needs to have their own database connections.
- Syslog database is an external database, managed by the operations department, and will e read-only for this system.
- The database is a MariaDB-instance storing all the exception reports.

- The API-module will be providing the API for requesting exception data and statistics for the front-end, and potentially for the mobile applications
- The data service is the front-ends model, and handles requesting data from the back-ends API. As with the database connection this should be handled by a single module for easier modification of the API and usage of it.
- The controller is the controller module for handling timers and the data flow to the view, and responding any input from the view.
- The graphics module is the view module, handling displaying everything, and being the module receiving external input from users.

6.3 Time- and resource plan

This section features a breakdown of the workload for each person on the team and the priorities for each person. Note that all numbers or percentages given are approximations and not to be taken as more than guidelines.

For all the group members it is expected that about 10% of available working time will be spent on group meetings and meetings with the supervisor and customer. In addition it is expected that another 10% will be spent on updating the project report and source code documentation, with the exception of the group leader will which spend 20% or more on the report and documentation.

During pre-project phase

The group leader will spend 50% or more time on the project plan and related documentation. The remaining time will be on learning Laravel and AngularJS, and assisting in creating architecture and API-specification.

The rest of the group will spend approximately equal time (30/30/30) on testing tools, learning Laravel/AngularJS, and architecture and API-specification.

During development phase

Vegard and Olafur will have their main focus on back-end development and the reporting API. In addition Vegard will spend a days worth of work per week on writing the report.

Lars and Aleksander will have their main focus on front-end development and graphical design.

During pre-delivery phase Pre-delivery phase starts a month before the project report delivery, when the system under development should be scope complete, and remaining development work is code optimisation and fixing bugs. The group leader will be spending about 50% of work on the project report, with 20% - or about a days worth of work - of assistance from each of the other group members. The remaining time will for all group members be spent on said code optimisation and bug fixing.

References

1 Innit AS. Historie [website]. Innit AS; 2015 [updated 2015; cited 2016.01.05]. Available from: http://innit.no/om-selskapet/.

2 Kniberg H, Skarin M. Kanban and Scrum making the most of both. InfoQ, C4Media Inc.; 2010.

3 Otwell T, et al. Laravel Docs [website]. GitHub.com; 2016 [updated 2016; cited 2016.01.25]. Available from: https://github.com/laravel/docs.

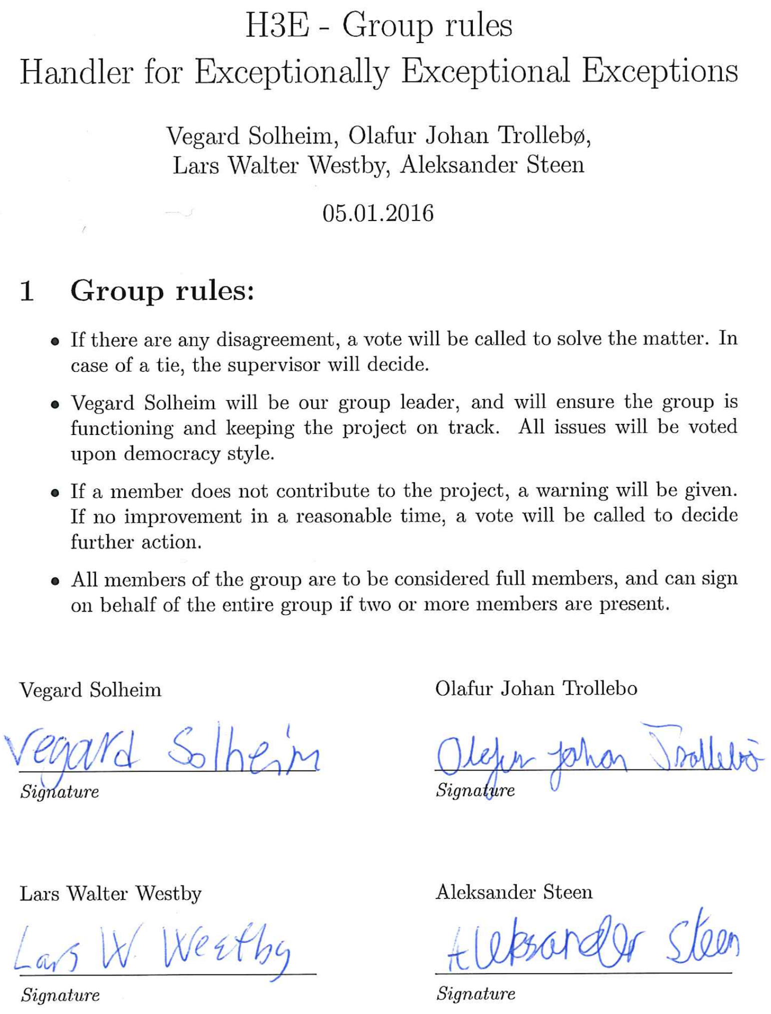
4 Papa J, et al. Angular Style Guide [website]. GitHub.com; 2016 [updated 2016; cited 2016.01.25]. Available from: https://github.com/johnpapa/angular-styleguide.

5 Google. Google HTML/CSS Style Guide [website]. Github.io; 2016 [updated 2016; cited 2016.01.25]. Available from: https://google.github.io/styleguide/htmlcssguide.xml.

6 Sahni V. Best Practices for Designing a Pragmatisk RESTful API [website]. Vinay Sahni; 2016 [updated 2016; cited 2016.01.25]. Available from: http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api.

7 Barashev D, et al. About GanttProject [website]. GanttProject Team; 2016 [updated 2016; cited 2016.01.18]. Available from: http://www.ganttproject.biz/about.

A Group rules document



H3E - Group rules
Handler for Exceptionally Exceptional Exceptions

Vegard Solheim, Olafur Johan Trollebø,
Lars Walter Westby, Aleksander Steen

05.01.2016

1 Group rules:

- If there are any disagreement, a vote will be called to solve the matter. In case of a tie, the supervisor will decide.
- Vegard Solheim will be our group leader, and will ensure the group is functioning and keeping the project on track. All issues will be voted upon democracy style.
- If a member does not contribute to the project, a warning will be given. If no improvement in a reasonable time, a vote will be called to decide further action.
- All members of the group are to be considered full members, and can sign on behalf of the entire group if two or more members are present.

Vegard Solheim

Vegard Solheim

Signature

Olafur Johan Trollebo

Olafur Johan Trollebö

Signature

Lars Walter Westby

Lars W. Westby

Signature

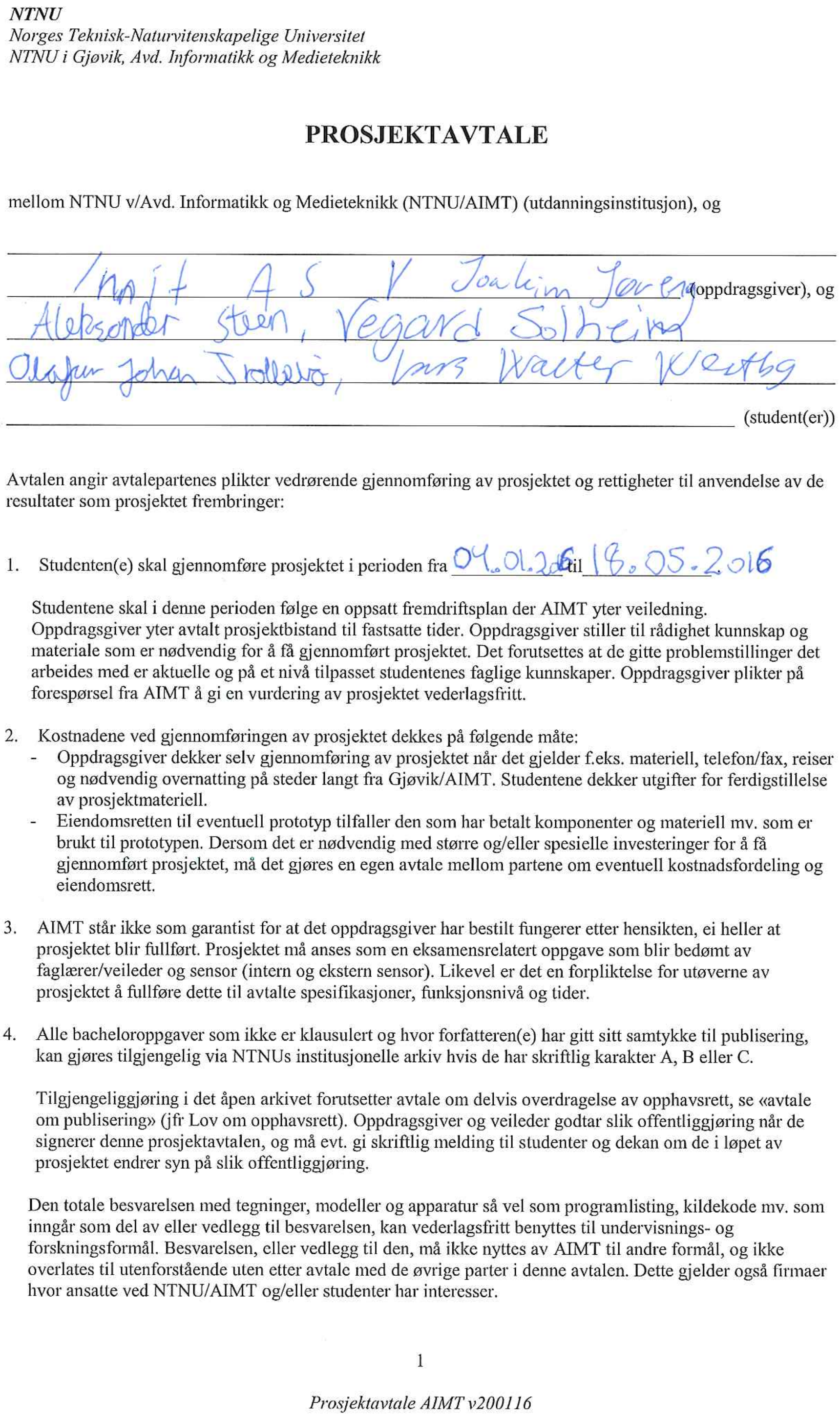
Aleksander Steen

Aleksander Steen

Signature

Figure 3: Signed group rules document

B Project Contract



NTNU
Norges Teknisk-Naturvitenskapelige Universitet
NTNU i Gjøvik, Avd. Informatikk og Medieteknikk

PROSJEKTAVTALE

mellom NTNU v/Avd. Informatikk og Medieteknikk (NTNU/AIMT) (utdanningsinstitusjon), og

Innit AS v Joakim Jøren (oppdragsgiver), og

Aleksander Steen, Vegard Solheim
Olafur Johan Trollevö, Lars Walter Westby
(student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 04.01.2016 til 18.05.2016

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der AIMT yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra AIMT å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon/fax, reiser og nødvendig overnatting på steder langt fra Gjøvik/AIMT. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. AIMT står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor (intern og ekstern sensor). Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv hvis de har skriftlig karakter A, B eller C.

   Tilgjengeliggjøring i det åpen arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av AIMT til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU/AIMT og/eller studenter har interesser.

1

Prosjektavtale AIMT v200116

6. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

7. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

8. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i Fronter. I tillegg leveres et eksemplar til oppdragsgiver.

9. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av AIMT er det dekan/prodekan som godkjenner avtalen.

10. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og AIMT som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten AIMT som partner.

11. Når NTNU/AIMT også opptrer som oppdragsgiver, trer NTNU/AIMT inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

12. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

13. Deltakende personer ved prosjektgjennomføringen:

NTNU/AIMTs veileder (navn): Ivar Farup

Oppdragsgivers kontaktperson (navn): Joakim Jøreng

Student(er) (signatur): Aleksander Steen dato 28.01.2016
Vegard Solheim dato 28.01.2016
Olafur Johan Trolldør dato 28.01.2016
Lars Walter Westby dato 28.01.2016

Oppdragsgiver (signatur): Joakim dato 27/1/16

Signert avtale leveres digitalt i Fronter(IMT3912)
Godkjennes digitalt av AIMTs dekan

Om papirversjon med signatur er ønskelig, må papirversjon leveres til AIMT i tillegg.
Plass for evt sign:
AIMT Dekan/prodekan (signatur): ____________________ dato __________

C Meeting Summaries

C.1 Desember 2015 - Pre-Meeting with Innit

C.1.1 Things to read/learn

- Laravel
- Check out Homestead.
- Angular JS
- Taskrunners, e.g. Grunt or Gulp
- Syslog
- Yeoman.

C.1.2 What system should do

- Useing established standards is preferred. Check out Syslog
- Frontend in Angular JS:
- Must show choosable and filterable overview of exceptions
- Be able to filter on e.g. files, classes, modules, customers, exception types.
- Completely modular. Modulers should be easily replaceable
- Will get back to how much to make of the graphical.
- Backend in Laravel
- Receive, organise and store exception reports.
- Pre-specified format for the exception reports.
- Potentially create new issues in issue tracking system.
- Separate exceptions from development and from operations.
- To be run on Debian 8.
- Format: Currently manual handling of reports received in an IRC-channel.

C.1.3 Questions and answers

- Use as direct API or parsing of text from existing reporting channel? **API, and likely POST-calls with JSON-data or something like that. If there are good reasons they can change the functionality for reporting in the existing systems.**
- Requirements to format of data? **Not certain yet, check out Syslog. Also check if there are other good standards that could work.**
- Special requirements for presentation of data? **Look for other applications or packages for presentation of data and generation of diagrams**

C.2 016.01.13 - First Meeting with Innit

Meeting Summary	**Information meeting**
Duration:	3 hours
Date:	2016.01.13

Present:
Joakim Jøreng, Marius Haugen (both Innit)
Vegard Solheim, Aleksander Steen, Lars Walter Westby, Olafur Trollebøe
Next meeting: 2016.01.27

Announcements
No particular announcements were voiced.

Agenda for the meeting

- Discuss questions we sent to Innit earlier.
- Discuss framework choices and alternatives.
- Discuss extensions to the project.

Discussion
Most of the questions sent beforehand were satisfyingly answered, and we got answers to a whole lot of other questions as well during the meeting. Generally it was an informational meeting, discussing the questions sent, ideas about architecture and system/module separations. In particular some in-depth information about how best to modularize the front-end system based on the experiences those at Innit have had.

Innit left us full freedom as to which development process we wish to follow, and supports our choice of Kanban.

Retrospective discussion
Innit has the impression we are working well, and have no particular points to raise regarding the progress we are making.

For next weeks

- Further familiarize ourselves with Laravel, Angular, and test the more optimized API-specific micro-framework Lumen.
- Create more detailed outlines of system architectures and APIs/interfaces.
- Continue the current progress we are making on the project plan.

Meeting Questions Summary

- Authentication, login for the site or the API, or connection- and authentication-less? State-less authentication? (Token being given after login, used until it expires) **Idea is that all systems that is to report are behind the local firewall. Isn't priority to build authentication for the API. All systems that will report are hosted at Innit.**
- Resolution, what res will the screen run at? Do we need to make a responsive design for many devices? Presumably only large screens (1920x1080). **Assume only large screens. Don't use large frameworks, try to keep it small and without too many heavy dependencies. Small Bootstrap and equivalent is likely enough. Can check out Foundation. Will be viewed on several screens. Don't assume any mobile sizes, most if not all screens will be 40-60 inches and full HD.**
- What kind of data does the solution make? Some dummy data would be nice to have so we can design our database around it. We decided to use MariaDB seeing as we are looking at a format that fits relation databases well. Maybe at least some screenshots of the IRC chatlog thing. **We'll get back to this. Can be taken over mail/video meeting/other. There is currently one standard for the reports, but they're very adaptable as to new standards for it. All systems will always use the same standard, thus if we have a proposal then all systems will receive that quickly. We will receive a mail on this, and feel free to request updates on it relatively often.**
- We will make our solution compatible with Syslog, probably make two different views to separate sysadmin from developer "feeds" on the GUI to separate the content (sysadmin probably has no interest in developer feeds). Do you use standard syslog format, and what solution do you use presently to store/receive all the Syslog data? Would it be feasible to change your setup, so we could get Syslog/current exception handling to post to our back-end. Otherwise we will create middleware that receives Syslog messages on a specific port, then send it to our solution as a POST. This probably the better option if you don't want to change your current setup. Aleksander can setup a Syslog server, so we don't need dummy data for that if you use a standard Syslog setup. **We'll get back to this. See answer to question 3, about dummy data. Can be taken over mail/video meeting/other**. No current Syslog server at the moment, Aleksander will test out different solutions. Probably end up talking to a syslog solution API instead of direct DB connection. Innit Drift said they would give me a syslog switch if we needed, probably would be cool. Linux server.
- Any preference on data type, we discussed using post and examining the data when received what type it is and act accordingly. Potentially just have the exception format be the same, so that any exception adapts the Syslog format. **See answer to question 3, about dummy data.**
- Customisation, would you want to be able to change the format of incoming messages on the back-end easily, and/or be able to accept different types of exceptions with simple configuration? Do you want us to make the back end very adaptable, meaning it accepts a lot of different input types but processes it correctly? **See answers to question 3, about dummy data. All systems will use the same standardised format, and if a change occurs then it will propagate to all systems.**

- Do you want us to make some sort of algorithm that sees if there are a lot of one type of exception from one particular client/software and make an visual alert to the developer that made it, or a general alert dashboard? **There should be some differentiated handling of different types of exceptions. E.g. PDO-/database connection exceptions can be critical after the first, while NotFoundExceptions are typical and isn't a problem unless it happens way too much. Will have to come back on the specifics, but keep it in mind. We'll likely get a overview system/report with generated exception reports, though it doesn't reflect the actual usage patterns. Self note: Test if a degradation-graph, where each exception happening gives more "points", and then the "points" degrades**
- We need information on the issue tracker, if we are going to automatically create new issues when exception reports come in. **Not prioritised, only a bonus for Innit. They use JIRA, and it would be nice to have, but only if we have time. Can have it propose issues while each issue have to be accepted manually. That way somebody knows it has been added, and it won't potentially spam the issue tracker with erroneous/useless/unnecessary data. If done, prefer to have it done on the front-end through the back-ends API.**
- Are there different issue types, eg. minor, major, critical? Do you want them handled differently? Possibly the internal Syslog types? **See answer to the algorithm for handling different types of exceptions. This will be further handled later, doesn't have a perfect answer yet.**
- Ownership/copyright of the code and repository? **Innit will own and control repository.**
- Are you going to control the database, alternatively you have an existing Syslog database? **We can just do pretty much what we want, and they'll adapt their software and get it set up on the server.**
- Configuration-file or web-interface for configuration? **Go with web-interface. In particular it is by far preferable in order to handle all things issue-related (see question about issue tracker).**

Import things to note:

- Do injections properly
- Have some proper overview of the modules that are used.
- Be certain about best practices for single page vs. multi-page sites with Angular.
- Could be fun with 'leaderboards' for whom generates the most exceptions.

Things to be able to sort by

- Customer
- Class
- File
- Module
- Type of exceptions
- Meeting every other week.

C.3 2016.01.19 - Second Meeting with Innit

Meeting Summary	**Q&A Skype meeting**
Duration:	1 hour 20 minutes
Date:	2016.01.19

Present:
Joakim Jøreng, Ronny Bohrmann (both Innit)
Vegard Solheim, Aleksander Steen, Lars Walter Westby, Olafur Trollebøe
Next meeting: 2016.01.27

Announcements
Unplanned Q&A meeting. Requested by both parties and agreed upon the day before the meeting.

Agenda for the meeting

- Questions relevant to architecture, API-specification and development.

Discussion
See appended Q&A-sheet for the questions discussed.

Meeting Questions Summary

- Should one be able to filter by customers? Does a customer have more than one system?
  It is relevant to get information about which user was using the system when the exception was created, instead of actual filtering per user/customer. Can have the actual username that caused the exception as part of the normal reporting data, instead of The "Client"-part in the API specification is which application, not the actual customer or user. No need to take special care about this. Could be added some particular information about customer/user, e.g. ID, username, email, other relevant fields.
- What languages are you using in house?
  Mainly PHP and Javascript. All languages will be using the same format. The actual type, line number + character column, and stacktrace dump can be different depending on languages, but this shouldn't be a problem as it's handled as text.
- Do any of the systems share users, or are there unique users for every system? Say John Doe can also show up in other systems. Would it be beneficial to have a list of users encountered, or just have the information coupled with the exception in case of problems with that specific user account (Credentials, permissions and the likes)
  See answer to question 1. Generally no, not in a way that we specifically have to take care of.
- The field "location" in the log-message. How would an eventual "client" be handled there? Would it first send it to an internal system and then forwarded to us? **An exception happening in our system (during development) would be a client-exception instead of server. Another way to check would be that most Javascript-exceptions (Except for Node) would be client exceptions. Set location to client when ....? Client would help separate between errors in the APIs/back-end systems and errors happening client-side, like in the browser/on clients machine. All messages will be sent through the systems, none directly from the client machine.**
- Do you use any custom exceptions in your applications, or do you only use the standard variants, like PDOException, NotFoundException? **They do have some custom exceptions. Need to have dynamic adding of new types of exceptions added after the system is created. Aka. definitely no manual adding of types in the database.**
- Syslog. What kind of gear? Seems that most manufacturers that use syslog doesn't follow the same log format. We need to know what kind of equipment that we should support. Web servers? Switches? HP or Cisco gear? **We'll be borrowing a switch from operations to get the exact Syslog format used.**
- Custom E-mail warnings if something user defined happens(or doesn't happen?), is this something you want? **Don't know. This can be a potential feature if we have leftover time.**
- Could there be any privacy related issues from the exception reports? **There are possibilities for this, but will be handled by Innit. Not an issue for development for us**
- Specific license for the system? Is important for what code and what graph mod-

ules/tools can be used. **Will come back to this on next meeting**

- Will/should state information of the machine/server be sent as part of the reports, and handled thus? **Not really important as of now. Make opening for adding this later, since it might not be very far into the future.**
- Severity based on type of exception? "Hardcode" manually into database or into source code, or have it in a separate config file? **Should be done in a separate config file on the back-end, but the information will be loaded to the front-end. Depends a little whether statistics will be calculated on back-end or front-end, and how the information is delivered. Manual adding of severity "points" in the config is most preferable. For the heatmap, the heat will always have to be separated on which application/installation in addition to which type of exception. Naturally this will be filterable manually.**
- How are different installations of the same application separated? **The application name will be the same, but the URL will be different depending on which installation is reporting. Different versions will be reported how?**
- Authentication for exception reports API? **We don't have to create authentication, but should be possibilities for adding it later. Possibilities for extending should be handled by the system being fully modular. We can assume that all external system uses SSH or VPN to the local network.**

C.4 2016.01.27 - Third Meeting with Innit

Meeting Summary	**General discussion**
Duration:	1 hours 30 minutes
Date:	2016.01.27

Present:
Joakim Jøreng (Innit)
Vegard Solheim, Aleksander Steen, Lars Walter Westby, Olafur Trollebøe
Next meeting: Not planned.

Announcements
No particular announcements were made.

Agenda for the meeting

- Accept and sign the project contract.
- Display the low-fi prototype for front-end
- General information

Discussion
Discussed the questions appended below.

Meeting Questions Summary

- Specific license for the system? Is important for what code and what graph modules/tools can be used. **We're going with MIT-license.**
- Modular vs fixed design. (Show image). **Will be going with fixed design that can be configured per part in the design. Prefer to have the heatmap as a all-page background colour rather than a single limited coloured box. Be careful with nuances of the colours, while keeping the other data readable. Heatmap should only be for the one most critical application. We'll be talking with Eivind regarding usability and general UX. Have several different types of static sites that can be switched between, while not fully configurable without creating new sites/changing the code per site. Have one overview that's not intended to be clicking anything at all, for the main screen overview. Separate overviews that are intended for actual interaction, for when it's accessed locally individually. Assumption that when something goes wrong, one will have to check it manually for further information.**
- State of syslog
  - syslog(LOG_ERR, "Something bad has happened");
  - Switch
  - We have Apache/PHP logs working
- What kind of features would be good? If we have time, what extra features would you like the most?
  - Authentication **Not very interested.**
  - Mobile hybrid app (iPhone + Android) **As of now, the most preferable alternative.**
  - JIRA integration/issue tracker **Likely too annoying for the developers, not preferable.**
  - End-of-the-month-reporting. Combines all the data at the end of the month, and sends out a summary report to 'insert group of people'. May also contain particular trivia, e.g. who to blame for the most exceptions.

  Most preferable feature: hybrid mobile app. A system where users "subscribe" to applications, and receive push-notifications if one of the subscribed applications goes critical. Second most preferable feature: end-of-the-month-report.
- Only integration with JIRA for issue tracking, or more generic integration with issue trackers?

Import things to note:

- Consider which kinds of meta-information are interesting to display and play around with. E.g. icons for the source of the exception, like server-icon for server side exception, client-icon for client-side, etc.
- Check out Ionic for creating hybrid phone apps.

C.5 2016.03.07 - First Demo Meeting with Innit

Meeting Summary	**Demo and feedback**
Duration:	2 hours
Date:	2016.03.07

Present:
Ronny Bohrmann, Joakim Jøreng, Marius Haugen (Innit)
Vegard Solheim, Aleksander Steen, Lars Walter Westby, Olafur Trollebøe
Next meeting: No date set.

Agenda for the meeting

- Displaying a demo of the system. Front-end connected to the back-end.

Discussion
The severity of most exception types are more difficult to set beforehand, because it can both depend on which application, installation, and the context of the use. E.g. it would be worse to get a 404 or similar with a POST-request compared to a GET-request. Seems most likely to have a severity set per exception type in the config, and have an additional context-based severity rating that is made based on some kind of config-input done later.

Perhaps the reports themselves should send the rating if it's supposed to have one, and if there isn't a rating, then it'll either be set as unrated, or simply a middle-ground. A potential problem is where exceptions aren't caught correctly, and thus will report the cause as "storage file" or something similar, even though the issue is actually somewhere else. Is rather important to have the system automatically identify the same exceptions/error situations happening several times in a limited time. Check later whether this is dependent of time or not, and within how long a time frame.

For next week(s)

- Should be able to filter on the front-end based on if it's development, testing or production environment. Should also be shown which environment an exception is from, default to production only
- Check that filtering on version is possible, and that version is displayed.
- Check that the list of applications doesn't become a list too far down.
- Check parsing of base URL whether an application or installation already exists, or to add a new one.
- Config option whether to send exceptions to the system or not during development.
- Add feature to set an exception for review. I.e. it has to be manually marked as seen. Can have some kind of local installation of the system for receiving the exceptions, so that the developer himself can review the exceptions received.
- Can add that heat-calculation uses a context-severity rating in the report multiplied with the exception type severity rating. The exception type severity rating will be set in config, and the context-severity rating will be set somewhere else, but where is not certain yet.

C.6 2016.04.26 - Second Demo Meeting with Innit

Meeting Summary	**Demonstration**
Duration:	1 hour 30 min.
Date:	2016.04.26

Present:
Joakim Jøreng, Marius Haugen, Ronny Bohrmann
Vegard Solheim, Aleksander Steen, Lars Walter Westby, Olafur Trollebøe
Next meeting: No date set

Announcements

No particular announcements were made.

Agenda for the meeting

- Demonstration of feature complete system.
- Receive final feedback on small things to improve.

Retrospective discussion

- While reading a single exception, bold the start of each line, for easier reading.
- Make a reasonable README.txt for installation and simple use.
- Increase size and readability of the report view. Have the first one automatically be un-minified.
- Check that all views are in proper containers.
- Try out having the heat colour as a large bar on the top or bottom 1/3 of the status view, so it doesn't make the graph unreadable. Alternatively have the colours start from white instead of green, aka. going white to yellow to red.

Meeting Questions Summary

- Should anything be built to be cached? **Up to the team's best judgement.**
- Any further endpoint/data sets than what we have thus far? **Not that Innit can come up with immediately.**
- Should we avoid skewing the results on the front page due to some applications having far more installations, and some installations having far more users than the others? Perhaps use an exponential function to reduce the effect it has. **Not within the scope of the project.**
- How far back should historical heat data be stored? **Just store everything. Not large enough amount of data to become an issue.**
- How far back should historical data be changed when as severity rating be changed? Max a week perhaps? **A week would be nice, but as of now recalculation will not be implemented due to its limited benefit compared to the potentially major pitfalls it can cause. In particular how to avoid concurrency issues if multiple updates to the same severity rating are attempted at the same time.**
- Should the current heat be reset at some time? Perhaps each night, or each end of week? No point watching leftover heat from previous week at least, though perhaps from the weekend is relevant, thus resetting Friday night, so one can see if stuff failed during the weekend. **No need, let the degradation handle avoiding spillover heat affecting new days and weeks.**
- Do you want to be able to change severity ratings for exception types in the front-end, or only do it in the database itself? Changing is potentially risky, and can demand a high amount of recalculation of historic data, depending on answers to the previous questions. **See question 5; recalculation will not be implemented.**
- How often do you want historic heat data to be stored? Are you likely to change how often to store? **Up to the team's best judgement.**
- Should one be able to remove fixed exception occurrences from the graph, and thus remove its contributed heat? This will require a fair bit of extra logic and state being stored, in order to remove only the proper degraded amount of heat from a fixed exception, instead of its initial heat added. E.g. a report that three hours ago added 10 heat to the graph, and just now was fixed would (with 50% degrading per hour) remove 1.25 heat now: $10^{0.5^3} = 12.5\%$ of initial heat to be removed. **No need.**

C.7 2016.05.13 - Final Demo Meeting with Innit

Meeting Summary	**Demonstration**
Duration:	50 minutes
Date:	2016.05.13

Present:
Joakim Jøreng, Ronny Bohrmann (Innit)
Vegard Solheim, Aleksander Steen, Lars Walter Westby, Olafur Trollebøe
Next meeting: No more meetings

Announcements
H3E is complete and the team will be delivering the thesis report later the same day. Innit will take over the repositories by copying the entire repositories and uploading it to their own repositories.

Agenda for the meeting

- Demonstration of the finished H3E-system

Discussion Innit is very satisfied with the solution we have created for them. There were some small quality-of-life improvements that could be made regarding clicking checkboxes, gradients and fonts. These are small configurations we will make for Innit in the coming few days. We will dismantle our systems after the thesis presentation is completed.

Retrospective discussion
The development process has worked out very well, and the team has noticeably taken in all constructive criticism and used it to improve the solution. The communication between the team and Innit could have been better, in part because Innit has been very busy during the spring.

C.8 2016.04.15 - GUI Feedback Meeting

- Exceptions: Take care with the contrast in e.g. the heat map and similarly.
- Exceptions: What are normal numbers for heat. Can show what the maximum values for the colours on the heat map. Preferably the same for the full-screen heat map.
- Exceptions: Could move colours in the list of latest exceptions to just before the name of the exceptions. Can also symbolise colour with amount of "flames" or similarly. Preferably have some way of showing which exception belongs to which hour. Can also remove the year and month from the date, and perhaps even day. Columns sorted as "timestamp" "heat indicator" "exception name". Make certain it is easy to read what belongs to each line. Can have a tiny bit more space between each line. Add which application an exception occurred in.
- Exceptions: Higher contrast on the menu buttons. Colour contrast analyser.
- Applications: Could set grey as undefined. Black is a little too "beyond critical".
- Applications: Space between each character in the name labels for the graphs (letter spacing).
- Applications: "All applications" overview have a tile-based overview, showing which applications have the most heat.
- Applications: On each application the graph can show per installation/customer, not per exception type.
- Applications: The same columns in the exception types list as in the exceptions view. Change list to list of exceptions types with a counter for how many times that type has occurred, and a type does not repeat in the list but rather have an "expand" functionality for showing all.
- Syslog view: fix list in better way, e.g. severity as small colour box.

D API Structure

All arguments to routes or return values are listed in bullet points.

Exception Handler API example usage

All routes are configured to use these return values:

- Request that yield data: Content and status code 200 (OK).
- Request that yield no data: Empty array and status code 200 (OK).
- Invalid item ID: "Resource not found" and status code 404 (Not Found) E.g /exceptions/0).
- Invalid URL (Non-existent route): "Resource not found" and status code 404 (Not Found).
- Invalid query (Failed validation): Array with error messages and status code 422 (Unprocessable Entity).

D.1 Exceptions route

/exceptions/

Lists all exceptions

Arguments specific to /exceptions/:

- **after=:exceptionID** – Gives all exception data received after a given ID, e.g: after=50 gives exceptions from 51 and later.
- **from=:timestamp** – Gives all exception data received after a given timestamp, e.g: from=2016-01-01T13:10:23Z gives everything received after that timestamp. Can be used in conjunction with to=:timestamp for a granular search of dates.
  NB: ‘from’ and ‘after’ are mutually exclusive. Can only use one or the other.
- **installation_ids=:installationIDs** – Shows exceptions that belongs to any of the installation IDs, given by a comma-separated string.
- **exception_type_ids=:exceptionTypeIDs** – Shows exceptions that has any of the exception types IDs, given by a comma-separated string.
- **application_ids=:applicationIDs** – Shows exceptions that belongs to any of the application IDs, given by a comma-separated string.
- **search=:string** – Shows exceptions that contain the string given.

/exceptions/:exceptionID

E.g: /exception/53 List a specific exception by ID

Special notes:

This route sorts by default with ID decrementing, as well as having a default limit of 1000 records returned. This can be overruled by specifying ‘limit’ or ‘orderby’ parameters in the URL referenced in ’Supported global arguments across all routes’

D.2 ExceptionTypes route

/exceptionTypes/
Lists all exceptionTypes

/exceptionTypes/:exceptiontypeID
E.g: /exceptionType/2 List a specific exceptionType type by ID

D.3 ExceptionContexts route

/exceptionContexts/
Lists all exception contexts

/exceptionContexts/:exceptionContextID
E.g: /exceptionContexts/2 List a specific exception context by ID

- **application_id=:applicationID** – Gives all exception contexts data about all installations of a given application.
- **context_ids=:contextIDs** – Shows exception contexts that has any of the IDs, given by a comma-separated string.

D.4 Severity Rating route

/severityRatings/
Lists all exceptionTypes

/severityRatings/:severityRatingID
E.g: /exceptionType/2 List a specific severity rating by ID

severityratings/application/:applicationID
E.g: severityratings/application/2 Lists the severity rating of each exception type for a given application by ID

D.5 Applications route

/applications/
Lists all applications

/applications/:applicationID
E.g: /applications/1 List a specific application by ID

D.6 Installation route

/installations/
Lists all installations

/installations/:installationID
E.g: /installations/1 List a specific installation by ID

- **application_ids=:applicationIDs** – Shows installations that belongs to any of the application IDs, given by a comma-separated string.

D.7 Reports route

/reports/

Index list with links to every report type.

/reports/monthly

Lists all End-Of-Month reports

/reports/monthly/:year-month

E.g: /reports/monthly/2016-03 List a specific End-Of-Month report

/reports/yearly

Lists all End-Of-Year reports

/reports/yearly/:year

E.g: /reports/yearly/2016 List a specific End-Of-Year report

All endpoints without specified dates support these arguments:

- **from=:date** - Gives all exception data received after a given date, E.g: since=2016-01-01 gives everything received after that date, unless an upper limit is given with to=:date
- **to=:date** - Gives all exception data received up to a given date, E.g: to=2016-02-02 gives everything received before that date, unless an lower limit is given with from=:date.

D.8 Statistics route

/statistics/

Lists all statistics

/statistics/total

Lists the total number of statistics by severity

/statistics/application/:applicationID

List the latest statistics for a given applications ID

/statistics/application/:applicationID/total

List the total statistics for a given applications ID

/statistics/installation/:installationID

List the latest statistics for a given installations ID

/statistics/installation/:installationID/total

List the total statistics for a given installation ID

/statistics/exceptiontype/:exceptionTypeID

List the latest statistics for a given exception type ID

All non-total endpoints support these arguments:

- **from=:date** - Gives all exception data received after a given date, e.g: since=2016-01-01 gives everything received after that date, unless an upper limit is given with to=:date
- **to=:date** - Gives all exception data received up to a given date, e.g: to=2016-02-02 gives everything received before that date, unless an lower limit is given with from=:date.
- **modes=:modes** - Allows filtering on modes, given in a comma-separated string. If only production statistics should be shown, modes=production is used. If both production and test should be shown, modes=production,test.

D.9 Graph route

/graphs/
Index list with links to every graph type.

/graphs/application/:applicationID
Show graph data for a specific application ID.

/graphs/exceptiontype/:exceptionTypeID
Show graph data for a specific exception type ID.

/graphs/mixed
Show graph data for multiple application IDs and exception type IDs. Used when you have more than one of both types.

/graphs/total
Show graph data for all application IDs and exception type IDs as a single datapoint.

- **mode=:mode** - Which mode to use, must be either heat or exceptioncount
- **timespan=:timespan** - The timespan to fetch graph data about. If the value is 1, the last 24 hours will be fetched. Everything else will be handled in days.
- **application_ids=:applicationIDs** - Comma-separated list of application IDs to fetch information about. Only has value in either the exceptiontype or mixed route, does nothing in application.
- **exception_type_ids=:exceptionTypeIDs** - Comma-separated list of application IDs to fetch information about. Only has value in either the application or mixed route, does nothing in exceptiontype.

D.10 Syslog route

/syslog/
Lists all syslog data

/syslog/:syslogID
List syslog data by ID

/syslog/hosts
Lists all syslog hosts

/syslog/hosts/:syslogID
List syslog entries by ID

- **ignored=:yes/:no** – retrieves only syslog hosts that is either ignored or not ignored
- **gt=:number** – retrieves only entries with priority higher than the number given
- **lt=:number** – retrieves only entries with priority less than the number given gt and lt must be between 0 - 7, including endpoints. If outside bounds, not given, or otherwise invalid, default values are used that retrieves every single entry.
- **host=:name** – retrieves all entries from a specific host

D.11 Supported global arguments across all routes

All endpoints can handle chunking of data, by using any of these parameters:

limit=:limit – Limits the result-set returned by the server to a given number. If offset is not set, it will return the first results in the database.

offset=:offset – Offset the result set returned by the server from a given number. If limit is not set, it will return everything from the offset provided.

order_by=:fieldname – Order by any field, e.g line, file etc. Default behavior is ascending, but by putting a "-" before the field will order descending, i.e. "-id".

E Refactored Code

This Chapter contains examples of code that has been refactored to satisfy the requirements for the course Professional Programming, in which we are to with our Bachelor Thesis and the system development process in a professional manner. It will contain several links to the relevant commits for the code that is being refactored, from the initial code that needed refactoring to the finished code, and any intermittent refactoring work that we see as relevant.

E.1 Front-end

E.1.1 Grapher.Controller

The first example of refactored code is a function that receives either statistical data or straight up exception data, and converts it into a format required by the graphing package we are using. The example is the formatToGraph function, found on line 468 onwards in the following direct historic link.

https://bitbucket.org/exhandle/h3e-frontend/src/c9210f925464f0d65195ce4df00e5618bc150f32/app/shared/grapher/grapher.controller.js?fileviewer=file-view-default#grapher.controller.js-468

To list the issues with the initial implementation.

- The function is far longer than it should be
- It is not sufficiently documented, the amount and explanatory value of the comments is poor.
- There are lots of non-informative variable names.
- It has a large amount of indentation that worsens readability.
- It has several long array specifiers which would be far more readable if made into temporary local variables.
- There are large chunks of code that could and should be put into separate functions.

Refactored

In the following link, the code has been greatly refactored, and while it has expanded from about 130 lines to 370 lines, much of that is separating functionality out into their own functions, and large amounts of comments and whitespace. The functionality is largely unchanged, but the readability has been greatly improved through fixing much of the issues listed in the previous paragraph. https://bitbucket.org/exhandle/h3e-frontend/src/a0bf2527252185c33a59fcefc2c30863fa4f2a0e/app/shared/grapher/grapher.controller.js?fileviewer=file-view-default#grapher.controller.js-467

Refactored Again, Functionality Moved to Back-end

Also see Section 6.3.1 for the discussion on how we ended up refactoring it away from the front-end altogether. The new code for the graph controller file can be seen here.

https://bitbucket.org/exhandle/h3e-frontend/src/35c4736a4a812a3e5f8e17ab86935348ea960a6f/app/shared/grapher/grapher.controller.js?fileviewer=file-view-default

The code on the back-end that replaces the front-end code for transforming data into the correct format can be seen here.

https://bitbucket.org/exhandle/h3e-backend/src/0c93bc53689a4057cefb2dfad68667fa3334bc3a/app/Http/Controllers/GraphController.php?fileviewer=file-view-default

E.2 Back-end

There has been much refactoring of smaller pieces of code on the back-end. The biggest single refactoring event was in early May, involving major changes in the graphs controller.

E.2.1 Graph Controller

After moving much functionality for the graph data format to the back-end a few days earlier, the refactoring mentioned in Section E.1.1, we sat with a reasonably large single blob of code that did everything. This blob even included functionality to zero-pad the data for days/hours for empty hours/days. https://bitbucket.org/exhandle/h3e-backend/src/0c93bc53689a4057cefb2dfad68667fa3334bc3a/app/Http/Controllers/GraphController.php?fileviewer=file-view-default

The code was not completely unreasonable, though there were a good bit of ad hoc solutions that should be simplified and generalised. Some parts also required more commenting. For instance the functions 'getHeatTimespanHourly', 'getHeatTimespanDaily' and 'getMixedExceptionCountTimespanDaily' have been visibly shortened by better judging of what could and should be done on the front-end or the back-end.

In particular we noticed that zero-padding hours/days that had no exceptions was far better to do on the front-end since it had all the keys required in the form of the very labels that had been selected by the user. It was a typical example of refactoring and moving some code, and ending up moving too much.

This refactoring resulted in a rather good bit cleaner code that did exactly what it is intended to and nothing more; only serving the information requested. Note that there are a large amount of if-else cases, and many functions seem to be doing quite similar things. We considered all of this but found ourselves unable to generalise them away without hurting readability more than benefiting it. https://bitbucket.org/exhandle/h3e-backend/src/401061cf548d/app/Http/Controllers/GraphController.php?fileviewer=file-view-default

E.2.2 Refactoring to adhere to code standard

Another case of back-end refactoring was in the syslog controller. While not inherently bad code, we did some work on it to make sure it adhered to the coding style and standards we had established, for instance:

- Curly brackets should be to the right of a declaration, not on the next line.
- Making use of Eloquent functions for database queries when possible, instead of using raw SQL-queries.
- When handling structured input, like JSON, transform it into an array/collection

for easier validation and handling.

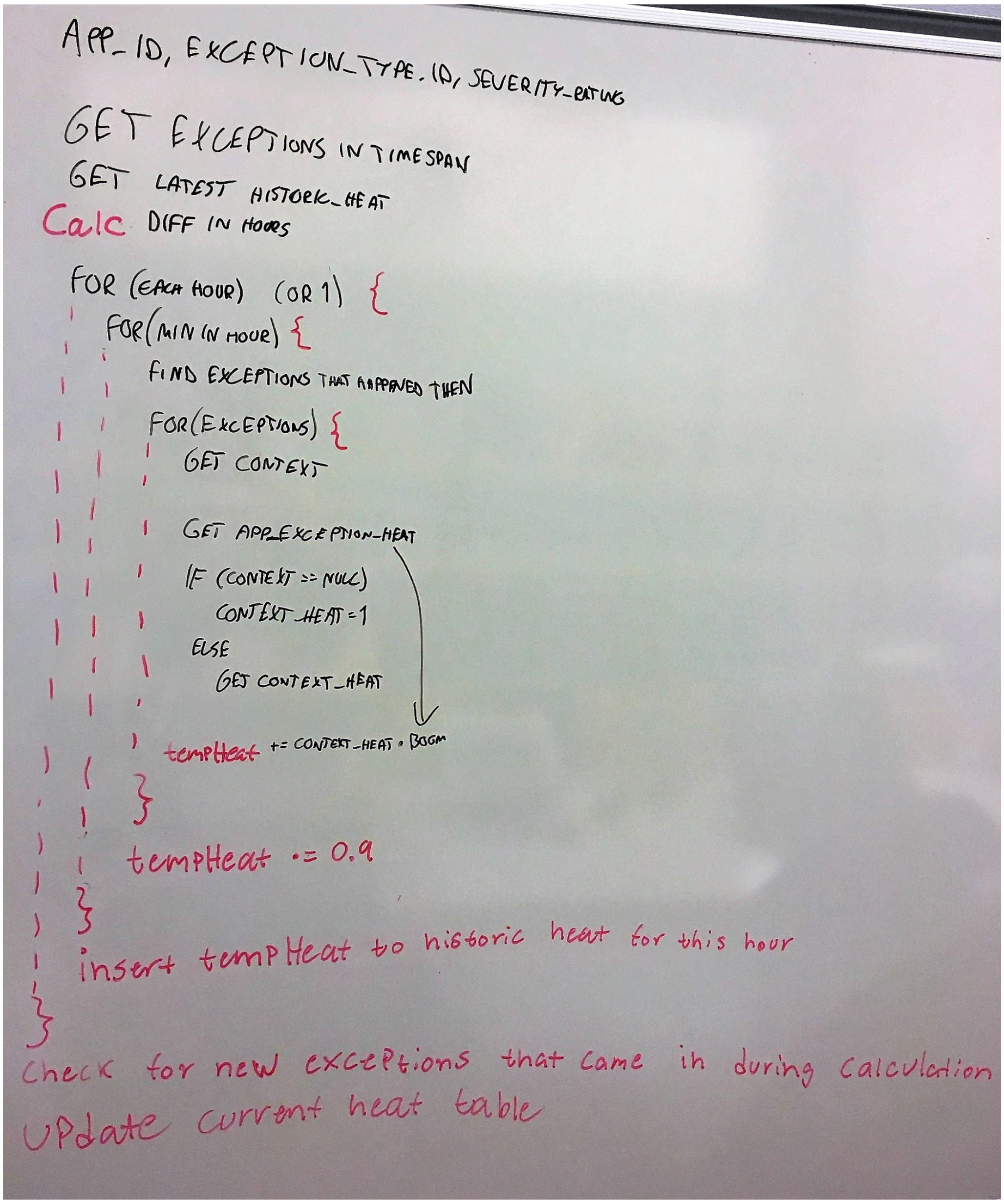
- Using brackets '[]' instead of 'array()' to submit an array into a function. Variables should use 'array()' for the more verbose readability, making the intention completely clear to the reader.

The initial code is available here: https://bitbucket.org/exhandle/h3e-backend/src/f8691cd75179604567c585b3586be66afdd5ea18/app/Http/Controllers/SyslogController.php?fileviewer=file-view-default

The result of this refactoring was a fair bit shorter code that adhered to our coding style and standards. While this should always be ascertained before a pull request is accepted, e.g. through the linting, this file managed to slip through the reviews. https://bitbucket.org/exhandle/h3e-backend/src/25b07bba1c50375fb3998980bf09cccf57ed47da/app/Http/Controllers/SyslogController.php?fileviewer=file-view-default

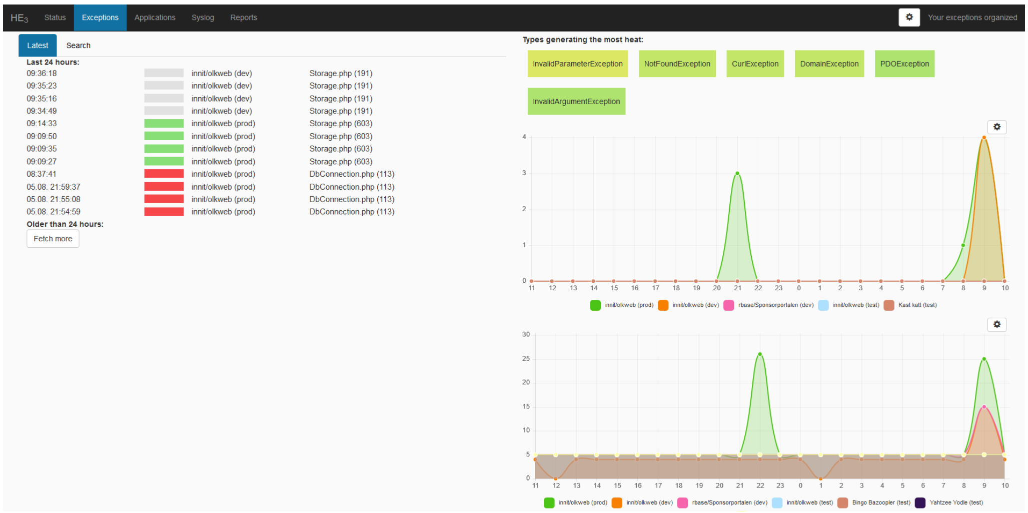
F Images

Here follows sketches made during group discussions throughout the development process, images that did not fit elsewhere in the report but are referred to for completeness.



```
APP_ID, EXCEPTION_TYPE.ID, SEVERITY_RATING

GET EXCEPTIONS IN TIMESPAN
GET LATEST HISTORIC_HEAT
Calc DIFF IN HOURS

FOR (EACH HOUR) (OR 1) {
    FOR(MIN IN HOUR) {
        FIND EXCEPTIONS THAT HAPPENED THEN
        FOR(EXCEPTIONS) {
            GET CONTEXT

            GET APP_EXCEPTION_HEAT
            IF (CONTEXT == NULL)
                CONTEXT_HEAT = 1
            ELSE
                GET CONTEXT_HEAT

        tempHeat += CONTEXT_HEAT · BOOM
        }
    tempHeat ·= 0.9
    }
    insert tempHeat to historic heat for this hour
}
Check for new exceptions that came in during calculation
Update current heat table
```

Figure 1: Concept-code for heat recalculation.



Last 24 hours:

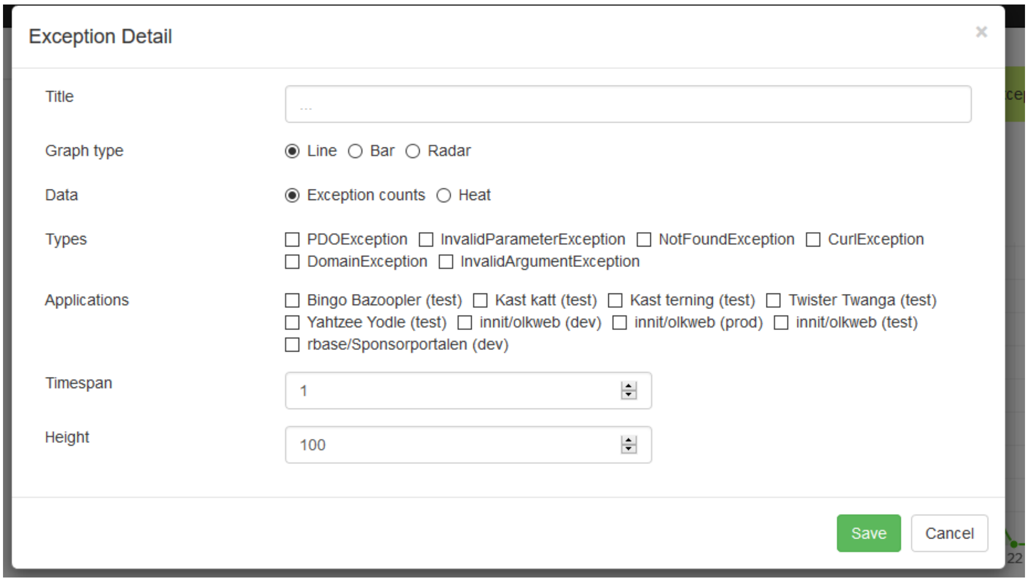
09:36:18	⬜	innit/olkweb (dev)	Storage.php (191)
09:35:23	⬜	innit/olkweb (dev)	Storage.php (191)
09:35:16	⬜	innit/olkweb (dev)	Storage.php (191)
09:34:49	⬜	innit/olkweb (dev)	Storage.php (191)
09:14:33	🟩	innit/olkweb (prod)	Storage.php (603)
09:09:50	🟩	innit/olkweb (prod)	Storage.php (603)
09:09:35	🟩	innit/olkweb (prod)	Storage.php (603)
09:09:27	🟩	innit/olkweb (prod)	Storage.php (603)
08:37:41	🟥	innit/olkweb (prod)	DbConnection.php (113)
05.08. 21:59:37	🟥	innit/olkweb (prod)	DbConnection.php (113)
05.08. 21:55:08	🟥	innit/olkweb (prod)	DbConnection.php (113)
05.08. 21:54:59	🟥	innit/olkweb (prod)	DbConnection.php (113)

Older than 24 hours:

Types generating the most heat:

InvalidParameterException NotFoundException CurlException DomainException PDOException InvalidArgumentException

Figure 2: Front-end overview. This is the starting view of the front-end, and contains two customisable graphs, the latest incoming exceptions, a search field, and an overview of exception types generating the most heat.



Exception Detail

Title ...

Graph type ● Line ○ Bar ○ Radar

Data ● Exception counts ○ Heat

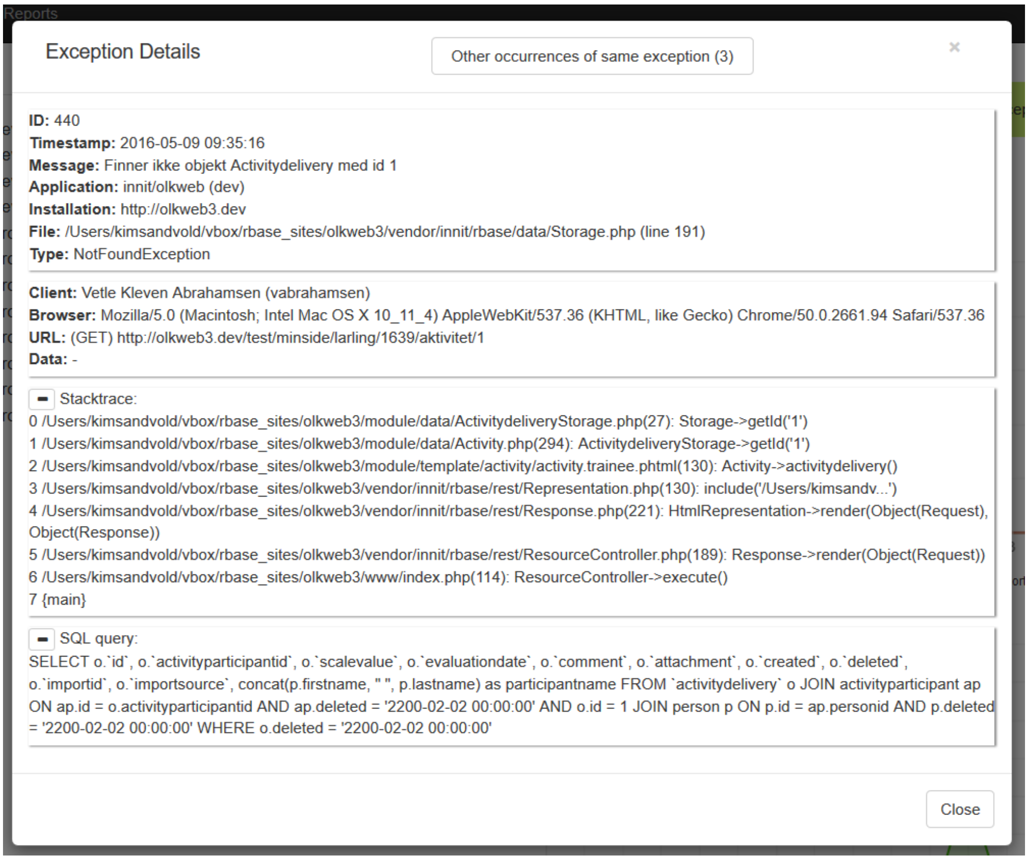
Types □ PDOException □ InvalidParameterException □ NotFoundException □ CurlException □ DomainException □ InvalidArgumentException

Applications □ Bingo Bazoopler (test) □ Kast katt (test) □ Kast terning (test) □ Twister Twanga (test) □ Yahtzee Yodle (test) □ innit/olkweb (dev) □ innit/olkweb (prod) □ innit/olkweb (test) □ rbase/Sponsorportalen (dev)

Timespan 1

Height 100

Figure 3: Graph settings. This is the settings window for customising what data to show in the graphs.



ID: 440
Timestamp: 2016-05-09 09:35:16
Message: Finner ikke objekt Activitydelivery med id 1
Application: innit/olkweb (dev)
Installation: http://olkweb3.dev
File: /Users/kimsandvold/vbox/rbase_sites/olkweb3/vendor/innit/rbase/data/Storage.php (line 191)
Type: NotFoundException

Client: Vetle Kleven Abrahamsen (vabrahamsen)
Browser: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94 Safari/537.36
URL: (GET) http://olkweb3.dev/test/minside/larling/1639/aktivitet/1
Data: -

▬ Stacktrace:

```
0 /Users/kimsandvold/vbox/rbase_sites/olkweb3/module/data/ActivitydeliveryStorage.php(27): Storage->getId('1')
1 /Users/kimsandvold/vbox/rbase_sites/olkweb3/module/data/Activity.php(294): ActivitydeliveryStorage->getId('1')
2 /Users/kimsandvold/vbox/rbase_sites/olkweb3/module/template/activity/activity.trainee.phtml(130): Activity->activitydelivery()
3 /Users/kimsandvold/vbox/rbase_sites/olkweb3/vendor/innit/rbase/rest/Representation.php(130): include('/Users/kimsandv...')
4 /Users/kimsandvold/vbox/rbase_sites/olkweb3/vendor/innit/rbase/rest/Response.php(221): HtmlRepresentation->render(Object(Request), Object(Response))
5 /Users/kimsandvold/vbox/rbase_sites/olkweb3/vendor/innit/rbase/rest/ResourceController.php(189): Response->render(Object(Request))
6 /Users/kimsandvold/vbox/rbase_sites/olkweb3/www/index.php(114): ResourceController->execute()
7 {main}
```

▬ SQL query:

```
SELECT o.`id`, o.`activityparticipantid`, o.`scalevalue`, o.`evaluationdate`, o.`comment`, o.`attachment`, o.`created`, o.`deleted`, o.`importid`, o.`importsource`, concat(p.firstname, " ", p.lastname) as participantname FROM `activitydelivery` o JOIN activityparticipant ap ON ap.id = o.activityparticipantid AND ap.deleted = '2200-02-02 00:00:00' AND o.id = 1 JOIN person p ON p.id = ap.personid AND p.deleted = '2200-02-02 00:00:00' WHERE o.deleted = '2200-02-02 00:00:00'
```

Figure 4: Displaying a single exception report. Note that this is test-data, it is not a real person.

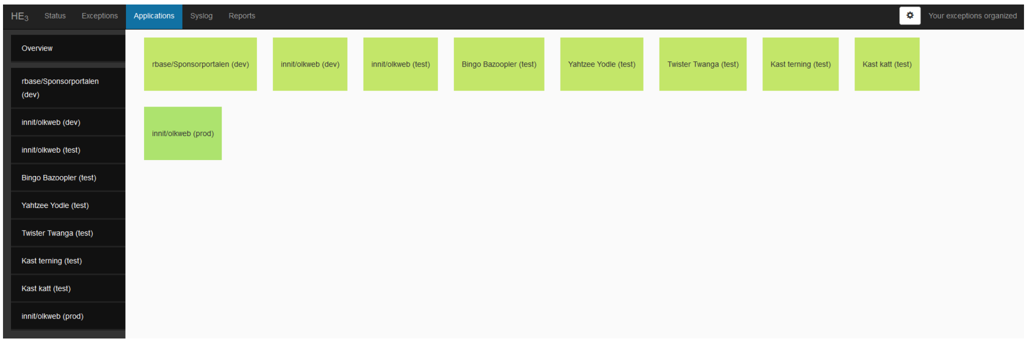
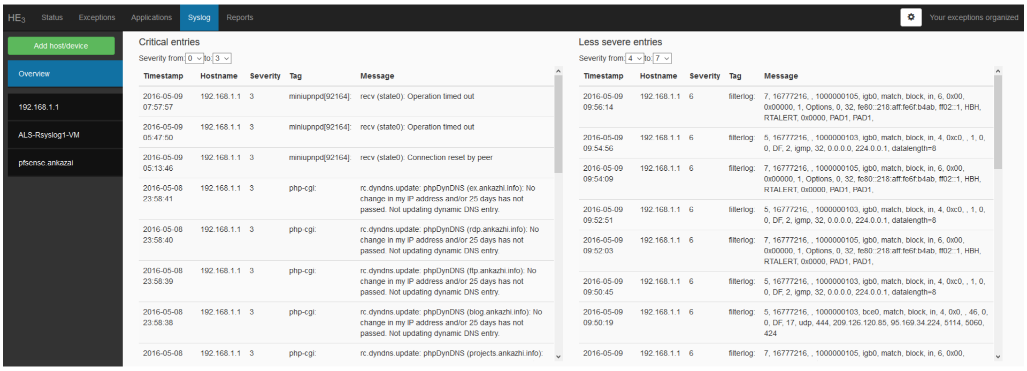


Figure 5: Application overview. Each box is a link to each application, and changes colour depending on the heat of the application.



Critical entries

Severity from: 0 to: 3

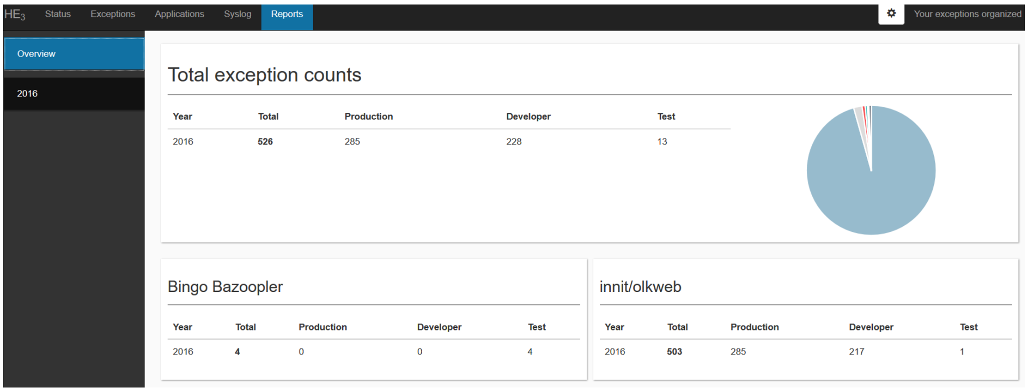
Timestamp	Hostname	Severity	Tag	Message
2016-05-09 07:57:57	192.168.1.1	3	miniupnpd[92164]:	recv (state0): Operation timed out
2016-05-09 05:47:50	192.168.1.1	3	miniupnpd[92164]:	recv (state0): Operation timed out
2016-05-09 05:13:46	192.168.1.1	3	miniupnpd[92164]:	recv (state0): Connection reset by peer
2016-05-08 23:58:41	192.168.1.1	3	php-cgi:	rc.dyndns.update: phpDynDNS (ex.ankazhi.info): No change in my IP address and/or 25 days has not passed. Not updating dynamic DNS entry.
2016-05-08 23:58:40	192.168.1.1	3	php-cgi:	rc.dyndns.update: phpDynDNS (rdp.ankazhi.info): No change in my IP address and/or 25 days has not passed. Not updating dynamic DNS entry.
2016-05-08 23:58:39	192.168.1.1	3	php-cgi:	rc.dyndns.update: phpDynDNS (ftp.ankazhi.info): No change in my IP address and/or 25 days has not passed. Not updating dynamic DNS entry.
2016-05-08 23:58:38	192.168.1.1	3	php-cgi:	rc.dyndns.update: phpDynDNS (blog.ankazhi.info): No change in my IP address and/or 25 days has not passed. Not updating dynamic DNS entry.
2016-05-08	192.168.1.1	3	php-cgi:	rc.dyndns.update: phpDynDNS (projects.ankazhi.info):

Less severe entries

Severity from: 4 to: 7

Timestamp	Hostname	Severity	Tag	Message
2016-05-09 09:56:14	192.168.1.1	6	filterlog:	7, 16777216, , 1000000105, igb0, match, block, in, 6, 0x00, 0x00000, 1, Options, 0, 32, fe80::218:aff:fe6f:b4ab, ff02::1, HBH, RTALERT, 0x0000, PAD1, PAD1,
2016-05-09 09:54:56	192.168.1.1	6	filterlog:	5, 16777216, , 1000000103, igb0, match, block, in, 4, 0xc0, , 1, 0, 0, DF, 2, igmp, 32, 0.0.0.0, 224.0.0.1, datalength=8
2016-05-09 09:54:09	192.168.1.1	6	filterlog:	7, 16777216, , 1000000105, igb0, match, block, in, 6, 0x00, 0x00000, 1, Options, 0, 32, fe80::218:aff:fe6f:b4ab, ff02::1, HBH, RTALERT, 0x0000, PAD1, PAD1,
2016-05-09 09:52:51	192.168.1.1	6	filterlog:	5, 16777216, , 1000000103, igb0, match, block, in, 4, 0xc0, , 1, 0, 0, DF, 2, igmp, 32, 0.0.0.0, 224.0.0.1, datalength=8
2016-05-09 09:52:03	192.168.1.1	6	filterlog:	7, 16777216, , 1000000105, igb0, match, block, in, 6, 0x00, 0x00000, 1, Options, 0, 32, fe80::218:aff:fe6f:b4ab, ff02::1, HBH, RTALERT, 0x0000, PAD1, PAD1,
2016-05-09 09:50:45	192.168.1.1	6	filterlog:	5, 16777216, , 1000000103, igb0, match, block, in, 4, 0xc0, , 1, 0, 0, DF, 2, igmp, 32, 0.0.0.0, 224.0.0.1, datalength=8
2016-05-09 09:50:19	192.168.1.1	6	filterlog:	5, 16777216, , 1000000103, bce0, match, block, in, 4, 0x0, , 46, 0, 0, DF, 17, udp, 444, 209.126.120.85, 95.169.34.224, 5114, 5060, 424
2016-05-09	192.168.1.1	6	filterlog:	7, 16777216, , 1000000105, igb0, match, block, in, 6, 0x00,

Figure 6: Syslog display. The left navbar are all devices sending syslog reports. The left part of the screen are the most critical syslog reports as described by the standard, sorted in descending order of arrival. The right part are all syslog reports, sorted in descending order of arrival.



Total exception counts

Year	Total	Production	Developer	Test
2016	**526**	285	228	13

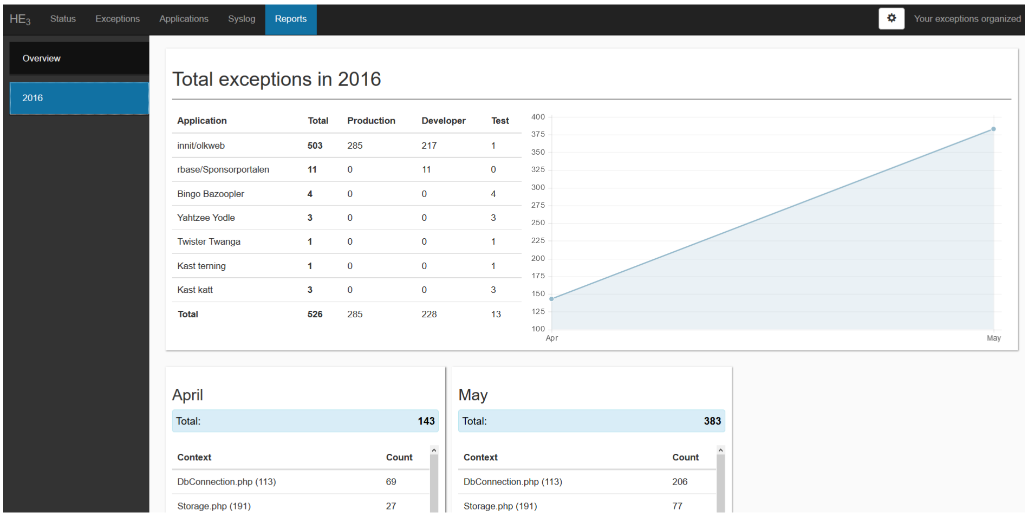
Bingo Bazoopler

Year	Total	Production	Developer	Test
2016	**4**	0	0	4

innit/olkweb

Year	Total	Production	Developer	Test
2016	**503**	285	217	1

Figure 7: Report display. This is the automatically generated monthly report feature, displaying the overview page. It contains the number of exceptions per year, split by production, development and test modes, along with a pie chart. Below it shows the total number per application.



Total exceptions in 2016

Application	Total	Production	Developer	Test
innit/olkweb	**503**	285	217	1
rbase/Sponsorportalen	**11**	0	11	0
Bingo Bazoopler	**4**	0	0	4
Yahtzee Yodle	**3**	0	0	3
Twister Twanga	**1**	0	0	1
Kast terning	**1**	0	0	1
Kast katt	**3**	0	0	3
Total	**526**	285	228	13

April

Total: **143**

Context	Count
DbConnection.php (113)	69
Storage.php (191)	27

May

Total: **383**

Context	Count
DbConnection.php (113)	206
Storage.php (191)	77

Figure 8: Report display. This displays all exceptions in a year, split by each application and the mode (production, development or test), along with a line graph of the data. Below the same data is displayed per month.

G Readme

This is a tutorial for setting up the "HEEE" front-end and back-end respectively.

G.1 Front-end

Dependencies

- Webserver of your choice. Tested with Apache2.
- Node.js - 0.12 or later. If earlier is wanted, such as 0.10 from apt package, remove minification of css in gulp's build task (cssnano).

Setting up the front-end

Option A: using the latest built version

1. Copy the "build" folder from the repository (git@bitbucket.org:exhandle/h3e-frontend.git) into the web root.
2. Configure app/appConfig.js to match the backend setup.

Option B: building the app from source

1. Pull the repository down (git@bitbucket.org:exhandle/h3e-frontend.git)
2. Run 'sudo npm install - -production' (double dash) and 'sudo bower install - -allow-root (double dash) in the directory. If there is a warning about mismatch, pick the latest.
3. Configure app/appConfig.js to match the backend setup.
4. Run 'sudo npm run build' to build the application and generate a build folder.
5. Place the aforementioned folder into the web root.

G.2 Back-end

The setup has been tested extensively on Ubuntu 14.04 LTS. Should however work on other Debian based operating systems.

Dependencies

All dependencies except PHP 7 and MariaDB exists in the apt repository. Other versions might work, but only tested with these:

- rsyslog - 7.4.4-1 or later
- rsyslog-mysql - 7.4.4-1 or later
- PHP version: 7.0.3-9 or later
- 10.1.13-MariaDB or later
- Git
- Webserver of choice (Tutorial assumes Apache)

Installing and configuring the back-end

Enable download of MariaDB and install:

```
$ sudo apt-get install software-properties-common
$ sudo apt-key adv --recv-keys --keyserver hkp://keyserver.
    ubuntu.com:80 0xcbcb082a1bb943db
$ sudo add-apt-repository 'deb [arch=amd64,i386]
http://lon1.mirrors.digitalocean.com/mariadb/repo/10.1/ubuntu
    trusty main'
$ sudo apt-get update
$ sudo apt-get -y install mariadb-server
```

Enable download of PHP7 and install plus required packages:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:ondrej/php
$ sudo apt-get update
$ sudo apt-get -y install php7.0 php7.0-mbstring php7.0-xml
    php7.0-mysql
```

Download rest of the dependencies:

```
$ sudo apt-get -y install rsyslog rsyslog-mysql apache2
    libapache2-mod-php7.0 git
```

Clone and pull down the repository:

```
$ git clone username@bitbucket.org:exhandle/h3e-backend.git /
    tmp/h3e-backend
```

Move repository to location to www data and chown it:

```
$ sudo mv /tmp/h3e-backend /var/www/
$ sudo chown www-data:www-data /var/www/h3e-backend
$ sudo chmod -R o+w /var/www/h3e-backend/storage
```

Move the old webroot and symlink the public folder to the webroot:

```
$ sudo mv /var/www/html /var/www/html-old
$ sudo ln -s /var/www/h3e-backend/public /var/www/html
```

Install Composer modules:

```
$ sudo php /var/www/h3e-backend/composer.phar self-update
$ sudo php /var/www/h3e-backend/composer.phar install
```

Create database and user:

```
$ mysql -u root -p
# Enter the password you chose during installation of MariaDB.

MariaDB > CREATE DATABASE `h3e-backend`;
MariaDB > CREATE USER 'h3e-backend'@'localhost' IDENTIFIED BY
    'SecretPW';
MariaDB > GRANT ALL PRIVILEGES ON `h3e-backend`.* TO 'h3e-
    backend'@'localhost';
```

Move and edit the environment file:

```
$ sudo mv .env.example .env
$ sudo php artisan key:generate
$ sudo vi .env

# Ensure that APP_KEY is set to a value starting with base64.
# Change the DB_DATABASE, DB_USERNAME and DB_PASSWORD to what
    you chose on the previous step.
# Also change DB_DATABASE2, DB_USERNAME2 and DB_PASSWORD2 to
    these values,
# if you let rsyslog do the setup for you.

DB_DATABASE2 = Syslog
DB_USERNAME2 = rsyslog
DB_PASSWORD2 = (The password you chose in the dialog during
    install of rsyslog)

# Optional
# Change the values of the following two variables if need by.
# They control the percent that the heat calculation uses for
    degradation, as well as how often it occurs.
DEGRADATION_PERCENTAGE=10
DEGRADATION_TIMING_MINUTES=1
```

Migrate the database tables:

```
# To create the database structure, artisan must be run.
    However the database user also requires the
# SUPER privilege for the initial setup, so we will grant this
    and then revoke it afterwards.
mysql -u root -p
# Enter the password you chose during installation of MariaDB.

MariaDB > GRANT SUPER ON *.* TO 'h3e-backend'@'localhost';
```

```
  # Run the migration:
  $ sudo php artisan migrate:refresh

  #If all succeeds, we can revoke the super privileges.
  mysql −u root −p
  # Enter the password you chose during installation of MariaDB.
  MariaDB > REVOKE SUPER ON ∗.∗ FROM ’h3e−backend’@’localhost’;
```

Check that it works: That is everything required to set up our backend. Try navigating to the root document at server/v1/

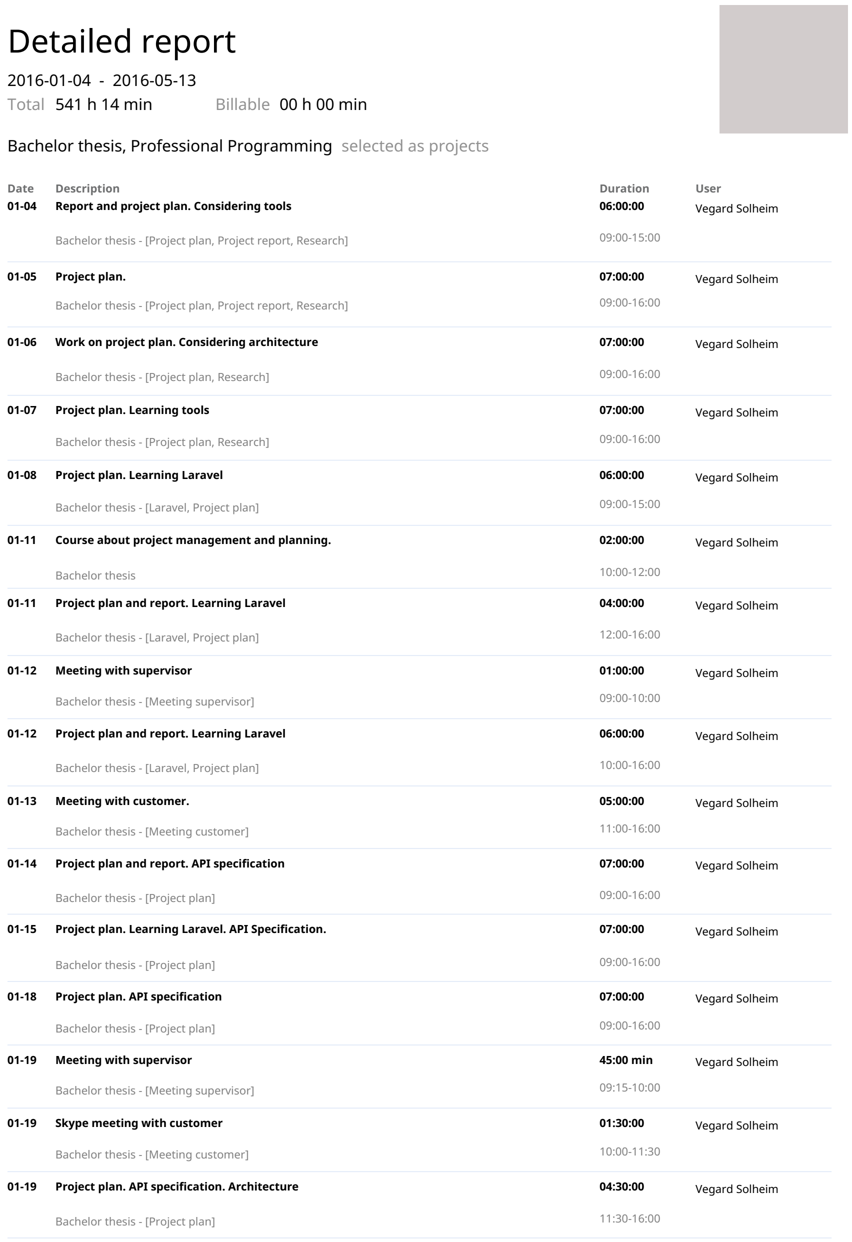
If it shows an information documentation, you are good to go.

If however you get an 404 ’Page Not Found’ error, mod_rewrite is not enabled or the .htaccess is not allowed to override the configuration.

```
Enable mod_rewrite:
  # Enable the mod and restart Apache2
  $ sudo a2enmod rewrite && sudo service apache2 restart


If the page still is not shown, you need to allow override.

# Allow override
Edit the Directory for /var/www/ in the configuration file for
   apache2 (/etc/apache2/apache2.conf).
sudo vi /etc/apache2/apache2.conf
Change AllowOverride from None to All

<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>

# Restart Apache2
sudo service apache2 restart
```

H Time Log

H.1 Time Log Vegard



Detailed report

2016-01-04 - 2016-05-13

Total 541 h 14 min Billable 00 h 00 min

Bachelor thesis, Professional Programming selected as projects

Date	Description	Duration	User
01-04	**Report and project plan. Considering tools** Bachelor thesis - [Project plan, Project report, Research]	**06:00:00** 09:00-15:00	Vegard Solheim
01-05	**Project plan.** Bachelor thesis - [Project plan, Project report, Research]	**07:00:00** 09:00-16:00	Vegard Solheim
01-06	**Work on project plan. Considering architecture** Bachelor thesis - [Project plan, Research]	**07:00:00** 09:00-16:00	Vegard Solheim
01-07	**Project plan. Learning tools** Bachelor thesis - [Project plan, Research]	**07:00:00** 09:00-16:00	Vegard Solheim
01-08	**Project plan. Learning Laravel** Bachelor thesis - [Laravel, Project plan]	**06:00:00** 09:00-15:00	Vegard Solheim
01-11	**Course about project management and planning.** Bachelor thesis	**02:00:00** 10:00-12:00	Vegard Solheim
01-11	**Project plan and report. Learning Laravel** Bachelor thesis - [Laravel, Project plan]	**04:00:00** 12:00-16:00	Vegard Solheim
01-12	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 09:00-10:00	Vegard Solheim
01-12	**Project plan and report. Learning Laravel** Bachelor thesis - [Laravel, Project plan]	**06:00:00** 10:00-16:00	Vegard Solheim
01-13	**Meeting with customer.** Bachelor thesis - [Meeting customer]	**05:00:00** 11:00-16:00	Vegard Solheim
01-14	**Project plan and report. API specification** Bachelor thesis - [Project plan]	**07:00:00** 09:00-16:00	Vegard Solheim
01-15	**Project plan. Learning Laravel. API Specification.** Bachelor thesis - [Project plan]	**07:00:00** 09:00-16:00	Vegard Solheim
01-18	**Project plan. API specification** Bachelor thesis - [Project plan]	**07:00:00** 09:00-16:00	Vegard Solheim
01-19	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**45:00 min** 09:15-10:00	Vegard Solheim
01-19	**Skype meeting with customer** Bachelor thesis - [Meeting customer]	**01:30:00** 10:00-11:30	Vegard Solheim
01-19	**Project plan. API specification. Architecture** Bachelor thesis - [Project plan]	**04:30:00** 11:30-16:00	Vegard Solheim

01-20	**GUI mocking and testing. Learning Laravel** Bachelor thesis - [AngularJS, Group meeting, Laravel]	**07:00:00** 09:00-16:00	Vegard Solheim
01-21	**Learning Laravel** Professional Programming - [Laravel]	**07:00:00** 09:00-16:00	Vegard Solheim
01-22	**Learning Laravel** Professional Programming - [Laravel]	**05:00:00** 09:00-14:00	Vegard Solheim
01-22	**Professional programming class** Professional Programming - [Class]	**02:00:00** 14:00-16:00	Vegard Solheim
01-25	**Project plan** Bachelor thesis	**04:44:53** 09:20-14:04	Vegard Solheim
01-25	**Learning Laravel coding standards.** Professional Programming - [Laravel, Research]	**01:27:19** 14:05-15:32	Vegard Solheim
01-26	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 09:00-10:00	Vegard Solheim
01-26	**Project plan** Bachelor thesis - [Project plan]	**06:00:00** 10:00-16:00	Vegard Solheim
01-27	**Meeting with Innit, incl. hour drive each way** Bachelor thesis - [Meeting customer]	**03:30:00** 11:00-14:30	Vegard Solheim
01-27	**Learning AngularJS** Bachelor thesis - [AngularJS]	**45:00 min** 16:30-17:15	Vegard Solheim
01-27	**Checking out Haskell tutorial** Professional Programming - [Learning other language]	**01:15:00** 17:15-18:30	Vegard Solheim
01-28	**Working on and delivered project plan** Bachelor thesis - [Project plan]	**04:00:00** 09:00-13:00	Vegard Solheim
01-28	**Learning Laravel** Professional Programming - [Laravel]	**03:00:00** 13:00-16:00	Vegard Solheim
01-29	**Learning Laravel** Bachelor thesis - [Laravel]	**03:00:00** 09:00-12:00	Vegard Solheim
01-29	**Learning Laravel coding standards** Professional Programming - [Laravel]	**02:00:00** 12:00-14:00	Vegard Solheim
01-29	**Professional programming class** Professional Programming - [Class]	**02:00:00** 14:00-16:00	Vegard Solheim
02-01	**Discussing architecture and API design** Professional Programming - [Front-end, Group meeting]	**02:00:00** 09:00-11:00	Vegard Solheim
02-01	**Creating use case diagram and architecture models** Bachelor thesis - [Back-end, Front-end]	**03:00:00** 11:00-14:00	Vegard Solheim
02-01	**Moving info to Confluence** Professional Programming	**02:00:00** 14:00-16:00	Vegard Solheim
02-02	**Worked on system architecture** Bachelor thesis - [Back-end, Front-end]	**02:00:00** 09:00-11:00	Vegard Solheim

02-02	**Updated Confluence** Professional Programming - [Confluence]	**01:00:00** 11:00-12:00	Vegard Solheim
02-02	**Learning to use JIRA and Confluence** Professional Programming - [Confluence, JIRA]	**04:00:00** 12:00-16:00	Vegard Solheim
02-03	**Learning to use JIRA and Confluence** Professional Programming - [Confluence, JIRA]	**03:00:00** 09:00-12:00	Vegard Solheim
02-03	**Development on back-end. Application route.** Bachelor thesis - [Back-end, Developing]	**01:30:00** 12:00-13:30	Vegard Solheim
02-03	**Research JIRA plugins** Professional Programming - [JIRA, Research]	**30:00 min** 13:30-14:00	Vegard Solheim
02-03	**Development on back-end. Installation route.** Bachelor thesis - [Back-end, Developing]	**01:15:00** 14:00-15:15	Vegard Solheim
02-03	**Development on back-end. Exception type route.** Bachelor thesis - [Back-end, Developing]	**45:00 min** 15:15-16:00	Vegard Solheim
02-04	**Learning about Laravel/PHPUnit testing** Professional Programming - [Laravel]	**02:00:00** 09:00-11:00	Vegard Solheim
02-04	**Development on backend. Fixed PHPUnit bug** Bachelor thesis - [Back-end, Developing, Laravel]	**35:00 min** 11:00-11:35	Vegard Solheim
02-04	**Learning about Laravel/PHPUnit testing** Professional Programming - [Laravel]	**55:00 min** 11:35-12:30	Vegard Solheim
02-04	**Working on test cases for backend.** Bachelor thesis - [Back-end, Developing, Laravel]	**30:00 min** 12:30-13:00	Vegard Solheim
02-04	**Code review on pull request.** Professional Programming - [Back-end, Code review, Laravel]	**01:15:00** 13:00-14:15	Vegard Solheim
02-04	**Working on test cases for backend.** Bachelor thesis	**01:00:00** 14:15-15:15	Vegard Solheim
02-05	**Development on backend. Fixed file that didn't get committed** Bachelor thesis - [Back-end, Developing]	**01:45:00** 09:00-10:45	Vegard Solheim
02-05	**Code review on pull request.** Professional Programming - [Back-end, Code review]	**15:00 min** 10:45-11:00	Vegard Solheim
02-05	**Created tests and fixed bug causing tests to fail** Professional Programming - [Back-end, Developing, Laravel]	**02:00:00** 11:00-13:00	Vegard Solheim
02-05	**Research for Laravel** Professional Programming - [Laravel, Research]	**30:00 min** 13:00-13:30	Vegard Solheim
02-05	**Code review backend** Professional Programming - [Back-end, Code review]	**45:00 min** 13:30-14:15	Vegard Solheim
02-05	**Trying to configure JIRA** Professional Programming - [Class, JIRA]	**01:45:00** 14:15-16:00	Vegard Solheim

02-08	**Preparing for meeting with supervisor** Bachelor thesis - [Group meeting]	**01:00:00** 09:00-10:00	Vegard Solheim
02-08	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 10:00-11:00	Vegard Solheim
02-08	**Research C-language** Professional Programming - [Learning other language, Research]	**01:15:00** 11:00-12:15	Vegard Solheim
02-08	**Learning Laravel** Professional Programming - [Laravel, Research]	**02:45:00** 12:15-15:00	Vegard Solheim
02-09	**Learning Laravel** Professional Programming - [Laravel]	**05:00:00** 09:00-14:00	Vegard Solheim
02-09	**Professional programming class** Professional Programming - [Class]	**01:00:00** 14:00-15:00	Vegard Solheim
02-09	**Learning Laravel** Professional Programming - [Laravel]	**45:00 min** 15:00-15:45	Vegard Solheim
02-10	**Documenting decision on Confluence** Professional Programming - [Confluence, Documenting]	**35:00 min** 09:00-09:35	Vegard Solheim
02-10	**Code review** Professional Programming - [Code review]	**02:10:00** 09:35-11:45	Vegard Solheim
02-10	**Updating diagrams** Bachelor thesis - [Confluence, Documenting]	**45:00 min** 11:45-12:30	Vegard Solheim
02-10	**Code review** Professional Programming - [Code review]	**30:00 min** 12:30-13:00	Vegard Solheim
02-10	**Updating diagrams** Bachelor thesis - [Confluence, Documenting]	**35:00 min** 13:00-13:35	Vegard Solheim
02-10	**Code review on pull request.** Professional Programming - [Code review]	**01:15:00** 13:35-14:50	Vegard Solheim
02-10	**Updated documentation** Bachelor thesis - [Confluence, Documenting]	**01:10:00** 14:50-16:00	Vegard Solheim
02-11	**Development on backend.** Bachelor thesis - [Back-end, Developing, Laravel]	**04:15:00** 09:00-13:15	Vegard Solheim
02-11	**Development on backend, making tests** Professional Programming - [Back-end, Developing]	**02:15:00** 13:15-15:30	Vegard Solheim
02-12	**Moving past timelog to Toggl** Bachelor thesis	**01:15:00** 09:00-10:15	Vegard Solheim
02-12	**Project report** Bachelor thesis - [Project report]	**03:45:00** 10:15-14:00	Vegard Solheim
02-12	**Professional programming class** Professional Programming - [Class]	**02:00:00** 14:00-16:00	Vegard Solheim
02-15	**Project report** Bachelor thesis - [Project report]	**07:00:00** 09:00-16:00	Vegard Solheim

02-16	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**35:00 min** 09:00-09:35	Vegard Solheim
02-16	**Project report** Bachelor thesis - [Project report]	**05:25:00** 09:35-15:00	Vegard Solheim
02-17	**Documenting formats** Professional Programming - [Back-end, Documenting]	**02:30:00** 09:00-11:30	Vegard Solheim
02-17	**Assisting bugfixing** Professional Programming - [Back-end, Developing, Laravel]	**01:00:00** 11:30-12:30	Vegard Solheim
02-17	**Project report** Bachelor thesis - [Project report]	**03:30:00** 12:30-16:00	Vegard Solheim
02-18	**Project report** Bachelor thesis - [Project report]	**01:40:00** 09:20-11:00	Vegard Solheim
02-18	**Discussing front-end design** Professional Programming - [Front-end]	**45:00 min** 11:00-11:45	Vegard Solheim
02-18	**Project plan - database design and requirements** Bachelor thesis - [Project report]	**02:45:00** 12:45-15:30	Vegard Solheim
02-19	**Project report** Bachelor thesis - [Project report, Research]	**02:00:00** 09:00-11:00	Vegard Solheim
02-19	**Discussing front-end design** Professional Programming - [Front-end]	**01:00:00** 11:00-12:00	Vegard Solheim
02-19	**Updating front-end diagram** Professional Programming - [Documenting]	**02:00:00** 12:00-14:00	Vegard Solheim
02-19	**Professional programming class** Professional Programming - [Class]	**02:00:00** 14:00-16:00	Vegard Solheim
02-22	**Project report** Bachelor thesis - [Project report]	**04:00:00** 09:00-13:00	Vegard Solheim
02-22	**Project report** Bachelor thesis	**50:00 min** 14:30-15:20	Vegard Solheim
02-22	**Updating Confluence documentation** Professional Programming - [Confluence, Documenting]	**40:00 min** 15:20-16:00	Vegard Solheim
02-23	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**45:00 min** 09:00-09:45	Vegard Solheim
02-23	**Project report - fixing from meeting feedback** Professional Programming - [Project report]	**01:15:00** 09:45-11:00	Vegard Solheim
02-23	**Updating diagrams** Professional Programming - [Documenting]	**02:15:00** 11:00-13:15	Vegard Solheim
02-24	**Project report** Bachelor thesis - [Project report]	**07:00:00** 09:00-16:00	Vegard Solheim
02-25	**Project report** Bachelor thesis - [Project report]	**03:00:00** 09:00-12:00	Vegard Solheim

02-29	**Project report** Bachelor thesis - [Project report]	**03:57:00** 09:00-12:57	Vegard Solheim
02-29	**Project report** Bachelor thesis - [Project report]	**02:45:00** 13:15-16:00	Vegard Solheim
03-01	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 09:00-10:00	Vegard Solheim
03-01	**Project report and updated diagrams** Bachelor thesis - [Project report]	**03:00:00** 10:00-13:00	Vegard Solheim
03-01	**Updated documentation on Confluence** Professional Programming - [Confluence, Documenting]	**01:30:00** 13:00-14:30	Vegard Solheim
03-01	**Project report** Bachelor thesis - [Project report]	**45:00 min** 14:30-15:15	Vegard Solheim
03-02	**Project report** Bachelor thesis - [Project report]	**02:00:00** 09:00-11:00	Vegard Solheim
03-02	**Updating Confluence documentation** Professional Programming - [Confluence, Documenting]	**02:30:00** 11:00-13:30	Vegard Solheim
03-02	**Project report** Bachelor thesis - [Project report]	**02:30:00** 13:30-16:00	Vegard Solheim
03-03	**Project report** Bachelor thesis - [Project report]	**04:00:00** 12:30-16:30	Vegard Solheim
03-04	**Project report** Bachelor thesis - [Project report]	**02:00:00** 09:00-11:00	Vegard Solheim
03-04	**Project report** Bachelor thesis - [Project report]	**02:30:00** 11:30-14:00	Vegard Solheim
03-04	**Professional Programming class** Professional Programming - [Class]	**02:00:00** 14:00-16:00	Vegard Solheim
03-07	**Meeting with Innit** Bachelor thesis - [Meeting customer]	**04:15:00** 11:00-15:15	Vegard Solheim
03-08	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 09:15-10:15	Vegard Solheim
03-08	**Group meeting on how to do severity ratings** Professional Programming - [Group meeting]	**01:00:00** 10:15-11:15	Vegard Solheim
03-08	**Recorded decision on Confluence** Professional Programming - [Confluence]	**10:00 min** 11:15-11:25	Vegard Solheim
03-08	**Discussing best database setup** Professional Programming - [Group meeting]	**01:00:00** 11:25-12:25	Vegard Solheim
03-08	**Code review** Professional Programming - [Code review]	**10:00 min** 12:25-12:35	Vegard Solheim
03-08	**Project report** Bachelor thesis - [Project report]	**25:00 min** 12:35-13:00	Vegard Solheim

03-08	**Updating use case diagrams** Professional Programming - [Documenting]	**55:00 min** 13:00-13:55	Vegard Solheim
03-08	**Project report** Bachelor thesis - [Project report]	**01:35:00** 13:55-15:30	Vegard Solheim
03-09	**Project report** Bachelor thesis - [Project report]	**03:30:00** 08:45-12:15	Vegard Solheim
03-09	**Course in report writing** Bachelor thesis - [Class]	**45:00 min** 12:15-13:00	Vegard Solheim
03-09	**Project report** Bachelor thesis - [Project report]	**02:30:00** 13:00-15:30	Vegard Solheim
03-10	**Working on informational website** Bachelor thesis - [Infosite]	**03:45:00** 09:00-12:45	Vegard Solheim
03-10	**Reporting that Confluence is down** Professional Programming - [Confluence]	**15:00 min** 12:45-13:00	Vegard Solheim
03-10	**Working on informational website** Bachelor thesis - [Infosite]	**03:00:00** 13:00-16:00	Vegard Solheim
03-11	**Project report** Bachelor thesis - [Project report]	**01:00:00** 09:00-10:00	Vegard Solheim
03-11	**Small fixes on informational website** Bachelor thesis - [Infosite]	**20:00 min** 10:00-10:20	Vegard Solheim
03-11	**Updating documentation on Confluence** Professional Programming - [Confluence]	**25:00 min** 10:20-10:45	Vegard Solheim
03-11	**Project report** Bachelor thesis - [Project report]	**01:00:00** 10:45-11:45	Vegard Solheim
03-11	**Discussing front-end filtering options** Bachelor thesis - [Group meeting]	**01:45:00** 11:45-13:30	Vegard Solheim
03-11	**Documenting decision on Confluence** Professional Programming - [Confluence]	**45:00 min** 13:30-14:15	Vegard Solheim
03-11	**Professional Programming class** Professional Programming - [Class]	**01:45:00** 14:15-16:00	Vegard Solheim
03-14	**Project report** Bachelor thesis - [Project report]	**02:00:00** 09:00-11:00	Vegard Solheim
03-14	**Gantt diagram for actual time spent** Bachelor thesis - [Project report]	**30:00 min** 11:00-11:30	Vegard Solheim
03-14	**Project report** Bachelor thesis - [Project report]	**02:40:00** 12:00-14:40	Vegard Solheim
03-14	**Updating database model diagram** Professional Programming - [Documenting]	**01:20:00** 14:40-16:00	Vegard Solheim
03-15	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**50:00 min** 09:15-10:05	Vegard Solheim

03-15	**Updating database model diagram** Professional Programming - [Documenting]	**02:55:00** 10:05-13:00	Vegard Solheim
03-15	**Project report** Bachelor thesis - [Project report]	**03:00:00** 13:00-16:00	Vegard Solheim
03-16	**Project report** Bachelor thesis - [Project report]	**03:00:00** 09:00-12:00	Vegard Solheim
03-17	**Project report** Bachelor thesis - [Project report]	**07:00:00** 09:00-16:00	Vegard Solheim
03-18	**Discussing data model** Bachelor thesis - [Back-end]	**01:00:00** 09:00-10:00	Vegard Solheim
03-18	**Learning AngularJS** Professional Programming - [AngularJS]	**04:35:00** 10:00-14:35	Vegard Solheim
03-29	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 09:00-10:00	Vegard Solheim
03-29	**Learning AngularJS** Professional Programming - [AngularJS]	**01:30:00** 10:00-11:30	Vegard Solheim
03-29	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**04:30:00** 11:30-16:00	Vegard Solheim
03-30	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**07:00:00** 09:00-16:00	Vegard Solheim
03-31	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**01:30:00** 09:00-10:30	Vegard Solheim
03-31	**Discussing and building algorithm for heat degradation** Professional Programming - [Back-end, Group meeting]	**01:45:00** 10:30-12:15	Vegard Solheim
03-31	**Documenting heat functionality** Professional Programming - [Documenting]	**01:25:00** 12:15-13:40	Vegard Solheim
03-31	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**02:20:00** 13:40-16:00	Vegard Solheim
04-01	**Discussing API** Professional Programming - [Group meeting]	**01:00:00** 09:00-10:00	Vegard Solheim
04-01	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**04:00:00** 10:00-14:00	Vegard Solheim
04-01	**Fixed algorithm for heat degradation** Professional Programming - [Developing, Documenting]	**50:00 min** 14:00-14:50	Vegard Solheim
04-01	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**01:10:00** 14:50-16:00	Vegard Solheim
04-04	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**03:00:00** 09:00-12:00	Vegard Solheim
04-05	**Refactoring front-end** Professional Programming - [AngularJS, Front-end]	**03:10:00** 09:00-12:10	Vegard Solheim

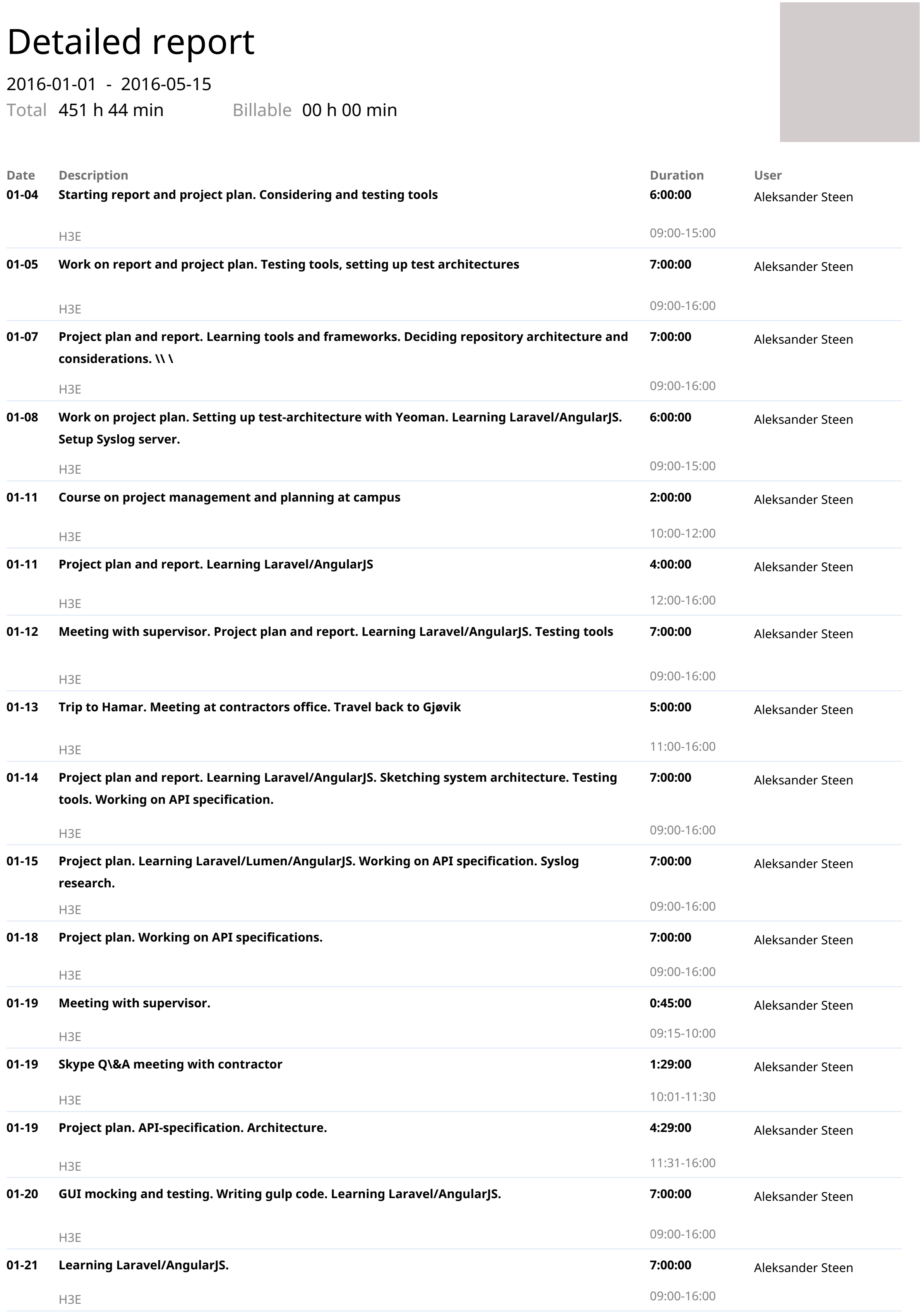
04-05	**Project report** Bachelor thesis - [Project report]	**03:50:00** 12:10-16:00	Vegard Solheim
04-06	**Project report** Bachelor thesis - [Project report]	**03:00:00** 09:00-12:00	Vegard Solheim
04-06	**Updated database documentation** Professional Programming - [Confluence, Documenting]	**01:00:00** 12:00-13:00	Vegard Solheim
04-06	**Project report** Bachelor thesis - [Project report]	**03:00:00** 13:00-16:00	Vegard Solheim
04-07	**Project report** Bachelor thesis - [Project report]	**05:30:00** 08:45-14:15	Vegard Solheim
04-07	**Code review** Professional Programming - [Code review, Front-end]	**30:00 min** 14:15-14:45	Vegard Solheim
04-07	**Project report** Bachelor thesis - [Project report]	**45:00 min** 14:45-15:30	Vegard Solheim
04-08	**Project report** Bachelor thesis - [Project report]	**05:30:00** 08:30-14:00	Vegard Solheim
04-08	**Professional Programming class** Professional Programming - [Class]	**02:00:00** 14:00-16:00	Vegard Solheim
04-11	**Project report** Bachelor thesis - [Project report]	**04:00:00** 09:00-13:00	Vegard Solheim
04-12	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 09:00-10:00	Vegard Solheim
04-12	**Project report** Bachelor thesis - [Project report]	**05:00:00** 10:00-15:00	Vegard Solheim
04-13	**Project report** Bachelor thesis - [Project report]	**01:20:00** 09:00-10:20	Vegard Solheim
04-13	**Reading up on ORM** Professional Programming - [Research]	**50:00 min** 10:20-11:10	Vegard Solheim
04-13	**Project report** Bachelor thesis - [Project report]	**04:35:00** 11:10-15:45	Vegard Solheim
04-14	**Project report** Bachelor thesis - [Project report]	**01:00:00** 09:00-10:00	Vegard Solheim
04-14	**Project report** Bachelor thesis - [Project report]	**01:15:00** 14:45-16:00	Vegard Solheim
04-15	**Project report** Bachelor thesis - [Project report]	**01:00:00** 09:00-10:00	Vegard Solheim
04-15	**Meeting with GUI professor** Professional Programming	**01:00:00** 10:00-11:00	Vegard Solheim
04-15	**Project report - code listing** Professional Programming - [Project report]	**02:30:00** 11:00-13:30	Vegard Solheim

04-15	**Project report** Bachelor thesis - [Project report]	**45:00 min** 13:30-14:15	Vegard Solheim
04-15	**Professional Programming class** Professional Programming - [Class]	**01:45:00** 14:15-16:00	Vegard Solheim
04-18	**Project report** Bachelor thesis - [Project report]	**03:30:00** 09:00-12:30	Vegard Solheim
04-18	**Assisted with heat recalculation algorithm** Professional Programming - [Back-end, Developing]	**01:00:00** 12:30-13:30	Vegard Solheim
04-18	**Project report** Bachelor thesis - [Project report]	**01:20:00** 13:30-14:50	Vegard Solheim
04-19	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**01:00:00** 09:00-10:00	Vegard Solheim
04-19	**Project report** Bachelor thesis - [Project report]	**03:30:00** 10:00-13:30	Vegard Solheim
04-19	**Discussing heat data format and transfer** Professional Programming - [Group meeting]	**01:15:00** 13:30-14:45	Vegard Solheim
04-19	**Project report** Bachelor thesis - [Project report]	**01:15:00** 14:45-16:00	Vegard Solheim
04-20	**Project report** Bachelor thesis - [Project report]	**02:00:00** 09:00-11:00	Vegard Solheim
04-20	**Discussing heat map** Professional Programming - [Group meeting]	**01:00:00** 11:00-12:00	Vegard Solheim
04-20	**Project report** Bachelor thesis - [Project report]	**03:30:00** 12:00-15:30	Vegard Solheim
04-22	**Project report** Bachelor thesis - [Project report]	**05:00:00** 09:00-14:00	Vegard Solheim
04-22	**Professional Programming class** Professional Programming - [Class]	**02:00:00** 14:00-16:00	Vegard Solheim
04-25	**Project report** Bachelor thesis - [Project report]	**04:00:00** 09:00-13:00	Vegard Solheim
04-25	**Updating diagrams** Professional Programming - [Confluence, Documenting]	**02:30:00** 13:00-15:30	Vegard Solheim
04-26	**Project report** Bachelor thesis - [Project report]	**03:00:00** 09:00-12:00	Vegard Solheim
04-26	**Travel to meeting with Innit** Bachelor thesis	**01:30:00** 12:00-13:30	Vegard Solheim
04-26	**Meeting with Innit** Bachelor thesis - [Meeting customer]	**01:30:00** 13:30-15:00	Vegard Solheim
04-26	**Travel home from meeting with Innit** Bachelor thesis	**01:00:00** 15:00-16:00	Vegard Solheim

04-27	**Project report** Bachelor thesis - [Project report]	**06:05:00** 09:00-15:05	Vegard Solheim
04-28	**Project report** Bachelor thesis - [Project report]	**06:30:00** 09:00-15:30	Vegard Solheim
04-29	**Project report** Bachelor thesis - [Project report]	**03:30:00** 09:00-12:30	Vegard Solheim
05-02	**Project report** Bachelor thesis - [Project report]	**02:15:00** 09:15-11:30	Vegard Solheim
05-02	**Brief chat with supervisor** Bachelor thesis - [Meeting supervisor]	**30:00 min** 11:30-12:00	Vegard Solheim
05-02	**Project report** Bachelor thesis - [Project report]	**04:00:00** 12:00-16:00	Vegard Solheim
05-03	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**50:00 min** 09:15-10:05	Vegard Solheim
05-03	**Project report** Bachelor thesis - [Project report]	**05:05:00** 10:10-15:15	Vegard Solheim
05-04	**Project report** Bachelor thesis - [Project report]	**06:30:00** 09:00-15:30	Vegard Solheim
05-05	**Project report** Bachelor thesis - [Project report]	**03:30:00** 09:00-12:30	Vegard Solheim
05-05	**Project report** Bachelor thesis - [Project report]	**02:15:00** 13:15-15:30	Vegard Solheim
05-06	**Updating text about refactored code** Professional Programming - [Documenting, Project report]	**01:15:00** 09:30-10:45	Vegard Solheim
05-06	**Updating API documentatiion** Professional Programming - [Documenting, Project report]	**03:05:00** 10:55-14:00	Vegard Solheim
05-06	**Project report** Bachelor thesis - [Project report]	**15:00 min** 14:00-14:15	Vegard Solheim
05-06	**Professional programming class** Professional Programming - [Class]	**01:45:00** 14:15-16:00	Vegard Solheim
05-09	**Project report** Bachelor thesis - [Project report]	**05:00:00** 09:00-14:00	Vegard Solheim
05-10	**Meeting with supervisor** Bachelor thesis - [Meeting supervisor]	**45:00 min** 09:15-10:00	Vegard Solheim
05-10	**Project report** Bachelor thesis - [Project report]	**05:00:00** 10:00-15:00	Vegard Solheim
05-11	**Project report** Bachelor thesis - [Project report]	**03:25:00** 08:50-12:15	Vegard Solheim
05-11	**Project report** Bachelor thesis - [Project report]	**01:45:00** 12:45-14:30	Vegard Solheim

05-12	**Project report** Bachelor thesis - [Project report]	**05:30:00** 09:00-14:30	Vegard Solheim
05-13	**Project report** Bachelor thesis - [Project report]	**01:05:00** 09:00-10:05	Vegard Solheim
05-13	**Meeting with Innit** Bachelor thesis - [Meeting customer]	**50:00 min** 10:05-10:55	Vegard Solheim
05-13	**Finishing up project report for delivery** Bachelor thesis - [Project report]	**03:05:00** 10:55-14:00	Vegard Solheim

H.2 Time Log Aleksander



Detailed report

2016-01-01 - 2016-05-15

Total 451 h 44 min Billable 00 h 00 min

Date	Description	Duration	User
01-04	**Starting report and project plan. Considering and testing tools** H3E	**6:00:00** 09:00-15:00	Aleksander Steen
01-05	**Work on report and project plan. Testing tools, setting up test architectures** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-07	**Project plan and report. Learning tools and frameworks. Deciding repository architecture and considerations. \\ \\** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-08	**Work on project plan. Setting up test-architecture with Yeoman. Learning Laravel/AngularJS. Setup Syslog server.** H3E	**6:00:00** 09:00-15:00	Aleksander Steen
01-11	**Course on project management and planning at campus** H3E	**2:00:00** 10:00-12:00	Aleksander Steen
01-11	**Project plan and report. Learning Laravel/AngularJS** H3E	**4:00:00** 12:00-16:00	Aleksander Steen
01-12	**Meeting with supervisor. Project plan and report. Learning Laravel/AngularJS. Testing tools** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-13	**Trip to Hamar. Meeting at contractors office. Travel back to Gjøvik** H3E	**5:00:00** 11:00-16:00	Aleksander Steen
01-14	**Project plan and report. Learning Laravel/AngularJS. Sketching system architecture. Testing tools. Working on API specification.** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-15	**Project plan. Learning Laravel/Lumen/AngularJS. Working on API specification. Syslog research.** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-18	**Project plan. Working on API specifications.** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-19	**Meeting with supervisor.** H3E	**0:45:00** 09:15-10:00	Aleksander Steen
01-19	**Skype Q\&A meeting with contractor** H3E	**1:29:00** 10:01-11:30	Aleksander Steen
01-19	**Project plan. API-specification. Architecture.** H3E	**4:29:00** 11:31-16:00	Aleksander Steen
01-20	**GUI mocking and testing. Writing gulp code. Learning Laravel/AngularJS.** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-21	**Learning Laravel/AngularJS.** H3E	**7:00:00** 09:00-16:00	Aleksander Steen

01-22	**Learning Laravel/AngularJS. Examining Apache SysLog.** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-25	**Project plan. Syslog DB setup.** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
01-26	**Meeting with supervisor** H3E	**1:00:00** 09:00-10:00	Aleksander Steen
01-26	**Syslog GUI view mocking** H3E	**5:59:00** 10:01-16:00	Aleksander Steen
01-26	**Made Lo-Fi Syslog view prototype.** H3E	**1:29:00** 22:30-23:59	Aleksander Steen
01-27	**Trip to Hamar. Meeting at contractors office. Travel back to Gjøvik** H3E	**5:00:00** 11:00-16:00	Aleksander Steen
01-28	**Final Project plan.** H3E	**4:00:00** 09:00-13:00	Aleksander Steen
01-29	**Setup HP 2510-24G switch from Innit, examined data logged. More Angular learning.** H3E	**5:00:00** 09:00-14:00	Aleksander Steen
02-01	**Laravel Envoy setup and research** H3E	**4:00:00** 00:00-04:00	Aleksander Steen
02-01	**Create bash script to deploy and setup server environment plus a script to append and remove hosts from rsyslog config** H3E	**3:00:00** 09:00-12:00	Aleksander Steen
02-02	**Laravel Syslog route, controller, model** H3E	**6:00:00** 09:00-15:00	Aleksander Steen
02-02	**Learning about pre-/post hooks for Bitbucket** H3E	**1:00:00** 15:00-16:00	Aleksander Steen
02-03	**Setup pre-/post hooks for Backend server** H3E	**2:00:00** 09:00-11:00	Aleksander Steen
02-03	**laravel** H3E	**2:10:00** 11:02-13:12	Aleksander Steen
02-04	**Store/delete Syslog data** H3E	**5:00:00** 11:00-16:00	Aleksander Steen
02-05	**Syslog tests. Re-worked syslog route. Reworked the bash scripts to not fail hard.** H3E	**6:00:00** 10:00-16:00	Aleksander Steen
02-08	**Syslog route refactoring** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
02-09	**Syslog route refactoring** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
02-10	**Syslog route refactoring** H3E	**7:00:00** 09:00-16:00	Aleksander Steen
02-15	**Refactoring Syslog functions** H3E	**1:57:56** 09:06-11:04	Aleksander Steen
02-15	**Refactoring Syslog functions** H3E	**0:49:43** 12:15-13:05	Aleksander Steen

02-15	**Ignored argument to syslog hosts**	**1:12:43**	Aleksander Steen
	H3E	13:05-14:17	
02-15	**review olafurs pull request 28**	**0:11:50**	Aleksander Steen
	H3E	14:27-14:38	
02-15	**small fixes and update documantation**	**0:44:00**	Aleksander Steen
	H3E	14:39-15:23	
02-16	**Meeting with supervisor**	**0:48:24**	Aleksander Steen
	H3E	09:00-09:48	
02-16	**Angular refreshing**	**3:32:48**	Aleksander Steen
	H3E	09:48-13:21	
02-16	**Update confluence API spec**	**0:14:33**	Aleksander Steen
	H3E	13:21-13:35	
02-16	**Testing angular + restful api**	**0:23:45**	Aleksander Steen
	H3E	13:36-13:59	
02-16	**Learning how the frontend is setup**	**0:37:02**	Aleksander Steen
	H3E	14:08-14:45	
02-16	**Finalize syslog view mocking**	**0:13:07**	Aleksander Steen
	H3E	14:45-14:58	
02-17	**More work on syslog view**	**0:59:13**	Aleksander Steen
	H3E	09:21-10:20	
02-17	**Code Review Olafur's pull request**	**0:00:51**	Aleksander Steen
	H3E	10:20-10:21	
02-17	**Code Review Olafur's pull request**	**0:13:43**	Aleksander Steen
	H3E	10:22-10:35	
02-17	**Filter by host**	**0:47:24**	Aleksander Steen
	H3E	10:36-11:23	
02-17	**lunch**	**0:19:32**	Aleksander Steen
	(no project)	11:23-11:43	
02-17	**More work on syslog view**	**4:04:23**	Aleksander Steen
	H3E	11:48-15:52	
02-18	**Create jira issues**	**0:22:17**	Aleksander Steen
	H3E	09:45-10:07	
02-18	**Modal edit popups**	**0:01:50**	Aleksander Steen
	H3E	10:07-10:09	
02-18	**Use chunking instead of getting all the records**	**0:25:53**	Aleksander Steen
	H3E	10:09-10:35	
02-18	**Use chunking instead of getting all the records**	**0:00:03**	Aleksander Steen
	H3E	10:36-10:36	
02-18	**Routing**	**3:07:38**	Aleksander Steen
	H3E	10:36-13:43	
02-18	**Modals**	**2:26:56**	Aleksander Steen
	H3E	13:43-16:10	
02-22	**frontend development**	**1:00:04**	Aleksander Steen
	H3E	09:11-10:11	

02-22	**angular videos** H3E	**5:42:41** 10:11-15:53	Aleksander Steen
02-23	**Meeting with supervisor** H3E	**0:53:15** 09:00-09:53	Aleksander Steen
02-23	**edit report** H3E	**2:07:26** 09:53-12:01	Aleksander Steen
02-23	**sort by after syslog hosts** H3E	**0:23:25** 12:33-12:56	Aleksander Steen
02-24	**More work on syslog view** H3E	**1:59:30** 09:15-11:15	Aleksander Steen
02-24	**More work on syslog view** H3E	**3:53:37** 11:25-15:18	Aleksander Steen
02-25	**tie modal forms to the corresponding functions** H3E	**0:55:02** 09:33-10:28	Aleksander Steen
02-25	**More work on syslog view** H3E	**2:57:10** 11:10-14:07	Aleksander Steen
02-29	**fix syslog test in backend** H3E	**0:13:42** 09:30-09:43	Aleksander Steen
02-29	**report** H3E	**2:49:52** 09:43-12:33	Aleksander Steen
02-29	**Code Review Olafur's pull request** H3E	**0:23:53** 12:33-12:57	Aleksander Steen
02-29	**firedrill** H3E	**0:13:09** 12:57-13:10	Aleksander Steen
02-29	**report** H3E	**0:15:04** 13:10-13:26	Aleksander Steen
02-29	**report** H3E	**2:22:23** 13:34-15:56	Aleksander Steen
03-01	**angular** H3E	**5:46:00** 09:11-14:57	Aleksander Steen
03-02	**troubleshooting backend** H3E	**0:49:35** 09:05-09:54	Aleksander Steen
03-02	**angular** H3E	**1:05:10** 09:54-10:59	Aleksander Steen
03-02	**angular HEEE-191** H3E	**4:09:14** 11:07-15:16	Aleksander Steen
03-04	**syslog tests** (no project)	**2:27:53** 09:35-12:03	Aleksander Steen
03-07	**meeting with init** H3E	**7:00:28** 04:14-11:14	Aleksander Steen
03-08	**Meeting with supervisor** H3E	**1:11:13** 09:08-10:20	Aleksander Steen
03-08	**front end discussion** (no project)	**0:53:20** 10:20-11:13	Aleksander Steen

03-08	**syslog fixes** H3E	**1:20:26** 11:35-12:55	Aleksander Steen
03-08	**report** H3E	**2:13:02** 12:55-15:08	Aleksander Steen
03-08	**lars pull request** H3E	**0:14:44** 15:08-15:23	Aleksander Steen
03-09	**front end discussion** H3E	**1:14:00** 10:00-11:14	Aleksander Steen
03-09	**moved timelog from latex to toggl** (no project)	**0:15:09** 11:25-11:40	Aleksander Steen
03-09	**course in report writing** H3E	**1:00:00** 12:00-13:00	Aleksander Steen
03-09	**front end discussion** H3E	**0:17:52** 13:05-13:23	Aleksander Steen
03-09	**report** H3E	**1:45:29** 13:30-15:15	Aleksander Steen
03-10	**edit report** H3E	**1:35:35** 09:47-11:22	Aleksander Steen
03-10	**Refactoring Syslog functions** H3E	**1:37:55** 11:45-13:23	Aleksander Steen
03-10	**Refactoring Syslog functions** H3E	**1:56:06** 13:36-15:33	Aleksander Steen
03-11	**olafur pull request|||** H3E	**0:30:00** 09:30-10:00	Aleksander Steen
03-11	**angular** H3E	**1:56:00** 10:01-11:57	Aleksander Steen
03-14	**bottom loader for syslog view** H3E	**1:35:29** 09:23-10:58	Aleksander Steen
03-14	**review olafurs pull request** H3E	**0:17:00** 11:22-11:39	Aleksander Steen
03-14	**angular** H3E	**3:41:00** 11:40-15:21	Aleksander Steen
03-15	**Meeting with supervisor** H3E	**1:02:00** 09:10-10:12	Aleksander Steen
03-15	**report** H3E	**2:16:49** 10:13-12:30	Aleksander Steen
03-15	**angular** H3E	**2:08:12** 12:30-14:38	Aleksander Steen
03-16	**angular** H3E	**2:02:47** 09:33-11:36	Aleksander Steen
03-16	**angular** H3E	**4:07:00** 11:44-15:51	Aleksander Steen
03-17	**angular** H3E	**2:40:38** 09:10-11:50	Aleksander Steen
03-17	**angular** H3E	**3:37:24** 12:17-15:54	Aleksander Steen

03-18	**angular** H3E	**4:05:00** 09:05-13:10	Aleksander Steen
03-29	**Meeting with supervisor** H3E	**1:02:10** 09:15-10:18	Aleksander Steen
03-29	**angular** H3E	**3:05:04** 10:19-13:24	Aleksander Steen
03-30	**report and angular** H3E	**5:58:12** 09:00-14:58	Aleksander Steen
03-31	**tests** H3E	**6:17:24** 09:00-15:18	Aleksander Steen
04-01	**even more tests** H3E	**3:20:53** 09:16-12:37	Aleksander Steen
04-04	**angular** H3E	**6:35:00** 09:15-15:50	Aleksander Steen
04-05	**angular** H3E	**6:37:00** 09:18-15:55	Aleksander Steen
04-06	**angular** H3E	**6:31:00** 09:19-15:50	Aleksander Steen
04-07	**reports view HEEE-219** H3E	**2:39:11** 09:19-11:58	Aleksander Steen
04-07	**(no description)** H3E	**0:41:49** 12:14-12:56	Aleksander Steen
04-07	**angular** H3E	**2:26:38** 12:56-15:23	Aleksander Steen
04-11	**report & infosite** H3E	**3:25:31** 10:04-13:30	Aleksander Steen
04-11	**lars pull request** H3E	**0:07:20** 13:30-13:37	Aleksander Steen
04-11	**report & infosite** H3E	**2:19:00** 13:37-15:56	Aleksander Steen
04-12	**meeting with supervisor** H3E	**0:48:00** 09:15-10:03	Aleksander Steen
04-12	**HEEE-243 + HEEE-228 + HEEE-279** H3E	**5:21:59** 10:04-15:26	Aleksander Steen
04-13	**reports view** H3E	**1:45:00** 09:09-10:54	Aleksander Steen
04-13	**reports view** H3E	**0:33:54** 11:37-12:10	Aleksander Steen
04-13	**reports view tests** H3E	**3:08:23** 12:21-15:30	Aleksander Steen
04-14	**refactoring** H3E	**5:59:00** 09:11-15:10	Aleksander Steen
04-15	**refactoring** H3E	**0:50:00** 09:08-09:58	Aleksander Steen
04-15	**front end** H3E	**2:20:00** 11:30-13:50	Aleksander Steen

04-15	**meeting with eivvind gui** H3E	**1:00:00** 22:00-23:00	Aleksander Steen
04-18	**angulr** H3E	**0:17:43** 09:10-09:27	Aleksander Steen
04-18	**Code Review Olafur's pull request** H3E	**0:17:00** 09:58-10:15	Aleksander Steen
04-18	**angular** H3E	**4:34:00** 10:16-14:50	Aleksander Steen
04-19	**meeting with supervisor** H3E	**0:51:00** 09:10-10:01	Aleksander Steen
04-19	**(no description)** H3E	**2:05:14** 10:02-12:07	Aleksander Steen
04-19	**lars pull request** H3E	**0:01:57** 12:07-12:09	Aleksander Steen
04-19	**Refactoring Syslog functions** H3E	**3:32:38** 12:09-15:42	Aleksander Steen
04-20	**Report** H3E	**2:53:11** 09:39-12:32	Aleksander Steen
04-20	**lars pull request** H3E	**0:26:06** 12:32-12:58	Aleksander Steen
04-20	**fixing syslog tests** H3E	**1:38:24** 13:38-15:17	Aleksander Steen
04-21	**fixing syslog tests** H3E	**0:28:00** 09:17-09:45	Aleksander Steen
04-21	**angular** H3E	**5:30:00** 22:00-03:30	Aleksander Steen
04-26	**syslog refactoring** H3E	**6:30:00** 09:15-15:45	Aleksander Steen
04-27	**Report** H3E	**1:52:00** 13:29-15:21	Aleksander Steen
04-27	**Edit front end design from innit feedback** H3E	**5:10:00** 21:12-02:22	Aleksander Steen
04-28	**Report** H3E	**5:15:00** 21:15-02:30	Aleksander Steen
04-29	**Report** H3E	**5:45:00** 21:15-03:00	Aleksander Steen
05-02	**Report** H3E	**6:08:00** 21:13-03:21	Aleksander Steen
05-03	**Meeting with supervisor** H3E	**0:55:00** 09:15-10:10	Aleksander Steen
05-03	**report** H3E	**0:43:43** 10:20-11:03	Aleksander Steen
05-03	**report** H3E	**2:25:45** 11:29-13:55	Aleksander Steen
05-03	**Code Review Olafur's pull request** H3E	**0:24:56** 13:55-14:20	Aleksander Steen

05-03	**reports view** H3E	**0:50:24** 14:20-15:10	Aleksander Steen
05-04	**reports view** H3E	**6:13:00** 09:25-15:38	Aleksander Steen
05-05	**reports view** H3E	**0:46:20** 10:06-10:52	Aleksander Steen
05-05	**report** H3E	**1:29:12** 11:05-12:35	Aleksander Steen
05-05	**Code Review Olafur's pull request** H3E	**0:15:08** 12:44-12:59	Aleksander Steen
05-05	**Report** H3E	**2:24:23** 13:07-15:31	Aleksander Steen
05-06	**reports_view** H3E	**3:14:47** 09:25-12:40	Aleksander Steen
05-09	**front end** H3E	**5:11:34** 09:00-14:12	Aleksander Steen
05-10	**meeting with supervisor** H3E	**0:55:02** 09:09-10:04	Aleksander Steen
05-10	**angulr** H3E	**3:49:33** 10:05-13:55	Aleksander Steen
05-10	**Code Review Olafur's pull request** H3E	**0:30:23** 13:55-14:25	Aleksander Steen
05-10	**Report** H3E	**0:01:02** 14:26-14:27	Aleksander Steen
05-11	**syslog tests** H3E	**1:22:48** 09:26-10:49	Aleksander Steen
05-11	**proof-reading report** H3E	**3:20:59** 10:50-14:11	Aleksander Steen
05-12	**lars pull request** H3E	**0:25:10** 09:05-09:30	Aleksander Steen
05-12	**Code Review Olafur's pull request** H3E	**0:14:15** 09:31-09:45	Aleksander Steen
05-13	**final meeting with employer** H3E	**1:04:11** 10:00-11:05	Aleksander Steen

H.3 Time Log Lars

Detailed report

2016-01-04 - 2016-05-13

Total 461 h 47 min Billable 00 h 00 min

Date	Description	Duration	User
01-04	**research angular - hottowel** H3E	**6:00:01** 09:00-15:00	Larswalwes
01-05	**john papa best practices** H3E	**6:00:01** 10:00-16:00	Larswalwes
01-06	**frontend skeleton of hottowel** H3E	**6:00:02** 09:00-15:00	Larswalwes
01-07	**laravel setup and testing** H3E	**6:00:03** 09:00-15:00	Larswalwes
01-08	**best practices - git / testing** H3E	**6:00:01** 09:00-15:00	Larswalwes
01-11	**bachelor tips tom røise, laracasts, hottowel** H3E	**6:00:01** 09:00-15:00	Larswalwes
01-12	**discussion, farup, adjusting template** H3E	**6:00:00** 09:00-15:00	Larswalwes
01-13	**meeting innit** H3E	**6:31:05** 09:29-16:00	Larswalwes
01-14	**laracasts, api architecure** H3E	**7:00:00** 09:00-16:00	Larswalwes
01-15	**db structure, gui basic ideas** H3E	**4:00:04** 09:00-13:00	Larswalwes
01-16	**skeleton for front end** H3E	**5:00:01** 11:01-16:01	Larswalwes
01-18	**api architeure ideas, skeleton front end, testing** H3E	**7:00:01** 09:00-16:00	Larswalwes
01-19	**farup, innit skype, front end skeleton** H3E	**6:00:02** 09:00-15:00	Larswalwes
01-20	**rough sketches of gui, gulp config** H3E	**5:00:34** 10:00-15:00	Larswalwes
01-21	**gulp tasks, live reloading etc** H3E	**4:00:01** 13:36-17:36	Larswalwes
01-22	**gulp tasks, angular directives research** H3E	**7:00:01** 09:00-16:00	Larswalwes
01-25	**graphs setup, page demo** H3E	**6:30:00** 09:00-15:30	Larswalwes
01-26	**farup, syslog talk** H3E	**2:00:11** 09:00-11:00	Larswalwes

01-27	**project plan, i18n, remove jquery** H3E	**3:00:01** 09:00-12:00	Larswalwes
01-28	**proofreading project plan** H3E	**1:24:00** 09:22-10:46	Larswalwes
01-28	**set up frontend i18n and move away from jQuery** H3E	**3:20:35** 10:47-14:07	Larswalwes
01-29	**settings module frontend** H3E	**1:42:06** 09:25-11:07	Larswalwes
01-29	**logger module tests and code coverage report** H3E	**2:06:56** 12:01-14:08	Larswalwes
01-29	**smart commits in JIRA** H3E	**0:18:02** 14:38-14:56	Larswalwes
02-01	**factories architecture on frontend discussion** H3E	**2:12:00** 09:24-11:36	Larswalwes
02-01	**heat directive and factory and exception factory and service skeleton** H3E	**3:28:24** 11:57-15:25	Larswalwes
02-02	**heat generation discussion** H3E	**1:45:00** 09:37-11:22	Larswalwes
02-02	**new overview min-info** H3E	**2:19:00** 12:18-14:37	Larswalwes
02-02	**Git commits weren't working** H3E	**0:58:27** 14:37-15:36	Larswalwes
02-03	**skeleton for factories** H3E	**4:23:00** 09:31-13:54	Larswalwes
02-03	**factory tests** H3E	**1:52:58** 13:54-15:47	Larswalwes
02-04	**finalize skeletons for factories/services** H3E	**1:33:03** 09:24-10:57	Larswalwes
02-04	**failed config test attempts** H3E	**3:47:00** 11:20-15:07	Larswalwes
02-05	**factory testing** H3E	**3:42:07** 09:39-13:22	Larswalwes
02-05	**fixing node/phantomjs** H3E	**0:10:54** 13:27-13:38	Larswalwes
02-05	**service tests** H3E	**0:55:00** 13:45-14:40	Larswalwes
02-05	**service tests / IMT3602** H3E	**1:34:33** 14:50-16:24	Larswalwes
02-08	**look over timelogs** H3E	**0:04:47** 09:14-09:18	Larswalwes
02-08	**testing the rest of services** H3E	**1:16:00** 09:19-10:35	Larswalwes
02-08	**try to get node working** H3E	**0:36:00** 11:10-11:46	Larswalwes

02-08	**try to get node working** H3E	**0:25:49** 11:56-12:22	Larswalwes
02-08	**service testing** H3E	**0:47:07** 12:22-13:09	Larswalwes
02-08	**fix setup for module tests** H3E	**1:29:54** 13:17-14:47	Larswalwes
02-08	**routerprovider tests** H3E	**0:13:25** 14:49-15:02	Larswalwes
02-09	**directive tests** H3E	**2:34:18** 09:24-11:58	Larswalwes
02-09	**commenting tests** H3E	**0:30:02** 12:00-12:30	Larswalwes
02-09	**syslog backend handling discussion** H3E	**1:00:01** 13:00-14:00	Larswalwes
02-10	**crash handling for gulp less task** H3E	**0:32:32** 09:12-09:45	Larswalwes
02-10	**try to fix HEEE-6 language default** H3E	**0:11:46** 09:48-10:00	Larswalwes
02-10	**try to fix HEEE-6 language default** H3E	**0:32:55** 10:10-10:42	Larswalwes
02-10	**implement plato / wait for simon** H3E	**0:34:19** 10:51-11:25	Larswalwes
02-10	**HEEE-63 fetch exception types and store in localstorage** H3E	**4:16:05** 11:37-15:53	Larswalwes
02-11	**store exception types HEEE-63** H3E	**2:34:55** 09:30-12:05	Larswalwes
02-11	**HEEE-106 exception batch fetching** H3E	**2:42:52** 12:14-14:56	Larswalwes
02-11	**testing** H3E	**0:40:26** 14:58-15:39	Larswalwes
02-12	**HEEE-115 commenting** H3E	**1:06:16** 09:09-10:15	Larswalwes
02-12	**testing** H3E	**0:50:06** 10:17-11:07	Larswalwes
02-12	**finishing exception controller tests** H3E	**0:40:21** 11:17-11:57	Larswalwes
02-12	**exception factory test** H3E	**1:32:17** 11:59-13:32	Larswalwes
02-12	**(no description)** H3E	**0:28:11** 13:45-14:13	Larswalwes
02-15	**store exceptions in factory** H3E	**0:06:36** 09:16-09:23	Larswalwes
02-15	**fix timestamp directive and test** H3E	**0:41:35** 09:23-10:05	Larswalwes

02-15	**looking at CORS with Olafur** H3E	**1:10:07** 10:05-11:15	Larswalwes
02-15	**change exception to save in factory** H3E	**0:05:44** 11:15-11:20	Larswalwes
02-15	**exception refactoring to factory** H3E	**1:44:52** 12:58-14:43	Larswalwes
02-15	**redoing tests to fit new exception architecture** H3E	**0:47:14** 14:46-15:34	Larswalwes
02-16	**refactor set exception types** H3E	**1:44:00** 09:32-11:16	Larswalwes
02-16	**sort exceptions into today and earlier** H3E	**0:18:46** 11:41-12:00	Larswalwes
02-16	**failed date spit list directive** H3E	**0:34:05** 12:11-12:46	Larswalwes
02-16	**tests** H3E	**0:48:20** 12:49-13:38	Larswalwes
02-16	**util factory tests** H3E	**1:05:54** 13:40-14:46	Larswalwes
02-17	**exception types race condition** H3E	**0:43:01** 09:20-10:03	Larswalwes
02-17	**more heat discussion** H3E	**1:03:13** 10:21-11:24	Larswalwes
02-17	**redo fetching of exceptions** H3E	**2:26:46** 11:33-14:00	Larswalwes
02-17	**HEEE-143** H3E	**0:11:35** 14:04-14:15	Larswalwes
02-17	**routing in app view** H3E	**1:40:11** 14:24-16:04	Larswalwes
02-18	**continued routing in app view** H3E	**0:53:15** 09:45-10:38	Larswalwes
02-18	**HEEE-59 cleaning up dependencies for modules** H3E	**0:01:31** 10:49-10:51	Larswalwes
02-18	**(no description)** H3E	**0:37:46** 11:05-11:43	Larswalwes
02-18	**missing tests** H3E	**1:32:33** 13:38-15:11	Larswalwes
02-19	**application partials testing** H3E	**0:22:31** 11:32-11:54	Larswalwes
02-19	**diagramming** H3E	**1:37:07** 12:23-14:00	Larswalwes
02-19	**imt3602** H3E	**2:02:23** 14:00-16:03	Larswalwes
02-22	**HEEE-156 exception details pop up** H3E	**0:09:47** 09:14-09:24	Larswalwes

02-22	**HEEE-175 earlier excp if none today** H3E	**1:11:46** 09:27-10:39	Larswalwes
02-22	**HEEE-156 exception details pop up** H3E	**1:21:48** 10:54-12:16	Larswalwes
02-22	**HEEE-156 tests** H3E	**0:43:14** 12:16-12:59	Larswalwes
02-22	**179 h3e label to directives** H3E	**0:12:02** 13:20-13:32	Larswalwes
02-22	**171 loading spinner** H3E	**2:24:07** 13:35-15:59	Larswalwes
02-23	**ivar meeting** H3E	**0:40:09** 09:15-09:55	Larswalwes
02-23	**170 bottomloader refactor** H3E	**0:49:47** 10:25-11:15	Larswalwes
02-23	**generate dummy data function** H3E	**0:20:02** 11:30-11:50	Larswalwes
02-23	**180 continious excp checking** H3E	**0:55:11** 15:45-16:40	Larswalwes
02-24	**180 exception continious checking tests** H3E	**3:18:00** 09:30-12:48	Larswalwes
02-24	**146 log error on navigation error** H3E	**0:32:01** 13:12-13:44	Larswalwes
02-24	**186 grapher** H3E	**1:28:00** 14:00-15:28	Larswalwes
02-25	**186 grapher** H3E	**2:32:09** 09:17-11:49	Larswalwes
02-26	**186 grapher** H3E	**2:53:32** 10:41-13:35	Larswalwes
02-29	**186 grapher** H3E	**7:17:22** 09:15-16:32	Larswalwes
03-01	**(no description)** H3E	**3:07:27** 10:20-13:27	Larswalwes
03-02	**186 grapher testing** H3E	**2:32:36** 09:33-12:05	Larswalwes
03-02	**186 readjust grapher for days with no exceptions** H3E	**1:04:10** 12:07-13:11	Larswalwes
03-02	**186 application graphs** H3E	**0:52:01** 15:08-16:00	Larswalwes
03-03	**186 application graphs** H3E	**3:55:00** 10:20-14:15	Larswalwes
03-04	**186 grapher settings module** H3E	**2:03:57** 09:13-11:17	Larswalwes
03-04	**186 grapher settings module** H3E	**3:16:12** 11:37-14:54	Larswalwes

03-07	**innit meeting** H3E	**4:30:01** 11:00-15:30	Larswalwes
03-08	**meeting with farup** H3E	**0:55:02** 09:10-10:05	Larswalwes
03-08	**architecture discussion** H3E	**1:00:01** 10:10-11:10	Larswalwes
03-08	**looking at pull request heee-169** H3E	**0:11:49** 11:28-11:40	Larswalwes
03-08	**heee-186 rebasing** H3E	**1:20:02** 11:40-13:00	Larswalwes
03-08	**186 fixing tests** H3E	**1:11:47** 13:08-14:20	Larswalwes
03-09	**settings and severity discussion** H3E	**1:10:01** 09:32-10:42	Larswalwes
03-09	**new settings view** H3E	**2:36:28** 10:50-13:26	Larswalwes
03-09	**send mass severity update** H3E	**2:01:06** 13:20-15:21	Larswalwes
03-10	**settings test** H3E	**0:28:22** 09:35-10:03	Larswalwes
03-10	**Refactor grapher functions** H3E	**1:10:46** 10:28-11:39	Larswalwes
03-10	**Refactor grapher functions** H3E	**4:07:14** 11:58-16:05	Larswalwes
03-11	**Refactor grapher functions** H3E	**4:04:26** 09:30-13:35	Larswalwes
03-14	**Refactor grapher functions** H3E	**3:00:10** 09:30-12:31	Larswalwes
03-14	**Refactor grapher functions** H3E	**3:05:36** 12:33-15:39	Larswalwes
03-15	**farup meeting** H3E	**1:00:01** 09:15-10:15	Larswalwes
03-15	**Refactor grapher functions** H3E	**4:58:28** 10:46-15:44	Larswalwes
03-16	**Refactor grapher functions** H3E	**2:39:54** 09:10-11:50	Larswalwes
03-16	**Refactor grapher functions** H3E	**1:47:00** 13:33-15:20	Larswalwes
03-17	**Refactor grapher functions** H3E	**6:23:13** 09:33-15:56	Larswalwes
03-18	**Refactor grapher tests** H3E	**1:03:09** 09:02-10:05	Larswalwes
03-18	**send mass severity update** H3E	**1:02:37** 10:15-11:17	Larswalwes

03-18	**236 check all in severity updater** H3E	**0:36:54** 11:52-12:29	Larswalwes
03-29	**236 check all in severity updater** H3E	**1:27:57** 10:18-11:46	Larswalwes
03-29	**discussion on exception contexts** H3E	**0:40:01** 11:46-12:26	Larswalwes
03-29	**Remove types from localstorage** H3E	**3:14:16** 12:29-15:44	Larswalwes
03-30	**Remove types from localstorage** H3E	**0:13:32** 00:12-00:25	Larswalwes
03-30	**contexts update form** H3E	**1:51:11** 09:15-11:06	Larswalwes
03-30	**contexts update form** H3E	**3:01:35** 20:39-23:41	Larswalwes
03-31	**Refactor grapher with vegard** H3E	**1:26:13** 09:30-10:56	Larswalwes
03-31	**contexts update form** H3E	**1:09:58** 10:56-12:06	Larswalwes
03-31	**contexts update form** H3E	**1:02:09** 12:16-13:18	Larswalwes
03-31	**contxts update form** H3E	**0:31:34** 13:18-13:50	Larswalwes
03-31	**ticketing system for loading** H3E	**1:35:15** 14:05-15:40	Larswalwes
04-01	**ticketing system for loading** H3E	**0:07:06** 09:18-09:25	Larswalwes
04-01	**Redo and rework test coverage** H3E	**3:10:36** 09:27-12:38	Larswalwes
04-01	**261 - Exception factory tests** H3E	**2:15:29** 12:41-14:56	Larswalwes
04-04	**261 - Exception factory tests** H3E	**1:27:13** 09:10-10:37	Larswalwes
04-04	**217 grapher setting tests** H3E	**0:12:00** 10:56-11:08	Larswalwes
04-04	**refactoring grapher with vegard** H3E	**0:27:02** 11:08-11:35	Larswalwes
04-04	**217 grapher setting tests** H3E	**1:14:37** 11:35-12:50	Larswalwes
04-04	**look at syslog pull request** H3E	**1:11:53** 12:50-14:02	Larswalwes
04-04	**217 grapher setting tests** H3E	**1:11:44** 14:07-15:18	Larswalwes
04-05	**polishing grapher, contex updating and new exception scanning** H3E	**3:49:38** 09:15-13:04	Larswalwes

04-05	last missing case in grapher controller H3E	2:05:10 13:05-15:10	Larswalwes
04-06	last missing case in grapher controller H3E	1:58:00 09:05-11:03	Larswalwes
04-06	last missing case in grapher controller H3E	3:02:54 11:03-14:06	Larswalwes
04-06	last missing case in grapher controller H3E	1:14:52 14:36-15:51	Larswalwes
04-07	remaining statistics tests H3E	0:28:48 09:30-09:59	Larswalwes
04-07	polishing exception details and etc H3E	3:47:23 10:00-13:48	Larswalwes
04-08	ask backend for severity changes H3E	2:04:22 13:06-15:10	Larswalwes
04-11	Ask for severity changes and refresh popup H3E	2:09:30 09:32-11:42	Larswalwes
04-11	severity change checker tests H3E	1:54:59 11:42-13:37	Larswalwes
04-12	farup meeting H3E	1:00:08 09:12-10:12	Larswalwes
04-12	search gui H3E	1:29:34 10:12-11:42	Larswalwes
04-12	conitnious fetching of current heat H3E	0:54:48 12:16-13:11	Larswalwes
04-12	fix display of exceptions app name H3E	2:15:51 13:11-15:26	Larswalwes
04-13	write about issues had H3E	0:38:27 09:11-09:50	Larswalwes
04-13	more tests and highlight tabs in exception page properly H3E	1:34:23 09:58-11:32	Larswalwes
04-13	advanced search discussion H3E	1:20:19 11:45-13:05	Larswalwes
04-13	change today to 24h H3E	0:54:12 13:05-14:00	Larswalwes
04-13	change today graph to last 24h H3E	0:59:56 14:10-15:10	Larswalwes
04-14	change today graph to last 24h H3E	0:43:32 09:16-09:59	Larswalwes
04-14	accenture H3E	3:12:04 09:59-13:11	Larswalwes
04-14	change today graph to last 24h H3E	0:07:45 13:11-13:19	Larswalwes
04-14	change today graph to last 24h H3E	1:35:37 13:25-15:01	Larswalwes

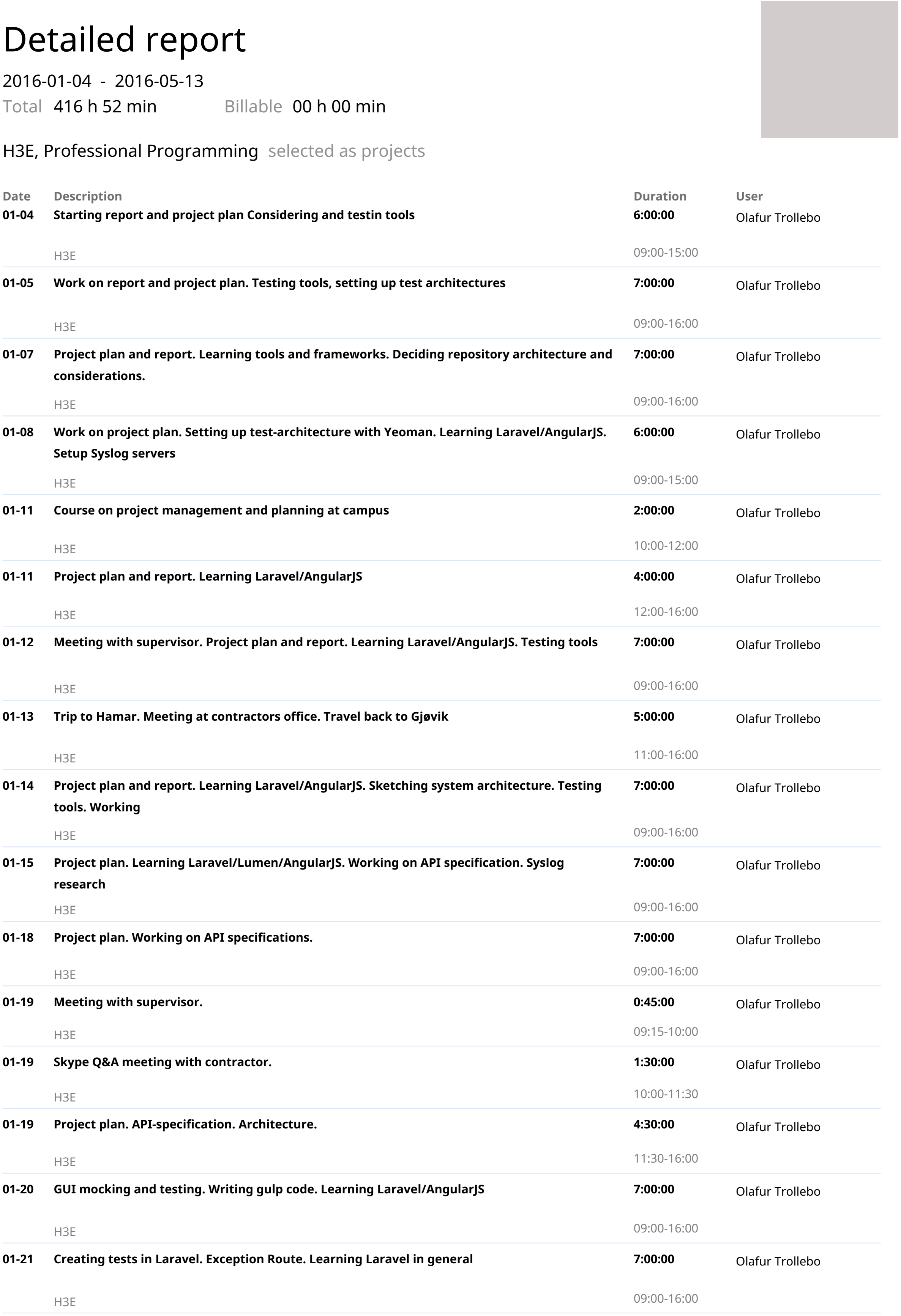
04-14	**dynamic graph settings** H3E	**0:53:27** 15:01-15:55	Larswalwes
04-15	**search filters** H3E	**1:00:28** 09:07-10:07	Larswalwes
04-15	**meeting with design guru** H3E	**1:07:09** 10:07-11:15	Larswalwes
04-15	**search filters and a little design redoing** H3E	**1:33:10** 11:35-13:08	Larswalwes
04-15	**appication overview** H3E	**0:37:09** 13:26-14:04	Larswalwes
04-18	**appication overview** H3E	**2:14:00** 09:09-11:23	Larswalwes
04-18	**testing** H3E	**1:14:44** 11:23-12:38	Larswalwes
04-18	**reading document** H3E	**2:59:27** 22:30-01:30	Larswalwes
04-19	**farup meeting** H3E	**1:04:13** 09:10-10:14	Larswalwes
04-19	**testing** H3E	**1:40:13** 10:19-11:59	Larswalwes
04-19	**rolling average discussion** H3E	**3:40:46** 12:00-15:41	Larswalwes
04-20	**testing** H3E	**1:32:31** 09:29-11:02	Larswalwes
04-20	**Adjust heat from average value** H3E	**0:50:16** 11:02-11:53	Larswalwes
04-20	**testing** H3E	**0:32:46** 11:55-12:28	Larswalwes
04-20	**Heat adjustments to average** H3E	**1:08:26** 12:56-14:04	Larswalwes
04-20	**Display heat in graph** H3E	**1:11:12** 14:05-15:16	Larswalwes
04-21	**Display heat in graph** H3E	**5:27:58** 09:30-14:57	Larswalwes
04-22	**Display heat in graph** H3E	**4:23:50** 09:35-13:59	Larswalwes
04-23	**Display heat in graph** H3E	**2:02:21** 01:30-03:32	Larswalwes
04-25	**Various and pull requests** H3E	**3:12:12** 09:13-12:25	Larswalwes
04-26	**general clean up and address change in api** H3E	**2:55:10** 09:15-12:10	Larswalwes
04-26	**innit meeting** H3E	**3:45:20** 12:15-16:00	Larswalwes

04-27	**add context data to exceptions** H3E	**1:11:33** 09:10-10:21	Larswalwes
04-27	**exception list directive** H3E	**2:45:38** 10:21-13:07	Larswalwes
04-27	**Adjust frontend to handle mode filters** H3E	**2:13:29** 13:15-15:28	Larswalwes
04-28	**make graph format for olafur** H3E	**0:19:23** 09:14-09:34	Larswalwes
04-28	**test fixing** H3E	**1:59:03** 09:35-11:34	Larswalwes
04-28	**gulp build** H3E	**3:27:04** 11:35-15:02	Larswalwes
05-02	**gulp build** H3E	**0:30:55** 09:15-09:46	Larswalwes
05-02	**gulp build testing and fixing** H3E	**1:09:38** 09:50-11:00	Larswalwes
05-02	**set data to exception error handling** H3E	**0:21:08** 11:00-11:21	Larswalwes
05-02	**HEEE-333 same exceptions list** H3E	**1:32:47** 11:25-12:58	Larswalwes
05-02	**Write about mocking and grapher** H3E	**0:54:00** 12:58-13:52	Larswalwes
05-02	**rewrite grapher** H3E	**1:05:02** 13:55-15:00	Larswalwes
05-03	**farup meeting** H3E	**0:59:39** 09:10-10:10	Larswalwes
05-03	**rewrite grapher** H3E	**3:19:17** 10:11-13:30	Larswalwes
05-03	**rewrite grapher** H3E	**0:29:06** 13:35-14:04	Larswalwes
05-03	**pull request and discussion** H3E	**0:38:12** 14:05-14:43	Larswalwes
05-03	**minor fixes** H3E	**0:12:48** 14:44-14:56	Larswalwes
05-03	**HEEE-341 heat value appearance** H3E	**0:05:29** 14:57-15:02	Larswalwes
05-04	**grapher color picker** H3E	**3:00:35** 09:30-12:31	Larswalwes
05-04	**various discussion** H3E	**1:10:03** 12:33-13:43	Larswalwes
05-04	**handle unseen IDs in setdatatoexp** H3E	**1:42:28** 13:42-15:24	Larswalwes
05-04	**handle unseen IDs in setdatatoexp** H3E	**1:31:47** 15:51-17:23	Larswalwes

05-05	**Handle 0 apps and 0 types graph** H3E	**1:05:34** 09:58-11:03	Larswalwes
05-05	**handle new apps and types in heat graph** H3E	**0:52:40** 11:03-11:56	Larswalwes
05-05	**Refactor syslog view** H3E	**3:26:47** 12:06-15:33	Larswalwes
05-06	**Refactor syslog view** H3E	**1:48:01** 09:15-11:03	Larswalwes
05-06	**pull request** H3E	**0:02:46** 11:04-11:06	Larswalwes
05-06	**Refactor syslog view** H3E	**2:58:05** 11:06-14:04	Larswalwes
05-07	**Refactor syslog view** H3E	**2:16:42** 23:26-01:42	Larswalwes
05-08	**fix stupid unignore syslog hosts function** H3E	**0:06:25** 14:03-14:10	Larswalwes
05-08	**adjust grapher to new index format** H3E	**0:18:02** 14:27-14:45	Larswalwes
05-09	**code example for report** H3E	**0:24:08** 09:03-09:28	Larswalwes
05-09	**refactor reports view** H3E	**1:55:00** 09:30-11:25	Larswalwes
05-09	**refactor reports view** H3E	**1:37:50** 19:28-21:06	Larswalwes
05-10	**refactor reports view** H3E	**0:33:55** 10:10-10:44	Larswalwes
05-10	**refactor reports view** H3E	**2:56:57** 11:00-13:57	Larswalwes
05-10	**refactor reports view** H3E	**1:19:16** 14:00-15:19	Larswalwes
05-10	**make favicon** H3E	**0:45:07** 21:45-22:30	Larswalwes
05-11	**single year reports view** H3E	**3:47:15** 09:16-13:04	Larswalwes
05-11	**single year reports view** H3E	**0:04:07** 13:17-13:21	Larswalwes
05-11	**max height directive** H3E	**0:35:13** 13:21-13:56	Larswalwes
05-12	**fixing up log** H3E	**1:08:27** 09:06-10:15	Larswalwes
05-12	**various bug fixes** H3E	**1:07:17** 10:15-11:22	Larswalwes
05-12	**Making tests** H3E	**2:39:01** 11:22-14:01	Larswalwes

05-12	**clean up logs** H3E	**0:33:14** 14:01-14:34	Larswalwes
05-13	**read through and edit paper** H3E	**0:40:10** 09:23-10:04	Larswalwes
05-13	**innit meeting** H3E	**0:52:18** 10:04-10:56	Larswalwes
05-13	**css fixing after meeting** H3E	**0:34:40** 10:58-11:32	Larswalwes

H.4 Time Log Olafur



Detailed report

2016-01-04 - 2016-05-13

Total 416 h 52 min Billable 00 h 00 min

H3E, Professional Programming selected as projects

Date	Description	Duration	User
01-04	**Starting report and project plan Considering and testin tools** H3E	**6:00:00** 09:00-15:00	Olafur Trollebo
01-05	**Work on report and project plan. Testing tools, setting up test architectures** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo
01-07	**Project plan and report. Learning tools and frameworks. Deciding repository architecture and considerations.** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo
01-08	**Work on project plan. Setting up test-architecture with Yeoman. Learning Laravel/AngularJS. Setup Syslog servers** H3E	**6:00:00** 09:00-15:00	Olafur Trollebo
01-11	**Course on project management and planning at campus** H3E	**2:00:00** 10:00-12:00	Olafur Trollebo
01-11	**Project plan and report. Learning Laravel/AngularJS** H3E	**4:00:00** 12:00-16:00	Olafur Trollebo
01-12	**Meeting with supervisor. Project plan and report. Learning Laravel/AngularJS. Testing tools** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo
01-13	**Trip to Hamar. Meeting at contractors office. Travel back to Gjøvik** H3E	**5:00:00** 11:00-16:00	Olafur Trollebo
01-14	**Project plan and report. Learning Laravel/AngularJS. Sketching system architecture. Testing tools. Working** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo
01-15	**Project plan. Learning Laravel/Lumen/AngularJS. Working on API specification. Syslog research** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo
01-18	**Project plan. Working on API specifications.** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo
01-19	**Meeting with supervisor.** H3E	**0:45:00** 09:15-10:00	Olafur Trollebo
01-19	**Skype Q&A meeting with contractor.** H3E	**1:30:00** 10:00-11:30	Olafur Trollebo
01-19	**Project plan. API-specification. Architecture.** H3E	**4:30:00** 11:30-16:00	Olafur Trollebo
01-20	**GUI mocking and testing. Writing gulp code. Learning Laravel/AngularJS** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo
01-21	**Creating tests in Laravel. Exception Route. Learning Laravel in general** H3E	**7:00:00** 09:00-16:00	Olafur Trollebo

01-22	**Tried fixing validation** H3E	**2:00:00** 09:00-11:00	Olafur Trollebo
01-22	**Exception route** H3E	**1:00:00** 12:00-13:00	Olafur Trollebo
01-22	**IMT3602** Professional Programming	**1:00:00** 14:00-15:00	Olafur Trollebo
01-25	**Exception Route - Backend** H3E	**0:58:29** 09:41-10:40	Olafur Trollebo
01-25	**Exception Route - Backend** H3E	**0:42:18** 11:28-12:10	Olafur Trollebo
01-25	**Exception Route - Backend** H3E	**1:04:11** 12:21-13:25	Olafur Trollebo
01-25	**Moving backend project from Github to BitBucket** H3E	**2:41:00** 14:19-17:00	Olafur Trollebo
01-28	**Learning about best practices for Seeders in Laravel** Professional Programming	**0:45:00** 09:32-10:17	Olafur Trollebo
01-28	**Migration/Seeder on Application Route - UPDATE: Had to do error fixing due to full path** Professional Programming	**0:41:00** 10:17-10:58	Olafur Trollebo
01-28	**Application Route Controller** H3E	**1:49:42** 11:01-12:51	Olafur Trollebo
01-28	**Application Route Tests** Professional Programming	**1:07:00** 13:03-14:10	Olafur Trollebo
01-29	**Setup for Migrations/Seeders on Exception Route** Professional Programming	**0:46:00** 09:50-10:36	Olafur Trollebo
01-29	**Bugfixes on the backend** H3E	**0:29:00** 10:36-11:05	Olafur Trollebo
01-29	**Talking to Simon about Professional Programming** Professional Programming	**0:58:00** 11:06-12:04	Olafur Trollebo
01-29	**Migration on Exception Route** H3E	**0:10:05** 12:34-12:44	Olafur Trollebo
01-29	**Creating Models for the Exception Route** H3E	**0:05:34** 12:47-12:53	Olafur Trollebo
01-29	**Research Models** Professional Programming	**0:06:35** 12:53-12:59	Olafur Trollebo
01-29	**Creating Models for the Exception Route** H3E	**0:08:06** 12:59-13:07	Olafur Trollebo
01-29	**Creating Seeders on Exception Route** Professional Programming	**1:18:29** 13:08-14:26	Olafur Trollebo
01-29	**Professional Programming Lecture** Professional Programming	**0:30:00** 14:26-14:56	Olafur Trollebo
01-29	**Creating Seeders on Exception Route** Professional Programming	**0:52:17** 15:39-16:31	Olafur Trollebo

01-30	**Creating Controller and input route.** H3E	**4:00:29** 19:11-23:11	Olafur Trollebo
01-30	**Creating the "show" route of Exceptions** H3E	**0:24:41** 23:11-23:36	Olafur Trollebo
01-30	**Bugfixes related to DB, seeds and factories** H3E	**0:51:12** 23:37-00:28	Olafur Trollebo
01-31	**Bugfixes on the backend - DB structure** H3E	**0:21:44** 00:28-00:50	Olafur Trollebo
02-01	**Group Meeting - Front and backend** H3E - [planning]	**1:55:00** 09:05-11:00	Olafur Trollebo
02-01	**Creating the "show" route of Exceptions** H3E - [backend]	**2:24:00** 11:10-13:34	Olafur Trollebo
02-01	**Bugfix - HEEE-12.** H3E - [backend, bugfix]	**2:20:00** 13:34-15:54	Olafur Trollebo
02-01	**Improved validation rules on incoming exceptions and some code cleanup** Professional Programming - [backend]	**0:12:00** 18:30-18:42	Olafur Trollebo
02-01	**Deploying Backend** H3E - [backend]	**3:21:00** 19:04-22:25	Olafur Trollebo
02-02	**Fixing backend that Aleksander broke** H3E - [backend, bugfix]	**0:17:23** 09:36-09:54	Olafur Trollebo
02-02	**Sending mail to Innit** H3E	**0:17:06** 12:09-12:26	Olafur Trollebo
02-02	**Creating issues in JIRA** Professional Programming	**1:31:53** 12:37-14:09	Olafur Trollebo
02-02	**HEEE-40** Professional Programming - [backend]	**0:37:14** 14:09-14:47	Olafur Trollebo
02-03	**Creating issues in JIRA** Professional Programming	**0:35:44** 09:41-10:17	Olafur Trollebo
02-03	**Creating issues in JIRA** Professional Programming	**0:07:03** 10:21-10:28	Olafur Trollebo
02-03	**Create Statistics route** H3E	**1:03:18** 10:50-11:53	Olafur Trollebo
02-03	**HEEE-32** H3E - [backend]	**1:55:19** 12:01-13:57	Olafur Trollebo
02-03	**HEEE-33** H3E - [backend]	**0:42:06** 14:22-15:04	Olafur Trollebo
02-03	**HEEE-34** H3E - [backend]	**0:19:28** 15:11-15:30	Olafur Trollebo
02-03	**HEEE-35** H3E - [backend]	**0:08:36** 15:30-15:39	Olafur Trollebo
02-03	**HEEE-68** Professional Programming - [backend, testing]	**0:14:10** 15:42-15:56	Olafur Trollebo

02-04	**HEEE-68** Professional Programming - [backend, testing]	**0:44:36** 09:25-10:10	Olafur Trollebo
02-04	**HEEE-68** Professional Programming - [backend, testing]	**0:02:17** 10:30-10:32	Olafur Trollebo
02-04	**Fixing backend that Aleksander broke** H3E	**0:39:00** 10:32-11:11	Olafur Trollebo
02-04	**Commenting the Statistics tests** H3E	**1:05:11** 11:58-13:04	Olafur Trollebo
02-04	**Reviewing pull request with Vegard** Professional Programming	**0:08:04** 13:12-13:20	Olafur Trollebo
02-04	**HEEE-78** H3E - [backend, planning]	**0:30:44** 13:20-13:51	Olafur Trollebo
02-04	**Learning more laravel** H3E	**0:06:06** 14:12-14:18	Olafur Trollebo
02-04	**HEEE-69** Professional Programming	**1:31:53** 14:22-15:54	Olafur Trollebo
02-04	**HEEE-81 (Blocking HEEE-69 temporary)** H3E - [backend, refactor]	**1:05:47** 17:17-18:23	Olafur Trollebo
02-04	**HEEE-69** Professional Programming	**0:13:41** 18:23-18:36	Olafur Trollebo
02-04	**HEEE-69** Professional Programming	**1:07:50** 19:22-20:30	Olafur Trollebo
02-05	**HEEE-78** H3E - [backend]	**1:00:00** 09:39-10:39	Olafur Trollebo
02-05	**HEEE-53** H3E	**0:07:47** 11:45-11:53	Olafur Trollebo
02-05	**HEEE-69** Professional Programming	**1:01:20** 11:53-12:54	Olafur Trollebo
02-05	**HEEE-69** Professional Programming	**0:13:18** 13:13-13:26	Olafur Trollebo
02-05	**Bugfixing** H3E	**0:38:11** 14:03-14:42	Olafur Trollebo
02-05	**HEEE-53** H3E	**0:01:15** 14:46-14:47	Olafur Trollebo
02-05	**Server went boom. Firefighting ahoy!!** H3E	**0:05:23** 14:48-14:53	Olafur Trollebo
02-05	**HEEE-86** H3E	**0:36:11** 14:53-15:30	Olafur Trollebo
02-05	**Helping Lars make tests** H3E	**0:41:24** 15:30-16:12	Olafur Trollebo
02-08	**Meeting with Farup** H3E - [meeting]	**0:56:05** 10:05-11:01	Olafur Trollebo
02-08	**Researching how to do HEEE-89** Professional Programming - [backend]	**0:30:48** 11:36-12:07	Olafur Trollebo

02-08	**HEEE-89** H3E - [backend]	**2:55:35** 12:07-15:03	Olafur Trollebo
02-09	**HEEE-89** H3E - [backend]	**0:24:55** 09:28-09:53	Olafur Trollebo
02-09	**HEEE-95** H3E	**0:12:32** 10:18-10:30	Olafur Trollebo
02-09	**HEEE-95** H3E	**0:29:43** 10:49-11:19	Olafur Trollebo
02-09	**HEEE-95** H3E	**2:00:55** 11:57-13:58	Olafur Trollebo
02-09	**HEEE-95** H3E	**0:42:39** 15:01-15:44	Olafur Trollebo
02-10	**HEEE-95** H3E	**1:48:41** 09:22-11:11	Olafur Trollebo
02-10	**HEEE-95** H3E	**0:29:06** 11:42-12:11	Olafur Trollebo
02-10	**Fixing HEEE-95 for Pull Request** Professional Programming - [backend]	**1:50:54** 13:00-14:51	Olafur Trollebo
02-11	**HEEE-103** H3E - [backend]	**0:34:35** 09:26-10:01	Olafur Trollebo
02-11	**HEEE-101** Professional Programming - [backend, refactor]	**0:42:22** 10:24-11:06	Olafur Trollebo
02-11	**HEEE-105** H3E	**0:10:24** 11:06-11:17	Olafur Trollebo
02-11	**HEEE-101** Professional Programming - [backend, refactor]	**0:23:49** 11:17-11:40	Olafur Trollebo
02-11	**HEEE-101** Professional Programming - [backend, refactor]	**0:56:10** 12:06-13:02	Olafur Trollebo
02-11	**HEEE-36** H3E	**0:19:21** 13:25-13:45	Olafur Trollebo
02-11	**HEEE-62** H3E - [frontend]	**1:47:18** 13:51-15:38	Olafur Trollebo
02-12	**Code review** Professional Programming	**1:59:00** 14:06-16:05	Olafur Trollebo
02-13	**HEEE-62** H3E - [frontend]	**3:58:03** 21:00-00:59	Olafur Trollebo
02-15	**HEEE-53** H3E	**0:10:34** 09:42-09:52	Olafur Trollebo
02-15	**Firefighting - CORS** H3E	**1:00:00** 09:52-10:52	Olafur Trollebo
02-15	**HEEE-121** H3E	**0:02:16** 12:09-12:11	Olafur Trollebo
02-15	**Misc work backend work** H3E	**0:49:44** 12:11-13:01	Olafur Trollebo

02-15	**HEEE-128** H3E	**1:19:43** 13:01-14:20	Olafur Trollebo
02-15	**Codereview - Pull Request** Professional Programming	**0:41:30** 14:20-15:01	Olafur Trollebo
02-16	**Meeting Ivar Farup** H3E	**1:42:00** 09:05-10:47	Olafur Trollebo
02-16	**HEEE-131** H3E	**1:12:41** 10:47-12:00	Olafur Trollebo
02-16	**HEEE-131** H3E	**0:26:01** 12:11-12:37	Olafur Trollebo
02-16	**HEEE-134** H3E	**1:01:44** 12:43-13:44	Olafur Trollebo
02-16	**HEEE-134** H3E	**0:12:33** 14:45-14:58	Olafur Trollebo
02-17	**Misc discussions - Code Review** H3E	**1:18:00** 09:02-10:20	Olafur Trollebo
02-17	**Heat discussion regarding frontend** H3E - [frontend]	**1:04:00** 10:20-11:24	Olafur Trollebo
02-17	**HEEE-138** H3E	**1:59:36** 11:41-13:41	Olafur Trollebo
02-17	**HEEE-144** H3E - [backend]	**1:07:04** 14:12-15:19	Olafur Trollebo
02-18	**HEEE-144** H3E - [backend]	**0:05:42** 09:30-09:36	Olafur Trollebo
02-18	**HEEE-148** H3E	**0:13:46** 10:13-10:27	Olafur Trollebo
02-18	**Researching how to do HEEE-142** Professional Programming	**0:40:09** 10:33-11:13	Olafur Trollebo
02-18	**Application view discussion** H3E	**0:17:56** 11:13-11:31	Olafur Trollebo
02-18	**HEEE-142** H3E	**0:32:16** 11:48-12:20	Olafur Trollebo
02-18	**HEEE-142** H3E	**0:32:45** 12:38-13:11	Olafur Trollebo
02-18	**Reopened and working on HEEE-142** H3E	**0:09:39** 13:42-13:52	Olafur Trollebo
02-18	**Reopened and working on HEEE-142 and HEEE-162** H3E	**0:51:37** 14:14-15:06	Olafur Trollebo
02-19	**Figuring out what to do - Creating issues** Professional Programming	**0:44:29** 11:11-11:55	Olafur Trollebo
02-19	**HEEE-163** H3E	**0:01:10** 11:56-11:57	Olafur Trollebo
02-19	**Code review - Pull Request** Professional Programming	**0:23:27** 11:57-12:20	Olafur Trollebo

02-19	**HEEE-163** H3E	**0:10:14** 12:38-12:48	Olafur Trollebo
02-19	**HEEE-163 - Subtask 165** H3E	**0:13:20** 12:48-13:02	Olafur Trollebo
02-19	**HEEE-172** H3E	**0:11:01** 13:03-13:14	Olafur Trollebo
02-19	**HEEE-167** H3E	**0:21:59** 13:47-14:09	Olafur Trollebo
02-19	**Listening to Simon and replying** Professional Programming	**1:07:53** 14:11-15:19	Olafur Trollebo
02-19	**HEEE-167** H3E	**0:46:59** 15:19-16:06	Olafur Trollebo
02-19	**HEEE-167** H3E	**0:18:09** 23:28-23:46	Olafur Trollebo
02-19	**HEEE-167** H3E	**0:10:11** 23:48-23:58	Olafur Trollebo
02-22	**Pull Request - HEEE-167** H3E	**0:12:41** 10:57-11:10	Olafur Trollebo
02-22	**HEEE-177** H3E	**0:50:53** 11:10-12:01	Olafur Trollebo
02-22	**HEEE-178** H3E	**1:34:01** 14:18-15:52	Olafur Trollebo
02-23	**Meeting Ivar Farup** H3E	**0:57:00** 09:00-09:57	Olafur Trollebo
02-23	**Code review - Pull Request** Professional Programming	**0:21:31** 09:57-10:18	Olafur Trollebo
02-23	**HEEE-178** H3E	**0:41:00** 10:18-10:59	Olafur Trollebo
02-23	**HEEE-178** H3E	**0:36:50** 11:37-12:14	Olafur Trollebo
02-23	**HEEE-178** H3E	**0:05:21** 12:14-12:19	Olafur Trollebo
02-23	**HEEE-178** H3E	**0:17:19** 12:19-12:37	Olafur Trollebo
02-23	**HEEE-178** H3E	**0:20:00** 12:37-12:57	Olafur Trollebo
02-24	**Frontend work** H3E	**1:00:04** 01:09-02:09	Olafur Trollebo
02-24	**HEEE-178** H3E	**0:20:06** 09:17-09:37	Olafur Trollebo
02-24	**HEEE-178** H3E	**0:33:08** 09:43-10:16	Olafur Trollebo
02-24	**HEEE-169** H3E - [frontend]	**0:34:20** 10:19-10:54	Olafur Trollebo
02-24	**HEEE-183** H3E	**0:12:14** 10:57-11:09	Olafur Trollebo

02-24	**HEEE-169** H3E - [frontend]	**0:04:54** 11:10-11:15	Olafur Trollebo
02-24	**HEEE-169** H3E - [frontend]	**0:57:52** 11:36-12:34	Olafur Trollebo
02-24	**Code review - Pull Request** Professional Programming	**0:11:15** 12:34-12:46	Olafur Trollebo
02-24	**HEEE-169** H3E - [frontend]	**0:35:16** 12:46-13:21	Olafur Trollebo
02-24	**HEEE-169** H3E - [frontend]	**0:56:11** 13:34-14:30	Olafur Trollebo
02-24	**HEEE-169** H3E	**0:26:56** 14:30-14:57	Olafur Trollebo
02-24	**HEEE-188** H3E	**0:38:38** 21:53-22:31	Olafur Trollebo
02-25	**HEEE-169** H3E	**0:39:23** 09:20-10:00	Olafur Trollebo
02-25	**Learning more about AngularJS testing** Professional Programming	**0:28:03** 10:00-10:28	Olafur Trollebo
02-25	**Creating issue based on mail from Innit** H3E	**0:04:11** 10:54-10:58	Olafur Trollebo
02-25	**HEEE-190** H3E - [backend]	**0:50:03** 10:58-11:49	Olafur Trollebo
02-25	**HEEE-190** H3E - [backend]	**0:16:24** 17:46-18:03	Olafur Trollebo
02-29	**Finding stuff to do** H3E	**0:07:18** 09:13-09:21	Olafur Trollebo
02-29	**Figuring out how to do HEEE-192** Professional Programming	**0:45:18** 09:21-10:06	Olafur Trollebo
02-29	**HEEE-192** H3E	**0:42:43** 10:06-10:49	Olafur Trollebo
02-29	**Firefighting - SQL exceptions in the backend** H3E	**0:25:30** 10:49-11:14	Olafur Trollebo
02-29	**HEEE-192** H3E	**0:03:01** 11:14-11:18	Olafur Trollebo
02-29	**HEEE-192** H3E	**0:38:20** 11:36-12:14	Olafur Trollebo
02-29	**HEEE-169** H3E	**0:03:24** 12:43-12:46	Olafur Trollebo
02-29	**HEEE-169** H3E	**0:57:39** 13:23-14:21	Olafur Trollebo
02-29	**Figuring out if statistics can output empty dates** H3E	**0:18:32** 14:21-14:39	Olafur Trollebo
02-29	**HEEE-169** H3E	**0:19:04** 14:42-15:01	Olafur Trollebo

02-29	**Creating issue to help out Lars with his graphing issue** H3E	**0:09:56** 15:05-15:15	Olafur Trollebo
02-29	**Figuring out how to do HEEE-195** Professional Programming	**0:23:55** 15:15-15:39	Olafur Trollebo
02-29	**HEEE-195** H3E	**0:16:29** 15:39-15:56	Olafur Trollebo
03-01	**Fixing things for Innit to send exceptions** H3E	**1:10:23** 13:20-14:30	Olafur Trollebo
03-01	**HEEE-198** H3E	**0:10:09** 14:30-14:40	Olafur Trollebo
03-02	**Figuring out why exceptions are not inserted** H3E	**0:16:09** 09:23-09:40	Olafur Trollebo
03-02	**HEEE-195** H3E	**0:05:53** 09:54-10:00	Olafur Trollebo
03-02	**HEEE-195** H3E	**0:10:55** 10:02-10:13	Olafur Trollebo
03-02	**Fixing documentation of API format** H3E	**0:06:22** 10:44-10:51	Olafur Trollebo
03-02	**HEEE-195** H3E	**0:08:10** 10:51-10:59	Olafur Trollebo
03-02	**Figuring out how to do HEEE-200** H3E	**0:15:58** 11:44-12:00	Olafur Trollebo
03-02	**Figuring out how to do HEEE-200** H3E	**0:04:37** 12:46-12:51	Olafur Trollebo
03-02	**HEEE-201** H3E	**0:50:23** 12:51-13:41	Olafur Trollebo
03-02	**Figuring out how to do HEEE-200** H3E	**0:27:33** 13:42-14:09	Olafur Trollebo
03-02	**HEEE-200** H3E	**1:03:42** 14:09-15:13	Olafur Trollebo
03-02	**HEEE-204** H3E	**0:02:34** 15:49-15:51	Olafur Trollebo
03-03	**Getting dev environment up and running** Professional Programming	**1:06:34** 09:20-10:27	Olafur Trollebo
03-03	**HEEE-199** H3E	**0:07:21** 10:27-10:34	Olafur Trollebo
03-03	**Getting dev environment up and running** Professional Programming	**0:16:40** 10:34-10:51	Olafur Trollebo
03-03	**HEEE-199** H3E	**0:13:24** 10:54-11:08	Olafur Trollebo
03-03	**HEEE-206** H3E	**0:30:37** 11:10-11:41	Olafur Trollebo
03-03	**HEEE-206** H3E	**0:14:47** 12:33-12:47	Olafur Trollebo

03-04	**Discussing our next steps and creating issues**	**1:04:42**	Olafur Trollebo
	H3E	09:50-10:55	
03-04	**Discussing our next steps and creating issues**	**0:04:51**	Olafur Trollebo
	H3E	11:13-11:18	
03-04	**HEEE-207**	**1:08:21**	Olafur Trollebo
	H3E	12:12-13:21	
03-04	**HEEE-207**	**1:28:20**	Olafur Trollebo
	H3E	13:21-14:49	
03-04	**Professional Programming Lecture**	**1:18:17**	Olafur Trollebo
	Professional Programming	14:49-16:08	
03-07	**Meeting with Innit at Hamar**	**4:20:00**	Olafur Trollebo
	H3E	11:00-15:20	
03-08	**Farup meeting and discussion afterwards about samples etc**	**2:05:02**	Olafur Trollebo
	H3E	09:10-11:15	
03-08	**Creating JIRA issues for Samples/Severity Rating**	**0:21:59**	Olafur Trollebo
	H3E	11:41-12:03	
03-08	**Asking Rune about MySQL Text**	**0:08:08**	Olafur Trollebo
	H3E	12:03-12:11	
03-08	**Creating JIRA issues for Samples/Severity Rating**	**0:25:28**	Olafur Trollebo
	H3E	12:11-12:37	
03-08	**Asking Rune about MySQL Text**	**0:11:21**	Olafur Trollebo
	H3E	12:37-12:48	
03-08	**Creating JIRA issues for Samples/Severity Rating**	**0:07:53**	Olafur Trollebo
	H3E	12:48-12:56	
03-08	**Figuring out how to do HEEE-208 (Threeway coupling)**	**1:53:03**	Olafur Trollebo
	Professional Programming	12:56-14:49	
03-08	**HEEE-208**	**0:34:29**	Olafur Trollebo
	H3E	14:49-15:24	
03-09	**Group Discussion**	**0:17:49**	Olafur Trollebo
	H3E	09:25-09:43	
03-09	**HEEE-205**	**0:06:44**	Olafur Trollebo
	H3E	09:44-09:50	
03-09	**Discussing settings view in frontend**	**0:51:24**	Olafur Trollebo
	H3E	09:50-10:41	
03-09	**HEEE-205**	**1:05:11**	Olafur Trollebo
	H3E	10:45-11:50	
03-09	**Learning more about Eloquent relationships**	**0:51:37**	Olafur Trollebo
	Professional Programming	12:02-12:53	
03-09	**HEEE-205**	**0:04:37**	Olafur Trollebo
	H3E	12:53-12:58	
03-09	**Discussing settings view in frontend**	**0:05:52**	Olafur Trollebo
	H3E	12:58-13:04	

03-09	**Discussing three-way coupling with the group** H3E	**0:14:06** 13:04-13:18	Olafur Trollebo
03-09	**HEEE-205** H3E	**0:02:07** 13:18-13:20	Olafur Trollebo
03-09	**Discussing three-way coupling with the group** H3E	**0:02:22** 13:20-13:23	Olafur Trollebo
03-09	**Asking Rune about MySQL Text and Threeway coupling** H3E	**0:14:55** 13:23-13:38	Olafur Trollebo
03-09	**HEEE-205** H3E	**1:37:31** 13:46-15:24	Olafur Trollebo
03-10	**HEEE-205** H3E	**1:31:06** 09:51-11:22	Olafur Trollebo
03-10	**HEEE-205** H3E	**0:30:14** 11:40-12:10	Olafur Trollebo
03-10	**Failed attempts at fixing test coverage in IDE** Professional Programming	**0:31:19** 12:10-12:42	Olafur Trollebo
03-10	**Asking Rune about MySQL Text and Threeway coupling** H3E	**0:13:14** 12:44-12:57	Olafur Trollebo
03-10	**HEEE-212** H3E	**2:20:34** 13:08-15:28	Olafur Trollebo
03-10	**HEEE-212** H3E	**0:28:07** 15:32-16:00	Olafur Trollebo
03-11	**Pull Request review and setting up work** H3E	**0:17:51** 09:31-09:49	Olafur Trollebo
03-11	**HEEE-209** H3E	**0:23:54** 09:49-10:13	Olafur Trollebo
03-11	**HEEE-209** H3E	**0:14:05** 10:13-10:28	Olafur Trollebo
03-11	**Asking Rune about MySQL Text and Threeway coupling - Rune still not here** H3E	**0:04:56** 10:34-10:39	Olafur Trollebo
03-11	**HEEE-210** H3E	**0:19:38** 10:39-10:59	Olafur Trollebo
03-11	**HEEE-210** H3E	**0:08:44** 11:20-11:29	Olafur Trollebo
03-11	**HEEE-210** H3E	**0:23:12** 12:02-12:26	Olafur Trollebo
03-11	**HEEE-210** H3E	**0:06:09** 12:26-12:32	Olafur Trollebo
03-11	**Discussion** H3E	**0:31:44** 12:32-13:04	Olafur Trollebo
03-11	**HEEE-210** H3E	**0:17:47** 13:05-13:23	Olafur Trollebo
03-11	**Professional Programming Lecture** Professional Programming	**2:00:00** 14:00-16:00	Olafur Trollebo

03-14	**HEEE-222** H3E	**0:49:43** 09:28-10:18	Olafur Trollebo
03-14	**HEEE-223** H3E	**0:39:57** 10:18-10:58	Olafur Trollebo
03-14	**Pull request for HEEE-222 and HEEE-223** H3E	**0:05:03** 11:16-11:21	Olafur Trollebo
03-14	**HEEE-207** H3E	**0:03:14** 11:21-11:25	Olafur Trollebo
03-14	**HEEE-207** H3E	**0:15:12** 11:26-11:41	Olafur Trollebo
03-14	**HEEE-207** H3E	**0:22:00** 12:02-12:24	Olafur Trollebo
03-14	**HEEE-207** H3E	**0:26:38** 12:24-12:50	Olafur Trollebo
03-14	**Fixing merge conflicts** H3E	**0:06:01** 12:50-12:56	Olafur Trollebo
03-14	**Creating issues in JIRA for test** Professional Programming	**0:14:16** 13:00-13:14	Olafur Trollebo
03-14	**HEEE-224** H3E	**0:17:34** 13:14-13:32	Olafur Trollebo
03-14	**Setting up test suites in PHPUnit** Professional Programming	**0:02:33** 13:32-13:35	Olafur Trollebo
03-14	**HEEE-224** H3E	**0:31:25** 13:35-14:06	Olafur Trollebo
03-14	**HEEE-224** H3E	**0:46:04** 14:06-14:52	Olafur Trollebo
03-14	**HEEE-224** H3E	**0:04:35** 14:53-14:58	Olafur Trollebo
03-14	**HEEE-225** H3E	**0:26:25** 15:01-15:27	Olafur Trollebo
03-15	**Farup meeting and discussion afterwards about samples etc** H3E	**1:00:04** 09:15-10:15	Olafur Trollebo
03-15	**HEEE-225** H3E	**0:16:27** 10:45-11:02	Olafur Trollebo
03-15	**HEEE-224** H3E	**0:41:10** 12:02-12:43	Olafur Trollebo
03-15	**HEEE-224 ExceptionsTests** H3E	**0:04:40** 12:43-12:48	Olafur Trollebo
03-15	**HEEE-224** H3E	**0:12:38** 12:48-13:01	Olafur Trollebo
03-15	**HEEE-224** H3E	**1:32:42** 13:07-14:40	Olafur Trollebo
03-15	**HEEE-224** H3E	**0:43:43** 14:51-15:35	Olafur Trollebo
03-16	**Reading about Laravel** Professional Programming	**0:30:28** 09:10-09:41	Olafur Trollebo

03-16	**HEEE-224** H3E	**0:49:58** 09:41-10:31	Olafur Trollebo
03-17	**Reading up on Laravel Best Practices** Professional Programming	**0:19:48** 10:00-10:20	Olafur Trollebo
03-17	**HEEE-226** H3E	**0:24:49** 10:23-10:48	Olafur Trollebo
03-17	**Talking about Syslog route** H3E	**0:07:49** 10:48-10:56	Olafur Trollebo
03-17	**HEEE-226** H3E	**0:05:23** 10:56-11:02	Olafur Trollebo
03-17	**HEEE-226** H3E	**0:18:06** 11:02-11:20	Olafur Trollebo
03-17	**HEEE-226** H3E	**1:06:29** 11:36-12:42	Olafur Trollebo
03-17	**HEEE-226** H3E	**0:24:50** 12:43-13:07	Olafur Trollebo
03-17	**HEEE-229** H3E	**0:41:08** 13:12-13:53	Olafur Trollebo
03-17	**HEEE-229** H3E	**1:31:57** 13:53-15:25	Olafur Trollebo
03-17	**HEEE-229** H3E	**0:28:42** 15:25-15:54	Olafur Trollebo
03-18	**HEEE-229** H3E	**0:12:50** 09:19-09:32	Olafur Trollebo
03-18	**Creating issues in JIRA** Professional Programming	**0:25:31** 09:32-09:58	Olafur Trollebo
03-18	**HEEE-233** H3E	**0:57:02** 09:58-10:55	Olafur Trollebo
03-18	**Create issues in JIRA** Professional Programming	**0:17:38** 11:55-12:13	Olafur Trollebo
03-18	**HEEE-237** H3E	**0:11:26** 12:18-12:29	Olafur Trollebo
03-18	**HEEE-237** H3E	**0:45:16** 12:55-13:40	Olafur Trollebo
03-23	**HEEE-238** H3E	**0:12:11** 16:36-16:48	Olafur Trollebo
03-23	**HEEE-234** H3E	**0:15:26** 17:07-17:23	Olafur Trollebo
03-23	**HEEE-234** H3E	**0:21:52** 17:23-17:44	Olafur Trollebo
03-29	**Meeting with Ivar Farup** H3E	**0:55:01** 09:10-10:05	Olafur Trollebo
03-29	**HEEE-231** H3E	**0:19:33** 10:27-10:47	Olafur Trollebo
03-29	**Helping Lars figure out updateRatings** H3E	**0:11:53** 10:47-10:59	Olafur Trollebo

03-29	**HEEE-231** H3E	**0:57:18** 10:59-11:56	Olafur Trollebo
03-29	**Discussing** H3E	**0:28:54** 11:56-12:25	Olafur Trollebo
03-29	**HEEE-231** H3E	**0:15:00** 12:25-12:40	Olafur Trollebo
03-29	**HEEE-231** H3E	**0:50:44** 12:40-13:31	Olafur Trollebo
03-29	**Creating issues in JIRA to help out the frontend** H3E	**0:41:48** 14:37-15:18	Olafur Trollebo
03-29	**HEEE-247** H3E	**0:19:00** 15:19-15:38	Olafur Trollebo
03-29	**HEEE-247** H3E	**0:06:08** 15:38-15:44	Olafur Trollebo
03-30	**HEEE-246** H3E	**0:59:55** 09:18-10:18	Olafur Trollebo
03-30	**HEEE-246** H3E	**0:39:36** 10:18-10:58	Olafur Trollebo
03-30	**Reading up on database triggers for HEEE-245** Professional Programming	**0:13:59** 11:02-11:16	Olafur Trollebo
03-30	**HEEE-245** H3E	**0:25:45** 11:16-11:42	Olafur Trollebo
03-30	**HEEE-248** H3E	**0:41:45** 12:02-12:44	Olafur Trollebo
03-31	**HEEE-249** H3E	**1:05:52** 09:43-10:49	Olafur Trollebo
03-31	**Creating JIRA issues based on discussion about Heat** H3E	**0:26:03** 11:24-11:50	Olafur Trollebo
03-31	**Creating JIRA issues based on discussion about Heat** H3E	**1:03:30** 12:09-13:12	Olafur Trollebo
03-31	**HEEE-252** H3E	**0:24:14** 13:12-13:37	Olafur Trollebo
03-31	**Creating JIRA issues based on discussion about Heat** H3E	**0:14:19** 13:40-13:55	Olafur Trollebo
03-31	**HEEE-251** H3E	**0:10:00** 14:03-14:13	Olafur Trollebo
03-31	**HEEE-254** H3E	**0:08:07** 14:13-14:21	Olafur Trollebo
03-31	**HEEE-255** H3E	**0:40:40** 14:51-15:32	Olafur Trollebo
04-01	**Creating JIRA issues** Professional Programming	**0:18:02** 09:06-09:24	Olafur Trollebo
04-01	**HEEE-256** H3E	**0:15:25** 09:25-09:40	Olafur Trollebo

04-01	**Creating JIRA issues for Heat** H3E	**0:41:37** 10:08-10:50	Olafur Trollebo
04-01	**Reading up on database events for Heat degradation** H3E	**0:05:40** 10:50-10:56	Olafur Trollebo
04-01	**Reading up on database events for Heat degradation** Professional Programming	**2:33:12** 11:15-13:48	Olafur Trollebo
04-01	**HEEE-257** H3E	**2:07:44** 13:48-15:56	Olafur Trollebo
04-04	**HEEE-257** H3E	**1:25:29** 09:31-10:56	Olafur Trollebo
04-04	**Asking Kolloen about API** Professional Programming	**0:54:23** 10:57-11:51	Olafur Trollebo
04-04	**Reworking on suggestion from Mariusz and Rune** H3E	**0:14:56** 12:22-12:37	Olafur Trollebo
04-04	**HEEE-257** H3E	**0:28:25** 12:42-13:10	Olafur Trollebo
04-04	**HEEE-258** H3E	**0:18:28** 13:14-13:32	Olafur Trollebo
04-04	**Emergency damage control** H3E	**0:50:53** 13:32-14:23	Olafur Trollebo
04-04	**HEEE-260** H3E	**0:11:42** 14:56-15:07	Olafur Trollebo
04-05	**HEEE-264** H3E	**0:57:51** 09:42-10:40	Olafur Trollebo
04-05	**HEEE-266** H3E	**0:12:29** 10:40-10:53	Olafur Trollebo
04-05	**HEEE-266** H3E	**0:20:13** 11:23-11:43	Olafur Trollebo
04-05	**HEEE-262** H3E	**0:39:30** 12:09-12:48	Olafur Trollebo
04-05	**HEEE-259** H3E	**0:22:14** 13:27-13:49	Olafur Trollebo
04-05	**Figuring out how to couple for trigger - HEEE-259** Professional Programming	**0:30:29** 13:49-14:20	Olafur Trollebo
04-05	**HEEE-259** H3E	**1:28:33** 14:24-15:52	Olafur Trollebo
04-06	**HEEE-259** H3E	**0:48:56** 09:33-10:22	Olafur Trollebo
04-06	**Asking Rune about things** H3E	**1:12:00** 10:22-11:34	Olafur Trollebo
04-06	**HEEE-259** H3E	**0:14:44** 11:39-11:54	Olafur Trollebo
04-06	**HEEE-259** H3E	**1:21:19** 14:29-15:51	Olafur Trollebo

04-07	**HEEE-268** H3E	**0:16:41** 09:19-09:36	Olafur Trollebo
04-07	**Documenting what happened the last few days.** Professional Programming	**0:22:24** 09:38-10:01	Olafur Trollebo
04-07	**Removing ON UPDATE from timestamp in current_heat** H3E	**0:31:25** 10:18-10:49	Olafur Trollebo
04-07	**HEEE-259** H3E	**0:47:52** 11:03-11:51	Olafur Trollebo
04-07	**HEEE-259** H3E	**1:00:05** 12:17-13:17	Olafur Trollebo
04-07	**HEEE-271** H3E	**0:22:23** 13:59-14:21	Olafur Trollebo
04-07	**Running tests before creating pull request** H3E	**1:07:35** 14:21-15:29	Olafur Trollebo
04-08	**HEEE-271** H3E	**0:26:12** 10:35-11:01	Olafur Trollebo
04-08	**HEEE-271** H3E	**0:08:04** 12:06-12:14	Olafur Trollebo
04-08	**HEEE-271** H3E	**0:23:35** 12:14-12:37	Olafur Trollebo
04-08	**Creating JIRA issues** Professional Programming	**0:04:41** 12:45-12:49	Olafur Trollebo
04-08	**HEEE-274** H3E	**0:32:44** 12:49-13:22	Olafur Trollebo
04-08	**Professional Programming** Professional Programming	**2:04:26** 14:15-16:20	Olafur Trollebo
04-11	**HEEE-269** H3E	**1:44:34** 09:46-11:30	Olafur Trollebo
04-11	**Trying to find a link I had long time ago for the report** H3E	**0:05:42** 12:04-12:10	Olafur Trollebo
04-11	**Trying to find a link I had long time ago for the report** H3E	**0:05:45** 12:26-12:31	Olafur Trollebo
04-11	**HEEE-269** H3E	**1:22:58** 12:31-13:54	Olafur Trollebo
04-12	**Meeting with Ivar Farup** H3E	**0:48:15** 09:12-10:00	Olafur Trollebo
04-12	**HEEE-269** H3E	**0:23:37** 10:25-10:48	Olafur Trollebo
04-12	**HEEE-269** H3E	**0:04:20** 10:54-10:58	Olafur Trollebo
04-12	**HEEE-269** H3E	**0:13:24** 10:58-11:11	Olafur Trollebo
04-12	**HEEE-269** H3E	**0:49:31** 11:26-12:15	Olafur Trollebo

04-12	**Creating issue in JIRA** Professional Programming	**0:19:35** 12:35-12:55	Olafur Trollebo
04-12	**HEEE-282** H3E	**0:12:46** 12:55-13:08	Olafur Trollebo
04-12	**HEEE-280** H3E	**0:03:06** 13:51-13:54	Olafur Trollebo
04-12	**HEEE-282** H3E	**0:17:07** 13:54-14:11	Olafur Trollebo
04-12	**Talking about frontend** H3E	**0:31:46** 14:11-14:42	Olafur Trollebo
04-12	**HEEE-280** H3E	**0:09:00** 14:43-14:52	Olafur Trollebo
04-12	**Reviewing pull requests!** H3E	**0:03:18** 14:52-14:56	Olafur Trollebo
04-12	**HEEE-280** H3E	**0:12:40** 15:12-15:25	Olafur Trollebo
04-13	**HEEE-281** H3E	**0:52:39** 10:01-10:54	Olafur Trollebo
04-13	**HEEE-281** H3E	**0:37:01** 11:30-12:07	Olafur Trollebo
04-13	**HEEE-290** H3E	**0:56:41** 12:08-13:04	Olafur Trollebo
04-13	**Creating JIRA issues** Professional Programming	**0:14:17** 13:28-13:42	Olafur Trollebo
04-13	**HEEE-293** H3E	**0:25:31** 13:42-14:08	Olafur Trollebo
04-13	**Figuring out why things fail** H3E	**0:07:53** 14:08-14:16	Olafur Trollebo
04-13	**HEEE-293** H3E	**0:08:05** 14:16-14:24	Olafur Trollebo
04-13	**HEEE-294** H3E	**0:32:13** 14:27-14:59	Olafur Trollebo
04-13	**Figuring out how to do HEEE-253** Professional Programming	**0:32:56** 15:03-15:36	Olafur Trollebo
04-14	**Figuring out how to do HEEE-253** Professional Programming	**0:15:37** 10:08-10:24	Olafur Trollebo
04-14	**Waiting for Aleksander to upload the images taken yesterday - Need them for the issue.** H3E	**1:19:23** 10:24-11:44	Olafur Trollebo
04-14	**Figuring out how to do HEEE-253 - Resulted in resolving it in code** Professional Programming	**0:20:50** 12:39-13:00	Olafur Trollebo
04-14	**HEEE-253** H3E	**0:59:13** 13:00-13:59	Olafur Trollebo
04-14	**HEEE-253** H3E	**1:38:09** 13:59-15:37	Olafur Trollebo

04-14	**HEEE-253** H3E	**0:26:00** 15:38-16:04	Olafur Trollebo
04-15	**Discussing work** H3E	**0:44:18** 09:10-09:54	Olafur Trollebo
04-15	**Meeting with Eivind** H3E	**1:05:44** 09:55-11:00	Olafur Trollebo
04-15	**HEEE-295** H3E	**0:19:51** 11:32-11:52	Olafur Trollebo
04-15	**HEEE-295** H3E	**1:06:16** 11:52-12:58	Olafur Trollebo
04-15	**HEEE-295** H3E	**1:15:18** 13:03-14:18	Olafur Trollebo
04-15	**Professional Programming Lecture** Professional Programming	**0:29:43** 14:18-14:48	Olafur Trollebo
04-15	**HEEE-295** H3E	**0:55:12** 14:48-15:43	Olafur Trollebo
04-18	**HEEE-253** H3E	**0:00:56** 09:27-09:28	Olafur Trollebo
04-18	**Working with Pull request** H3E	**0:14:09** 09:29-09:43	Olafur Trollebo
04-18	**HEEE-253** H3E	**0:16:58** 09:43-10:00	Olafur Trollebo
04-18	**HEEE-253** H3E	**1:43:17** 10:25-12:09	Olafur Trollebo
04-18	**Writing up HEEE-253 on the whiteboard** H3E	**0:12:07** 12:09-12:21	Olafur Trollebo
04-18	**HEEE-253** H3E	**0:06:57** 13:25-13:32	Olafur Trollebo
04-18	**Reading our report** H3E	**0:39:26** 13:32-14:12	Olafur Trollebo
04-18	**Reading our report** H3E	**0:05:37** 14:39-14:44	Olafur Trollebo
04-19	**Reading our report** H3E	**0:53:25** 10:18-11:12	Olafur Trollebo
04-19	**HEEE-296** H3E	**0:30:58** 11:51-12:22	Olafur Trollebo
04-19	**Discussing Rolling average** H3E	**2:09:02** 12:22-14:31	Olafur Trollebo
04-19	**Creating JIRA issues** Professional Programming	**0:04:19** 14:37-14:41	Olafur Trollebo
04-19	**HEEE-306** H3E	**1:02:06** 14:41-15:43	Olafur Trollebo
04-20	**Fixing things I did yesterday up to standards** H3E	**0:06:07** 11:56-12:02	Olafur Trollebo
04-20	**HEEE-306** H3E	**2:06:53** 12:12-14:19	Olafur Trollebo

04-20	**HEEE-309** H3E	**0:40:33** 14:53-15:34	Olafur Trollebo
04-21	**HEEE-306** H3E	**1:12:37** 09:47-11:00	Olafur Trollebo
04-21	**HEEE-278** H3E	**0:31:40** 12:05-12:37	Olafur Trollebo
04-21	**HEEE-278** H3E	**0:57:58** 12:50-13:48	Olafur Trollebo
04-21	**Lars said something was wrong with averageheat - Nothing was wrong** H3E	**0:11:15** 13:48-13:59	Olafur Trollebo
04-21	**HEEE-278** H3E	**0:51:37** 13:59-14:51	Olafur Trollebo
04-22	**Reading up on events in MySQL - Figuring out how to do it every whole hour** Professional Programming	**0:36:33** 10:05-10:42	Olafur Trollebo
04-22	**HEEE-312** H3E	**0:10:47** 10:47-10:58	Olafur Trollebo
04-22	**HEEE-313** H3E	**0:47:38** 11:19-12:06	Olafur Trollebo
04-22	**Discussing** H3E	**0:09:16** 12:53-13:03	Olafur Trollebo
04-22	**HEEE-313** H3E	**0:33:00** 13:03-13:36	Olafur Trollebo
04-22	**Commiting the last hours work** H3E	**0:42:43** 13:44-14:27	Olafur Trollebo
04-22	**Creating JIRA issues for validation** Professional Programming	**0:14:35** 14:37-14:52	Olafur Trollebo
04-22	**Code review** Professional Programming	**0:59:46** 15:05-16:05	Olafur Trollebo
04-25	**Reviewing pull request and discussing possible changes** H3E	**1:33:42** 09:32-11:06	Olafur Trollebo
04-25	**HEEE-314** H3E	**0:14:23** 11:24-11:39	Olafur Trollebo
04-25	**HEEE-314** H3E	**1:57:39** 11:57-13:55	Olafur Trollebo
04-25	**HEEE-314** H3E	**0:22:49** 14:02-14:25	Olafur Trollebo
04-25	**HEEE-314** H3E	**0:54:32** 14:25-15:20	Olafur Trollebo
04-26	**Creating JIRA issues** Professional Programming	**0:12:54** 09:13-09:25	Olafur Trollebo
04-26	**HEEE-315** H3E	**1:42:06** 09:26-11:08	Olafur Trollebo
04-26	**Waiting for Aleksander to approve the Pull Request** H3E	**0:17:05** 09:33-09:50	Olafur Trollebo

04-26	**HEEE-315** H3E	**0:07:43** 11:42-11:50	Olafur Trollebo
04-26	**Meeting with Innit** H3E	**16:10:00** 23:50-16:00	Olafur Trollebo
04-27	**HEEE-316** H3E	**1:19:22** 09:58-11:17	Olafur Trollebo
04-27	**Committing HEEE-316** H3E	**0:09:06** 11:54-12:03	Olafur Trollebo
04-27	**HEEE-320** H3E	**0:16:03** 12:03-12:19	Olafur Trollebo
04-27	**HEEE-325** H3E	**0:09:57** 12:41-12:51	Olafur Trollebo
04-27	**HEEE-326** H3E	**2:08:21** 13:25-15:34	Olafur Trollebo
04-28	**HEEE-326** H3E	**0:48:18** 09:28-10:16	Olafur Trollebo
04-28	**HEEE-328** H3E	**0:01:22** 10:16-10:17	Olafur Trollebo
04-28	**HEEE-327** H3E	**0:06:01** 10:18-10:24	Olafur Trollebo
04-28	**HEEE-327** H3E	**0:06:40** 10:25-10:31	Olafur Trollebo
04-28	**HEEE-327** H3E	**0:12:21** 10:32-10:44	Olafur Trollebo
04-28	**HEEE-330** H3E	**0:24:40** 10:50-11:15	Olafur Trollebo
04-28	**HEEE-330** H3E	**3:45:52** 11:38-15:24	Olafur Trollebo
05-02	**HEEE-330** H3E	**1:44:33** 09:31-11:15	Olafur Trollebo
05-02	**HEEE-330** H3E	**0:13:29** 11:33-11:46	Olafur Trollebo
05-02	**HEEE-334** H3E	**0:06:57** 11:46-11:53	Olafur Trollebo
05-02	**HEEE-330** H3E	**1:47:13** 11:53-13:40	Olafur Trollebo
05-02	**HEEE-330** H3E	**0:05:56** 14:13-14:19	Olafur Trollebo
05-02	**HEEE-332** H3E	**0:02:23** 14:32-14:34	Olafur Trollebo
05-02	**HEEE-332** H3E	**0:16:33** 14:34-14:51	Olafur Trollebo
05-03	**Meeting with Ivar Farup** H3E	**1:00:11** 09:10-10:10	Olafur Trollebo
05-03	**Figuring out how to do HEEE-331** Professional Programming	**0:09:29** 10:30-10:40	Olafur Trollebo

05-03	**HEEE-331** H3E	**0:21:45** 10:40-11:01	Olafur Trollebo
05-03	**HEEE-331** H3E	**0:03:35** 11:20-11:23	Olafur Trollebo
05-03	**Discussing with Lars regarding the graph route not working correctly** H3E	**0:19:21** 11:23-11:42	Olafur Trollebo
05-03	**Figuring out what is wrong - and fixing it - HEEE-340** H3E	**1:19:02** 11:42-13:01	Olafur Trollebo
05-03	**HEEE-339** H3E	**0:45:41** 13:06-13:52	Olafur Trollebo
05-03	**Reviewing pull request and discussing possible changes** H3E	**0:29:19** 14:11-14:40	Olafur Trollebo
05-03	**HEEE-345** H3E	**0:05:41** 14:51-14:56	Olafur Trollebo
05-03	**HEEE-344** H3E	**0:09:55** 14:56-15:06	Olafur Trollebo
05-04	**Creating JIRA issues** Professional Programming	**0:06:33** 09:33-09:39	Olafur Trollebo
05-04	**Reading over the report and inputting stuff** H3E	**0:33:23** 09:40-10:13	Olafur Trollebo
05-04	**HEEE-347** H3E	**0:17:47** 10:14-10:31	Olafur Trollebo
05-04	**HEEE-347** H3E	**0:41:41** 10:31-11:13	Olafur Trollebo
05-04	**HEEE-347** H3E	**0:20:19** 11:19-11:39	Olafur Trollebo
05-04	**HEEE-347** H3E	**0:22:16** 11:39-12:01	Olafur Trollebo
05-04	**HEEE-347** H3E	**0:22:30** 13:35-13:58	Olafur Trollebo
05-04	**Reading coverage reports** H3E	**0:36:44** 14:42-15:18	Olafur Trollebo
05-05	**HEEE-350** H3E	**0:48:14** 09:59-10:47	Olafur Trollebo
05-05	**HEEE-353** H3E	**1:58:27** 10:47-12:46	Olafur Trollebo
05-06	**Figuring out what is wrong with graph route** H3E	**1:22:14** 09:32-10:54	Olafur Trollebo
05-06	**Figuring out what is wrong with graph route** H3E	**0:35:37** 11:03-11:39	Olafur Trollebo
05-06	**Agreeing with Lars to edit the format being sent from the backend so that we do not have to zeropad in the backend** H3E	**0:07:51** 11:39-11:47	Olafur Trollebo
05-06	**HEEE-355** H3E	**1:14:55** 11:47-13:02	Olafur Trollebo

05-06	**HEEE-351** H3E	**0:25:01** 13:02-13:27	Olafur Trollebo
05-06	**HEEE-349** H3E	**0:20:32** 14:19-14:40	Olafur Trollebo
05-06	**Simon** Professional Programming	**0:49:44** 14:40-15:29	Olafur Trollebo
05-06	**HEEE-349** H3E	**0:52:59** 15:29-16:22	Olafur Trollebo
05-08	**Fixing some small bugs with Graph and commiting the tests - Lacking Graph test at the moment** H3E	**0:13:29** 14:50-15:03	Olafur Trollebo
05-08	**HEEE-349** H3E	**0:12:15** 15:03-15:16	Olafur Trollebo
05-09	**Finding code fragments for the report** H3E	**0:29:45** 09:10-09:40	Olafur Trollebo
05-09	**HEEE-349** H3E	**0:57:24** 09:40-10:38	Olafur Trollebo
05-09	**HEEE-349** H3E	**0:51:57** 10:39-11:31	Olafur Trollebo
05-09	**HEEE-349** H3E	**2:09:19** 12:03-14:12	Olafur Trollebo
05-10	**Meeting with Ivar Farup** H3E	**1:00:00** 09:05-10:05	Olafur Trollebo
05-10	**Discussing with Lars what to do with Reports view** H3E	**0:14:10** 11:02-11:17	Olafur Trollebo
05-10	**HEEE-358** H3E	**0:30:31** 11:37-12:07	Olafur Trollebo
05-10	**HEEE-349** H3E	**0:38:53** 12:11-12:50	Olafur Trollebo
05-10	**HEEE-349** H3E	**0:21:28** 12:50-13:11	Olafur Trollebo
05-10	**HEEE-349** H3E	**0:30:37** 13:11-13:42	Olafur Trollebo
05-10	**Fixing a small bug in ExceptionContext** H3E	**0:51:40** 13:53-14:44	Olafur Trollebo
05-10	**HEEE-347** H3E	**0:18:00** 15:07-15:25	Olafur Trollebo
05-11	**HEEE-347** H3E	**0:40:33** 09:18-09:58	Olafur Trollebo
05-11	**HEEE-347** H3E	**0:45:15** 10:23-11:08	Olafur Trollebo
05-11	**HEEE-347** H3E	**1:14:20** 11:26-12:40	Olafur Trollebo
05-11	**Report - Finding refactored code in backend** H3E	**0:09:04** 12:56-13:05	Olafur Trollebo

05-11	**Report - Finding refactored code in backend** H3E	**0:39:40** 13:18-13:57	Olafur Trollebo
05-11	**Checking Codecoverage** H3E	**0:26:46** 13:57-14:24	Olafur Trollebo
05-12	**Fixing up the timelog** H3E	**1:34:59** 09:10-10:45	Olafur Trollebo
05-12	**Fixing an error in the backend related to null values** H3E	**0:25:31** 10:45-11:10	Olafur Trollebo
05-12	**Reading report** H3E	**0:22:40** 12:47-13:10	Olafur Trollebo
05-12	**Talking to Simon about the timelogs** H3E	**0:20:12** 13:10-13:30	Olafur Trollebo
05-12	**Fixing up the timelog once again** H3E	**0:44:59** 13:30-14:15	Olafur Trollebo
05-12	**Reading report** H3E	**0:23:47** 14:15-14:38	Olafur Trollebo
05-13	**Skype meeting with Innit** H3E	**0:50:10** 10:05-10:55	Olafur Trollebo
05-13	**Change the default degradation time and percentage** H3E	**0:04:17** 11:06-11:10	Olafur Trollebo

Created with toggl.com