



Norwegian University of
Science and Technology

Management System for CS Assignments

Author(s)

Johan Aanesen
Brede Fritjof Klausen
Svein Are Danielsen

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

19.05.2019

Supervisor

Ivar Farup

Sammendrag av Bacheloroppgaven

Tittel:	Administrasjonssystem for datavitenskapsoppgaver
Oppgave no.	53
Dato:	19.05.2019
Deltakere:	Johan Aanesen Brede Fritjof Klausen Svein Are Danielsen
Veiledere:	Ivar Farup
Oppdragsgiver:	Norwegian University of Science and Technology
Kontaktperson:	Christopher Frantz, christopher.frantz@ntnu.no, 61135400
Nøkkelord:	Go, HTML, JavaScript, Docker, Docker-Compose, Open-Stack, Database
Antall sider:	50
Antall vedlegg:	9
Tilgjengelighet:	Åpen

Sammendrag:	Prosjektets mål har vært å lage et web basert system som simplifiserer oppgave innlevering og gjennomgang av andre sine oppgaver, dette erstatter den tidligere brukte organiseringen gjennom regneark. Gjennom bacheloren har systemet blitt testet på, og brukt av studenter i fem forskjellige innleveringer i tre forskjellige emner. Bruk av tilbakemeldingene fra både studentene og lærerne, har resultert i et alt i alt bedre system og applikasjon som fungerer for oppgavene nevnt ovenfor.
-------------	--

Summary of Graduate Project

Title:	Management System for CS Assignments
Project no.	53
Date:	19.05.2019
Authors:	Johan Aanesen Brede Fritjof Klausen Svein Are Danielsen
Supervisor:	Ivar Farup
Employer:	Norwegian University of Science and Technology
Contact Person:	Christopher Frantz, christopher.frantz@ntnu.no, 61135400
Keywords:	Go, HTML, JavaScript, Docker, Docker-Compose, Open-Stack, Database
Pages:	50
Attachments:	9
Availability:	Open

Abstract: The goal of this project was to create a web solution that simplifies assignment submissions and peer reviewing of programming related assignments, replacing the spreadsheets used to organize this from before. Through the bachelor the system has been tested on, and used by students on five different assignments in three different courses. Using the feedback from both the students and the teacher has resulted in an overall better system and application that works for the purposes stated above.

Preface

This Bachelor thesis is written by Johan Aanesen, Brede Fritjof Klausen and Svein Are Danielsen at NTNU Gjøvik.

We would like to thank our supervisor Ivar Farup, for extensive feedback, interesting and educational discussions, help on the paper as well as being available throughout the project. We would also like to thank our product owner, on behalf of NTNU, Christopher Frantz for being available all the time, coming up with great feedback and ideas needed to complete the project as well as being interested in our development process and helping us along the way. Lastly we want to thank the students who used our system throughout the development stages for giving us valuable feedback to improve the system.

Contents

Preface	iii
Contents	iv
List of Figures	vii
List of Tables	viii
Listings	ix
Acronyms	x
Glossary	xi
1 Introduction	1
1.1 Task Description	1
1.2 Goals	1
1.2.1 Result	1
1.2.2 Effect	1
1.2.3 Learning	1
1.3 Project restrictions	2
1.3.1 Technological	2
1.4 User base	2
1.5 Development framework	2
1.6 Project Organization	2
1.6.1 Project members	2
1.6.2 Responsibilities and roles	3
1.6.3 Why we chose this assignment	3
1.7 Report layout	4
2 Development Process	5
2.1 Choice of method	5
2.1.1 Argumentation	5
2.1.2 Conclusion	6
2.2 Execution of the method	6
2.3 Tool usage	7
2.3.1 Selected tools	7
3 Requirements	9
3.1 Use cases	9
3.1.1 Use case-diagram	10
3.1.2 Use case descriptions	11
3.2 Requirements	15
3.2.1 Functional requirements	15
3.2.2 Non-functional requirements	16
4 Technical Design	17
4.1 System overview	17
4.2 Architecture	17
4.2.1 Service Oriented Architecture	17
4.2.2 Model-View-Controller	18
4.3 UI Design	20
4.3.1 Graphic Design	20
4.3.2 Interaction Design	20
4.4 Database design	23
4.5 Project/File structure	24
5 Implementation	27
5.1 Web server	27

5.1.1	Routing	27
5.1.2	HTTP Handlers / Controllers	28
5.1.3	Parsing the request body	29
5.1.4	Parsing form data	29
5.1.5	View	30
5.1.6	Templating	30
5.1.7	Form Builder	31
5.2	Database	32
5.3	Mail service	34
5.4	User Authentication	36
5.5	Login Sequence	37
5.6	Register Sequence	38
5.7	Codebase	38
6	Deployment	39
6.1	OpenStack	39
6.2	Docker	40
6.3	Docker Compose	40
6.3.1	Web Service	40
6.3.2	Mail Service	40
6.3.3	Database	40
6.4	Makefile	41
7	Testing and User Feedback	42
7.1	Testing	42
7.1.1	Unit Testing	42
7.1.2	User Testing	42
7.2	Quality Assurance	42
7.3	User Feedback	43
8	Discussion	44
8.1	Results	44
8.2	Choice of technical solutions	44
8.3	Peer Review	44
8.4	Logging	44
8.5	Repository And Service	44
8.6	Form Builder	45
9	Conclusion	46
9.1	Evaluation of the Group's Work	46
9.2	Final Words	46
9.3	Future Work	47
9.3.1	Auto-Validation Functionality	47
9.3.2	API Based Micro Service Architecture	47
9.3.3	Front-end	47
9.3.4	Notification and Messaging Service	47
9.3.5	Time-zone Error	48
9.3.6	Feedback From Project-Owner And Students	48
	Bibliography	50
A	Project Repository	51
B	Project Agreement	52
C	Project plan	56
D	Meeting Logs	66
D.1	Record of meetings	66
D.1.1	17.01.19 - Thursday	66
D.1.2	28.01.19 - Monday	66
D.1.3	11.02.19 - Monday	66

D.1.4	18.02.19 - Monday	67
D.1.5	04.03.19 - Monday	67
D.1.6	11.03.19 - Monday	67
D.1.7	21.03.19 - Thursday	68
D.1.8	01.04.19 - Monday	68
D.1.9	04.04.19 - Thursday	68
D.1.10	11.04.19 - Wednesday	69
D.1.11	29.04.19 - Monday	69
E	Daily Logs	70
E.1	Johan Logs	70
E.2	Brede Logs	78
E.3	Svein Logs	88
F	Peer Review Discussion	100
G	Screenshots	103
H	Trello Board	112
I	Toggl Summary	120

List of Figures

1	Use Case Diagram	10
2	System View	17
3	Webservice MVC	19
4	UI Badges in Card example	20
5	Initial Site Map	21
6	Final Site Map	22
7	Initial Database Design	23
8	Final Database Design	25
9	File Structure	26
10	Form Builder screen dump	32
11	Authentication Model	37
12	Sequence Diagram for logging in	37
13	Sequence Diagram for registering an user	38

List of Tables

1	Group members & roles	3
2	Trello-board description	7
3	Use case for registering new user	11
4	Use case for authenticating	11
5	Use case for logging out	11
6	Use case for forgotten password	11
7	Use case for joining a course through URL	11
8	Use case for joining a course with course hash	12
9	Use case for delivering assignment	12
10	Use case for re-delivering assignment	12
11	Use case for withdrawing assignment	12
12	Use case for performing review	12
13	Use case for updating performed review	13
14	Use case for editing profile	13
15	Use case for creating course	13
16	Use case for updating course	13
17	Use case for emailing students in a course	13
18	Use case for creating an assignment	14
19	Use case for updating an assignment	14
20	Use case for generating report of all user submissions from one assignment	14
21	Use case for creating a submission/review form	14
22	Use case for updating a submission/review form	14
23	Use case for updating submission/review form weights	15
24	Use case for removing student from course	15
25	Use case for updating a student's password	15
26	Use case for updating the FAQ	15
27	Languages used in the project	38

Listings

5.1	Simple HTTP-server in Go	27
5.2	Mux routing	27
5.3	HTTP Handler	28
5.4	Using Mux inside handlers	28
5.5	Parsing POST-request body	29
5.6	Parsing form data	29
5.7	Usage of View	30
5.8	Sample of a Template	30
5.9	Template Plugin	31
5.10	Plugin usage inside templates	31
5.11	MySQL Database Connection	32
5.12	MySQL Update Example	33
5.13	MySQL Insert Example	33
5.14	MySQL Fetch Example	34
5.15	SendMail function in Go, line 217	35
5.16	Send mail from webservice, line 199	35

Acronyms

BIDAT Bachelor in Engineering - Computer Science. [2](#)

BPROG Bachelor in Programming. [2](#)

FAQ Frequently Asked Questions. [15](#), [67](#)

IaaS Infrastructure as a Service. [39](#)

IDE Integrated development environment. [8](#)

JSON JavaScript Object Notation. [29](#)

MD Markdown. [15](#)

NTNU Norwegian University of Science and Technology. [8](#)

QA Quality Assurance. [42](#), [68](#)

RUP Rational Unified Process. [5](#)

TA Teaching assistant. [2](#), [11–15](#), [48](#), [68](#)

UI User Interface. [4](#)

VPN Virtual Private Network. [39](#)

XP eXtreme Programming. [6](#)

Glossary

- backlog** An accumulation of uncompleted work or matters needing to be dealt with. [5](#), [6](#)
- code review** A developer reads and tests another developers code, seeing if the code standards is begin held, and securing good code quality. [5](#), [7](#)
- eXtreme Programming** An agile development model that focuses on programming and rapid delivery. [5](#)
- Go Programming Language** A statically typed, compiled programming language designed at Google. [1](#), [2](#)
- Kanban** An agile development method that focuses on limiting the amount of operation done simultaneously, for securing that a operation is done before starting on another. [5](#), [6](#)
- Kanban board** A kanban board is an agile project management tool designed to help visualize work, limit work-in-progress, and maximize efficiency. [5-7](#)
- pull request** A request to merge two branches in a version control system. [7](#)
- Scrum** An agile development method for software-development, based on incremental delivery. [5](#), [6](#)

1 Introduction

1.1 Task Description

The purpose of this project is to create a system that will replace the Excel spreadsheets used by the teachers previously to organize assignment submissions and peer reviewing. The system will give the users, both students and teachers, an easy to use tool for managing and distributing assignment related information and tasks while also being the same place for submitting assignments and giving/receiving feedback. The teachers are able to see student progress in the course, expected grading based on performance, and export the information to a format that allows for simpler grading of the student. Its main purpose is to solve the hardships we as a group encountered through the courses we had, with a easy to use interface and managing system.

1.2 Goals

1.2.1 Result

The goal for this project is to develop and deliver a working system to replace the one used today with shared spreadsheets for submitting computer science assignment, and performing peer review by the students. The system shall also give the lecturer a complete overview of all submissions and reviews, as well be able to generate a report for each assignment and course.

1.2.2 Effect

The current system used for handling peer reviews between the students is sub optimal, and is a problem both for the students and the lecturers, a new system would simplify this for both sides. A new system should make it seamless for the student to perform a review on another student without having to look through a spreadsheet to find another students work to perform a review, but just click a button, and get a review to perform instantly.

The lecturer will have a plain and simple user interface to get an overview of submissions and perform reviews for all students, on all assignment, in all courses.

1.2.3 Learning

In this project, our learning goals are to gain more knowledge about:

- Working with a bigger project from the planning phase
- Working closely with a product owner
- Programming larger applications in [Go Programming Language](#)
- Working with full-stack programming
- System architecture

1.3 Project restrictions

The product owner want to have a working system delivered by the 16th of May, that can be used for the next semesters courses.

1.3.1 Technological

The product owner wishes to use [Go Programming Language](#) as the main language for development for this project. Go has to be used for the back-end part of the system, with no specification on the front-end. The project has some technological restrictions in the description of it, made by the product owner, shown here:

Back-end Has to be programmed with [Go Programming Language](#).

Infrastructure The system has to be deployed on OpenStack, using Docker as the chosen container technology.

Database Has to be local, using containers to serve it.

1.4 User base

The product is a system to be used by lecturers and students, for their Computer Science courses, where they want to customize their submission forms, and optionally use the peer review feature for their students.

Teachers will be the main user for this system, as they will be administrators for the system. They will use the system to create courses, assignments, submission and review forms, and generate reports from assignments.

Teaching assistant will use the system as the teachers, but with fewer privileges.

Students will use the system to submit their assignments and see details about deadlines for assignment, and perform reviews on their fellow students in the course.

1.5 Development framework

In this project, we planned on using an agile method for developing. For more details on the development, framework see chapter [2](#) Development Process.

1.6 Project Organization

1.6.1 Project members

The members partaking in this project are from different programs, where two is from [BPROG](#) and one from [BIDAT](#) at NTNU i Gjøvik. In terms of our education, there are some courses that benefit us in regard to this project.

Front-end development

- IMT2291 - Web Technology
- IMT1362 - Experience Design

Back-end development

- IMT2681 - Cloud Technologies
- IMT2571 - Data Modelling and Database Systems
- IMT2021 - Algorithmic Methods

Operational

- IMT2282 - Operating Systems

Documentation and report

- IMT2243 - Software Engineering

1.6.2 Responsibilities and roles

The responsibilities and roles for the group was setup as seen in Table 1.

Name	Role
Christopher Frantz	Product Owner
Johan Aanesen	Project leader
Brede Fritjof Klausen	Member and QA tester
Svein Are Danielsen	Member

Table 1: Group members & roles

1.6.3 Why we chose this assignment

The group had several reasons to why we wanted to do this assignment in particular. Through the courses of our bachelor degrees some of the assignments required peer reviewing as an additional task after the assignments were delivered. This was organized through a spreadsheet which made it difficult to see the overall progress of the reviews done, and who has done how many. There were always some students who did not perform their reviews and as such, some students did not receive the reviews they should. There was also cases of students that would say they reviewed another users assignment, but the user never received any feedback. This has been a continuous hardship throughout the courses and the group thought it would be beneficial to create a web based solution that streamlines this process.

This assignment is a bit special compared to a traditional bachelor assignment where you deliver a thesis based on research to see if a task is possible and feasible to do. Instead we are delivering a functional product that will be used in future courses, and already has been tested on this years IMT3673 Mobile Programming, IMT4307 Serious Games and IMT2531 Graphics Programming. The group wanted to create a system from the bottom and make it usable for both students and the teachers. The group members are all going into software consulting and development after the studies, so much of the incentive was to take a project from start to finish and deliver the working system.

After the first initial meeting with the product owner, the group confirmed the overall vision of the project with our own thoughts and vision of what issues from the past courses that we could solve with this system. A lot of the motivation for this project came from wanting to rectify the negative feedback we had from the previous delivery system in place.

1.7 Report layout

Introduction Describing the thesis, with goals, restrictions, project details and organisation.

Development Process Describes the development method used for this projects and the tools used for development.

Requirements Shows set requirements for the system and use cases.

Technical Design Gives an insight about the system overview, architecture, database and [UI](#) design.

Implementation Shows how the system is implemented, with different aspects of the system.

Deployment Shows how the system is deployed for using it.

Discussion Shows results, and choices made under the development.

Conclusion A short closing chapter to present evaluation of the group work, and closing words for the thesis.

2 Development Process

2.1 Choice of method

The group has developed a system for managing Computer Science assignment for a four to five month period. The group is composed of three developers and a product owner, that is a highly technical competent person, that will be a part of the team developing this system.

The product owner has his office at the university, and wish to be closely involved in the development, with rapid communication and discussion. This made the development method to an agile approach, where the product owner can quickly come with input and requests for new features, if it was needed.

The project had a tight schedule to create an usable version of the system, that was used in a course at the same time as the development. Because the product owner is a lecturer in the same course, he wanted to use the system for gathering assignments and peer reviews from the students. The system was deployed in different iterations, with different features developed in between them.

2.1.1 Argumentation

With the lack of specified requirements from the start of the project, any non-agile development methods can be excluded as Waterfall, and [Rational Unified Process](#) as it seems unnecessary complex, with regards to our team size. And the amount of time that would have been used to plan the project, would not make it possible to develop any early versions for live testing with a course.

The group has chosen to look at three development models that seems to be relevant for this assignment: [Kanban](#), [Scrum](#), [eXtreme Programming](#).

Kanban

With [Kanban](#) [1] the group is always informed on the amount of tasks that is ongoing at the same time. This secures having unfinished tasks in production like documentation, and secures more [code review](#) in the process.

A [Kanban board](#) will also give a good overview of tasks that is up next, ongoing, ready for code review or finished. The board can be split into different columns, to show what state a task is currently in, table 2 shows how our table is arranged.

As the [backlog](#) can be prioritized for the next tasks to be developed, with a *Up Next* column, the product owner can change the priority for tasks. And this makes [Kanban](#) very applicable for this projects, as the product owner wants different features in the final system.

The group has previous experience with using [Kanban](#) for development, especially performing [code review](#) continuously.

Scrum

[Scrum](#) [2] is an agile development model, that allows rapid releases, and fills most of the groups requirements for choosing a model. It allows to start early on with the development,

without having the requirement specifications finished.

Scrum has different roles for the team: Product owner, Scrum Master and Scrum Team, which would fit our development team well.

Since the project leader will have some absence geographically half way into the project, doing a full approach of [Scrum](#) will not fit our team. But we will take aspects from [Scrum](#) sprints, with larger periods with goals to complete bigger features, can be seen appendix C. And smaller periods, weekly, where we work on tasks from week to week. Starting the week with meetings and planning, and finishing the week with bug-fixing and finishing up code.

eXtreme Programming

[XP](#) has five basic values; *communication, simplicity, feedback, courage* and *respect* which fits well into the project, when communication and feedback is an essential part of it. In this model the developers needs to be receptive to changes under the development, as the product owner is in contact with the developers all the time [3].

Pair programming is also a part of [XP](#), where two and two developers works together, where one writes code, and the other observes. As our team consists of three people throughout the project, this becomes a bad choice as a model.

2.1.2 Conclusion

There is one development method that sticks out, that has the characteristics wishes to use. The product owner indicates to rapid development, with frequently contact, that suggest that the project group should choose an agile method. The assignment is presented with some main features, and is therefore natural to pick a method with rapid development.

The group wanted to use mainly [Kanban](#) with some aspects from [Scrum](#). From Kanban we used [Kanban board](#), and a [backlog](#) open for the product owner. From Scrum we want to adapt the sprints to periods, where we divided the development time for the main features into three main periods, and had a variation of weekly sprints. Where we started the weeks with meetings with the product owner, the supervisor and then within the group. Start developing new features from Monday, work throughout the week, and finish up ongoing tasks on Fridays, or fix bugs found throughout the week.

2.2 Execution of the method

Weekly meetings

On the weekly meetings with the product owner we started with showing and discussing newly implemented features to the product owner, discussing how they have been implemented and design decisions. Then going on to the next features in the backlog, and/or talking about new features, that the product owner would have ready for the meeting, or think about under the meeting.

After the meetings the group would discuss internally what tasks we were currently working on, and what tasks we would go on after. Then delegate tasks, and discuss how to approach difficult tasks we were facing, to get a knowledge about what the rest of the group was working on and let every member get a whole overview of the projects progression.

Workflow

Under the development the group used a workflow called *Git Feature Branch Workflow* [4]. This works the way that all new features is made in a new branch, that then is merged with the *master* branch through a [pull request](#). This secures that the *master* branch always contains production-ready code, and does makes it easier for the developers to perform [code review](#).

[Code review](#) were performed on the [pull requests](#) by the other group members, looking for possible bugs, and checking if the feature implemented is working as it should. By testing the new feature on different systems, we could also test system compatibility for the feature. Only when all of the other group members had approved the [pull request](#), it would be merged into the master branch.

Meeting with supervisor

The group had weekly meetings on Mondays with the supervisor, for updating him about the project, as well to discuss the project, and ask for advice on less technical tasks with the project. As the supervisor also is a highly technical competent professor, he could come with advice from another point of view.

2.3 Tool usage

The group has agreed on different tools that will be used for the project and the development. The tools selected was chosen so every group member had an overview of what the others where working on and keeping information of the development streamlined. The tools used was necessary in terms of getting the most out of the agile process, keeping an overview of progress as well as detecting irregularities under the development process.

2.3.1 Selected tools

Trello

Trello¹ is a collaboration tool used to organize projects into boards. We have used Trello as our [Kanban board](#) and organized tasks for the development. The setup for our Trello-board can be seen in table 2.

Column	Description
<i>Backlog</i>	Contains all untouched user stories. These are made by the product owner, or the group; based on requirements by the product owner. All user stories starts in this column.
<i>Up Next</i>	User stories picked out by the project leader or the product owner, that the other team members pick out when available. This column had a maximum of 6 user stories.
<i>In Progress</i>	User stories under development. A single or multiple group members is assigned to a single user story. This column had a maximum of 3 user stories.
<i>Peer Review</i>	User stories that are marked as done by a developer, and needs to be reviewed before it can be put into production.
<i>Code Done</i>	User stories that are no longer in progress, reviewed and put into production.

Table 2: Trello-board description

¹<https://trello.com/>

GitHub

GitHub² was chosen by the group as everyone is comfortable with it and has experience from using it from before. GitHub was used for version control and storage for the source code under development.

GitHub has an issue-tracker built-in where the developers or the product owner could add any kind of issues, bugs or new features.

Toggl

Toggl³ is a time tracking tool, that offers both a web-site and mobile-application for tracking time. This was used to count hours logged by the developers, so the group members could see their own time, as well as the others.

OpenStack

OpenStack⁴ is a web-based service for controlling *instances*, where we put our production system, and several testing systems. This was provided by the school, NTNU. Our usage of OpenStack can be read at section 6.1.

Discord

Discord⁵ was used as the main communication channel outside meetings. Discord is a communication application, with a chat feature with many functionalities. Discord allows the users to create their own servers for free, where we had only the three developers, as well as the product owner in different topic based chat-channels. In the channel with the product owner we could contact him for questions, or vice versa, and this was highly used throughout the project.

Google Drive

Google Drive⁶ were used for storing documents that did not fit into the repository on GitHub. Mostly used for storing the document for the project plan, figures, diagrams and spreadsheets from user feedback 7.3.

JetBrains IDE

For the coding the group used two different IDE; JetBrains IntelliJ⁷ and GoLand⁸. IntelliJ is mainly an IDE for Java-development, but supports other languages like Go. GoLand is the equal IDE for IntelliJ, but focused on Go. This environment gave us live feedback on syntax correction and variable referencing within the whole project.

²<https://github.com>

³<https://toggl.com>

⁴<https://openstack.org>

⁵<https://discord.gg>

⁶<https://drive.google.com>

⁷<https://www.jetbrains.com/idea/>

⁸<https://www.jetbrains.com/go/>

3 Requirements

3.1 Use cases

Since the development method was so agile, a lot of the use cases was developed along side the process. Further in this chapter we will elaborate several use cases in high level, with a few in detail, to give an overall better overview of the functionality we have developed in the system.

3.1.1 Use case-diagram

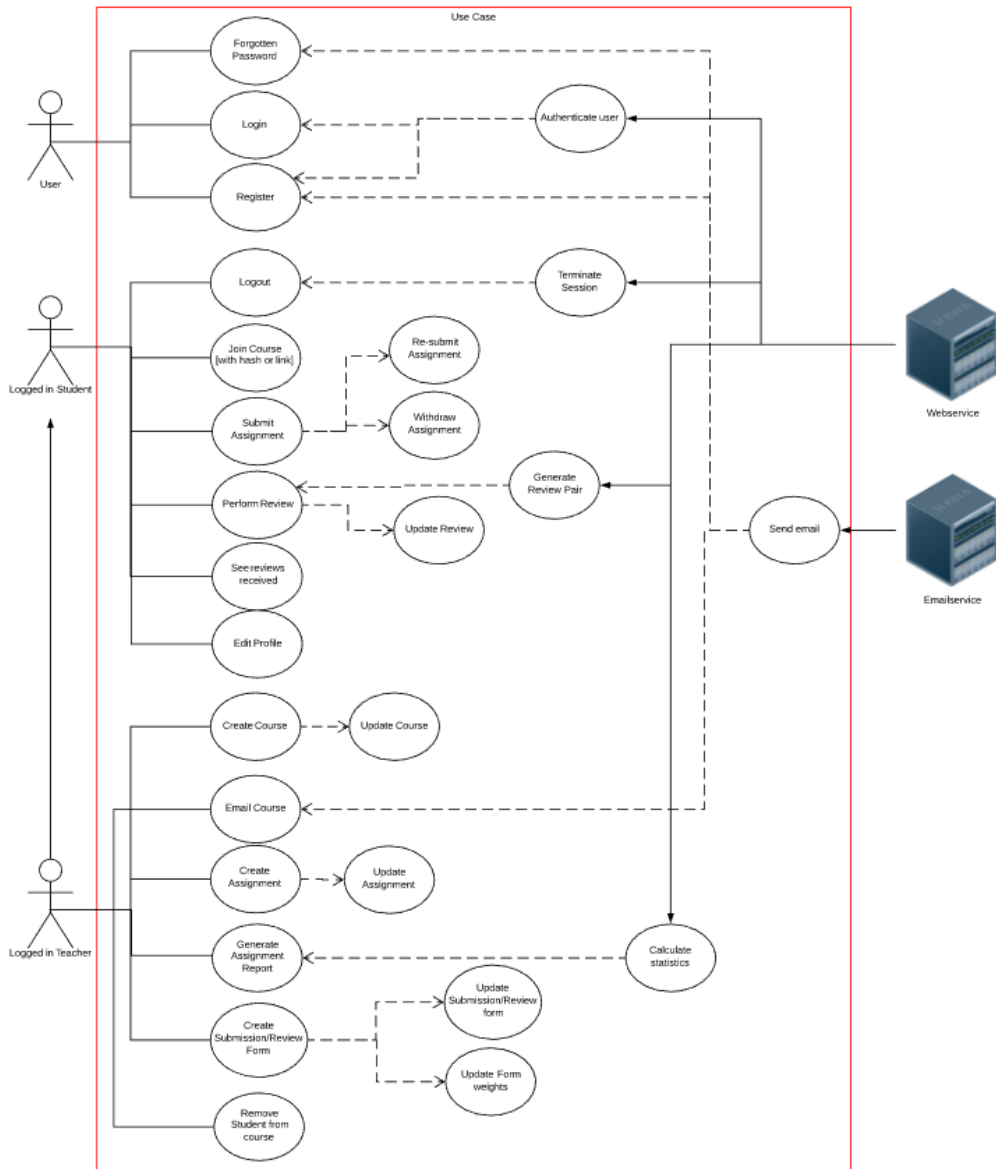


Figure 1: Use Case Diagram

3.1.2 Use case descriptions

Use Case	Register
Actor	Student, TA or Teacher
Goal	Create new user with name, email and password
Pre-condition	The email is valid and does not exist in system database
Description	User fills in name, email and password and clicks on the register-button. If the email belongs to the user s/he will receive an email to confirm the new user creation.

Table 3: Use case for registering new user

Use Case	Log in
Actor	Student, TA or Teacher
Goal	Authenticating to the system with email and password
Pre-condition	User is already registered with their email
Description	User fills in email and password and clicks on the login-button. If the user already is logged in, and opens the website on a fresh tab, the user will be automatically logged in, if the last visit did not exceed the session-time.

Table 4: Use case for authenticating

Use Case	Log out
Actor	Student, TA or Teacher
Goal	Log out of the system
Pre-condition	User is already signed in
Description	User clicks on log out-button. The user will be logged out

Table 5: Use case for logging out

Use Case	Forgotten password
Actor	Student, TA or Teacher
Goal	Create new password
Pre-condition	User is already registered with their email
Description	User clicks on forgotten password and enters the relevant email. The user will then get an email to change the password.

Table 6: Use case for forgotten password

Use Case	Join course through URL
Actor	Student, TA or Teacher
Goal	Join course on the site through URL, for unlocking access to assignments
Pre-condition	User has the join-course URL
Description	The user pastes the URL into their browser, and if they are logged in, they will successfully join the course. If the student is not logged in, they will get sent to the login-page. Authenticating them self with successfully join the course. And if the user hasn't registered an user, the user can register a user and gets the email to confirm and join course.

Table 7: Use case for joining a course through URL

Use Case	Join course with hash
Actor	Student, TA or Teacher
Goal	Joining a course on the site, for unlocking access to assignments
Pre-condition	User has the course hash
Description	User goes to front page of the website, and enters the course hash into the input field right below the menu bar, and clicks the appended button to the field.

Table 8: Use case for joining a course with course hash

Use Case	Deliver assignment
Actor	Student
Goal	Submit assignment
Pre-condition	User is student in course with assignment due
Description	User goes to the course page and select due assignment under the assignment tab. And then fills out the submission form for assignment and clicks on deliver-button

Table 9: Use case for delivering assignment

Use Case	Re-deliver assignment
Actor	Student
Goal	Update delivered assignment
Pre-condition	User is student in course with assignment due and has delivered assignment
Description	User goes to the course page and select delivered assignment under the assignment tab. User changes something on assignment and then clicks on re-deliver-button to update assignment

Table 10: Use case for re-delivering assignment

Use Case	Withdraw assignment
Actor	Student
Goal	Delete delivered assignment
Pre-condition	User is student in course with assignment due and has delivered assignment
Description	User goes to the course page and select delivered assignment under the assignment tab. And then clicks on withdraw-button

Table 11: Use case for withdrawing assignment

Use Case	Perform review
Actor	Student
Goal	Perform review on another students assignment
Pre-condition	User is student in course with assignment delivered before deadline
Description	User goes to the course page and select delivered assignment under the assignment tab. The user clicks on reviews-tab, and selects a review to do on another students assignment and then fill out the form and submits

Table 12: Use case for performing review

Use Case	Update review
Actor	Student
Goal	Update performed review on another students assignment
Pre-condition	User is student in course with assignment delivered before deadline and has performed the review.
Description	User goes to the course page and select delivered assignment under the assignment tab. The user clicks on reviews-tab, and selects performed review. The user then updates the review form and submits

Table 13: Use case for updating performed review

Use Case	Edit profile
Actor	Student, TA or Teacher
Goal	Edit the profile, add/change secondary email or change password
Pre-condition	User is logged in
Description	User goes to the profile site and clicks on edit. The user can add/change secondary email and change password

Table 14: Use case for editing profile

Use Case	Create course
Actor	TA or Teacher
Goal	Create a course
Pre-condition	User is a teacher
Description	User goes to dashboard and on the site for courses. User clicks on new-button and fills out the data and creates the course

Table 15: Use case for creating course

Use Case	Update course
Actor	TA or Teacher
Goal	Update a course
Pre-condition	User has created a course
Description	User goes to dashboard and on the site for courses. User clicks on a created course and enter the changes and submits

Table 16: Use case for updating course

Use Case	Email course
Actor	TA or Teacher
Goal	Email all students in a course
Pre-condition	User has created a course and there is more than zero students in course
Description	User goes to dashboard and on the site for courses. User clicks on email students-button on a created course. The user fills the subject and message and sends the email.

Table 17: Use case for emailing students in a course

Use Case	Create assignment
Actor	TA or Teacher
Goal	Create an assignment
Pre-condition	User has created a course
Description	User goes to dashboard and on the site for courses. User goes to the assignment site and clicks on the new-button. User fills in fields and submit.

Table 18: Use case for creating an assignment

Use Case	Update assignment
Actor	TA or Teacher
Goal	Update an assignment
Pre-condition	User has created a course and an assignment
Description	User goes to dashboard and on the site for courses. User goes to the assignment site and clicks on the new-button. User fills in fields and submits.

Table 19: Use case for updating an assignment

Use Case	Generate submissions report
Actor	TA or Teacher
Goal	Generate report of all submissions from one assignment
Pre-condition	User has created an assignment and students has submitted their assignment
Description	User goes to dashboard and on the site for courses. User goes to the assignment site and clicks on the see-submissions-button. User clicks on generate report and opens file in excel office.

Table 20: Use case for generating report of all user submissions from one assignment

Use Case	Create submission/review form
Actor	TA or Teacher
Goal	Create a submission/review form for an assignment
Pre-condition	User is a teacher
Description	User goes to submission/review form site and clicks on new, then customizes the form and submits

Table 21: Use case for creating a submission/review form

Use Case	Update submission/review form
Actor	TA or Teacher
Goal	Update a submission/review form
Pre-condition	User has crated a submission/review form and form is not assigned to an assignment
Description	User goes to submission/review form site and clicks on existing form, then updates the form and submits

Table 22: Use case for updating a submission/review form

Use Case	Update submission/review form weights
Actor	TA or Teacher
Goal	Update weights on submission/review form
Pre-condition	User has crated a submission/review form
Description	User goes to submission/review form site and clicks on existing form, then updates the weights and submits

Table 23: Use case for updating submission/review form weights

Use Case	Remove a student from course
Actor	TA or Teacher
Goal	Remove a student from a course
Pre-condition	User is teacher of a course that has more than zero students
Description	User goes to manages student site and chooses the course and student, then clicks on remove-button

Table 24: Use case for removing student from course

Use Case	Update a student's password
Actor	TA or Teacher
Goal	Update a student's password from a course
Pre-condition	User is teacher of a course that has more than zero students
Description	User goes to manages student site and chooses the course and student, then clicks on change password-button and chooses to use the auto generated password or a custom password

Table 25: Use case for updating a student's password

Use Case	Update the FAQ
Actor	TA or Teacher
Goal	Update the FAQ
Pre-condition	A FAQ exists
Description	User goes to FAQ site and clicks on edit-button, then edits the MD text and submits

Table 26: Use case for updating the FAQ

3.2 Requirements

3.2.1 Functional requirements

Remove Students The teacher should be able to remove students from the course, e.g., in case of an abandoned account, duplicate sign up, etc.

Confirm Delivered Assignment The student should get an explicit feedback that the submission was successful (to feel confident that everything works).

Deleteable User Submission Students should be able to withdraw their delivered submission.

Default Weight And Input Type When starting to edit a form, the user should be able to

specify the default type of entry (since it often repeats) and default weight for any newly created field.

Order List Allow lists to be ordered by clicking on columns. List of interest: Participants, submissions, reviews, etc.

Manual Activation Of Review Instead of relying only on the system for activating reviews, it would be good to be able to override the system and manually activate the reviewing process instead.

Manage Assigned Reviews The teacher(admin) should be able to view the assigned reviews to individual students and modify those. For example in the case of empty repository or test users, those should be excluded from the reviewing process - which may require manual reassignment.

Allow For Minor Changes In Forms Allow for minor modifications of forms that do not involve structure. For example, teacher(admin) should be able to fix some typos or adjust descriptions. The teacher should also be able to activate/modify weights.

Modify Default Settings For Weights Submission forms generally don't have weights, whereas review forms effectively always have weights. The current settings can lead to mistaken activation of weights when editing assignments.

Log For Submission And Review Add logging of submission and review submissions. This could make retracing/following easier. Looking ahead: Provide hooks for logs to discord.

Optional Form Input Have an option to make an item optional selectively in the forms would be good.

Show Assignment Status Indicate the assignments that are closed. and open. Perhaps an additional review label would be good for assignments that have a review stage.

Grab A Review Let users "grab" the reviews them self, i.e show the user how many minimum reviews s/he has to do and provide an option to "grab" a review to do. Also have the option to do more reviews than needed.

Display When A User Joins Course In the course participants tab, it would be good to see when a user joined the course.

See Users Performed Reviews In addition to being able to see the reviews a user has received for their submission, it would be good to have a separate overview of the reviews a user has done.

Display Statistics In Percentage Display standard deviation in calculated percentage for reviews, both in individual view and summary sheet.

Submission Status Show if an assignment has been delivered by the user or not.

Markable Textbox for repository URL When delivering the repository URL, make it a link or button to easier open.

Change Review Add possibility to edit delivered review before deadline is over.

3.2.2 Non-functional requirements

Branding Brand the system properly. Clearly state what the name of the system is.

Extendable Make the system able to be extended with future development.

4 Technical Design

4.1 System overview

In this chapter we will go through the technical aspects and design of the finished system as well as the smaller parts of the system. In Figure 2 you can see how the system is deployed on the OpenStack infrastructure. The system is deployed through separate containers using Docker-compose in Docker on a Linux virtual machine, and can access the outside world through the security group settings in place. Docker-compose lets us have a shared virtual network between the containers so they can communicate with each other, this is done by specifying which ports each container can be accessed through. The ports are only available locally unless the security group has opened them for the external networks.

Because of the nature of Docker and running everything in containers, the high level system architecture for the project is a service oriented architecture. In the lower level architecture, inside the containers or services, we used the Model-View-Controller architecture.

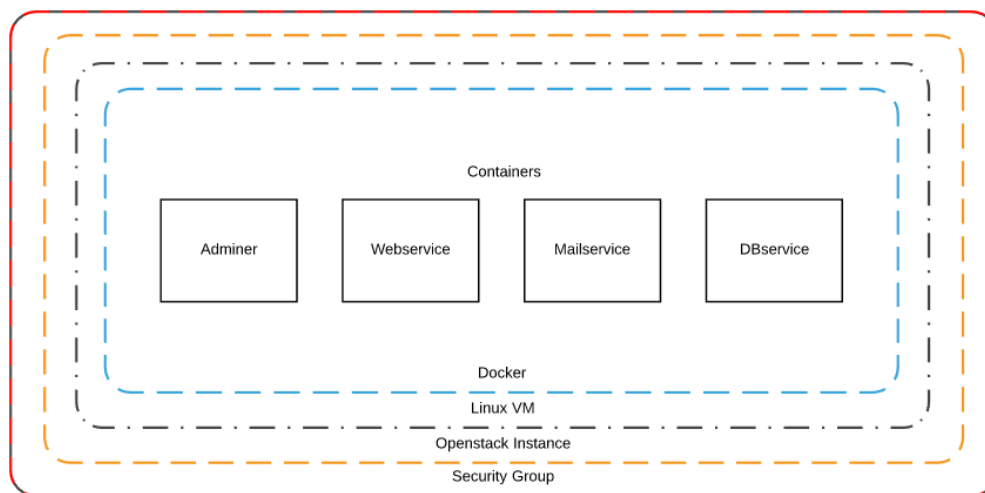


Figure 2: System View

4.2 Architecture

4.2.1 Service Oriented Architecture

Service oriented architecture [5] means that the system is run in services and can be interpreted like the system is divided into different sub systems that contain themselves. In our project we are currently running four services to maintain the system functionality, although there were six. We are currently running the webservice, mailservice, dbservice and Adminer. Adminer is just a standalone 3rd party package for a simple GUI interface to interact with the dbservice. dbservice is the MySQL image/server running in a container, keeping all the tables and data for the whole system. The webservice serves all the pages to the user, handles

all the requests made, and is the core of the system. The mailservice will act upon receiving specific payloads, and send emails to the users specified therein.

All our services are built through Docker-Compose which again builds their correlating Dockerfiles to ensure that they are built correctly and maps to the correct ports on the internal network.

The two services that we removed were the schedulerservice and the peerservice. The main objective of these were to schedule tasks ahead of time, in reality it was to schedule submission deadlines and run the peer-review distribution algorithm at the deadline.

4.2.2 Model-View-Controller

Before we started coding we already decided on using the Model-View-Controller (MVC) pattern [6] for the architecture of the system. This is implemented in both the webservice and mailservice, although a bit different from each other. The idea of the pattern is that the model holds the data, the view displays the data, and the controller updates model with new data, or inserts data from the model to the view. It is somewhat close to a three layered model.

Model

The model part holds all the structs or classes, and is where the data exists. The model layer is the bottom layer of this architecture, and the layer that communicates with the database. In our project, and especially the webservice, the model layer was divided further into a service layer and a repository layer as you can see in Figure 3. The repository layer holds all the functions needed to update, insert and fetch data from the database, by taking data from a model, or populating one, while the service layer is used to hold all the pointers to the repository layer's functionality. The service layer acts like a tunnel for the information stream, and gives the controller access to all its functionality through one pointer.

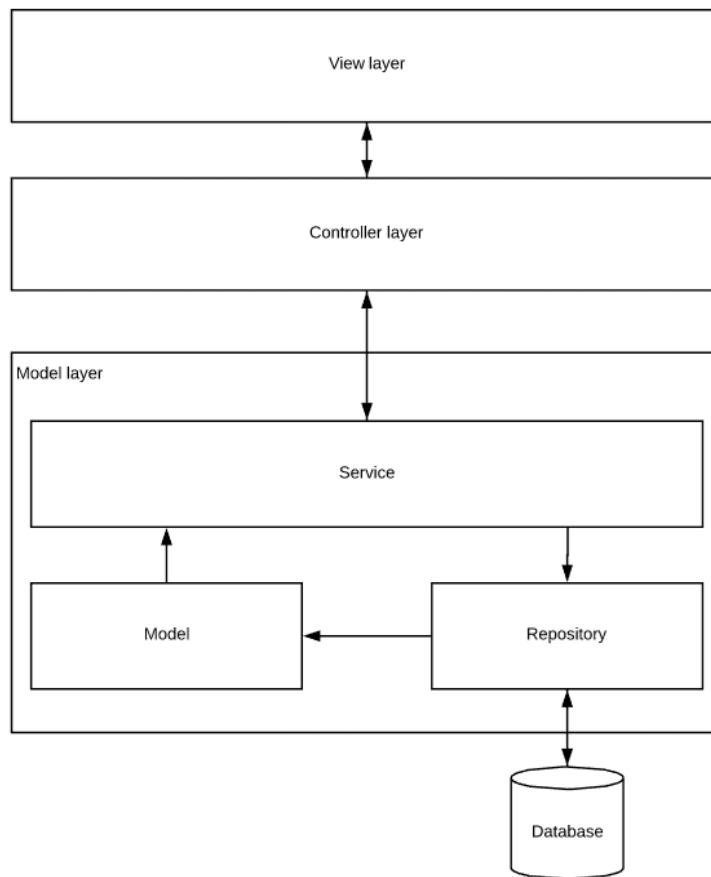


Figure 3: Webservice MVC

View

The view displays the web page to the users, this can be seen as all the template files in our project. Through the `http/template` package in Go, the template files can have different template tags that allows the controller to insert data into it. It is also through the view that the user can interact with the site through forms and input fields. The information provided in such fields are sent to the controller through POST requests when forms are submitted.

Controller

The controller has a tight connection with both the model and the view layer, because it controls the data flow between them. In our project this is the layer where all requests are directed. For GET requests, the controller usually takes data from the model layer, and inserts it into the view layer before it is presented to the user with all the correct information. For POST requests, the controller is receiving data from the user or another service, and updates the model layer with this new information.

4.3 UI Design

4.3.1 Graphic Design

The User Interface was designed with simplicity and clarity in mind. We used Bootstrap 4 to create a clean and responsive design throughout the system. The interaction between the system had always some connection to the courses or assignments, and therefore we used a card/block styled look that each class or assignment was contained inside. This makes the interface intuitive because all the information related to a specific course is inside each own block and naturally further information available through the buttons that resides inside it. Through badges and meaningful coloring schematics we show basic information that is relevant to each assignment or course inside the course blocks, while the detailed information is displayed in the designated pages. An example of this can be seen in figure 4.

For the button choices we wanted a blended design where the buttons does not stand out, but still were obvious that they were buttons. For the most important buttons, like the submission buttons, we would add more color to them, but otherwise we would place them in strategic positions that instantly lets the users know that they were indeed, buttons. Take an example from Figure 4 where the important information lies in the badges and the deadlines, while the button is blended into the background, yet it is very apparent that it will lead the user to the detailed assignment page.

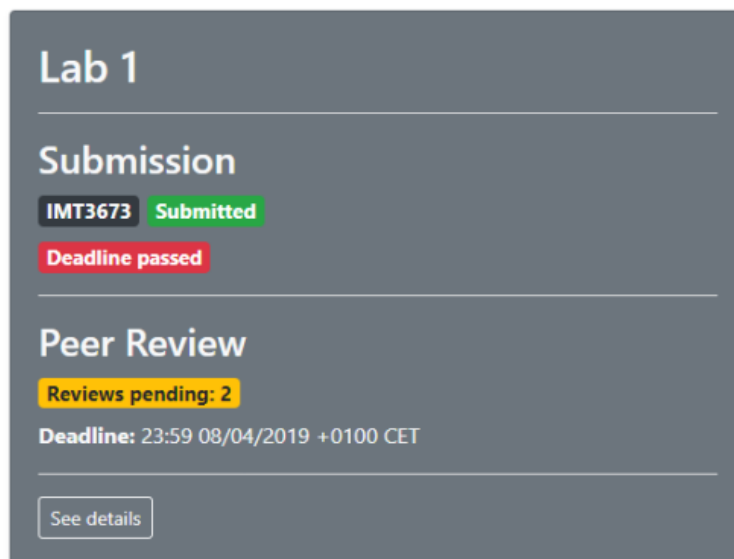


Figure 4: UI Badges in Card example

Screenshots of the final system is added in Appendix G.

4.3.2 Interaction Design

The group had a vague idea of how the website should look like and how the users would navigate it, therefore we created a site map as seen in Figure 5. This initial design allowed us to speed the development at the start of the project, because we knew at least the basic pages of the site and what functionality they required. The actual pages changed during the process according to the always changing functional requirements. The most major changes to the initial site map was to implement an admin dashboard to control all the admin functionality,

instead of having it appear inside every sub page. Another large part of the change was taking out all of the automatic validation pages.

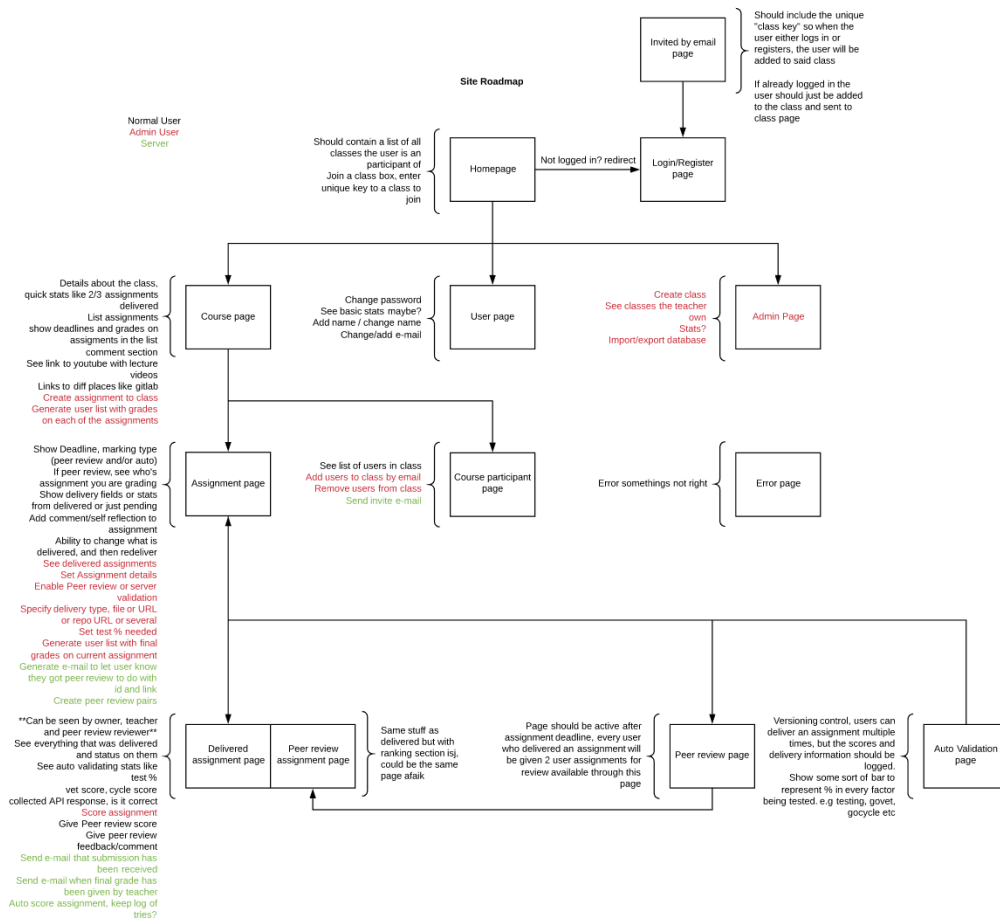


Figure 5: Initial Site Map

The final sitemap is as you can see in Figure 6 largely modified to fit all the admin/teacher functionality in one convenient place. A reason to structure the admin functionality into the admin pages was to avoid cluttering of the template files used in the rest of the pages available to the user.

Another reason for separating the admin functionality into the admin pages, was to not clutter the normal pages with buttons. Most of the functionality we developed were for the admin users, and it might be very unnatural to keep in certain pages because it would be too much buttons or interaction points in one page. This had an impact on the graphic designing as well because we always wanted the website to be easy to understand for all the users.

All the pages has an navigation bar at the top to ensure that the users always can go back to the front page, and where it is natural, back buttons on the page itself to go back one step.

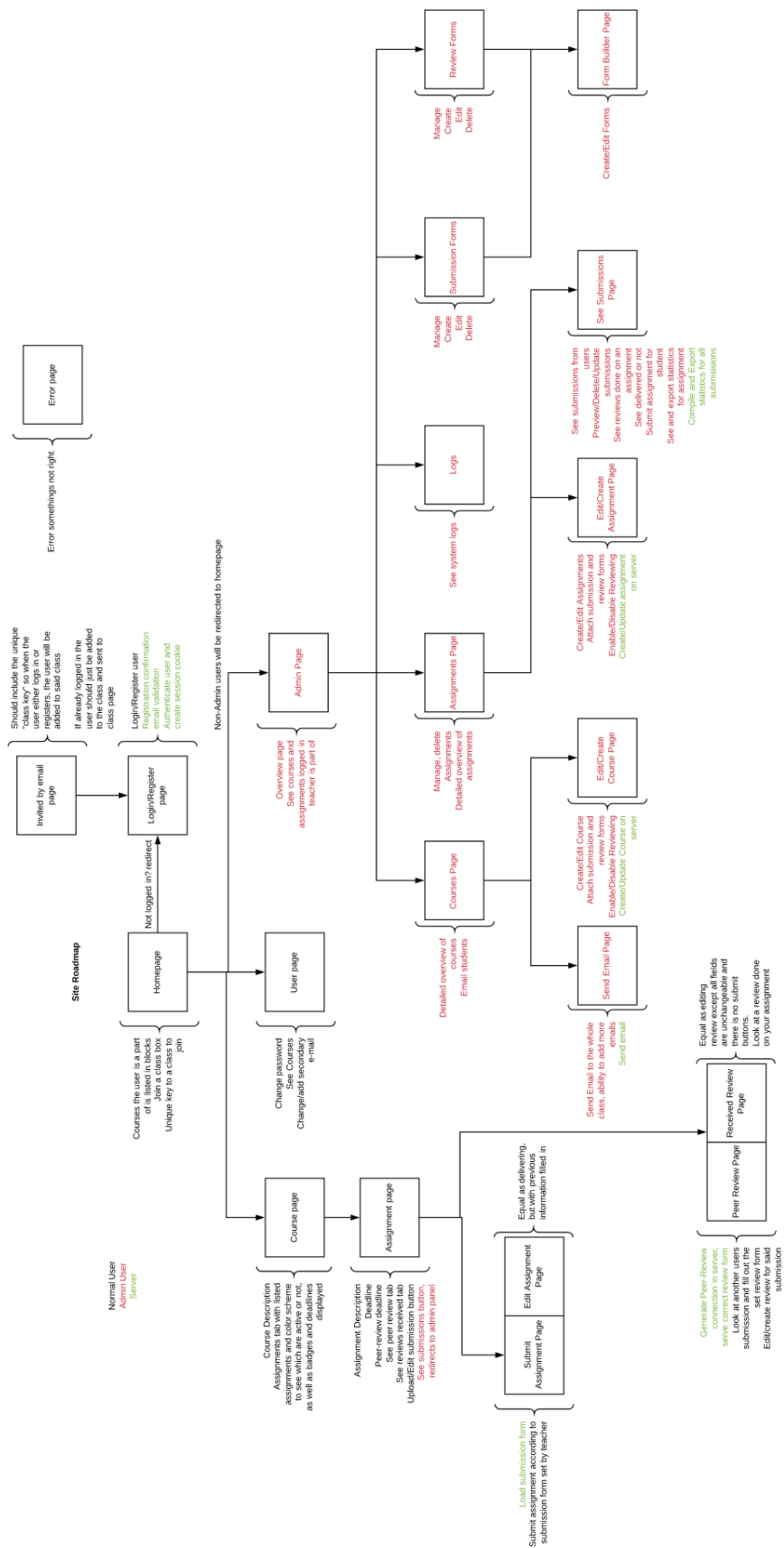


Figure 6: Final Site Map

4.4 Database design

Through our studies it has been a great focus on using relational databases and normalization of data. MySQL was the natural choice to implement the database in, because all the group members has gone through several courses directly or indirectly using this technology. It has the ability for data transactions, which allowed us to program for redundancy and stability.

The first database design was initially planned for the first milestone cycle, where we tried to figure out the basic tables needed to implement the users, courses, assignments, submissions and peer review. Figure 7 shows how we designed it. We did not want to spend too much time at the initial model because of the iterative approach to the system development, but it was created as a basic database structure to continue building on.

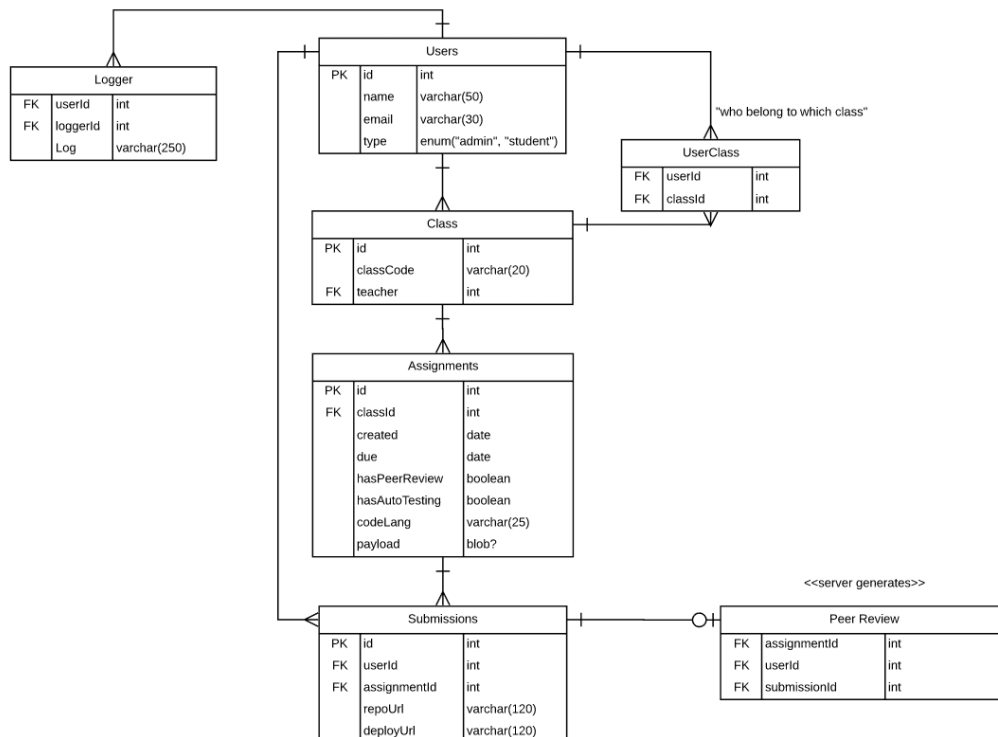


Figure 7: Initial Database Design

Through the development the database design changed quite a lot to fit the needs of the application, and it has several times been refactored and made more efficient. Figure 8 shows the final design of the database, and all its connections. The Forms part of the database structure needed to be modified in so that it was both flexible in terms of different types of input structures inside the form. We wanted the teacher to create forms much like Google Forms¹. As you can see both the review forms and submission forms use the forms table to store the actual data of each form in the fields table, making the flexible and reusable approach we wanted possible.

The user_reviews table and the user_submissions table are very similar in both appearance and the data they contain, but we decided that because of the nature of the different type of

¹<https://www.google.com/forms/about/>

information they contain, they should stay separated.

We have also added a `user_pending` and validation table to ensure that new users must register with valid emails, this was implemented as a result of actual users unable to write in their correct emails and thus being unable to log back in at a later time.

The logging table was also extensively worked on to ensure that each log would contain all the necessary data it might need to display accurate and informational logs for the admin users to take advantage of.

4.5 Project/File structure

The project and file structure is largely governed by the container structure at the high level. In figure 9 you can see how our project folder is split into the different containers/services, which therein have their own structures.

The `dbservice` structure has a volume mounted data folder in order to keep the data even if the container is shut down. Further it includes a Makefile which tells Docker to specifically mount the volume, and insert the `database.sql` and `database_insert.sql` files into the container on build time in order to setup the database correctly.

The other services are built on the same server layout, with a `main.go` executable that initializes database connection and further utilizes the `server.go` files in order to launch the http endpoints and router functionality.

The `mailservice` has no inside folders, but has the functionality separated into different files. It has a variant of the MVC architecture, but since it is setup as an API endpoint for the webservice to communicate through, it does not have a View layer.

Unlike the `mailservice`, the `Webservice` has a lot more functionality and it separates this into related folders. As shown in Figure 3 the model layer is split up as previously described, into the model, service and repository folders.

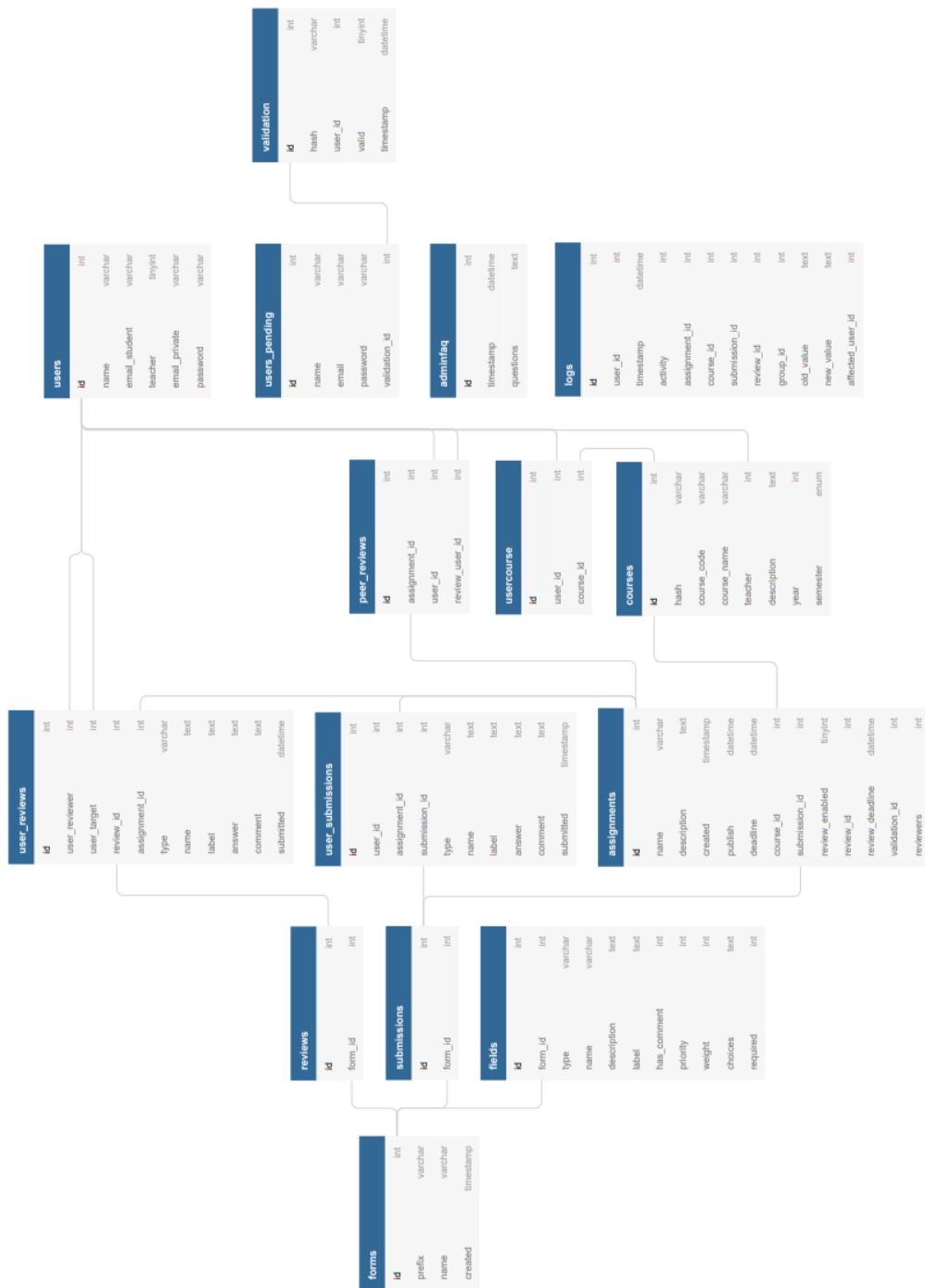


Figure 8: Final Database Design

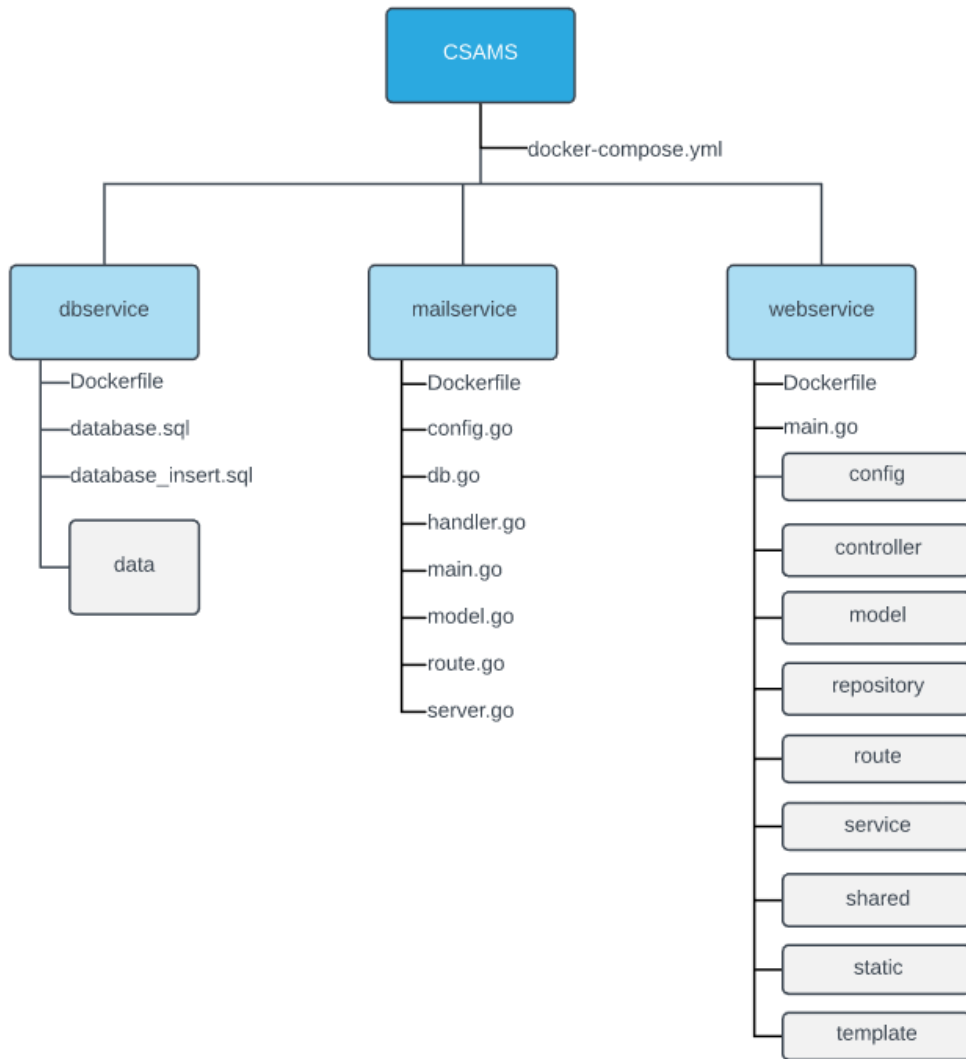


Figure 9: File Structure

5 Implementation

5.1 Web server

We will be showing how the group has used Go's features as a programming language for back- and front-end development. All the code examples is a representation of how we did it in the project, for a more detailed view of how we did it in the production code, see the Git-repository (Appendix A).

Creating a HTTP-server in Go is a simple task [7], as it has a HTTP-package builtin to the standard libraries. All needed is to call a function called *ListenAndServe* from the *net/http* package. This function takes in two arguments: a string, the address for the server, and a handler, that can be *nil*. In the example 5.1 will start a HTTP-server at the localhost with the port 80, without a handler. This example also shows how to create a request-handler for the root URL, sent to a function called *ExampleHandler*.

Listing 5.1: Simple HTTP-server in Go

```
1 package main
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 func main() {
9     http.HandleFunc("/", ExampleHandler)
10
11     http.ListenAndServe(":80", nil)
12 }
```

5.1.1 Routing

In this project, a package called *Mux* made by *Gorilla*¹ has been used for handling the routing for all requests.

Mux allows us to route specific handlers to specific HTTP request methods, as in native Go you would have to condition check for what kind of request method was called for a specific route. This let us route many routes to handlers with a single type of request method. In the code sample 5.2 it is shown how we implemented *Mux* for handling our routes.

Listing 5.2: Mux routing

```
1 import (
2     "net/http"
3     "github.com/gorilla/mux"
4 )
5
6 func main() {
```

¹<https://www.gorillatoolkit.org/>

```

7   router := mux.NewRouter()
8
9   router.HandleFunc("/", ExampleGetHandler).
    Methods("GET")
10  router.HandleFunc("/", ExamplePostHandler).
    Methods("POST")
11  router.HandleFunc("/{message}", HelloHandler).
    Methods("GET")
12
13  http.ListenAndServe(":80", router)
14 }

```

5.1.2 HTTP Handlers / Controllers

All request handlers have two arguments in their functions, a response writer and a request, these functions is used as the controller for the MVC-architecture. The first parameter, the *ResponseWriter* handles all the response data, that goes back to the client that perform the request, like the response header, status code and content. In the sample code 5.3 shows how to set the content type and status code for a request, and output a simple string.

Listing 5.3: HTTP Handler

```

1  func ExampleHandler(w http.ResponseWriter, r *http.
    Request) {
2      w.Header().Set("Content-Type", "text/html;
        charset=utf-8")
3      w.WriteHeader(http.StatusOK)
4      w.Write([]byte("Hello World"))
5  }

```

Mux inside handlers

Mux also gives us a helper function for retrieving parameters in the URL. All values retrieved from the URL will be the a *string* type. And if this needs to be converted to any other types, it can be done with the builtin string converter.

Fetching URL parameters with *Mux* is shown in the code sample 5.4.

Listing 5.4: Using Mux inside handlers

```

1  func HelloHandler(w http.ResponseWriter, r *http.
    Request) {
2      vars := mux.Vars(r)
3
4      message := vars["message"]
5      if message == "" {
6          message = "World"
7      }
8
9      w.Write([]byte("Hello " + message))
10 }

```


5.1.3 Parsing the request body

JSON is one of the most commonly used for sending data over HTTP-requests, as many programming languages has support for parsing and encoding JSON. In the code sample 5.5 shows how we handle JSON data sent over the request body.

On line 16 the empty initialized struct is passed as a reference which lets the decoder write data into it. As the struct is declared with a JSON-tag on line 8 and 9, it lets the decoder reads in the JSON-data and maps it to the related variables.

Listing 5.5: Parsing POST-request body

```
1 import (
2     "log"
3     "encoding/json"
4     "net/http"
5 )
6
7 type ExampleStruct struct {
8     Id int `json:"id"`
9     Message string `json:"message"`
10 }
11
12 func ExamplePostHandler(w http.ResponseWriter, r *
13     http.Request) {
14     decoder := json.NewDecoder(r.Body)
15
16     data := ExampleStruct{}
17     err := decoder.Decode(&data)
18     if err != nil {
19         log.Println(err.Error())
20         return
21     }
22
23     log.Println(data.Id, data.Message)
24 }
```

5.1.4 Parsing form data

Handling data from a form with Go can be done with just fetching the field-names from the request. All values fetched this way will be as a string, and can converted to other types with the builtin packages for string-conversion.

Listing 5.6: Parsing form data

```
1 import (
2     "log"
3     "net/http"
4 )
5
6 func ParseForm(w http.ResponseWriter, r *http.
7     Request) {
8     name := r.FormValue("name")
9     message := r.FormValue("message")
10 }
```

```

10     log.Println(name, message)
11 }

```

5.1.5 View

We implemented a view based on an Go web application example by Joseph Spurrier [8], where the view handles the loading of template files and plugins. This makes it flexible to create several views and reuse files, and keep the file count down. The view is created with the request to handle some base variables. Then the name is declared, which is the template file within the template folder. Declaring more variables is optional, but useful for creating a dynamic page. The *Vars* variable is a map with string keys with *interface* as values, that makes it possible to have all kind of types as values. Lastly the view will render everything, to the *ResponseWriter*.

Listing 5.7: Usage of View

```

1  func ExampleHandler(w http.ResponseWriter, r *http.
   Request) {
2      v := view.New(r)
3      v.Name = "path/to/template"
4
5      v.Vars["Message"] = "World"
6      v.Vars["Todo"] = []string{"Task_1", "Task_2", "
   Task_3"}
7
8      v.Render(w)
9  }

```

5.1.6 Templating

Go's package *html/template*² provides a rich templating language for HTML templating, that gives us the possibility to display data in a structured way.

One of the big benefits of using the native HTML templates from Go, is that is built with security in mind, so everything that is rendered will be automatically escaped.

In the code sample 5.8 we can see how different kind of types can be used for displaying data. This example shows how to print out a single value, and loop over a slice and display the values inside it.

Listing 5.8: Sample of a Template

```

1  <!DOCTYPE html >
2  <html >
3      <head >
4          <title>Go Template Example</title >
5      </head >
6      <body >
7          <h1>Hello {{ .Message }}</h1 >
8          <h2>Todo-List:</h2 >
9          <ul >
10             {{ range .Todo }}
11                 <li>{{ . }}</li >

```

²<https://golang.org/pkg/html/template/>

```

12         {{ end }}
13     </ul>
14 </body>
15 </html>

```

Plugins

Plugins is helper functions used inside templates, and needed because the basic support for handling data inside Go's template does not support enough operations, and does not have any support for checking conditions on different value types. In the listing 5.9 it is shown how a plugin is made, using the function map from the *html/template* package. Several function maps can be created inside a single plugin, for grouping functions maps.

Listing 5.9 shows a simple example for a "Hello World" program with functions map, and listing 5.10 show how it is used inside the template.

Listing 5.9: Template Plugin

```

1  import "html/template"
2
3  func Plugin() template.FuncMap {
4      f := make(template.FuncMap)
5
6      f["HELLO"] = func(name string) string {
7          if name == "" {
8              return "Hello World"
9          }
10
11         return "Hello " + name
12     }
13
14     return f
15 }

```

Listing 5.10: Plugin usage inside templates

```

1  <!DOCTYPE html >
2  <html >
3      <head >
4          <title>Go Template Example</title >
5      </head >
6      <body >
7          <h1>{{ HELLO .Message }}</h1 >
8      </body >
9  </html >

```

5.1.7 Form Builder

A form builder was created from scratch for making forms for the submissions and peer-reviews for the system. It was built with native JavaScript, and have a single dependency (Sortable³) for a drag and drop feature, but will still work without it, but will not support the drag and drop feature. The script is built up with 3 classes, *FormBuilder*, *Form* and *Field*. The

³<https://github.com/SortableJS/Sortable>

FormBuilder-class is a to wrap up all the code into a single class, that initializes the script. The *Form*-class handles all interactive functionalities, and holds all the data for the form. And the *Field* handles each individual field in the form, keep the data, and handles interactions with each single field.

The form builder allows several field-types; *text-field*, *textarea*, *select-bar*, *checkbox*, *URL*, *number-input* and a *paragraph*, which is used for displaying text only [9].

In figure 10 there is a screen shot off how the form builder looks like.

The screenshot shows a web interface titled "Create new Review Form - Example Form". On the left, there is a configuration panel for the form. It includes a "Form Name" field with the value "Example Form", a checked "Enable weights" checkbox, a "Default weight" field with the value "1", and a "Default type" dropdown menu set to "RADIO". Below these are two buttons: "Add a new Field" (blue) and "Submit" (green). The main area displays a "Radio Example" form configuration. It has a "Type" dropdown set to "RADIO", a "Label" field with "Radio Example" (with a "Max length: 256 characters" note), a "Description" field with "Please select a choice, and leave a comment", a "Comment" field with a checked "Enable comment" checkbox (with a note: "Enable this will append a textarea after the field with the option to enter an comment to their answer."), a checked "Make field mandatory" checkbox, a "Choices" field containing a list: "A", "B", "C" (with a note: "Enter each choice on a new line. (Do not use '|' (pipe character), as this is the separator for list)"), and a "Weight" field with the value "1". At the bottom of the configuration area is a "Remove this field" button. A small note at the bottom of the form configuration area reads: "Weights on radio will be calculated linear based on which is selected. Ep. Weight: 5, Choices: [A, B, C, D, E], Weights: [A:1, B:2, C:3, D:4, E:5]".

Figure 10: Form Builder screen dump

5.2 Database

Go does not have any SQL drivers in their standard packages, but it has a list of recognized public drivers⁴ where we decided on using the [go-sql-driver/mysql](https://github.com/go-sql-driver/mysql) driver which is included in, and pass the compatibility test suite⁵.

The database is built on the [MySQL 5.6 Docker Image](#) and settings further customized through our docker-compose file. Further elaboration on this can be read in Chapter 6.

Connecting to the database is done through simple code

Listing 5.11: MySQL Database Connection

```

1 import "database/sql"
2 import _ "github.com/go-sql-driver/mysql"
3
4 db, err := sql.Open("mysql", "user:password@dbname
  ")

```

The connection is always initialized through the main.go file in each service using the database, and fetched through the different controllers and functions. This makes the system use as few

⁴<https://github.com/golang/go/wiki/SQLDrivers>

⁵<https://github.com/bradfitz/go-sql-test>

as possible connections to the database, so it will not crash because of max connection parameters inside the database service.

The connection lets us run queries, transactions and executions on the database through the Go code, and is a simple way of handling data throughout the system. Below we will showcase different examples of how we update, insert and fetch data from the database.

Listing 5.12: MySQL Update Example

```

1 // UpdatePassword for a user
2 func (repo *UserRepository) UpdatePassword(id int,
   hashedPassword string) error {
3     query := "UPDATE users SET password=? WHERE id=?"
4
5     tx, err := repo.db.Begin()
6     if err != nil {
7         return err
8     }
9
10    _, err = tx.Exec(query, hashedPassword, id)
11    if err != nil {
12        tx.Rollback()
13        return err
14    }
15
16    err = tx.Commit()
17    if err != nil {
18        tx.Rollback()
19        return err
20    }
21
22    return err
23 }

```

Listing 5.13: MySQL Insert Example

```

1 // InsertUser to a course, gives a relationship
   between a user and a course in the database
2 func (repo *CourseRepository) InsertUser(userID int
   , courseID int) error {
3     // Query string
4     query := "INSERT INTO usercourse (userid,
   courseid) VALUES (?, ?)"
5     // Begin transaction
6     tx, err := repo.db.Begin()
7     if err != nil {
8         return err
9     }
10    // Execute query with parameters
11    _, err = tx.Exec(query, userID, courseID)
12    if err != nil {
13        tx.Rollback()
14        return err
15    }

```

```

16 // Commit transaction
17 err = tx.Commit()
18 if err != nil {
19     tx.Rollback()
20     return err
21 }
22 // Return no error
23 return nil
24 }

```

Listing 5.14: MySQL Fetch Example

```

1 // FetchHash hashed password for a user
2 func (repo *UserRepository) FetchHash(id int) (
3     string, error) {
4     var result string
5     query := "SELECT password FROM users WHERE id=?
6     "
7     rows, err := repo.db.Query(query, id)
8     if err != nil {
9         return result, err
10    }
11
12    defer rows.Close()
13
14    for rows.Next() {
15        err = rows.Scan(&result)
16        if err != nil {
17            return result, err
18        }
19    }
20
21    return result, err
22 }

```

5.3 Mail service

When planning to implement the mailservice to the system, it was estimated around one to two weeks to get it done. This estimation was not based on experience, or any estimation model, it was just estimated of how complicated we thought it would be. The estimate was not correct as the mailservice was implemented within a day.

This was because Go had an internal package that implemented the Simple Mail Transfer Protocol (RFC 5321) called "[net/smtp](#)". All you need to send an email with this package is the authentication which consists of identity, user name/email, password and host. Then the message in a byte array consisting of the addressing of emails (to, cc or bbc), the email addresses, subject and then the message. The final element is just to send the email with the server and port, the authentication, the email it's sent from, the emails it's sent to and the message. Altogether it's only needed two functions to send an email in Go.

The email and email password are stored as environment variables to strengthen the security

as the code is public on GitHub.

Listing 5.15: SendMail function in Go, [line 217](#)

```

1  package main
2
3  import (
4      "net/smtp"
5      "os"
6      "strings"
7  )
8
9  // sendMail sends the mail to recipient(s)
10 func sendMail(toType string, recipients []string,
11     subject string, message string) error {
12
13     users := strings.Join(recipients, ",")
14
15     // Get authentication
16     auth := smtp.PlainAuth("", "noreply@example.com",
17         "123abcPassword", "smtp.gmail.com")
18
19     // Write message
20     msg := []byte(toType + ":_" + users + "\nSubject:
21         " + subject + "\n" + message)
22
23     // Send mail and check for errors
24     err := smtp.SendMail("smtp.gmail.com:587", auth,
25         "noreply@example.com", recipients, msg)
26     if err != nil {
27         return err
28     }
29
30     return nil
31 }

```

Because it was so simple to implement the mailservice, it was decided to make the service as user-friendly as possible to use. The way the service was implemented was to have an GET handler to give feedback that it doesn't support GET requests. The POST handlers checked if the payload isn't empty and if the mail authentication code is correct. The mail authentication is there so only the system, that has the code, can email others. If everything checks out, the email get sent. The figure below, listing 5.16, shows how simple there is to send an email from the webservice.

Listing 5.16: Send mail from webservice, [line 199](#)

```

1  package controller
2
3  import (
4      "github.com/JohanAanesen/CSAMS/webservice/shared/
5          mail"
6      "net/http"
7  )
8
9  //RegisterPOST validates register requests from
10 users

```

```
9 func RegisterPOST(w http.ResponseWriter, r *http.  
    Request) {  
10  
11     // Get mailservice  
12     mailService := mail.Mail{}  
13  
14     // Send email  
15     err = mailService.SendSingleRecipient("  
        noreply@example.com", "Example_subject", "  
        Example_message")  
16     if err != nil {  
17         ErrorHandler(w, r, http.  
            StatusInternalServerError)  
18         log.Println("mail.MailForgottenPassword, ",  
            err.Error())  
19         return  
20     }  
21 }
```

In the aftermath it would be better to actually use an estimation model to estimate how long it would take to implement. It would also be better to research a little more before starting to get a better grip of how the implementation would be.

5.4 User Authentication

User Authentication was handled through the user of Go's *bcrypt*⁶ package together with Gorilla Sessions⁷. The *bcrypt* package has the functionality to hash the user passwords, which we again store in the database. It also has the functionality to compare a string to the hashed string stored, and return whether it is correct. Gorilla Sessions was used to store session data about the user inside the server, while setting a cookie on the client side with a key relating to this data.

In Figure 11 you can see a high level approach to how the authentication process is completed, where the user gives email and password as input, the server then fetches the hashed password from the database and compares it through the *bcrypt* package. If the password matches then a session is created and the user is logged in.

⁶<https://godoc.org/golang.org/x/crypto/bcrypt>

⁷<http://www.gorillatoolkit.org/pkg/sessions>

Authentication

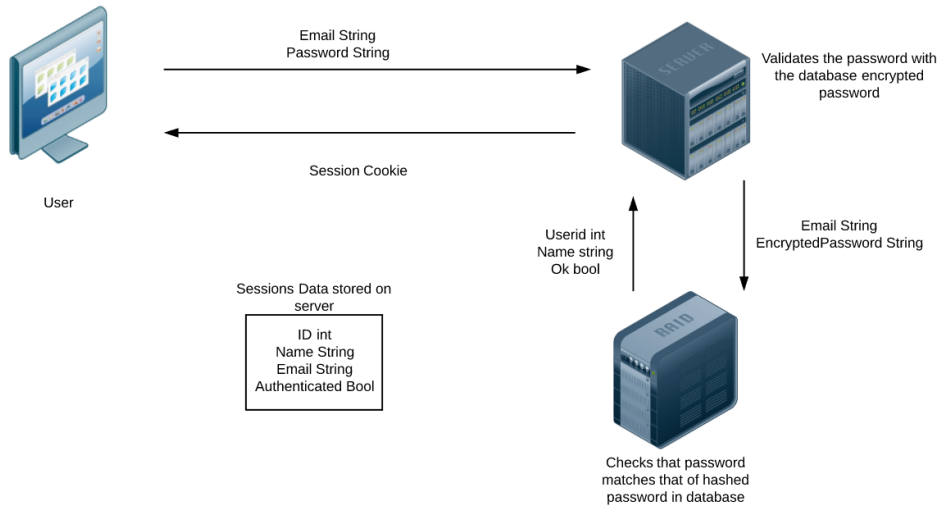


Figure 11: Authentication Model

5.5 Login Sequence

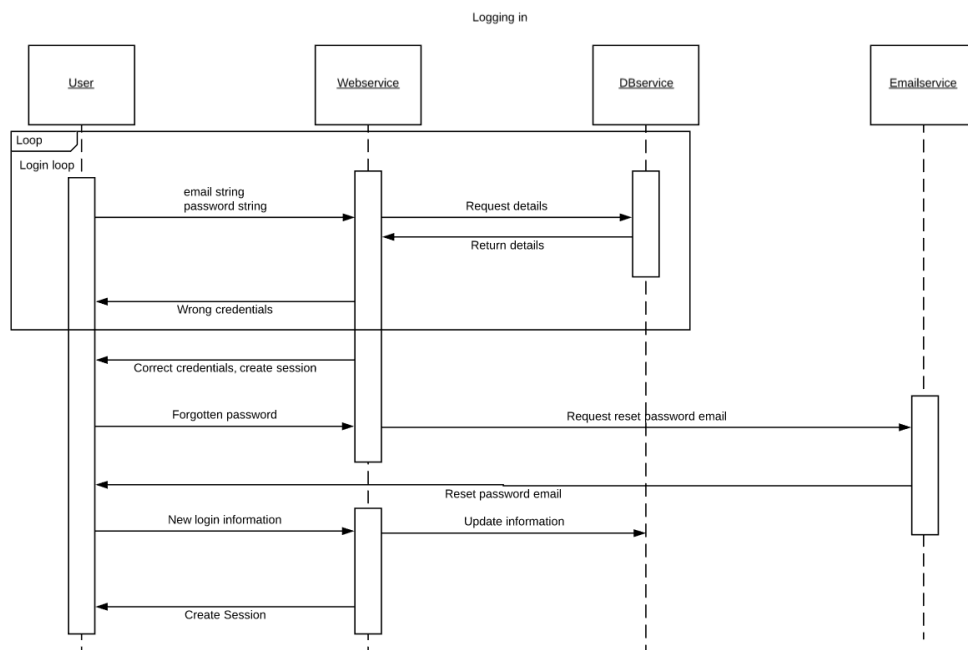


Figure 12: Sequence Diagram for logging in

5.6 Register Sequence

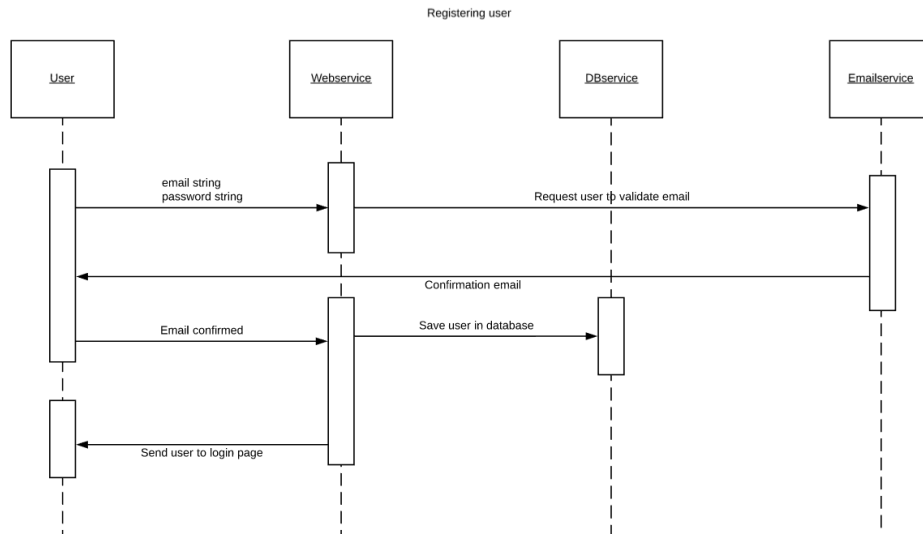


Figure 13: Sequence Diagram for registering an user

5.7 Codebase

In Table 5.7 you can see the distribution of languages used in the system.

Language	Percent	Rounded Lines
Go	58.6%	12 000
HTML	22.6%	4 600
Markdown	9.1%	1 800
JavaScript	5.7%	1 200
SQL	2.3%	500
Other	1.0%	160
CSS	0.7%	150

Table 27: Languages used in the project

6 Deployment

6.1 OpenStack

OpenStack was used as [Infrastructure as a Service \(IaaS\)](#) for this system, which meant we didn't have to think about any hardware for developing the system. The school provided us with the resources we needed for creating instances to run the production and development environments on.

OpenStack allowed us to select image type, flavor type and set security groups which makes it highly customizable, and this is also very similar to other big cloud service providers, as [AWS](#)¹, [Microsoft Azure](#)² and [Google Cloud](#)³.

As OpenStack is hosted by the NTNU i Gjøvik, and they have blocked outside access, we had to be on the Eduroam network, or use a [VPN](#) to use the system, or connect to the instance running it.

Flavor

Under a discussion early on in the project with the product owner, we talked about what kind of power an instance would need to maintain the system. We assumed we would need a powerful instance at the peak user-usage, so we went with a flavor with 2 VCPUs, 8GB of RAM and 40GB of persistent storage, called *m1.medium*.

Instance

Ubuntu⁴ was the choice of operating system, as it has good support for most software, and are one of the largest Linux distributions. The version selected was number 18.04, as it latest long term supported version for Ubuntu servers.

Shared access

The instances are paired up with only one public key, which only made it possible for one client to connect to the instance. At first the group members had one instance each for developing, but when we had an instance in production, only one had access to it.

To solve this problem, a SSH-tunnel was created. One group member setup a Raspberry PI⁵, and gave remote access to the rest of the group. And used the Raspberry PI as the client for the key pair for the production instance. This way all group members could access the production instance, in case something happened to it, and the other group members were unavailable.

¹<https://aws.amazon.com/>

²<https://azure.microsoft.com/>

³<https://cloud.google.com/>

⁴<https://www.ubuntu.com/>

⁵<https://www.raspberrypi.org/>

Security Groups

We only chose to open a few ports for the system, port 80 and 443 for HTTP/HTTPS requests, and port 8080 for HTTP request to Adminer⁶.

The amount of ports open were reduces to only a few, and only TCP for making the attack surface for the system as small as possible.

6.2 Docker

Docker was installed on the virtual machine using Snap⁷ to ensure we got the latest version. Further we created Dockerfiles written in yaml, for all the services we developed. The Dockerfile lets us create a set of instructions Docker will use when building the images.

The services we coded in Go were set up in a special way through multi-stage building, allowing us to minimize the sizes of the containers, so they do not include the whole Go library [10]. This made the containers lightweight at 20MB, while a normal build of the container would render it at around 800MB.

6.3 Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications⁸. It let us insert different environmental variables into the services, as well as connecting them through an internal network created by Docker.

The *docker-compose.yml* file was set up to start the different services in the system with each corresponding Dockerfiles. Except for the Adminer service which we just pulled the image from Docker Hub⁹. Docker Compose allowed us to say which services were dependent on each other, so it would build the most important service first. Docker Compose also handled what ports were open from the local network, to the external network and which containers should be able to talk to each other.

6.3.1 Web Service

The Web Service was built through the multi-stage building process as mentioned earlier, by copying the project files into the Go image, building it, and copying out the built certificates and built files. The Dockerfile also exposes the port the service will use, in this case port 8088. Further the Docker Compose connects the service to the database by dependent tags and environmental variables used for connecting to it.

6.3.2 Mail Service

The Mail Service was built and deployed the same way as the Web Service, although on port 8085.

6.3.3 Database

The Database is the simplest build of them all, where the Dockerfile uses the MySQL 5.7 image from Docker Hub and copies in the "database.sql" and "database_insert.sql" files into

⁶<https://www.adminer.org/>

⁷<https://snapcraft.io/>

⁸<https://docs.docker.com/compose/>

⁹https://hub.docker.com/_/adminer

the image. This makes the service setup the tables and most basic data so the database is ready to operate for the system. The Docker Compose would then set all the credentials for the service.

To insert previous data from an earlier database, we would connect to the database service through the Adminer service, and execute the previously exported insert data as a query.

For dependency reasons, we had the data folder from the database service mounted locally on the virtual machine. This made the data redundant even if the service were to go down, simply restarting the service would automatically load all previously stored data from this volume, and the service would also write all data to this mounted volume.

6.4 Makefile

We created a Makefile to automate the deployment of the system on the server, it includes a few couples of commands that let us build, run, clean or stop the service.

The build command would pull the latest code from GitHub, and then run Docker Compose with the "-build" command to ensure that the new code is built into a new container, instead of running a previously created container.

The run command would simply run the Docker Compose with the containers available.

The clean command deletes the mounted data folder from the db service, fetches new code from GitHub and builds the containers. This results in a fresh edition of the system and requires the administrator to import previous data into the database.

The stop command would just stop the running containers, meaning taking the system of-line.

7 Testing and User Feedback

7.1 Testing

The group talked about test driven development in the planning phase. Test driven development means to first create the test and then create the code that validates the test [11]. The method was rejected as the actual development method because it was more time-consuming than expected.

Creating tests for functionality we did not really know how to create at the time, resulted in a lack of progress and overall delay of functionality for milestones. Most of the unit testing was thus discarded because it was unsuitable for the functionality we developed, as well as we wanted to focus on progressing the application. However we had extensive user testing on the system, following three iterations of improvements and feedback. The development team themselves also tested the system thoroughly between every pull request.

The system was mostly tested by actual users and test protocols managed by a QA tester.

7.1.1 Unit Testing

The fundamentals of unit testing means to check a function with valid data, while assuring that the correct data is returned [12]. This makes it suitable for normal functions that do calculations or formats data in any way, most of our unit tests are inside the *util* package because it has these types of functions.

Most of the functionality on the system lies within the controller functionality, these controller functions are triggered through a series of requests to the server and are highly unsuitable for unit testing. This is one of the reasons why our project does not have that many unit tests in particular.

7.1.2 User Testing

The group had an early milestone to deploy a working prototype for the course [Mobile/Wearable Programming](#). The students used this prototype to deliver their assignments and check for vulnerabilities. The project went through three layers of user testing: alpha testing done by the group before deploying to project owner, black box testing done by the project owner and functional testing done by the students on the final system.

The main source for finding vulnerability, bugs and design flaws was the project owner. The admins on the system has the most functionality to test and the project owner is also going to use the system after this thesis is done, so he wanted it to be robust and bug free. There were also some students that went the extra mile and helped to look for flaws while using the system. Because of this, both the teacher and student user-type was represented.

7.2 Quality Assurance

A script was created in PowerShell to automate the quality check on the Go code in the project. The script ran [mod verify](#), [go vet](#), [go fmt](#), [golint](#), [gocyclo](#) and [go test](#). This made it much faster to check if something was wrong and gave a good overview of what needed

to be changed. The script also removed the mistake of forgetting one of the commands. An identical script written in bash was later added.

The tests for the project wasn't any good for finding bugs as mentioned in section 7.1. A [test protocol](#) was created to check if there were any bugs in the project before each deployment. The group also had an own unwritten protocol to thoroughly check each new functionality before making a pull request.

7.3 User Feedback

The extensive testing done by users resulted in a lot of feedback, have a look at section 3.2.1 for the implemented feedback and section 9.3.6 for the unimplemented feedback.

8 Discussion

8.1 Results

The final project do not have the auto-validation feature which was one of the main milestones, but have succeeded in getting all the other basic and more high level functionalities that the project owner wanted. The project did not have all the requirements before starting because the project used an agile development method and got all the requirements throughout the development timeline. The final project works as it was intended and got a little more improved by each iteration to the course that was testing the system.

8.2 Choice of technical solutions

The system was deployed on OpenStack¹ because that's what the original requirement stated.

8.3 Peer Review

The Peer Review was one of the largest initial functionality requirements because the Product Owner wanted the students to be able to review each other submissions. This functionality ended up having a major refactor after the first edition was released, because the automated aspect of the first version made it necessary to create complex algorithms for every small special case. This discussion has been further elaborated in the course Professional Programming, written by the project leader Johan, and is available under Appendix F.

8.4 Logging

The original way to log something to the database in the project was to use only one function "LogToDB(log model.Log)" with a log struct parameter. That function again checked the containing enumerated type in a switch and passed the data to another function that inserted the log to the database.

This solution had one huge problem, the user had to first create a variable of the log struct and fill in the information manually, but the user also had to know which exact variables should be filled in the struct. This wasn't a problem at the start of the project because it was under 10 logging events, but at the end of the project there was slightly more than 40 different logging events. The solution to this problem was to create one function for each logging event where the parameters was more clear. The function still worked as the old one, where it pointed to a switch which then again pointed to the function inserting the data to the database. This resulted in a lot of code that could be replaced by only one function, but this way it was really clear which function to use when logging different events.

8.5 Repository And Service

We decided to create more layers between the controller and the database for this system, and this was done adding two layers, repository and a service layer. The repository layer would

¹<https://www.openstack.org/>

be the one closest to the database, executing all queries and handle transactions between the back-end application and the database.

At the start of the project we let the repository layer be used by controllers, but after a while this ended up creating a code base with near to duplication, as the database calls may have small differences that did different things. So this is where the service layer was added, to separate the controller and the repository layer. This made it possible to reuse the repository functions for different functionalities as we could handle the data in the service layer with Go, and not do more work on the database.

We decided to gather all services into one single service *struct* for reducing lines of code in the controllers, as well as making readability of the code better. As anyone who reads the code can see that the data fetched for a functionality is used by services, and then specified what kind of service that was used. This also made expanding the code easier, as you would only have to follow the pattern used, and adapt it to the business model needed.

8.6 Form Builder

Since the product owner wanted to create submission forms for assignments, and review forms for the peer-review, we decided to create a form builder from scratch, based on the functionalities from Google Forms. The choice of making it in native JavaScript was made because it gives a better backwards compatibility's on older browsers, and possibilities for expanding it. The form builder is dependent on using Bootstrap as the style sheet, and another JavaScript library called Sortable², used for a more interactive user interface. Using Sortable was a choice made, as creating a sorting library our self would have taken up too much time, and the integration of Sortable was easily done, as the API for it was well documented.

The result of the form builder was far better than expected, as how configurable it got, both for the end-user and for developing it. So the hours used for planning how the code should be structured was worth the time.

²<https://github.com/SortableJS/Sortable>

9 Conclusion

9.1 Evaluation of the Group's Work

The group has throughout the bachelor worked together as a team with great communication, work dedication and interest in what we have developed. Through teamwork, we have managed to create a system more complex than we imagined on the start of the bachelor, and we have learned a lot from it. The group has mostly been working separately from each other home, but everyone has always been available at the set times and even when they were not working.

Through the great communication with the other group members and Product Owner, we have managed to rectify issues in short periods of time, and always improve the overall code of the system. As a team we have been able to solve technical problems by discussing and improving it, and the close dialog with the Product Owner has helped us extensively when designing and implementing the different functionality. The project leader has been effective when it comes to communicating with both the supervisor, Product Owner and group, and has been able to share work and responsibility of different functionality between us. Every group member has always had something to work on, and because of the large amount of feedback and functionality that could be implemented, every group member could work on something they felt dedicated about and interested in.

Because we are a separated group in terms of BIDAT and BPROG, we could share expertise between us that further expanded our ability to think, learn and develop. The group has been able to think about ideas together, spin them around and develop the best solution to each problem that we could think of. Through extensive code reviewing during pull request we have also been improving overall code quality, functional dependency and learning outcome from the functionalities we have developed.

The final system is the result of intensive work, collaboration and communication, and we think the work we have done reflects upon this.

9.2 Final Words

Through over 1300 hours of work, we have had the possibility to learn how to use different technologies, develop a system from the ground up and work cooperative. We have managed to develop a system that has already been used by three different courses to deliver their assignments, while working closely with the Product Owner. The group has been able to decide, in a large degree, the methods and technologies used to create the system. There have been a few setbacks, but they have always resulted in creating an overall better system. Through 52 pull requests, 1400+ commits, nearly 20.000 lines of code, weekly meetings and excellent teamwork, we have gotten a lot of experience that we will take with us from working on this bachelor.

We have met the goal of the Bachelor by creating a functional system to manage students assignments, proven by the students who have used the system throughout this semester. The finished system is something we as a group, as well as the Product Owner is proud of, and it will be used in future courses as well.

9.3 Future Work

In this section we have compiled a list of subjects and functionality that can be further developed on the system. It consists mostly of ideas received from feedback, or functionality and refactoring the group think will improve the system.

9.3.1 Auto-Validation Functionality

Auto-Validation was one of the large functionalities we planned ahead of the development start, but was later discarded because of time restraints and the Product Owner wanted the peer reviewing process to be refined and smoother.

The Auto-Validation functionality should enable the students to get feedback on their submitted code by having the server run different kinds of linting and code quality checks much like SonarQube¹. Another future would be to have the users deploy their web applications on a cloud, and have the server check all the specified API endpoints.

9.3.2 API Based Micro Service Architecture

Something we realized towards the end of the development cycle was that the Model layer of the webservice got really big as described in Chapter 4. For future work and easier scalability and modularity, it would be effective to change the service and repository layer inside the model layer, to stand alone services that are available through Rest API [13]. Making the different services standalone simplifies unit testing for each service, because the necessary functions are contained within and are not dependent on any other service.

The Micro Service architecture would also allow the system to use technologies like [Kubernetes](#) to enable resource monitoring and scaling throughout the system.

Through the course of this bachelor all the group members have gone through job interviews and application processes for real software companies, and through our experience it seems like this is the industry standard when it comes to web applications.

9.3.3 Front-end

Changing the front end from the Go Templating solution that we are currently using, to a front-end framework like [Angular](#) or [React](#). It is important to mention that this would require the Micro Service API structure of the back end as mentioned above.

The most extensive front-end frameworks enables the developer to move the controller and view layer into it, while using API endpoints for the model layer. This can greatly improve code quality across the system because it requires the developer to separate the code.

We thought about this approach before starting development of the system, but because the Product Owner wanted the system to be primarily programmed in Go, we decided that it would go against the requirement.

9.3.4 Notification and Messaging Service

A notification service or notification functionality paired with messaging capabilities on submitted assignments and in the review process is something that has been mentioned by both the students and the Product Owner, in terms of functionality they would want implemented. Unfortunately time constraints kept us from developing this at the end, but it is planned to

¹<https://www.sonarqube.org/>

be implemented after the bachelor is delivered, as part of the course Rapid Prototyping.

The functionality is in short terms, the ability to write comments on submissions while reviewing them, as well as receiving these comments on your own submissions. To complement the commenting, it would require a notification system to let the users know they have received comments and direct the users to the location of interest. The notification system can be further developed to notify users when the deadline is closing in, or the teacher has opened up for submissions on an assignment.

9.3.5 Time-zone Error

One of the system requirements was to have the possibility to quickly change the timezone. The solution to this goal was to use environment variables. This way, the person deploying the system could just change a variable one place from ex. "Europe/Oslo" to "America/Chicago" and the system time would be changed all over. The variable was originally used in Go and JavaScript, but had to be changed to only be used in the Go files. The environment variable and JavaScript's way of handling time-zones wasn't good enough, so the timezone was later just hard-coded for JavaScript.

If someone wants to change the time-zone, they have to look for the "TODO : time" in the project and change to the preferred time-zone.

9.3.6 Feedback From Project-Owner And Students

Show Logs For Scheduler Expose log information in the UI, so the admin sees what has been triggered and when. (Scheduler was later removed)

Simulate Student Perspective The teacher should be able to simulate the student's perspective to preview what students see.

Deletable Course, Assignment And Forms Allow the user to delete course, assignment and forms, but ask for confirmation and log the activity.

Copy Forms Allow copying of existing forms and adaptation.

Separate Display With Actual Deadline For any deadline, provide an actual deadline and a displayed deadline. Sometimes the actual deadline could be set slightly later to deal with situations of heavy load.

Allow Late Submissions And Include In Review Pool Allow selective late submission and automatically include in review cycles. Late submissions can happen in the case of system glitches as well as unforeseen medical cases.

Override Final Mark Allow the admin to override the final (calculated) review points.

Reward Reviewers The reviewer should be rewarded for doing the review.

Non Edit Admin Have a non-edit admin mode for TA's who should be able to see the progress, but not be able to modify anything.

Remove Unsigned Students Add ability to list and remove students that are not signed up to any course. Implement an all students, and not-assigned students selection in the student management sections.

Let User Leave Course Allow students to withdraw themselves from a course, but retain data in case s/he rejoins.

Group Based Assignments Add a different kind of assignment type: a group assignment.

Students can sign up to a group (or create one) and submit together. The group may then be involved in reviewing other groups' assignments. All members of the group share the resulting marks.

Review Non-reviewed Submission If a student didn't get any reviews on his/hers assignment submission, let the teacher add a review manually.

Layout Radio Buttons Vertically Lay out radio buttons vertically if the description is too long to maintain horizontal layout.

See Performed Reviews If the deadline for reviewing has expired the user should still be able to see the review they created.

Review Status Admin should see on dashboard overview whether reviewing is active or not.

Ask Before Deleting Assignment Submission When a user requests a withdrawal of the assignment submission, present a dialog to confirm the withdrawal.

Disable A Field In Review And Submission Forms Allow admin to remove a field from view (not delete) in review and submission view, even after the submission/review has started.

Respond To Feedback Implement a way for the students to respond to received review.

Flag A Submission When a student is reviewing a submission, let the user flag a question if it's something unclear or hard to evaluate.

Private Repository Warning Let peer reviewers notify the relevant student if the repository is private i.e. not accessible.

Dark Theme Add dark theme to avoid making the users' eyes tired.

Ability To Add Screenshot Add the ability to add screenshot to review and submissions.

Ability To Chat When reviewing another student, the reviewer should be able to chat with said student.

Email User When Profile Is Edited The system should email the user if their profile has been edited in any way.

Give Deadline Warning The system should send an email if the user hasn't delivered the assignment and it's only X time left until the deadline is due.

Bibliography

- [1] Jon Terry. What is kanban? <https://www.planview.com/resources/articles/what-is-kanban/>, 2019.
- [2] Scrum.org. What is scrum? <https://www.scrum.org/resources/what-is-scrum>, 2019.
- [3] Don Wells. Extreme programming: A gentle introduction. <http://www.extremeprogramming.org/>, 2013.
- [4] Atlassian. Git Feature Branch Workflow. <https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>, 2019.
- [5] Service-oriented architecture (SOA). https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.5.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html, 2019.
- [6] Design Patterns MVC Pattern. https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm, 2019.
- [7] Go Web Examples: HTTP Server. <https://gowebexamples.com/http-server/>, 2019.
- [8] Joseph Spurrier. Go web app example - views, request workflow, and view plugins. <http://www.josephspurrier.com/go-web-app-example-views/>, 2016.
- [9] HTML Input Types. https://www.w3schools.com/html/html_form_input_types.asp, 2019.
- [10] Building Docker Containers for Go Applications | CalliCoder. <https://www.callicoder.com/docker-golang-image-container-example/>, 2018.
- [11] Test-Driven Development (TDD). <https://technologyconversations.com/2014/09/30/test-driven-development-tdd/>, 2014.
- [12] Unit Testing - Software Testing Fundamentals. <http://softwaretestingfundamentals.com/unit-testing/>, 2019.
- [13] Roy T Fielding and Richard N Taylor. Architectural styles and the design of network-based software architectures, volume 7. University of California, Irvine Doctoral dissertation, 2000.

A Project Repository

<https://github.com/JohanAanesen/CSAMS>

B Project Agreement

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

Christopher Frantz (NTNU)

(oppdragsgiver), og

Svein Arz Danielsen, Brede Fritjof Klausen
& Johan Aanesen

(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 01/02/19 til 16/05/19.

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik 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 NTNU 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 NTNU på Gjøvik. 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. NTNU IE på Gjøvik 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 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 åpne 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 instituttleder/fagenhetsleder 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 NTNU 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 og/eller studenter har interesser.

5. 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.
6. 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.
7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.
8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.
9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU 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 NTNU som partner.
10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.
11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. 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.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Ivar Farup

Oppdragsgivers kontaktperson (navn): Christopher Frantz

Student(er) (signatur): Svenn Arne P. dato 11/1/19

Breda F. Kløkken dato 12/1/19

[Signature] dato 11/1-19

_____ dato _____

Oppdragsgiver (signatur): Christopher Frantz dato 11/01/2019

*Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggrupeleder.*

*Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:*

Instituttleder/faggrupeleder (signatur): _____ dato _____

C Project plan

Management system for CS assignments

Svein Are Danielsen
sveiad@stud.ntnu.no

Johan Aanesen
johanaan@stud.ntnu.no

Brede Fritjof Klausen
bredefk@stud.ntnu.no

May 16th 2019

1 Goals and limits

1.1 Background

Managing over a hundreds of assignments several times in a semester, in a single course can be challenging. With help from the students with peer review assessments, and for the students to get a instant respond on their codes, based on the standards and requirements set by the lecturer. Time will saved both for the lecturer, and for the students looking for feedback.

1.2 Project goals

1.2.1 Effect goal

Computer Science lecturers get a tool that efficiently and seamlessly can automate assignment validation, and can automatically assign peer review for students. The system should make it more clear what is expected from the students, and giving them extra learning in doing peer review anonymously on each others code. This will remove miss understandings that exists in the current peer review, and assignment delivery systems.

1.2.2 Result goal

Replace existing systems and deliver a working solution for automating computer science assignment validation and assigning students to peer review assignments.

1.2.3 Learning goal

- Learn to work with a project from the planning phase, all the way til the finished product.
- Learn to work with the product owner on-site in a agile method.
- Become better at Go, and full-stack programming.
- Become better at estimating programming tasks, and project-features for real life application.

1.3 Limits

The project is time-limited until May 16, when the application has it's due date for the product owner.

The thesis is time-limited until May 20, set by the school. Internally in the project group, the due date for the thesis is set to May 16.

There are no software limits for this project. All languages, frameworks and code is publicly available, or the product owner has the licences.

2 Scope

2.1 Subject area

Main scope for this project is programming.

Cloud programming will be the main subject area within this project, as the project requires the application to be cloud-based.

Web development will be the secondary subject area, where the front-end of the application will be a website, for all users. And information will be stored in a relational-database.

Container technology will be used for making the software non-dependant of any operation system.

2.2 Delimitation

Online system, written in Go on the back-end. Needs to be deployed on OpenStack with underlying container technology, running with a local database. The system should work with Go-programming assignments, and be extendable to other programming languages.

2.3 Task description

The group will develop a system that a lecturer easily can deploy on the schools infrastructure, OpenStack, and configure for their specific course. The system will be needs to be able to create custom submission forms for different course assignments. The system needs to be using the application programming interfaces from several websites using Git-repository-technology, mainly GitHub and Bitbucket. After a assignment is done, the system need to distribute review forms to the students, making them peer review themselves.

The system also must do a queued responds for the students, with a configurable validation of the submitted assignment, that will within short time give a feedback to the student.

3 Project organization

3.1 Responsibilities and roles

Here is the list of the group members name and roles.

Name	Role
Johan Aanesen	Project leader
Brede Fritjof Klausen	Member
Svein Are Danielsen	Member

The project leader will be the one responsible for contacting the supervisor and the product owner.

3.2 Routines and rules

The group members are committed to work on the project between 09:00 and 16:00 every day of the week except for compulsory lectures and meetings. All members should be available either in person or online during this time.

The group has weekly meetings with the supervisor, Ivar Farup, Mondays at 13:15, in room A218.

If the group cannot meet, it must be reported to the supervisor no later than the first Friday before the meeting.

The group has meetings with the product owner, Christopher Frantz, every Monday at 12:00, in the Ametyst-building.

If the group cannot meet with the product owner, it must report to them no later than the first Friday before the meeting.

Ordinary working hours for joint work, are from 09:00 to 16:00, unless otherwise agreed. Each member must work weekly for at least 30 hours, and is responsible for their own logging.

Lost working hours can be retaken by working in the weekends, or longer working days.

For exceptions and special cases where the group is not assembled, each member will be responsible for performing the weekly workforce and how it is distributed.

Tasks will be discussed and distributed in plenary session - each group member will be responsible for performing these at the agreed time.

Should insecurity arise during work tasks or other problems during this period, the relevant members are obliged to take this up with the rest of the group.

Group members are obliged to ask the group for help when they are stuck or are consuming too much time on a given task. In worst cases Project Leader can assign the task to another group member in order to keep the project moving forward.

3.2.1 Rules

- If a member does not have the opportunity to meet, this should be reported preferably one day in advance. Valid justification for any certificate is required.
- If a group member feels that another group member is not contributing enough, this must be addressed with the given member. If he or she continues to do the same, the group will take the problem to the supervisor for counseling. Continuous lacking of contribution will in the worst case lead to exclusion from the group.

- Disagreements on major decisions where the group does not arrive at an agreement, then the issue should be addressed with the supervisor before a possible decision is made.
- At each meeting, each group member must, if possible, submit a progress plan/status report.
- At each meeting, a meeting record shall be written, which shall be available to the entire group.

4 Planning, monitoring and reporting

4.1 Main division of the project

The group will be using Kanban [1] with some elements from Scrum [2] as the development method. The reason for this choice, is that Kanban gives the group the ability for more flexible planning, clearer focus throughout the development cycle. And will use weekly cycles that can resemble Scrum-sprints, but will be used for focusing on specific back-stories. Mondays will be used for meet-ups and planning for the rest of the week, and Fridays will be used for debugging. The rest of the week will be used for development, and writing the thesis.

The group will be present on-site, and have access to the product owner daily, that will be present on Mondays for creating stories for the back-log.

Trello [3] will be used as the Kanban-board, [4] which is a board-based website, for personal use, or team-based work.

4.2 Plan for status meetings and decision points during the period

For the project the group will have continuous contact with the product owner on a chatting application called Discord. [5] There will also be meetings every Monday where the group can ask for direction, or input on the development. The group has set of four weeks for the main features in the application, in three phases.

5 Organization of quality assurance

5.1 Documentation, standard use and source code

- Every member has to write documentation while writing the code.
- Reasons why the member chose or changed the solution, it should be written in the report as it's happening. (i.e don't start on documentation when you're done coding, but meanwhile.)

- If the code comes from another source than yourself, the link or name of person should be included in the documentation for the relevant function.
- Use tabs, not spaces!
- Thoroughly test code before issuing a pull-request to the master branch.
- Pull-request has to be reviewed and approved by the rest of the team before merging.
- Follow the key points of test driven development.
- Lint and cycle code before issuing pull requests, with *golint*, *go vet*, *go fmt*.
- Update Trello [3] as you go and link boards to commits/pull-requests.
- Follow "Internal Peer Review Checklist" when doing peer review.
- Every friday, write a short recap what you have done, what went smooth and why, what took more time than expected and why, and what could be done better and why
- Use Effective Go [6] as code standard

5.2 Configuration management

Managed through existing solutions in git like commits and pull-requests.

5.3 Risk Analysis

What	Impact	Probability	Rating	Mitigation
Data loss	High	Low	Medium	Cloud-based storage and backup
Lack of contribution	High	Medium	Medium	Weekly meetings, talking with other group members about tasks.
Disease	Medium	Medium	Medium	Eat and drink healthy, follow national health guidelines
Hardware failure	Medium	Low	Low	Treat hardware with care, and follow the manual of the product
Loss of motivation	Low	Medium	Low	Work on different tasks, be curious. Don't golden hammer

5.4 Technology, Business, Project Group

The programming language Go will be primary used for this project, as the back-end of the application. Front-end will consist of Bootstrap, which is a toolkit for development with HTML, CSS and Javascript, which is under MIT-licence. [7]

The application will be deployed on OpenStack [8] where each group-member has a licence through NTNU. And the application will be containerized with Docker (licences Apache 2.0). [9]

The application for this project will be under "GNU General Public License v3.0" licence. [10]

6 Plan for implementation

6.1 Milestones

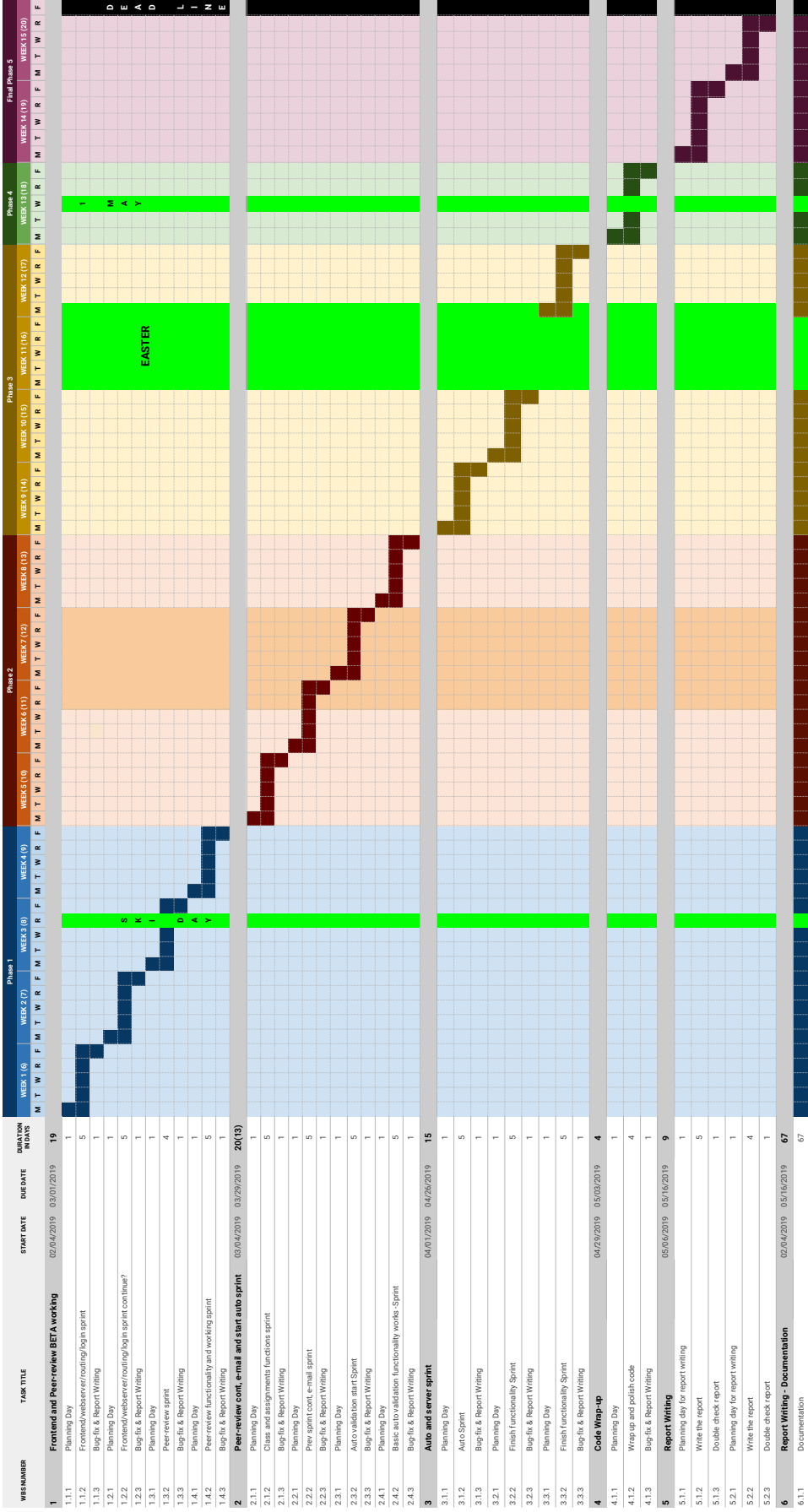
As seen in the Gantt-schema the first milestone are after the first 4 week period, where the first feature of the system will be done; a working beta of the front-end and peer-reviews.

The next milestones will be decided by the product owner as the first 4 week period is over.

The schema for the groups future plan can be found on the next page.

GANTT CHART - BACHELOR 53

PROJECT TITLE Bachelor 53
PROJECT LEADER Johan Aarnesen
PRODUCT OWNER Christopher Franz-HITM, Cgnik
START DATE 27/7/19
END DATE 5/16/19
Holiday Parashute Jumping for Johan
Deadline



References

- [1] Atlassian - What is kanban?
<https://www.atlassian.com/agile/kanban/boards>
- [2] Atlassian - What is Scrum?
<https://www.atlassian.com/agile/scrum>
- [3] Trello - Tour
<https://trello.com/tour>
- [4] Atlassian - What is a Kanban Board?
<https://www.atlassian.com/agile/kanban/boards>
- [5] Discord - About
<https://discordapp.com/company>
- [6] Effective Go
https://golang.org/doc/effective_go.html
- [7] Bootstrap - Licence
<https://getbootstrap.com/docs/4.1/about/license/>
- [8] OpenStack Documentation
<https://www.ntnu.no/wiki/display/skyhigh/Openstack+documentation>
- [9] Docker License
<https://github.com/moby/moby/blob/master/LICENSE>
- [10] GNU General Public License v3.0
<https://www.gnu.org/licenses/gpl-3.0.en.html>

D Meeting Logs

The group had some technical difficulties that resulted in loss of data, this is why some logs are missing.

D.1 Record of meetings

D.1.1 17.01.19 - Thursday

- We have the possibility to switch the report language to Norwegian anytime (but we won't)
- Make a good plan, so it's easier to know the progression!
- Weekly meetings with [Project Owner](#) at 12:15 and [Supervisor](#) 13:15 on Mondays.
- Find model (we chose [Kanban](#)), and assign roles.
- Make Gantt scheme.
- Report and comment while working on project.
 - How/why did you solve the task like that?
- First week in June is presentation time!

D.1.2 28.01.19 - Monday

- We presented the Gantt scheme to [Supervisor](#), we needed to make it better.
- **Very important:** Comment and write log while programming.
- Make a prototype for peer review and get good feedback, preferably have it done before march.
- Milestones: can be seen on the Gantt scheme, also talk to [Supervisor](#) after each milestone.
- Maybe make the project test driven?
- Maybe export Trello comments?
- Write why we chose Kanban in plan report.
- What license do we use? ([GNU General Public License v3.0](#))
- Add possibility to add a second email
- Add possibility to have stats (% of reviews done and grades) and make it possible to export (json?).
- Have a check-box for permission to share users email.
- Detailed view for courses
- Save version build - ex for mobile development
- Log for delivery
- Log everything
- User can request to be removed.

D.1.3 11.02.19 - Monday

- Make it possible to upload course information for later use, have it in front-end.
- Add weight to assignments, but not max at 100, just fix it when all the weight is written and then calculate percentage and show to admin.
 - Also add option to show the percentage

- Make project open for adding other tools than Go tools.
- Change 'Priority' to 'Order' in assignment creation
- Let users write an description explaining why he/she rated X.
 - also add comment section for each assignment for the students to ask question to admin or other students.
- Have an [FAQ](#) for admins
- Look up Go modules.
- We got good feedback that we had worked faster than expected.

D.1.4 18.02.19 - Monday

- Assignment review
 - Mandatory and voluntary questions
 - Talked about [#12 - Create Assignment form/page](#) for 25min
- Add deadline to assignments and gray out former assignments
- We informed [Project Owner](#) that it's skiday on thursday
- Document all smaller things that's vital for others later, ex. timezone in the sql file.

D.1.5 04.03.19 - Monday

Project Owner

- Check URL in back-end
- Sanitize input
- If a user has answered submission form, don't let admin change the form
- Tell why stuff be like it be to the end-user
- Get hash from course and display to admin in front-end
- Give admin login/register link for joining course
- Add convert to MD plugin
- Set time uploaded on submissions admin
- Get the total number of submissions admin
- Maybe track view for submissions admin, to see whats new
- Remove joined course bug
- Fix submit new course button
- Add order by course.semester back again
- Make real name uneditable, maybe add user name instead
- Forgot password functionality?

Supervisor

- Tabs on course should be more visible
- Maybe don't let admin submit assignment submission
- Add equation possibility in MD? ([mathjax](#))

D.1.6 11.03.19 - Monday

- Deploy with different settings
- Demo of project
- Fix timezone
- Clearly state the timezone on the system
- Submission forms

- Remove name from reviewing
- Convert weight to percent
- Change update button on assignment from 'update' to 're-upload'
- Change students password (admin feature)

D.1.7 21.03.19 - Thursday

- Test Protocols - [QA](#)
 - Use cases
- Fix JavaScript time bug
- Use Github's issue tracker as backlog
- Implement function to override scheduler, maybe delay time too
- Keep manage students site
 - Persistence design

D.1.8 01.04.19 - Monday

Project owner

- Add buttons on submissions table (admin)
- Edit review
- Show reviews done by user
- Don't implement auto-validation after all
- BUG: If user not in review table after deadline, it could go wrong on the assignment card.
- Confirm submission is delivered
- Only use one '!' after messages, instead of multiple
- Implement functionality to add screenshot in review/submission
- Let admin change users name (We didn't implement this)
- Separate first and last name (Not this either)
- Maybe implement student number (nah)
- Create a no reply email
- Admin view, but user can't to admin stuff. Ex for [TA](#)
- Separate deadline. Add around 5 min or something to deadline that is displayed
- Create new in assignment?
- Add tags for submission/review
- Marks for giving review and receive review

Supervisor

- Ask user before using data
 - [GDPR - General Data Protection Regulation](#)

D.1.9 04.04.19 - Thursday

- Form Builder: multi check box
- Reviews: able to edit until deadline
- Feedback: Next Tuesday
- Group Assignment: Deploy after Easter - Deadline, May 2nd
 - Add link to forms on «View Assignment»
 - Add review deadline to «View Assignment»

- Logs: Select logs by time - AFTER GIVEN TIME
- Remove course-list for students, without consent (Course-setting)

D.1.10 11.04.19 - Wednesday

Supervisor

Maybe look into:

- [ITICSE - Innovation and Technology in Computer Science Education](#)
- [SIGCSE - Special Interest Group on Computer Science Education](#)

Project owner

- Implement functionality to email all students in course
- Split first and last name
- BUG: if there is zero peer review on assignment cards
- Fix update form to none
- Log everything. Filter by system, course, all, admin.
- Add functionality to have group
 - Create
 - Equal rights
 - Teacher can kick users

D.1.11 29.04.19 - Monday

- Group: user_id for the whole group
- Document Adminer (import/export)
- Document how to install the system
- Fix configuration file. No public information (Mock-up .env file)
- «Download Statistics» button, generate .csv-file
- Ask [Supervisor](#) about how GitHub-Issues should be implemented in the thesis
- Development Process - Trello - Move to: Introduction - Work process

E Daily Logs

E.1 Johan Logs

Johan's logbook :)

Tuesday 29/1

- I created the database model through phpmyadmin, worked fine I guess although the final specs are not complete
- Distributed tasks for the first week or two on trello :)
- Sick af, no really

Wednesday 30/1

- I started early because I am a straight up boss, actually I caught a cold and I'm sick and got nothing better to do
- Started with card #3 Router
- Made all the basic endpoints and tests, ran into some issued where the test would run as if it was inside the /handlers folder, and thus the template couldn't find /web/test.html. Got fixed by a small init function which ships the test suite back.

```
func init() {
    if err := os.Chdir("../"); err != nil { //go out of /handlers folder
        panic(err)
    }
}
```

- Found out a designated error page would be nice so I added that to the UML and router too

Monday 4/2

- Started doing the card #14 database connection class
- Decided on a global approach to DB access, aka all the files that import the db package can access the global DB (dbcon*) variable :)
- <https://www.alexedwards.net/blog/organising-database-access>
- Svein reminded me with need some sort of session management, although GO doesn't include it's own session functionality
- Could implement the functionality myself by creating unique values on cookies, and storing that value in the database, although this would cost IO operations
- Decided on going with the Sessions package from Gorilla
- <https://www.gorillatoolkit.org/pkg/sessions>
- Together with the more powerful mux router package and secureCookies package, all part of the Gorilla Web Toolkit
- Seems like the go-to toolkit for most people on stack overflow
- Password encryption added with bcrypt <https://godoc.org/golang.org/x/crypto/bcrypt>

Tuesday 5/2

- Link to execute go commands on scripts: <https://github.com/gojp/goreportcard/blob/master/check/utils.go>
- Changed a bit on the login/register things, works nicely

Wednesday 6/2

- Started the day setting up the project on my laptop, ugh
- Changed from local imports to package imports
- Wrapped up the login/register functionality, works nicely afaik
- Started on #11 Create Course
- Svein already made most of the frontend so I dont have to, score

Thursday 7/2

- 11 Create Course cont.
- Added url link fields, working on DB query to 'save' the course permanently
- Implemented input fields for 3x url's, after talking with Svein we decided on implementing markdown for the description field, so the teacher can just add links there.
- <https://simplemde.com/>
- Christopher said it was 'sensible' :)
- Kinda back on tests, working on making some now

- Changed DB functions to handle User objects instead of 3-4 fucking variables
- Now checks if user is a teacher before serving /admin pages

Friday 8/2

- 11 Create Course cont/testing
- Finished up 11, found out that the tests that needed some kind of session could be hacked because i have the cookiestore/sessionstore stuff available in the test suite, sweet
- Helped Brede with some bugs on his 8-userpage branch
- didn't have too much time to work this day :(

Sunday 10/2

- Refactored 8
- Made more session function and moved them into the util package
- Moved DB functions out of db.go and into feks. coursedb.go and userdb.go
- Had to rewrite a lot of UserHandler code because it was messy and unreadable/inefficient

Monday 11/2

- Meeting day
- Found out it was time to do a major restructure of the code
- Restructure took the whole day :(

Tuesday 12/2

- Restructure cont.
- Svein knew what was supposed to be done with the restructure, I had to find out by trying
- This took longer than it should have, next time we should have a better plan for what goes where and how it should work
- BEFORE starting the restructuring.. ffs
- Lots of time went into getting the god damn tests to work again

Wednesday 13/2

- Cont. on #7
- Got the md -> html functionality working :)
- This gonna look nice
- Also some more db schtuf

Thursday 14/2

- Cont #7 jesus fuck will it ever end
- Made front end look nice :)
- Changed markdown processor so it would read 'github flavored markdown' better
- yeah!

Friday 15/2

- Wrapping up #7
- Added comment tab

Monday 18/2

- Wrapped up the coursepage
- Meetings n stuff <3
- Started on participant list

Tuesday 19/2

- Wrapped up the participant list thing, wasn't very hard to do

- Started doing research on docker containers and docker-compose kind of setups to "knit" together the different services we are creating

Wednesday 20/2

- Started writing the peer review service
- The peer review service is supposed to retrieve a request whenever, the request should include some sort of authentication, haven't decided what yet. And what submission is supposed to be reviewed as well as how many submissions every person is supposed to review.
- I distribute the who-reviews-what by fetching every submission and their user from database
- Randomly shuffle the array
- If everyone is reviewing 2 tasks, then every person get's the 2 next in the array.

Thursday 21/2

- skiday!

Friday 22/2

- Hungover day, no work was done except reviewing a pull request.

Saturday 23/2

- 28 cont.
- Trying to setup Docker on my pc, had to enable hypervisor in powershell to make it work :(

```
bcdedit /set hypervisorlaunchtype auto
```

- Sitting for 8 hours now trying to make the docker thingies work, but without luck. Will try again tomorrow.
- <https://www.melvinvivas.com/my-first-go-microservice/>
- Multi staged docker builds learned through: <https://levelup.gitconnected.com/multi-stage-docker-builds-with-go-modules-df23b7f91a67>
- Shows how to make small docker images for GO applications

Monday 25/2

- Dockerizing cont
- IT WORKS
- I can finally say I'm starting to understand Docker, and to this usage I mean no specific article helped out a lot
- PO wanted the database to be persistent, implemented this through a Dockerfile for the dbservice which will add the database.sql file to the initialization folder in the container, but now it will only generate the db if it doesn't exist. Then in the docker-compose file I set the persistent volume, works like a charm.

Tuesday 26/2

- Setup OpenStack
- Deployed application with docker, a lot easier than expected
- Added Auth, POST and more functionality to the peer service
- Need a Scheduler Service to run tasks at a given time

Wednesday 27/2

- Finished off the peer review service, although lacking of a duplicate request failure
- Helped Brede setup the project on both openstack and on his computer with docker installed <3
- There is a lot of duplicate files across the services, tried to move this out of the services and into a /internal folder
- No luck in making the services use those files tho
- Creating the SchedulerService
- Making it so it doesn't necessarily only take peer scheduling request
- First bit of the request has to be the same, but from there I can add different execution methods and functionality :)
- Using Goroutines to keep track of when to execute what services

- Switched to use the AfterFunc(duration, func) which triggers func after duration has passed.
- Perfect for this usecase

Thursday 28/2

- Scheduler service is now working wonderfully, both registering through a http post request and it actually triggering the peerservice correctly works.
- Still missing RUD out of CRUD, but that should be pretty easy to do tomorrow.
- Scheduler will not accept requests with the same submission id as a current timers submission id
- The peer service and Scheduler service both needs some polishing, but are functional for the alpha planned on monday

Friday 1/3

- Implemented GET, PUT, DELETE to schedulerservice

Saturday 2/3

- Added tests to schedulerservice
- Issues with computer, piece of crap
- Made edit course functionality
- Changed course model to courseRepository to access db functionality of model

Sunday 3/3

- Created 'see active assignments' functionality for frontpage
- Fixed course_code to show in the badges on assignments
- Fixed insert to use transaction in submissionRepo
- Had to redo shcedulerservice to also use assignmentID because apparently one submission might have several rows of input
- Made update and delete functions to schedule peer tasks in webservice
- Setting and updating reviewer count/nr functionality
- Bug fixes from hell

Monday 4/3

- Fixed course create button

Tuesday 5/3

- Added XSS Sanitization with the bluemonday package
 - Login
 - Register
 - Assignment submission page
- Made tabs more clearer and design neater on course page
- Added css library for Go syntax highlighting from markdown code tags

Wednesday 6/3

- Added time submitted and a count of nr of submissions to admin assignment submissions page
- Fixed width issues with course and assignment boxes
- Updated design on assignment page

Thursday 7/3

- Assignment page frontend finished :)
- Added 'edit assignment' button to assignment boxes in admin dashboard

Friday 8/3

- Added invalid login message
- Changed the message to be stored in session

- Added e-mail already in use message
- Various error messages added
- Added CourseID and AssignmentID to admin page overview boxes on frontend

Monday 11/3

- Changed to using showdown.js with highlight.js to convert markdown into html code and syntax highlighting for code snippets
- Removed set header to update flash message/session message
- Added AssignmentID to assignment details

Tuesday 12/3

- Added scheduler files to see scheduled tasks in admin view
- Refactored database.sql to not include everything
- Separated out the insert to it's own database_insert.sql file
- Updated insert file with dummy data for testing and debugging
- Added a frontend for seeing data from the scheduler, taken from the api provided by schedulerservice

Wednesday 13/3

- Linting errors fixed in session files
- Updated update function for assignments
- Updated scheduler to update reviewers count
- Imported sveins changes
- Updated the frontend panels for scheduler in admin view
- Added the functionality to delete scheduled tasks from the frontend

Thursday 14/3

- Fixed schedule delete form
- Fixed lint errors

VACATION until 25/3

Tuesday 26/3

- Fixed bug related to database not initiating timers in schedulerservice
- Removed submissionID from peerservice and schedulerservice as it was not needed
- Updated database table to exlude the submissionID for the same reason

Thursday 28/3

- Added reviews overview for assignments
- Refactored a large spelling mistake done to the repository folder, it was named repositroy, fixed it.
- fixed count issue for reviews done

Friday 29/3

- Moved back button to a better location in admin/assignment/review
- Added email service template for brede
- Fixed issue when seeing two reviews done on an assignment, only one of the reviews's radiobuttons would be selected.
- Solved by wrapping them in its own forms.

Tuesday 2/4

- Changed title on reviewers overview page
- Removed redundant div row tag

Thursday 4/4

- Merging done

Monday 8/4

- Mailservice merge and once again updating the repository folder type

Tuesday 9/4

- Added privacy policy
- Added it as a term of registering
- During the monday meeting we decided that the automated way of the reviewing system was far too complex to address minor issues like:
 - Removing an user from the review cycle
 - Adding a late submission to the review cycle
 - etc
- These scenarios would need it's own algorithms that I had been mocking up the past weeks, but it was a lot of work for code that would rarely be run
- Decided to use a review-pool instead, so by pushing a button you would receive a review, instead of automatically assigning all the reviews at the deadline
- Removed scheduler-service and peer-service
- Updated docker-compose
- Removed more traces of the services
- Removed scheduler overview from admin page
- Updated assignment setting with a 0 value for reviews
- Disable reviews button added to edit assignment
- Updated create assignment page with the same disable button

Wednesday 10/4

- Added review-pooling functionality backend
 - This would query all submissions from the database and count all reviews done on them, then randomly distribute one of the least reviewed submissions to the user requesting to do a review.

Thursday 11/4

- Added the slice shuffle function as a util
- Removed traces of the auto validation service as we decided there was not enough time to do it.
- Peer review button active when peer review is active
- Removed penile object from comment
- Review pooling updated, tested and bug checked, working.

Friday 12/4

- Tried to figure out why assignments were no longer updating
- Rolled back some files to figure out the issue
- Was database related as well as some computer issues on my part.

Wednesday 17/4

- Fixed a 'go backs' typo on a button
- Assignment page dynamically shows the users how many reviews they have left to do.

Thursday 18/4

- Buttons to do reviews removed after review deadline is over
- Reviewers count changed 0 value to 'any' so any number of reviews can be done

Sunday 21/4

- Fixed issue with simpleMDE where it wouldn't render because of a template comparison issue

Monday 22/4

- Fixed a bug where 'any' reviewers would crash the submissions overview page for admins
- Removed scheduler service link
- fixed updating review_id on assignment by deleting a foreign key
- Changed naming in assignment tabs for reviews received
- Added an error for reviews limit reached
- Made that message better.

Tuesday 23/4

- Added error message when changing email to an existing email
- Added a bootstrap badge to received reviews column on assignment page, to see how many reviews the user has received on their assignment without entering the tab.
- Updated submissions forms so they can be deleted if they have no reviews done on the form yet
- Same changes done to review forms.

Wednesday 24/4

- removed scheduled tasks table from database
- fixed deleting submissions forms properly
- updated function names for deleting review forms

Thursday 25/4

- added review_enable column to database table for assignments
- cleared out unused functions from model/assignment
- Removed unused structs
- cleared out unused functions from model/course
- cleared out unused functions from model/review
- cleared out unused functions from model/submission
- cleared out unused functions from model/user
- cleared out unused functions from model/userSubmission
- Also fixed functions still using these to use the ones in /repository instead.
- Changed review_enabled column to normal bool, not nullable
- added enable/disable functionality to assignment, so it's really disabled even if a review_id is specified.
- added enable/disable functionality to update assignment page

Friday 26/4

- enabling/disabling peer review functionality working for assignments creating and updating.
- If updating an assignment and the review_deadline is not set, it will be set one month after the assignment deadline
 - This won't have any impact as long as the peer reviewing is disabled
- changed the hasReview function to check the reviewEnabled variable instead.

Tuesday 30/4

- Fixed changing passwords on users in admin/manage students
- Changed session messages to use the gorilla sessions.flashes instead
- Merged the branch into master.

E.2 Brede Logs

Brede's Daily Log

Week One

Monday - 04/02/19

Started late today because of Super Bowl airing until 04.00 at the night. * Finished setting up the environment. * Started working/preparing for task 8(User Profile). * Watched a tutorial about bootstrap.

Tuesday - 05/02/19

- Continued working on the design for the user profile page.
- Had a bug about not getting the data from go file to html file. I fixed it with changing the struct variables to have capital letters :)))
- Almost finished with #8, just need to get actual user information and actually change the information.
- if the user wants to change information, it happens on the same side.
- information is displayed in input element for making the changes easier.

Wednesday - 06/02/19

I overslept 45min today, but that's fine, I'm just working 1h 45min longer :) * I found a bug yesterday with adding a secondary email, that I need to fix today * A new bug about the form in user.html is sending unvalidated input to the handler. I/we choose to not fix this not, but added an TODO to fix it later :) * We were a bit unsure how to update user user information. Since you can choose to only change one thing or all the information, should we request a change to the db for each change or all together every time? * Solution: Different queries to different changes * Another bug with getting the password-hash from the DB, it only works when the secondary-email is not NULL (I think). * Solution: Ask Johan for help. * He kinda fixed it with sql.Nullstring, it still doesn't work right :/ * Today was just bug after bug, not a fun day, I also worked an hour longer so I can leave earlier on Friday.

Thursday - 07/02/19

I overslept 30min today, I was supposed to start one hour earlier. * The real courses show up on the profile now :D * I had to make an own function for getting the password hash because of an annoying bug >:(* User can now change, secondary email, name and password :D * Only need to refactor some code, write some tests and lint and add some confirmation that the information is changed * Fixed error where the page went blank after submitting form. It was solved by requesting the view function to start again. * Didn't work a full day today because not good :/

Friday - 08/02/19

I'm going home today to celebrate my birthday, so I'm leaving 2hours earlier * Fixed the hash bug, Johan saw that i 'fixed' it on the wrong variable so now the hash appeared too! * Fixed the input validation bug, I had to switch from onclick to onsubmit that started the javascript script in the button-save. * Merged with master and got some problems, but Johan helped me fix it. * Almost done with #8 now, only need some refactoring and I have to fix one test.

Sunday - 10/02/19

I'm 5h too short for this week, so I have to work some today too * Started up with setting up the environment on my laptop since I'm home and not in Gjøvik. * I'm going to fix one test today and Johan is refactoring the code, so after that, I can make a pull request. * Merged my branch to master

Week Two

Monday - 11/02/19

- Started on #16 - Logging to database - Log stuff that the user and/or admin does
- Everything went fine for now at least, that's nice 😊
- Had two meetings and plan to discuss new project structure
- I made a powershell script that runs go fmt,vet,lint,cyclo and test. I did it to make the linting/testing go faster and more "clean"

Tuesday - 12/02/19

- I started on looking how the logs table will be.

- I created an struct for keeping the log data for easier use and fewer parameters.
- Logs when the user change name/email/password now.
- bug: Added foreign keys to logs, if course,submission or assignment id is blank, it doesnt work. So I have to figure out how to send nil instead of a number to the db.
- I know one way to fix the bug, but it's too much and messy code :/ (hella many if-else), and that would be too awful.
- This day went more to thinking about how to solve something than actually do it

Wednesday - Birthday edition 13/02/19

Wooooo birthday boi, halfway to 44 * I still had a bug with adding nil instead of int to the db with #16. I solved it with creating nasty if-else statements, I hope to refactor it later. * I also commented a lot for the function so it's more clear how to use it :D * Pushed #16 to master and started on #23. Need to fix homepage first tho :/ * I'm unsure of how to go further as the home page seem to just be empty on purpose and other stuff. * I think I can finish tomorrow tho <3 * Nothing much happened today :/

Thursday - 14/02/19

I am 22 now, also the power went out home, so I went to the School of NTNU * Started on restoring home page, currently stuck in a bug where the post function in index.go won't start. * I can now check if with a given course id, if it exists. * I need to figure out how to confirm that the user joined a class. I want to start an js script from go :/ * Solution: Global variable and send to template * Can now add user to course through home page :D * Course have an unique ID now, got inspiration from [here](#), I chose github.com/rs/xid because it was the length I wanted and quick and easy to use. * Added logging for create/join course

Friday - 15/02/19

- Added functionality for adding user to course through link when the user is logged in and when the user logs in.
- Added functionality for adding a new user to course through link.
- Added a test for RegisterGET handler :)
- Short day I guess, I'm going to write the report today, also -> no bugs today woo.

Week Three

Monday - 18/02/19

- #23 Changed courseID back to int and auto increment so it's similar to the other tables. But I added a column for hash instead.
- #20 Started on home page, but could only finish one of two tasks since assignments isn't done yet.
- We almost didn't have anything to talk about today with the [supervisor](#).
- I'm going to start on [#22 - Admin FAQ Page](#) now :D
- #22 Looked at [this page](#) for inspiration to the faq page, I liked the animations and stuff, but... If I use markdown instead it would be way easier to just add a new faq in frontend and is over all less code and easier to implement. Agreed with [Project Owner](#) to use md, it's also more consistence with this solution.
- Have to find out if we just store a hardcoded md file for faqs or make it possible to edit in the front-end by any teacher tomorrow.

Tuesday - 19/02/19

- #22 Decided on storing the md in db and let any teacher edit it, but also log every update.
- I chose to copy some of the design [Johan](#) used for course page to keep it more constance all over <3
- I have some bugs on the faq site, but the main functionality is all done soon.
- I also moved db functions from shared/db to model and gave a temp fix to the extremely annoying go lint errors...
- All functions for faq is now done, each time a teacher updates the faq, the time is added IN NORWEGIAN TIME, this has to be written somewhere as we talked about with the [Project Owner](#) yesterday.

Wednesday - 20/02/19

- #22 Tried to switch to a new package to get the text in the textarea editor to show, it still didn't work >:(
 - I think the problem is that id doesn't load again when the tab is shown.
 - It was bootstrap all along :(
 - I fixed the problem by removing the tabs and having the edit page on it's on page.
 - I also changed the design so it's more consistence to the admin dashboard.
 - Not really sure how to write tests for #22 yet so I'll leave that for later.
 - Finished for now
- Started on [#21 - Admin Page Dynamic](#)
- Used some time looking at a pull request

- Updated the `getCoursesToUser` to get the courses sorted by the year and then semester in descending order. It's nicer this way and more effective to have current classes at the top where they are easy reachable.

~~Thursday - 21/02/19~~ Ski day!

Friday - 22/02/19

Ski day was rough * Looked at pull request * #21 Merged main to branch * Displaying courses and assignments in dashboard, course and assignment for admin now. I need to make assignments in sorted order by deadline and display only active courses and assignments on dashboard, and then I'm done.

Week Four - One month woo

Monday - 25/02/19

- Continuing the work on #21
- #21 I created a new function in `model/assignment.go` called `GetAllToUserSorted()` which gets all the assignment to the user sorted by the deadlines desc.
- Course and assignment is displayed in correct order now
- Had some meetings, played some Minecraft, back now
- Made a pull request for [#21 - Admin Page Dynamic](#)
- Starting later on [#18 - Assignment Delivery page](#)

Tuesday - 26/02/19

- Made an instance `BredeVM` on Openstack
 - Tried to do stuff there but I didn't understand shit
- Going back to [#18 - Assignment Delivery page](#) again
 - Started on front-end draft for delivery page
 - Stuck on how to get the forms value out of the sql statement #bug
 - It was just stupid errors that shouldn't happened! >:([Johan](#) helped me see them again <3
 - I'm going to die now, bye! <3
 - I'm back for another 1h, Made a shitty draft of the assignment delivery page, I think I can be done within two days actually.
 - Added countdown on delivery page, inspiration from https://www.w3schools.com/howto/howto_js_countdown.asp

Wednesday - 27/02/19

I overslept 1h today * Continuing on [#18 - Assignment Delivery page](#) * Made a function for getting name and code from course * I used 2h one one pull request because I don't like Openstack... * And now I'm going to merge with master -_- * I'm just trying to run the project now... shit day * Almost all day has gone to Openstack and trying to run the project local * [Johan](#) fixed docker in IntelliJ <3 can now run it local from intellij * Environment is up-ish again and I'm gone

Thursday - 28/02/19

- After 1h, I can finally start coding again, since the env is completely up!
- Added functionality to get form values in an array
- Added functions for inserting and getting answers from db
- Merging with master
- Added test values in db, but it took some while since I didn't get errors about what was wrong :/

Friday - 01/03/19

- Added test data to `.sql` file
- Users that has delivered can see what they have delivered.
- Users can update what they have delivered
- I made the design much better
- Fixed a bug with deadline
- Added real link to see assignment button
- Done with [#18 - Assignment Delivery page](#), thought I would use 21h, actually used 24h 17min, this was mostly because I use ca. 7h all together to get the project starting.
- worked more on 18 do improve consistency and slicker design

Sunday - 03/03/19

- started on [#32 - Admin Show User Submissions](#).
 - Can now show list of all students in course for each assignment
 - The teacher can see the submission outside admin/ because other students that are going to peer review it should also see the submission, and this way I am saved from duplicate code :D
 - Merged with main
 - Checks if review user is verified now
 - Gets the links to the submissions user is going to review

Week Five

Monday - 04/03/19

- Had some meetings
- Started on [Alpha fix functionality Req](#) with Svein and Johan

Tuesday - 05/03/19

- Fixed joined course bug

Wednesday - 06/03/19

- Course hash is now displayed to Admin
- It's only one link for joining course now
- Admin can copy the one join course link
- Changed name to Full name and made it unchangeable
- Switched to PRETTYTIME in faq and updated the faq

Thursday - 07/03/19

- Hafjell

Friday - 08/03/19

- Added confirm to user if admin wants to delete submission form or not.
- Updates submission_id in assignment even if it's nil now (That's good)
- Former submissions will now be deleted if submission form is changed

Week Six

Monday - 11/03/19

Merged following to master

- * Fixed join class bug
- * Fixed submission/review bug
- * Updated faq to be more professional
- * Fixed it so it's only one join course link now
- * When a user updates an assignment, the submitted time is updated too
- * Also added TODO : time on places that used time, for later refactoring

- Starting on Hardcode in norwegian timezone- alpha and Display PRETTYTIME in Norwegian format
- Prettytime is fixed
- Go function for norwegian time is done
- Bug on js function for norwegian time

Tuesday - 12/03/19

- deleted unnecessary .sql files
- fixed schedulerservice error, I used an function from webservice before
- I changed some data to be better testdata in .sql
- Added two new functions

- one for padding numbers with 0
- and one for replacing `toISOString()` so it doesn't alter the time before formatting.
- Also changed go and js `getTimeNorwegian()` to use locale "Europe/Oslo"

Wednesday - 13/03/19

- Removed timezone from sql and use go lang instead, this is so it's easier to change time later
- Tried to remove go.mod/.sum but we needed them after all
- Fixing assignment card
- Added time zone on date formatting
- Also added env var for the timezone so it's easier to change later
- Changed to transaction some places in SQL queries
- Changed logtodb to return error instead of boolean

Thursday - 14/03/19

- I'm waiting for 3 pull request to be passed wohoo
- Starting on forgotten password
- Finished on forgotten password
- Not the best code, but it works for the alpha :D
- Also finished `don't show review name`

Friday 15/03/19

- Merged shit together

Week seven

Monday 18/03/19

- Fixed some alpha functionality

Tuesday 19/03/19

- Fixed some alpha functionality

Wednesday 20/03/19

- Fixed some alpha functionality

Thursday 21/03/19

- Went to Hafjell

Friday 22/03/19

- Checked for bugs
- Fixed dups email in register
- Fixing `stay-on-same-page-after-managed-student`

Week Eight

Monday 25/03/19

- Had some meetings
- Ran through the cycle of reviewing assignments

Tuesday 26/03/19

- Fixed join course through hash error and enhanced it a little by searching for the hash in the given string

Wednesday 27/03/19

- Sorted participant list by teacher and then name
- Looked at prettytime bug, it was an user error
- Updated go to 1.12.1, I have to fix some errors because of that now...
- Active assignments stay on home page until review deadline goes out
- Active assignments display Submitted or Not submitted for students only now
- Fixed deadline passed design on home and in course + the rest in admin without delivered status

Thursday 28/03/19

- Peer Review is on every assignment card now, admins also
- Review deadline can not be before Assignment deadline
- Also fixed a bug in create assignment that removes submission/review form when an error occurred
- And smaller stuff.

Friday 29/03/19

- Fixed some bugs on pullrequest and merged to master

Week Nine

Monday 01/04/19

- Created a new gmail and started on mailservice
- Can already send an email since it's only 3 lines of code, too easy :/
- Function in mailservice for resetting password is ready
- Changed from https to ssh!!!

Tuesday 02/04/19

- Added check to see if the email exists in the database
- Added feedback to the user
- Fixed EmailExists function
- Added mailservice to webservice, it works to get email from front-end now <3 :D
- Added authentication to the mailservice, so that only the system can trigger mailservice
- Added new table in db for forgotten passwords OBS! TODO brede : ADD THIS TO PULLREQUEST!!!!!!!!!!!!!!!!!!!!!! <3

```
CREATE TABLE `validation`
(
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `hash` varchar(64) NOT NULL,
  `user_id` int(11) NULL,
  `valid` tinyint(1) NOT NULL DEFAULT '1',
  `timestamp` datetime NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`user_id`) REFERENCES users (`id`)
);
```

- Now adds to the table if the email is correct
- Implemented check for valid hash
- Changes password and added a new column in forgottenPass
- Also logs password has been changed
- Added some tests for the handlers
- Implemented two more functions, one for sending email to single recipient and one for multiple

Wednesday 03/04/19

- Fixed forgotten password email message text
- Fixed goFixShit.ps1 script and added colours <3
- Fixed golint errors
- Merged emailservice to master
- Changed name from NTNU-Bachelor-Management-System-For-CS-Assignments to CSAMS short for Computer Science Assignment Management System in project and repo
- Possible solution to confirm email issue [#97](#) below

~~forgotten_password~~ validation table

id	hash	user valid	time
1	jhdsgjshgjfjsdgsdfgusdgyusdgyuysgeuyfesg	1 1	2019-03-24 14:42:00
2	jhdsgjshgjfjsdgsdfgusdgyusdgyuysgeu	32 0	2019-03-24 14:42:00
3	jhdsgjshgjfjfdgdfgfgusdgyuysgeuyfes	11 1	2019-03-24 14:42:00

pending_user table

id	email	name	password	validationID
1	dummy@hotmail.com	Dummy dummy	adhjguhgygedwuyGYDGWEIUYG	2
2	YEET@mymeat.com	Svein Klausen	hjfsuyefgsuyegfuiygef	3
3	swag@yolo.com	Johan Nilsen	kjewgfhiuefwoig	1

How to solve **confirm email** function

- Create user
- Create new entry in validation
- Add user to pending_user and link to validation
- Get email with one time use link (including hash from validation)
- Check if validation hash exists, not expired and is valid
- Get row from pending_user matching validationID to validation field id
- Move user from user_pending to user table

Thursday 04/04/19

- Don't have time to work a full day today, but i have 6h to spare from mon-wed.
- I added repository and service for the table users_pending also struct in model
- Users can now create user through email confirmation

Week Ten

Tuesday 09/04/19

- Fixed some smaller stuff
- Added send email to whole course functionality
- Have to fix it more tomorrow, but it works!

Wednesday 10/04/19

- Had some long meetings
- We have established the core functionality we need to be done now and added them to trello
- Added list of all student or private emails the email is going to be sent to

Thursday 11/04/19

I don't have too much time to work today, I will have to work some in the easter * Had to set up environment on my laptop * Checked out an pullrequest and fixed merge conflicts when it was merged to master * Created a new email csams.noreply@gmail.com and replaced the old with this * Added count of how many student, private and total emails there is * STarted on logs converting to service/repository

Friday 12/04/19

- Fixed nil pointer bug when logging, I used db.GetDB().Begin() instead of repo.db.Begin()
- Changed name on variable names fro logs in database to have underscore instead of camelcase
- Removed all uses of old logging function and changed name from log.go to logs.go
- Implemented one function for one logging event in service so it's easier to use, ut more dups
- Added new logging events
- Implemented functions for all student activity
- Added review_id to logs table and logs struct

Monday - EASTER EDITION - 15/04/19

- Finished the last functions for logging admin control student logging. It's a lot of functions now, and most are almost identical. I'm considering solving this another way to have less code...
- Implemented and tested:
 - Profile information change for students
 - Create/update/delete submission for students
 - Create/update user review for students
 - Create/update assignment
 - Create/Update/Delete submission form
 - Create/Update/Delete review form

Tuesday - EASTER EDITION - 16/04/19

- Found a bug when updating faq, converting to service and repository gave another nil panic bug, so im waiting to implement that for later
- Implemented and tested:
 - Join/create/update course
 - change student pass/remove student from course by admin

Week Eleven

~~Monday - 22/04/19~~

Tuesday - 23/04/19

- Implemented and tested Create/update/delete submission for student by admin
- Fixed log update faq bug, it was the wrong import for db...
- Implemented use of repository/service for admin faq and removed deprecated use in model/
- Added `group_id` to logs in the db
- Adding more logging events:

```
AdminAddUserToGroup
AdminRemoveUserFromGroup
AdminEditGroupName
AdminDeleteGroup
AdminCreateGroup
```

```
CreateGroup
EditGroupName
DeleteGroup
JoinGroup
LeftGroup
KickedFromGroup
```

- Added functions for logging admin group events
- Added functions for logging student group events
- Changed `Activity` from `logs` from string to int, to easier deal with the current 42 different logging events. Also way easier to put in the `.tmpl` file this way.

Wednesday - 24/04/19

- Completed explanation message for all logging events, just need to test them all
- Created a new plugin for go template `INT64_TO_INT` to easier get username when it's `sql.NullInt64`
- Got the name and role straight from the db one time, instead of asking too many times
- Added failsafes if a user has been deleted, both this user and affected user
- Tested all logging events that can be triggered now

Thursday - 25/04/19

- Added type (of log) to the log table in frontend
- Added another failsafe if activity id is unknown
- Logs is sorted by date desc now :D
- User has to confirm the secondary email now by clicking a link they get sent from the systems email

- Optimized code for confirming secondary email

Friday - 26/04/19

- Changed services from ex userservice to use services.User
- Removed commented code
- Removed some finished TODO's
- Fixed join course through register link now

Week Twelve

Monday - 29/04/19

- Fixed some issues with my branch for the pull request, ex: changing log type to plugin
- Changed name to managae_students from change_pass
- When giving students a new password, it's generated from js and is random each time now
- Added functionality for adding new faq and also log the event

E.3 Svein Logs

Svein's log

08/05/2019

Worked on

- Thesis
 - Requirements chapter
- Updating logs

07/05/2019

Worked on

- Thesis
 - Development Process chapter
- Updating logs

06/05/2019

Worked on

- Thesis
 - Development Process chapter

03/05/2019

Worked on

- Thesis
 - Development Process chapter

02/05/2019

Worked on

- Thesis
 - Development Process chapter

01/05/2019

Worked on

- Thesis
 - Development Process chapter

30/04/2019

Worked on

- Thesis
 - Development Process chapter

29/04/2019

Worked on

- Fixed filtering in choices, on the Form Builder

27/04/2019

Worked on

- Added more views for group-feature

26/04/2019

Worked on

- Worked on another project, other course

25/04/2019

Worked on

- Worked on another project, other course

24/04/2019

Worked on

- Worked on another project, other course

23/04/2019

Worked on

- Groups back-/front-end
 - Creating/joining/leaving groups
 - Functions for future implementations

22/04/2019

Worked on

- Group back-end

17/04/2019

Worked on

- Removed some more deprecated code
- Fixed error on updating assignments
- Group-based back-/front-end
- Fixed old bug in deleting submissions
 - Renamed functions for convenience

15/04/2019

Worked on

- Removed some deprecated code
- Added new templates for group-delivery
- Added code for implementing groups
- Simple refactoring, usage of services

12/04/2019

Worked on

- Group-delivery
 - Added groups to the database
 - Let user create, join and leave group

Bad

- Big task with little time to do (group-delivery)

Good

- Added some missing commenting from before

Decisions

-

11/04/2019

Worked on

- Added a new handler/view/page for reviews done by a user (admin-view)

10/04/2019

Worked on

- Added a new plugin for checking if user had been reviewed
- Added current user to all views
- Added ability to update reviews after submission
- Merged with master

09/04/2019

Worked on

- Testing exporting data to Excel-file
- Refactoring some code
- Merged with another branch

08/04/2019

Worked on

- Fetching raw data about review scores
- Processing review-score data
- Displaying review-data

04/04/2019

Worked on

- Merged with master branch
- Fixed imports after renaming of the repository
- Fixed import path for config-file

Good

- New laptop worked fine

Bad

- Previous laptop broke, so tested out a new one

03/04/2019

Worked on

- Implemented Sprintf for templates
- Added link for url-fields on rendering for some users who could not open the url's
- Merged with another branch

Decisions

- Shorten down float numbers to two decimals for better readability

02/04/2019

Worked on

- Fixed a bug on the Form Builder
- Made some QoL changes to the design on submission-tables
- Removed label on comments-title (Feedback from students)
- Worked statistics for assignments, calculating average score and standard deviation
 - Displaying statistics for the course-submissions
 - Calculating radio-selection linear
 - Created own functions for statistics outside handler

Decisions

- Created internal lib for calculating standard deviation and average score for reviews

01/04/2019

Worked on

- Merged with the master branch

29/03/2019

Worked on

- Fixing TODO's
- Updated code
 - More informative log messages
- Adding more security to the application
- Added selection for SimpleMDE (QoL)
- Removed deprecated functions

Decisions

- Redirect all attempts to go to /admin* to /

24/03/2019

Worked on

- Implemented default weight and field type to the form builder
- Implemented deleting submissions by admin
- Implemented Bootstrap-table extension
- Fixed bugs for beta
- Added default weight and field to form-builder
- Made submissions withdrawable
- Implemented feedback to student after submission

Good

- Deployed beta for testing with real students and a real assignment

Bad

- Wrong query-string overwrote all submissions

22/03/2019

Worked on

- Finished refactoring of service implementation
- Implemented weights display for reviews

Good

- Refactoring done

21/03/2019

Worked on

- Continue: Refactoring

Bad

- Not full group, so many hours of work in a few days

20/03/2019

Worked on

- Refactoring
 - Adding service/repository pattern

Good

- Works good, allow us to be less reliable of SQL queries

Bad

- Takes a lot of time, not full group, so did 15 hours today

Decisions

- Refactoring DAL (data access layer) away from controllers

13/03/2019

Worked on

Fixed things from feedback from the pull request

Good

- One step closer to a good alpha

Bad

Decisions

- Changed time formatting to norwegian time

11/03/2019

Worked on

Finished of the alpha work for reviews, displaying, submitting them.

Good

- Got the pull request out for the rest of the group, early on monday morning

Week 10 report

Worked on

- Review features
- Was gone 3 days for a work interview, lost does days for production

Bad

- Lost 3 days of programming due to traveling

08/03/2019

Worked on

- Created a few new view plugins for helping with the review-form
 - Increment
 - Split Choices

07/03/2019

Worked on

- Generating the review form
- Remodelling the `user_reviews` table

04/03/2019

Worked on

- Generating the review form

03/03/2019

Worked on

- Updating reviews
- Creating reviews

Good

- Found bug in Javascript-framework, but was easily fixed

02/03/2019

Worked on

- Updating submissions
- Creating reviews
- Tests for the view-plugins
- Parsing JSON to Javascript Form

Good

- Javascript framework for dynamic forms is working great, and easily changeable

01/03/2019

Worked on

- Nothing, was sick

28/02/2019

Worked on

- Assignment page
- View plugins

Good

- View plugins works great

27/02/2019

Worked on

- QoL (Navbar)

26/02/2019

Worked on

- Assignment update/view
- Setup OpenStack VM

Good

- Parsing `time.Time` to `datetime-local` from Go to HTML
- OpenStack: first configuration worked as a charm

25/02/2019

Worked on

- Settings page
- Assignment update/view

Good

- Group talked about the next weeks need-to-be-done stuff

Bad

- Wasted some time on the settings-stuff

Week 8 report

Worked on

- I have been working on the assignment and submission features for the lecturer-side of the software.
- Making the forms for creating the submission and review-forms dynamic. Doing this with Javascript, making a small reusable library with some configuration.
- Database for the dynamic form

Decisions

- The database tables were changed a few times throughout the week, but I think the last version will be more suited for the application, as it is scalable.

Good

- Got the story done, and feel the way it ended up, will be good for the rest of the application, making it scalable for other types of submissions, and assignments.

Bad

- Used too much time on this story
- Had problems with the database, cause of old relations, making it hard to merge new tables together with old ones.

22/02/2019

- Did some QoL on the form submits
- Fixed some bugs of redirecting and form-validation
- Researched best practice for Go

Research examples

Simple example with Courses

Go-code:

```
package research

type Course struct {
    ID int
    Data map[string]string
}

// Current usage
type Courses struct {
    Items []Courses
}

// Researched usage
type Courses []Courses
```

HTML-code:

```
<!-- Current usage -->
{{range .Courses.Items}}
    <div class="col">...</div>
{{end}}

<!-- Researched usage -->
{{range .Courses}}
    <div class="col">...</div>
{{end}}
```

20/02/2019

- Created a working tables for the assignments, submissions, forms and fields.
- Did pull request on a bigger card, but group found a lot of bugs, got most of them fixed
- Found a new way of doing the forms and fields tables:

forms

id	prefix	name	description	created
PK prefix for fields (HTML)	Display name	Description	..	TIMESTAMP

fields

id	form_id	type	name	label	description	priority	weight	choices
PK FK		type of field (text, radio, checkbox)	Name of field (HTML)	Label (Display name)	Description	Order number	Weight for grading	Choices (Array, split by ',')

19/02/2019

- Short day of working. Got the basic framework for the dynamic form together

18/02/2019

- Talked with product owner about the database-design for the custom forms, and found a solution together that will not be too hard to implement.
 - Later found a more generic way to design the database, with a forms and fields table:

Forms-table

id	name	description	created	prefix
PK	Display name	Description ..	TIMESTAMP	prefix for fields (HTML)

Fields-table

id	data	order	form_id
PK	JSON needed?		FK for forms

Week 7 Report

This week I have been looking into data structure for the dynamic review form, and where and what should take care for the input/output for the form. From the research done, it seems like Javascript will be the best choice of creating and parsing data to strings, with JSON, as Javascript is well-equipped with JSON-functions.

The database is also a challenge, because of the dynamic form, the database has to be design for a agile software, that needs to be flexible today, as well in a few years. I think I have found a good solution for the desgin of the database, but it is hard to implement cause of the auto-generated schema we are using from MySQL Workbench, but I think we need to rewrite the database-schema, and look more into normalization of relational databases, to make it flexible enough for the requirement-specs.

Have also been working on the last touches of the restructuring of the project, making it more flexible with a MVC-architecture. Making less files, and all the files in a folder, does the same type of tasks.

15/02/2019

- Updated teh time.Time convert-function from datetime-local (HTML).
- Looked at the database for assignments,

14/02/2019

- Worked on the design of assignments with the peer reviews, thinking about the data-structure, and how to store it, and where it should be written/read.

13/02/2019

- Worked on the assignments page for admins, looking at data-structure, and an easy approach for designing the form.

12/02/2019

- Worked on restructuring the folder-/file-structure for the project. Making it a MVC-structure.

11/02/2019

- Started the planning for the project restructure, for a more efficient development later on in the project.

Week 6 Report

I have worked mainly with Bootstrap to get a concise design on every page of the application. Templating has also taken up quite a bit of time, as I did not have much knowledge of this before, and had to read about this, and talk to the other team members about this issue. It was resolved in a straightforward way, but afterwards I see that it could have been solved even better, with the creation of features to simply enter simple parameters to be able to change the whole page, as well as help reuse code.

I have also worked part with the dynamic form to be used for peer review. Since I have a good deal of knowledge from before with dynamic front-end programming, this task went pretty smoothly. Met some challenges with the prioritization of fields, but found a solution in the end that makes it easy for both me and the person who will set up the form. Will look later on this, but it works pretty well as it does now. Only thing missing is adding a button to remove items from the form.

The group had a discussion on the database structure and form structure of the application, and found that we could reduce the number of fields in the database, using Markdown as word processing, which means that user has great freedom in relation to text and content that can added to the subject page.

08/02/2019

- Worked more on the assignments-form
- Had some problems with ordering on the form, but found an easier solution that works perfectly fine

07/02/2019

- Added navbar to every pag
- Worked on the assignments-form, for peer-reviews
- Refactored some of the previous code, cause of navbar on every page

06/02/2019

- Added another function for loading JSON-data from file
 - Dummy-data for reuse, displaying data on site
- Created form for creating new courses
- Created am flexible error handler

05/02/2019

- Made some simple data-structures for page-data, and made a more agile template for the page-title, menu and content.
- Made a simple function for loading data from JSON-file
 - Menu-items

04/02/2019

Worked on nested templates, and design for the website, both for the main site, and the dashboard for admin. Created one more test for the HTTP-requests, checking for response body-size.

01/02/2019

Worked on the project plan: * Rewrote some phrases * QA

31/01/2019

Worked on the project plan: * Risk Analysis * Main division of the project

30/01/2019

Worked on the project plan: * Subject area * Limits * Gantt-implementation

28/01/2019

Worked on the project plan: * Technology, Business, Project Group * Risk Analysis

F Peer Review Discussion

Detailed experience: Peer-Review Functionality

Situation:

The overall thought of the system was to have as maintenance free as possible, and thus automating most of the processes included in it. The auto validation functionality would just check the users code right after they submit it and give instant feedback, and the peer reviewing functionality would instantly give all the users an n number of pre assigned submissions that they would have to do.

Challenges:

Since the project was run on docker, the container doesn't have a direct access to Linux's scheduling functionality. The scheduling was needed because at the deadline of an assignment, everyone were to be assigned their reviews, thus I coded a scheduling service that would run asynchronous and you could register a given task with a set json payload to a specific time. This would then send the payload to the given task at the given time and launch the peer reviewing algorithm.

This worked like a charm, wasn't cpu heavy or hogging resources, but the problem was that the algorithm I designed for the system was thought to take every user that has delivered the assignment in a list, randomize it and make everyone review the n number of reviews of people behind them in the list. In a way creating a 'circle' of reviewers and reviewees. The problem we discovered with this scheduling and automated approach was that it needed excessively hard algorithm to include late submissions in this 'review circle'.

To include a late submission before people had started reviewing was easy, just delete the last review circle and run the main algorithm again. But when the users had already performed a large set of reviews I had to create an algorithm that would find a set of users that had not done any reviews yet, take them out of the circle, link the circle back up where I took them out, then create a new smaller circle with the late submission and the users I took out of the main circle.

Removing a user from the course would also prove troublesome because they might have received reviews on an assignment, removing them would cause some users to either do more than the stated amount of reviews or some users to receive less reviews.

After two weeks of writing algorithms for every special use case to these problems, the product owner decided that the automation was not the way to go anymore because of the sheer complexity of the functionality needed to be written, implemented and tested, for functionality that were technically never to be used.

The product owner then went away from the idea of automating the process, to instead create an algorithm that holds a 'pool' of submissions, and then just randomly assign a user to the submission with the least amount of reviews on it. This meant that 4 weeks of developing the automation algorithms, code and implementation it needed had to be deleted.

The review 'pooling' solved all the issues we had above, but it is a shame that all the time spent and code written had to go to waste. But that is just how it is with so agile development models sometimes.

Experience:

I learned a lot about multi-service programming and making the systems talk to each other on different ports, as well as making a flexible and extendable scheduler. I think I got a lot better at thinking out and writing efficient algorithms and overall finding solutions to problems.

More planning and thinking about the automation process could maybe discover the issues I experienced after many weeks of work, and could be prevented. But at the time we wanted it to be automated so it was the way to go.

G Screenshots

Please sign in

[No account? Sign up](#)

[Forgotten password?](#)

[Privacy Policy](#)

Course hash [Join course](#)

Courses

IMT2531
Graphics Programming

[See Course](#)

IMT3673
Mobile Programming

[See Course](#)

 [Edit Profile](#)

School Email: @stud.ntnu.no

Secondary Email: Secondary Email

Number of classes: 2

Courses	
IMT2531 Graphics Programming	Visit
IMT3673 Mobile Programming	Visit

IMT3673 | Mobile Programming spring | 2019

Home Assignments

Mobile Programming Course, NTNU Gjøvik

[Go Back](#)

Lab 1

[Peer Review](#)

Deadline: 23:59 26/03/2019 +0200 CEST

[Assignment](#) [Peer Review 2](#) [Received Reviews](#) [See submissions](#)

Lab 1 Submission

Dashboard [Homepage](#) [Sign out](#)

Courses New

- IMT2531** ID: 3
Graphics Programming
Hash: bj5em8hmvntq63m59lg
- IMT3673** ID: 1
Mobile Programming
Hash: bib6o0448c618r2jjdm0

Assignments New

- Exam
- Lab 1
- Lab 2
- Submission
- Submission
- Submission

Navigation: Dashboard, Courses, Assignments, Logs, FORMS, Submission Forms, Review Forms, SETTINGS, Manage Students, FAQ, Settings

[Homepage](#) | [Sign out](#)

Dashboard

- [Dashboard](#)
- [Courses](#)
- [Assignments](#)
- [Logs](#)

FORMS

- [Submission Forms](#)
- [Review Forms](#)

SETTINGS

- [Manage Students](#)
- [FAQ](#)
- [Settings](#)

Lab 1 submissions

37 / 38 submitted
Reviews per student: 2

Statistics (Review)

Entries	66
Absolute Min / Max	0 / 19
Local Min / Max	4.13 / 19.00
Average score	15.16
Average percent	79.78%
Standard Deviation	3.13

[Download Report](#)

This spreadsheet is only confirmed working with Microsoft Excel (latest update: 08.05.2019)

#	Name	Email	Submitted	Preview	Update	Delete	Reviews Received	Reviews Done
1	[Redacted]	[Redacted]	2019/03/29 16:07 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2
2	[Redacted]	[Redacted]	2019/03/28 10:50 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2
3	[Redacted]	[Redacted]	2019/03/27 19:38 +0200 CEST	Preview	Update	Delete	Reviews	1 / 2
4	[Redacted]	[Redacted]	2019/03/27 18:52 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2
5	[Redacted]	[Redacted]	2019/03/26 20:01 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2
6	[Redacted]	[Redacted]	2019/03/26 12:34 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2
7	[Redacted]	[Redacted]	2019/03/26 07:52 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2
8	[Redacted]	[Redacted]	2019/03/26 07:47 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2
9	[Redacted]	[Redacted]	2019/03/25 23:47 +0200 CEST	Preview	Update	Delete	Reviews	2 / 2

Dashboard | [Homepage](#) | [Sign out](#)

Create new Submission Form

FORMS

- Dashboard
- Courses
- Assignments
- Logs
- Submission Forms
- Review Forms

SETTINGS

- Manage Students
- FAQ
- Settings

Form Name

Enable weights

Default type: NUMBER

[Add a new Field](#)

Question 0

Type: TEXT

Label:
Max length: 256 characters.

Description:

Enable comment
Enable this will append a text area after the field with the option to enter an comment to their answer.

Make field mandatory

Required [Remove this field](#)

Question 1

Type: NUMBER

Label:
Max length: 256 characters.

Description:

Enable comment

H Trello Board



Backlog

FRONTEND **DATABASE**
BACKEND
 User roles

FRONTEND **DATABASE**
BACKEND
 IssueLogger
 0/2

FRONTEND **BACKEND**
 Site settings
 0/3

DATABASE **BACKEND**
 Move FAQ into settings

FRONTEND **DATABASE**
BACKEND
 Peer Review page
 0/1

DOCS
 Documentation for form-builder.js

Up Next (6)

FRONTEND **DATABASE**
BACKEND
 Delete functionality for assignment,
 course and forms

FRONTEND **DATABASE**
BACKEND
 Let users decide if they want emails
 about course stuff, they will get one
 from system anyway

FRONTEND **BACKEND**
 Message system


BACKEND **BUG**
Fix tests

In Progress (3)

Peer Review (3)

Code Done


BACKEND **BUG**
Fix delete form from assignment bug

DATABASE **BACKEND**
Export submission/review data to excel
🔗 2 🟩 2 🟩 1 merged 

FRONTEND **DATABASE**
BACKEND
Users can ask for reviews, review-pooling
🔗 2 🟩 2 🟩 1 merged 

FRONTEND **DATABASE**
BACKEND
Add group functionality
🔗 1 🟩 1 


DATABASE **BACKEND**
Remove/convert deprecated database queries in model folder to service/repository
🔗 1 🟩 1 🟩 1 merged 

FRONTEND **BACKEND**
Let teachers send group email to course
👁️ 🔗 2 ✉️ 4/4 🟩 2
🟩 1 merged 

FRONTEND DATABASE
BACKEND

Log every new entry/update in the db, and show it to admin


1 2 2 1 merged



FRONTEND DATABASE
BACKEND

Confirm secondary email with email validation

1 2 2 1 merged



BACKEND BUG

Fix register link, join course

1 2 2 1 merged



FRONTEND DATABASE
BACKEND




Edit review after submitting



FRONTEND DATABASE
BACKEND DESIGN BUG

Christopher Feedback

2/2

FRONTEND DOCS


Write a terms of use kind of thing

1 2/2 1 1 merged

DATABASE BACKEND

#29 - Scheduler Service

7/7




FRONTEND BACKEND

Course Page Dynamically list active and not active assignments


FRONTEND BACKEND


#26 - Front Page Dynamically list 'active' assignments


1 1 1 merged




FRONTEND DATABASE
BACKEND BACHELOR
 Alpha Functionality Requirements
 👁️ 📧 27/27 



FRONTEND BACKEND
 Dashboard - List Current Courses and
 Assignments from this semester/year
 ☰




FRONTEND DATABASE
BACKEND BUG
 Alpha fix functionality Req
 👁️ 📧 30/33 

FRONTEND BACKEND BUG
 #30 - BUG - from #23 join class, gets
 window.alert all the time after joining
 course
 👁️ 

FRONTEND DATABASE
BACKEND
 Errors
 👁️ 📧 14/16 

FRONTEND DATABASE
BACKEND
 #18 - Assignment Delivery page
 👁️ 💬 2 📎 3 📧 1/1  2
 1 merged



FRONTEND BACKEND
 #9 - Assignment page
 📎 1 📧 5/5  1 

DATABASE BACKEND
 #28 - Peer Service
 📎 2 📧 8/9  2  1 merged



BACKEND
#21 - Admin Page Dynamic
👁️ 📌 3 📧 3/3 🟩 3
🟩 2 merged




FRONTEND DATABASE
BACKEND
#12 - Create Assignment form/page
📌 2 📧 6/7 🟩 2 🟩 1 merged



FRONTEND BACKEND
#22 - Admin FAQ Page
👁️ 📌 2 📧 12/12 🟩 2
🟩 1 merged




FRONTEND BACKEND
#10 - Course participant page/list of users in course
📌 2 📧 1/1 🟩 2 🟩 1 merged



BACKEND IMPLEMENT LATER
#20 - Front Page Dynamic
👁️ 📌 2 📧 1/1 🟩 2
🟩 1 merged



FRONTEND BACKEND
#7 - Course page
📌 2 📧 4/5 🟩 2 🟩 1 merged



DATABASE
#2 - Initial database tables
📌 1 📧 8/8 🟩 1



BACKEND

#3 - Go Router / Web server

2 9/9 2 1 merged



FRONTEND

#4 - Home page / Landing page

2 8/8 2 1 merged




FRONTEND DATABASE

BACKEND

#5 - Login/Register page

2 6/6 2 1 merged



FRONTEND BACKEND

#6 - Admin page

2 3/3 2 1 merged




FRONTEND DATABASE

BACKEND

#8 - User page

2 13/13 2

1 merged




FRONTEND DATABASE

BACKEND

#11 - Create Course form/page

2 4/4 2 1 merged




FRONTEND DATABASE

BACKEND

#32 - Admin Show User Submissions

2 3/3 2


1 merged



FRONTEND DATABASE

#14 - Database connection class

2 4/4 2 1 merged




DATABASE BACKEND

IMPLEMENT LATER

#16 - Logging to database

4 2 5/11 2

1 merged



FRONTEND

#17 - Add navbar to every page

2 2 1 merged



BACKEND

#23 - Join Class Functionality

2 4/4 2

1 merged






Docs Done

DOCS

#25 - Write half a page weekly-report




17 Feb 4/4

DOCS

#15 - Write half a page report on what you did this week, what went good, what went wrong, what can be done better




10 Feb

BACHELOR DOCS

#1 - Prosjektplan

1 Feb 1 11/11 1

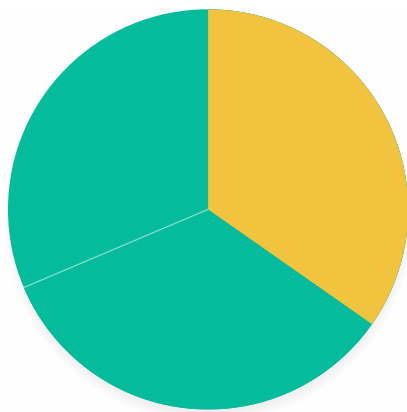
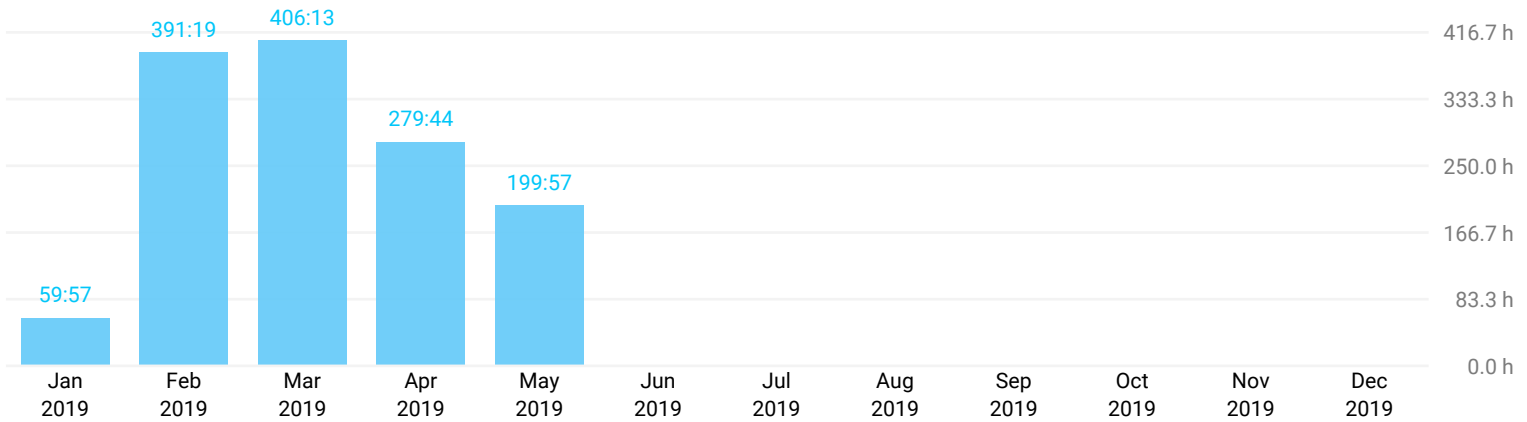
I Toggl Summary

All time tracking was done through [Toggl](#).

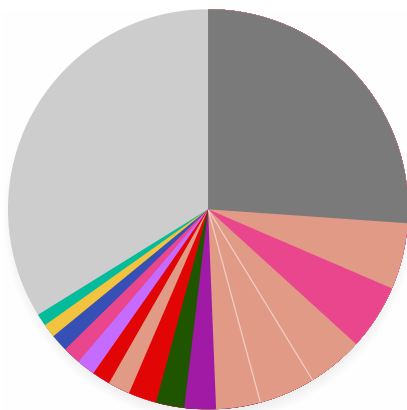
Summary Report

January 01, 2019 – December 31, 2019

TOTAL HOURS: 1337:11:54



USER	DURATION
Johan Aanesen	465:12:20
Brede Fritjof Klausen	453:08:40
Svein Are Danielsen	418:50:54




TIME ENTRY	DURATION
Without description	348:19:05
Report writing	72:18:20
rapport	70:53:18
Bachelor Rapport Skrivning	60:48:43
#Alpha functionality	59:45:29
12 - create assignment form/page	47:30:50
#28 Peer Review Service	34:12:55
Review Pooling	31:39:05
Logging	28:45:44
#29-scheduler-service	23:56:32
9 - assignment page	20:34:43
alpha-brede	20:15:38
#7 coursepage	19:14:28
Alpha-Brede	18:06:17
deleting scheduler n stuff	14:31:15
#alpha-brede	13:44:30

● Other time entries

452:35:02

USER - TIME ENTRY

DURATION

USER - TIME ENTRY	DURATION
 Brede Fritjof Klausen	453:08:40
#16 - Fix logging issue & start on #23	7:00:12
#18 - Assignment Delivery Page, rec 1	3:07:55
#18 - Assignment Delivery Page, rec 2	0:56:41
#18 - Assignment Delivery Page, rec 3	6:12:07
#18 - Assignment Delivery Page, rec 4	8:51:17
#18 - Assignment Delivery Page, rec 5	5:11:23
#18 and other stuff	2:28:55
#20 - Front Page Dynamic	8:01:37
#21 - Admin Page Dynamic	6:28:07
#21 - Admin Page Dynamic Part 2	1:07:52
#21 and meetings	7:15:42
#22 - Admin FAQ Page	7:09:57
#23 - Ivite link	4:20:07
#23 - test and refactor	1:45:42
#32 - Admin Show User Submissions	9:44:14

USER - TIME ENTRY

DURATION

USER - TIME ENTRY	DURATION
#alpha-brede	13:44:30
22 - bøg	7:21:28
Added ps1 script	0:10:00
alpha	1:43:07
alpha again	6:09:29
alpha fix	1:16:34
Alpha Fixes	4:59:30
alpha-brede	20:15:38
Alpha-Brede	18:06:17
beta-brede	7:13:12
beta-brede - emailservice	11:38:54
beta-brede - emailservice - confirm email	3:28:51
beta-brede - emailservice - email validation	9:52:18
beta-brede - emailservice - pullrequest	8:00:25
Beta-brede boii	7:16:02
Boring class, starting on #16	1:11:31
Checking out stuff - inkl. UML	0:17:48

USER - TIME ENTRY


DURATION

USER - TIME ENTRY	DURATION
configuring database	0:22:48
email research	1:06:09
emailservice	4:20:00
emailservice + logging	4:34:57
First week, Fifth day	5:05:05
First week, first day	0:18:12
First week, First day, part 2	3:38:59
First week, Fourth day	4:02:55
First week, Fourth day part 2	1:48:13
First week, Second day	7:03:44
First week, Third day	8:03:11
fixing logs	7:00:58
Gantt - meeting	0:28:18
going to school and continuing on #23	7:11:23
join course through register link	4:47:04
Logging	28:45:44
meeting	4:55:31

USER - TIME ENTRY

DURATION

Meetings and working on plan	3:33:00
Meetings and working on plan part 2	1:00:10
Moody Mondays	4:21:55
Openstack vm	3:20:37
Overtime	0:40:22
Planning day	0:56:10
Prosjektplan - Gantt	2:49:17
Refactoring, fixing of bugs and report writing	3:05:41
Report writing	72:18:20
Research and merging #8	0:24:46
Setting up enviroment and stuff	2:13:48
Without description	62:24:01

 Johan Aanesen	465:12:20
#10 Participant List	12:07:08
#11 Create Course Page	8:59:35
#11 Create Course Page - Testing	4:19:41
#28 Dockerizing The Project	9:35:11

USER - TIME ENTRY

DURATION

USER - TIME ENTRY	DURATION
#28 Peer Review Service	34:12:55
#29-scheduler-service	23:56:32
#29-scheduler-service && #Alpha functionality	8:20:36
#5 Register/login and setting up environment on laptop	5:40:17
#7 coursepage	19:14:28
#7 or #8, coursepage/userpage	3:29:57
#Alpha functionality	59:45:29
#Alpha functionality && Meeting	2:44:58
#research	1:48:49
#restructur	7:04:00
#Webservice scheduler frontend	4:23:36
:D updating project on main pc	0:51:52
Bachelor Rapport Skrivning	60:48:43
Database Connection Class	9:10:34
Day 2 :) #14 and #5 Session management and login / register	6:12:31
deleting scheduler n stuff	14:31:15
Fikse fiks	0:32:20


USER - TIME ENTRY

DURATION

USER - TIME ENTRY	DURATION
Filling trello with tasks and initial database creation	7:33:00
Functionality mapping	1:52:00
Meeting and Review Pooling	2:52:19
Meetings	7:46:33
Programming and wrapping up project plan	3:01:20
Programming start	8:48:00
Project meeting	3:50:03
Project planning	6:14:56
pull request n stuff	0:25:39
refactoring day!	2:17:31
Review Pooling	31:39:05
Review Pooling wrap up	5:14:25
Review Pooling wrap up / pull request	7:32:00
Review switch on/off	12:28:22
System Architecture	4:30:10
Updating Notes And Set up project on OpenStack	0:12:23
Updating weekly and daily notes	2:14:00

USER - TIME ENTRY
DURATION

Updating Weekly notes n challenges	0:26:29
Weekly Meeting	3:08:00
Weekly report and #8 pull request	0:42:24
Working on project plan	3:45:00
Without description	50:48:14

 Svein Are Danielsen	418:50:54
#4 - Home page / Landing page	10:11:46
#6 - Admin page	6:04:24
12 - create assignment form/page	47:30:50
17 - navbar	3:00:00
9 - assignment page	20:34:43
Go research	4:28:40
Monday planning	7:02:05
Project restructur	6:00:21
Prosjektplan	5:37:57
Prosjektplan, EDITED	2:00:00
rapport	70:53:18

USER - TIME ENTRY

DURATION

USER - TIME ENTRY	DURATION
Report	0:20:00
Without description	235:06:50