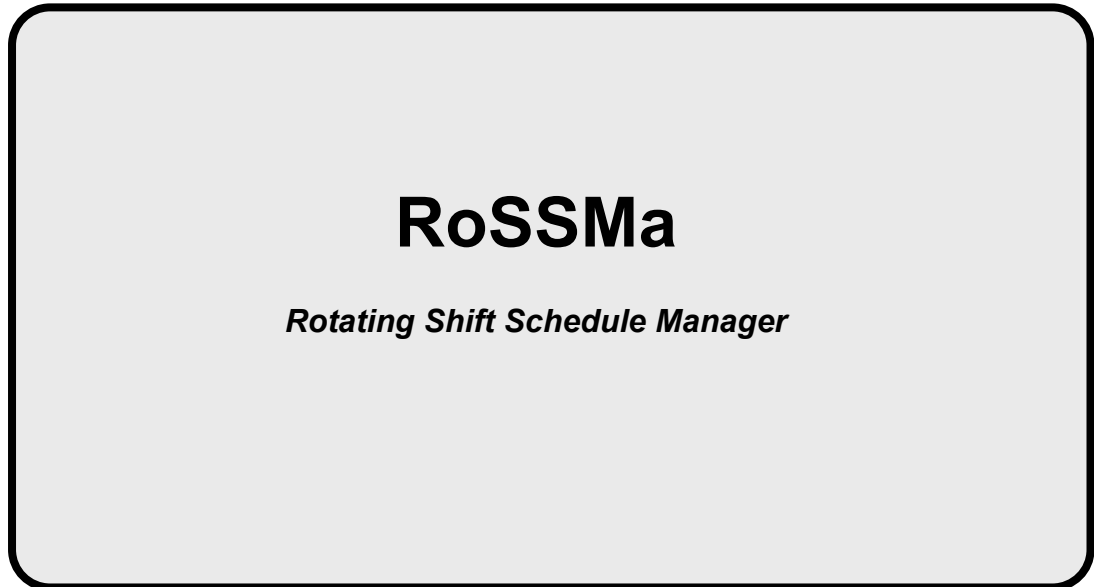


BACHELOR THESIS:



AUTHORS:

Glen M. N. Grongan
John Høegh-Omdal
Martin Jonassen

Dato: 19.5.2014

0.1 Summary

Title	RoSSMa - Rotating Shift Schedule Manager	Nr.	1
		Date	19.5.2014
Participant(s)	Glen M. N. Grongan John Høegh-Omdal Martin Jonassen		
Supervisor(s)	Frode Haug		
Employer(s)	Helsetjenestens driftorganisasjon		
Contact	Reidar Honningsvåg		
Keywords	Turnus, HDO PHP, Calendar		
Nr. of pages:	68	Nr. of attachments:	6
		Availability:	Open
Short description			
<p>This report describes RoSSMa and it's development process, a software designed to ease the management of rotation shift schedules for Helsetjenestens Driftorganisasjon. As a replacement solution, it needs to effectively fulfill everything the current implementation can accomplish, in which schedules are organized in highly advanced excel sheets.</p> <p>Rotating shift schedules are used in certain professions to ensure that their critical positions stay manned at every hour of the day, while ensuring that employees don't exceed the laws and regulations regarding appropriate durations of work schedules. It's accomplished by categorizing each day into clearly defined shifts which each last for a specific duration of hours, which are distributed into different types of weeks that are assigned to employees. These weeks are rotated between the participants each week, resulting in a different work pattern nearly every week. It's therefore critical to be able to review this schedule easily and hassle free.</p> <p>The solution was made web-based to take accessibility concerns into consideration, which is of substantial importance. The backbone of the software is developed in PHP as a result, so that the data can be accessed in HTML format by traditional browsers.</p> <p>The report is an in-depth look into the software, it's features and how they are implemented. It also describes the development process and the thought process regarding important decisions.</p>			

0.2 Preface

The Bachelor Project has been a great opportunity to foray into the realm of professional software development. As the culmination of three years of study and a chance to utilize all of our skills, we put a substantial amount of effort into making the case a reality. However, it's not all to our credit, and we would like to give our regards to the ones that have helped out along the way.

We would like to thank HDO for providing us with an exciting case to work with. Reidar Honningsvåg and Tor Kristan Hansen in particular, for all the useful feedback. Our insight into the world of an HDO employee is due to Reidar's efforts, and Tor Kristian's experience with a bachelor degree was a considerable benefit. The OREGO project that he completed together with Morten Holberg was additionally a great aid in structuring the report.

We would also like to thank Frode Haug for being accessible throughout the project. His guidance as our supervisor and input during the process has been valuable assets for writing the report.

0.3 Terminology

The English language has no suitable substitute for the Norwegian concept of "Turnus". **Schedule** is often used in its stead as a shorthand for **rotating shift schedule**. A schedule is an arranged list of week layouts distributed between people over a longer timespan. Each week contains **shifts**, the assigned working hours of a single person on a given day.

References to the **framework** means the underlying structure that determines how classes are implemented in the source code.

Contents

0.1	Summary	I
0.2	Preface	II
0.3	Terminology	II
1	Introduction	1
1.1	Project introduction	1
1.2	Team Introduction	5
1.3	Report Introduction	7
2	Requirements	9
2.1	Environment	9
2.2	Operational Requirements	10
2.3	Feature Requirements	13
2.4	Software Design Limitations	19
3	Design and Analysis	21
3.1	Similar Solutions	21
3.2	Views	22
3.3	Web Service	24
3.4	Progress Plan	25
3.5	Original Database	26
3.6	Tools	27
3.7	Risk Analysis	28
4	Development	30
4.1	Implementation	30
4.1.1	Framework	30

4.1.2	High-level System Description	35
4.1.3	Database	38
4.1.4	External Libraries	40
4.2	Ergonomics and Aesthetics	42
4.3	Testing and Quality Assurance	47
4.4	Encountered Problems	48
5	Installation	51
5.1	Downloads and Location	51
5.2	MySQL	52
5.3	Configurations	54
6	Summary	56
6.1	Discussions	56
6.2	Evaluation	62
6.3	Conclusion	66
	Appendices	69
A	Terminology	70
B	Meeting Summaries	73
C	Work Log and Progress Report	78
D	Progress Plan A	84
E	Progress Plan B	87
F	Project Agreement	89

List of Figures

2.1	<i>System Model</i>	10
3.1	<i>Excel Implementation</i>	22
3.2	<i>Early Design Mockup</i>	23
3.3	<i>Early Design Mockup</i>	24
3.4	<i>Early database draft</i>	26
4.1	<i>Model/View/Controller for this project.</i>	31
4.2	<i>Final database design</i>	38
4.3	<i>Template Picker</i>	43
4.4	<i>Different incarnations of a shift</i>	44
4.5	<i>Information message</i>	45
4.6	<i>Warning message</i>	45
4.7	<i>Navigation chart</i>	46
4.8	<i>Header</i>	47

1 — Introduction

The **Introduction** presents the case to the reader, delivering the scope of the project, the assigned students and the report itself. **Project Introduction** outlines the assigned task and related topics. **Team Introduction** gives a background description of the developer group, while **Report Introduction** provides meta information on the structure of this report.

1.1 Project introduction

1.1.1 Description

Helsetjenestens Driftsorganisasjon (HDO) have their employees working shifts using rotating shift schedules. These schedules are currently managed through an advanced excel document, with rigid functions controlling the rotation and various time calculations to ensure the integrity of the information. As a Bachelor Project, HDO wants a software that is both easily maintained by the administrators, as well as being convenient for the employees to use.

The goal of the project is to create a system able to replace the excel solution and make it accessible as a web interface. Requirements are centered around the ability to create a template, a type of “blueprint” that describes the structure of the rotating shift schedule. Arguably the most important functionality for the solution is to provide an interface for the employees to easily view the schedules and the deviations from a normal week, such as specific work tasks. Presenting different views to format the schedule is an important aspect in elevating the usability. The administrator module is centered around the capability to create and manage templates for schedules, organizing them in a database and make them an active part of the system

We initially contacted and met with three different project suppliers when the bachelor project started in October. HDO was the only project outside of campus, although the distance was largely insubstantial in comparison. They were also the only one without prior experience as a bachelor project provider. Coupled with an enthusiastic contact person that took the role of the consumer provided a lifelike experience that weighed our preference. Personal enthusiasm regarding the different projects was of course factored in as well.

1.1.2 Domains

Rotating Shift Schedule

Certain professional areas such as health care and support need to have manned personnel available at any possible hour of the day, throughout each week. Clearly defined schedules are a common way to solve this, in which each day is divided into different types of shifts that together cover all 24 hours. These shifts are then manned by different employees, ensuring that there's always someone available. Schedules often require a great amount of time management, because of laws that govern limitations on how long an employee is allowed to work under certain economic parameters. Working more than the indicated time will result in overtime, which follows other payout structures. In Norway, these rules are managed by Arbeidstilsynet.

One type of shift management is by using a *rotating shift schedule*. These types of schedules have weeks consisting of different shift compositions structured into a cyclical pattern over the course of set number of weeks. Employees are assigned to a week, and rotate through different types of weeks over the course of the entire schedule. When an employee is set on the last week of the schedule, the next week of that employee will be the first week of the schedule.

HDO

HDO install and maintains the systems required for “Nødnett”, the radio network built specifically for emergency and rescue services in Norway. It's created for fast flow of information during times of crisis. Such a network needs to be highly reliable and stable, as health support has to be available to respond to any event without delay. They

have implemented a rotating shift schedule to accomplish this.

Arbeidstilsynet

Arbeidstilsynet enacts the rules to protect employees in Norway. Schedules designed by HDO need to go through a validation process with Arbeidstilsynet to ensure that it properly follows the regulations they have created.

1.1.3 Target Audience

Report Audience

This report is aimed at anyone interested in gaining an understanding of our software and the underlying reasons behind our choices. It is primarily constructed for the examiner, but will hopefully find use as an academic resource in the Gjøvik University College library or for anyone intending to do further development on the project.

The project is written entirely in English, primarily because it eases translation of established programming terminology, and makes it approachable for a broader audience. Some of the screenshots of the software have been manipulated to accommodate for this.

Software Audience

The primary audience of the software are the employees at HDO participating in a predefined shift schedule. They need an easily accessible interface to get an overview of their shift structure, as well as additional information regarding their tasks.

The other target of the application are the schedule administrators. The software needs to provide them with a reliable way to create and export the template for schedules, and the ability to manage ones currently in effect.

1.1.4 Goals and limitations

Goals

- Result goals
 - The working software should replace the current excel solution without losing out on any preexisting functionality.

- Better usability for administrators
 - * Easier to learn.
 - * Less overhead.
- Make information presentable on any common internet browser.
- Easily maintained and expandable.
- Effect goals
 - Reduce the necessary steps for administrative tasks by 50%.
 - Remove the need to manually manage shift changes through email.
 - The software should be displayed correctly on all common browsers. Furthermore, HDO wants the shift schedule to be accessible through mobile devices.

Limitations

- Time: The deadline for the project, and consequentially this software, is 19.5.2014
- Language: Needs to be developed in a well established programming language.
- Modularity: The software needs to be easily expandable.

Restrictions

- If we run low on time, some requirements will be dropped, depending on necessary prerequisites and priority evaluations.
- Validating templates in the editor to meet laws and regulations was discussed during initial meetings with the product owner. This will not be implemented, as it would have required a broader knowledge of these rules and how Arbeidstilsynet approves and rejects schedules. The laws are also subject for change, which the system then would need means to accommodate for.
- We were provided the option of integrating the application with HDOs current user system, LDAP. We believe that there is too much overhead to integrate with the system, and would be too time consuming. The major concern comes in the difficulty of testing an implementation, so we found it more suitable to leave it outside the scope of the project.

1.2 Team Introduction

1.2.1 The Group

The three group members all started their Bachelor Degree in Software Development at Gjøvik University College in fall 2011. It has given opportunities to learn several different programming languages, with an emphasis on C++, Java and PHP. Additional courses focused on software design, security, development and data structures has prepared the members for this final task. Having shared several previous projects since the first shared effort during third semester has accustomed the group to working together. The members, and their assigned roles in the project, are as follows.

- Glen M. N. Grongan - Scribe, Secretary, Programmer
- John Høegh-Omdal - Project leader, Programmer, Design
- Martin Jonassen - Scribe, Programmer, Design

Roles

The role of Project Leader was given to John Høegh-Omdal, giving him the responsibility of controlling group decisions and managing project priorities. Glen M. N. Grongan is the secretary, which primarily consists of being the contact person of the group and documenting all important meetings. It's his role to maintain communication with the employer and supervisor, as well as keeping a log on events and meetings.

Programmer, designer and scribe are minor roles. All group members are programmers, developing source code for the product. Designers are concerned with the ergonomics of the project, in areas such as aesthetics and interaction design. Scribes are the group members assigned to write the majority of the report.

Reidar will front as the product owner on the behalf of HDO, as he's the main administrator of the schedule

Group Rules

- Tuesdays and Wednesdays will be communal work periods between 12:00 and 18:00.

- Thursdays is dedicated to individual work.
- Fridays will mainly be used for meetings and discussions.
- There will be at least one meeting with our supervisor every week, as long as there are relevant topics to examine.
- There's a meeting with the employer every other week. Consistent progress will be necessary in order to have a prototype to present for each session.

1.2.2 Development Model

The contact persons at HDO expressed an eagerness in working with the development of the project as it progressed, by providing their guidance and feedback. An agile development method was adopted to accommodate for this need, as they are more suited for flexibility than sequential development methods. Using a traditional method like the waterfall model makes it difficult to react to customer needs during the later stages, when the structure has already been established, and can no longer be changed. Agile methods, on the other hand, develop each module separately, allowing them to be altered without severe ramifications. The inexperience within the group also favors agile methods, given the low likelihood of accurately predicting the amount of work that could feasibly be accomplished.

Previous exposition to the commonly used Scrum model gave a starting point, but it was lacking in some areas. Scrum uses a series of Sprints, in which developers work without external interference for a set period of time, which could possibly pose problems for the project. The limited time to work on the project combined with other interfering school courses resulting in adopting a more flexible method, eventually settling on Kanban, with some artifacts adopted from Scrum.

Kanban is a lean development method centered around using a task board to manage task flow and provide a visual representation of the work flow. It stresses incremental development and keeping the focus on the current state instead of making too many plans for the future. The task board contains tasks that need to be fulfilled to meet the project requirements, which are divided into different sections depending on which phase they are in. Common sections are “backlog”, “under development”, “testing” and “completed”, but may be customised depending on preferences. The “work-in-progress” areas of the

task board have an upper boundary on the number of tasks that may be assigned to it, preventing developers from losing focus by undertaking too many tasks simultaneously. A few Scrum artifacts were incorporated alongside Kanban to emphasize discussions during the project. Daily Meetings were used to discuss current progress, in addition to having consistent meetings with the employer. **Use of model in project**

- A task board is used to manage the tasks. Following the Kanban principles, tasks are assigned within areas to specific individuals. Keeping an overview of current progress and a method of organizing the task load improves the development flow.
- Each workday will begin with a meeting, in which the current progress is shared. These discussions are used to track progress and discuss current issues.
- There will be a demonstration for the product owner every other week, providing the people at HDO a chance to offer feedback from their perspective.

1.3 Report Introduction

The following chapters are dedicated to inform about the project and those involved with it. Here is a short introduction as to what they contain:

- Chapter **2 -Requirements** describes topics regarding the requirement specification and the scope of the project. Of particular note is the project's feature list, which is contained therein.
- Chapter **3 -Design and Analysis** primarily contains drafts of different designs and ideas that resurged in the initial phases of the development phase.
- Chapter **4 -Development** is a look into the development of the project, focusing on the implemented features and various choices made regarding these.
- Chapter **5 -Installation** is an installation manual for the project, how to deploy the source code on a server and set up the database.
- Chapter **6 -Summary** is a final discussion around the final results and the project as a whole.

There is a few products created during the project period that could not be put anywhere in the report, but helps to present the bigger picture. These are therefore put in the following appendices:

- Appendix **A -Terminology** is a comprehensive list of jargon and acronyms used throughout the report that may need further elaboration.
- Appendix **B -Meeting Summaries** contains the notes made during meetings with the product owner and as a group.
- Appendix **C -Work Log and Progress Report** is a worklog roughly outlining what each group member worked with each given week.
- Appendix **D -Progress Plan A** is the gantt diagram made during the planning phase, which describes the initial expectations to the implementation phase.
Appendix **E -Progress Plan B** is a diagram of how it ultimately ended up after the development was finished.

2 — Requirements

The **Requirements** chapter outlines the different specification to the project that were established in the project description, as well as those that cropped up under development. **Environment** contains the domain model, as well as the deployment requirements. **Operational Requirements** describes how the application adheres to service requirements and **Feature Requirements** provides information on all the features. **Software Design Limitations** is a short overview of restrictions the group made for the development.

2.1 Environment

2.1.1 Deployment

As a web service, the software necessitates a running online server that can be accessed through a browser. The server will need to run an apache web server able to execute php 5.4 or higher.

The server will need access to a MySQL database as well. This could run on an independent server to lessen the load created by only having one, although it's unlikely for the server load to reach critical numbers with such a low user base.

Details on how to deploy the solution can be found in the installation chapter.

2.1.2 Domain Model

Figure 2.1 shows the basic structure of the system. Most employees will be assigned to a specific week in a plan. Some users are granted administrator privileges. The administrators can create and change new schedules, as well as manage the active ones. A plan lasts

for a period of time, and is effectively an activated template. Templates are created in the editor by making a set of different shifts types, and using these to fill out an arbitrary number of weeks.

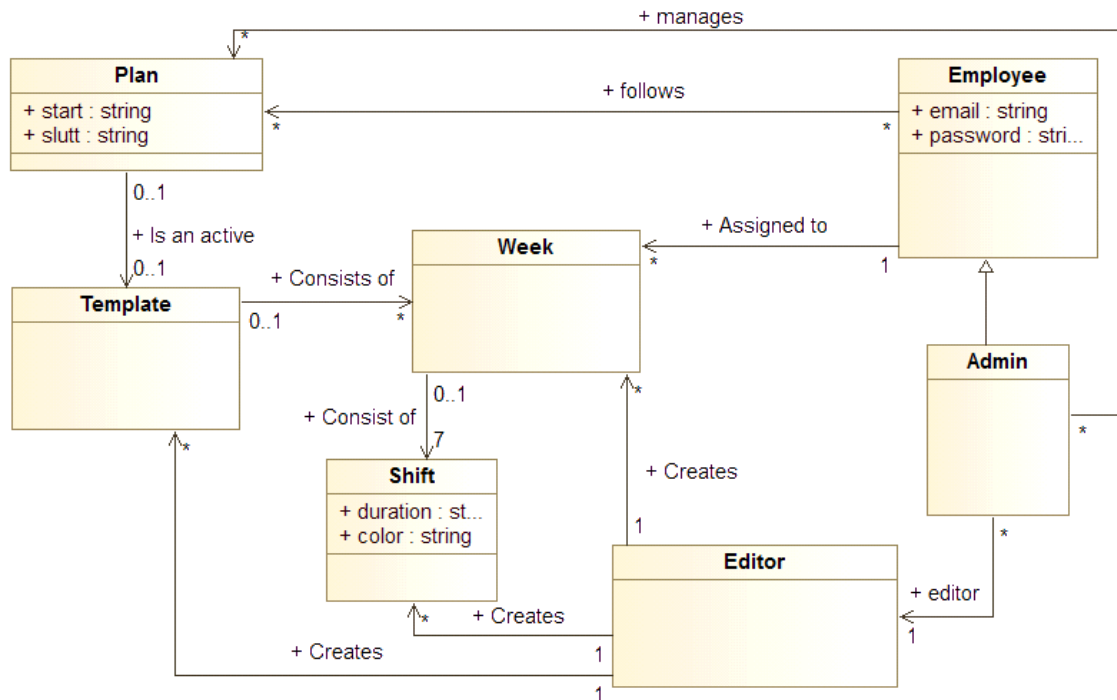


Figure 2.1: *System Model*

2.2 Operational Requirements

2.2.1 Normal Operation

Usability

The application should be as accessible for the employees following the schedule as well as the administrators of it by making it a web page possible to view using common web browsers.

Some authentication mechanism is necessary to prevent access to advanced functionality. Viewing the schedule does not require user login, making the plan easy to access from

a user perspective. The amounts of rights for a profile will vary depending on whether they are set as a regular user or an administrator.

There will be multiple different ways to organize the data to fulfill the user requests of having a more personal schedule view. This includes the traditional view that only contains information on a given user, in addition to other formats such as a daily, monthly and agenda view. Navigation options such as header buttons provides a simple method of switching between the perspectives.

Administrators are given the possibility of building new schedules in the application. This means that they need to have the ability to create and manage new schedules, and activate them for a given date.

Safety

View access to most of the website is permitted while logged out, so it's possible to view a schedule without going through the login interface. The exceptions are the personal shift exchange page and administrator features. Any management rights requires some form of login requirements, depending on context. Utilizing personal pages, such as adding notes to one's own shift or viewing the shift exchange overview needs the appropriate user to be logged in. Manipulating data without limitations, and making changes to the users and schedules requires an administrator. A timer system was considered to prevent these responsibilities from being abused by inappropriate individuals, but was discarded due to usability concerns.

Users have full control over the access password to their own account. It can be changed in the event that it should be compromised or forgotten by the user. In such a case, an email can be requested, linking to the area in which a new password may be provided.

Availability

Work in the health department means that it's necessary to have manned employees available for support at any hour of the day. It's therefore important that the schedule is accessible without fault. As a web service, it therefore needs to be accessible through the internet, and not just on a local network. It also needs a stable server, so that hardware

fault doesn't result in unfortunate consequences.

2.2.2 Problem Handling

Error report

Errors will be handled through the framework. A debugging mode can be enabled in the files for testing purposes. It's turned off by default to prevent errors messages from occurring for normal users, whom would be unable to properly understand the content of the message.

Any errors that do occur will be saved in a log file, so that it may be accessed by the server administrators. The log file is unsuitable as a warning system, but may be useful in case a problem is observed and reported.

Retrieval after error

MySQL has implemented methods for exporting and importing a database. Consistent backups can therefore ensure that the information is retained after a crisis, so that it can be restored to a previous state. The structure of the schedules will likely suffer minor consequences, as they are created very seldomly, but some data regarding shift exchanges such as notes may be lost in the time interval between the backup and the system fault.

Safety

Prepared statements will protect the application from sql injections. This puts a layer between user inputs and the database by preventing content that could have a detrimental impact on the stored data. The input will also be validated by the framework in certain areas, as an additional precaution.

The profile passwords will be encrypted, making it more difficult to obtain administrator rights by people with malicious intent.

2.3 Feature Requirements

- **Overview of Standard Schedule**

On a general level, it should be possible to access a view over the schedule, similar to HDO's current method. It's a list of the week schedule of each participant listed vertically, and is useful for getting an overview of their coming shifts.

It should contain the names of each participant mapped to a week, color codes for each day to indicate working hours, and symbols, text boxes or other types of markers, to indicate shift that deviate from the normal structure and contain additional information. Overall, this view will be best suited to give an overview of the structure, as a given schedule will look mostly the same each week, except for the rotating participants and deviations specific for that week.

Technically, it will be a combination of the view for a template, a table containing the original structure of a schedule, and a plan, the dataset that conveys the information of an active schedule.

- **View expanded information on a Shift**

Selecting a single shift should view an expanded page with dedicated information. This page should display all notes, information about shift exchanges, the assigned employee, date information and other relevant data. Some functionality should be available in accordance to the user's privileges, for management features accessible to administrators or specific individuals.

- **Profile login and information**

Users should be able to log in. The site will control privileges based on user status, separated between logged out, logged in, given user and administrator. Given user represents situations in which the identity of the user is important, such as an employee viewing his own shift.

	Normal user	Logged in user	Administrator
Standard plan	Can access	Can access	Can access
Monthly and agenda view	Can access	Can access - Defaults to personal schedule	Can access - Defaults to personal schedule
Expanded shift information	Can access	Can access	Can access
Add and edit notes on a shift	No rights	Can create notes on personal shift and edit these	Can create, edit and delete any notes on any shift
Shift changes	No access	Can request and accept shifts on own behalf	Can change any shift. An exchange must be approved by an administrator.
Manage schedules	No access	No access	Full access

Administrators will have additional access allowing them to perform a bundle of additional tasks such as:

- Create a new schedule template and edit them.
- Create a new shift template, and edit existing ones.
- Set a schedule template as an active schedule.
- Manage active schedules.
- Create notes on any given day
- Edit all notes on any given day
- Administrate user profiles

The plan is to modularize the login process to such an extent that if sufficient time is available before product delivery, it can be swapped out with the LDAP solution of the product owner, or at the very least make it feasible for the product owner to do this post-delivery.

- **Overview of personal shift schedule**

This task is mainly about outputting alternative views of the shift plans, that only

displays the schedule for a selected user. Schedules of other users will therefore be excluded from those views, as long as they're not selected.

The different views will have their own pages that can be accessed with ease from the header of any other page. The alternatives will be a monthly and agenda view. The first can show all the weeks of a month for a user, this can be used to easily learn the layout of the next couple of weeks. The latter will list shifts day by day in a vertical manner and have more room to show notes compared to other two.

- **Email notifications.**

Send notification of updates from various events from the application to the affected users. Such events include:

- A comment/note is added to a day (or an existing comment/note is edited) (email sent to affected employee and admin)
- A shift is directly exchanged (email sent to affected employees, and admin)
- A shift exchange is requested (email is broadcasted)
- A shift exchange is accepted (email to the exchange initiator and admin)

- **Mobile interface**

Product owner has requested mobile compatibility. In effect, this means that the website must be viewable and fully functional on mobile devices.

In practical terms, this will be implemented by creating and adding various CSS rules and JavaScript functionality optimized for the mobile platform. In general, guidelines put constraints on amount of information displayed simultaneously, how interaction with the website is handled, and navigation optimized for the mobile platform.

All major mobile platforms have built-in calendar applications as well, which can be used through “Export to other services” to enhance integration between the website application and mobile interfaces.

- **Shift exchanges and requests**

A logged in user should be able to select one of his own days, and from it create a request to swap it. The request will appear to everyone on the general overview, and contains a list of alternative working days the requesting user will accept for the trade.

A specific page is set up to handle interaction with shift exchanges. Users can from this panel get an overview over their own and other employees' request, and use it to overview and propose exchanges. Administrators can use this area to accept exchanges between two users. Administrators may also do direct exchanges from the expanded shift interface.

- **Export to other services (Google, .cvs/.ical formats, etc)**

Major mobile platforms have built-in cloud-based calendar solutions, which can be used to give the user additional benefits such as notifications on changes or start of day, offline viewing of schedule and the possibility to view schedules for several different categories simultaneously. This would also give access to the schedule for all platforms supported by the given calendar solution.

In practical terms, this is implemented by allowing the website application to export a schedule plan to a given format (.cvs, .ical, etc) by a consistent URL link, a so called iCal feed, which can be generated by the website, and given to the appropriate calendar solution.

The calendar solution will after importing the URL, frequently check the link for updates, and synchronize these updates to all devices and platforms in use by the user.

Since these formats are fairly standardized, compatibility can be expected with most calendar solutions, such as Apple iCal, Google calendar, outlook calendar, Windows Live calendar, etc.

- **Profile management**

A simple page containing information about the user, such as email used for login

and notifications, password, name, and other options. Most of these options can be considered trivial, and unimportant for the administrator group. By allowing the user to edit these directly, the user no longer has to rely on administrator to update this information, and the administrator saves the time otherwise used to maintain the profiles. Passwords are assigned randomly and given to the assigned email, but can be changed freely.

While profile management can be considered trivial, profile creation/removal is not, and remains solely as the administrator's responsibility. As such, regular users, logged in or not, will not have access to create/remove profiles, and will only be able to edit their own profile while logged in.

- **Create/edit/remove shift templates**

A shift template will define a blueprint for a single shift, including code, full name, color, work start, end and total duration. When creating shift schedule templates, each day in each week must select one of these shifts that it uses as a base. This information is used to display most of the information of an active schedule.

- **Create/edit/remove schedule templates**

The schedule contains an undetermined set of weeks, each of which contains seven shift templates. This defines the initial layout of a schedule, which will be rotated each week.

The template is like a blueprint, and will not contain any actual dates, or any mapping to employees. As a template, it's purpose is to define the layout that all the dynamic data, such as exchanges, employees and notes, interact with. It will consist of an unique name, and a list of weeks

- **Manage user profiles**

Profiles are used throughout the site, to assign ownership of work schedules, to allow login, and to allow logged in users to perform additional tasks. Each profile can also be marked with administration rights, granting access to all administrator tasks.

Profiles will be visible in a list for the administrator, and can be selected to edit or remove the individual profile. Controls on this same page will allow the administrator to create new profiles. When starting new shift schedule plans, the administrator will also have the ability to assign employees to each week using a dropdown list containing all active profiles.

- **Manage schedules**

Administrators should be able to select a template and turn it into an active schedule. The area will provide controls to select a start date, as well as an optional end date. Each week can also be mapped to an employee, or several as long as the time doesn't overlap. Management of users can be controlled at a later date, as well as week notes, also controlled from this environment.

The interface is accessed by clicking on the active button, either on a plan from the admin panel, or from the template editor. Activating a plan from this interface only requires a valid start date in the input field, which can be selected through a datepicker. All other fields are optional. Employees and Week Notes are added by similar methods, through a pop-up interface that appears when selecting a week. Adding an employee provides a drop down menu of all users in the system, while Week Notes are free text. Both of them have an optional field for both start and end date, which also utilizes a datepicker. Empty fields will use the plans values as default until changed.

While the database could theoretically support a template being used for multiple schedules, it also makes the admin panel more difficult to use and understand. Because of this, it has been limited such that a copy of the template is required for each schedule, which are easy to create.

- **Export schedule template**

The possibility to export a schedule template to external format such as .csv/.xml/excel for external processing. Primarily as a convenience for validation purposes. It can be accessed through the template editor, and provides a list of the schedule using day

codes, as well as the duration of each week and the definition of each day code.

- **Weekly note**

Certain type of notes should be able to span an entire week, as opposed to a single shift. These notes are static in position, and do not rotate with a user as the dates change. They are commonly used for specific tasks related that relate to all the shifts on the given week.

These notes should be managed through the administrator's plan manager interface, in which he can specify the content of the notes, as well as the duration. Multiple notes cannot occupy a week on the same date, but can be added at a point in which the previous one has expired.

- **Manage participants**

Participants may leave and consequently join a plan that is currently active, resulting in a change between the participants on different dates. Weeks may also be left vacant in situations where no new users have replaced the previous one.

The current implementation solves this by overwriting the user, but doing so would destroy the integrity of the plan, as earlier dates would be retroactively changed. The implementation to solve this is equal to that of the Weekly Notes, in which a start date and end date may be specified. Leaving them at default means they are participating for the entire span of the plan, or until the administrator changes the leaving date.

2.4 Software Design Limitations

2.4.1 Standards and Languages

The application will primarily be written in PHP, utilizing javascript, HTML and CSS. Source code should adhere to the java standards for structuring program code, and be written in English. Non self-explanatory code should always be commented over the code

in question. Information for PHP-doc will always be included for functions and classes.

The database will be created and managed through MySQL.

2.4.2 Software packages and tools

The product owner has provided the opportunity to connect the applications with the LDAP solution for profile management currently in use by HDO. The possibility will be explored if time allows for it, or if it can be included with ease post-production.

The server the application is to be installed on needs to run Apache and a version of PHP 5.4 or higher.

3 — Design and Analysis

The **Design and Analysis** chapter is a detailed overview of many preliminary design choices, presenting many of the decisions that helped shape the project from the start. **Similar Solution** looks at applications that fulfill a shared purpose with our project. **Views** discusses the approach to the original drafts of the interface, and the inspiration behind it. **Web Service** details why a web site was preferred over other software implementation methods. **Original database** shows the original look of the database. **Tools** is a short introduction to the tools that were used throughout the project, and **Risk Analysis** presents the evaluated risk of the project as assumed in the beginning.

3.1 Similar Solutions

Numerous programs that fulfill the same purpose as this product already exist on the market. ShiftPlanning is an example of one of these solutions, and will be used as a comparison. It's a well developed application of significant size and prestige. Like the product of this project, ShiftPlanning displays the shifts in a calendar format, and have different possible views to present the information in. It's additionally integrated with third party payroll managers, sending the information of work hours per employee. They provide solutions for creating custom rules for vacations and resignations, and makes it possible to set employees as unavailable for periods of time.

A problem with buying an existing solution or find a free alternative will make it problematic to have the solution be more customized for the required needs of the organization. In the case of HDO, there is no need for the feature found in ShiftPlanning allowing an administrator to move shifts independently, as the schedules themselves are usually very stable. No solution was found that took the week rotation into consideration, which invalidates

them for this specific purpose.

3.2 Views

The project description requested a method for the employees to organize the plan in a more personal format. Google's calendar service was an important source of inspiration when attempting to fulfill this. Early drafts of the software initially had the full spectrum of week, month and agenda views, in addition to the standard implementation of the plan view. Using the same format as commonly utilized calendar applications makes the interface more recognizable, and consequently more intuitive for already experienced users. Using the similarities between the systems also makes it easier to develop potential export mechanism.

Supporting three different views of the same information was regarded as a relatively low priority, and the week format was eventually abandoned in favor of other, more valuable features.

The current excel implementation of the shift schedule seen in figure 3.1 was the largest

	A	B	C	K	L	M	N	O	P	Q	R	S
1	7. mai 2014		Velg uke:	Velg år:	TURNUSPLANLEGGER FOR DRIFT OG BRUKERSTØTTE							
2	uke: 19		05	2014								
3				Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag	Sum:	Avk.
4				27/1	28/1	29/1	30/1	31/1	1/2	2/2		
5				FRI	D1	D4	D1	D1	FRI	FRI	32,63	32,63
6												
7				D1	D1	D4	D1	D1	FRI	FRI	40,63	36,63
8												
9				D1	D1	D4	D1	D1	FRI	FRI	40,63	37,97
10			VO operatør									
11				D1	D1	D4	D1	D1	FRI	FRI	40,63	38,63
12												
13				D1	D1	D4	D1	D1	FRI	FRI	40,63	38,03
14												

Figure 3.1: *Excel Implementation*

source of inspiration on how to implement the default view, which was an important factor for shaping the layout of the rest of the system. Much of it was designed to be familiar, in part to not alienate the employees accustomed to the previous version, but also because

the established approach seemed quite sensible.

There was a nearly identical resemblance in the early mockups as seen in 3.2, one of the earliest design drafts shown to the product owner. It displays one of the early ideas that persisted throughout the project. Having an inner and outer section of each shift cell made it possible to use the central area to display the current shift type and the outer section to indicate the previous shift type, in case an exchange has occurred. The comment field also displays the intent of having several different notes on a single shift, although it does a poor job of separating them, and has faced a significant overhaul since that point. It became impossible to have enough space to display more than one note after author and post date was taken into account. Multiple notes are therefore only shown in the expanded view as a result. Several other parts of the layout has also evolved since that point. Working with a dynamic screen format and being able to isolate the admin functionality gave some leeway to work with in screen size, resulting in various different interface changes.

Uke 42	Monday 3/2	Tuesday 4/2	Wednesday 5/2	Thursday 6/2	Friday 7/2	Saturday 8/2	Sunday 9/2	
Uno User	N1 Free	N2	D2	D2	D2 Vomstrod	D2 Vomstrod	D2 Vomstrod	
Another Associate	D1	X1	D2	D2 System set up Don't forget 2 lunchbox Make sure to check for... 3 more...	D2 Free	X2	D2	
Elaborate Employee	D1 Work at Holdmoen	N2 Work at Holdmoen	D2	N4	N5 Free	N5 Free	D2 Free	

Figure 3.2: *Early Design Mockup*

Figure 3.3 shows a very early design of a pop-up containing expanded shift information. Shifts were designed to be able to contain several different notes on a day, and some functionality need to be put in place to make these more easily accessible than what the standard plan view offered. A pop-up solution was concluded as the best way to solve this problem. The pop-up would also be the container for all relevant information on a day, providing extensive information instead of the compressed cell in the standard view. Having it as a pop-up also prevents the flow from being disrupted too heavily, although a redirect is implemented for mobile devices to take the limited screen size into consideration.

The mockup shows a window titled "Uno User". At the top left, it displays "D1 - Day 1" and "08:00 - 17:00". At the top right, it displays "Originally N2" and "6.2.2014". Below this is a form area with a dotted border containing a "Submit" button. The text inside the form area lists several tasks: "Set up system", "Don't forget a lunchbox", "Remember to check for errors, so what happened last week won't happen again.", "Check the commandrod", "Cut the old wires", and "Up-Up-Down-Down-Left-Right-Left-Right B, A, Start."

Figure 3.3: *Early Design Mockup*

3.3 Web Service

One of the primary requirements in the project description was the accessibility of the software. The employees should be able to view the plan with ease, with either a computer or a mobile device. Portability problems surface when developing a mobile application because of the different manufacturers. The application would also have to be synchronized through the internet regardless of implementation. A web service would be the most effective method of fulfilling this accessibility aspect. Although the internet requirement is non-negligible, it can be accommodated for rather easily, and the application already has a real-time component that it would otherwise be impossible to cover.

Presenting the information on a website comes with a few distinct advantages. The most important one for to the group was the ease of portability. Mobile devices are designed to display web content, making it several times easier to develop the application cross-

platform. While the mobile display carries some limitations, it would still be easier to make it function at a sufficient level, and custom libraries can help shore up some of the weaknesses. Web services also removes the burden of having to download new software, as computers are expected to have some form of web browser installed. While the employees are likely to be a fairly static factor, it does enable them to view it at both their home computer, working computer, mobile phone and tablets without having to prepare for it beforehand.

The choice of programming language was largely unanimous. While the discussion of using options such as the .NET framework, Ruby or Java Servlets did come up, it ultimately ended up as a PHP implementation, largely due to prior experience among the group members.

3.4 Progress Plan

Appendix D is the original progress plan, showing the estimated sequence for implementation of functionality and how much time each task would need. Additional information is provided on specific entries to clarify how and why they were made.

- Standard view
Regarded as the core of the project, it was regarded as the highest priority and the first thing to implement after the framework had reached a stable point.
- Alternative views
It was assumed that the implementation of additional views after the standard view was implemented would in many ways just be to reuse or make variants of that view, and was as such considered to be rather simple.
- Login
The group had a login system from an earlier student project that worked well and tailoring it to this project was estimated to be a simple task.
- Sequence of user management
The option to manage users directly through the MySQL interface pushed this down on the priority list, as it was unnecessary for the implementation of other features.

3.5 Original Database

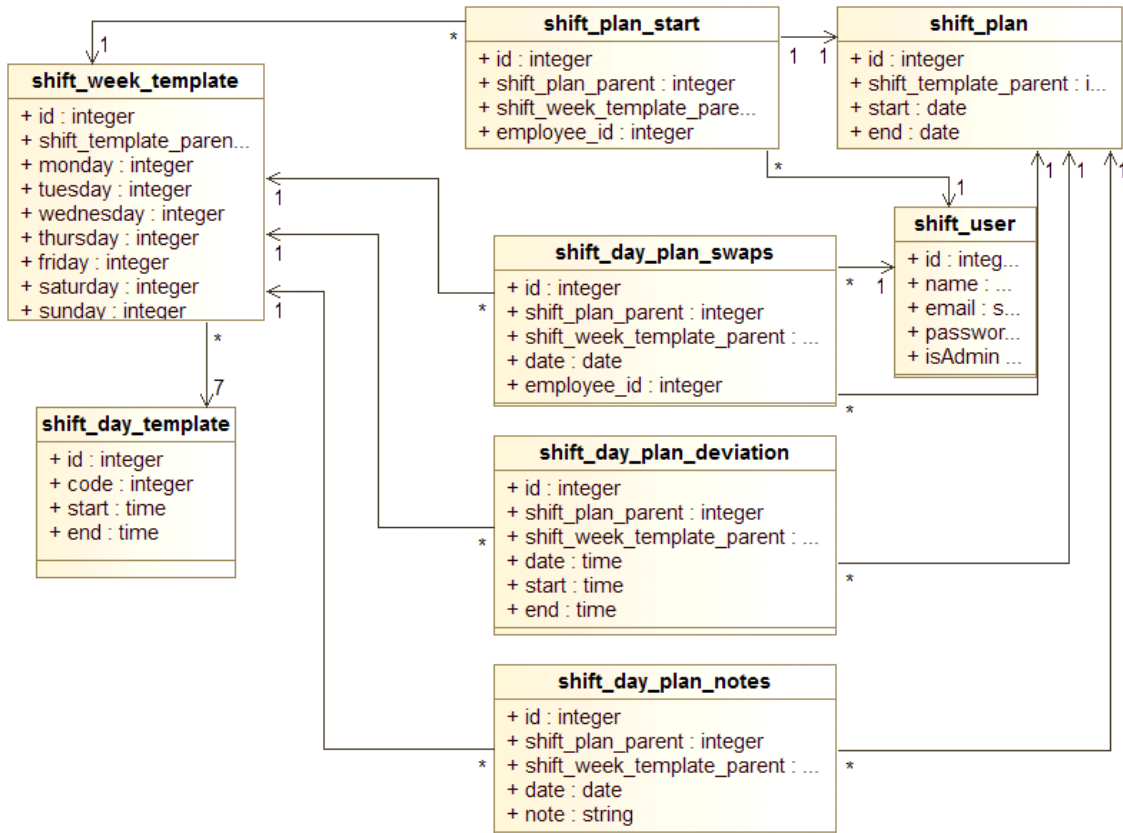


Figure 3.4: *Early database draft*

Figure 3.4 shows the original database design created late January, shortly before full development started. The day-deviations was an idea of a table that would contain information about specific days where the shift of an employee would deviate from the normal, cases where a shift would be delayed, pushed, extended or otherwise have a duration outside the norm. An entry in this table would serve as a replacement for that shift. This was dropped after the contact person deemed it unnecessary. The three different tables that mark deviations, named "shift_day_plan_*" used the plan reference to determine what plan they belonged to, the week template reference find the correct week sequence and the date field as means to decide the correct day of the week it was linked to.

The `shift_plan_start` table contained the data for the initial week of an active schedule, mapping each week of the schedule to a specific participant.

3.6 Tools

Inclinations towards using Eclipse surfaced early while exploring the options for developer environments, and was a likely choice given previous experience in using it for other projects. However, the commercial software PHPStorm ultimately became the choice, despite Eclipse's familiarity and open business model. The trial version provided a superior work environment, leading to a purchase of the IDE through the academic license by each member. While primarily done out of preference, it also provided the benefit of working with a shared interface throughout the team.

The shared repository is hosted at BitBucket, which synchronized all work through Mercurial. Kanban tasks are hosted at the website Trello, a task board approach to collaborative organizing. Several files are hosted on a shared Dropbox folder, while writing was done collaboratively through Google Documents. The final report was compiled with LaTeX, however.

The website is written in PHP, utilizing HTML, CSS, JavaScript, JQuery and MySQL. Considerations were made for other client-side languages with the .NET framework as the frontrunner, but they were ultimately rejected in case the inexperience would prove detrimental to the final product.

3.7 Risk Analysis

	Description	Likelihood	Impact
1	It may prove difficult to create the mobile interface separately.	Likely	Moderate
2	Misunderstandings with regards to the employer's requirements can cause unnecessary amounts of time to be spent correcting previous implementations.	Likely	Moderate
3	Conflict in the group	Low likelihood	Moderate
4	Unpredicted delays that would break the overall deadline	Near certainty	Moderate
5	Loss of resources	Low likelihood	Significant
6	Illness amidst group members	Low likelihood	Minor

Solutions

1. By making research beforehand, it's possible to gain a better perspective on the situation before making a full commitment. If the task appears to be too time consuming, an attempt to implement it through export methods may be considered as a substitute.
2. By using an agile development method with semi-weekly prototypes, the employer will get the ability to continuously review the product.
3. Internal disagreement will result in a meeting to solving the issue. A discussion with the supervisor will follow if no consensus is reached on the subject, trying to reach a solution based on resulting advice. It's the project leaders responsibility to organize the initial meeting.
4. Rework the estimates of time usage. If the deadline is breached too much, some of the the less important features will be dropped as earlier, determined by the requirement specification.

5. The files will be hosted in a repository, which will ensure that they stay accessible and synchronized.
6. By developing the software through small independent modules, sickness will only reduce the overall production the team, rather than create stalls as some parts remain incomplete over longer durations of time.

4 — Development

The **Development** section is dedicated to the creation of the software, implemented features and decisions made along the way. In short, it's how the software was developed. **Implementation** details some of the more important structural implementations in the software. **Ergonomics and Aesthetics** is, comparatively, about the look and feel of the software, and how it behaves from a user standpoint. **Testing and Quality Assurance** describes the various methods the software was testing during the course of the project. Finally, there's a list of substantial issues that were detected, and how they were resolved, in **Encountered Problems**.

4.1 Implementation

4.1.1 Framework

Inspiration

The original idea of the framework was intended to have an implementation much like the common GUI interface library for Java by the name of Swing. Swing enables the developer to create different types of interface elements and fill them with information. These can be added to other Swing elements dynamically under runtime, and as such has much in common with the DOM structure found on web pages. Commonly used elements could be implemented in a library, reducing the complexity of creating web pages. Examples of possible usage areas include warning messages, submit forms and pop-ups, with parameters to enable different features and appearances.

Model/View/Controller

The framework of the project use a variant of the Model/View/Controller design pattern to handle use and page construction. Systems following this pattern mainly share the fact that the responsibilities for the interactions are shared between three segments: model, view and controller. However, there is seemingly no uniform agreement on how these responsibilities should be properly assigned.

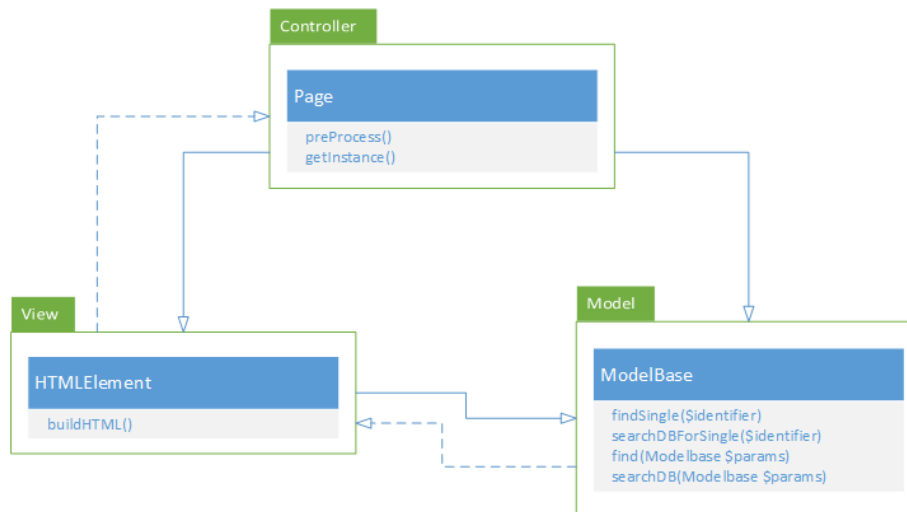


Figure 4.1: *Model/View/Controller for this project.*

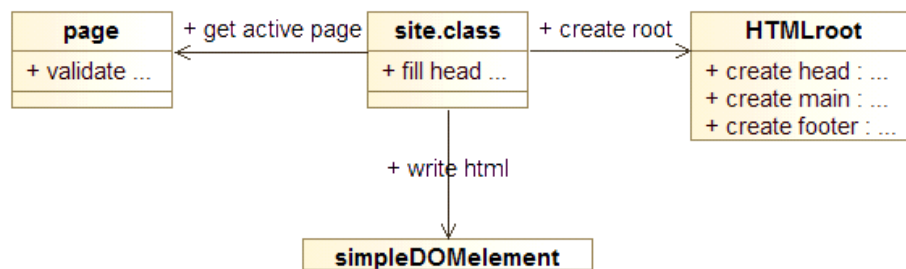
In this project the models are the objects that inherit from ModelBase. They usually contain a row from a corresponding table in the database. The exception to this is the ModelWeek class, which consist of selected data from two different tables. Models are instantiated during runtime when the application need access to the data found in the database. Instantiated objects are only used to get the data, not edit it. That is done through the “Page” files.

The “Page” files are the controllers. These files instantiate models and process these objects. Those objects, or the data processed from them are usually sent to the view, called “HTMLElements”, which builds the DOM-document with it. The pages handles manipulation of the database as well, such as editing, saving and removing data

The view segment was initially split into two parts so the pages could call the view-classes and have them create HTML, XML or JSON, based on what was needed. Later, it was realized that the project only ever used HTML output, and the functionality and wrappers for the other other output types were dropped from the framework and project.

Dom-document creation

The web pages viewed by the users of the system is generated in php by creating a DOM-Document tree. Initially, an external library, QueryPath, was used for this purpose. However, serious performance concerns, as well as compatibility issues with the rest of the system, eventually led to an effort to replace QueryPath with our system. SimpleDOMElement was created for the task, which had the goals of eliminating the performance issues QueryPath had, while also being simpler in it's usage.



The elements of the web-page that will always stay the same, mainly the header and footer, is created by `site.class.php` through a call to the “HTMLRoot” class. The container available by the request page is defined here as well. The content of that container is filled by having the controller get the data and send it to the various HTMLElements that handle the representation. The controller is selected through a call to “`Page::findNavPage()`”, which selects a page based on the requirements of each page compared to the user input. The resulting page is built from the bottom up, constructing a tree structure of SimpleDOMElements, ready for output.

Following is an example of how SimpleDOMElement creates a day cell as can be seen in Standard view, amongst others (using the global shortcut function of spe()).

```

// Frame
$cell = spe("td")->addClass("day");
$cell->attr("style", "background-color: _{$this->mDay->
    getOriginalDayTemplate()->getColor()}");
$cell->attr("data-pop-up", $this->mDay->getURI());
$cell->attr("title", "Dag_{ $this->mDay->getDate()->format("Y-m-
    d")}");

// Inner frame
$cell->append( $innerCell = spe("div")->attr("style", "
    background-color: _{$this->mDay->getDayTemplate()->getColor()
    }") );
$innerCell->append( $title = spe("div")->addClass("title") );

// Title (code, time)
$title->append( spe("span")->addClass("code")->append( $this->
    mDay->getDayTemplate()->getCode() ) );
$title->append(
    spe("span")->addClass("time")->append(
        "{ $this->mDay->getDayTemplate()->getStartTime()->format("H:
        i")} _-_{ $this->mDay->getDayTemplate()->getEndTime()->
        format("H:i")}"
    )
);

```

Any SimpleDOMElement object, “spe”, can be extended with an unlimited amount of attributes. These need each a name and a value. The addClass() method was added specifically because css classes were very frequently used,, but also as a remaining artifact from the time that Query Path was used.

Children are appended to a parent using “*parent* -> *append*(child)”. The append method was made flexible in that one can append not just other SimpleDOMElements, but entire

Blocks, or simple strings as well.

pop-ups

The web-page relies heavily on pop-ups. It uses its own system to create them by creating a jQueryUI dialog that floats on top of the page, which it subsequently fills using ajax, with the requested html content from a page. Serverside, pop-ups are handled no differently then a regular page. Instead, the javascript pop-up system will extract the main contents of the retrieved page, and fill the pop-up/dialog with the contents. As a side effect, each pop-up page can be served as a full stand-alone page instead of a pop-up, on demand.

A requirement going into the project was that it should be viable for use on mobile platforms. The implementation of pop-ups provided some problems in terms of screen space, and were often unwieldy in practice. Handheld devices therefore use a redirect instead of opening a pop-up, which adds some unwanted navigation time but brings considerable benefits to the interface.

Date

The various views always assume that participants and schedules starts at a Monday. Rotation shift schedules are entirely centered around weeks, and the provided information specified that employees never start participating in the middle of a week. Specified dates that refer to a day other than Monday will still redirect to the Monday of that week. A global function was designed for this purpose. PHP operates under the assumption that weeks start with Sunday, which had to be taken into consideration for the function, and was also one of the reason a global method to handle it was made.

```
function getMonday(DateTime $date) {
    $difference = 1 - $date->format("N"); if ($difference >
        0) { $difference -= 7; }
    $date->modify($difference . "_days");
    return $date;
}
```


Shift Templates

A schedule need to be approved before it can be activated. Because of this, it should be impossible to change the structure after that point. However, some cases may occur where small changes are necessary between different schedules, or a name has to be reused for something different.

It was necessary to make it possible to change shifts between templates but not create ramifications for earlier shifts, to ensure that the integrity of the history is not lost. Each schedule template therefore uses it's own set of shift templates, independent from the rest. New templates copy the entries from the previous one to reduce unnecessary work, as they are likely to stay the same for most of the time. Any deleted template will also remove all associated shift templates.

Shift changes

Making an exchange between two shifts on a date that has already occurred is meaningless, as the shift has already transpired. It's therefore impossible to access the exchange interface for prior days. The overview over exchange requests and responses will additionally hide those for previous dates, removing unnecessary clutter and distractive elements.

Shift requests also provides users with the possibility of specifying the parameters for what type of shifts may be provided in exchange, chosen from the list of templates connected to the schedule. This limits the number of employees that can respond down to those who have a shift of the designated type on that day. The final conclusion on whether an exchange request is acceptable or not is still determined by the administrator.

Date checks and calculations are all done through PHP, and therefore use the server time instead of the client's time clock.

4.1.2 High-level System Description

The root folder contains 4 subfolders and the index file. The javascript and css folders are used for javascript and css files. Each page includes these files on their own, often having one dedicated javascript and css file for each different view. The site.css file is a notable exception to this, being included in every web page. The image folders are used to store

pictures used for backgrounds and icons. The final system folder is the most important one, containing all the php source code, excluding the aforementioned index file. It contains 2 php files; `globalFunctions.php`, which as the name suggest, is a list of global functions, and `init.php`, which is launched through `index.php` and is the beginning of the system stack. The following 6 subfolders can be found in the system folder, and determine the majority of the framework:

- **Classes:** Contains the superclasses most of the other classes inherit from, namely `ModelBase`, `HTMLElement` and `page.class.php`. This folder also contains `SimpleDOMElement.class.php` used for DOMtree generation. The `arghandler.class.php` used for handling arguments sent through forms and URI, retrieved through GET and POST values, can be found in this folder as well.
- **Config:** This folder only contains `config.php`, and is used for accessing the database. It consists of the name of the database, as well as the proper name and password used to access it. This specified database profile should have read and write access to the database, as it will otherwise prevent the application from operating properly. It's also possible to set the debugging options in the config file, which determines the level of information that are provided by the system when errors occur.
- **HTMLElements:** Contains the views and elements used to design the interface when creating a page. Most classes in this folder extends `HTMLElement.class.php`. The constructor of these files will initializing the member variables, in addition to specifying the required css or javascript through the following two functions:

```
Site::requireStylesheet(*css filename*);  
Site::requireStylesheet(*javascript filename*);
```

File extensions are not used for these functions.

- **Models:** Consists of the classes used to access and manipulate information stored in the database. The constructor can contain several parameters that are used to specify the type of objects to create, such as all shifts on a specific week, or within a given time frame.

The majority of the objects have a “searchDB” and “searchDBForSingle” method, which are utilized by the inherited “find” and “findSingle” functions. They are used

to find all objects that match a specific parameter, and the object with a specific ID, respectively. The two find functions will access the database if the matching objects have yet to be loaded into memory.

- Pages: Contains the classes that the system uses to create pages or manipulate the database.
- Requirements: Originally contained files which checked and specified the privileges of a user, returning true or false on whether a user was logged in, a user specified by the system, or an administrator, depending on the type of request. Checks on user privileges determined by user id was moved outside this system, and is used more preemptively in the system, due to difficulties in determining the proper user during page load. The file is therefore only used to determine if a user has admin privileges or not.

4.1.3 Database

Various changes has occurred to the database since the original draft was implemented, as can be seen when comparing 4.2 and 3.4. Some of it has been simple naming conventions for clarification purposes. More drastic changes, such as adding additional fields or tables, were done to accommodate to changes in the framework and unexpected features.

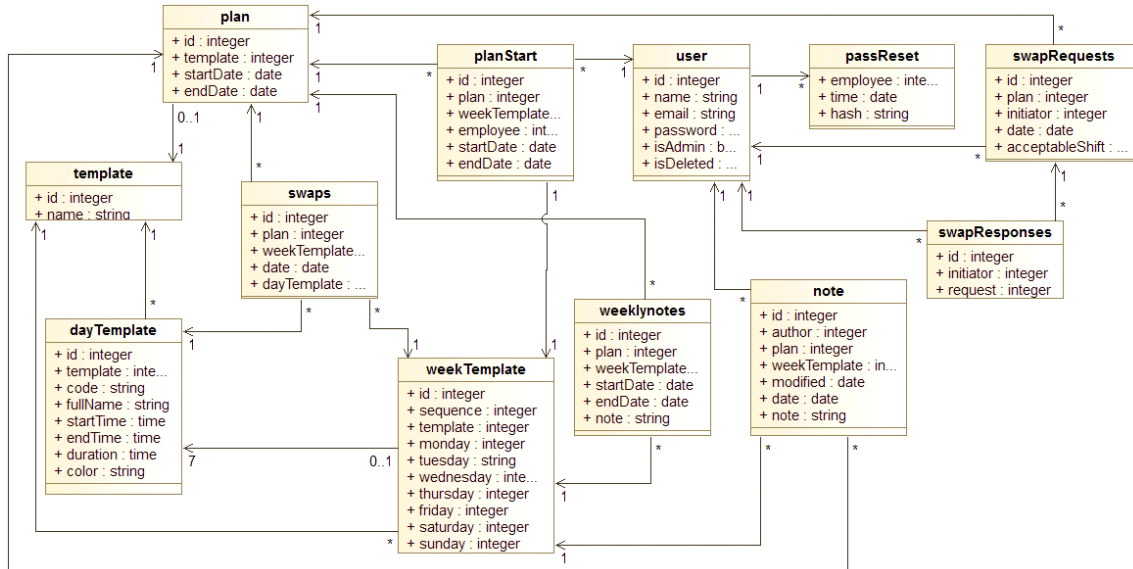


Figure 4.2: *Final database design*

The database went through a significant change in February, to help clear up some confusion and ambiguity. The large amount of prefixes were often redundant, and had a substantial impact on readability. Some important changes were removing the “day” identifier and only use “shifts” to denote them instead instead, as well as removing the parent suffix in many table fields. The “shift” and “plan” prefix was removed in most areas where it was unnecessary, such that tables like “shift_week.template” are now simply “weekTemplate” instead.

The “template” table was added for two reasons. First, it made it easier to find a list of templates, instead of assembling them through other tables, namely the template links found in “plan” and “weekTemplate”. Secondly, and more importantly, it enabled the

possibility of naming them, which is important both to organize them and as a reference amongst employees.

The "swapRequest", "swapResponses" and "passReset" tables are necessary for certain functionality and were added accordingly. The first two are used to handle employee requests to exchange a shift, and the responses to these. The third one was added for the "forgotten password" feature, where each row stores a password change request that must be completed within a time period of 6 hours before becoming invalid.

Week notes and day notes originally used the same table, as they shared a lot of the same attributes, with a single attribute used to tell the two types apart. They were divided into the two tables "weeklynote" and "note" fairly late into the project, as they became noticeably different, resulting in several of the fields becoming superfluous. Notes had no need to know their end date, and week notes didn't need an author, as a couple examples.

The initial implementation of "planStart" didn't have a field to determine when a user joined or left the plan, working under the incorrect assumption that employees wouldn't leave a plan after being assigned. The primary purpose was to determine the rotation in a schedule. However, after a start and end attribute was added, it could also preserve the history of participants.

A "duration" field was added to the "dayTemplate" table. The first incarnation determined the duration by calculating it from the start and end time attributes. Adding a duration field removed the need to calculate using DateTime objects in php, which is rather cumbersome. However, the primary reason for this change was because certain type of shifts have a duration that doesn't correspond with the time interval, sometimes being shorter than normal. This distinction is important when calculating the duration of an entire week.

As seen in the figure, most tables owns a reference to "weekTemplate". Those references existed in the early database as well, but not as prevalent as it eventually became. It shows how incorporated the system is around the idea of depending on weeks and their rotation, as mentioned previously .

4.1.4 External Libraries

Software development is a well explored area, and solutions to common situations have been devised by many intellectual people. Several libraries have been implemented in the project to save valuable time, with features that have been developed by well experienced teams.

Name TimePicker
Type Javascript
Licences MIT and GPL
Usage Timepicker is a jQuery based featured used to list time entries. This list is navigable and bound to an input element containing the last selected entry value. It's used in the shift template editor to specify the start and end point of the shift, as a more convenient method than traditional input fields. Some modifications were done to the source to enable auto-complete.

Name Spectrum
Type Javascript
Licences MIT
Usage Spectrum is a library designed for color selection. It provides a color selector that provides a user interface for specifying colors and their saturation level, and is used as the default option for color type inputs in Google Chrome. It's used in the shift template editor to specify the color of the shift.

Name detectmobile

Type javascript

Usage Discerning if the browser used by a user to connect to the webpage is a browser for a mobile phone or not, is error prone at best. This library have seen more extensive testing than anything done by us would have been able to see during the project period. Used for checking if the browser is a mobile browser or not when opening a pop-up.

Name jQuery.Datepicker

Type javascript

Usage Datepicker is an interactive tool used to fill out date values for input elements. It provides an interface to select date values from, instead of making the user type it in manually, which can result in possible formatting errors. The datepicker is used in several places, such as specifying what week to view, or determining the date interval of a week note.

Name jQuery.Drag & Drop

Type javascript

Usage The jQuery library for drag and drop is utilized in the Template Editor. Each week row can be dragged up or down to change the position of the table layout, which determines the starting structure of the plan.

Name Contextmenu

Type javascript

Licences MIT and GPLv3

Usage The context menu is used in the template editor for accessing the options used to copy, cut, delete and paste rows. It's effectively a right click menu for web interfaces.

4.2 Ergonomics and Aesthetics

Plan Views

Three different views were created to enable users to access the information in different formats according to preferences. The primary view, used as the front page of the site, shows a single week for all participants. It defaults to the current week of the month. Users can gain information regarding the current week with ease because of the general nature of the standard view. If a user wants to view a plan dedicated to his own schedule, then both the monthly and agenda view fulfill this purpose. The monthly view is constructed to show the time schedule, while the agenda is more suitable for displaying notes. As a default option, the monthly and agenda view navigates to the user's personal schedule if he or she is currently logged in, to save unnecessary time selecting the correct user.

Administrator Panel

The administrator panel was designed to express a large amount of stored information while providing easy methods to access areas to manipulate it. The area is divided into two sections. The primary area contains plans and templates that the administrator has made. They are created as a simplified representation of a schedule template, and can be identified through both name and structure. Useful features, such as editing or copying an object, is presented when the user interacts with the given item. If it's a template, then it's also possible to delete it or make an active plan out of it through this menu. Unnecessary clutter on the screen is reduced by ensuring that these options are only available when someone hovers over an object.

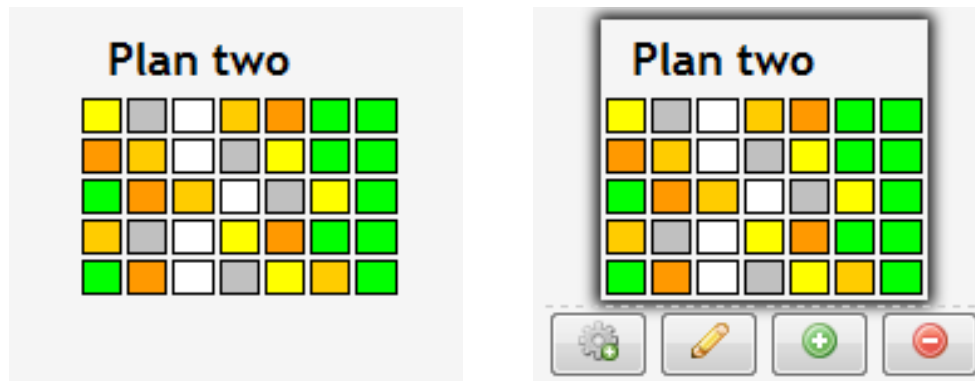


Figure 4.3: *Template Picker*

The other section of the panel administrates users. It displays every user registered in the database, and provides options to make changes to the existing users or add new ones to the system. The header field can be used as a search field in case a user becomes difficult to locate.

Editor

This view is designed for administrators to create the backbone structure of shift schedules. It allows creating of shift templates and plan templates. The former is done through the interface on the right side. The editor area for shift templates are done through a pop-up accessible by clicking on an existing template or the area for creating new ones, in which you get the opportunity to change the values of an existing template or create an entirely new one.










The section on the left is dedicated to creating the plan itself. It contains a number of diagonal rows that represent the different weeks, containing the seven days and the total duration of the week. Each day contains an input field with a shift codes, and uses an autocomplete feature that helps fill out the days more efficiently. New empty rows are added incrementally as each one is filled out. Each row also has mouse support for certain features. A jQuery drag and drop implementation makes it easy to reorganize rows after they have been created, and a right click menu provides options to delete, cut, copy and paste entire rows at once.

Manager

The manager is the view dedicated to making changes to an active shift. It contains a table with a schedule. Each week can be selected to get additional information, and more importantly add users and week notes to the selected week. Users and notes are specified with a start and end date that determines the duration that they are included in the schedule. Date pickers are included in the input fields to make these values easier to input. Leaving the start and end field empty makes the system assume that it's included from the start of the schedule and to the end of the schedule. Users are also selected from a drop down to prevent types and increase efficiency, while notes are entirely in free text.

Iconography

The site has a series of tiny icons that are used across the various pages that convey context dependant actions.

-  Create a new entry of an item.
-  Delete an entry of an item.
-  Edit the entry of an item.
-  Indicates that an item contains a single note.
-  Indicates that an item contains several notes.
-  Indicates that an exchange has occurred on a shift.
-  Indicates that an exchange has been requested on a shift.
-  Denotes an information box.
-  Denotes a prohibition or warning box.

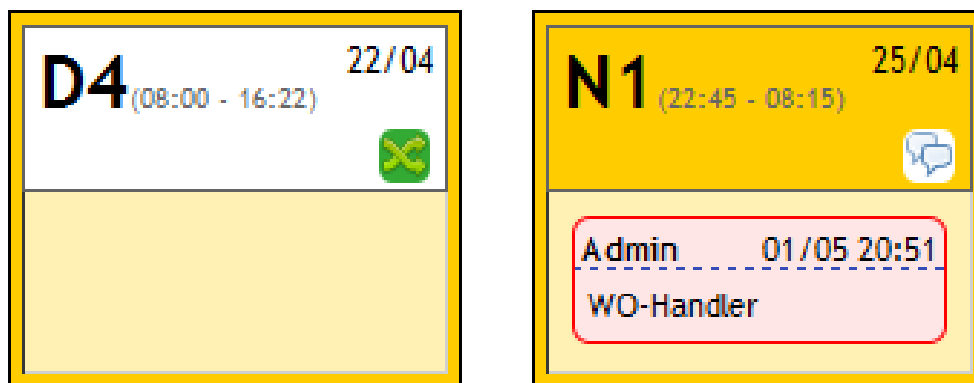


Figure 4.4: *Different incarnations of a shift*

Shift instances generally follow a consistent pattern throughout the different views. Central between all of them are the inherited color coding and day code. The outer frame are used whenever there's a possibility that there has been an exchange between two shifts, in which case the outer color represents the original value of the day. This puts the focus on the current value of the day, making it easier to read the flow of a schedule. However, the border is an important visual cue that draws attention to the fact that a shift has been exchanged, while also conveying the original status of the day. Being able to determine the original layout of a schedule where exchanges had occurred was an important concern for the consumer.

Easily discernable alert messages are utilized throughout the system to draw the user's



Figure 4.5: *Information message*

attention to important details. They come in two forms, either as blue information boxes like seen in figure 4.5 or orange warning boxes like figure 4.6, and are used both proactively and reactively. Reactive alert messages usually inform the user on one of his actions, such as confirming that he successfully logged in. Proactive messages informs of specific circumstances, like preemptively notifying the user that he lacks the privileges to add notes to a selected shift.

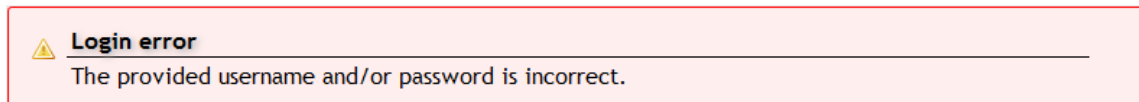


Figure 4.6: *Warning message*

Navigation

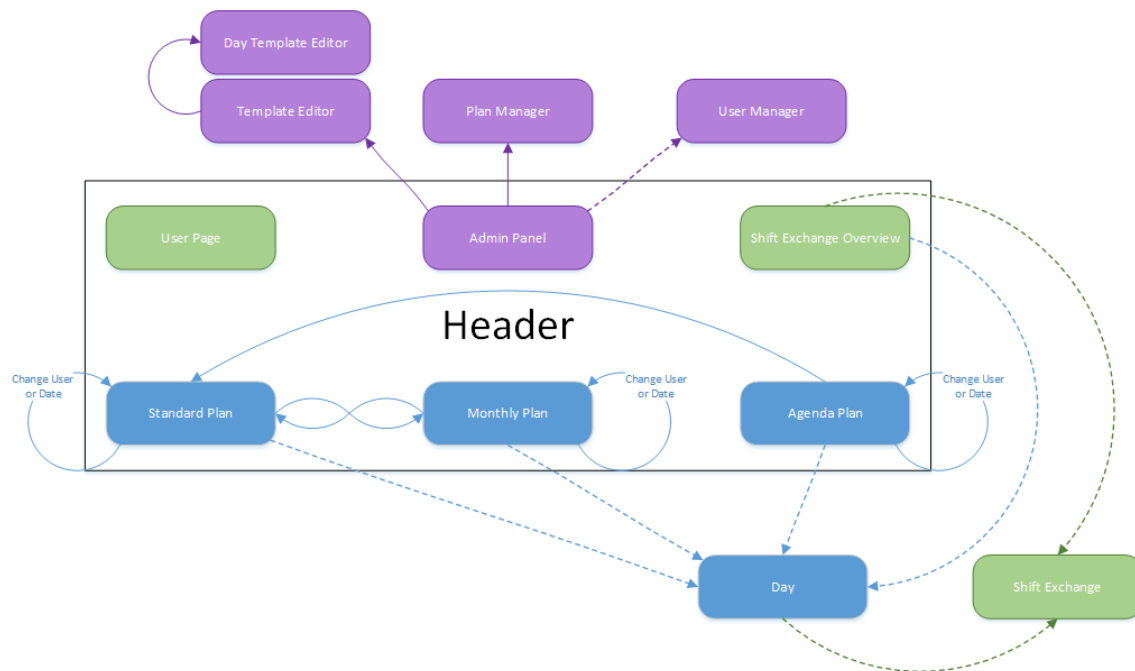
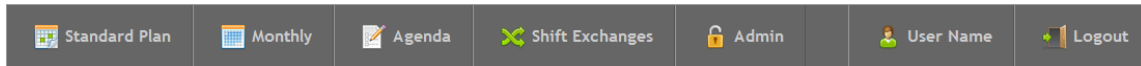


Figure 4.7: *Navigation chart*

Figure 4.7 shows the navigation map between pages. Solid arrows denotes hyperlinks, while dotted arrows indicates areas where pop-ups are utilized. Colors are used to show areas restricted through login, with blue as the default. Pages in green indicate that the content is only accessible to users currently logged in, while purple is used for areas that requires administrator privileges. The Header container indicates all the pages that can be accessed through the header tabs, available from any page.

The number of necessary steps to reach a page are related to how commonly a user will access it as well as how much information is necessary to display the relevant content. It rarely requires more than three steps to reach a single page when taking the optimal route. Most of the information that is most commonly relevant to the user all are found in the header tabs, a single click away.

Figure 4.8: *Header*

4.3 Testing and Quality Assurance

The majority of the testing process was done through co-operative work within the group. Unintended functionalities were brought up whenever they would occur, so that they could be dealt with as soon as possible. The current state of the software would always be accessible to the group members by virtue of having consistent synchronization of the repository. Usability and interface choices were likewise discussed by the team during development of the feature utilizing the object in question, before being presented to HDO.

The continuous meetings between the product owner and the development team, scheduled to Friday every other week, proved vital for the progress of development. Having to present the current state of the software at repeated intervals required the product to hold a certain standard. Alternate weekly meetings also ensured that HDO could provide their opinions on the current iteration, preventing the project from drifting away from their vision of the software. The responsibilities that comes with working with an employer helps to keep the quality both consistent and in line with the desired outcome.

A web server was deployed during the later stages of the project, enabling HDO to try out the current builds and provide feedback accordingly. This utility was useful to let HDO test how the application worked in a real scenario. Giving the product owner a deeper insight into the flow of the program provided some important information that could otherwise be received too late in the process to take into account. Hosting it on the GUC server provided to each student was under consideration. However, it was ultimately deployed on a personal server, as the GUC servers run PHP 5.2, which disables a lot of important functionality.

A profiling was initiated in the middle of March because of issues with the loading time of the pages. Testing data suggested that it regularly used around 4 seconds to fully load a single week on the standard schedule. A combination of the Google Chrome developer tools and the Xdebug PHP extension was used to get a detailed analysis on the response time.

Google Chrome was used to check the loading time of javascript, while Xdebug inspected the delay from PHP functions. The solutions provided in the next chapter provided a substantial decrease in the site's response time.

4.4 Encountered Problems

Name Surpassed Nesting Level

Problem The website frequently crashed as a result of having too many recursive function. The nesting level of the program stack breached the PHP limitations.

Solution Several of the major functions that used recursion were changed to use a stack based method instead. While functionally identical, it ensures that the nesting level doesn't reach a critical level by stacking the functions after the "Last in, First out" principle.

Name Object References

Problem All of the Model classes initially had direct references to relevant child objects. It made them easy to access once the parent object was attained. However, since the system stores serialized objects to reduce unnecessary database queries, it made their serialized form too complex. The creation of one new Model object almost always lead to the creation of a multitude other Models, most of them doing a search in the database. They increased the load time of each page with an unacceptable amount. This problem was discovered during the performance testing in March.

Solution Child parents are now stored as identifiers instead, to reduce the complexity of the objects. The system now uses find and find-Single together with the ID's to accommodate for the structural changes. This means that the page need to search for the correct data when it's needed, but it saves time by not having to create the entire model structure when a single object is instantiated.

- Name** Query Path
- Problem** Earlier incarnations of the framework used the PHP library Query Path to construct the HTML layout in the views. Performance testing in March showed that Query Path demanded an unacceptable amount of time to load. It was also deemed as inflexible to use, as the ability to affect an element were lost the moment it was appended to the tree.
- Solution** The Query Path class was substituted with a custom DOMElement class instead. This fixed the problem with loosing control over the element, as the custom class builds the tree structure when the entire thing is finalizes, as opposed updating it whenever a new element is appended.
-
- Name** Schedule Rotations
- Problem** The rotation was first implemented with the assumption that weeks would be changing the position under rotation, as opposed to the participants. This assumption was faulty.
- Solution** The weeks are from a technical standpoint still rotated instead of the employees, although in the opposite direction. Weeks are kept static from an aesthetics standpoint by using a different week template as the first week depending on the date. Early attempts at fixing this caused the employee list to be mirrored on the first participant entry, an issue that was eventually resolved.
-
- Name** Change of participants in active plan
- Problem** Multiple planStart entries that all references the same week in a plan prevented the schedules from loading, even if they were assigned to different time intervals, because the start and end dates weren't taken into consideration.
- Solution** This was fixed by using a new Model, ModelParticipant, to determine the participants of a week based on the date they join and leave the schedule.

- Name** DateTime objects
- Problem** The DateTime object found in PHP and utilized in this project use the normal conventions for weeks in an English calendar, which means they start with Sunday. This was rather troublesome, as calendars in Norway use Monday as the start of a week. Another frequently problem was how DateTime objects was handled between features, often causing changes done to the DateTime object in one place have unintended ramifications in another.
- Solution** Instead of using standard functions existing in the DateTime class, the system calls a global custom “getMonday” function. The DateTime reference problem was solved by cloning the DateTime whenever it was passed, so functions operated on a new object instead of one inherent to another object.

5 — Installation

This chapter introduces topics regarding how to implement the completed software. **Downloads and Location** shows where to get the necessary software, **MySQL** provides how to get the necessary data for the software to operate and **Configurations** is a guide through the configuration options.

5.1 Downloads and Location

- This manual assumes a working installation of Windows Server 2008 R2.
- Download and follow the instructions found on www.apachelounge.com/downloads.
- Extract the files to somewhere in the machine, make note of the path. For these instructions, assume that they are extracted to `C:\webServer\`
- Extract the project files to `C:\webServer\apache24\htdocs\`
- Download a non-experimental, Thread-Safe zip from <http://windows.php.net/download>.
- Extract the files to the same path that apachelounge was extracted to.
- Download the web community MySQL installer (1.5M) from <http://dev.mysql.com/downloads/installer/> by clicking download. You can press “No thanks, just start my download” on the following page to avoid having to create an account on oracle. The MySQL installer might ask you to install .NET, which you can find a download for at <http://www.microsoft.com/en-us/download/details.aspx?id=17113>, click “Download”. If prompted to install additional microsoft software in “Microsoft recommends”, you can press “No, thanks and continue”.

5.2 MySQL

5.2.1 Installation

1. Install MySQL and MySQL workbench through the installer by starting it, then clicking install MySQL products.
2. At setup type, choose custom, as only a few extensions is needed.
3. Check for all basic main tools, uncheck everything else except MySQL workbench.
4. In the configuration menu, choose “Show advanced configurations”
5. Give the root user a password you’ll remember.
6. Choose wanted logs, default is acceptable. Choose path for where the logs go, if so preferred.
7. Start workbench after the installation is complete, and click “local instance *name of instance*”
8. To import the structure of the database
 - (a) Click data import/restore.
 - (b) Click “Import from Self-Contained File”, and set it to “*projectzip*\sql\hdoturnus_structure.sql”.
 - (c) Click “Start import”.
9. To insert data into the structure:
 - (a) Go to “data import/restore”.
 - (b) Click “Import from Self-Contained File”, and set it to “*projectzip*\sql\hdoturnus_data.sql”.
 - (c) Set default target schema as hdoturnus.
 - (d) Click “Start import”

5.2.2 Password

For the sake of security, it is widely recommended to create a limited MySQL user that is only authorized to access site resources, while limiting other potentially harmful operations such as creating or destroying entire databases, dropping tables, and so forth. Since the site only reads and writes data using SELECT, INSERT, UPDATE, DELETE, it can be safely assumed that any other operation should be completely unused, and if executed, would be the work of a malicious user.

You can create the user in one of two ways:

There's a default user bundled in the project folder

1. Go to "data import/restore" again.
2. Click "Import from Self-Contained File", and set it to `"*projectzip*\sql\hdoturnus_user.sql"`.
3. Set default target schema as hdoturnus.
4. Click "Start import"

Or you could create it manually through sql workbench:

1. To create a limited user, open MySQL workbench, connect, and navigate to "Users and Privileges".
2. Login name can be anything you'd like, example: "hdoturnus"
3. Authentication type can be left to standard
4. Limit to hosts matching: localhost
5. Password: Pick any password. This password will only be used once, in the configuration file of the site itself. It is otherwise unimportant to note or remember, and it is therefore recommended to follow practices for very strong passwords. Mix letters, numbers, and symbols, and use many characters (10+).
6. Leave everything in "Administrative Roles" unchecked/unmarked
7. In Schema Privileges click "Add entry". In the following window, first click "Selected schema:", then the database used in this project (hdoturnus by default). Click Ok. In the category of object rights, select "SELECT", "INSERT", "UPDATE", "DELETE". Leave everything else unchecked. Finally, click Apply at bottom

5.3 Configurations

5.3.1 Apache

First of all, note that all paths in apache config uses forward slashes, meaning a path should look like “C:/apache24/”. Comments are prefixed with the symbol “#”, which can be used to leave in notes, sections, categories, comments, etc.

1. Open the apache configuration files with a text editor at C:\webServer\apache24\conf\httpd.conf.
2. Find all occurrences of “c:/apache24/” and replace it with your path, in this instance: “c:/webServer/apache24/”. Only change the directory part of the path. (Note, wrapping the path name in quotes will allow space in the path name.)
3. Add “index.php” to the DirectoryIndex, making it look like this:

```
<IfModule dir_module>
DirectoryIndex index.html index.php
</ifModule>
```

4. At the end of the file, add:

```
LoadModule php5_module C:/webServer/php/php5apache2_4.dll
AddType application/x-httpd-php .php
PHPIniDir C:/webServer/php
```

5. If logs should go somewhere specific edit the line
”ErrorLog ”C:/webServer/apache24/logs/apache_error.log” and
”CustomLog ”C:/webServer/apache24/logs/access.log” common” to your preferred path.
6. Save the changes.

5.3.2 PHP

The paths are written with backslash, and not slash. Lines are commented out with semi-colon (;)

1. Copy “php-ini-production.ini” found in `c:\webServer\php\`, and rename it to “php.ini” and open the file in a text editor.
2. Edit all occurrences of `C:\php\` to say `C:\webServer\php`
3. Edit `extension_dir`: Uncomment the line under “On windows”. Insert your path into into the quotes.

```
        ; Directory in which the loadable extensions (modules
        ) reside .
; http://php.net/extension-dir
; extension_dir = "."
; On windows:
extension_dir = "C:\webServer\php\ext"
```

4. Uncomment the line containing `extension=php_pdo_mysql.dll`
5. Edit `sessions.save_path`: Set to temporary files directory in windows.
6. Add the PHP directory to the environment variables.
 - (a) Go to “Control Panel”, then navigate to system.
 - (b) Open advanced system settings, continue to Environment Variables.
 - (c) Find a variable named “Path” among the System Variables, and append “;C:\webServer\php”.

5.3.3 Project

If you created the database user yourself, or decided to use the root user do the connection:

1. Open `*projectpath*\config\config`
2. Edit “db_user” to the username of your user.
3. Edit “db_pass” to the password of your user.

6 — Summary

This is a summary of the project's conclusion. It looks into the end result and topics surrounding it. **Discussion** brings up matters regarding the software within the perspective of the specification requirements, what was accomplished and the possible expansion options. **Evaluation** discusses the development process and personal opinions regarding the final software. Some final words are offered in **Conclusion**, which concludes the report.

6.1 Discussions

6.1.1 Unimplemented features

- **Caching**

The framework was created with caching in mind. The model system's `find`, and `findSingle` methods will actually look through previous searches to determine if any prior search contains or is equal the requested search parameters. If the search parameters match, it will return the result collection in its entirety. If the search parameters are contained within one of the cached search parameters, the system will automatically filter through each result in the collection, and filter away each result that does not match. The advantage of such a system is that data from previous searches can be reused without querying the database a second time.

Each model is compared using a method aptly named `ModelBase::compare(Modelbase param)`. Each model can override this method if special treatment is necessary. By default, the implementation will assume a match if the cached model A has at the minimum all the same attributes matching the search parameter B's attributes. Using this system, a search can be constructed by creating an empty model, and setting the attributes one wish to search for. The specified attributes will require an exact

match, while the attributes that remained unset will be matched as a wildcard. If the search is not cached, `ModelBase::searchDB`, or `ModelBase::searchDBForSingle` is queried for results from the database, which is then subsequently cached using the previously described system.

Because previous searches are reused, the order in which searches are performed becomes significant. If a generic match-all search runs before a bundle of more specific searches does, maximum reuse of data can be ensured. For this reason, certain parts of the system will preload data using a generic search, to prevent the many subsequent sub-searches from each querying the database. However, the cache system's real potential is in combination with a memory-based cache system such as memcached, where the search results can essentially be reused between many page loads.

Unfortunately, the implementation of memcached, or any similar such system was down-prioritized to the point where it got dropped due to time constraints. As such, while the system has potential to run with great data-efficiency, it wasn't completed, and loses its cache between each page load, which in turn means a lot of the queries that now will run for each page load could have been avoided. Overall, the performance is still clearly within margin-levels considering the scope and size of the project, and a fully optimized solution can be easily delayed until the project scales to 20-40x its size, or the framework is reused in another, much bigger, more demanding project.

- **LDAP integration**

The LDAP system that has been referenced several times could have substituted the user module, using an API to connect to HDOs current user management system. This would, as mentioned, only be done towards the end of the project, as the scope of such an endeavour was difficult to estimate. It would also require access to the system under development, and make testing more troublesome after completion.

This was discarded as time became too limited, which was to be expected. The difficulty of integrating such a system is too unclear, making the risk too high from a development standpoint. It's highly likely that the login system has become too integrated with the rest of the system, making a transition to LDAP unnecessary

difficult, which is opposite to the original intentions of a flexible solution. The `ModelUser` class in particular is probably the biggest obstacle. While the object itself doesn't present many difficulties, all the references to it does. When the references were changes to use identifiers instead of objects, it also changed how users were accessed. The current `find` method retrieves data from the database, and would need modifications to use an LDAP API instead.

- **Notes with extended duration**

The note functionality was quite likely the source of most discussion, in addition to shift exchanges. It didn't differ a whole lot from the previous implementation, but gave a new degree of flexibility that could be explored. Making a custom platform also meant the option to bypass many of the limitations that excel brings with it. One of the examples was the possibility to have a note apply to all the days within a set interval of dates, such as Monday to Friday. The effect would be quite similar to how shift exchanges are handled. Time restrictions and convolutions in trying to implement it resulted in the feature being excluded, as it was ultimately a convenience feature, although it had a lot of merit.

It was under consideration for some time, however, in case a sufficient solution should come up. The database, for example, had attributes to fulfill this possibility in case a solution was devised. Notes contained a field for end dates to specify the duration of the note, and the employee field was used to track whom the note was assigned to. The latter was eventually made redundant by a week template fulfilling the same purpose, and was changed to store the author of the note instead.

- **Separation of different schedules**

Different methods of separating multiple plans that were active simultaneously was discussed under various occasions during April. The software technically already took this possibility into consideration, as it lists all current schedules sequentially in the main view. This implementation had some flaws, as it was primarily an afterthought, and not something it was specifically designed to do. A header was added to serve this new request. The header lists all active plans in the specified time period, and contains links that makes it easier to navigate to these schedules.

A more elaborate solution would require more time to develop. Possible implementa-

tions that were brainstormed included a drop-down menu of all the possible schedules, or splitting them into different tabs. It was omitted in favor of other more beneficial features because such a change would have a minor impact on functionality compared to the current implementation.

- **“Message of the week”**

A message of the week feature was brought up during the last meeting with HDO, at which point the project was nearly over. It would feature a global message to all participants of a schedule, with information for that specific week. Most likely displayed as some form of header.

The application had no support for such a feature at the time, as it hadn't been discussed at any previous opportunities. As such, it would require changes to the interface, updates to the database, and most importantly some sort of method to create and manage these messages. The current stage of the project meant that the emphasis had to be shifted towards writing the report, which didn't leave room for any last minute request. It would in all likelihood have been implemented if it was included in the initial draft, not being too difficult to implement and having distinct benefits.

- **Weekly view**

Early versions of the database included a field for shift deviations, as shown in figure 3.4. The deviations table was intended to store situations where a single shift wouldn't last for the normally specified duration. Rotating shift schedules don't account for such situations due to the rigid time restrictions, so it was discarded from the database. The functionality can technically also be done through the shift exchanges already, although it would be highly cumbersome.

The primary feature that would make use of this was the weekly view, a dedicated display of the number of work hours on each individual day in a week. The week view was included in the early drafts because of various different reasons. It didn't differ too much from the other views, so it required little effort to implement. It also gave the employees more options, and wasn't obtrusive for those who felt no need to use it. Lastly, it was part of the Google Calendar application, the source of inspiration for a majority of the views. When the deviation table eventually got dropped, it

became more and more evident that the week view didn't add a whole lot to the software. Most participants will be well accustomed to the exact definition that a day code represents, so the monthly view is already suited for the situation. A week view is more beneficial if there are several different events that needs to be taken into consideration, which doesn't apply to this case. With that in mind, it was deemed unnecessary and dropped from development.

6.1.2 End results

What is delivered?

The requests made in the project description is met, as it's possible to access and view the plans by users through the internet, and administrators are able to create new templates and manage certain aspects of an active plan. In addition, a minor requirement mentioned in the project description showing the shifts for one user have been accommodated through the implementation of Monthly View and Agenda View. The solution is available for use on mobile platforms as well.

As it was thoroughly specified at various meetings that the original structure of the schedule never should be overwritten while viewing it, early designs to accommodate this was made. These designs lasted in a big part throughout the whole project period, and shows adequately and intuitively both of the original structure as well as changes affecting a single week after they're made like seen in figure 4.4 on page 44

The solution can keep a history of the schedules, which include earlier schedules that are no longer active, how participants of a schedule have changed throughout the active period of it and likewise with the different fixed work assignments mapped to weeks. This feature was requested and not possible to attain in the excel solution without keeping a history of files. The solution supports multiple active schedules at the same time as well, treating them as independent entities, as it should. The pop-up showing expanded shift information by clicking on a day functions much like originally envisioned and written in the early operational requirements, so are the access rights for users of the system.

What is not delivered?

The final solution provides a large degree of freedom with regards to shift exchanges. However, one of the larger appeal factors of the implementation was automating the process to reduce the strain on the administrators. Shift exchanges between two employees usually necessitates that they compensate for it at a later date with another exchange, to keep the total work duration valid. It can be accomplished manually, but there is no method of proposing several related trades simultaneously. It's theoretically possible that the lack of this feature puts too much of the responsibilities on the admin to oversee that everything is handled correctly.

The extensive numbers of browsers and digital devices makes it difficult to properly support them all. Some areas of the interface may be distorted or not work properly on specific platforms, especially on outdated browsers without support for modern features.

The aforementioned inability to create several notes simultaneously is a convenience feature that never reached implementation.

6.1.3 Expansion Opportunities

While the project did complete the specified requirements, they commonly end up expanding beyond the initial goal. Feature request cropped up throughout the project period, and while some were taken into account, some had to be abandoned as well. The most important one, which could possibly be undertaken as a project in itself, would be to integrate the LDAP login system as part of the framework. As mentioned in section 6.1.1 the system ended up being heavily integrated with the user table, making it a bigger project than originally intended. It would require a significant amount of work to reorganize it to take into account the external system, as well as integrating the LDAP interface itself into the software, though libraries for this does exist. However, it's difficult to measure the extent of work that such an undertaking would require, as the group never had the time to fully invest into exploring this possibility.

Another possibility for expansion of this extent is the implementations features that could help the administrators in their tasks by adding information of how much an employee has worked, and how it translates in the form of overtime and payment. This features could

include the creation of new schedules as well, not allowing changes that would infringe the laws and regulations of Norway.

A resource intensive feature that could be implemented, but probably isn't big enough to justify as a bachelor project is the possibility to have a note written for all shifts in a certain period, as it would have improved the usability of that feature immensely.

Other aforementioned features are likely not as resource intensive to fulfill, but could still provide a significant improvement for the software. The structure doesn't put too many restrictions on how to create a new interface for several active plans, nor does a "Message of the Week" demand a whole lot other than a new database table and the proper functionality to edit it. The export feature also has the flexibility to be expanded for other formats.

6.2 Evaluation

6.2.1 Tools

Several tools were used during development, both familiar and new. The majority of these were adopted with ease, but some were of higher relevance than others. In particular, the PHPStorm IDE, Trello task board and Mercurial repository were both new and of significant importance.

Trello The Trello Task Board was used with the intent of organizing the Kanban development process. While utilized in previous projects, it had never been used to the extent that was intended this time around. Trello certainly proved it's worth in the initial stage of development, where it was a valuable tool in outlining the feature list. However, most of the work were co-ordinated between the group members on a case-by-case basis, so Trello lost a lot of it's purpose after a while. It continued to be used individually to some extend, in part as some advanced checklist software. Ultimately, it proved useful, but was never fully actualized for it's intended purpose.

Mercurial A repository is almost mandatory for software development when working in a group. The group adopted Mercurial and BitBucket as their synchronization tool of choice,

over Subversion and Git which have been used for previous group assignments. Having been underwhelmed with the aforementioned softwares, the group adopted Mercurial and found it to be to its liking. Mercurial can technically be used to support Git as well, so the decision to host it at BitBucket instead was quite likely arbitrary. Using the tool before getting accustomed to the featured resulted in some difficulties. The most common one was the resolution of merge conflicts, which has been troublesome in other software as well. It was resolved when the group became accustomed to the update feature, after which it became trivial to keep an updated version of the software at any point. A somewhat reckless consistency in pushing features did have some repercussions on group performance whenever severe bugs would crop up, but it also helped resolve said bugs, as well as supporting the Kanban development method.

PHPStorm The group chose JetBrains' PHPStorm as the IDE of choice despite being familiar with Eclipse. Because of a well developed ergonomic design, it was easy to get accustomed to. Furthermore, having a shared IDE made it easier to learn the features through each other. Using the same interface was also useful when drafting more general programming questions.

PHPStorm also had a lot of new and convenient features. While general utilities such as autocomplete, syntax highlighting and error checking were certainly appreciated, they didn't play any major roles in the development other than increasing effectivity. Of more particular note is the database integration and debugging environment. Being able to manage the database without doing a context switch to the mysql interface saves a lot of time, and reduces the number of areas to interact with. The XDebug integration, on the other hand, was invaluable in finding obscure errors through warning messages and run-time object inspection.

6.2.2 Process

The work schedule identified in section 1.2 has been followed throughout the project period as specified with communal work on Tuesdays and Wednesdays. There was situations where they didn't start entirely on time, but usually lasted well past 18:00, commonly into the range of 20:00 to 21:00. Group development also occurred on Thursdays from time to time, as the productivity often diminished on individual work days. These deviations were

usually for reaching specific deadlines. Fridays were originally meant for guidance with the supervisor and demonstrations for the product owner. Discussions about the meetings were supposed to fill the rest of the day. However, it commonly resulted in group development, a result of the supervisor becoming less important during development, and most discussion topics reaching a consensus relatively quickly.

The work schedule was maintained with acceptable consistency, but was insufficient to uphold the initial expectations. The original progress plan underwent some adjustments to reflect the current situation, as the initial draft was deemed to be a little too optimistic. Further adjustments had to be considered around April times, as some features had taken too long to implement. There are several possible explanations as to why things ended up this way:

- The tasks in the list of functionality and features that had to be implemented were too wide, and tried to cover too much. This made it hard to track the progress of them, and the project as a whole, since the tasks never reached a finished state.
- The time it would take to implement a feature was estimated in days in the original work breakdown sheet. Small problems could therefore have severe impacts on the estimation dates, compared to an estimation system with work hours.
- A new schedule was not written as the initial development fell behind schedule. This resulted in each individual task not having a deadline that the members of the group could relate to.
- The deadlines of the original schedule was poorly adhered to in the first place, only utilizing it for the implementation sequence.

Work related to project management was generally seen as low priority tasks, and often dropped because of that. This made it hard to keep the worklog updated, which was intended to contain the number of hours each member used on different tasks each day. The final log, found in appendix C is the product of that endeavor, tailored together with the commits and commit messages from the bitbucket repository. The reduced priority of the managerial tasks made it so that the second and third progress report never got made.

A test server was set up in early April, giving HDO the possibility of testing the currently implemented features. It would probably have been a better idea to set up the

server earlier, giving them access to test as the development progressed. This would also let other employees get accustomed to the software, increasing the options for white-box testing. Most of the feedback on the prototype came from only one individual.

6.2.3 Subjective Evaluation

Working with a development project on such a grand scale has been an exciting venture into the real world, bringing with it a fair share of positive encounters and difficult situations. Being left to our own devices to solve pressing issues was both exciting and frightening, knowing that we are responsible for a software that HDO could potentially use several years into the future if well executed.

Being in a development group for such an extended period of time was also a something relatively new, although prior experience in working together have already made us accustomed to each other's traits. The most relevant aspect introduced was likely trying to have an organized work schedule, in which we had partial success. We feel that the group cohesion was good, but with certain rooms for improvement.

Having a real employer to work together with was the most refreshing aspect, and provided an overall positive experience. Getting continuous feedback on the progress was both encouraging and educational. Most important was the need to account for the requests of an external source, and conclude on a common consensus on how to fulfill it. Meetings every other week was a great discussion opportunity, as well as a motivation for performing better towards the end result, creating virtual deadlines on certain features.

We are very pleased with how the Standard Plan ended up, which we agree is the most well-executed part of the software. It's visually clean, and provides sufficient information to the user without demanding a lot of effort. While it has no extraordinary features to speak of, it's overall quite solid. We appreciate how it turned out, as the centerpiece of the application. The Template Editor, on the other hand, is the most underwhelming implementation in our opinion. It looks a little alien, not adhering to the consistent interface of the rest of the application. It's also rather unworkable through a mobile interface, as it was designed with efficiency towards keyboard and mouse.

6.3 Conclusion

The opportunity to work with a case of this large a scale has been a valuable experience for the group. Having such a tight relationship with the employer and putting together our collective knowledge provided an environment closer to a real scenario than previous group projects have up until this point. It also gave us the ability to see the full value of the bachelor degree courses from a new perspective, from the beginning to end of the development cycle.

There are certainly things left undone, that would have been nice to fulfill. However, as with most development projects, certain features will have to be prioritized in favor of others. All the initial feature demands were sufficiently fulfilled at the end of the day, leaving only requests that cropped up under development, as well as software tweaks to improve existing solutions. Our priorities for the future would quite likely be implementing a message of the week system, data caching and interface tweaks on the Template Editor. Nevertheless, it ultimately fulfills all the necessary functionality, and prototypes indicated that HDO will put it to good use.

Attempting to use a development methodology for an extended period of time was an important learning experience. There were some glaring issues in the attempt to utilize it, but it wasn't entirely without merit either. The biggest benefit probably came from using a lean development process, involving small tasks and regular meetings, both with HDO and within the group. Keeping the repository updated and individual tasks small was handled sufficiently without too many faults. However, using the namesake task board should with certainty have been utilized better. It was rarely taken into account with regards to task distribution, often leaving some tasks under development while fulfilling others. It was at times necessary when a segment unexpectedly necessitated the development of another feature, but it shouldn't have occurred with the frequency, even in spite of such events.

The other significant educational aspect of the bachelor project is interacting with someone else invested in your project, providing the perspective of a consumer. Previous endeavours have been very isolated in comparison, so it was great to have some expectations to strive towards. We are personally satisfied with the end result, and hope that HDO benefits from our parting gift, signaling the conclusion of three years at Gjøvik University College.

Bibliography

Sources:

- [1] HDO. Om HDO[Online]
Available from: https://www.hdo.no/no/Om_HDO/
- [2] Directorate for emergency communication. Nødnett - a brief description -[online]
Available from: <http://www.dinkom.no/en/Development-of-Emergency-Network/About-Nodnett-the-Norwegian-Public-Safety-Network/Nodnett—a-brief-description/>
- [3] ShiftPlanning. Product Touronline
Available from: <http://www.shiftplanning.com/tour/>

Tools:

- [4] rellor[online]
Available from: <https://trello.com/>
- [5] PHPStorm[online]
Available from: <http://www.jetbrains.com/phpstorm/>
- [6] Bitbucket[online]
Available from <https://bitbucket.org/>
- [7] Mercurial[online]
Available from: <http://mercurial.selenic.com/>
- [8] Google DevTools[online]
Available from <https://developer.chrome.com/devtools/index>

[9] xdebug[online]

Available from <http://xdebug.org/>

Libraries:

[10] jQuery Foundation. jQuery[online]

Available from: <http://jquery.com/>

[11] jQuery Foundation. jQueryUI[online]

Available from: <http://jqueryui.com/>

[12] Rodney Rehm. jQuery contextMenu[online]

Available from: <http://medialize.github.com/jquery-contextMenu/>

[13] Anders Fajerson. timePicker[online]

Available from: <https://github.com/perifer/timePicker>

[14] Brian Grinstead. Spectrum[online]

Available from: <http://bgrins.github.io/spectrum/>

[15] Detect mobile browser - Open source mobile phone detection[online]

Available from: <http://detectmobilebrowsers.com/>

[16] QueryPath: Find your way[online]

Available from: <http://querypath.org/>

Appendices

A — Terminology

B

BitBucket	Web-based hosting system for the revision control systems Mercurial and Git.
------------------	--

C

Client	A program accessing a server, often used to denote the user in a client-server relationship.
---------------	--

CSS	Cascading Style Sheets, a language used to format markup languages. Commonly used to determine the layout of web pages together with HTML.
------------	--

csv	Comma-separated values. Some eCalendars use this format.
------------	--

D

Dropbox	Free cloud based hosting service for file storage and synchronization.
----------------	--

E

Eclipse	Open source IDE written in Java that uses plugins to support numerous programming languages.
----------------	--

Excel	Microsoft's spreadsheets solution included in the Microsoft Office package.
--------------	---

G

Google Drive	Cloud based file storage application hosted by Google, commonly used as a collaborative text editor.
Google Calendar	Calendar service used to synchronize and view schedules.
H	
<hr/>	
HDO	Helsetjenestens Driftorganisasjon.
HTML	Hypertext Markup Language, the standard markup language used for creating web pages.
HTMLElement	Parent class for classes tasked with creating the content of web pages.
I	
<hr/>	
IDE	Shorthand for Integrated Development Environment, applications designed for software development.
iCal	Format readable by most eCalendars
J	
<hr/>	
JavaScript	Programming language commonly implemented as client-side scripts in web sites to dynamically alter page content.
jQuery	Open Source JavaScript library commonly employed for HTML traversal and manipulation.
L	
<hr/>	
LDAP	Shorthand for Lightweight Directory Access Protocol, an application protocol used by HDO's login system.
M	
<hr/>	
Mercurial	Distributed revision control system and interface.
ModelBase	Superclass for objects that represent the stored database values.
MySQL	Open Source relational database management system developed by Oracle.

P

PHP	Server side programming language for websites.
PHPStorm	PHP IDE developed by JetBrains.
Page	Custom class used to handle page requests from the browser.

R

Rotating Shift Schedule	A list of predetermined weekly schedules that is distributed among a group of employees and rotated between the participants after each week.
--------------------------------	---

S

Schedule	Used as a shorthand for Rotating Shift Schedules.
Server	A system used to manage a computer network, such as hosting a website.
spe	Shorthand for the custom DOMElement custom class implementation.
Stack	Software implementation using the Last-in-First-out data structure, where items are retrieved in the reverse order of how they were deposited.

T

Trello	Web based task board application used for collaborative task management.
---------------	--

Q

Query Path	DOM traversal PHP library.
-------------------	----------------------------

B — Meeting Summaries

15.1.2014

- HDO
 - Notes
 - * Administrator should be able to write notes on the shifts of all employees, while other employees should just be able to write on their own.
 - * Note history?
 - * Comments with author.
 - Day deviations should be written as comments, and not change the viewing of plan.
 - HDO wants the possibility to export templates
 - Accesscontrol
 - * Possibility to see standard view without login.
 - * Log-in for user specific information and adding notes.
 - * Seperate administrator "type"
 - Administrator must approve all shift changes.
 - possibility to create plans running on the side of the active plan.
- Decisions
 - Treat "no-shift" as an type of shift with no time.
 - Ask for vacation the same way as an shift is changed.
 - Remove day-deviations from the database.
 - Drop-down menu for shift change.

21.1.2014

- Decisions
 - We'll test php-storm and buy licenses if we think it works well.
 - Try to sell the idea of keeping a frame around a day as the original template, but change the content, including shiftname.

31.1.2014

- HDO
 - Framework will handle GUI creation and database connection.
 - We estimate to begin actual development in one week.
 - See own role.
 - Define weeks.
 - Need to be able to change participants in active schedule.
 - * Make weeks "open"/"free".
 - Weeknotes
 - * Worktask supposed to be manned whole week.
 - Error reports
 - Modulbased profilesystem
 - * Can be integrated with LDAP later.
 - * Profiles must be created by administrator.
 - Create installation package.
 - Administrator need to be able to set days available for change, or do it manually.
 - Exchanged shifts should preferably be of the same type.
 - Browser versions 1-2 versions backwards compatibility. (IE and Chrome mentioned.)
 - Server: Windowsserver (Normal machine/2008 R2)

14.02.2014

- HDO

- Standard view
 - * White symbols in the plan
 - * Seem happy with current design.
 - * Seemed happy with current visualization of day comments
 - Importancy of author.
 - Mark author type
 - * Keep hours per week.
 - * Discussion: Permament notes on weeks
 - Beside name of user. (Solution of today.)
 - Beneath name of user.
 - Directly beneath shiftcode.
 - * Weekly sum of hours.
 - Always show sum for original template.
 - Show correct sum after shift exchanges.
- Editor
 - * Show average on all weeks in editor.
 - * History of earlier schedules.
 - * Autofill combobox for shifts.
 - * Autocreate new week.
 - * Navigation.
- Active schedule
 - * Change color on shift where employees are not present.
 - * Switch
 - Actual exchange of two shifts.
 - Switch/overwrite
 - * Shifting
 - Days should not be changed.
 - Do not show as exchange.
 - Possibility to change a dayshift after a nightshift.

14.3.2014

- HDO
 - Standard view
 - * Show multiple schedules.
 - * Normal weekly hours without changes, and hours with change.
 - * More employees at the same week? (Likely not).
 - * Bulk editation of days
 - Show day
 - * Edit a note.
 - * Print pop-up
 - * Add change.
 - * Full name ("Dag", "Dag(lengre)")
 - Editor
 - * Decide number of weeks first.
 - * On mouseclick.
 - Shift editor
 - * Shift information is never changed for a code.
 - * Restricted color choice is alright.
- Decisions
 - Button to copy templates
 - Let all plan editation(set active, enddate, fill with employees) happen in the same editor.

18.3.2014

- Discovered time problem
 - The cache system
 - Make models slightly less integrated.
 - Query path
 - Dynamic generation of objects.

- Update
 - * Nearly solved.
 - * 116 DB queries to 20
 - * QP: Nearly completely rewritten.

28.3.2014

- HDO

- Export: Keep shift codes, start and end for template export.
- Export: Standard Outlook calendar.
- Schedule: Set start of schedule to end of previous.
- E-mail: Standard outlook mailservers.
- E-mail: Shift changes

25.4.2014

- HDO

- Shift exchanges
 - * Unsure about automatic shift exchanging among participants due to restrictions.
 - * Finish possibility of administrators changing shift over a period.
- Bulletin: Box at top of page showing information specific for week. (Message of the Week)
- “Bakvakt”: Shift set as lasting the whole day, but only a part of it is effectively considered as work hours.
- John will try to work on site some time next week, trying to implement and test mail services.
- Assortment of bugs listed on trello.

C — Work Log and Progress Report

Work Log

Week 3 — 13.1- 19.1

Met HDO to get the process started and specify some requirements.

Had initial meeting with Frode Haug, our supervisor, discussing the process.

Glen: Projectplan and requirement specification

John: Projectplan and requirement specification

Martin: Projectplan and requirement specification

Week 4 — 20.1 - 26.1

Got feedback about the project plan and requirements from Frode

Glen: Projectplan and requirement specification

John: Development - Framework

Martin: Projectplan and requirement specification

Week 5 — 27.1 - 2.2

Meeting with HDO, discussed our understanding of the requirements.

Glen: Projectplan and requirement specification

John: Development - Framework

Martin: Projectplan and requirement specification

Week 6 — 3.2 - 9.2

Meeting with Frode, discussed start of development.

Glen: Web page content creation, figure creations.

John: Development - Framework

Week 7 — 10.2 - 16.2

Meeting with HDO, first demonstration, mostly early design drafts.

Glen: Created initial testdata, Development - Shift Template editor

John: Development - Framework, Development - basic webpage structure

Martin: Development - Standard view

Week 8 — 17.2 - 23.2

First progress report, discussed what discovered with Frode.

Glen: Development - Shift Template editor

John: Development - basic webpage structure, Development - Standard view

Martin: Development - Standard view, Development - Admin panel, Development - Template editor

Week 9 — 24.2 - 2.3

Glen: Development - Shift Template editor

John: Web page - Produced code, Development - Standard view

Martin: Development - Admin panel, Development - Template editor

Week 10 — 3.3 - 9.3

Glen: Development - Shift Template editor

John: Development - weekly notes, Development - Pop-up system.

Martin: Development - Admin panel, Development - Template editor

Week 11 — 10.3 - 16.3

Meeting with HDO, second demonstration. There had been much progress since the previous, as there had been no meeting for four weeks.

Glen: Development - Shift Template editor

John: Development - Pop-up system, Development - view day

Martin: Development - Template editor

Week 12 — 17.3 - 23.3

Glen: Development - Shift Template editor, Research - Export,

John: Development - Pop-up system, Development - view day

Martin: Development - Template editor

Week 13 — 24.3 - 30.3

Demonstration for HDO, not much progress.

Glen: Development - Export, Development - Create Shift templates through editor

John: Development - Framework(answer time problems)

Martin: Development - Template editor, Development - Plan manager

Week 14 — 31.3 - 6.4

Glen: Development - Export, Development - User creation and management

John: Development - Standard view, Development - Monthly view, Development - view

day

Martin: Development - Plan manager

Week 15 — 7.4 - 13.4

Glen: Development - User creation and management, Development - View agenda

John: Development - view day, Development - Shift edit

Martin: Development - Plan manager

Week 16 — 14.4 - 20.4

Easter. Group got somewhat split.

Glen: Development - View agenda, Development - Admin Panel, Development - Template Editor

John: Development - Shift changes, Development - Layout

Martin: Development - Plan manager

Week 17 — 21.4 - 27.4

Last development meeting with HDO, still work to do.

Glen: Development - Template Editor, Development - Template Export, Development - Participant changes.

John: Development - Shift changes, Development - Participant changes.

Martin: Development - Plan manager

Week 18 — 28.4 - 4.5

Glen: Report - writing, Development - Day duration

John: Development - Plan manager, Development - Shift changes, Development - Emails.

Martin: Report writing

Week 19 — 5.5 - 11.5

Glen: Report - writing, Development - Bug fixing

John: Development - Bug fixing, Development - Mobile tabs

Martin: Report - writing

Week 20 — 12.5 - 18.5

Meeting with Frode regarding the report.

John worked on-site to check out some bugs and such discovered by Reidar.

Glen: Report - writing

John: Development - Bug fixing, Development - emails, Report - writing

Martin: Report - writing

Progress Report 20.2.2014

Status

Planning/Work breakdown sheet

Framework

The framework has been implemented, but will be expanded to meet further functionality as it becomes necessary.

Standardplan

Supposed to be finished as of 19.2. While the logical structure is in place, but some design choices remains undecided.

Still in progress, and will be revisited.

Standard admin tools

Supposed to be finished by 26.2. Work on the implementation is in process.

Organization of work and responsibility

The group as a whole needs to improve at task distribution. The Kanban principles aren't utilized enough in the workflow, such as the taskboard on Trello and daily meetings. This is mitigated to a certain degree by the frequent meetings throughout the weeks.

Report

There has been no noteworthy work done on the report since the completion of the project plan.

Summary

The group faces a minor delay with regards to the standard plan view, but the project as a whole is mostly on track. There needs to be more awareness regarding the development method, and some work should be delegated towards the report.

Risks

Our work breakdown sheet is poorly planned with short time estimates, which may make us fall disproportionately far behind as time goes by.

Finished functionality

The logical structure of the application, the framework, is completed and currently used in the other tasks. It will evolve to accommodate for future needs, but is unlikely to undergo any radical changes in structure.

The methods for processing data with regards to the standard plan is nearly complete, and the design is at a satisfactory level.

We expect our current database design to stay without any big changes for the rest of the project. Small changes may still occur.

Work in progress

We're currently developing controllers and interface for managing shifts(days) and schedules(templates), as well as working on the design for the administrator panel. Some components needed for these views are being worked on as well, like pop-up windows and autocomplete comboboxes.

The standard plan needs some finishing touches.

Time constraints

We intend to have a presentable prototype for the product owner at our next meeting on 07.03.2014. The prototype should be able to accommodate for the basic functionality already handled by their current spreadsheet implementation.

Motivation

There have been no noteworthy incidents within the group that has caused any impact on the motivation.

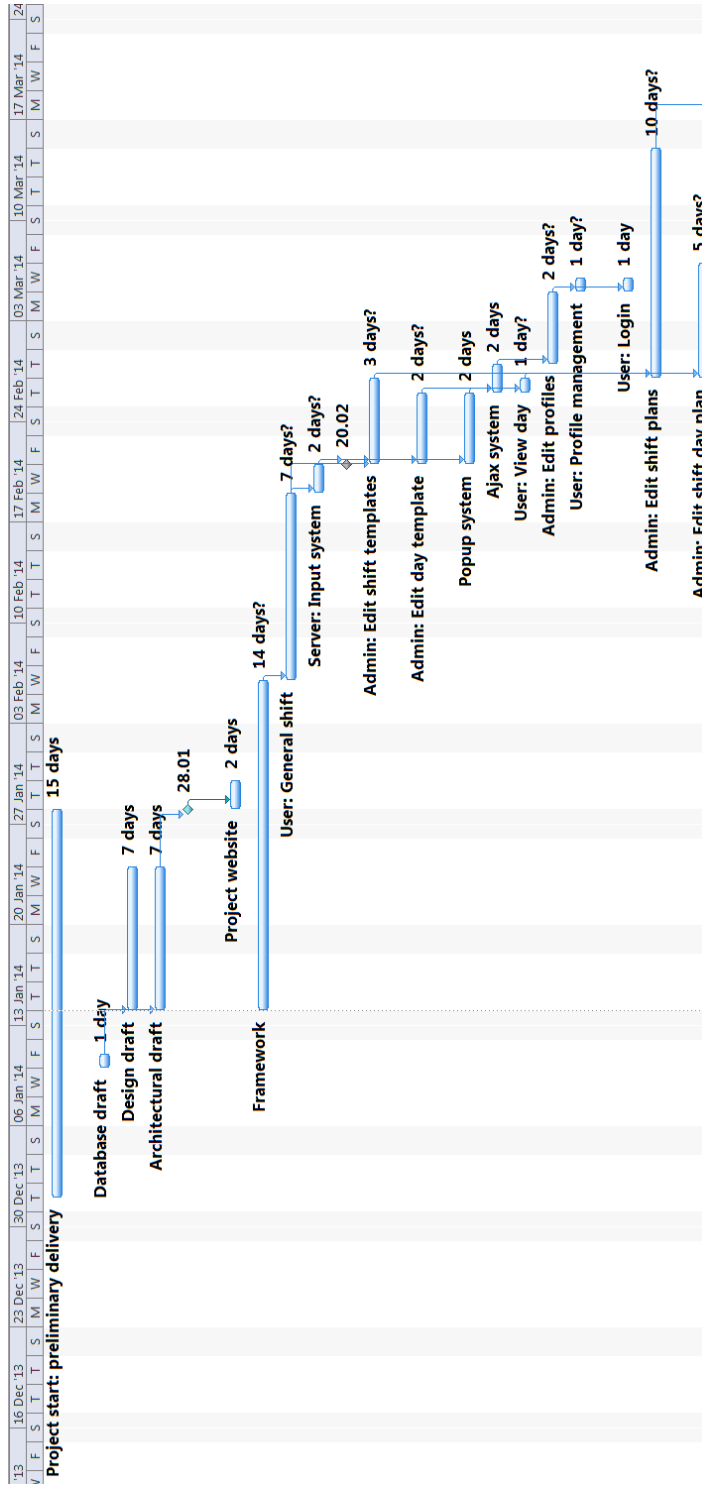
Supervisor

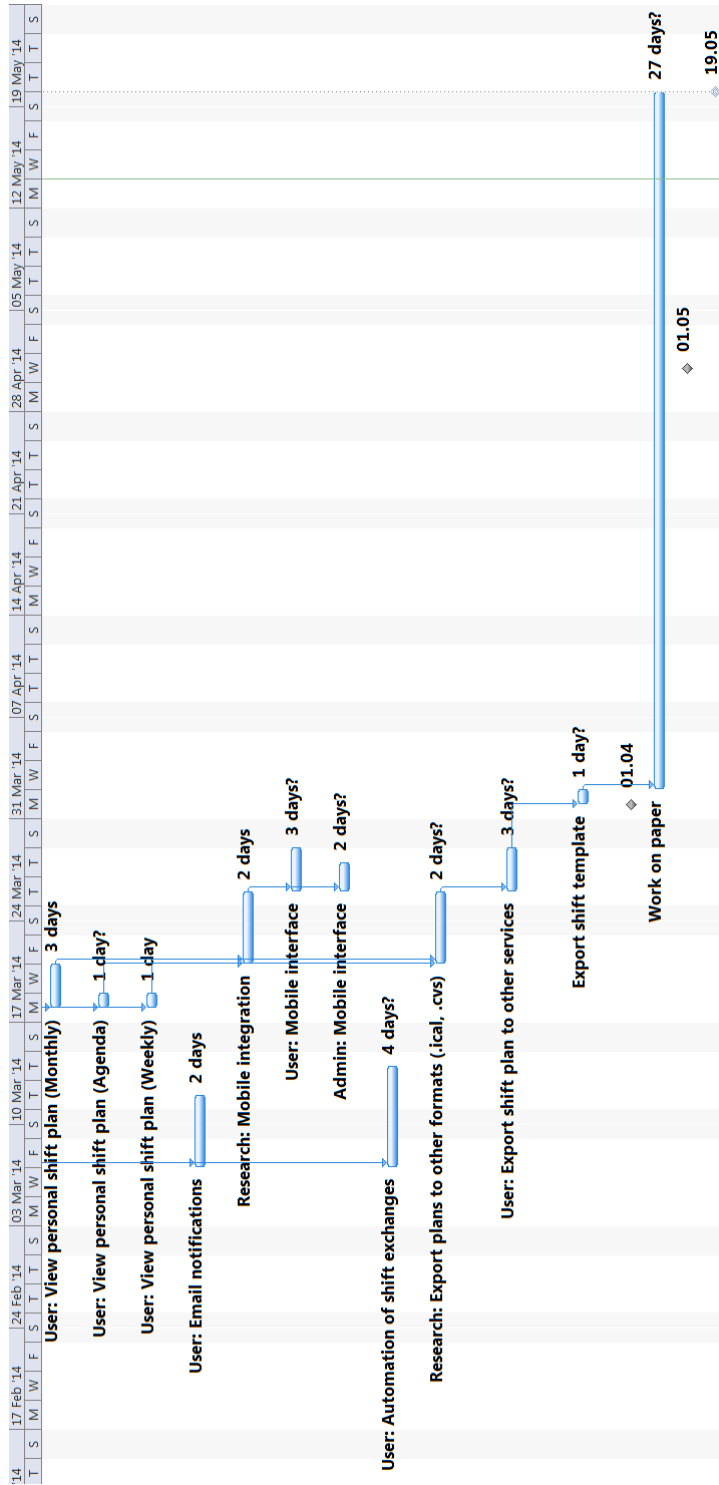
The group is somewhat uncertain about how to properly utilize the supervisor, but has no major qualms.

Progress Report to Frode, 7.3.2014

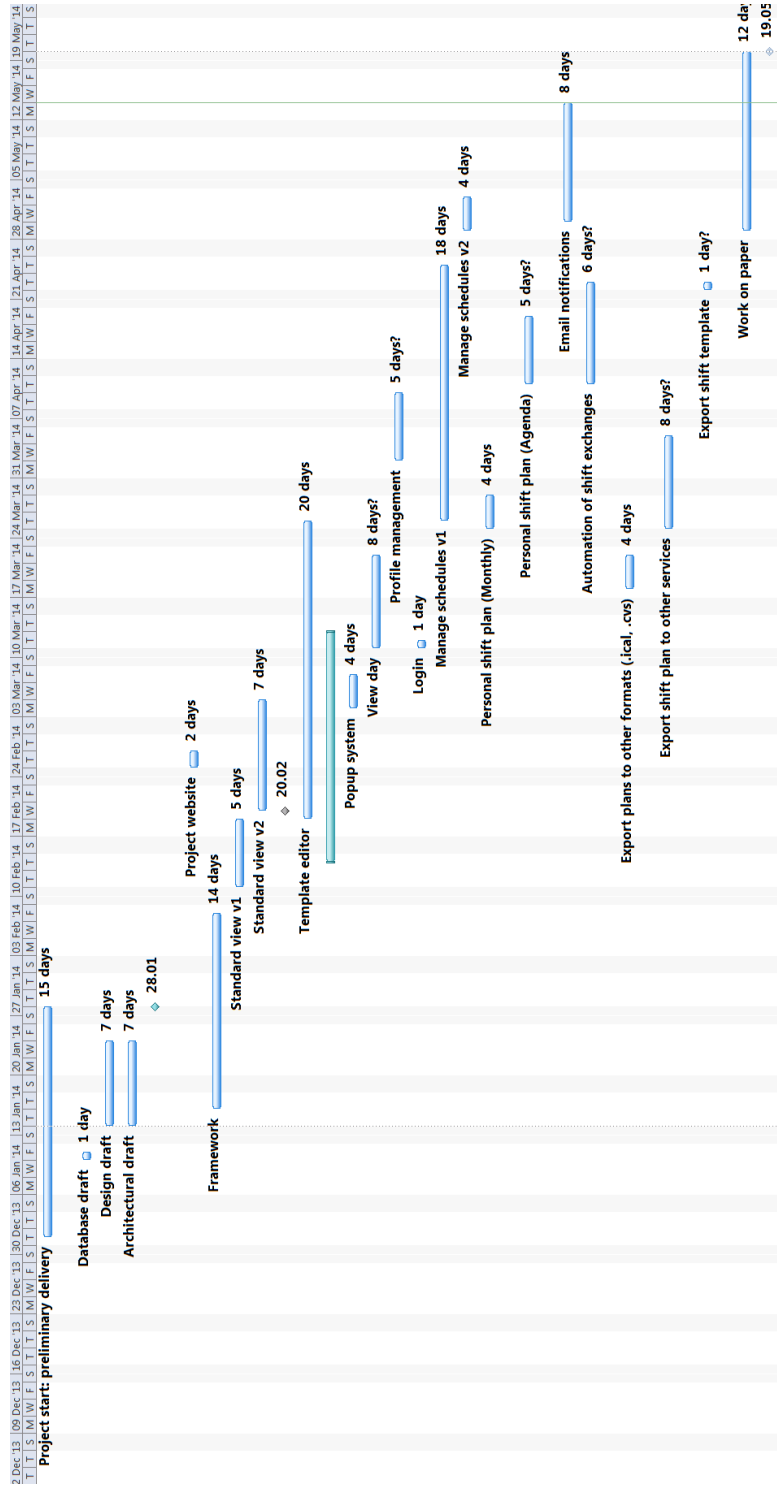
- The solution will be able to do what the current solution do relatively soon.
- The progress plan is likely breached, and might need to be reworked.
- There will be no demonstration with product owner in about a month, due to time off and vacation

D — Progress Plan A





E — Progress Plan B



F — Project Agreement



HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Helsetjenestens Driftsorganisasjon (HDO)

(oppdragsgiver), og

John Hæge-Omdal

Glen Martin Nordhagen Gronoan

Martin Jonassen

(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 7/1-2014 til 19/5-2014.

Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG 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 HiG 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 HiG. Studentene dekker utgifter for trykking og ferdigstillelse av den skriftlige besvarelsen vedrørende prosjektet.
 - 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. HiG står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av faglærer/veileder og sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.
4. Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode, disketter, taper mv. som inngår som del av eller vedlegg til besvarelsen, gis det en kopi av til HiG, som vederlagsfritt kan benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av HiG 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 HiG og/eller studenter har interesser.

Besvarelser med karakter C eller bedre registreres og plasseres i skolens bibliotek. Det legges også ut en elektronisk prosjektbesvarelse uten vedlegg på bibliotekets del av skolens internett-sider. Dette avhenger av at studentene skriver under på en egen avtale hvor de gir biblioteket tillatelse til at deres hovedprosjekt blir gjort tilgjengelig i papir og nettutgave (jfr. Lov om opphavsrett). Oppdragsgiver og veileder godtar slik

offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og dekan om de i løpet av prosjektet endrer syn på slik offentliggjøring.

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 Fronter. I tillegg leveres et eksemplar til oppdragsgiver.
8. Denne avtalen utferdiges med et eksemplar til hver av partene. På vegne av HiG er det dekan/prodekan som godkjenner avtalen.
9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og HiG som nærmere regulerer 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, skjer dette uten HiG som partner.

10. Når HiG også opptrer som oppdragsgiver trer HiG inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.
11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene i mellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

HiGs veileder (navn): Frode Haug

Oppdragsgivers
kontaktperson (navn): Reidar Honningsvåg

Student(er) (signatur): Glen M. N. Utangan dato 15/1-2014

Markus Jansson dato 15.1.2014

John Hoegh-Omdal dato 15.1.2014

_____ dato _____

Oppdragsgiver (signatur): Reidar Honningsvåg dato 15/1-14

IMT Dekan/prodekan (signatur): _____ dato _____