BACHELOR THESIS:

CONTRACT MANAGER – A MODERN APPROACH:

Creating a Contract Management System

Using Modern Designs and Frameworks

AUTHORS:

Joakim Jøreng
Martin Storø Nyfløtt
Thomas Mellemseter

DATE: 14.05.2015

ABSTRACT

Title:	Contract Manager – A Modern Approach: Creating a Contract Management System Using Modern Designs and Frameworks		Date : 14.05.2015
Participants:	Joakim Jøreng Martin Storø Nyfløtt Thomas Mellemseter		
Supervisor:	Tom Røise		
Employer:	Electronic Time Car AS		
Keywords:	Contracts, web, JavaScript, Java, REST, SPA		
Number of pages: 89	Number of appendix: 9	Availability: Open	

This thesis describes the process around the development of Contract Manager, a system for administration of contracts in cooperation with ETC AS. The system lets users administrate contracts through a web-application that is also scalable on mobile devices. Additionally, the project involves two hybrid-apps that makes it possible to take pictures of contracts as an attachment and lets the user receive push notifications regarding their contract statuses. The target group for this system is home-users and organization users.

Contract Manager consists of a web-application, two mobile-applications and one backend. The mobile apps and the web-app communicates with the backend through a REST-full web-api. The system also involved Facebook integration for login and a subscription module that uses PayPal.

Much of the focus throughout the project has been on using modern, popular and well established frameworks that solves several problems web-applications face today regarding scalability on different devices, modularity and flexibility.

SAMMENDRAG

Tittel:	Contract Manager – A Modern Approach:		Dato : 14.05.2015
	Creating a Contract Management System		
	Using Modern Designs and Frameworks		
Deltakere:	Joakim Jøreng		
	Martin Storø Nyfløtt		
	Thomas Mellemseter		
Veileder:	Tom Røise		
Oppdragsgiver:	Electronic Time Car AS		
Stikkord:	Kontrakter, web, JavaScript, Java, REST, SPA		
Antall sider: 89	Antall vedlegg: 9	Tilgjengelighet: Åpen	

Denne oppgaven beskriver prosessen rundt utviklingen av Contract Manager, et system for administrering av kontrakter, i sammarbeid med ETC AS. Systemet lar brukere administrere kontrakter via en web-applikasjon som også er skalerbar på mobile enheter. Det er i tillegg utviklet to hybrid-apper som gjør det mulig for brukere å ta bilde av kontrakter og legge de inn i systemet, samt motta push-notifikasjoner for varsler om kontrakter. Målgruppen for systemet er privatbrukere samt organisasjonsbrukere.

Contract Manager består av en web-applikasjon, to mobil-applikasjoner og en backend.
Mobilappene og web-appen kommuniserer med en server via et REST-full web-API.
Det har også blitt implementert Facebook-støtte for login og en betalingsmodul som tar i bruk PayPal som betalingstjeneste for systemet.

Mye av fokuset igjennom prosjektet har vært på å ta i bruk moderne, populære og veletablerte rammeverk som løser flere problemer web-applikasjoner møter i dag deriblant skalerbarhet på forskjellige enheter, modularitet og fleksiblitet.

Preface

We would like to thank everyone that has contributed to this project. Special thanks goes to the employer of this project, Dag L Solhaug, who made this project possible. Thanks to Tom Røise for excellent supervising through the entire project. Thanks to Eivind Arnstein Johansen for giving us feedback on the user-interface, and thanks to Mark Bertels (NL) for helping us with Hibernate. We would also like to thank Dennis André Østvik Gjerdingen, Pål Storsveen and Trine Jeanette Storsveen for taking our user tests.
We would also like to thank the staff members at Gjøvik University College (GUC) that has answered questions related to this project and anyone else who has not been mentioned earlier who has contributed in any way.

Gjøvik, 14.05.2015

Table of Contents

1. Introduction 1
    1.1 Background 1
    1.2 Project Description 1
    1.3 Scope 2
    1.4 Target Audience 3
    1.5 Application Target Audience 3
    1.6 Purpose 3
    1.7 Academic Background 4
    1.8 Roles 4
    1.9 Glossary 5
    1.10 Document Structure 6
2. Project Management 7
3. Contract Lifecycle 10
4. Project Planning 11
    4. 5 Conceptual Data Model 11
    4.6 User Types 11
    4.7 Legal 12
    4.8 Groups and Rights 13
    4.9 Backlog 14
    4.10 Survey 14
5. System Architecture 16
6. Technologies 20
    6.1 Restlet 20
    6.2 Hibernate 22
    6.3 Angular 23

6.4 Bootstrap 25

6.5 Android and Windows 28

6.6 Gradle 29

6.7 Grunt and Bower 30

6.8 Project Dependencies 31

7. Design 33

7.1 Angular 33

7.2 Implementation of Restlet and API 46

7.3 Hibernate Integration 51

7.4 Graphical User Interface 55

7.5 Internationalization 67

7.6 Hybrid Apps 70

8. QA 74

8.1 API 74

8.2 SonarQube 76

8.3 Angular 77

8.4 JSHint 79

8.5 User Testing 79

9. Technical Memos 81

9.1 Hosting Options for Java 8 Web-Applications in the Cloud 81

9.2 Secure Storage of Passwords 82

9.3 Platform Selection 83

10. Summary 85

10.1 Results 85

10.2 Further Development 86

10.3 Group Evaluation 87

10.4 Conclusion 88

References 90

Appendices 97

A – Project Plan 97

B – Hour Log 113

C – Meeting summaries 122

D – Backlog 134

E – Group Rules 137

F – Deployment on Microsoft Azure using FTP 138

G – User Tests 143

H – Email Communication with the Norwegian Data Protection Authority (Norwegian) . 146

I – Project Agreement 149

1. Introduction

1.1 Background

During our everyday lives, we get relations with several different vendors. Typically through contracts, such as a rent, insurance, or a mobile subscription. These contracts are kept in different IT systems or kept as a sheet of paper hidden in a drawer nobody knows where is. With an even increasing stream of services from the digital world: bookkeeping, managing and controlling all the different contracts and subscriptions becomes increasingly difficult. Organizations may often want to get an overview and find a specific insurance contract without having to use time on tracking down a specific coworker, and then wait for that person to locate and fetch the contract while the server park is literally on fire. This problem also applies to home users, but in another context. In other words, no one wants to spend time finding contracts.

Electronic Time Car AS (ETC) is an innovative IT-company with cutting-edge solutions for managing vehicles used by three or more drivers in small and large car pools. Dag L. Solhaug, the CEO of ETC, has recognized the need for a system where all contracts are organized and where users can archive their contracts and get notifications when these expires. He wants the system to aim at a new market for contract management despite differing from the market ETC usually targets. This system should also be available on several platforms as users has an increasing use of tablets and mobile devices both professionally and at home. Additionally, the system should be able to store the contract and its associated information in a safe and secure manner available as a subscription system.

Contract Manager attempts to solve the issues described above. It is a system designed for both private users and organizations available through different subscription models depending on the user type.

1.2 Project Description

In brief overview, the system makes it possible for users to store and manage contracts. It helps users to organize their contracts by organizing them through this system. Different configurations based on the contract type and the life span of the contract determines how the system should act. Several users can be added to a contract as a notification receiver that will make the system notify these users once a contract should be renewed or is about to expire. Users can also easily find contracts by searching using keywords for metadata of a contract.

The organization administrator can manage their users, who has access to which contracts and which rights they have. This is managed by separating users into groups and contracts into categories. Home users does not have these options, but they have a simplified version of categories.

The system is available as a web application and a mobile app. These will then communicate to a backend (referred to as the API) that stores the user data in addition to the contracts and provides a common interface for both the mobile and the web application.

1.3 Scope

Field of Study

- Responsive web application using HTML, CSS and JavaScript
- Android mobile application development
- Windows application development
- RESTful API-design
- Java servlets
- Internationalization
- Relational database design together with an ORM framework
- Unit testing
- Contracts

Project Restrictions

- The web development aims at browsers supporting HTML5, which includes newer versions of Chrome, Firefox and Safari and minimum Internet Explorer 9.
- No iOS application is developed as this required additional cost in terms of development license and a Mac computer. This project targets only the Windows and Android platform.
- The Android application development targets minimum API level 16 (Android 4.1) and higher in order to make the development and testing of the Android app easier.
- The system supports English and Norwegian in addition to support for further internationalization of the system.

1.4 Target Audience

There are several target audiences for this project: The project as a whole, the report and the target audience of the final product. The target audience of the report will be those who want to get an insight into how this system was developed. This may be examiner, supervisor, customer, those who will work with the system at ETC, or any future employer. This report is written in English in order to aim towards a more international audience. The project attempts to contribute to both society and our own experience and knowledge by demonstrating the usage and challenges of commonly used tools and frameworks.

1.5 Application Target Audience

The target audience of the final product is broad since the product aims for both the private and small to medium-sized organizations. It is assumed that a contract management system appeals to those of an adult age. This means people under 18 are not considered as the target audience, but they can still use the product. The large target audience has led to an extensive use of universal design principles.

The initial release will be focusing on the Norwegian market even though it is a system requirement to support internationalization and will include the English language package.

1.6 Purpose

It was in early November that all the different bachelor project descriptions were announced on Fronter. We quickly saw that the project description from ETC stood out from the other projects. By having a well-defined description, we felt that we could use and build on most of what we have learned through our three years here at Gjøvik University College (GUC). This project was also a quite concrete project that was not too abstract, which makes it easy to explain to people without any IT background. After a discussion with our supervisor and a short meeting with the customer, it was decided that the project was given to us based on our academic background and the nature of the project.

We had several goals with our project. As mentioned in the title, the project is a modern approach by using modern designs and frameworks. We also wanted to end up with a good product that someone would be likely to use by utilizing tools and frameworks that are frequently used in the software engineering industry, which the customer was open to.

ETC also wanted to end up with a good product that was easy to use and could also be maintained and further extended. This is the reason why the backend languages were limited to PHP and Java (further discussed under 6.1 Restlet).

The name ctrctmgr is short for Contract Manager and comes from having to make a short name for our project website that had to be maximum 8 characters long. This abbreviation became frequently used as it was a shorter and easier way to write Contract Manager. Therefore it became the codename for this project.

1.7 Academic Background

Each of the team members have studied Bachelor of Science in Software Engineering (BPU) full-time for three years. This has provided experience with how software should be planned, and implemented together with experience from some tools and languages. This includes C++, PHP, Java, Android application development and MySQL, in addition to experience with web development in PHP, HTML, CSS and JavaScript with jQuery. Thomas and Joakim have taken mobile development project in addition to web coding as elective courses, which has provided them experience with development of webpages together with Bootstrap. Martin has studied multithreaded programming that focused on different multi-threading frameworks in C and C++. He has also experience with C#, C++CLI, the .NET Framework and the MySQL DBMS (Database Management System) from outside the studies. This report assumes the reader has the same academic background as any student would have from studying IMT (Information Media Technology). Some of the aspects that the reader should be familiar with that is also covered in IMT includes the HTTP-protocol, software engineering patterns, relational databases, the Java programming language, common concepts in web development such as cookies and sessions, HTML, JavaScript and CSS. The reader should also be familiar with the concepts of annotations and reflection in the Java programming language.

This knowledge provides the baseline of understanding how to use the different tools and frameworks in this project. Only a small number of tools in this project is covered in IMT, which are specified under chapter 6. Technologies and chapter 8. QA.

1.8 Roles

Each group member had one certain role in the project. During the project-planning phase, it was decided which roles each member should have. Martin was the group leader. He took responsibility for the team, scheduled meetings, wrote reports and acted as the scrum master by maintaining the task board. He was also responsible for the backend. Thomas was responsible for the user interface of the application such as layouts, colors, fonts and making sure that the layout became user-friendly. Joakim was responsible for the tools and frameworks being used on the web application.

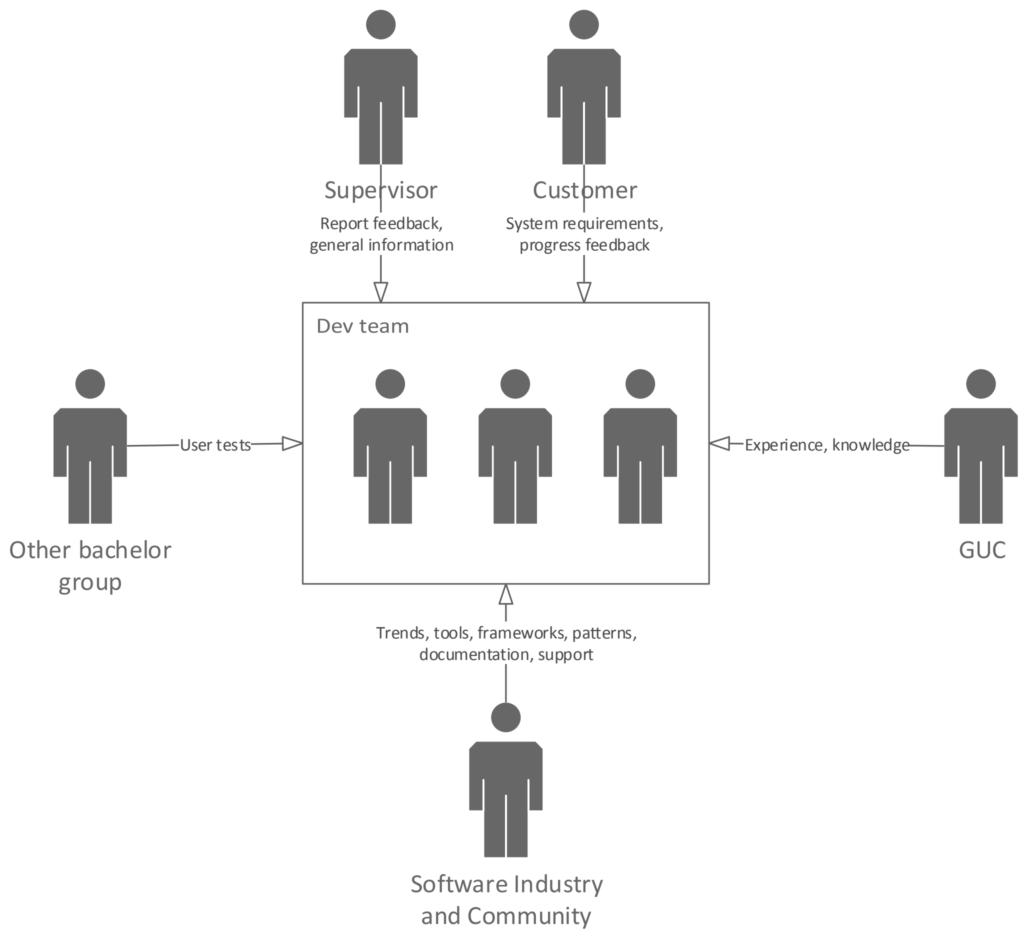

Supervisor
Report feedback, general information
Customer
System requirements, progress feedback
Dev team
Other bachelor group
User tests
Experience, knowledge
GUC
Trends, tools, frameworks, patterns, documentation, support
Software Industry and Community

Figure 1 – How different sources has contributed to the project.

Figure 1 shows how different parts have contributed to the project. Another bachelor group contributed to this project by giving feedback on Contract Manager through user tests. The software industry and community contributed with different frameworks, environments and support through different forums such as Stack Overflow.

1.9 Glossary

Contract Manager (ctrctmgr): Also referred to as the product.

Customer: Employer of the project (Dag L Solhaug). He is also the Product Owner.

User: Someone that uses Contract Manager as either an organization user, home user or system administrator.

Organization user: An organization user is a user who belongs to an organization.

Home user: A home user is someone that uses the product under private circumstances.

Point of entry: The first web page that the user sees when they want to start using Contract Manager. This web page provides information such as product features, support and registration.

Core product: The web page that provides login and all the features regarding contract management.

Web-app: Unless core product or point of entry is mentioned, it means both of them seen as one web site.

API: Unless nothing else is mentioned, it refers to the web-API that stores user-data (for further description see chapter 5. System Architecture).

1.10 Document Structure

This thesis is structured into the following sections:

Introduction: This section gives the reader a basic understanding about what this project is about and who was involved.

Project management: Provides the reader an overview of how the project was managed, such as how software development methodologies were used in this project.

Project planning: How the system was planned in addition to an overview of different system requirements.

System architecture: A description of the overall system architecture while also discussing alternatives.

Technology: Provides the reader an introduction to some of the most central technologies being used in this project; alternatives, how they work and why they were used.

Design: How the system has been implemented with the different technologies being discussed under 6. Technologies.

QA: How quality assurance was performed on different components.

Summary: Summarizes the entire project as a whole.

Each code snippet is embedded in a frame and referred to as a figure. These snippets will follow the conventions, fonts and formatting from their language and the IDE which was used.

2. Project Management

As discussed in the project plan (see appendix A – Project Plan), it was decided to use Scrum as the software methodology as both the team and the customer had good experience with it. Additionally, due to incomplete system requirements, an agile system development methodology became a reasonable choice. The customer had also good experience with one-week sprints as it would lead to a steady work-flow and the team could respond rapidly to any items that may have been misinterpreted. Despite some hints from our supervisor that the sprint length could be too short and would generate too much overhead, it seemed to work well in this project. Some of the factors that made this go well was that the customer was a five-minute walk away from campus and it made the team get used to a weekly routine.

Meetings with Customer

Meetings was scheduled every Tuesday with the customer. During these meetings, new features and functionality developed since last meeting was demonstrated, followed by a sprint retrospective meeting, then a sprint-planning meeting for the next sprint. In the planning meeting, the project leader suggested which items would be appropriate for next sprint, taken from the backlog, where the customer could intervene and change the priorities. The project leader took notes and published a meeting summary that became available on the project website after the meeting. These summaries reflected the outcome and what we discussed during the retrospective meetings and which items that became scheduled for next sprint.

When the customer was absent on a scheduled meeting, the team would start working on new features from the backlog that seemed reasonable to start working on as long as the items in the current sprint was completed.

Meetings with Supervisor

The meetings with the supervisor was scheduled after the meeting with the customer. In these meetings, the status of the project was discussed in addition to subjects related to this project report. If the group had no questions and nothing to discuss with the supervisor, the meeting would be canceled and the next meeting would then follow on next Tuesday.

Task Board

The task board was the center of the project organization. It made it easy for everyone to see the current stage for each item with its tasks. Although many project use online boards, it was decided to use a physical board. This provided motivation as the post-it notes were moved when there was progress to tasks and items. This also made each developer get up from the chair and get their blood circulation going in order to maintain a clear mind. As the project made progress, the team became better at splitting items into smaller tasks. For example, in the beginning of the project, the team members worked with larger items and did not split these into smaller tasks. This made it harder for the team to keep track of what they were doing. After a while, items were more frequently split down into tasks that also made the progress more transparent. This issue has also been reflected in the meeting summaries (see appendix C – Meeting summaries).

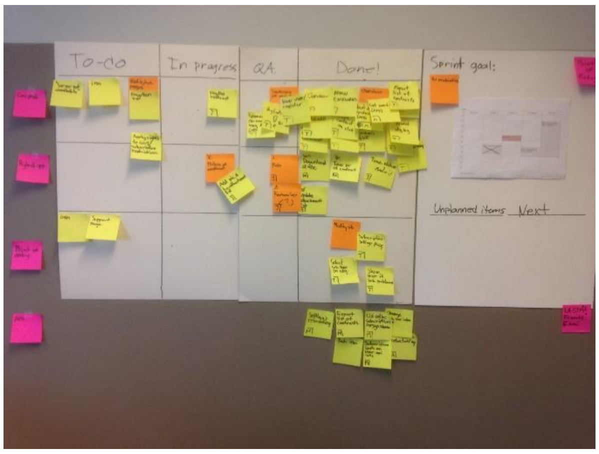


Figure 2 – Photo of the task board. Picture is taken from the last sprint.

Figure 2 shows the task board that was used during the project. It was located in a computer lab on campus with very low activity. Unless this room was used for a lecture, this was the room where the team usually did the development between 10:00 and 18:00. As seen in Figure 2, there were four phases: to-do, in-progress, QA and done. The to-do phase was items that nobody had started working on yet, in-progress was items that were under implementation, QA was a test phase, and done was items that were completed. A further description of the QA phase can be found under chapter 8. QA. Sometimes the items that entered the done had some missing functionality or had some extra work that had to be done. In this case, an extra post-it note was added to the item, describing what had to be done.

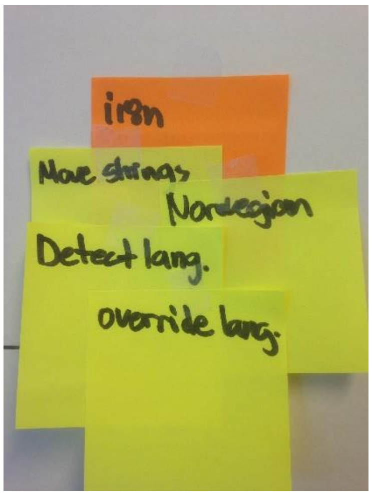


Figure 3 – How an item (orange) is broken down to tasks (yellow).

Also note that the post-it notes on Figure 2 has different colors. The pink notes to the left of the board described which part of the project or module each note belonged to in their row. Orange notes were items, and yellow were tasks (Figure 3). On the right side of the board were the sprint goal, a time schedule for the room, new items that appeared during the sprints that were not planned, and items that were to be added in the next sprint.

After the meeting with the customer and our supervisor that was usually scheduled on a Tuesday, the scrum master would be responsible for removing the old tasks and items, then add the new items. Usually if there were items that were to be completed after the sprint, only the items under the done phase were taken down. New items were added the next day.

3. Contract Lifecycle

One challenge in this project was to generalize contracts to a format that a computer could easily interpret. There is no specific standard for all the contracts, but there are however certain attributes that normally appears on contracts, such as a start and end date, involved parties, a reference number, and so forth.

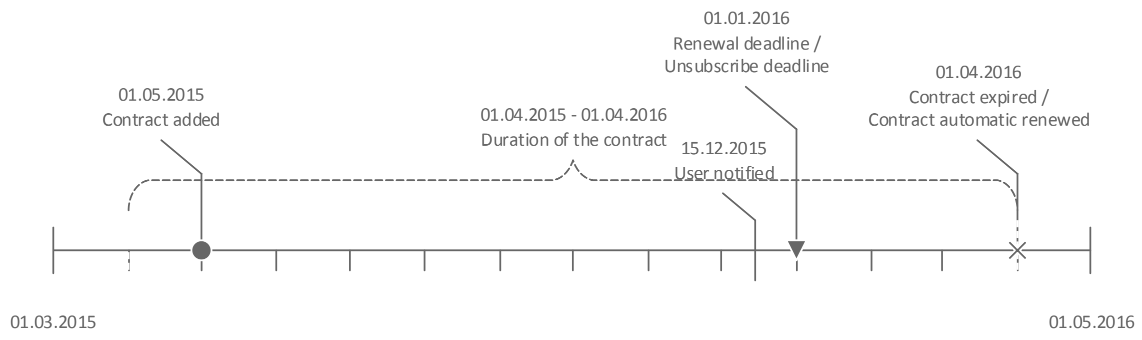


Figure 4 – Illustration of the lifecycle of a recurring contract.

Since the system does not restrict users from entering a contract that has already become active or is about to begin, it also stores the date a contract is added to the system. This is illustrated in Figure 4 where the curly bracket represents the lifespan of the contract.

While the system supports automatic renewing, the outcome of a renewal would depend on the contract type. This system manages two types of contracts: Recurring and date-to-date contracts. A recurring contract will be automatically renewed once the expiration date is reached, while a date-to-date contract would be automatically archived. The system also allows users to specify a deadline before a contract has to be terminated or renewed, depending on the contract type.

Although there are several other variants of contract lifecycles with other important dates, this is the general lifecycle-model which Contract Manager is based on.

4. Project Planning

This chapter will discuss how the system were planned together with the customer in addition to the backlog for the project. How items were taken from the backlog is discussed under chapter 2. Project Management.

After the first meeting with the customer, low-fi prototypes on paper were created, which was basic drafts on how the user-interface would look like. These prototypes were then demonstrated to the customer. This made it possible to get a common understanding of how the product should behave and which features it should have, by using a GUI-down approach. As the project made progress, items were added to a backlog in a spreadsheet available through Google Drive.

4. 5 Conceptual Data Model

After a few weeks of planning and prototyping, we had managed to get a good overall picture of how the system should be created. As discussed in the project plan, we were going to create a conceptual data model of the system. This allowed us to get a deeper understanding of what needs to be implemented and it also acted as a good tool to keep track of what was implemented in the system and what was left to be done. Figure 5 shows the final conceptual model that was used throughout the project.

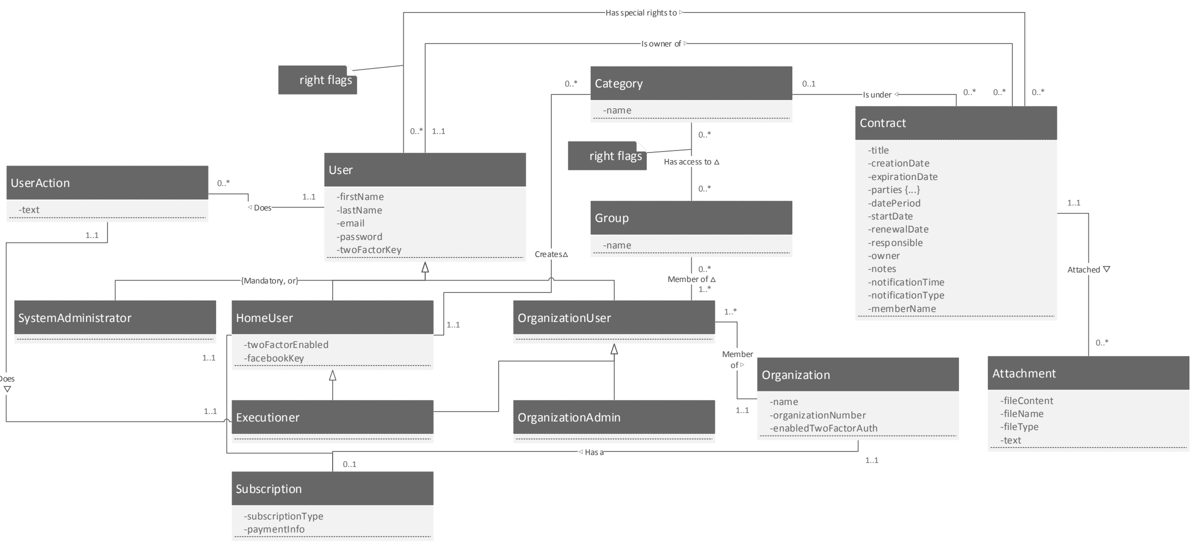


Figure 5 – Conceptual data model of the system.

4.6 User Types

The system supports several different user types. The two main categories are home-users and organization users. We saw that home users and organizations has different needs in terms of contract management; however, there is some overlap between these two user types.

Home User

Home users are users who uses the system on an individual-basis. These users can pay for using the system in order to get access to more features such as custom categories, the ability to add more contracts and so forth. All the users has access to basic functionality such as adding, removing, exporting contracts and so forth. The active subscription of the user may for example determine how many contracts a user can store.

Organization Administrator

The organization administrator has the responsibility for administrating the organization settings in addition to providing members of the organization access to the system. By using this approach, it is possible to create rights and groups. This aspect is discussed later under 4.8 Groups and Rights. The organization administrator is also responsible for the subscription for the organization.

Organization User

The organization user is added to the system by an organization administrator through email. How much functionality this user has access to is determined by which rights the user has been assigned by other users and/or the organization administrator. Limits on features are determined by the active subscription for the organization.

System Administrator

The system admin is a person who is responsible for maintaining the system and is not meant for storing contracts. This user has access to view all the active subscriptions of the system in addition to view and modify system settings.

4.7 Legal

During the beginning of the project, we saw that it would be necessary to make an evaluation whether the system would require special approval from the Norwegian Data Protection Authority, as mentioned in the project plan (see appendix A – Project Plan). However, after contacting them, it was made clear that a terms-of-service where the user accepts the processing of the information they supply to this system would be sufficient (see appendix H – Email Communication with the Norwegian Data Protection Authority (Norwegian)) in addition to following laws regarding data deletion and routines specified by the Norwegian Data Protection Authority.

4.8 Groups and Rights

A lot of time during the preparation stage of the project was used to discuss how groups and rights should be modelled in the system. This system manages rights on two levels: Group-based and individual-level. Groups are targeted at organization users to avoid micromanagement of rights towards different contracts. Both organization users and home users can use individual-level albeit home users can only use this for managing rights to contracts they have chosen to share with other individual home users.

Individual level rights are used to give one specific user rights to a contract. This right determines whether the user has access to read, modify, delete and assign other user rights to the contract.

Group-based rights are meant for only organization users. A user can be a member of a group. This group of users are then given rights to a category. All the rights given to the relationship between the group and the category will then apply to the contracts under that category, as seen in Figure 6.

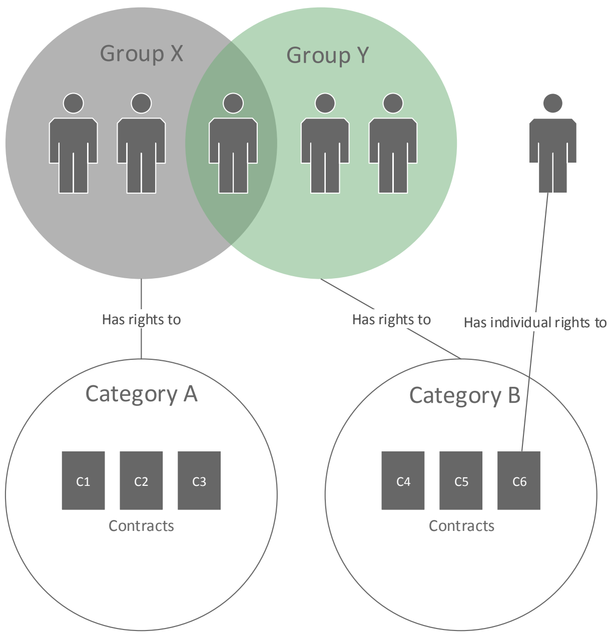


Figure 6 – Shows how an organization can manage its users and assign rights to different contracts.

4.9 Backlog

The customer were also given access to the backlog so that he could keep track of what was added in addition to keep track of what was left of the product. Once items were done, they were highlighted as green in the backlog. Towards the end of the project, items became highlighted as red when the team saw that there was not enough time to implement them. Figure 7 shows a full use-case description of the system with different external actors and use-cases from the product backlog (see appendix D – Backlog). The API is not included as the figure shows the use-cases for different user-types and external systems. Most of the items in the backlog were added in the beginning of the project. It has been stable throughout the project with fewer new system requirements as the project made progress.

4.10 Survey

During the second sprint, a survey was performed to get a picture of how managing contracts were currently done by both organizations and home users. Finding which features that were used most frequently would guide us in the right direction, knowing which features should be focused on. The team interviewed five people where three were representing an organization. The result were contradictory. Some rarely looked up old contracts, while other more frequently, but what most seem to agree on was that they were willing to pay to remove advertisement. The most frequent feature they saw as most important was adding a contract.

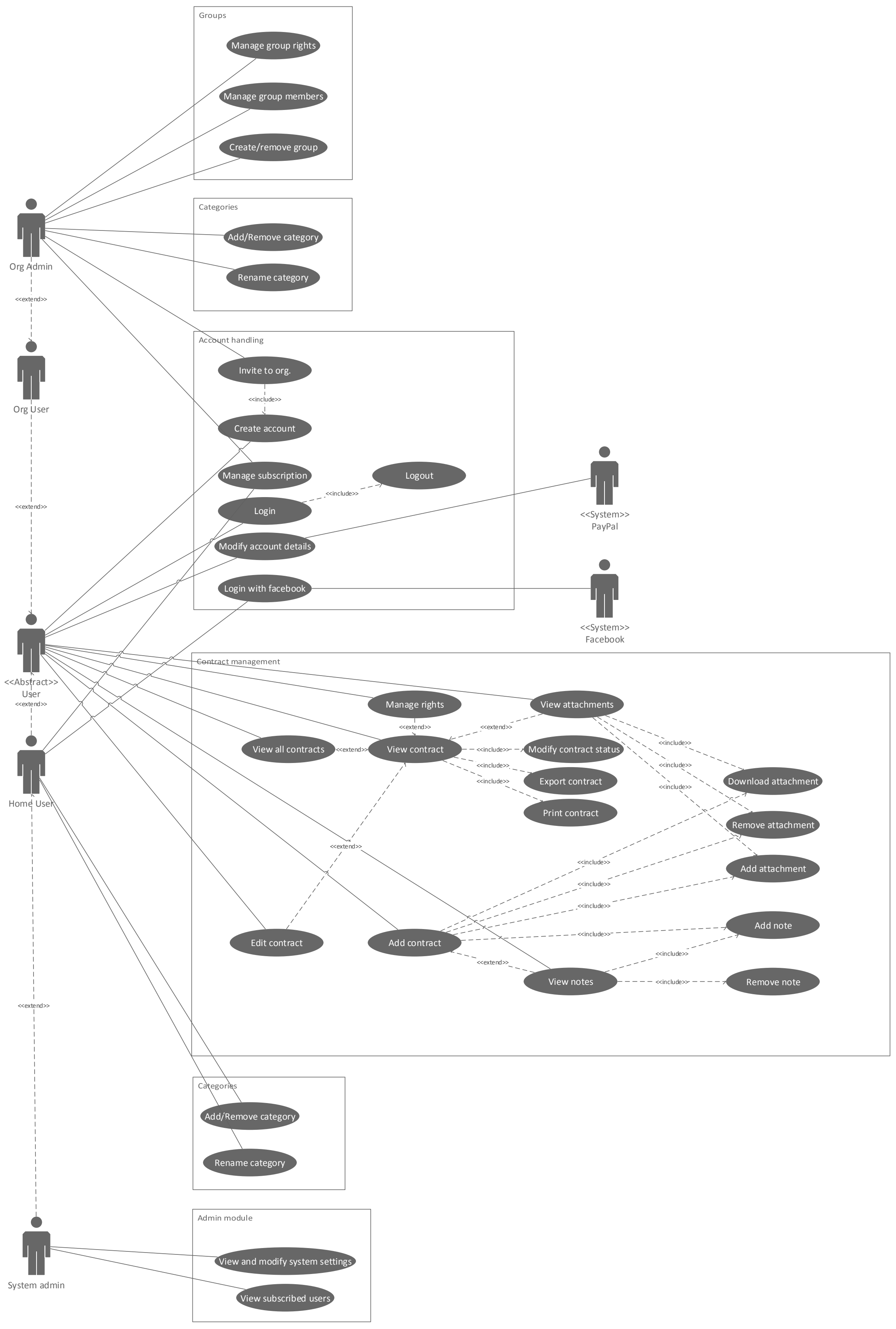


Figure 7 – Use-case diagram of different actors in the system.

5. System Architecture

Contract Manager consists of the following components: The API, the web app, the 2 hybrid apps and the database. The two hybrid apps both relies on this web app by embedding the web app in a web-view. The API (also referred to as backend) is a REST-full (Representational state transfer) web service that manages and stores user data. REST-full means a service following REST principles. The API also creates a common interface for both the hybrid apps and the web app so that several API-calls can be re-used in the hybrid-apps and the web app. The relation between these different components is illustrated in Figure 8.

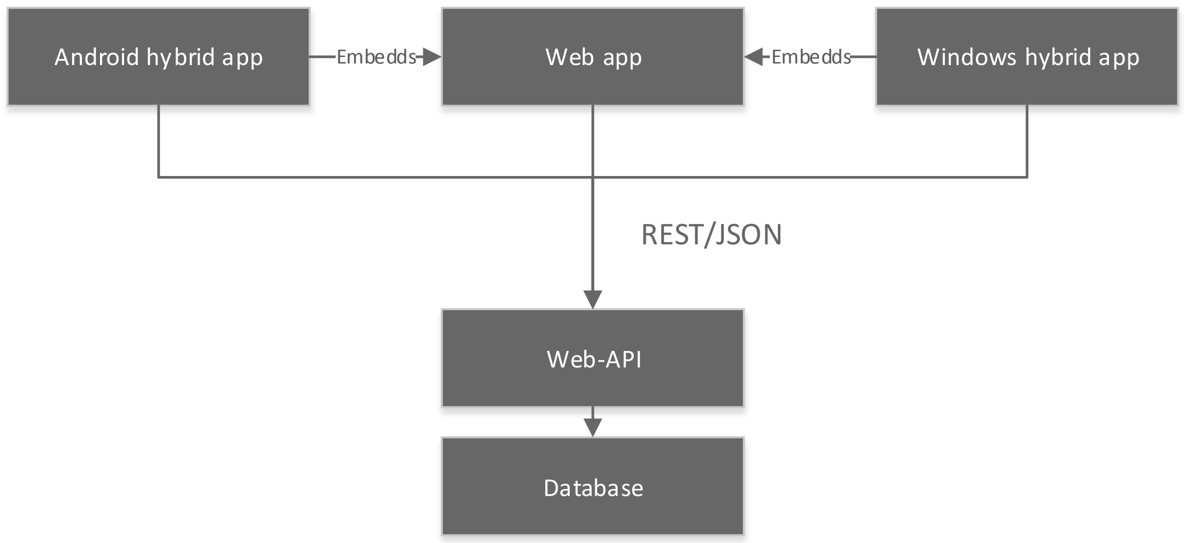


Figure 8 – A brief overview of the different components that makes up Contract Manager.

It was from early on in the project that it became clear it would be necessary to re-use code and logic in the hybrid apps from the web app. While some calls could be directly re-used by calling the API calls in the web apps from the hybrid app, some calls would still have to be coded in the native part such as uploading a photo of a contract and session handling. This is due to the fact much functionality is not available through APIs in JavaScript for a web-application. This is further discussed under 7.6 Hybrid Apps.

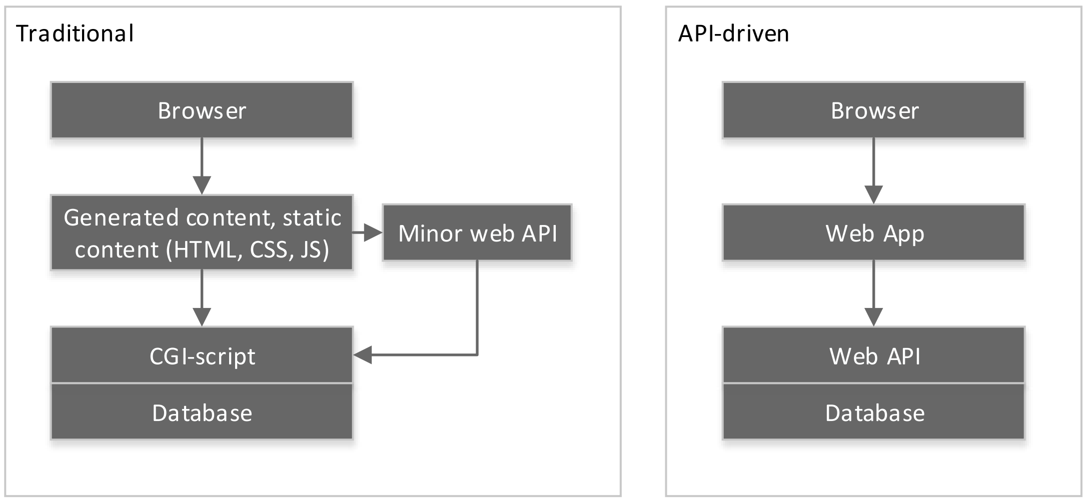


Figure 9 – Two different architectural patterns for web applications.

Figure 9 shows two different architectures that were discussed when developing Contract Manager. The pattern on the left is a traditional web-application [1] [2] where the HTML code is manipulated on the server-side before being sent to the client using a CGI-script (Common Gateway Interface) such as PHP, ASP.NET or Java. The client could be seen as a thin client as it would only view HTML code and do minimal modifications with JavaScript. In some cases, the JavaScript would rely on some API calls (often the case if the page uses AJAX) when the website needs to make modifications on the client-side. A system following this pattern would be a more monolithic system as all the components are tightly coupled in one system. A side effect of this is that generalists could easier work with the system. On the other hand, it makes it harder to split up tasks as the system exists as a whole and it gets harder to work on each component separately. The system would also become harder to scale due to the amount of different components a developer has to relate to when working on the system.

The pattern to the right in Figure 9 is an API-driven (or API-centric as [3] calls it) web application where the web app will communicate with an API, then put together the HTML document on the client-side. In this case, the client could be seen as a thick client that may rely on a JavaScript framework in order to modify and view the web page. It is important to note that the client does not have to be a web app; it could be a mobile application like the mobile apps in this project or a different system that has integration towards the API. This approach follows more service-oriented architecture (SOA) [4] and microservice architecture [5] as each system can be developed, scaled and deployed independently. This leads to a weaker coupling between each component and makes it easier for experts to work on the project and easier to split up tasks when implementing new features. In Contract Manager, the web-app is a single-page application (SPA) [6] built with an MVC-framework (Model View Controller). SPA makes it possible to save bandwidth on mobile devices, as the entire webpage is not re-loaded when the user navigates throughout the system. This also saves battery resources since only parts of the website is reloaded.

We chose an API-driven architecture as it would allow us to reuse several calls in the web application and the hybrid apps. It would also make the system more extensible as other systems could integrate this system with little to no modifications on the API. By having a static web-app, modifications by a CGI-script is eliminated, thus making it easier to spread the web-app on a content delivery network and doing maintenance work since the developer only has to relate to the API specifications and the SPA framework. From an object-oriented perspective, the web app gets a lower coupling to the back-end as the server code is abstract through what can be considered as a standardized protocol, in this case HTTP with REST. REST was chosen as it is commonly used for web APIs and was a good fit for our case [7].

REST is an architectural style, mostly used in web APIs [8] [9]. REST relies heavily on the HTTP 1.1 protocol, as REST was developed in parallel with the HTTP 1.1 protocol. Having different resources exposed on the web through HTTP does however not mean it is RESTful as the resources would have to be stateless, which is discussed later on. The idea in REST is that the server has different resources that are available from an URI (Unique resource identifier). A call to a RESTful API would consist of an HTTP verb and the URI of the resource in addition to additional HTTP headers, for example authentication tokens and optional form data. Another important aspect of REST is that the HTTP verb in addition to the URI should be as self-documented as possible. The call bellow (Figure 10) is a simplified HTTP call to get the preferred language for a user in Contract Manager. A further discussion regarding this call can be found under 7.5 Internationalization.

```
Request: GET http://localhost:8080/ContractManagerAPI/languages/preferred
Response: HTTP 200: OK
Response body:
{
    "language": "en-us",
    "name": "English (US)"
}
```

Figure 10 – The http call for getting the preferred language and the response.

The HTTP response code will indicate whether the call was successful and if there was a response body or not. In REST, this is described as the state of the representation of a resource. In this case, the response is a JSON object indicating the preferred language for the user. The response is defined as a representation of a resource. It is important to note that REST does not explicitly say that the response objects should be JSON; it could be HTML, XML or even binary data depending on the HTTP content-type header.

Another architectural constraint in REST is that the server should be stateless. This means that the server should not store any state about the user directly or care about the state of the user-interface of the client. In other words, the call to the API should contain all the information required to obtain the state of the client. Information about different session data does then have to be stored in a database in order to comply with the stateless constraint in REST. A further discussion regarding sessions can be found in 7.2 Implementation of Restlet and API. One of the advantages of having a stateless server is that it is easy to scale by adding another instance of the server together with a load balancer between the instances.

REST provides a standardized way for client-server communication, which also makes the system extensible and scalable. In Contract Manager, REST allows a standardized way of communication between different components (Figure 8) and facilitates re-use, scalability and flexibility.

6. Technologies

This chapter will discuss some of the technologies being used in Contract Manager. The reader will get insight into what the different technologies are, a short overview of how they work, alternatives, and why they are used in this project. Later on, the reader will get insights into how each technology is used under chapter 7. Design. Figure 11 shows all the different technologies that will be discussed later and where in the system they have been applied. Note that it does not cover every single framework/technology being used. The database does not have any specific technology attached, as this is abstract through Hibernate.

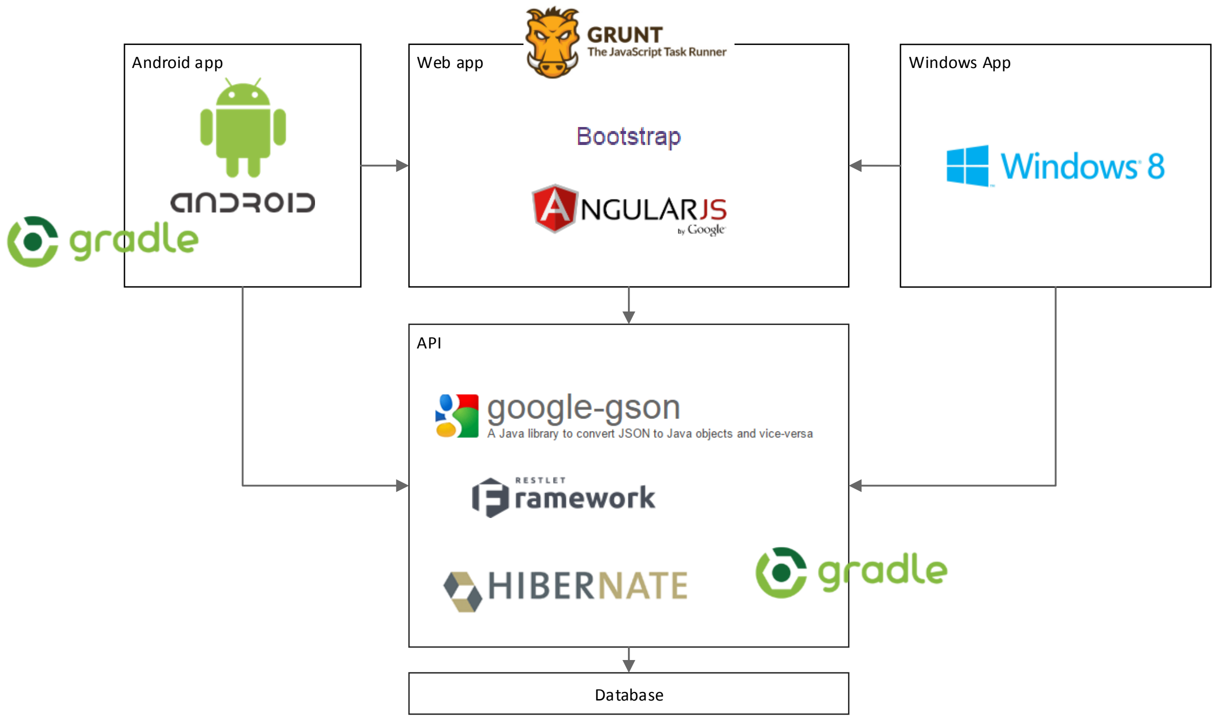


Figure 11 – Overview of where the different technologies that will be discussed in this chapter is applied in Contract Manager.

6.1 Restlet

During the project-planning phase, the customer initially wanted a system in PHP where Java was a second alternative. It was decided to use Java due to the extent of the system in addition to the different tasks and tools the backend would rely on. As the customer wanted support for OCR, there was better libraries for Java based on the documentation and reputation of the different libraries. Another key factor was that the backend was going to be a RESTful API where Java could provide better scalability, design and tools compared to PHP. However, as features such as OCR and encryption did not get implemented due to

prioritizations, the system could have been built with PHP. On the other hand, the final code would not have the same design that would allow as easy integration with these features.
It was decided to implement the API with Restlet as it provided good documentation and is a framework that is easy to understand. Restlet is a lightweight, open source framework for developing RESTful APIs in Java [10]. The framework uses the same terminology on its components as what REST does. It mainly consists of four central components: Router, representation, resources and connectors. The connector specifies which protocol the application should receive calls on, for example HTTP. However, if the application is deployed as a servlet container, no connector is required. A router is the component that redirects a call to a resource, if found. The resource will handle the call based on the HTTP verb for the call and return a representation, for example a string. Each verb is annotated above the method that will handle the call. The code bellow (Figure 12) is an example of how a representation of the preferred language is returned to the client. A demonstration of the response form this call can be found under chapter 5. System Architecture.

```
@Get
public GsonRepresentation<Locale> getLocale() {
    Series<Header> headers = HeaderHelper.getRequestHeaders(getRequest());
    String languageString = headers.getValues("accept-language");
    Locale locale = LocaleFactory.getPreferredLocale(languageString);

    return RepresentationFactory.makeRepresentation(locale,
                LocaleRepresentation.class);
}
```

Figure 12 – Code snippet for getting the preferred language for a user. The code is from the class PreferredLocaleResource.java.

Each resource has to extend the `ServerResource` class in order to be registered in the router together with its URI. The annotation `@Get` above `getLocale()` implies that this function should handle GET requests. One important aspect of Restlet is GSON integration. GSON is a JSON (JavaScript Object Notation) library developed by Google for serializing and de-serializing JSON objects in Java [11]. It relies on reflection and annotations to map fields between classes in Java and the fields in the JSON object. In Contract Manager, GSON makes it possible to avoid having to create one class for each representation. This leads to less code duplication.

6.2 Hibernate

Hibernate ORM *(Also referred to as just Hibernate)* is an open source ORM (Object relationship mapping) framework for Java applications, available under the GNU Lesser General Public license [12]. An ORM framework provides database access through a higher level of abstraction compared to writing traditional SQL queries that would fetch entries from a database. It achieves this by mapping classes in the code towards tables in a relational database. Without using ORM, parser code (also known to as a mapper) is required to map the structure of a class towards its structure in a database. These classes have SQL code and maps different fields in a class towards fields in a database. No SQL code is required when using Hibernate as the framework would generate the required SQL code to fetch, modify and delete entries in a database. A result of this is that the application using Hibernate can easily move to a different DBMS (Database management systems) as if used correctly, the only modification required when switching between DBMS is changing the database driver (also referred to as the database connector). Hibernate does however have a higher level SQL language named Hibernate Query Language (HQL) that makes it possible to write DBMS independent SQL queries.

In this project, Hibernate made it possible to simplify code by eliminating mapper classes and reducing the time required to create tables in the database. As of less code, the code also becomes more maintainable. Hibernate also facilitates re-use of code and classes. As Hibernate is frequently used in Java and Java EE applications, a large amount of documentation is available in addition to support through different communities on the Internet. We also have experience with Hibernate from a system made in IMT3281 – Software Development. An alternative to Hibernate was using JDBC (Java Database Connector) directly, thus removing the layers of abstraction provided by Hibernate, as it is not an ORM framework. Using JDBC would lead to extra code and development costs compared to Hibernate, which was the main reason why Hibernate was selected instead of JDBC. One disadvantage of using Hibernate is that it has far longer startup time compared to JDBC. However, in this project it does not have any significant impact, as it is an operation that only would have to be executed on the first page-load.

Hibernate can map classes in two ways: Either through an XML document or annotations in the code. It relies on reflection in order to map the data towards classes. It was decided to use annotations, as metadata about a class and its members is natural to store in annotations. Regardless of using annotations or XML mapping, each class that Hibernate would map has to be listed in the hibernate.cfg.xml file, which is the configuration file for Hibernate. This XML document contains information about which database driver to use, database name, username, password, hostname, caching and logging settings and so forth. It is also worth mentioning that Hibernate can efficiently optimize and cache queries and the query result because of the high level of abstraction.

6.3 Angular

AngularJS is a client side web application framework based on JavaScript. Angular is developed and maintained by Google and is an SPA designed for CRUD (Create, Read, Update, Delete) applications [13]. It offers a wide set of tools and is described by its creators as the way HTML would have been if it was designed for creating web applications. Angular makes it possible to create reusable HTML templates which serves as the view and can dynamically bind the data retrieved by a controller.

While researching which approach we that should be used on web application, several different JavaScript SPA frameworks was considered, evaluating the tools they provided, documentation, popularity and general developer feedback.

Metric	AngularJS	Backbone.js	Ember.js
Stars on Github	27.2k	18.8k	11k
Third-Party Modules	800 ngmodules	236 backplugs	21 emberaddons
StackOverflow Questions	49.5k	15.9k	11.2k
YouTube Results	~75k	~16k	~6k
GitHub Contributors	928	230	393
Chrome Extension Users	150k	7k	38.3k

Table 1 – Statistics of usage as of August 16 2014 [14].

The range of frameworks was limited to consist of KnockoutJS, EmberJS, BackboneJS and AngularJS, which are all available under the MIT licence [15], which permits usage in proprietary software.

Angular and Ember seemed to offer a lot of the same tools and are both using the MVC (Model View Controller) pattern, whereas KnockoutJS and BackboneJS is a lot more lightweight.

Angular was chosen instead of Ember because of the strength of the Angular community and amount of shared content on the web as shown in Table 1, as well as Angular focuses on CRUD as Contract manager is a typical CRUD application. There was performed a small pilot project using Angular to develop the bachelor project website, which was a positive experience.
Using Angular made it possible to create modularized code that was easy to reuse and modify for each individual use case. It also had a wide support for external tools like automated testing and external code packages that was easy to use right out of the box.

Core Components

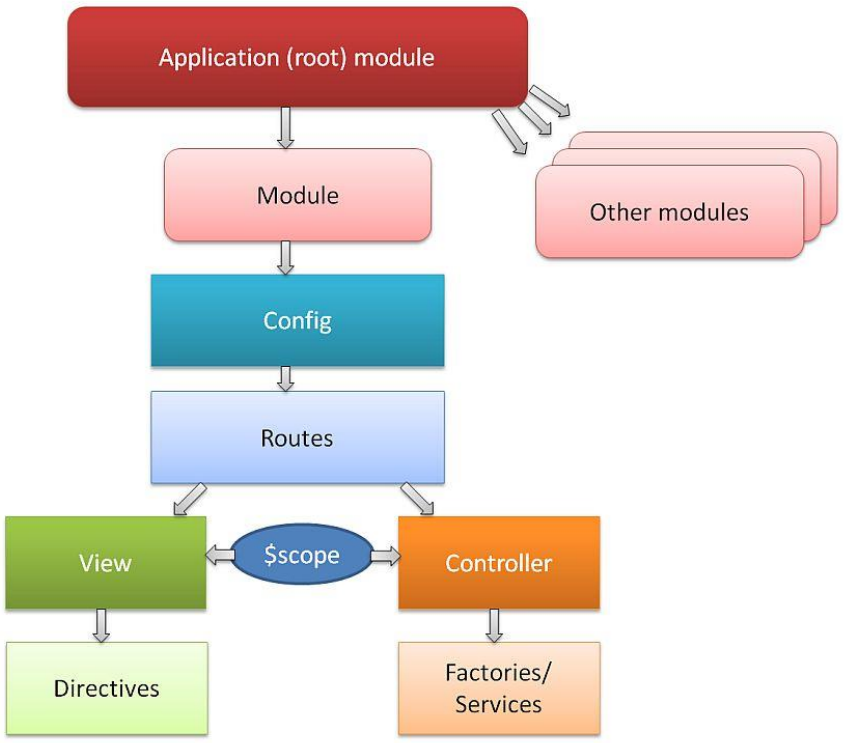


Figure 13 – Angular architecture [14].

An Angular app consist of a root module with an arbitrary amount of child modules as shown in Figure 13. Some modules may be simple and not contain any configuration and routing options besides what is provided from the root module. Each module usually have one or more controllers, which is used to bind data and handle user interaction in the view. A controller will be assigned to a view, creating a scope. The scope can be thought of as an area of code in a view, available for the assigned controller to administrate.

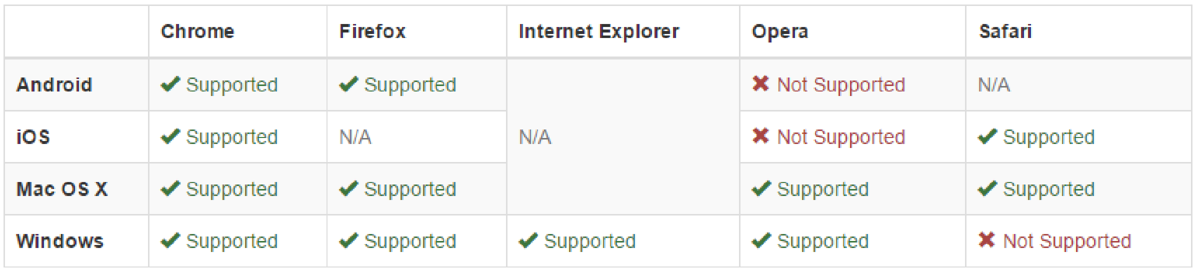
Directives are more or less customized HTML tags that can be inserted straight into a module view. A directive can be thought of as an isolated component. Directives can have their own controller, view and scope, but are also able to inherit the parent scope. Directives are easy to reuse and can be thought of as a standalone web component. This results in reuse of HTML and JavaScript code.
Factories contain business logic and utilizes Services to communicate with a web-server. Module controllers use these factories to retrieve data to insert it into the scope.

6.4 Bootstrap

Bootstrap is an open source HTML, CSS and JavaScript framework for developing responsive web pages. It is released under the MIT license and is according to their website, the most popular framework in their category [16]. The reason why Bootstrap was chosen was because we already had experience with it and knew it had a good reputation in the web development community.
Aside from creating a website responsive, the framework also provides a set of styled components such as dropdown lists, input fields and alert boxes. Bootstrap provides over 250 free icons which helps scaling more easily as button labels can be replaced with icons and still convey the same meaning. While having over 250 icons to choose from, this made it more difficult finding the suitable icons conveying the right context. Bootstrap also did not have icons that represent general concepts such as groups. We ended up using the open eye icon which originally was used to convey rights, which was then changed to a lock. We could have created our own icon, but valued more being consistent and only using Bootstrap icons.

Browser Support

Bootstrap version 3 supports the latest browsers and platforms shown in Figure 14. It also supports Internet Explorer 8 and newer [16].



	Chrome	Firefox	Internet Explorer	Opera	Safari
Android	✓ Supported	✓ Supported		✗ Not Supported	N/A
iOS	✓ Supported	N/A	N/A	✗ Not Supported	✓ Supported
Mac OS X	✓ Supported	✓ Supported		✓ Supported	✓ Supported
Windows	✓ Supported	✓ Supported	✓ Supported	✓ Supported	✗ Not Supported

Figure 14 – Showing latest browser support for the Bootstrap framework [16].

Scaling With Bootstrap

Bootstrap performs scaling by using a grid system. This is a method of dividing an HTML-element into rows and columns which dynamically changes depending on the screens width. One starts off by using either the `container` or the `container-fluid` class. The difference being the latter fills the whole element width. Within these elements one can define rows with the `row` class and their child elements with different `col` classes. Bootstrap has defined four different width boundaries where changes will takes place. These boundaries are displayed in detail in Figure 15.

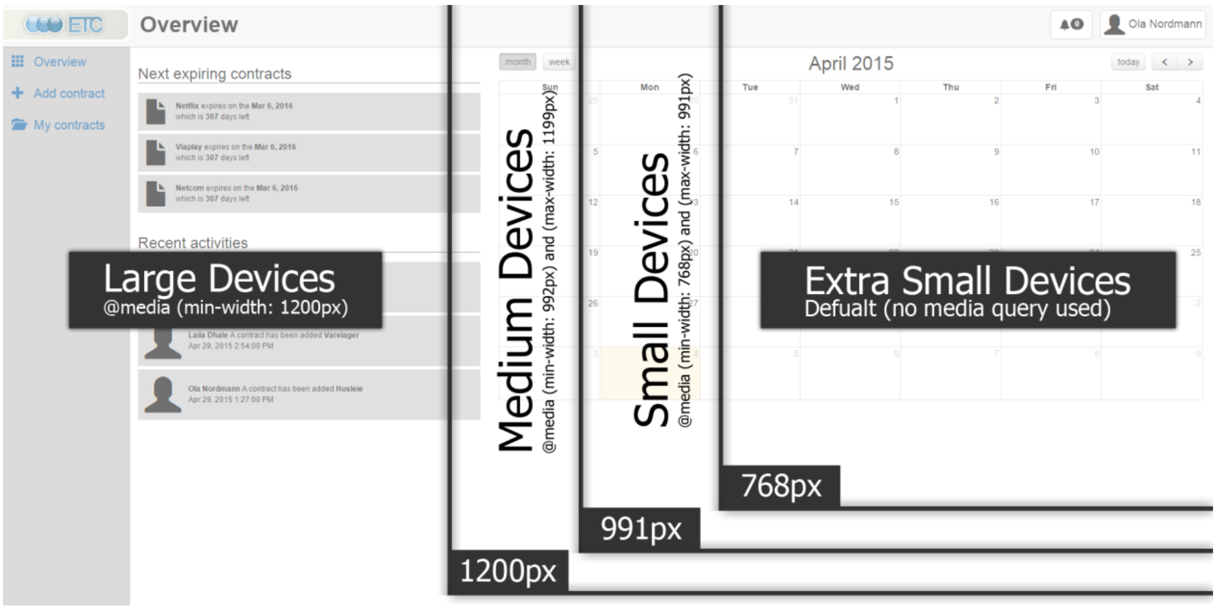
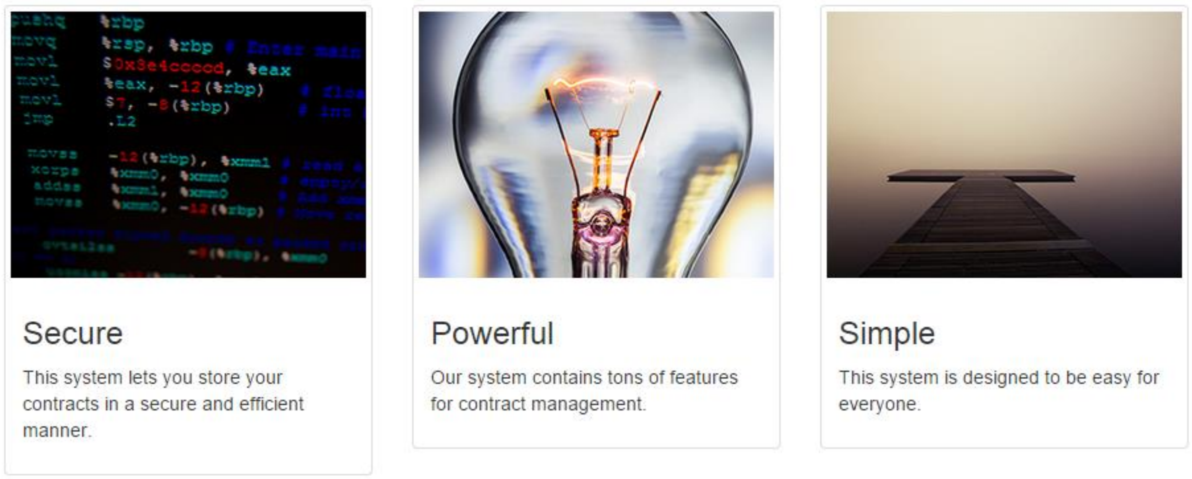

Large Devices
@media (min-width: 1200px)
Medium Devices
@media (min-width: 992px) and (max-width: 1199px)
Small Devices
@media (min-width: 768px) and (max-width: 991px)
Extra Small Devices
Defualt (no media query used)
1200px
991px
768px

Figure 15 – Showing defined boundaries from Bootstrap with corresponding media query.

A column element has 12 different options for how much space it should take up in each of the defined devices. Figure 16 shows how this is done in HTML and CSS. It has one row with three columns representing different aspects of the core product. A column class starts with the `col` keyword followed by which device it represents and ends with a number between 1 and 12. If the sum of all similar `col` classes exceeds 12 within one row, the row starts to wrap columns to a new row. It can be seen in the code example that on small (`sm`), medium (`md`), and large (`lg`) devices all have 4 in size. This adds up to 12, meaning it fits perfectly within the row. On extra small devices (`xs`) however, each column gets all the width, making each column appear as rows. The corresponding layout is displayed in Figure 17.

```
<div class="row">
  <!-- Information / advertisement -->
  <div class="col-xs-12 col-sm-4 col-md-4 col-lg-4">
  <div class="col-xs-12 col-sm-4 col-md-4 col-lg-4">
  <div class="col-xs-12 col-sm-4 col-md-4 col-lg-4">
</div>
```

Figure 16 – A simplified code snippet from point of entry that shows how Bootstrap use column classes in order to scale on different devices.



Secure

This system lets you store your contracts in a secure and efficient manner.

Powerful

Our system contains tons of features for contract management.

Simple

This system is designed to be easy for everyone.

Figure 17 – Corresponds with the code in Figure 16. Example is from point of entry. The text in each box is a subjective message to a customer and is not relevant in this text.

Colors and Fonts

Since Bootstrap provides both color codes and font selection, this has not been one of our main priorities. This is also something that is easily changeable due to the framework. There are also plugins available for adding different themes to Bootstrap, one of them called Bootswatch. As the current version of Bootstrap is not colorblind friendly, adding a customized theme should be implemented before a release, even though a revision of their color scheme is happening in the next bootstrap version [17]. On another note, as users have different preferences, providing the option for changing the theme would satisfy more users, like adding a dark theme to the core product, making it an option in the settings menu.

Supporting Screen Readers

Blind people need assistance reading content and helping them navigate around in an application such as using screen readers. By designing for screen readers, these users can become more independent. These applications read HTML and provides functionalities for skimming through content from paragraph to paragraph. One way of navigation through a

page is by pressing the tab key and the screen reader tells the user what each link is. This however depends on how the HTML is put together [18].
Bootstrap provides classes for screen readers to add to the HTML, which is used in this project. This means that each input field has an associated label which describes what information the input expects. This is done by adding the `sr-only` class to the label which hides the label. In cases where the label should be visible, the `sr-only-focused` class should be used instead. Header elements is also important as they give the users the ability to separate content more effortlessly. This is also an HTML element that has been used.
This was further tested using ChromVox, which is a screen reader extension for Chrome. By pressing tab, we get voice feedback for the current link that the user can interact with. Before stating that the core product fully supports screen readers more user tests must be completed.

6.5 Android and Windows

It was specified in the project description provided by the customer that the system should have multiplatform app support. As mentioned in the project plan (See appendix A – Project Plan), it was decided not to develop for the iOS platform, as this would involve additional hardware and developer licenses. The project would then involve a portable Windows and an Android app. It was initially planned to create two native apps, however, due to prioritizations, a hybrid application had to be developed instead.
After deciding to use hybrid apps, some cross-platform compilation tools were considered. Xamarin [19] is a framework where an app can be written in C# and be compiled to a variety of platforms. However, this was not chosen since it is not free. Cordova [20] was another framework that was considered. It allowed us to take the web app and easily generate projects that was ready to deploy on a range of mobile platforms. After discussing with the customer regarding Cordova, he seemed critical to using it, as it would introduce more work when changes were made to the web application. Therefore, it was decided to create a hybrid application for Android and Windows using the native tools available on each platform.
There was initially two features requested from the customer that lead to making the mobile applications: Push notifications about expiring contracts, calendar integration and being able to take a picture of a contract. Without these features, the app could just be a web app that

the user could pin to the home screen of their device. There is currently an API for camera interaction through newer HTML5 standards [21]. However, this API is meant as a replacement for web-cameras and is not created for capturing an image that is uploaded to a website. HTML-5 push notifications is still under development where W3C has only published a draft of the push API recommendation [22].

6.6 Gradle

Gradle is a project automation tool [23] that is used for building the project and adding external libraries. Instead of making each developer download each dependency for the project manually, the project automation tool would do this automatically. The team has experience with Gradle from IMT3662 – Mobile Development Theory and IMT3672 – Mobile Development Project, as Gradle is the default project automation tool when developing Android apps with Android Studio. In this project, Gradle was used both for the API and for the Android application. The Windows app did not use Gradle, but a similar tool called NuGet, which is a package manager for .NET applications [24].

One alternative to using Gradle was Maven, which is perhaps the most commonly used project automation tool with Java. Gradle has the build automation script in addition to the dependencies in a JSON-like file named build.gradle. Maven however has an xml-file named pom.xml. One of the advantages of using Gradle is that it allows using dependencies from Maven repositories. Maven repositories can for example contain libraries that can be used in the project, for example Restlet. We decided to use Gradle for both the Android and the API as we have good experience with it, it integrates well with Android Studio and we are consistent in terms of which tools are used where. Below (Figure 18) is a stripped down example from `build.gradle` for adding Restlet as a dependency to the API.

```
dependencies {
    compile 'org.restlet.jee:org.restlet:2.3.0'
    compile 'org.restlet.jee:org.restlet.ext.jackson:2.3.0'
    compile 'org.restlet.jee:org.restlet.ext.servlet:2.3.0'
    compile 'org.restlet.jee:org.restlet.ext.gson:2.3.0'
    compile 'org.restlet.jee:org.restlet.ext.fileupload:2.3.0'
}
```

Figure 18 – Excerpt from `build.gradle` declaring dependencies for Restlet.

6.7 Grunt and Bower

Grunt

Grunt is a JavaScript task runner, which provides an ecosystem of plugins that is related to web application development such as testing, deployment and code inspection [25]. Grunt can be configured to automate processes that would normally take up a lot of time such as testing procedures and preparing the application for deployment. Grunt provides a command line interface where the processes can be executed. An example would be running the local web server plugin being used by typing “grunt serve” in the command line interface. This would set up a local development environment to host the web application.

```
connect: {
  options: {
    port: 9000,
    hostname: 'localhost',
    livereload: 35730
```

Figure 19 – Local web-development server configuration. Simplified snippet from gruntfile.js

Configurations for the local HTTP server can be found within the gruntfile (Figure 19), along with configuration for any other Grunt plugins.

All the installed Grunt plugins for a project are added to the file “package.json” on installation, which contains the name of the plugin along with their version number. This makes it possible for other developers to install these tools automatically with the required version upon project setup on their local pc with a single NPM (Node Package Manager) command [26]. NPM is also the tool being used for installing Grunt and Bower.

Bower

Bower is used for handling installation and management of front-end frameworks, libraries and assets. Like Grunt, Bower registers all installed packages that are registered as dependencies in a file. This file is called “bower.json”.

To install a new package e.g. “jQuery”, the command line “bower install jquery” has to be typed in the command line interface. Bower will search its library for the jQuery package, prompt the user for the preferred version and then download it into the folder “Bower components” within the project.

Bower in combination with Grunt offers many of the same services as Gradle (See 6.6 Gradle).

6.8 Project Dependencies

This section lists all the frameworks and plugins that has been used in different parts of the project. Plugins/frameworks that has a N/A license means that there was not specified any license together with the source code or on the information website for the plugin/framework.

The different licenses has not had any impact on the project as each library is only linked to and not modified. There are however, licenses such as the GNU LGPL (Lesser General Public License) which only allows dynamic linking [27]. This means code and libraries under the GNU LGPL license cannot be used in commercially systems if the code is compiled together with the system.

Web app

Name	Version	License	Description
Angular	1.3.15	MIT	SPA Framework
json3	3.3.2	MIT	JSON utility
Bootstrap	3.3.2	MIT	CSS and JavaScript framework for responsive webpages
angular-animate	1.3.11	MIT	Animation support with Angular
angular-resource	1.3.11	MIT	API integration
angular-route	1.3.11	MIT	Routing support between webpages
angular-webstorage	0.11.0	MIT	Session storage (alternative for cookies)
ng-facebook	0.1.6	MIT	Facebook integration
angular-wizard	0.4.2	MIT	Library for creating a wizard in Angular
ng-flow	2.6.0	MIT	File uploading framework
angular-bootstrap	0.12.0	MIT	Bootstrap integration within Angular
Oclazyload	0.5.2	MIT	Lazy loading of controllers and modules
angular-feedback	0.9.4	MIT	Automatically closing popup
ngBootbox	0.0.4	N/A	Message dialog/popup library
angular-ui-calendar	0.8.1	MIT	Calendar plugin

Table 2 – Licenses for different libraries used in the web-app.

API

Name	Version	License	Description
junit	4.12	Eclipse Public License 1.0	JUnit testing framework
com.lowagie:itext	4.2.1	GNU GLPL 3.0	PDF framework
org.hibernate:hibernate-core	4.3.8	GNU GLPL 2.1	Hibernate ORM

org.restlet.jee:org.restlet	2.3.0	Apache 2.0	Rest framework
org.restlet.jee:org.restlet.ext.servlet	2.3.0	Apache 2.0	Restlet servlet integration
org.restlet.jee:org.restlet.ext.gson	2.3.0	Apache 2.0	Restlet GSON integration
org.restlet.jee:org.restlet.ext.fileupload	2.3.0	Apache 2.0	File upload support with Restlet
mysql:mysql-connector-java	5.1.34	GNU GLPL 2	MySQL database connector
org.hibernate:hibernate-hikaricp	4.3.8	Apache 2.0	Database connection pooling framework
org.mindrot:jbcrypt	0.3m	ISC/BSD	Blowfish crypt
org.apache.commons:commons-email	1.3.3	Apache 2.0	Email support
javax.transaction:jta	1.1	N/A	Java transaction support (required by Hibernate)
org.apache.poi:poi	3.11	Apache 2.0	Spreadsheet generation support
org.apache.poi:poi-ooxml	3.11	Apache 2.0	Support for newer Office formats (eg. XLSX)
com.paypal.sdk:rest-api-sdk	0.5.1	N/A	PayPal integration
org.slf4j:slf4j-simple	1.6.6	MIT	Logging framework (required by HikariCP)
org.apache.tomcat:tomcat-catalina	7.0.41	Apache 2.0	Embedded Tomcat support
org.apache.tomcat:tomcat-util	8.0.21	Apache 2.0	Embedded Tomcat support
org.apache.tomcat.embed:tomcat-embed-core	8.0.21	Apache 2.0	Embedded Tomcat support

Table 3 – Licenses for different libraries used in the API.

Name	**Version**	**License**	**Description**
Newtonsoft.JSON	6.0.8	MIT	JSON library for C#
Facebook	6.8.0	Apache 2.0	Facebook C# SDK

Table 4 – Licenses for different libraries used in the Windows app.

7. Design

This chapter will discuss how each component has been implemented using the technologies discussed earlier. The reader will first become familiar with the file-structure of the source code, which shows where each file that is referred to is highlighted in bold. In these figures, files are dark blue, while folders are grey, ending with a dash. Only the structure of the API and web-app is illustrated with these figures.

7.1 Angular

We made a decision to split up the web application into two parts. These two parts were codenamed *point of entry* and *core product* where point of entry handles everything concerning product support, registration and being the products face towards potential customers. It also provides relevant information about the product. It seemed logical to do this split as they had different purposes. Point of entry was aimed towards potential customers, and the core product was for existing users.

Angular is used in both the point of entry and the core product app, although their structure is different due to the difference in size and scope.

Point of Entry

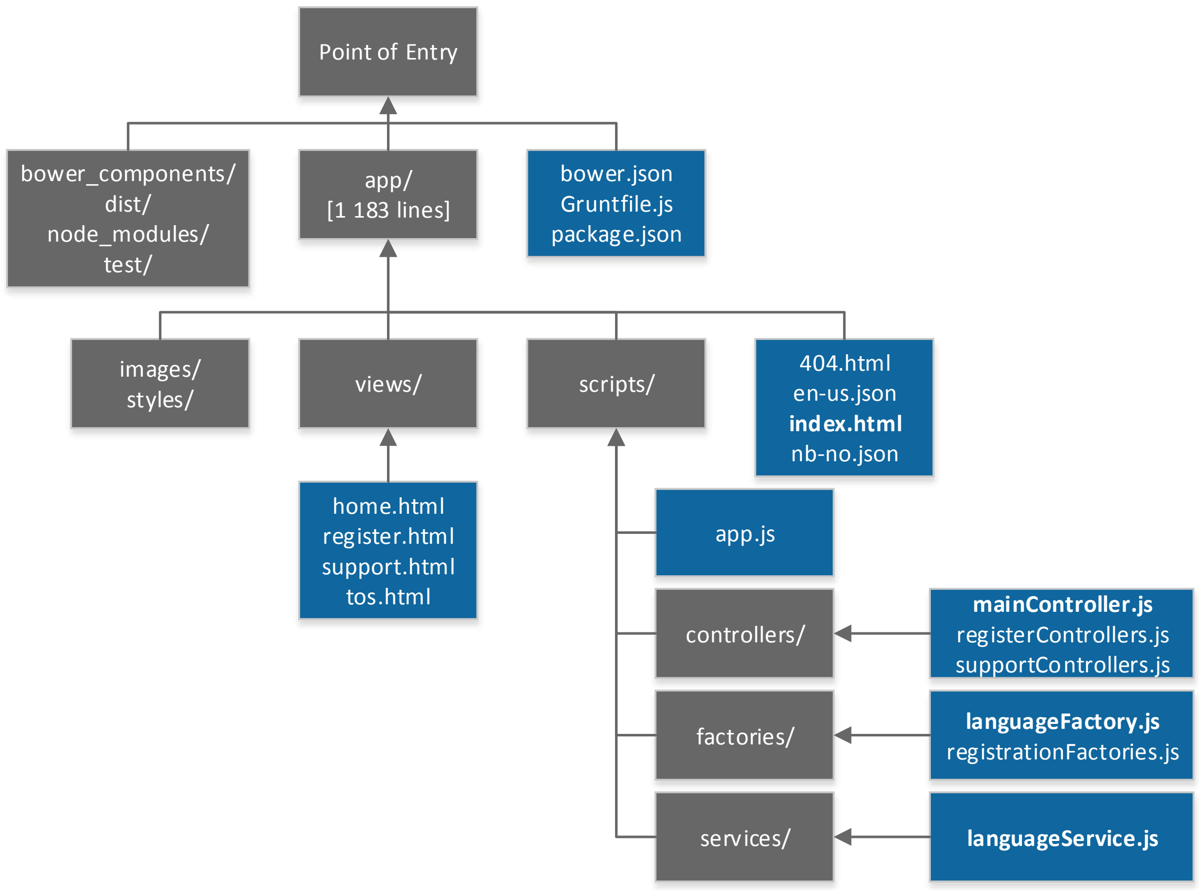


Figure 20 – Point of entry file structure expanded on the files being referenced.

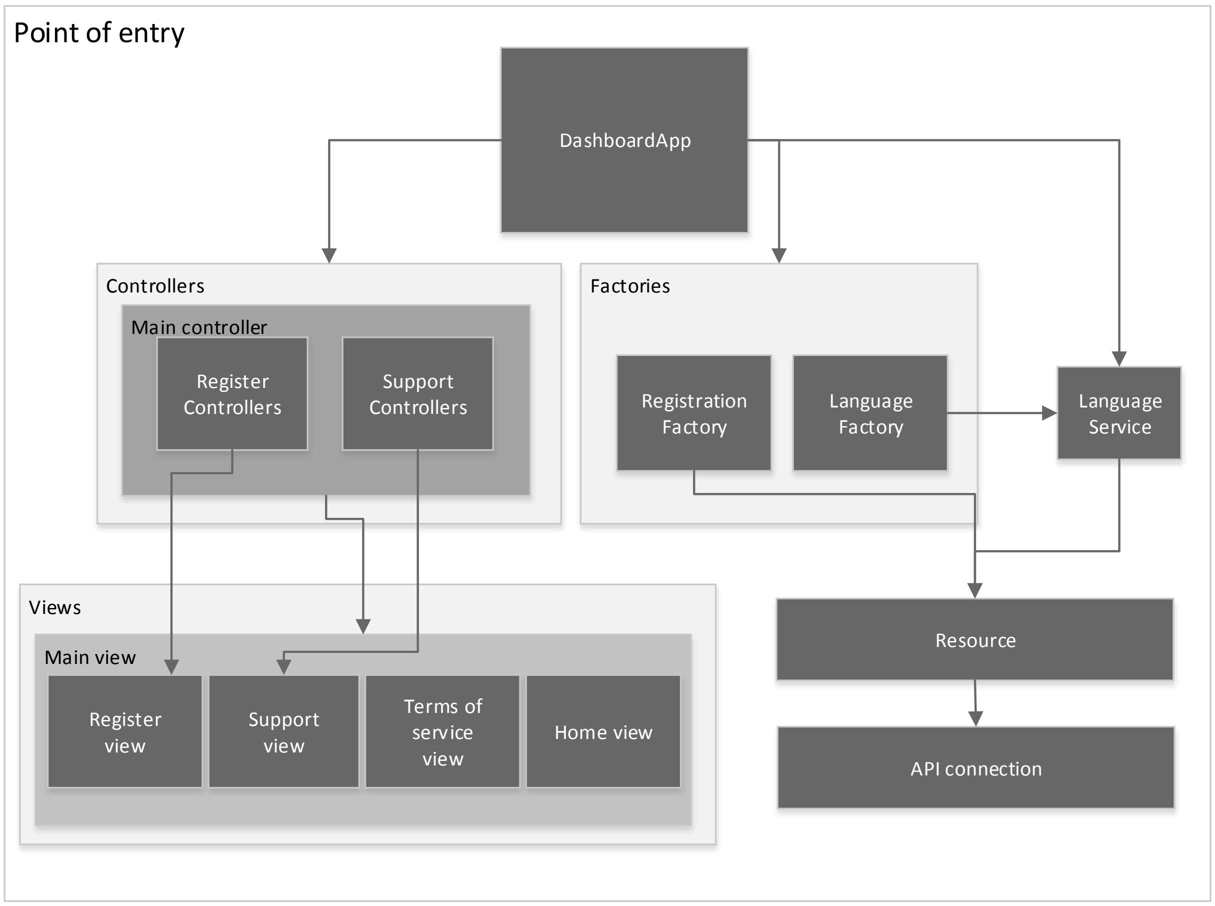


Figure 21 – Point of entry application structure.

The structure of the Point of entry Angular application is illustrated in Figure 21, which provides an overview of the application with low level of complexity and how it uses covers a lot of the core Angular features.

The application has four views that the user will experience as web pages, but only three of them have assigned controllers. This is because the Terms-of-service view contains only static content. The main view wraps all the other views and serves as content that shall remain in the application GUI (Graphical User Interface) regardless of which one of the other views that is currently loaded.
Point of entry consists of a single module called DashboardApp and has the following controllers assigned to it: Main, Register, and Support together with their respective views.

```
<body data-ng-app="dashboardApp" ng-controller="mainCtrl">
 <…>
   <!--This will be visible on all pages -->
 </…>
   <main class="container-fluid">
     <div data-ng-view class="…">
       <!—- Templates will be displayed here -->
     </div>
   </main>
</body>
```

Figure 22 – Simplified code snipped from index.html in Point of Entry showing main controller wrapping ng-view.

Figure 22 shows how the main-controller is registered in the body tag which wraps the entire application including the element with the `data-ng-view` attribute. This element is where views are loaded into. This means that whenever a view is loaded it will be available for the scope of the main-controller, but each view has been assigned its own controller to maintain readability in the code. This enables the main-controller to focus on functionality that involves the application as a whole (e.g. language and navigation) while the specialized controllers can focus on their own tasks.

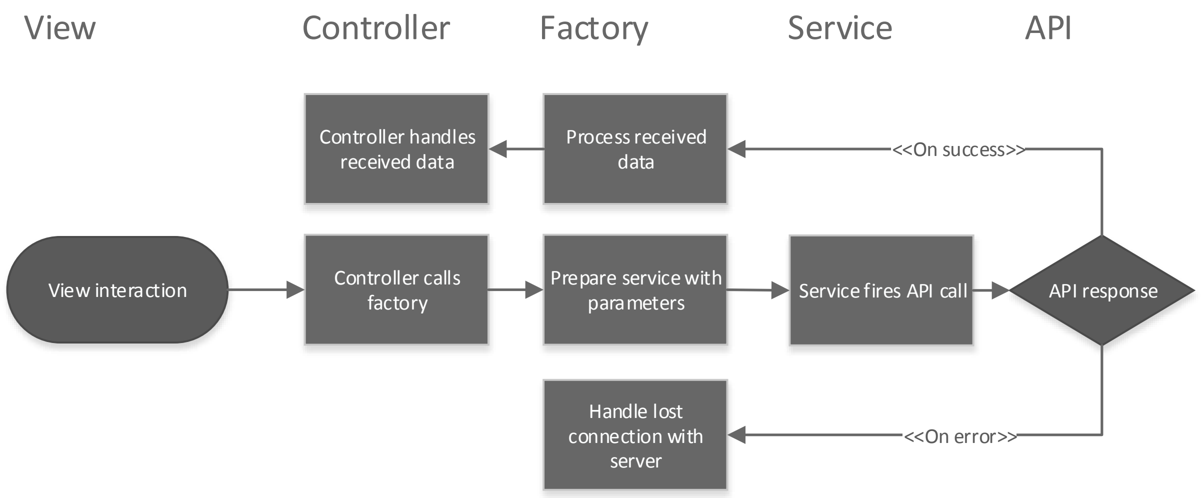


Figure 23 – Interaction flow between different angular components for an API GET request

Figure 23 illustrates how the different components in Figure 21 interact in a generalized setting for interaction between the user and the API. A concrete example follows.

Case: *Change Current Language*

View Interaction

The user selects a new language in the dropdown menu as shown in Figure 24.

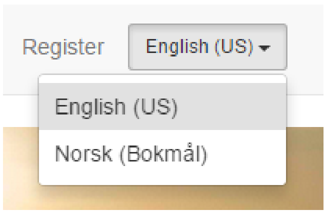


Figure 24 – Selecting language in the view.

```
<li … ng-click="changeLanguage(language)">…<li>
```

Figure 25 – Code outtake from the main view corresponding with Figure 24. Simplified snippet from main.html.

When a language has been selected, the on-click listener for the selected item will be fired (Figure 25). This listener is an Angular directive called ng-click, which can be passed JavaScript code that will be executed when the element is clicked. In this case the `changeLanguage` function, which belongs to the Main-controller scope. This function gets passed the language that has been associated with list element [28].

Controller Calls Factory

```
scope.changeLanguage = function(language) {
  languageFactory.changeLanguage(language, function(langPack) {
    scope.langPack = langPack;
    scope.currentLanguage = language.name
  });
};
```

Figure 26 – changeLanguage from mainController.js.

The Main-controller function `changeLanguage` illustrated in Figure 26 invokes the `languageFactory` to change the current language.

Prepare Service with Parameters

```
var setLanguage = function(languageCode, onSuccess, onError){
  languageService.getLanguagePackage.get({
    language:languageCode}, function(langPack) {
      onSuccess(langPack);
  },
  function(response) {
    // Connection lost, redirect to service unavailable
  });
};
```

Figure 27 – Language Factory method setLanguage triggered by languageFactory.changeLanguage. Snippet from languageFactory.js.

Figure 27 shows how `languageFactory` initiates an HTTP-GET request using `languageService` with code for the selected language, success and error scenario methods.

Service Fire API call

```
angular.module('dashboardApp')
  .factory('languageService', ['$resource', 'RESOURCE',
    function(resource, RESOURCE) {
      …
      var languagePack = resource(':language.json',
              {language:'@language'});
       …
```

Figure 28 – Language service utilizing Angular $resource for http requests. Snippet from languageService.js.

The Angular `$resource` is utilized in both of our Angular services. `$resource` is designed to easily interact with RESTful data sources and provides an abstract interface to the more low-level `$http` service [29]. In this particular case the language package object will be executed and attempt to get the JSON language file from the server.

API Response Success

A success scenario will involve a returned language package JSON file. This will trigger the `onSuccess` callback function in the `languageFactory` (Figure 27).

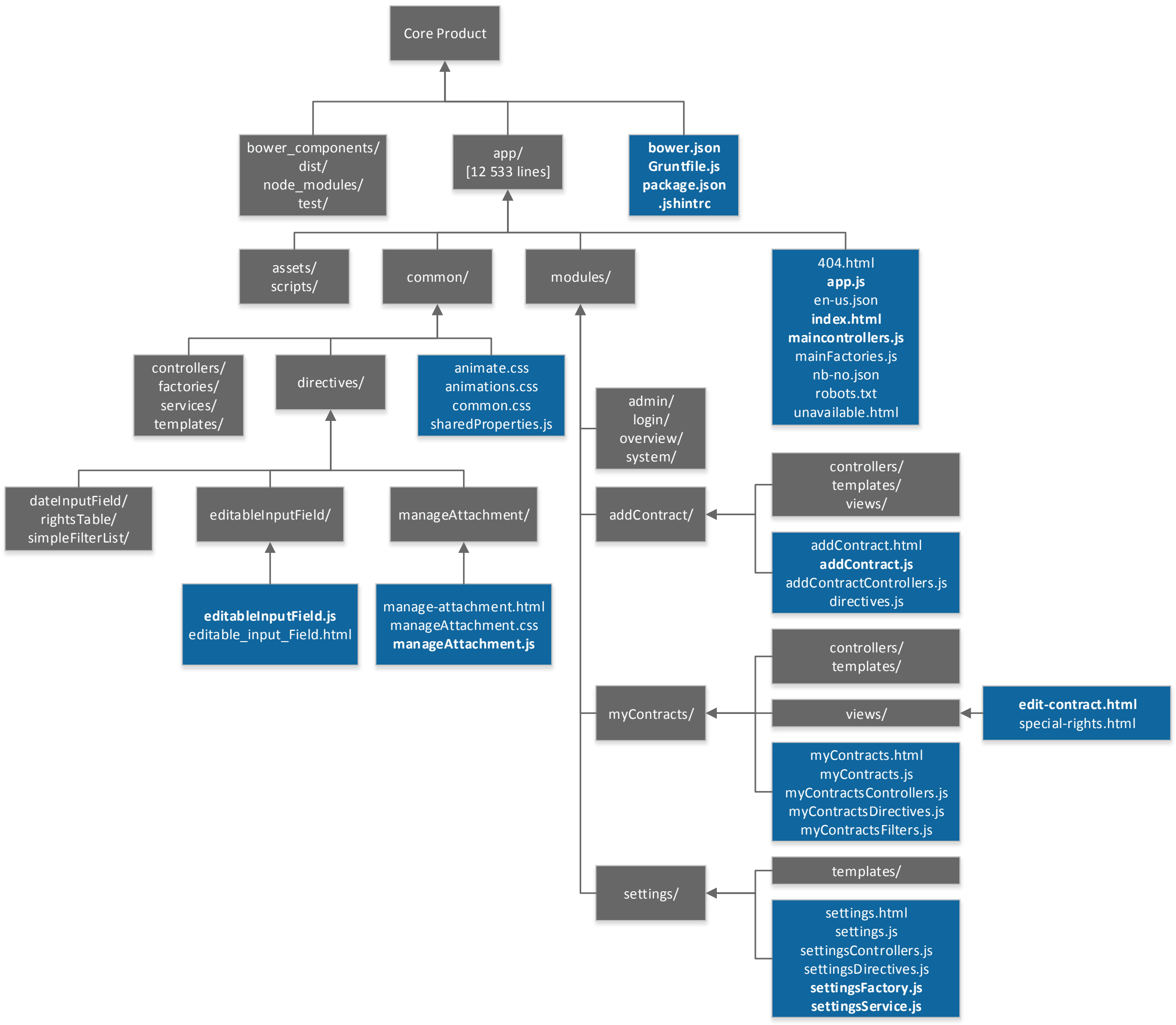
API Response Error

An error occurred and there was no response from the server which is likely to be loss of connection or server error. Therefore the user will be redirected to the “service unavailable” site. This action is performed by the `onError` callback function in `languageFactory` (Figure 27).

Controller Handles Received Data

The callback function in `changeLanguage` (Figure 26) receives the language package from the API and updates the scope variable `langPack` to point to the received data. It also updates the current selected language variable. The new language package is now available through the entire application as the main controller scope wraps the entire application.

Core Product



Core Product
bower_components/ dist/ node_modules/ test/
app/ [12 533 lines]
bower.json Gruntfile.js package.json .jshintrc
assets/ scripts/
common/
modules/
404.html app.js en-us.json index.html maincontrollers.js mainFactories.js nb-no.json robots.txt unavailable.html
controllers/ factories/ services/ templates/
directives/
animate.css animations.css common.css sharedProperties.js
admin/ login/ overview/ system/
dateInputField/ rightsTable/ simpleFilterList/
editableInputField/
manageAttachment/
addContract/
controllers/ templates/ views/
addContract.html addContract.js addContractControllers.js directives.js
editableInputField.js editable_input_Field.html
manage-attachment.html manageAttachment.css manageAttachment.js
controllers/ templates/
myContracts/
views/
edit-contract.html special-rights.html
myContracts.html myContracts.js myContractsControllers.js myContractsDirectives.js myContractsFilters.js
settings/
templates/
settings.html settings.js settingsControllers.js settingsDirectives.js settingsFactory.js settingsService.js

Figure 29 – File structure of core product expanded on the files being referenced.

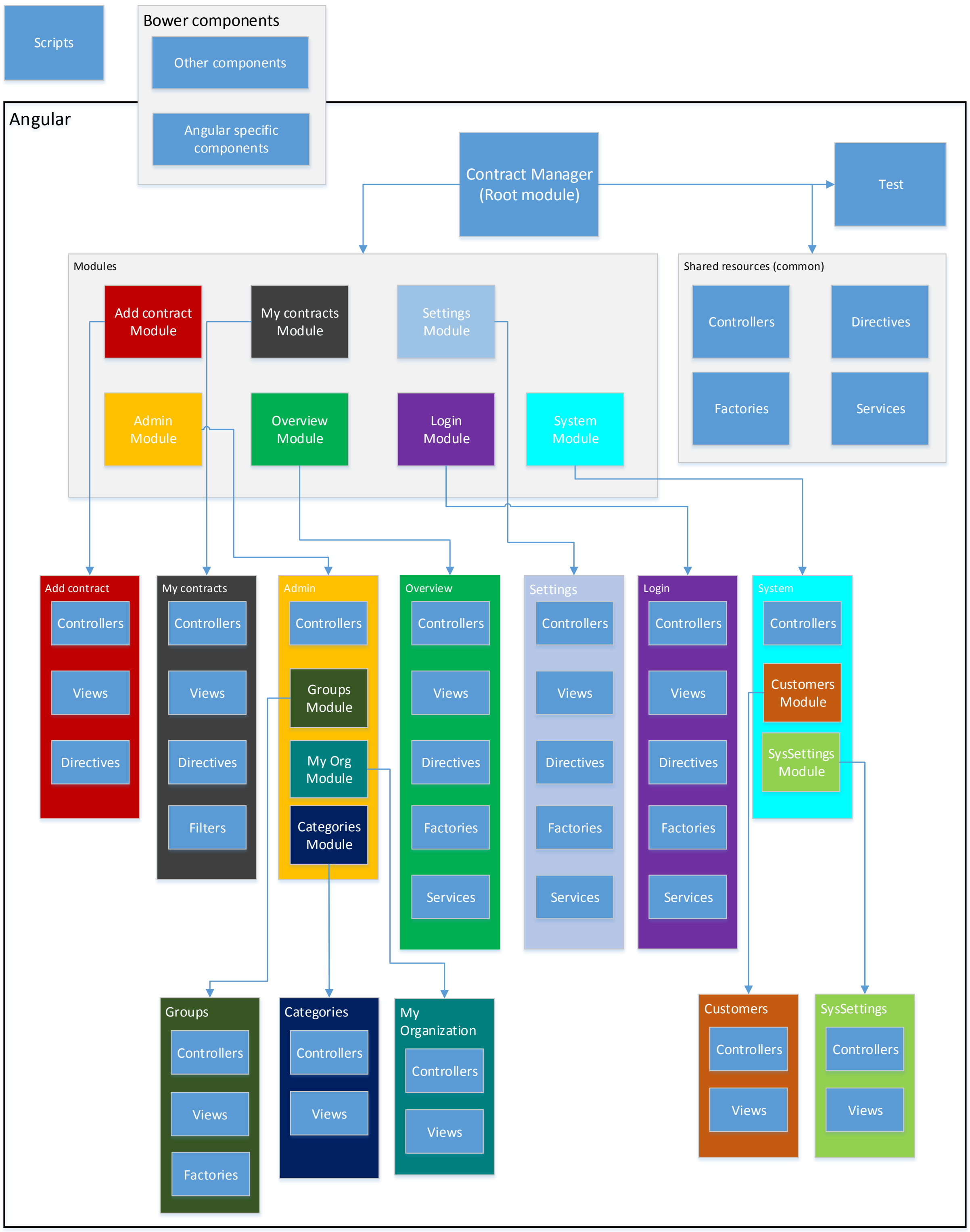


Figure 30 – Core product structure. The arrows points to an expanded view of each module.

Figure 30 provides a more abstract overview of the core product structure than in Figure 29 than what the point of entry did as shown in Figure 21. It provides an overview of all the core components in the Core product application by displaying all the modules, abstract content overview and their shared resources, all of which are built using the Angular framework.

Modules

Each module in Figure 30 is represented by its own color along with its resources and points to their respective content. All modules displayed and the shared resources (Figure 30) have been developed in this project, except for the bower components. Bower components are external plugins that have been included through Bower, even the Angular framework is a Bower component (see 6.7 Grunt and Bower).

The single exception is “Scripts”. Scripts contain standard JavaScript functions used as a bridge between the Angular application and external services like Facebook and the Hybrid mobile applications.

Instantiating

```
angular.module('ContractManager', [
    'loginModule',
    'myContractsModule',
    'overviewModule',
    'addContractModule',
    'settingsModule',
    'adminModule',
    'systemModule',
     …
    'Feedback'
  ])
```

Figure 31 – Instantiating ContractManager module (Root). Simplified snippet from app.js.

```
angular.module('addContractModule', ['mgo-angular-wizard']);
```

Figure 32 – Instantiation of addContract. Snippet from addContract.js.

The root module has seven modules attached to it, all of these listed as dependencies during instantiation as illustrated in Figure 31. All other dependencies are listed here as well as long as two or more sub modules have them as dependencies. An example would be the Bower component ‘Feedback’ module which purpose is to display feedback to the user [30]. This is used in several of our modules and therefore listed as dependency of the root module.

Figure 32 illustrates the instantiation of the addContract module. This module is the only module utilizing the wizard and therefore it is listed as a dependency here instead of the root module.

Routing

```
angular.module('ContractManager').config(['$routeProvider',
  function (routeProvider) {
    routeProvider
      .when('/login', {
         templateUrl: 'modules/login/login.html',
         controller: 'loginCtrl'
      })
…
```

Figure 33 – Sets up routing configuration for ContractManager module. Simplified snippet from app.js

The root module (Top of Figure 30) is configured to handle routing related to its own submodules. Routing is done using the ngRoute directive which provides access to the `$routeProvider` object [31]. We have utilized the `when()` and `otherwise()` functions from the `$routeProvider` to determine which template files to load into the ng-view depending on the URL address. In this case, when the location URL changes to #/login, the main view of the login module will be loaded and assigned its controller as illustrated in Figure 33.

```
.otherwise({redirectTo: '/login'});
```

Figure 34 – Unassigned paths will result in redirect to /login. Simplified snippet from app.js.

This is done for all the main views within the application with their own assigned paths. Invalid paths will result in a redirect back to /login. If the user has an active session, the user will again be redirected back to overview.

```
.when('/myContracts', {
 templateUrl: 'modules/myContracts/myContracts.html',
 controller: 'myContractsCtrl',
 resolve: {
  lazy: ['$ocLazyLoad', function(ocLazyLoad) {
   return ocLazyLoad.load([{
    name: 'myContractsModule',
    files: [
     'modules/myContracts/myContractsControllers.js',
     …
    ]…
…
```

Figure 35 – Setup for lazy loading files on request. Simplified snippet from app.js.

Despite this application being an SPA, we have tried to utilize some lazy loading as the application has specialized modules for specific operations and a typical user session does not require all the modules. To achieve this, we have utilized a third party Angular plugin called ocLazyLoad [32]. An example of usage is illustrated in Figure 35. By using lazy loading, the network traffic is reduced, which is favorable for both the server and users on mobile devices with limited bandwidth. The initial reason for using lazy loading was that it solved an issue with the language package not being distributed properly around in the system due to everything loading in at the same time asynchronously. This is no longer a problem as the language package is set to load before everything else, but it was decided to keep the lazy loading due to the positive effects mentioned above.

Factories and Services

As illustrated in Figure 30, factories and services are spread around different modules and in the shared content area. The reason for this is that some factories and services are used by several modules and some are specialized for specific modules. The `ContractFactory` for example is responsible for everything related to creating and updating contracts. This functionality is needed from several locations in the application and therefore the `ContractFactory` has been placed among the shared resources. On the other hand, settings factory is only utilized by the settings controller to update profile settings, no other modules depends on this factory or the settings service and has therefore no reason to be a shared resource.

```
// Simplified snippet from settingsFactory.js
angular.module('settingsModule')
    .factory('settingsFactory','settingsService,
        'sessionFactory', '$window',
        function(settingsService, sessionFactory, window) {
          ...
        }
    ]);
// Simplified snippet from settingsService.js
angular.module('settingsModule')
    .factory('settingsService', ['$resource', 'RESOURCE',
         function(resource, RESOURCE) {
           ...
         }
    ]);
```

Figure 36 – Factories and Services both utilize the angular.module.factory receipt.

We chose to utilize factories and services as illustrated in Figure 23 where factories contain business logic and often use one or more service whereas services acts as an interface towards the API. However, both factories and services use the angular.module.factory recipe [33], even though there is an angular.module.service available(Figure 36). There are several reasons why we decided to use the factory receipt in both cases. The factory and service recipes both inherit the same angular.module.provider receipt and have marginal differences. While doing research regarding Angular, we observed that the service receipt seemed pretty much neglected by the entire Angular community. In most cases, references to service objects was not coherent with objects created from the service receipt, but with factory.

Directives

Directive is a versatile Angular tool which enables creation of customized HTML tags either as an element, attribute or class, or a combination of them. This made it possible to minimize code duplication as we could put components that was needed several places into a directive and reuse it whenever needed [34]. Directive components can be thought of as a closed box with no connection to its surroundings. Although connection with the surroundings such as the parent controller scope can be achieved and even inherited. The Angular toolbox includes directives such as `ng-click` illustrated in Figure 25. The `ng-click` directive is restricted to be an attribute and its purpose is to set an on click listener on the element it is attached to. Most of the directives we made ourselves are restricted to be elements.

It seems to be consensus in the Angular community that all DOM manipulation should be done within directives and keep the controller as unintelligent as possible [35]. When we realized that this would make our code more reusable and testable, we settled for a more extensive use of directives.

```
<editable-input-field
   original-val="selectedContract.title"
   label="Contract name"
   show-label="true"
   update-fn="updateName(newValue)">
</editable-input-field>
```

Figure 37 – Editable input field directive utilized in edit-contract.html

```
angular.module('ContractManager')
 .directive('editableInputField', function() {
  return {
   restrict: 'E',
   templateUrl: 'common/directives/editableInputField/e…
   scope: {
    label: '@',
    showLabel: '@',
    originalVal: '=',
    updateFn: '&'
   }, …
```

Figure 38 – Editable input field has an isolated scope. Snippet from editableInputField.js

Figure 37 illustrates usage of our ‘editable input field’ directive utilized in the edit-contract template file. In this particular case, it is used to edit the name of the contract. Since this directive has its own closed scope, the variable to be edited has to be passed from the parent controller scope. The update function must be provided by the parent controller and passed as attribute. This was a choice we made to make the directive more adaptable and reusable for future usage.

As Figure 38 illustrates, the editable input field is attached to the root module which makes it available for all submodules to use. This directive is restricted to be used as an element, which means that if it is implemented as an attribute to other tags it will cause a compilation error. Template URL points to an HTML file which is the GUI layout for this directive which can be seen in Figure 39.

The different isolated scope variables in Figure 38 are handed as @, =, or & which are keywords for the Angular compiler that represent strings, two way binding and function passing.

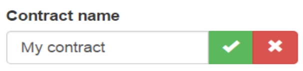


Figure 39 – editable input field in action

7.2 Implementation of Restlet and API

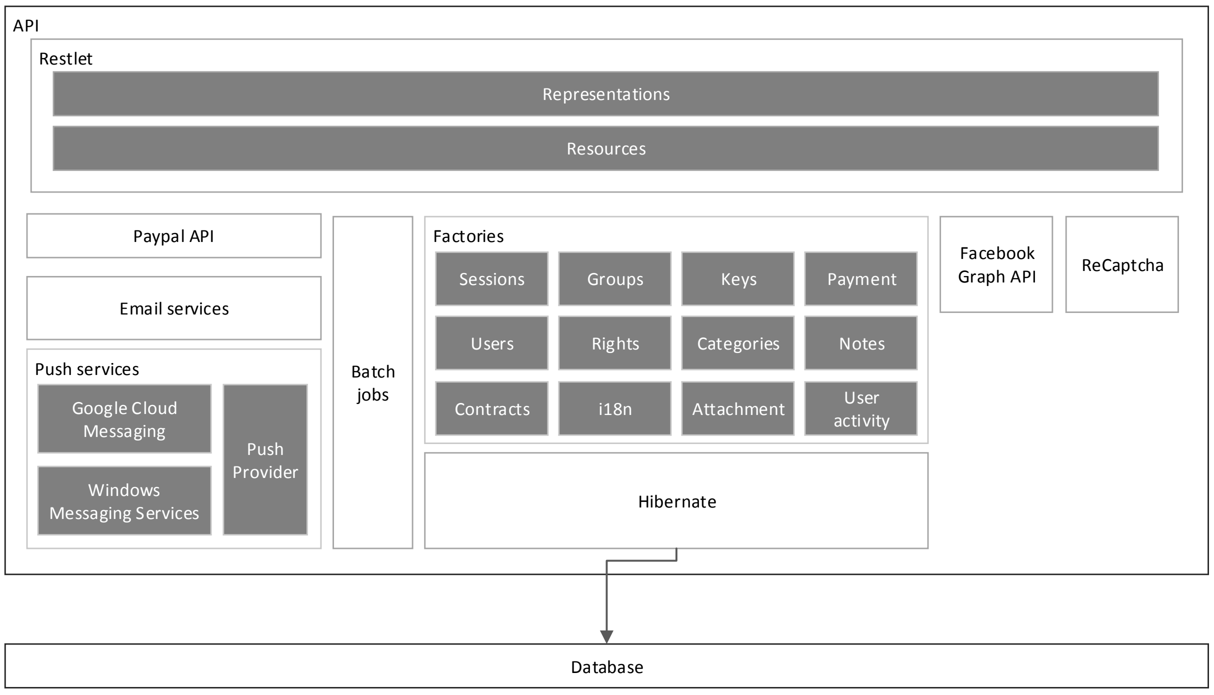


Figure 40 – A representation of the main components in the API. Each call is routed to the appropriate resource that will return a representation depending on the request.

This section will discuss central components in the API and how it has been built using Restlet. Figure 40 shows an overview of the different main components in the API. It also relies on different external services such as email, push services, Facebook and ReCaptcha. Internal classes such as `sessions`, `groups`, etc. are available through factory classes that interacts with Hibernate. These classes are exposed through annotations for making an appropriate representation, given the context.

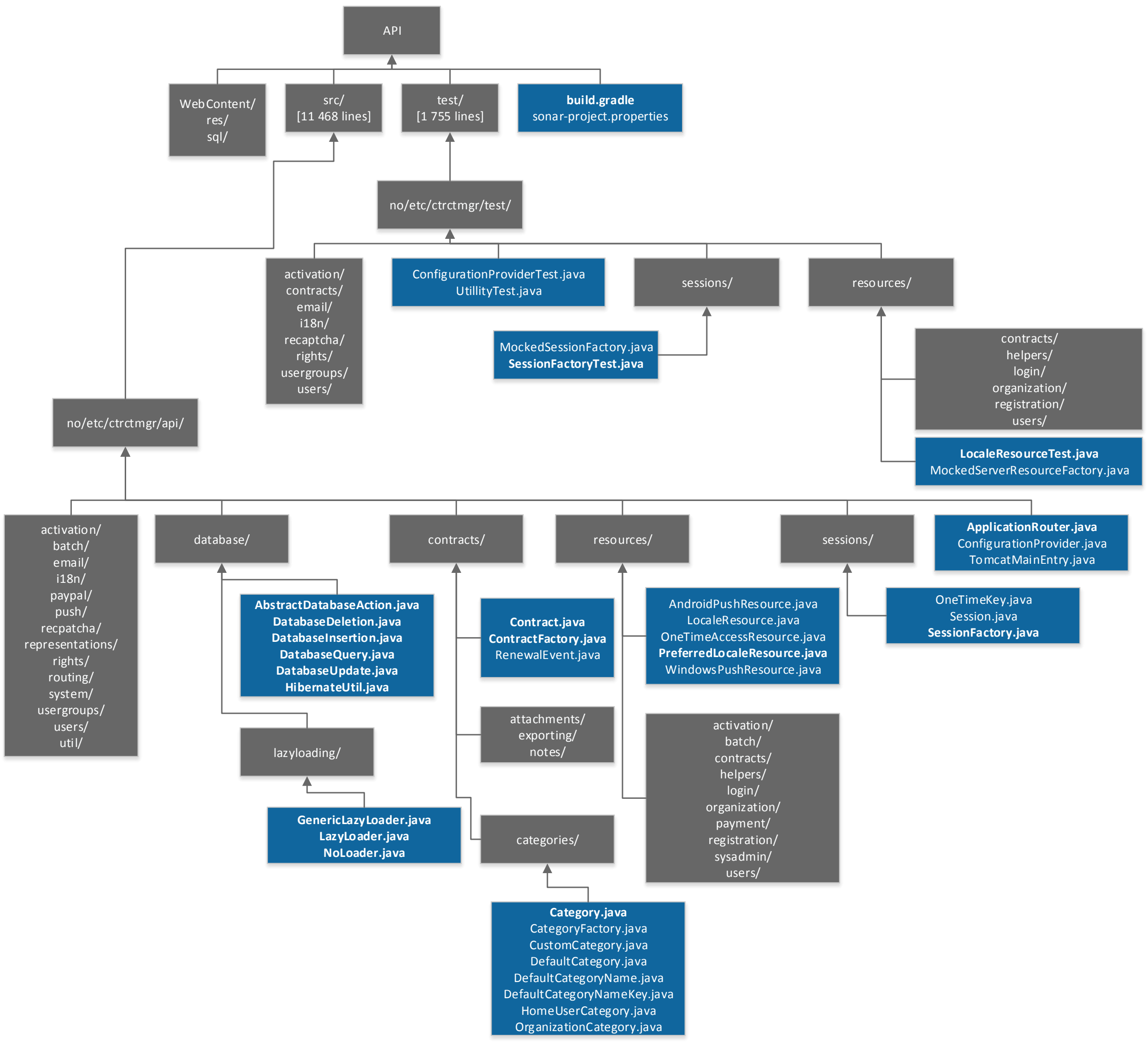

API
WebContent/ res/ sql/
src/ [11 468 lines]
test/ [1 755 lines]
build.gradle sonar-project.properties
no/etc/ctrctmgr/test/
activation/ contracts/ email/ i18n/ recaptcha/ rights/ usergroups/ users/
ConfigurationProviderTest.java UtillityTest.java
sessions/
resources/
MockedSessionFactory.java SessionFactoryTest.java
contracts/ helpers/ login/ organization/ registration/ users/
LocaleResourceTest.java MockedServerResourceFactory.java
no/etc/ctrctmgr/api/
activation/ batch/ email/ i18n/ paypal/ push/ recpatcha/ representations/ rights/ routing/ system/ usergroups/ users/ util/
database/
contracts/
resources/
sessions/
ApplicationRouter.java ConfigurationProvider.java TomcatMainEntry.java
AbstractDatabaseAction.java DatabaseDeletion.java DatabaseInsertion.java DatabaseQuery.java DatabaseUpdate.java HibernateUtil.java
Contract.java ContractFactory.java RenewalEvent.java
AndroidPushResource.java LocaleResource.java OneTimeAccessResource.java PreferredLocaleResource.java WindowsPushResource.java
OneTimeKey.java Session.java SessionFactory.java
lazyloading/
attachments/ exporting/ notes/
activation/ batch/ contracts/ helpers/ login/ organization/ payment/ registration/ sysadmin/ users/
GenericLazyLoader.java LazyLoader.java NoLoader.java
categories/
Category.java CategoryFactory.java CustomCategory.java DefaultCategory.java DefaultCategoryName.java DefaultCategoryNameKey.java HomeUserCategory.java OrganizationCategory.java

Figure 41 – Overview of the file structure of the API source code that has been written during this project expanded on the source files being referenced.

When the user is requesting all the contracts that the user has access to, an HTTP-GET call is made to a `ContractOverview` resource. Before the resource attempts to find any contracts, the session key-id is validated in order to verify and identify the user making the request. After this, the resource will request a list of contracts from a `ContractFactory`, which will find the contracts that the user has sufficient rights to view through HQL. The resource will then return a `GSONRepresentation<Contract>` from a `RepresentationFactory` that would build up a GSON-representation based on the annotation provided by the resource to the factory.

```
@Override
public Restlet createInboundRoot() {
    if (!Utility.inDev()) {
        Logger.getLogger("org.restlet.Component.LogService")
                .setLevel(Level.SEVERE);
    }

    Router router = RouterFactory.makeRouter(getContext());
    router.attach("/register/homeuser", HomeUserRegistrationResource.class);
    ...
    router.attach("/access/{key}-{id}", OneTimeAccessResource.class);
    router.attach("/activation/{key}-{id}", ActivationResource.class);
    router.attach("/languages", LocaleResource.class);
    router.attach("/languages/preferred", PreferredLocaleResource.class);
    router.attach("/batch/{key}", BatchResource.class);

    return router;
}
```

Figure 42 –The function `createInboundRoot()` that registers all the different API calls. This function is a simplified version of the original code from `ApplicationRouter.java`.

The first HTTP call after startup of the API will instantiate `createInboundRoot()` in `ApplicationRouter` (Figure 42). This will build up the router that routes calls to their appropriate resource and registers each server resource together with its URI. One issue was handling sessions, as we did not want to add an if-statement in each function in the server resource that checks if the connection has sufficient rights. This was achieved by creating the abstract class `AuthenticatedServerResource` that would intercept the HTTP call and verify the session before it reached the `ServerResources`. That way, we could easily require a valid session for a resource by making it extend the abstract class.
One challenge when developing the API was to avoid having duplicate classes for different representations. Data from an entity should for example not be copied over to a new class just to select which fields should be serialized. An approach like this would lead to duplicates and varieties of classes spread around the system. It would be hard to maintain and not very good in terms of object-orientation as there exists variants of the same class throughout the system. The goal was to reduce code and duplications in the same way class duplications can be avoided with Hibernate. This was solved by creating an annotation that specified which fields of an object should be serialized together with GSON by using reflection. When a representation of a class was required, the class together with the annotation type was passed to a `RepresentationFactory` that would create the GSON representation of the fields annotated with the given annotation. Being limited to class members would be a

disadvantage as there are cases where the return value of a function is required in a JSON object. This was solved by finding methods that returns a string and is annotated with `@ExposedMethod(String propertyName)` inside the object being subject to serialization (Figure 43). Using annotations also solved the issue where a class needs to have several different representations due to the fact methods and properties can have several annotations.

```
@DefaultContractRepresentation
@CategoryRepresentation
@ExposedMethod("name")
public abstract String getName();
```

Figure 43 – Snippet of how the abstract function `getName` under the class `Category` is annotated for different representations.

As discussed earlier, it is required that the API must be stateless in order to be RESTful. Usually, session tokens are attached to requests using cookies and stored on the server through the application layer. For example, a traditional PHP web-application (See 5. System Architecture) would store the session token in cookies and retrieve the session state from the `$_SESSION` variable [36]. PHP would either store the session data in memory or to the file system of the server. This does not follow RESTfull principles as the state is persistent through memory or the file system through the PHP runtime. It can therefore not be accessed through another instance of the webserver. However, if stored on a common medium such as a database, each instance would be able to retrieve the session state. Storing session identifiers in the database is a common approach when handling session in REST [37] and is the approach we decided to use when storing sessions in our system.

```
public static Session createSession(User user, boolean longExpTime) {
    Session session = new Session()
            .setSessionKey(Utility.randomString(120)).setUser(user)
            .setLongLived(longExpTime);
    setSessionTime(session);

    try (DatabaseInsertion<Session> query = HibernateUtil.getInstance()
            .insert()) {
        int id = (Integer) query.insert(session);
        session.setId(id);
    }

    return session;
}
```

Figure 44 – Generation of a random, unique session key. This function is located in `SessionFactory.java`.

Another challenge was to create session tokens that was unique to prevent sessions from colliding and prevent others from entering it. As Hibernate requires entities to have a primary key, the auto-incremented primary key value could be used together with a random string as the session token. This would result in a string both random and unique. (Figure 44) We decided to put the session token in the request URL as this would provide easy access to the key and easy integration on different platforms as it provides a more standardized and self-documenting implementation compared to using cookies. However, in retrospect we can see that using OAuth 2.0 [38] by using standard HTTP authorization and putting the session key in the Authorization field would be a better approach. This would lead to some more code for header modifications when making API-calls that requires authentication in the web-app and mobile apps. On the other hand, it would lead to a more standardized way of authentication, which would make it easier for new developers to understand how the system deals with authentication.

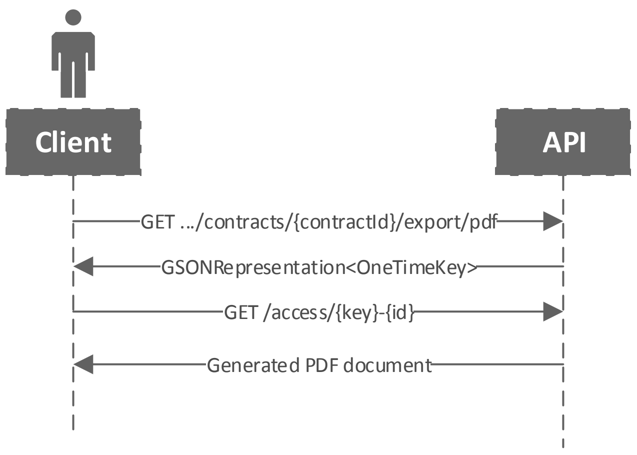

Client
API
GET .../contracts/{contractId}/export/pdf
GSONRepresentation<OneTimeKey>
GET /access/{key}-{id}
Generated PDF document

Figure 45 – How a client downloads a PDF export of a contract using a one-time key. The first URI has been shortened.

One issue of placing the keys in the URI is that they can easily be exposed, for example when downloading an export of a contract. If Alice sends the link to Bob, Bob would have the session token for Alice and could potentially access that session. It is possible to limit each session to an IP address, however there is no guarantee for which person is behind an IP-address as they may be on the same network and a user may be traveling between different network and would not want to re-authorize. This problem was solved by generating a one-time key that would allow a user to access a resource one time only as seen in Figure 45. After accessing, the one-time key is deleted and is no longer usable. That means if Alice sends a download link for exporting a contract to Bob, Bob would only have a one-time link that does not relate to the session of user Alice.

Encryption was a system requirement specified early on in the project. Having stored contracts in a secure manner is important to prevent intruders from obtaining the contracts in case of a data-breach. However, this was down prioritized as encryption became more extensive than first anticipated (further discussed under 10.2 Further Development).

7.3 Hibernate Integration

Hibernate plays an important role in Contract Manager as it acts as the communication layer towards the database when data are made persistent such as user information and contracts. This section will discuss the following central aspects of Hibernate: the `HibernateUtil` singleton class that acts as a façade towards Hibernate, the Hibernate configuration file, query helpers for database interaction, lazy loaders and the entities themselves.

Access to Hibernate is available through the singleton class `HibernateUtil`. This class acts as a wrapper around different APIs in Hibernate in order to reduce duplicate code. The class was implemented as a singleton as it holds references to a connection pool, which Hibernate uses to manage database connections. The code snippet in Figure 46 shows how the class is implemented using singleton and how it achieves thread safety and lazy loading through static initialization.

```
public class HibernateUtil {
    private static final HibernateUtil INSTANCE = new HibernateUtil();

    private HibernateUtil() {
        // Initializes Hibernate
    }

    public static HibernateUtil getInstance() {
        return INSTANCE;
    }
}
```

Figure 46 – The singleton patter applied to the `HibernateUtil` class. This snippet is a simplified version of the actual class.

Once `getInstance()` is called, the singleton instance is created. One reason why it is implemented as singleton is that the class needs to be initialized and there should be only one instance of it. Although it could be implemented as a static class and initialized in `ApplicationRouter.java`, a static class would require an initialization function that must be called before any database calls are made. Applying the singleton pattern would remove the initialization call as the class is initialized at the first call to `getInstance()`. It is therefore not possible to forget to initialize Hibernate as the initialization call is there by design through the singleton pattern, for example for unit tests. Another important aspect is thread safety. Initializing Hibernate may take several seconds, as it has to load classes, map them, build up queries, connect to the database and so forth. If two calls are made to the API where both would attempt to create the singleton instance, the Java language specifications guarantees that its static fields are initialized before any calls to the class are made [39]. As the class initialization is thread safe, the singleton implementation becomes thread-safe, therefore, there is no `synchronized` keyword in the function `getInstance()`.

Setting up the Hibernate configuration is made by replacing strings in the `hibernate.cfg.xml` file with environment variables. Although it is possible to use a traditional configuration file, environment variables is easier in a cloud environment as the environment variables for a deployment is configured through an XML deployment file (see appendix F – Deployment on Microsoft Azure using FTP). The string "`${env.CTRCTMGR_DB_USERNAME}`" in the XML file is then replaced by the environment variable "`CTRCTMGR_DB_USERNAME`".

Database operations are made through query-helper objects that acts as a wrapper around a `Transaction` and a `Session` instance. The `Transaction` instance has to be retrieved before a `Session` instance can be retrieved out from it, which performs the database operations. Once this is done, it can then perform any required database operation. If no exception was thrown, it should commit its changes, otherwise rollback its changes. As having to call all of these functions in addition to handling exceptions would result in duplicated code, these calls are then abstracted through the query-helper classes. Figure 47 shows the different query helpers and their relation.

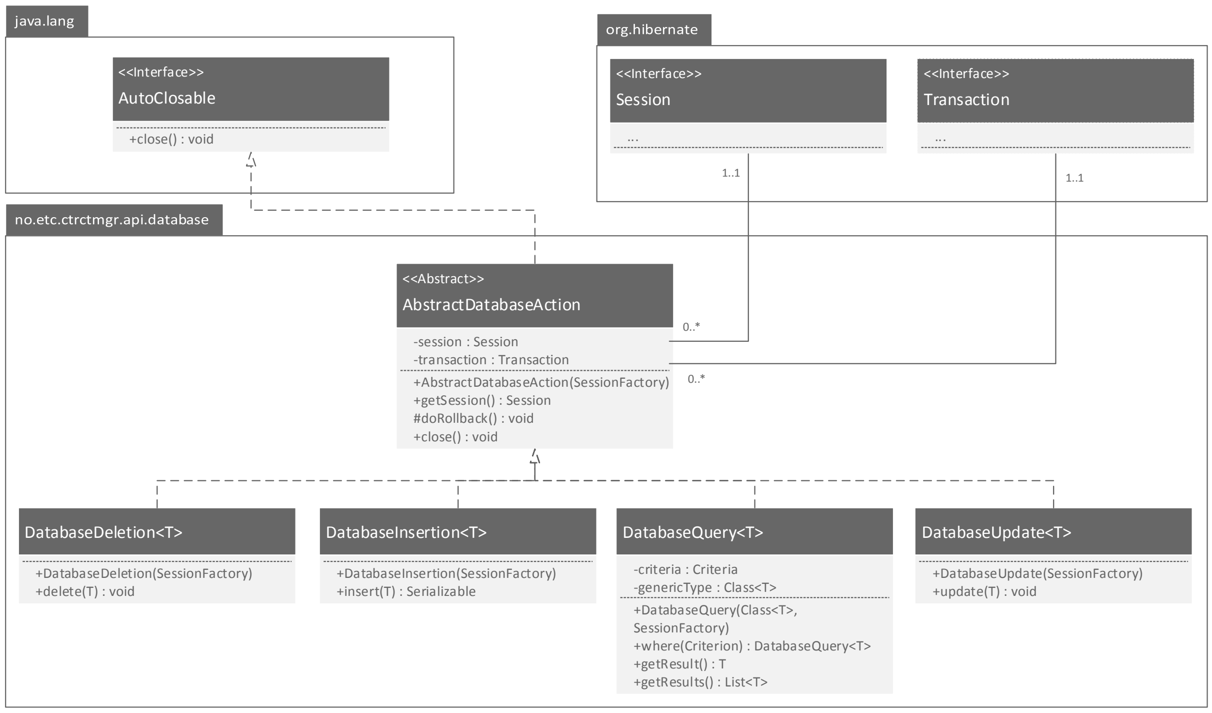


Figure 47 – UML diagram of the different query helpers in addition to how it is built on top of existing library function.

Note that the `DatabaseQuery<T>` has a function `where(Criterion)`, which returns a `DatabaseQuery<T>`. This is to use the cascade pattern [40] in order to simplify conditions on a query. The code snippet in Figure 48 from `SessionFactory.java` demonstrates how `Session` entities are retrieved.

```
try (DatabaseQuery<Session> query =
             HibernateUtil.getInstance().query(Session.class)) {
    session = query.where(Restrictions.eq("id", id))
            .where(Restrictions.eq("sessionKey", key))
            .getResult();
}
```

Figure 48 – Example usage of the query helper `DatabaseQuery`. The snippet is from the function `getSessionKey` in `SessionFactory.java`.

These queries are mostly used in factory classes. The factories are implemented as static classes unless they need initialization, where in that case the singleton pattern is applied. These factories performs different database actions such as inserting, updating and retrieving.

A class that is capable of being persistent through Hibernate needs the `@Entity` annotation. Parent entities are annotated with `@Inheritance` to declare they can be inherited. Entities must also have a field declared with `@Id` that becomes the primary key. A field that is auto-incremented, for example a contract id must be annotated with `@AutoGenerated`. N-to-N

relations can also be implemented by using @OneToOne, @OneToMany, @ManyToOne or @ManyToMany annotation on the foreign member, depending on the type of relation. These annotations may also hold information such as whether the member should be lazy loaded or not, what member name it is mapped to in the other entity and what should happen on object deletion. It is important to note that if the entity Contract that has a one-to-many relation to the entity Note, Note must have a many-to-one relationship to Contract and the one-to-many annotation must describe which member it maps to in Note. Otherwise, Hibernate will create an additional table contract_note with the references between the two classes, as it will not understand how to map them.
As objects may hold references to other objects, there has to be a mechanism to decide when objects should be loaded and when they should not be loaded. By default, many-to-one relations are loaded once an entity is retrieved from the database, unless their annotation specifies they should be lazy loaded. One-to-many has lazy loading by default [41], unless the annotation specifies it should follow the eager fetching strategy, which means the items should be fetched once an entity is retrieved from the database. A disadvantage of using this strategy is that objects quickly becomes expensive to retrieve from the database, therefore the lazy loading strategy should be used whenever possible on one-to-many relations.
One challenge with using lazy loading was to create a wrapper that would perform the lazy loading on class members, as the entity had to be associated with an open session before the member could be fetched from the database. This was solved by first checking if the member was lazy loaded, if it was not loaded the entity should be reloaded from the database and then member could be loaded. The issue here, however, was that this could not be performed in one call. If there was a function void <T> assertLoaded(T owner, Object member) that first refreshed the owner then loaded the object, the member parameter would hold a reference to the member in the old instance before the owner refreshed from the database. Figure 49 shows how this is applied to getting notes for a contract.

```java
public List<Note> getNotes() {
    return HibernateUtil.getInstance()
            .using(this, this.notes)
            .assertLoaded(this.notes);
}
```

Figure 49 – Code snippets for getting all the notes from a contract from `Contract.java`*.*

The `using` function will refresh the entity if the member (in this case, `notes`) is not loaded, thereby associating it with a session. Once the call is performed, `this.notes` is sent to the lazy loader as this is now a collection in a refreshed entity associated with an open session. An important aspect of the `HibernateUtil.using` function is that it will return a concrete implementation of `GenericLazyLoader<T>` where T is the type of the object being lazy loaded. It will decide which type of `GenericLazyLoader<T>` to create and return based on the return value from `Hibernate.isInitialized(member)` that checks whether the member is initialized or not. If it is initialized, it will return a `NoLazyLoader<T>` that simply returns the member in the `assertLoaded` function.

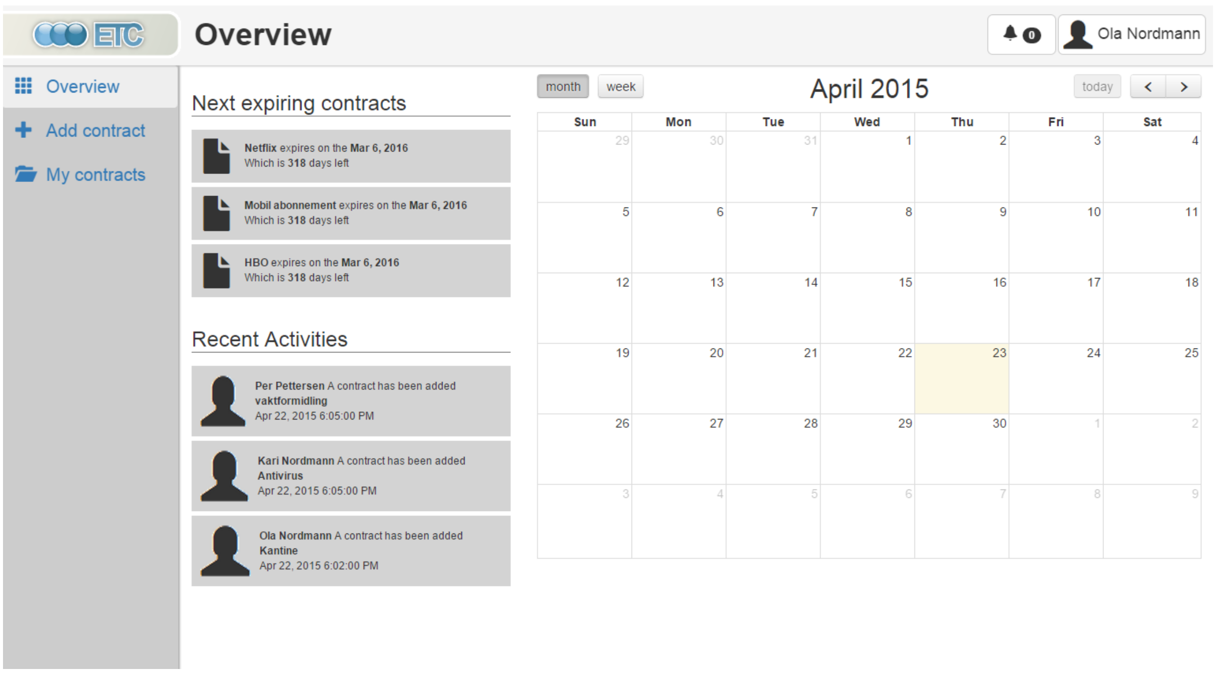
7.4 Graphical User Interface

The Graphical User Interface (GUI) is one of the most important factors in any computer system. It defines the first impression for the user and would later on have an impact on the overall user experience. If the user finds the software to be difficult to learn and/or use, an otherwise excellent product could fail [42]. Universal design principles have been used in order to satisfy parts of the target audience.

It was predetermined that the web app had to be responsive and thus support different screen resolutions. It was challenging to maintain a good user experience while displaying less information on smaller devices. Having a scalable website is becoming more and more mandatory and especially for businesses. For example, Google punishes websites that is not mobile friendly by reducing their priority when displaying search results [43]. AngularJS directives provides making custom HTML tags with directives which leads to eliminating duplicate HTML. This makes the design easily extendable. Designing for desktop was prioritized from the beginning, but how the layout would scale down was always considered and adjustments were made accordingly. By maintaining a familiar user interface on all screen sizes makes it easier for the user when changing devices with different screen resolutions.

Main Layout

During one of the first sprints we discussed ideas of how the main layout should look like for the core product. We got inspiration from some of the most popular social media websites, news webpages, and similar software such as Novatus contracts [44] and eContracts [45]. They all have a header where the account button is placed to the right. Novatus and eContracts also used the whole width given as available space for displaying content. These two layout principles can be recognized in our system as well. After showing this structure to our customer, he was satisfied with the approach.



Next expiring contracts

Netflix expires on the **Mar 6, 2016**
Which is **318** days left

Mobil abonnement expires on the **Mar 6, 2016**
Which is **318** days left

HBO expires on the **Mar 6, 2016**
Which is **318** days left

Recent Activities

Per Pettersen A contract has been added
vaktformidling
Apr 22, 2015 6:05:00 PM

Kari Nordmann A contract has been added
Antivirus
Apr 22, 2015 6:05:00 PM

Ola Nordmann A contract has been added
Kantine
Apr 22, 2015 6:02:00 PM

Figure 50 – Main structure of the core product. Showing the entry module - Overview.

When taking advantage of the full width, the header occupies the entire width. The main navigation were placed to the left, and the rest is the view for loading each module. This is shown in Figure 50. Our reasoning for a side menu rather than having menu items in the header was to make it more extendable. The number of such items has a limit before they start overlapping each other. Naming the menu-items also comes into play for how much space they take up. By keeping a side menu, the core product also scales more natural down on smaller screens and the users would recognize how to navigate through the app. This can be seen in Figure 51 where the button to the top left toggles the menu with a slide effect. One might argue why we did not put the menu items in the header as long as there was space there. This was due to the fact that the amount of items there could be were unknown at the time, and it would be harder to scale on mobile devices.

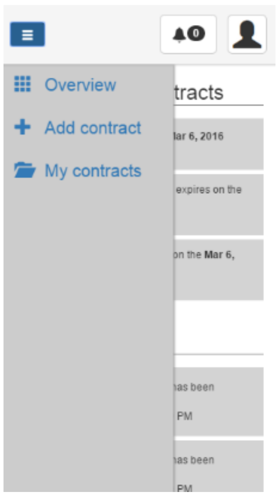
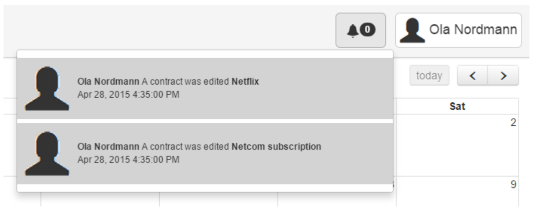


Figure 51 – The core product on a mobile phone. Having the same side menu.

The header in core product plays two important roles. It gives easy and quick access to main functionalities that does not necessarily relate to contracts and it tells the user where he or she is located within the app. The notification button acts as a dropdown menu where the user will find unseen notifications about contracts or the organization he or she is a part of. For example new members joining or when a new category has been created. The number represent the amount of unseen notifications and as soon the bell icon is clicked, the notifications is flagged as seen and thus reset to 0 as it can be seen in Figure 52.



Ola Nordmann A contract was edited Netflix
Apr 28, 2015 4:35:00 PM

Ola Nordmann A contract was edited Netcom subscription
Apr 28, 2015 4:35:00 PM

Figure 52 – The notification dropdown menu. Keep up with events concerning the user.

The button to the right from the notification is also a dropdown menu, but only for functionalities that is linked with the user account. For instance in Figure 53, one can see that Ola Nordmann is an organization administrator and has therefore access to managing groups (see 4.6 User Types) and his organization. It also has a link to general account settings for changing password, profile picture, and subscription for administrators. Even though the profile picture for home users is only seen by themselves, it gives the option to personalize their account. However, this is a feature that could later on be used in other features that may be added to the system. Also note that categories is not an admin option, so every user independent of their user type has access to this feature. Although free users get a dialogue message that the account needs to be upgraded when trying to add new categories. This was something our customer wanted as an attempt to encourage the users to subscribe to use the system.

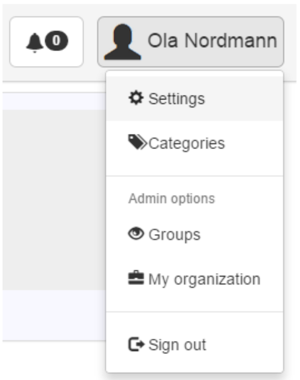


Figure 53 – Account options for an organization administrator.

When it comes to user feedback about where the user is currently located, we considered using the breadcrumb design-pattern [46]. It solves the problem of providing the user context in the application and it gives an easy way of navigating back. However, a pattern like this is more useful in applications where the link hierarchy is about four or more levels deep. Based on the functionalities that was required by our customer did not result in a deep link hierarchy, and instead a simpler solution were used by only displaying the current location in the header. For navigating back, the user has access to the side menu. This can be seen in Figure 54.

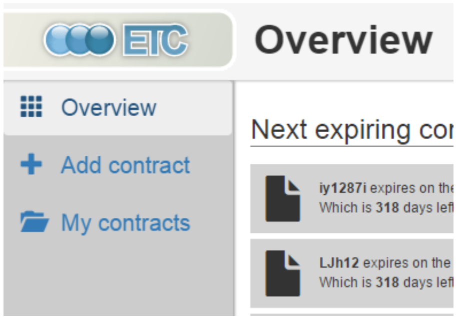


Figure 54 – Shows the side menu and how the user easily can see where in the app he or she is located.

Responsive Issues

When it comes to myContracts module, a lot is displayed to the user when selecting a contract; both options and information. This led to element wrapping and overlapping when the width is balancing on the Bootstrap boundaries (See 6.4 Bootstrap). As shown in Figure 55, the vertical option button overlaps the export dropdown menu, and the tabs below the header title has wrapped to several lines. This is not an easy problem to fix as the length of the tab names play a role of how much space they take up and was therefore ignored due to prioritization. One of the solutions involved making the side menu a toggle button just like it is on the smaller devices, but this required some refactoring and leaves desktop users having to do twice the work when navigating around the app. Another solution was to make the tabs and some button labels only displaying icons, however this would be confusing as some of the buttons would share the same icons have we been consistent.

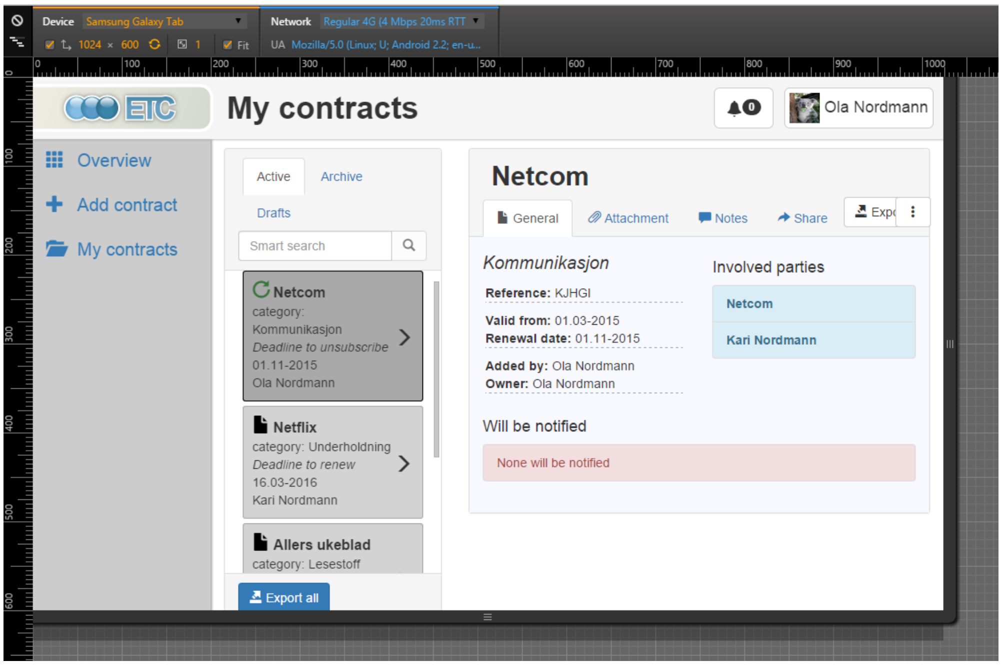


Figure 55 – When the screen width is balancing on the Bootstrap boundaries, some issues appeared. Notice the tabs to the left and the option buttons.

Finger Friendly Buttons

When developing for mobile devices, touch becomes the main source for input, and smaller objects (also referred to as target) on the device becomes increasingly harder to interact with. It requires more accuracy from the user and with grouped buttons they can accidentally hit a neighboring target and trigger an unintended action. Using the thumb to press buttons also has the risk of covering the whole button, leaving it to display no feedback when pressed, as shown in Figure 56. Having bigger buttons is also consistent with Fitts's Law [47], as the user is not slowed down because he or she needs to focus on accuracy. This concerns all screen sizes.

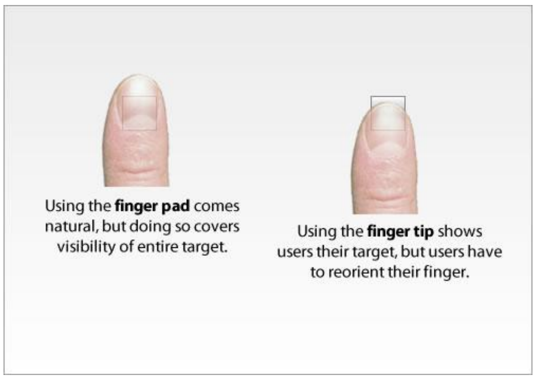

Using the **finger pad** comes natural, but doing so covers visibility of entire target.
Using the **finger tip** shows users their target, but users have to reorient their finger.

Figure 56 – Showing that small buttons tends to make the user use their fingertip instead of their finger pad (8).

Massachusetts Institute of Technology (MIT) have studied human fingertips to investigate the mechanics of tactile sense and found out that the average width of the index finger is about 1.6 cm – 2 cm. This is actually wider than what most mobile guidelines suggests. Google recommends about 7 mm – 10 mm [48], while Microsoft recommend not going smaller than 9 mm on square buttons. They also note that rectangle-shaped buttons can have a minimum height of about 7 mm [49].

Bootstrap provides three different button sizes where the standard is 30 pixels in height. By appending the `btn-sm` class to the button element, it becomes 22 pixels, while the `btn-lg` makes it 46 pixels. However, CSS pixels is not a physical unit and converting it to millimeters depends on the density of the screen or DPI (Dots per Inch). In other words, physical and logical pixels are two different units. For instance, a device can have a screen resolution of 1080 x 1920 with a pixel ratio of 3. This calculates to an interpret resolution to 360 x 640. This way, a phone with high resolution would not render the content too small, and it is this resolution CSS relates to [50]. To fit the content to the devices viewport is simply by adding the meta-element in the HTML header shown below in Figure 57.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0"/>
```

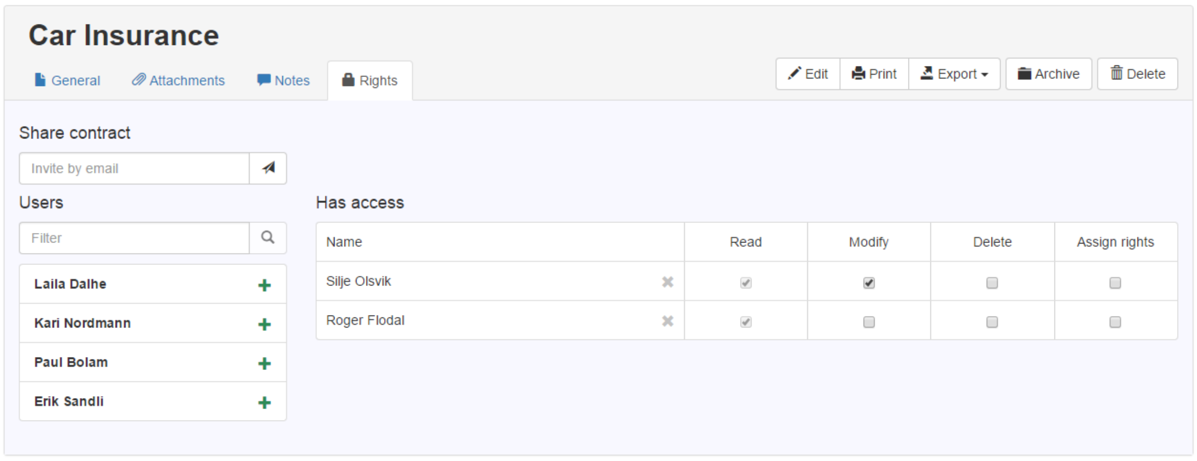
Figure 57 – The meta element which make sure the content fits on different devices. Notice width=device-width. This is taken from the header in the index.html from core product.

Not all Bootstrap components provides these options of changing its size, which led to overriding the CSS rules adjusting it to comply with our requirements. An example of this is the dropdown menu from the account button shown in Figure 53. Originally, the menu items were 26 pixels in height, but has been adjusted to 40 pixels. The reason for not going bigger was because a list of these buttons would take up too much space. For instance, having many finger friendly buttons displayed at once, the space for content gets limited on small screens. That being said, finger-friendly buttons is not always practical, as discussed in [51].

```
ul#account-list > li > a {
 padding: 10px 20px;
}
```

Figure 58 – Shows a CSS rule for the account list for adjusting the clickable area. This is from the main.css file from the core product.

Adjusting the size of a button is done in CSS by changing the padding attribute. This is done to all anchor elements within the `account-list` as shown in Figure 58. The first value represent top and bottom padding, while the second value represent left and right padding. In other words, the height of the button becomes the font size in height + top and bottom padding. If the customer decides to change how all buttons are at default, he has to make changes to the Bootstrap framework.



Laila Dalhe	+
Kari Nordmann	+
Paul Bolam	+
Erik Sandli	+

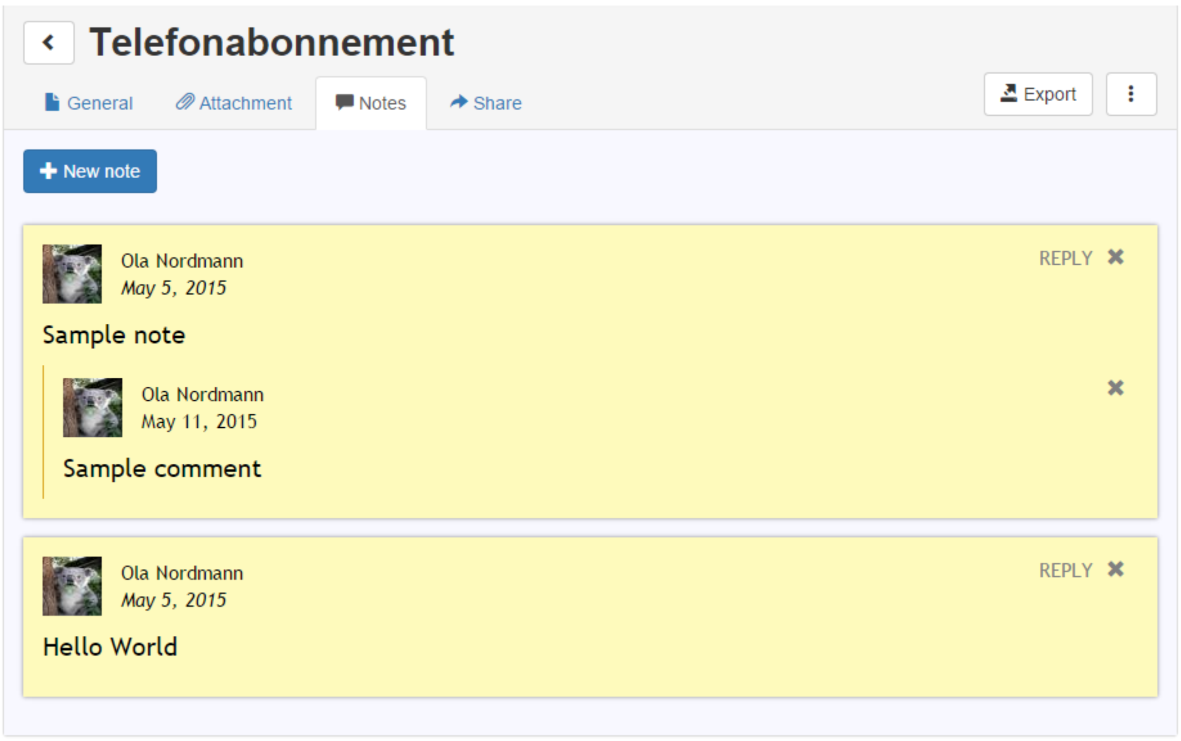
Name	Read	Modify	Delete	Assign rights
Silje Olsvik	✓	✓	☐	☐
Roger Flodal	✓	☐	☐	☐

Figure 59 – Displaying the clickable area in light green to support finger friendly buttons.

Mental Model

We have also been concerned with keeping a user interface that tells the user how the system works as simple and correct as possible. This way the user would have a more correct image of what happens behind the scene and then have a better understanding of how to operate it. For instance, in Figure 59 those who have access to a contract and those who do not is separated. Those with access also has the read privilege by default. In other words read is the same as having access to the contract, but by splitting them into two distinct groups, it gives an easier understanding of who has access and who do not. This could have been solved differently by only having the list of privileges, where those who do not have access at all, would have all unchecked. But again, this solution lacks an easy illustration of who has access. This has also been used in managing all members in an organization, those who is active and those who have their account frozen [52].

Using known metaphors for our target audience is a good practice as they would already relate to it when first encountering such features. When viewing notes to a contract, post-it notes has been used as a metaphor. This was achieved by making the notes appear as a square shape with a yellow background color. This can be seen in Figure 60.


Telefonabonnement

Ola Nordmann
May 5, 2015
Sample note

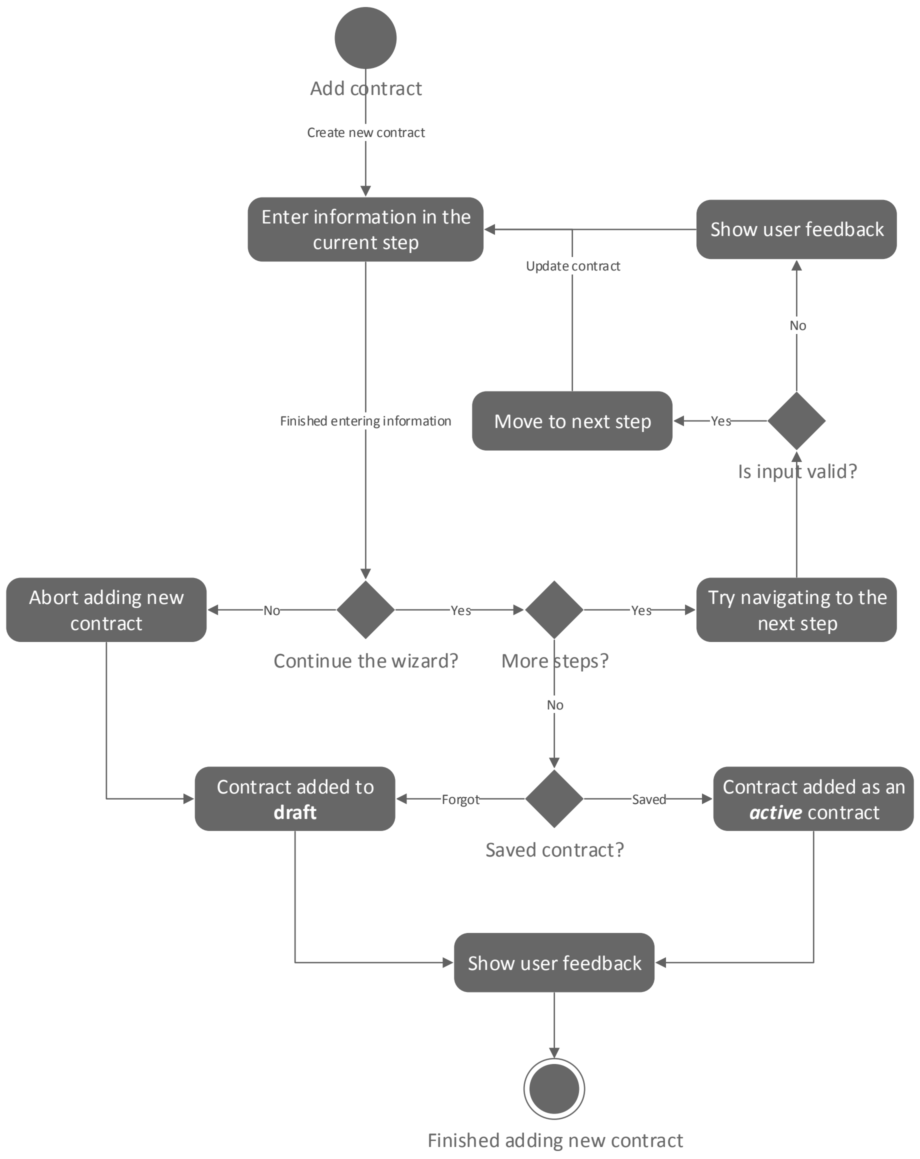
Ola Nordmann
May 11, 2015
Sample comment

Ola Nordmann
May 5, 2015
Hello World

Figure 60 – How the post-it metaphor is used for notes. One can also notice that the notes are displayed on a small device as a lot of the contract options is hidden into the vertical dotted button.

Adding a Contract Using the Wizard Pattern

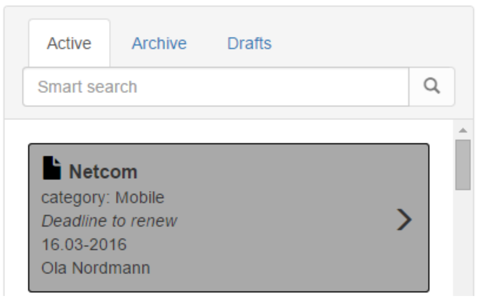
A contract contains multiple inputs representing different aspects of a contract like involved parties, dates, and adding attachments to name a few. As a lot of choices can lead to not choosing at all (also known as the paradox of choice [53]) to minimize the number of input fields dividing the goal into multiple steps for adding a new contract were done. This way, optional inputs would more likely to be filled in as well. The wizard pattern solves this problem, where the balance between number of steps and how much each step requires to continue is the key to encourage the users to fill in as much information as possible. It is recommended between 3 and 10 subtasks with not too many steps [54]. By categorizing all inputs resulted in a total of 6 steps where the last step is a summary of the contract and the user can double check the inputs before saving it as an active contract. The process of going through the wizard is illustrated in Figure 61.



Add contract
Create new contract
Enter information in the current step
Show user feedback
Update contract
No
Finished entering information
Move to next step
Yes
Is input valid?
Abort adding new contract
No
Yes
Yes
Try navigating to the next step
Continue the wizard?
More steps?
No
Contract added to **draft**
Forgot
Saved
Contract added as an ***active*** contract
Saved contract?
Show user feedback
Finished adding new contract

Figure 61 – Showing the process of going through the wizard and illustrating the states the contracts can end up being in.

A contract can have one of three different states: active, expired, or draft. Contracts are grouped together based on their state, which is represented through tabs, as seen in Figure 62. Contracts within the active tab is ongoing contracts, and expired contracts is available under the archive tab. The last one is contracts that has not been completed through the wizard, and has therefore been marked



Netcom
category: Mobile
Deadline to renew
16.03-2016
Ola Nordmann

Figure 62 – Under my contracts, one can change which group of contracts is displayed in the list.

as draft. Therefore, users does not need to re-enter contracts which may take several minutes.

According to the user survey (See chapter 8.5 User Testing), adding contracts was one of the features that would be most used. The third party plugin called angular-wizard met the requirements for a wizard and minimized the workload against having full control over the appearance. It is easy to extend the wizard and modify a step since they have their own HTML file. How this looks in HTML is shown in Figure 63 where the first `wz-step` element tag is expanded showing the `ng-include` element that loads the HTML for that current step.

```
<div id="addContract" data-ng-init="init()">
 <wizard class="well col-xs-12 col-sm-12 col-md-10 col-lg-10">
  <!-- This step covers contract categories & the people involved -->
  <wz-step title="{{slideNames[0]}}" canexit="validateSlideGeneral">
   <ng-include
     src="'modules/addContract/views/general_information_slide.html'">
   </ng-include>
  </wz-step>
  <!-- This step covers date & time information -->
  <wz-step title="{{slideNames[1]}}" canexit="validateSlideTime">
  <!-- This step covers contract attachments -->
  <wz-step title="{{slideNames[2]}}" canexit="validateSlideAttachment">
  <!-- Notification settings -->
  <wz-step title="{{slideNames[3]}}" canexit="validateSlideNotify">
  <!-- Write a note for this contract -->
  <wz-step title="{{slideNames[4]}}" canexit="validateSlideNotes">
  <!-- Edit. Any final adjustments? -->
  <wz-step title="{{slideNames[5]}}" canexit="validateSlideConfirm">
 </wizard>
</div>
```

Figure 63 – Shows the angular-wizard plugin in action. This starts with the wizard element tag.

The wizard partially uses the breadcrumb pattern as one cannot skip steps. This is represented by the timeline placed in the bottom of the wizard showing each step at all times where they have both a name and a dot indicating its state with different colors. Green represent a finished step, dark grey the current, red indicates invalid input, and light grey being next steps. This is displayed in Figure 64 where the current location is the time and date step.

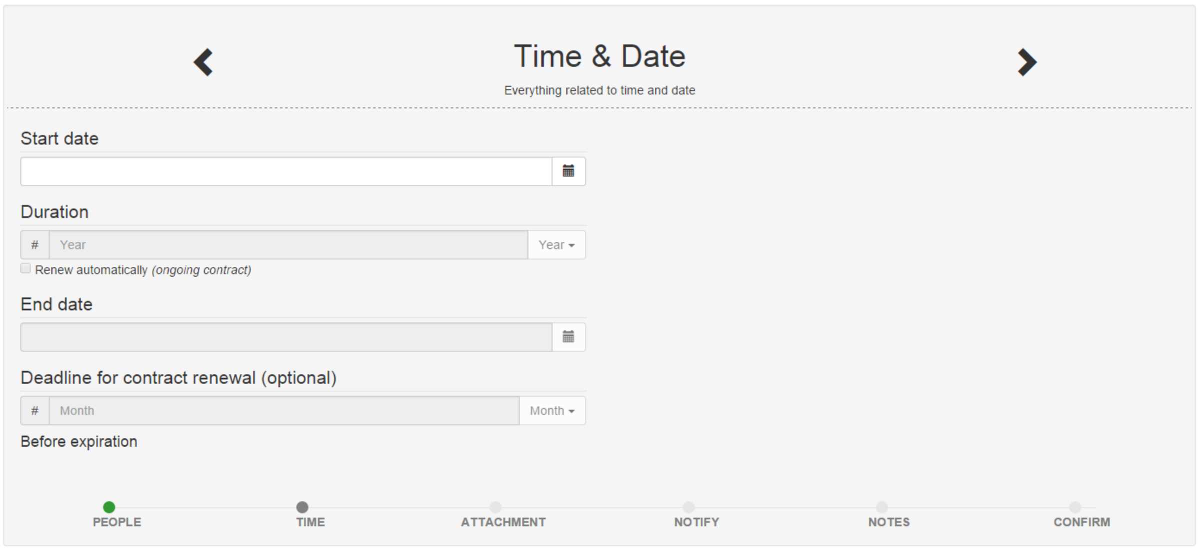


Figure 64 – The wizard currently on the second step as the timeline indicates.

The first input that is required is the start date to make the other inputs enabled. The length of the contract can be chosen in two ways, either setting the end date, or provide with a number indicating how long it lasts, either in years, month, weeks or days. As not all contracts has the option of being renewed the last input is optional representing the deadline before the contract can be renewed. If the contract has no restriction about this, the input could be one day, stating that the contract can be renewed the day before it expires.

The option of making the contract automatically renewed is provided with a checkbox. While this is checked the end date input is hidden with a slide effect, and the context of the last input changes from being about renewal deadline to the deadline for unsubscribing.

Angular has a Bootstrap plugin named angular-bootstrap. This has provided us with a fully customizable and flexible date picker which support localization of the date format. This also uses the default Bootstrap styling which keeps our look and feel consistent. As the HTML 5 date picker had some compatibility issues with Internet Explorer, the angular-bootstrap became a better alternative. Although, the plugin states it is flexible, viewing it on extra small screens could cause unwanted horizontal scrolling. By removing padding and limiting empty space we

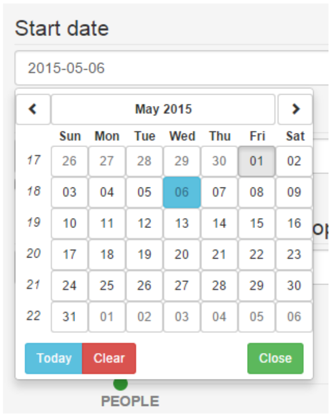
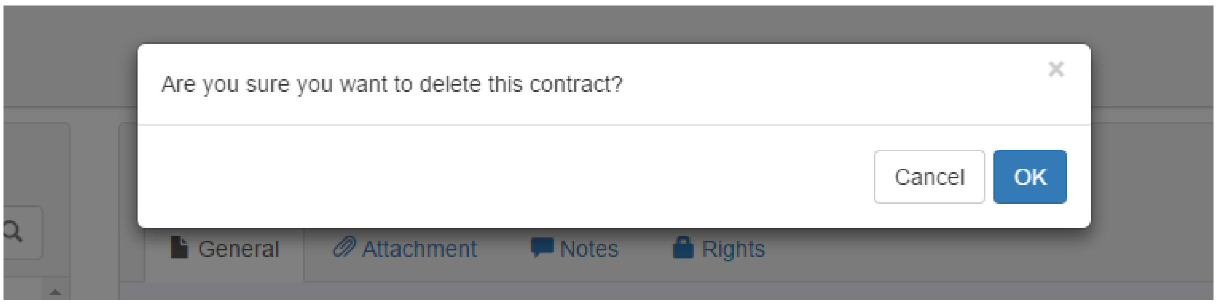


Figure 65 – The angular-bootstrap flexible and customizable date picker.

manage to scale even further down to smaller screens. This date picker is illustrated in Figure 65.

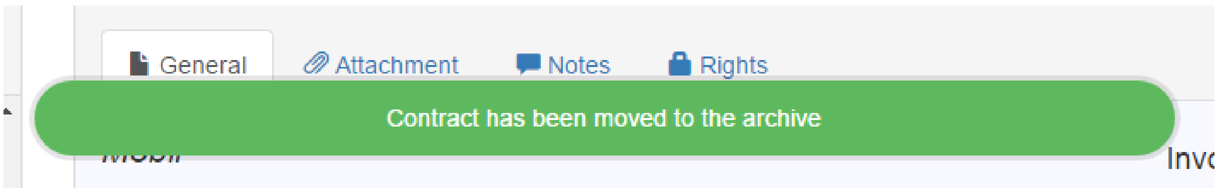
User Feedback

When a user interacts with the system, feedback is important to prevent confusion regarding results from performed actions. We have used two separate plugins for feedback, one is a dialogue box giving the user the option to confirm or cancel the action. This type of feedback is used where the outcome is not reversible, like deleting a contract. The other plugin is simpler, this just shows a message on the screen stating that an action succeeded or failed. These plugins in action is displayed in Figure 66 and Figure 67 respectively.



Are you sure you want to delete this contract?

Figure 66 – The ngBootbox plugin in action showing whether the user wants to delete the contract or not.

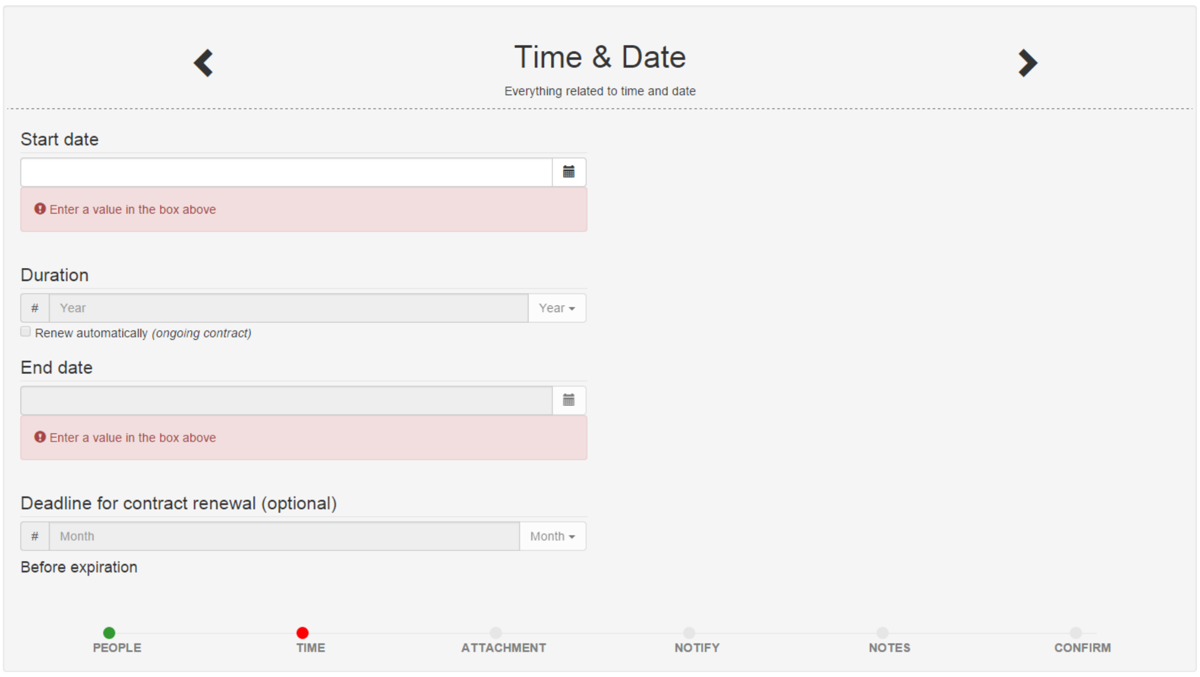


Contract has been moved to the archive

Figure 67 – Showing a success message using the ng-feedback plugin.

In cases were an action reflects what is being displayed on the screen, none of these types of feedback is used. The reason is that the changing content is feedback itself. An example of this is deleting a category, as one would immediately see that the category disappears from the list of categories. However, the API does not allow deleting categories that has contracts assigned to it, and since a category is created by entering its name, a dialogue box to confirm this deletion would therefore not be necessary.

We also have a third method of dynamically showing feedback to the user. These are mainly related to form validating where the user have missed one or more required input. In Figure 68, the user have tried to navigate to the next step in the wizard while leaving both the start and end date empty. This is however done with Angular and not by any third party plugin.


Time & Date

Everything related to time and date

Start date

Enter a value in the box above

Duration

Renew automatically *(ongoing contract)*

End date

Enter a value in the box above

Deadline for contract renewal (optional)

Before expiration

Figure 68 – Showing feedback on required input before the user can navigate to the next step in the wizard.

7.5 Internationalization

Internationalization (also referred to as i18n) is a large field and is important in applications that is going to be adapted to different languages. In Contract Manager, i18n was a system requirement that came quite early in the planning process and has been a continuous process through large parts of the development process. The customer wanted two languages to be initially supported in the system: Norwegian and English.

Before displaying the localized text, the system would have to know what language the user would prefer. Several options were considered for identifying the preferred language. One approach were to identify the preferred language through JavaScript APIs, a disadvantage for this approach is that there is no standardized call for getting the language which means that different calls for different browsers is required [55]. Another approach is to estimate the user-location based on the IP-address [56]. This would be highly unreliable as if a non-English speaking user is in Norway on vacation, the system would select Norwegian even though the language on their computer is set to English. A third approach is to look at the HTTP accept-language header on requests [57]. Using this approach, the API would easily detect what language the user would prefer. It also contains several languages that the user would prefer, so that the system can select a language that is known to the user if supported.

Language detection relies on the HTTP accept-language header value as it provides a standardized language selection. This way, the language selection can happen on the API. This approach compiles with REST principles as the responses from the API depends on the state of the client that is represented in the HTTP request. The user language is set to the preferred language based on the accept-language header value upon user registration. After this, the user can override the preferred language by going to settings and select a different language. Before the user is signed in, the language is decided by the return value of a resource that would return the preferred language for the user. The example bellow (Figure 71) shows how a request is made to get the preferred language together with the response body in JSON from the API:

```
GET http://localhost:8080/ContractManagerAPI/languages/preferred
Response body:
{
    "language": "en-us",
    "name": "English (US)"
}
```

Figure 69 – How the preferred language is retrieved from the API together with the response body.

After determining the language of the user, the user-interface needs to update and show localized strings in the returned language from the API. There was considered two options for doing this. One approach was to have a CGI-script that would replace strings in the HTML files that Angular uses. The second way was to use binding in Angular in order to show the strings where each string would be found in one JSON document for each supported language. We decided to use bindings in Angular, as having server-scripts that would modify the HTML files would go against some of the benefits by having a SPA discussed under chapter 5. System Architecture. The following HTML snippet (Figure 70) is an example of how strings are bonded into the HTML using Angular.

```
<li>
 <a href="" ng-click="logoutUser()">
  <span class="glyphicon glyphicon-log-out"
      aria-hidden="true"></span>
  {{langPack.main.logout}}
 </a>
</li>
```

Figure 70 – How strings are displayed in the HTML code in `index.html` under core product.

The `langPack` variable is set using the following JavaScript function from `mainController.js` in order to load the language. (Figure 71)

```
scope.loadLanguages = function(callback) {
 if (sessionFactory.getCurrentLanguage() == null) {
  languageProvider.getPreferredLanguage()
   .$promise
   .then(function(lang) {
    stringLibrary.getLanguagePack({
     lang: lang.language
    }).$promise
     .then(function(pack) {
      scope.langPack = pack;
      if(angular.isFunction(callback)) {
       callback();
      }
     });
  });
 } else {
  stringLibrary.getLanguagePack({
   lang: sessionFactory.getCurrentLanguage()
  }).$promise
   .then(function(pack) {
    if (angular.isFunction(callback)) {
     scope.langPack = pack;
     callback();
    }
  });
 }
};
```

Figure 71 – How the `langPack` instance is set in `mainController.js` under core product.

The language that is loaded is determined by whether the user has an active session or not. An active session will always have a language associated with it, which is returned by `sessionFactory.getCurrentLanguage()`. The `loadLanguages` function is called once the web app is initialized, when the user changes language and when the user signs in.

7.6 Hybrid Apps

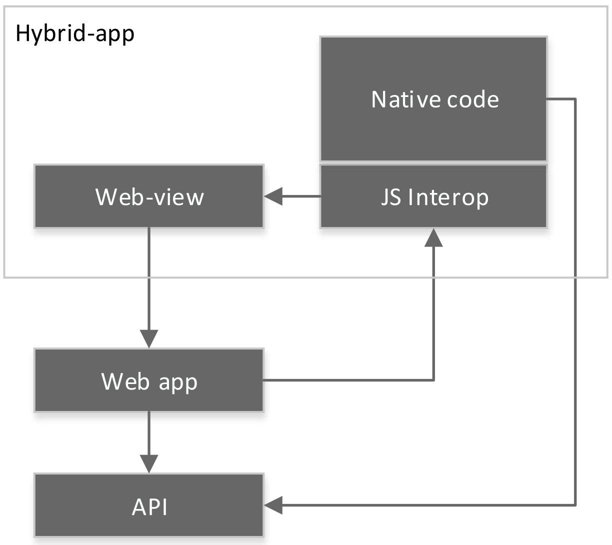


Figure 72 – Architecture of the Android and Windows hybrid apps.

Both the Windows and Android hybrid-apps follow the same architecture (Figure 72). They both use a web-view, which embeds the web-app through the Internet. Each hybrid-app can interact with the JavaScript code inside the web-app through a JavaScript interop. JavaScript interop is a bridge between the web app and the hybrid-app for communication between these two. The web-app can also interact with the hybrid-app by sending a message to the hybrid-app. This is necessary as the web app needs a way of telling the native part of the hybrid-app when to invoke certain functions, for example starting the camera when the user wants to upload a picture of a contract.

```
$scope.takePicture = function (contractId) {
    if ((window) && (window.external)
            && ("notify" in window.external)) {
        window.external.notify("PIC:" + contractId);
    }
    if (typeof Android != 'undefined') {
        Android.takePicture($scope.contract.id);
    }
};
```

Figure 73 – A code snippet of how a message is sent to the different apps when the user wants to upload a picture of a contract.

Figure 73 demonstrates how a message is sent to the native part of the hybrid applications. The main difference between Windows and Android is that Windows operates with strings and Android operates with exposing native methods through the global variable `Android` in

JavaScript. Functions are then exposed by annotating them with `@JavascriptInterface`, as seen in Figure 74.

```java
@JavascriptInterface
public void takePicture(int contractId) {
    Log.w("SESSION takepic:",
            contractManagerSession.getSessionKey());
    Intent cameraIntent =
            new Intent(mActivity, CameraActivity.class);
    cameraIntent.putExtra("contractId", contractId);
    cameraIntent.putExtra("sessionKey",
            contractManagerSession.getSessionKey());
    mActivity.startActivity(cameraIntent);
}
```

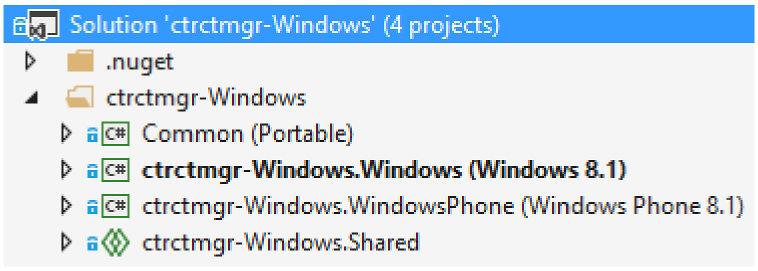
Figure 74 – The Java function called when the Javascript function `Android.takePicture(contractId)` is called. Code snippet is from the class `HybridAppInterface`.

On Windows, the received message has to be parsed and then invoked on the appropriate message handler. Each call is received through events in C# that is registered on the web-view (Figure 75). Events is a callback function that is called when a certain events occurs, it can be seen on as a language-level implementation of the observer pattern as an event can have several event handlers registered on itself. When the message for taking a picture is received (e.g. “PIC:1”), the code will attempt to find the message handler for the received message header (PIC) and pass the parameters to the handler, in this case “1” which is the contract id. The message handler would then start the camera on the device then later on upload a picture if a picture was taken to the API.

```
public AppHelper(WebView view, ICameraProvider cameraProvider)
{
    this.view = view;
    this.messageHandlers = new Dictionary<string,IMessageHandler>();
    this.messageHandlers.Add("LOGIN", new LoginHandler());
    this.messageHandlers.Add("FB", new FacebookLoginHandler());
    this.messageHandlers.Add("PIC", new PictureMessageHandler(cameraProvider));
    this.messageHandlers.Add("LINK", new ExternalLinkHandler());
    this.messageHandlers.Add("LOGOUT", new LogoutHandler());

    this.view.Navigate(new Uri(applicationUrl));
    this.view.ScriptNotify += ExternalEventReceived;
    BackgroundTaskHelper.RegisterTasks();
    Debug.WriteLine("AppHelper initialized");
}

private void ExternalEventReceived(object sender, NotifyEventArgs e)
{
    Debug.WriteLine("Message from window.external.notify: {0}", e.Value);

    string header = e.Value.Split(':')[0];

    IMessageHandler handler;
    if (messageHandlers.TryGetValue(header, out handler))
    {
        Debug.WriteLine("Invoking message [{0}]", header);
        handler.Invoke(this, e.Value.Substring(e.Value.IndexOf(":") + 1));
    }
    else
    {
        Debug.WriteLine("Unknown message: {0}", e.Value);
    }
}
```

Figure 75 – Callback function and the registering of message handlers for received calls from `window.external.notify` in the Windows app. Code snippet is from the class `AppHelper`.

One challenge with the Windows app is that the Windows Phone and the Windows APIs for invoking the camera are different. As mentioned earlier, the Windows application is a portable Windows application. This means it can run on both Windows and Windows Phone. All common code between each platform is added to two projects (modules) named “Shared” and “Common” (Figure 76). Platform-specific APIs such as camera is then wrapped by using an interface inside the projects specific to each platform. In Windows 10, Microsoft unifies all the APIs to one, which would lead to only one project for Contract Manager [58].



```
Solution 'ctrctmgr-Windows' (4 projects)
  .nuget
  ctrctmgr-Windows
    Common (Portable)
    ctrctmgr-Windows.Windows (Windows 8.1)
    ctrctmgr-Windows.WindowsPhone (Windows Phone 8.1)
    ctrctmgr-Windows.Shared
```

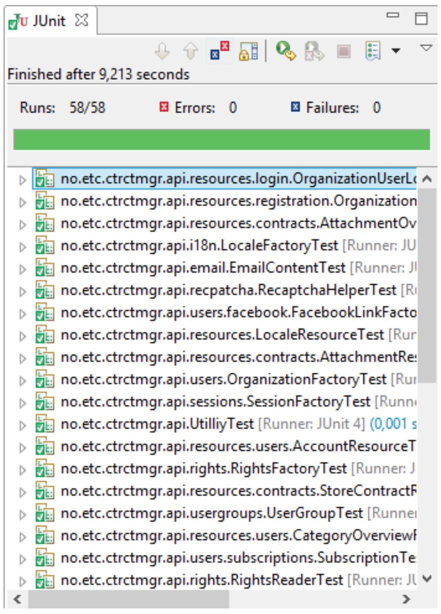
Figure 76 – Project structure of the Windows application.

When the user starts the application and signs in, the session key-id token is sent to the native part of the application. The native part will then register the device as a push device with the API. On every API-call the native part performs has the session key-id in the URL for authentication. This token is then removed when the user signs out in the application.

8. QA

8.1 API

We saw early on in the project that tests on the API would be the most important component in the system to test, as it is the most central component in the system. Unit tests were written using the JUnit framework as this was the framework we had most experience with in Java. The goal of the unit tests were to have a test on each single component unless the test would become too expensive to create. There are for example no unit tests on external services such as email and push messages as testing these modules would require a lot of code and would easily introduce new dependencies in the project. All the tests were executed in Eclipse, as seen in Figure 77.



```
JUnit
Finished after 9,213 seconds
Runs:   58/58        Errors:   0        Failures:   0

no.etc.ctrctmgr.api.resources.login.OrganizationUserL
no.etc.ctrctmgr.api.resources.registration.Organization
no.etc.ctrctmgr.api.resources.contracts.AttachmentOv
no.etc.ctrctmgr.api.i18n.LocaleFactoryTest [Runner: JU
no.etc.ctrctmgr.api.email.EmailContentTest [Runner: J
no.etc.ctrctmgr.api.recpatcha.RecaptchaHelperTest [R
no.etc.ctrctmgr.api.users.facebook.FacebookLinkFacto
no.etc.ctrctmgr.api.resources.LocaleResourceTest [Rur
no.etc.ctrctmgr.api.resources.contracts.AttachmentRe
no.etc.ctrctmgr.api.users.OrganizationFactoryTest [Ru
no.etc.ctrctmgr.api.sessions.SessionFactoryTest [Runn
no.etc.ctrctmgr.api.UtilliyTest [Runner: JUnit 4] (0,001 s
no.etc.ctrctmgr.api.resources.users.AccountResourceT
no.etc.ctrctmgr.api.rights.RightsFactoryTest [Runner: J
no.etc.ctrctmgr.api.resources.contracts.StoreContractF
no.etc.ctrctmgr.api.usergroups.UserGroupTest [Runne
no.etc.ctrctmgr.api.resources.users.CategoryOverviewF
no.etc.ctrctmgr.api.users.subscriptions.SubscriptionTe
no.etc.ctrctmgr.api.rights.RightsReaderTest [Runner: JU
```

Figure 77 – Execution of unit tests in Eclipse.

Each test for the API attempts to focus on one single component in system. However, each test usually depends on more classes than one class being subject to the test. Each test usually involves the database and a factory to create for example a user if needed. The classes creating test objects such as users in the system is called mocked factories. There are also mocked resource factories, as some resources requires a valid session or a contract to be executed. These factories were created to avoid having to duplicate code for user-creation throughout the tests. Some examples are `MockedHomeUserFactory`, `MockedOrganizationFactory` and `MockedSessionFactory`.

Tests were written at several levels in the API. Utility classes, factories and resource classes (See 7.2 Implementation of Restlet and API) are the class categories that has been most extensively tested. The tests were written for each level, as there is often a dependency between these. Resource classes depends on factory classes, and they both often rely on utility classes. This allowed us to identify if there was a problem e.g. parsing of POST or GET parameters or if there was a problem with the database operations. Figure 78 shows how a unit test was written on factory level together with a mocked factory. Figure 79 shows another unit test on resource level.

```java
@Test
public void testSessionKeyHandling() {
    Session session = MockedSessionFactory.mockSession();

    assertNotNull(session);
    assertNotNull(session.getUser());
    assertTrue(session.getExpirationTime() > Utility.currentTimestamp());
    assertTrue(session.getExpirationTime() > Utility.currentTimestamp() + 200);


    String[] key = session.getSessionKey().split("-");
    Session sessionCompleted =
            SessionFactory.getSessionKey(Integer.valueOf(key[0]), key[1]);

    assertNotNull(sessionCompleted);
    assertEquals(sessionCompleted.getId(), session.getId());
}
```

Figure 78 – A unit test of getting session keys on factory-level.

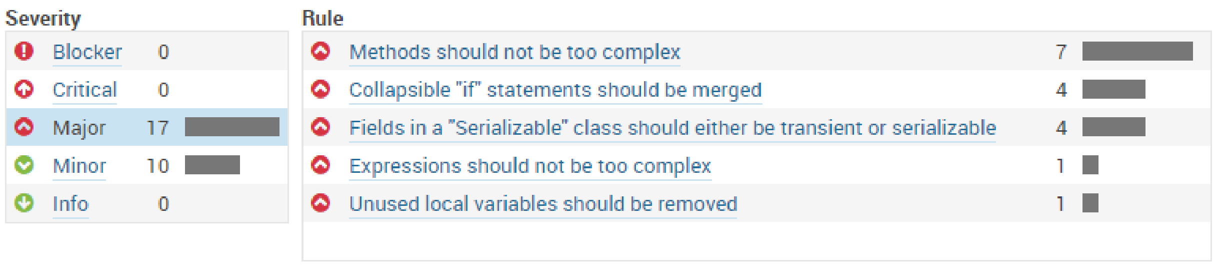
```java
@Test
public void test() throws Exception {
    LocaleResource resource = new LocaleResource();
    GsonRepresentation<List<Locale>> representation = resource.getLanguages();

    assertTrue(representation.getObject().size() > 0);
    Locale locale = representation.getObject().get(0);

    assertNotNull(locale);
    assertNotNull(locale.getLanguage());
    assertFalse(locale.getLanguage().isEmpty());
    assertNotNull(locale.getName());
    assertFalse(locale.getName().isEmpty());
}
```

Figure 79 – Test of getting available language on resource-level.

Towards the end of the project, the tests became more and more extensive. At one point, we had to consider whether we should be working on an extensive testing framework for the API or the API itself. It is possible to test every function and attempt to get every outcome of the function; however, writing tests that extensive would drastically increase the testing code. When a test did not run successfully, the cause would often be that there was minor changes in for example API parameter naming, or a change in the return value of a function. However, writing tests at this level has potentially reduced the amount of development time by e.g. knowing a contract that is added to the system can be retrieved back. The unit tests was also used as an indicated on whether the API was set up correctly on each computer used by the development team.

8.2 SonarQube

SonarQube (also referred to as Sonar) [59] is a quality assurance tool for making sure that code complies with a given set of rules. These rules are often based on language conventions. In this project, Sonar was used in the QA process of the API. Each rule that is violated is then reported as an issue through Sonar (See Figure 80). Sonar supports a wide range of languages, however we saw it sufficient to only use it on the API as there was other tools that was a better fit for the other languages. We had previous experience with Sonar from IMT3281 – Software Development and we felt it was appropriate to use Sonar as it is often used in larger Java projects.



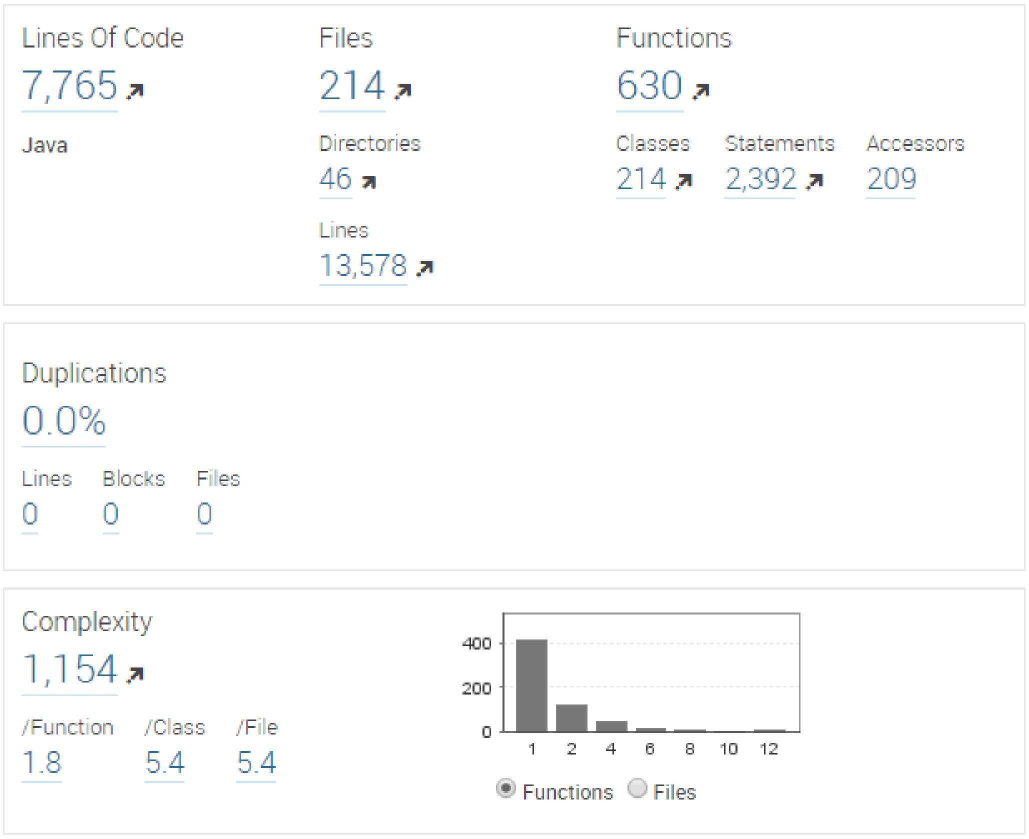
Severity	
Blocker	0
Critical	0
Major	17
Minor	10
Info	0

Rule	
Methods should not be too complex	7
Collapsible "if" statements should be merged	4
Fields in a "Serializable" class should either be transient or serializable	4
Expressions should not be too complex	1
Unused local variables should be removed	1

Figure 80 – A list of the different issues reported by Sonar.

As seen in Figure 80, there are still major and minor issues. The first issue in Figure 80 (Methods should not be too complex) is often at a function that has too many return values. Some of these functions are quite short (10-20 lines long), and complying with this rule would involve introducing unnecessary complexity in the code that would in the end make it less readable.

Sonar did also make it possible to discover some bugs. For example, if there was a parameter or a function that was not used or if there was a block of code that was commented out, it would appear as an issue in Sonar. As seen in Figure 81, Sonar would also detect duplicated code blocks and mark these as issues. This way, it would contribute to making the code more object-oriented and other insights such as how large the project currently is.


Lines Of Code
7,765
Java

Files
214
Directories
46
Lines
13,578

Functions
630
Classes 214
Statements 2,392
Accessors 209

Duplications
0.0%

Lines	Blocks	Files
0	0	0

Complexity
1,154

/Function	/Class	/File
1.8	5.4	5.4

Figure 81 – Statistics of the project size, duplications and code complexity.

As seen in Figure 82, Sonar was used about one time each month and more intensively towards the end of the project. The blue line indicates complexity and the yellow is a result of the estimated time Sonar has estimated it would take to fix the issues. Note that the complexity of the project grew due to the growth of the code-base as the project made more progress.

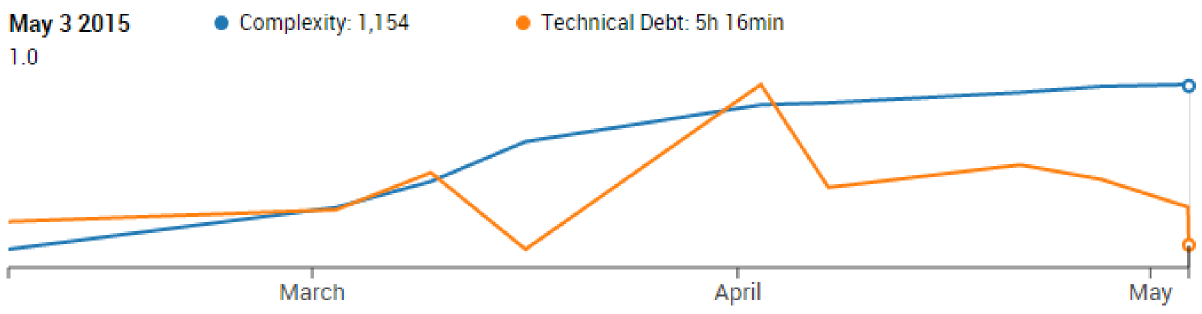


Figure 82 – History of Sonar issues throughout the project.

8.3 Angular

Refactoring

As the Core product grew in size, refactoring became necessary to maintain a lean working environment. At the early stages of development, the Core product had the same file structure as the Point of entry application, but had to move on to a more extensive structure several times during development (See Figure 20). File and folders needed restructuring as well as distributing responsibility into more specific units such as more specialized controllers, factories and services within each module.

Code Reviewing

As none of the group members had any experience using Angular before this project, it was crucial that those working with Angular shared their gained knowledge and reviewed code written by other members. This helped to utilize the Angular framework in a larger extent and create a shared perception of how it should be used.

Manual Testing

Manual testing was done mainly using Google Chrome tools such as the network log and the console. Angular provided error messages for everything related to compilation of the app in the web-console. These messages did not provide specific information about the issue at hand, but was a useful indicator to determine if the problem was a missing file or a syntax error. All scopes was monitored through the Chrome console during development, and all API calls through the network log.

Unit testing

```
  describe('$scope.changeLanguage', function() {
    var language = 'en-us';

    it('should initiate change of language...', function() {
      var $scope = {};
      var controller = $controller(
        'mainCtrl', {$scope: $scope}
      );
      spyOn($scope, 'changeLanguage');
      $scope.changeLanguage(language);
      expect($scope.changeLanguage).toHaveBeenCalled();
    });
...
```

Figure 83 – Simplified snippet from Point of entry, main.js, unit test.

For automated unit testing, the Karma test runner [60] was utilized with Jasmine [61] as the testing framework.

Figure 83 illustrates a unit test for the main controller in the point of entry application. It utilizes the Jasmine functions `describe`, `it`, `spyOn` and `expect`. These functions are used to describe what is being tested, declare specific test cases, generate behavioural information, and assertion. This test expects that the scope function changeLanguage to have been run.

We did not prioritize extensive use of unit tests in either of the Angular applications as we experienced it to be a lot more complex than first expected as the dependencies of the components to be tested would have a large impact the complexity of the test. It seemed crucial to write these tests along with or before the actual code instead of doing it later on.

8.4 JSHint

JSHint is a tool used for code inspection, like Sonar. It makes sure that standard JavaScript conventions are followed as well as customized project conventions.

JSHint was utilized in both Angular projects, installed through the NPM, and set up with Grunt. It provided feedback regarding the code whenever the command line interface command "grunt" was used.

The rules for validation was set up in the jshintrc file as displayed in Figure 84. This ruleset was applied to the entire project and reported all deviations from the given rules upon lint check [62].

```
"camelcase": true,
"curly": true,
"quotmark": "single",
```

Figure 84 – JSHint rules. Simplified snippet from jshintrc

```
app\modules\overview\overviewFactory.js
line 432  col 101  Strings must use singlequote.
line 41   col 14   Missing semicolon.
```

Figure 85 – Output from JSHint

Figure 85 illustrates a JSHint report that points to two deviations from the rule sheet. The report specifies what needs to be done in order for the code to comply with the rules.

8.5 User Testing

There was created three different test cases containing about five tasks each. These cases tested both minor and main functionalities, where the main purpose was to get subjective feedback from people outside of the development team. Another bachelor group consisting of three people tested the web application. One developer observed each test in case something went wrong. The goal was initially to get an overview of user feedback and report back to the customer for further development, however some issues were fixed.

After each test case, the testers answered four questions about the main functionalities. Questions like creating a new contract, finding a specific one, and more in general, how they

felt when navigating throughout the application. Adding a new contract was a lot to digest since they had little background about what a contract represents in our system. Knowing which step in the wizard they were currently in and what input was mandatory was difficult to know. Having the wizard scalable makes each input field also very long for big screens. When it came to finding a specific contract, the feedback was positive. One drawback that came up was that the list of contracts should been sorted from newest to oldest. As there were multiple contracts in the system, the testers used scrolling for finding a contract rather than using the search feature under the contract overview. In general, the testers reported that navigating through the app was intuitive and easy, but when viewing a contract the amount of options given became overwhelming for some. However, they appreciated the user feedback in the application.
When making a drafted contract active, the edit button was highlighted for guiding the user where this option was available. In this case, the main goal for looking at drafted contracts would be to approve them as active after checking its content. However, this made it more confusing as the users though they already were inside edit mode. After discussing the issue, it was considered as a design flaw and went back to the previous look. How it was before the fix is displayed in Figure 86.

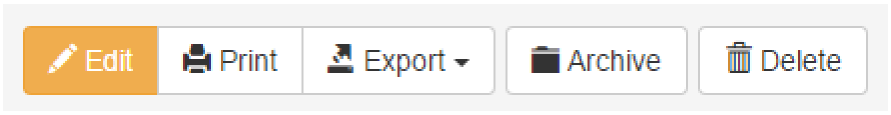


Figure 86 - Showing the highlighted edit button for guidance on drafted contracts.

In cases where there was a list of items, where each item contains an icon representing an action, the entire item should have been clickable, and not only the icon. The reason for not doing it in the first place was an attempt to prevent accidental clicks. This was an attempt to design to prevent human errors.

9. Technical Memos

9.1 Hosting Options for Java 8 Web-Applications in the Cloud

One discussion with the customer was hosting options for Java web-application (See appendix C – Meeting summaries). It was decided to look closer into which hosting options are available in the cloud. This memo describes some options for hosting an Apache Tomcat 8 container on the following PaaS (Platform as a Service) providers: Amazon Web Services (AWS), Microsoft Azure and Google Cloud.

A Tomcat container is a Java servlet container that is responsible for the interaction between the webserver and the Java servlet itself [63]. It routes HTTP calls to the desired resources in the servlet in addition to the lifecycle of the servlet.

AWS has a platform for web-applications called Elastic Beanstalk that supports a long range of different technologies. AWS has been supporting Java 8 quite early with Tomcat 8 [64]. Hosting in AWS using Elastic Beanstalk is trivial as a hosting-container can be set up in a few minutes through a wizard.

Compared to AWS, Microsoft Azure requires a bit more configuration as they are currently only providing Java 7 through their cloud portal. However, it is still possible to set up Tomcat 8 with Azure as they allow custom deployment of Java applications [65]. When the team followed this tutorial, the Java runtime would not work unless the JDK version was 1.8u0 [66]. The platform architecture did also have to correspond with the configuration on the Azure portal, as Azure allows customers to switch between x86 and x64.

Goolge Cloud supports older versions of Tomcat but not Tomcat 8. However, it is possible to use IaaS (Infrastructure as a Service) which is harder to set up and is more expensive compared to PaaS.

Provider	**Platform**
Google Cloud	Tomcat 7
Microsoft Azure	Tomcat 7/custom deployment
Amazon AWS	Tomcat 8

Table 5 – Different Java platforms for Google, Microsoft and Amazon.

In conclusion, it seems that it is good support for Java web-applications among cloud providers. Although Java 8 and Tomcat 8 is quite new in terms of Java platforms, it may take some time before every cloud provider supports the newest versions. It is also worth mentioning that hosting on an IaaS stack would give more flexibility compared to PaaS although it would require more configuration.

9.2 Secure Storage of Passwords

Storing passwords in a secure way is crucial when storing passwords. There are however, many ways to store passwords. This memo discusses methods for storing passwords in a secure manner.

Encrypting passwords is one way of storing passwords, but is considered as insecure as it is a two-way function. It is however, important not to confuse encryption with hashing, which is a one-way function. Password hashing is therefore used instead of encryption as it makes it far more complicated for an outsider to obtain the clear-text value.

There are several well-known hashing functions such as SHA-1, MD5 and the SHA-2 family [67]. A good hashing function for storing passwords should have the following criteria:

- It should be slow in order to prevent brute-force attacks, while at the same time not be too slow that would result in a resource hog.
- Generate unique hashes for each user to prevent dictionary attacks. Two users with the same password should therefore not end up with the same password hash.
- Follow a standardized and well-known hashing function that has been proven secure.

Both SHA-1 and MD5 are considered as cryptographically broken due to the fact they are vulnerable to collision attacks [68] [69]. This means it is possible to find values that would produce the same output value of a hashing function effectively.

One commonly used strategy is to use salts and peppers. The pepper can be used in order to increase the complexity of the password and prevents dictionary attacks by adding a string at the end of the password. Peppers should therefore be stored in a secure location and not in the database. Salts should be a unique value for each user that should be used to initialize the hashing function. The salts would prevent a dictionary-attack, as the brute-force attacker would have to generate a rainbow-table for each user.

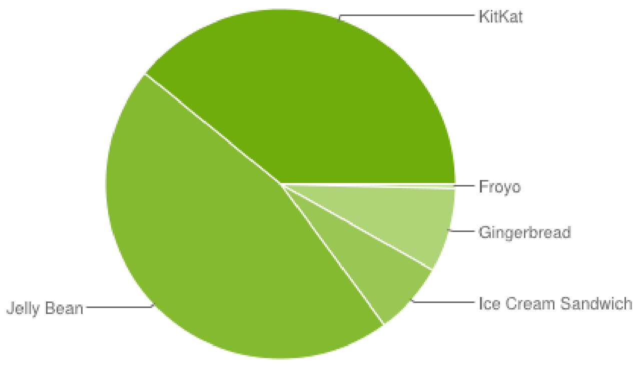
Some hashing algorithms supports taking a value that initializes the hashing function, as described above with salts. One example is bcrypt [70] that is a key derivation function, which can be used for hashing passwords. It is an adaptive function, meaning it is configurable how expensive the hashing process will be. It is also possible to use hashing

functions such as sha-256 and sha-512, but they are not derivative functions and does therefore not provide the same flexibility as bcrypt.
When storing passwords, bcrypt is a function that should be highly considered. It is still not considered as cryptographically broken and is well known for a quite secure way for hashing passwords. That being said, it is not possible to achieve a 100% secure way for hashing passwords.

9.3 Platform Selection

The type of platform will have an impact on who will use the application and how they will use it. This project aims at the web and a native mobile application, enabling multiplatform support while at the same time having application that targets specific platforms. During the first meetings with the customer, the customer suggested that the focus should be on the Android and Windows platform due to limited resources and knowledge about each platform. As the iOS platform required developer license and extra equipment (See 1.3 Scope) this was a platform we decided not to develop for.
By writing a Windows application targeted at Windows 8.1, it would make it possible to use the same code-base both on mobile and on other devices. Windows 10 is also an option; however, Windows 10 is currently in early preview stage. A better approach would be to first aim at Windows 8.1 and then later upgrade to Windows 10 when needed, as it is easier to upgrade an application together with new framework releases.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	7.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.7%
4.1.x	Jelly Bean	16	19.2%
4.2.x		17	20.3%
4.3		18	6.5%
4.4	KitKat	19	39.1%



Android usage statics from [71].

Choosing which Android API to aim at limits the user-reach of the application. By using a lower (older) API, the application would reach more users. On the other hand, lower API level would involve more testing and often more code in order to support different APIs. As API level 15 and lower only makes up 14.9% of the Android market share in the statistics above, a good approach would be to aim at API level 16 and higher.

A hybrid application would require API level 17 that would target 65.9% of the Android market.

Mentioned in 1.3 Scope, the web application should aim at browsers supporting HTML5. Because of this, users running Windows XP with Internet Explorer 8 would not be supported. Web application development often involves multi-platform support. However, some users are nevertheless excluded as of newer web standards and obsolete browsers. Enforcing legacy browser support would limit which frameworks that can be used in addition to extra testing. Since users that are still running Windows XP are advised to use other browsers such as Google Chrome or Firefox because of security reasons, dropping support for Internet Explorer 8 and older would be reasonable.

10. Summary

10.1 Results

As discussed under 4. Project Planning, most of the planned functionality for this system has been implemented. There was however features such as encryption, two-way authentication and OCR that became down-prioritized as there was not sufficient time to implement these. On the other hand, Contract Manager provides a baseline for contract management where all the fundamental features are ready from the project backlog. What is left to be done before the system can be deployed for customer is further discussed under 10.2 Further Development.

Contract Manager provides an alternative to having contracts in drawers and folders where there is often no system or structure. It makes it possible for users to find contracts far more efficiently than finding a contract in a drawer. Users does no longer have to be on the same location as they access their contracts through the Internet. Although this introduces new aspects in terms of security, the system has had focus on using robust authentication mechanisms and widely used, well-known password hashing functions.

By using frameworks such as Hibernate, GSON and Angular, modifications to existing system requirements has been rapidly implemented due to how modular this system has become, enforced by the frameworks listed above. If a new field has to be added to a contract, in the API, the field is added as a member in the contract class, getters and setters are created in addition to modifications on the creation and updating of a contract. In Angular, minor changes has to be made in the wizard and the displaying of a contract in order to display the new attribute.

Due to time restrictions, some modules does not have unit tests for it. For example, the Angular applications lacks sufficient tests. Since Angular was a new framework we did not have any experience with, understanding the fundamentals and focusing on developing the product itself became more prioritized than understanding how unit tests should be written in Angular. This also applies to the Windows application. On the other hand, there was more focus on writing sufficient tests on the API, as this became the most central component in the system.

If the project were to be developed from scratch again, we would still have used much of the same frameworks and technologies that is currently used. That being said, there are still implementations that should be done differently. There should be more focus on parameter validation to the API on an abstract level since much of the code in the resource classes today is code that validates the input data. Authentication standards such as OAuth 2.0 should also be considered, as mentioned under 7.2 Implementation of Restlet and API. Now that we are more familiar with Angular, there would be some structural differences and less refactoring compared to in this project.

10.2 Further Development

As mentioned earlier, Contract Manager is currently providing a baseline of the functionality for a contract management system. Some work remains to be done in order to make it a system that is available commercially. There should be done more testing on the system in addition to extra features that is discussed previously.

There was planned how encryption could be implemented early on in the project; however, this would not work and had to re-do the idea. After this, we saw that encryption would be far more extensive to implement than first thought. Initially, it was planned to have a key that decrypts the user-data on the API. The key is unlocked by using the password of the user. However, this does not work if the user forgets their password, which would lead to the user losing access to all their data. A better approach would be to have a generated key, which the user would have to store and enter in the client in order to access the system. On the other hand, it would still require that the user does not lose the encryption key.

Like mentioned earlier, OCR was another feature that was not implemented due to prioritization. It can however be implemented by identifying the received data when an attachment is received and then run the uploaded file content through an OCR library which would attempt to identify the text. This text can then be searchable by adding the text to a new member in the attachment entity.

The modularity of this system has been heavily affected by using the different architectures and frameworks. Having a RESTfull and microservice architecture has enforced weak coupling between different components in the application. For example, the API is not dependent on the web application, meaning a new system can easily start integrate the system, independently on the SPA in Contract Manager.

Better documentation on the API would be required if another system were to use our API. Although the API is documented using JavaDoc, there is no documentation on the HTTP-calls and responses to the API. The web-app is developed by tight cooperation between the developers internally in order to comply with which data the API provides and accepts. Additionally, there should be a way of specifying input data restriction on a more abstract level as much of the code in each resource class is data input validation. This can be implemented by creating a validator, which is a Restlet component for enabling restrictions on different patterns for various fields [72].

Organization administrators should also have a wizard that enables them to easily set up and get started with this system when first entering Contract Manager. This would allow the user to quickly invite users and it would make the user quickly see more use of the system than being greeted with an empty overview where the user may not understand what to do next.

10.3 Group Evaluation

During the beginning of the project, much time went onto researching on different frameworks in terms of functionality and then later on learning how to use each framework. This process lasted throughout the project.

Having one-week sprints made the team enter a weekly rhythm, as each week would contain meetings at the same day of the week. As mentioned under 4. Project Planning, we saw that having one-week sprints made it possible to quickly get feedback on the progress and quickly respond to features that has not been implemented correctly. This also forced the velocity of the project progress to be stable.

In this project, each team member had an assigned role. This made it possible for each team member to focus on their own task and have a good focus on what their role in the project is. It also played well together with how modular the system is, as someone who is working with the web app does not need to understand how Hibernate works. Another important aspect of having roles is that it made the team more agile since as team member could better understand what they were doing and therefore perform work far more efficiently than if they would have to understand how the entire system worked as one.

There has been no major conflicts throughout the project. Either through data loss, conflicts between team members or between periods where large amount of code has become irrelevant or implemented wrong. This is the result of sufficient planning in terms of backup routines, research and have a gathered team where each team member can easily discuss issues with others. There has been discussions where different views has been thoroughly discussed. These were solved by voting.

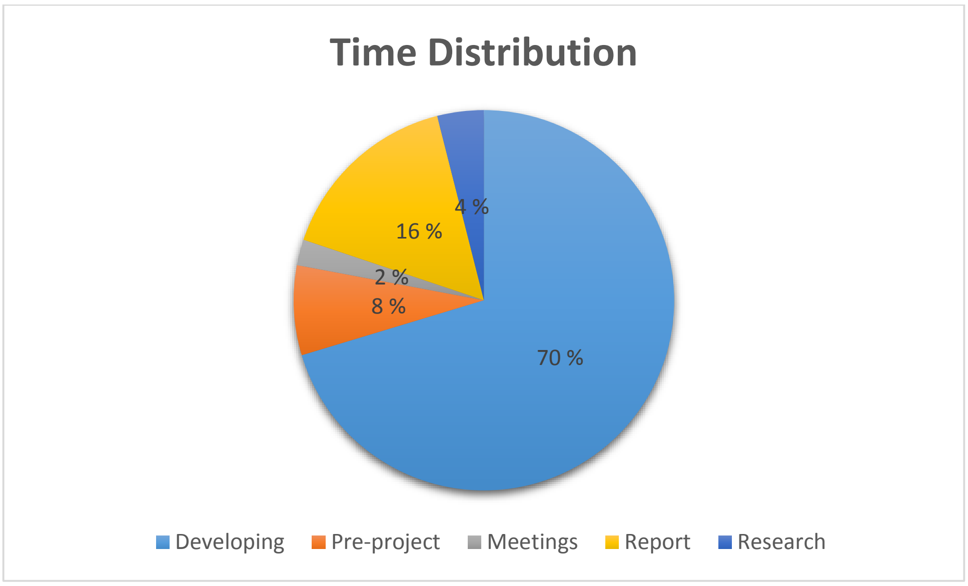


Figure 87 – Estimate of time distribution throughout the project based on the hour log.

Figure 87 shows the time distribution throughout the project, based on the hour log (see appendix B – Hour Log). In total, the project consists of about 1500 working hours. The length of each day has been stable (about 8 hours), however this was normally longer when the team had many items to complete during a sprint or was working behind schedule.

10.4 Conclusion

SPA is getting more and more relevant in web technology. Currently, there is no standardized framework for writing SPAs, however Google is currently working on a W3C specification for web components [73] which makes it possible to create a SPA with standardized HTML components without introducing any external SPA frameworks such as Angular. Additionally, SPAs works well together with the microservice architecture as it decouples the view from the business logic layer far more extensively compared to traditional web-pages.

SPA has played an important role for Contract Manager both in terms of project management and providing support for mobile devices. Having a wide range of different users that should access the same system has been challenging both while planning and implementing the system. By applying design patterns and following object-oriented practices, we managed to create a system that would appeal to our target audience that is both modular and attempts to be as user-friendly as possible.

Another challenging aspect of this project was to understand the general life-cycle of contracts. As this model was unveiled, the project grew drastically in complexity and size. Maintaining a user-interface that is easy to understand for both home users and organization users while maintaining the business aspect behind contracts was also challenging in this project.

This project has been built on the baseline of three years of studies at Gjøvik University College where knowledge and experience from almost every course has contributed. We have widened our horizon by learning new frameworks and gaining experience from a real-world scenario. Many of the frameworks that has been used in this project are widely used and well recognized in the software industry, which has been a valuable experience for the team.

In conclusion, Contract Manager attempts to solve many of the issues web-based applications face today in terms of scalability, modularity and flexibility. It has been developed by using agile system development methods, design and architectural patterns such as REST and SPA in addition to widely used and well recognized frameworks and architectures. As this project has had such a big focus on mobile devices and newer technologies such as SPA and REST, it is the reason why this thesis is titled *Contract Manager – A Modern Approach: Creating a Contract Management System Using Modern Designs and Frameworks*.

References

1. Wasson M. ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET. [Online].; [cited 29.04.2015]. Available from: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx.
2. Brehm S. Isomorphic JavaScript: The Future of Web Apps - Airbnb Engineering. [Online].; [cited 29.04.2015]. Available from: http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/.
3. Anuff E. API-Centric Architecture: All Development is API Development | Apigee Blog. [Online].; [cited 30.04.2015]. Available from: https://blog.apigee.com/detail/api_centric_architecture_all_development_is_api_development.
4. Barry DK. ervice-Oriented Architecture (SOA) Definition. [Online].; [cited 23.04.2015]. Available from: http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html.
5. James Lewis MF. Microservices. [Online].; [cited 23.04.2015]. Available from: http://martinfowler.com/articles/microservices.html.
6. Single-page application - Wikipedia, the free encyclopedia. [Online].; [cited 14.04.2015]. Available from: http://en.wikipedia.org/w/index.php?title=Single-page_application&oldid=656077250.
7. Why REST is So Popular. [Online].; [cited 14.04.2015]. Available from: https://www.serviceobjects.com/resources/articles-whitepapers/why-rest-popular.
8. Representational state transfer - Wikipedia, the free encyclopedia. [Online].; [cited 14.04.2015]. Available from: http://en.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=656323932.
9. Fielding RT TR. Principled Design of the ModernWeb Architecture. ICSE '00 Proceedings of the 22nd international conference on Software engineering. 2000: p. 407-416.

10 Restlet | Products | Restlet Framework | Features. [Online].; [cited 29.04.2015].
. Available from: http://restlet.com/products/restlet-framework/features/.

11 Google. google-gson - A Java library to convert JSON to Java objects and vice-versa -
. Google Project Hosti. [Online].; [cited 14.04.2015]. Available from:
https://code.google.com/p/google-gson/.

12 Hibernate (Java) - Wikipedia, the free encyclopedia. [Online].; [cited 14.04.2015].
. Available from:
http://en.wikipedia.org/w/index.php?title=Hibernate_(Java)&oldid=655723466.

13 Angular. Angular developer page. [Online].; [cited 29.01.2015]. Available from:
. https://docs.angularjs.org/guide/introduction.

14 entwiclertagebunch. [Online].; [cited 15.02.2015]. Available from:
. http://entwicklertagebuch.com/blog/2013/10/how-to-structure-large-angularjs-applications/.

15 Wikipedia. MIT licence. [Online].; [cited 12.05.2015]. Available from:
. http://en.wikipedia.org/wiki/MIT_License.

16 Twitter Bootstrap. [Online].; [cited 30.04.2015]. Available from:
. http://getbootstrap.com/.

17 Otto M. form validation state colors not colorblind-friendly · Issue #14744 ·
. twbs/bootstrap. [Online].; [cited 07.05.2015]. Available from:
https://github.com/twbs/bootstrap/issues/14744.

18 WebAIM. WebAIM: Designing for Screen Reader Compatibility. [Online].; [cited
. 07.05.2015]. Available from: http://webaim.org/techniques/screenreader/.

19 Mobile App Development - App Creation Software - Xamarin. [Online].; [cited
. 11.05.2015]. Available from: https://xamarin.com.

20 Apache Cordova. [Online].; [cited 04.05.2015]. Available from:
. http://cordova.apache.org/.

21 W3C. HTML Media Capture. [Online].; [cited 04.05.2015]. Available from:
. http://www.w3.org/TR/html-media-capture/.

22 W3C. Push API. [Online].; [cited 04.05.2015]. Available from:
. http://www.w3.org/TR/push-api/.

23 Polyglot Builds - Gradle. [Online].; [cited 14.04.2015]. Available from:
. http://gradle.org/why/polyglot-builds/.

24 NuGet Gallery | Home. [Online].; [cited 11.05.2015]. Available from:
. https://www.nuget.org/.

25 GruntJS. [Online].; [cited 08.05.2015]. Available from: http://gruntjs.com/.
.

26 NPM. NPM js. [Online].; [cited 11.05.2015]. Available from: https://www.npmjs.com/.
.

27 Google Answers: Using LGPL code for commercial application. [Online].; [cited
. 11.05.2015]. Available from:
http://answers.google.com/answers/threadview/id/439136.html.

28 team A. Angular documentation ng-click. [Online].; [cited 05.05.2015]. Available from:
. https://docs.angularjs.org/api/ng/directive/ngClick.

29 team A. angular $resource. [Online].; [cited 05.05.2015]. Available from:
. https://docs.angularjs.org/api/ngResource/service/$resource.

30 Pfeiffer A. Github angular-feedback. [Online].; [cited 01.03.2015]. Available from:
. https://github.com/andreipfeiffer/angular-feedback.

31 Team A. Angular route provider. [Online].; [cited 06.05.2015]. Available from:
. https://docs.angularjs.org/api/ngRoute/provider/$routeProvider.

32 ocombe. Github ocLazyLoad. [Online].; [cited 10.03.2015]. Available from:
. https://github.com/ocombe/ocLazyLoad.

33 Angular Team. Agular docs. [Online].; [cited 11.05.2015]. Available from:
. https://docs.angularjs.org/api/ng/type/angular.Module.

34 Angular team. Angular directive. [Online].; [cited 15.03.2015]. Available from:
. https://docs.angularjs.org/guide/directive.

35 Esteva S. consensus. [Online].; [cited 07.05.2015]. Available from: http://ng-
. learn.org/2014/01/Dom-Manipulations/.

36 PHP: Introduction - Manual. [Online].; [cited 29.04.2015]. Available from:
. http://php.net/manual/en/intro.session.php.

37 James P. Living Without Sessions. [Online].; [cited 11.05.2015]. Available from:
. http://www.peej.co.uk/articles/no-sessions.html.

38 Internet Engineering Task Force (IETF). RFC 6749 - The OAuth 2.0 Authorization
. Framework. [Online].; [cited 29.04.2015]. Available from:
https://tools.ietf.org/html/rfc6749.

39 Oracle. Chapter 12. Execution. [Online].; [cited 11.05.2015]. Available from:
. https://docs.oracle.com/javase/specs/jls/se7/html/jls-12.html#jls-12.4.2.

40 Method cascading - Wikipedia, the free encyclopedia. [Online].; [cited 10.04.2015].
. Available from:
http://en.wikipedia.org/w/index.php?title=Method_cascading&oldid=615215005.

41 OneToMany (Java EE 6). [Online].; [cited 29.04.2015]. Available from:
. http://docs.oracle.com/javaee/6/api/javax/persistence/OneToMany.html#fetch().

42 Landrum A. Focusing on UI/UX and The Importance of Good Design | Merge. [Online].;
. [cited 06.05.2015]. Available from: http://www.mergeagency.com/creative-design/importance-of-good-ui-ux.

43 Nygard-Hansen HP. Blog owned by Hans-Petter Nygard-Hansen. [Online].; [cited
. 09.04.2015]. Available from: http://hanspetter.info/2015/03/google-straffer-nettsider-som-ikke-er-mobiltilpasset/.

44 Novatus, Inc. novatusinc. [Online].; [cited 04.05.2015]. Available from:
. http://www.novatusinc.com/.

45 Optimus BT. eContracts for office 365. [Online].; [cited 04.05.2015]. Available from:
. http://www.optimusbt.com/econtracts-for-office/.

46 Welie Mv. Breadcrumbs - Interaction Design Pattern Library - Welie.com. [Online].;
. [cited 27.01.2015]. Available from:
http://www.welie.com/patterns/showPattern.php?patternID=crumbs.

47 Wikipeda. Fitts's law - Wikipedia, the free encyclopedia. [Online].; [cited 04.05.2015].
. Available from:
http://en.wikipedia.org/w/index.php?title=Fitts%27s_law&oldid=647665922.

48 Android. Android developer. [Online].; [cited 04.05.2015]. Available from:
. http://developer.android.com/design/style/metrics-grids.html.

49 Microsoft. Windows Dev Center. [Online].; [cited 04.05.2015]. Available from:
. https://msdn.microsoft.com/en-us/library/windows/apps/hh202889%28v=vs.105%29.aspx#BKMK_Touchtargetsandtext.

50 Vincent Hardy SG. understanding-css-units · tutorials · WPD · WebPlatform.org.
. [Online].; [cited 12.05.2015]. Available from:
https://docs.webplatform.org/wiki/tutorials/understanding-css-units.

51 T A. Smashingmagazine. [Online].; [cited 14.01.2015]. Available from:
. http://www.smashingmagazine.com/2012/02/21/finger-friendly-design-ideal-mobile-touchscreen-target-sizes/.

52 Interaction Design Foundation. Mental Models. [Online].; [cited 05.05.2015]. Available
. from: https://www.interaction-design.org/encyclopedia/mental_models_glossary.html.

53 Wikipedia. The Paradox of Choice - Wikipedia, the free encyclopedia. [Online].; [cited
. 06.05.2015]. Available from:
http://en.wikipedia.org/w/index.php?title=The_Paradox_of_Choice&oldid=649134981.

54 Welie Mv. Wizard - Interaction Design Pattern Library - Welie.com. [Online].; [cited
. 27.01.2015]. Available from:
http://www.welie.com/patterns/showPattern.php?patternID=wizard.

55 W3C. Don't call me DOM » Determining the user’s language in JavaScript. [Online].;
. [cited 14.04.2015]. Available from:
http://people.w3.org/~dom/archives/2010/04/determining-the-user%E2%80%99s-language-in-javascript/.

56 Geotargeting - Wikipedia, the free encyclopedia. [Online].; [cited 14.04.2015]. Available
. from: http://en.wikipedia.org/w/index.php?title=Geotargeting&oldid=653237571.

57 W3C. Setting language preferences in a browser. [Online].; [cited 14.04.2015]. Available
. from: http://www.w3.org/International/questions/qa-lang-priorities.en.php.

58 Gallo K. A first look at the Windows 10 universal app platform. [Online].; [cited
. 11.05.2015]. Available from: http://blogs.windows.com/buildingapps/2015/03/02/a-first-look-at-the-windows-10-universal-app-platform/.

59 SonarQube - Wikipedia, the free encyclopedia. [Online].; [cited 03.05.2015]. Available
. from: http://en.wikipedia.org/w/index.php?title=SonarQube&oldid=657238023.

60 Karma. Karma. [Online].; [cited 12.05.2015]. Available from: http://karma-
. runner.github.io/0.12/index.html.

61 Jasmine. Jasmine. [Online].; [cited 12.05.2015]. Available from:
. http://jasmine.github.io/2.0/introduction.html.

62 Wikipedia. Lint. [Online].; [cited 12.05.2015]. Available from:
. http://en.wikipedia.org/wiki/Lint_%28software%29.

63 Web container - Wikipedia, the free encyclopedia. [Online].; [cited 29.04.2015].
. Available from:
http://en.wikipedia.org/w/index.php?title=Web_container&oldid=657414748.

64 Amazon. AWS Elastic Beanstalk Supports Java 8 Tomcat 8. [Online].; [cited 15.04.2015].
. Available from: http://aws.amazon.com/about-aws/whats-new/2014/11/05/aws-elastic-beanstalk-supports-java8-tomcat8/.

65 rmcmurray. Upload a custom Java website to Azure. [Online].; [cited 15.04.2015].
. Available from: http://azure.microsoft.com/en-us/documentation/articles/web-sites-java-custom-upload.

66 Nyfløtt MS. java - Tomcat 8 on Azure websites - Stack Overflow. [Online].; [cited
. 15.04.2015]. Available from: http://stackoverflow.com/questions/27926673/tomcat-8-on-azure-websites/28221027.

67 Secure Hash Algorithm - Wikipedia, the free encyclopedia. [Online].; [cited 13.05.2015].
. Available from:
https://en.wikipedia.org/w/index.php?title=Secure_Hash_Algorithm&oldid=655252398.

68 Vulnerability Note VU#836068 - MD5 vulnerable to collision attacks. [Online].; [cited
. 13.05.2015]. Available from: https://www.kb.cert.org/vuls/id/836068.

69 Schneier B. SHA-1 Broken - Schneier on Security. [Online].; [cited 13.05.2015]. Available
. from: https://www.schneier.com/blog/archives/2005/02/sha1_broken.html.

70 bcrypt - Wikipedia, the free encyclopedia. [Online].; [cited 13.04.2015]. Available from:
. http://en.wikipedia.org/w/index.php?title=Bcrypt&oldid=644165711.

71 Dashboards | Android Developers. [Online].; [cited 29.04.2015]. Available from:
. https://developer.android.com/about/dashboards/index.html.

72 Validator (Restlet API 2.3.2 - Java Standard Edition). [Online].; [cited 06.05.2015].
. Available from: http://restlet.com/technical-resources/restlet-framework/javadocs/2.3/jse/api/org/restlet/routing/Validator.html.

73 Web Components - Wikipedia, the free encyclopedia. [Online].; [cited 06.05.2015].
. Available from:
http://en.wikipedia.org/w/index.php?title=Web_Components&oldid=659521053.

Appendices

A – Project Plan

PROJECT PLAN

ctrctmgr

Høgskolen i Gjøvik
IMT3912 – Bacheloroppgave IMT

Table of Contents

1. Introduction 1

1.1 Background 1

1.2 Learning Objectives 1

1.3 Impact Objectives 1

1.4 Performance Objectives 2

1.3 Target Audience 2

2. Scope 2

2.1 Field of Study 2

2.2 Project Restrictions 2

2.3 Project Description 3

3. Project Organization 5

3.1 Roles and Responsibility 5

3.2 Project Rules 5

4. Planning, Monitoring and Reporting 6

4.1 Software Development Methodology 6

4.2 Status Meetings 8

5. Organization of Quality Assurance 8

5.1 Documentation, Standardization and Source Code 8

5.2 Risk Analysis 9

5.3 Tools 10

5.6 Legal 12

6. Project Plan 13

6.1 Gantt Scheme 13

References 14

1. Introduction

1.1 Background

During our everyday lives, we get relations with several vendors. Typically through contracts: either a rent, insurance, or a mobile subscription. These contracts are kept in different IT systems or kept as a sheet of paper hidden in a drawer nobody knows where is. With an even increasing stream of services from the digital world: bookkeeping, managing and controlling of all the contracts becomes increasingly difficult. Organizations may often want to get an overview and find a specific insurance contract without having to use time on tracking down a specific coworker and wait for them to locate and fetch the contract while the server park is literally on fire. In other words, organizations and home users does not want to spend time finding contract.

Electronic Time Car AS (ETC) is an innovative IT-company with cutting-edge solutions for managing vehicles used by three or more drivers in small and large car pools. Dag L. Solhaug, the CEO of ETC, has recognized the need for a system where all contracts are organized and where users can archive their contracts and get notifications when these are expiring. This system should also be available on several platforms as users has an increasing use of tablets and mobile devices both professionally and at home. In addition, the system should be able to store the contract and its associated information in a safe and secure manner.

Our project, codenamed ctrctmgr that is short for contract manager, is the solution to the problems described above. It is a system designed for both private users and organizations.

1.2 Learning Objectives

- Use a wide range of frameworks and technologies.
- Understand the importance of unit tests.
- Practical usage of Scrum.
- UI design

1.3 Impact Objectives

- Create a system which both the customer and the group is happy with.
- Make the system available on the web, Android, Windows Phone 8.1 and Windows 8.1.
- Write a good project report that describes our work in-depth.
- Create a user-interface that is both easy and efficient in use.

1.4 Performance Objectives

- Create a robust and scalable system that can scale up on demand.
- Make it easier for users to systemize contracts and get notifications about their status.

1.3 Target Audience

The target audience is broad since the product aims for both the private and the corporate market in any scale. In other words, those who administrate and manage contracts. We can assume that a contract management system appeals to those of an adult age. In other words, we are not considering people under 18 as our target audience but they can still use the product. The initial release will be focusing at the Norwegian market even though we are planning the product to support internationalization and will have the English language package included in the initial release.

2. Scope

2.1 Field of Study

- Responsive web application using HTML, CSS and JavaScript
- Android mobile application development
- Windows application development
- RESTful API-design
- Java servlet
- Internationalization
- Relational database design
- Cryptology
- Unit testing
- Contracts
- OCR (Optical Character Recognition)

2.2 Project Restrictions

- The web development will aim at browsers only supporting HTML5, which is newer versions of Chrome, Firefox and Safari and minimum Internet Explorer 9.
- No iOS application will be developed as this required additional cost in terms of development license and a Mac computer.

- The Windows application development will target Windows 8.1, which makes it possible to run the same application on mobile and on desktop.
- The Android application development will target minimum API level 16 (Android 4.1) and higher in order to make the development and testing of the Android app easier.
- The system will use OCR (Optical character recognition) libraries in order to make it possible to search on text in uploaded attachments and images taken with the phone. Text from Word/OpenOffice documents or PDF documents with text will be directly extracted instead of using OCR. This processing will occur on the backend and not in a browser or a mobile device.

2.3 Project Description

In brief overview, the system will make it possible for users to store and manage contracts. This system will be available through a web application and an app for Windows and Android. Although the system can be thought of as an archive, the system will take the idea one step further. Users will get notifications either through email, the mobile app or in the system. It should be customized for both home users and organizations. Therefore, it is important to make a user-interface that is both simple and easy to understand.

The system will consist of the following main-features:

- Present contract metadata in a generalized print-friendly contract template.
- Contract archive that makes it possible for the users to store, manage and search through their own contracts.
- Attachments can be added to contracts that is often a scanned document, a picture or a PDF with the actual contract. Once the user have uploaded the document, the text is extracted from the document directly if possible or through OCR.
- It should be possible to earn money from making the system available. This is achieved through revenue from advertisement or subscriptions. There are specific subscriptions for home users and organizations that removes advertisement and adds extra security features. Subscriptions are paid through PayPal.
- As contracts are often security-critical and is not some kind of information that organizations would want everyone to know, these are encrypted using a key-chain that is in the end locked with the user password.
- Two-factor authentication for enhanced security.

- The web-app is scalable for mobile users and desktop users.
- Home users are able to sign in using Facebook.
- Overview that makes the user aware of expiring contracts and recent activity.
- The system administrator should be able to have an overview of users paying for the service.
- Contracts and overviews can be exported to a file or be printed.
- For organizations, users can be assigned to have individual read/write rights to a contract or through a group to a contract category.

The project will cover the following platforms:

- Windows 8.1 application development in C# and the .NET Framework
- Android application development
- Responsive web applications with AngularJS and Bootstrap
- Backend REST API written in Java together with MariaDB

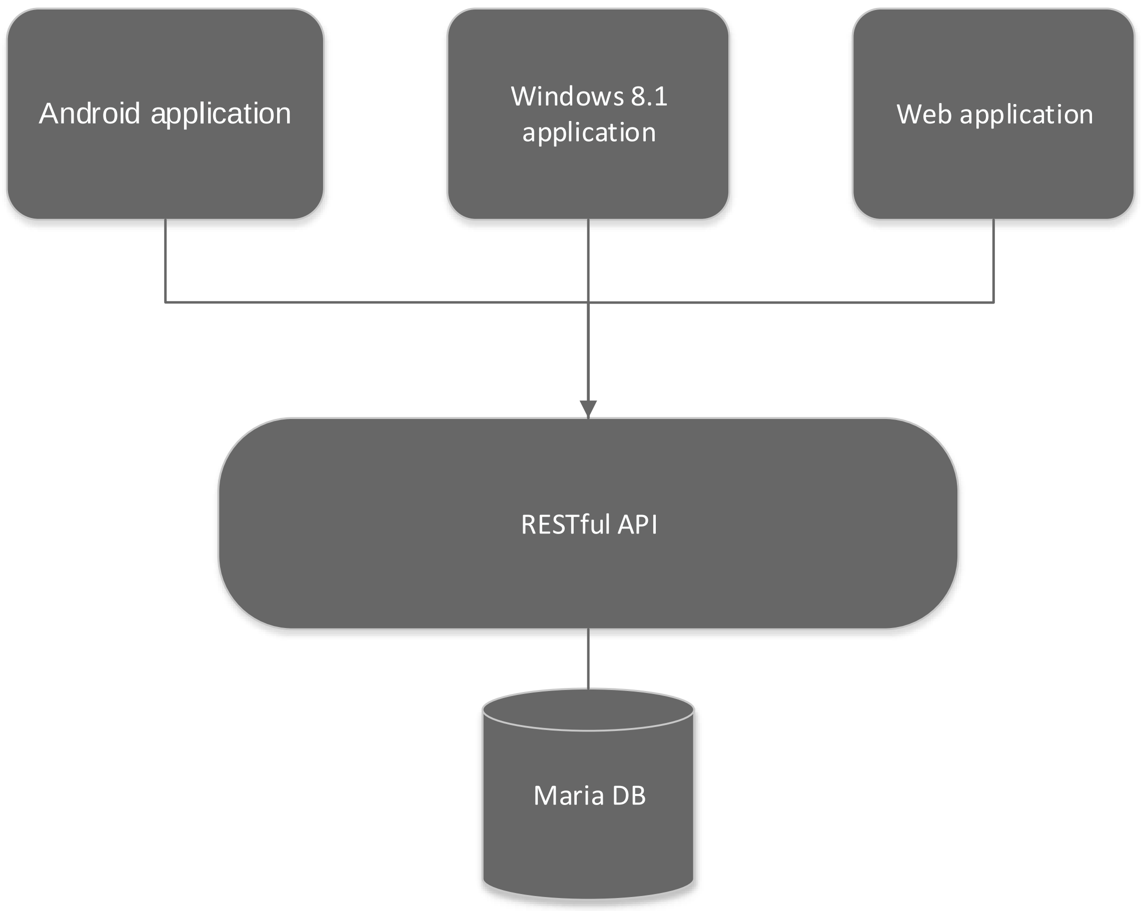


Figure 1 - System Overview.

3. Project Organization

3.1 Roles and Responsibility

Group Leader - Martin Storø Nyfløtt

See appendix A for roles regarding the group leader. Will also be taking notes from meetings with supervisor and customer and make sure these are published on the project website.

Technology Responsible - Joakim Jøreng

Design Responsible - Thomas Mellemseter

Supervisor - Tom Røise

The supervisor will be giving feedback and highlight what needs to be done. If the group leader fails to resolve issues in the group, the supervisor will be requested to resolve the given issue.

Employer/customer - Dag L Solhaug

Defines demands and limits to the software that should be developed. Will give feedback on the project progress.

3.2 Project Rules

The group has agreed upon and signed a sheet of group rules. See appendix E – Group Rules.

4. Planning, Monitoring and Reporting

4.1 Software Development Methodology

Due to rather unclear requirements in terms of functionality, we saw the need of using an agile methodology, the project size, and its natural module division was an important factor deciding what kind of methodology we wanted to use and having a product owner who has years of experience with Scrum.

The software development methodology of our project is heavily influenced by Scrum but is accompanied by a few RUP artifacts created in the inception/planning phase. Artifacts such as glossary, use case, risk analysis document and conceptual data models will be prioritized. Employer/customer Dag Solhaug is acting as product owner and the acting group leader has the role of being the Scrum master.

Our version of Scrum master consists of a simplified role with a specific set of continuous tasks that will be active during the entire project.

The reason for using a simplified and concretized version of the Scrum master is to avoid the problem having to "wear different hats". Mitch Lacey points out in the article mixing roles in Scrum that having the role combination of Scrum master and being member of the development team can be quite problematic and disastrous [1].

The following list is the tasks concerning the role of our Scrum master.

- Make sure to reserve a proper working environment for the development team during sprints.
- Set up day-to-day time schedule every week where the team will be working together. After the daily scrum on Fridays, the Scrum master must gather information about each team members schedule to achieve this.
- Lead the daily scrum meetings that will be held each day 10:00 except on Tuesday where the meeting will occur after the meeting with the supervisor and on Friday when Joakim is available after his lecture.
- Responsible for keeping backups of the task board.

The team will be using some of best practices provided by Henrik Kniberg in the book "Scrum and XP from the trenches" [2] such as seating the developer team together and using a physical task board.

This will be the initial sprint length as this was recommended by the employer. However, it may be adjusted if the group see that adjustment is needed.
Sprint review meetings will be held 12:30 to 13:00 where the development team will present the result of the previous sprint with a live demo. From 13:00 to 13:30, there will be sprint retrospective meeting where the development team will discuss what worked, what did not work and what to start doing. Sprint planning for the upcoming sprint will be held from 13:30 to 14:00 and will be used to achieve a sprint goal and which backlog items that will be included in this sprint. These meetings will be held at the ETC headquarters.

4.2 Status Meetings

The group will have status meetings with supervisor and employer every Tuesday. The group can cancel a status meeting with the supervisor or employer if needed. The group leader will take notes and be responsible for making a summary of each meeting that is later published to the project website.

5. Organization of Quality Assurance

5.1 Documentation, Standardization and Source Code

According to group rule §7 (See appendix A), all code and documentation should follow the standards and conventions for the language or the framework that is being used. All code should be documented once the core functionality is done. All Java code will be documented using JavaDoc while C# and JavaScript will be documented using the appropriate documentation standards according to IDE and language. One of the key areas of SonarQube is to make sure that the code is following standard coding conventions. SonarQube will be used for all the languages as it supports these. In C#, code will go through the code analysis tool in Visual Studio with the rule set “Microsoft Managed Recommended Rules”. The Android code will go through lint checks in order to follow standards and guidelines made by Google.

Code should be designed to be unit-tested. As long as a component is trivial or it is a critical component, unit tests should be added to that component. This applies to all platforms. If the group has enough time, user tests should also be considered for the web-application and the apps. UI and performance tests should also be considered together with Visual Studio. All Java code will use JUnit unit tests, C# will use integrated unit testing framework in the .NET Framework and JavaScript will have unit tests written in Jasmine. Integration testing will be done when a component is added to an application. The integration test will have its focus on communication with the RESTful API and the presentation of received data.

The customer has stated that the application should support several languages, so internationalization becomes an important element in the user-interface. Each platform has its own set of tools and guidelines on internationalizations.

Every critical document for the bachelor project should be stored in a shared folder on Google Drive. This way, the group would not have any issues with loss of data in case a device malfunctions or any loss of device.

Java conventions:

http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

C# conventions:

http://msdn.microsoft.com/en-us/library/ff926074.aspx

JavaScript Google conventions:

http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml

AngularJS Google conventions:

https://google-styleguide.googlecode.com/svn/trunk/angularjs-google-style.html

HTML/CSS Google conventions:

https://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml

5.2 Risk Analysis

#	Issue	Probability	Impact
1	The norwegian data protection authority decides that we should not be allowed to run the system.	Medium	High
2	Not able to complete the project.	Medium	Medium
3	Illness among one or more group members	Low	Medium
4	Bitbucket is down	Low	Low
5	Loss of data due to malware, loss of devices or hardware failure.	Low	High
6	Group member leaves	Low	High
7	Stored documents becomes corrupt or unbuildable after saving	Low	Medium
8	SonarQube fails at analyzing source code and corrupts previous analysis.	Low	Low
9	Inaccurate time estimation	High	Medium

Issue #	**Preventive action**	**On occurrence**
1	Make proper research during the project planning and request information and guidelines from the authorities.	Complain to authorities, discuss with supervisor
2	Develop the project in increments using agile software methods.	Deliver what is completed
3	Make sure each group member gets enough physical activity, extra vitamins, make sure that at least one more group member understands the code that is submitted to the project.	If someone is ill and gone over a longer time, shall the active group leader maintain the status with the ill group member and make sure that the group member is up to date on the group progress.
5	Use version control on the source code where everyone has the repository synced, use cloud technology for file sharing so that all the documents are available through the net.	Restore data from previous backups, cloud store or Git repository.
6	Make sure each group member is doing their work and that they are working well together as a team.	Notify supervisor.
7	Store Word documents for the final report as .xml to enable version control, use version control on all source code. Good backup routines.	Restore data from previous backups, cloud store or Git repository.
9	Estimate items with unforeseeable length as a group.	Reschedule sprint

5.3 Tools

Microsoft Visual Studio 2013

Visual Studio is an IDE by Microsoft for development on the Windows platform. It covers a wide range of technology from web development and .NET to low-level Windows driver development. We will use this IDE for the Windows application development.

Eclipse Luna IDE for Java EE Developers

Free, open-source Java IDE for development of Java Enterprise Edition applications. This will be used on the backend part of the system.

Android Studio

An IDE for Android development created by Google, which is built on top of the IntelliJ IDEA, developed JetBrains. It will be used for the development and testing of the Android application.

Google Drive

Cloud sharing solution by Google, which makes it easy and convenient to share files and documents with other group members. It is also tightly integrated with Google Docs.

Google Docs

A lightweight alternative to Microsoft Word, which makes it easy to collaborate on writing documents and plans. As this tool lacks some essential tools, we have chosen to write the final report using Word and to use Google Docs as a “mock up”-tool instead.

Bitbucket with Git

Bitbucket is an online repository that will be used for version control (Git). The group has good experience with Git and Bitbucket from previous project.

Trello

Online task-board that the group will use for organizing and bookkeeping workflow throughout the project. It makes it easy and convenient to keep track of what other group members are currently doing.

SonarQube

Java application that makes sure that the Java source code is following Java conventions and standards. It also makes sure that the code is simple and easy for others to read and understand. In some cases, SonarQube can reveal bugs.

Microsoft Office Word

Will be used for writing the final report, meeting summaries and longer documents that will be included in the report. As for the project planning, Word will be used for formatting the report. Word documents stored in source control will be saved as XML documents to make it convenient for merge issues.

StyleCop

Plugin for Visual Studio that notifies the developer when the code does not follow C# conventions.

Gradle

Build automation tool that is used in Java-related parts of the project. It makes it possible to easily include other libraries in the project that is stored in public repositories on the Internet.

Webstorm

An IDE developed by JetBrains for web development (CSS, JS, HTML).

Microsoft Visio

Illustration tool used to create flowcharts, UML diagrams and other illustrations.

5.6 Legal

All systems that stores and manages information that can be associated with a person is affected by The Personal Data Act. The law states that the Data Protection Authority shall be notified once systems are gathering this type of data in addition to the purpose of the gathering/processing. The ctrctmgr project gathers data that can be associated with persons such as name and email address. In addition, the law states that all systems that gathers sensitive information has to apply for license from the Data Protection Authority.

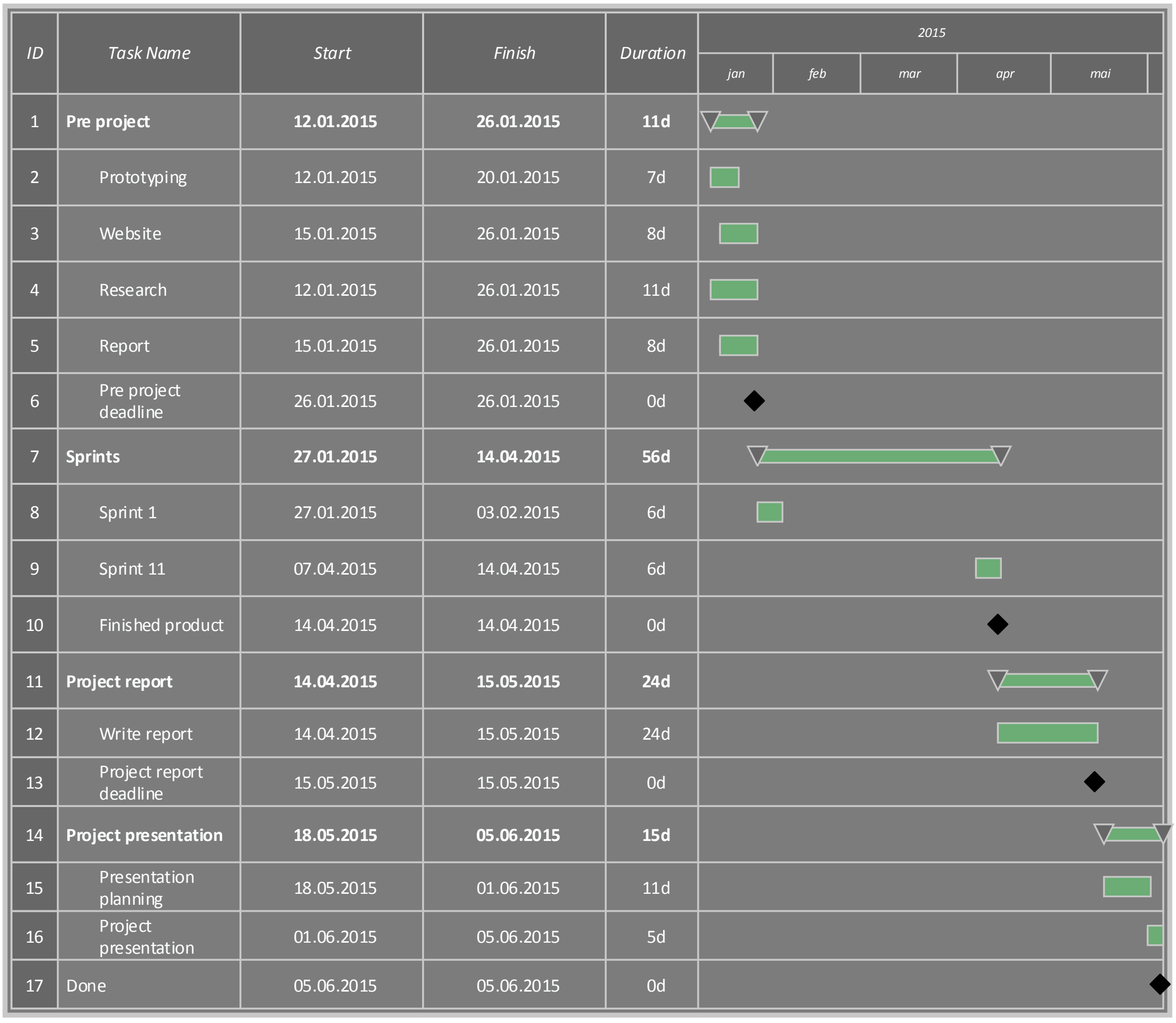
> Personal Data Act §2 section 8 - Definition of sensitive personal data
>
> a) racial or ethnic background, or political, philosophical or religious beliefs,
>
> b) that a person has been suspected, charged or convicted of a criminal offense
>
> c) health conditions,
>
> d) sexual relationships,
>
> e) membership in unions.

Personal Data Act §2 section 8, translated from lovdata.no [3]

As users can enter data that may apply under this law, it may be necessary to apply for a license from the Data Protection Authority. The group is in dialogue with the Data Protection Authority on whether the system requires a license or if it is sufficient to only notify them about the information processing.

6. Project Plan

6.1 Gantt Scheme



ID	Task Name	Start	Finish	Duration
1	**Pre project**	**12.01.2015**	**26.01.2015**	**11d**
2	Prototyping	12.01.2015	20.01.2015	7d
3	Website	15.01.2015	26.01.2015	8d
4	Research	12.01.2015	26.01.2015	11d
5	Report	15.01.2015	26.01.2015	8d
6	Pre project deadline	26.01.2015	26.01.2015	0d
7	**Sprints**	**27.01.2015**	**14.04.2015**	**56d**
8	Sprint 1	27.01.2015	03.02.2015	6d
9	Sprint 11	07.04.2015	14.04.2015	6d
10	Finished product	14.04.2015	14.04.2015	0d
11	**Project report**	**14.04.2015**	**15.05.2015**	**24d**
12	Write report	14.04.2015	15.05.2015	24d
13	Project report deadline	15.05.2015	15.05.2015	0d
14	**Project presentation**	**18.05.2015**	**05.06.2015**	**15d**
15	Presentation planning	18.05.2015	01.06.2015	11d
16	Project presentation	01.06.2015	05.06.2015	5d
17	Done	05.06.2015	05.06.2015	0d

Figure 2 – Gantt scheme for the planned project progress.

The development will take place over 11 sprints that will result in the final product. Which backlog items that will be included in each sprint will be decided by the product owner. Unit testing will be a continuous process during each sprint. For a more detailed description about each sprint, see 4.1 Software Development Methodology.

7. References

1. Lacey M. AgileConnection | Mixing Roles in Scrum. [Online].; [cited 22.01.2014]. Available from: http://www.agileconnection.com/article/mixing-roles-scrum.
2. Kniberg H. Scrum and XP from the Trenches: How we do Scrum: C4Media; 2007.
3. Lovdata - Lov om behandling av personopplysninger (personopplysningsloven) - Kapittel I. Lovens formål og virkeområde. [Online].; [cited 2015.01.19]. Available from: http://lovdata.no/lov/2000-04-14-31/§2.

B – Hour Log

Applies	Date	Start	End	Category	Description
All	08.01.2015	09:30	10:40	Meeting: Dag	In meeting with contracting authority
All	08.01.2015	11:00	11:45	Pre-project	Lo-fi prototyping
All	08.01.2015	12:10	13:05	Pre-project	Lo-fi prototyping
All	08.01.2015	13:15	14:20	Course	Project course
All	08.01.2015	14:30	16:00	Pre-project	Project management
All	08.01.2015	16:00	17:00	Pre-project	Lo-fi prototyping
Thomas	08.01.2015	18:30	19:30	Administrartivt	Hour log
All	09.01.2015	14:20	17:25	Pre-project	Lo-fi prototyping
Martin	09.01.2015	19:45	23:00	Learning: Practical	Bootstrap prototyping
Martin	10.01.2015	12:00	14:00	Learning: Practical	Bootstrap prototyping
Thomas	10.01.2015	16:30	18:30	Learning: Theoretical	Learning AngularJS
Joakim	11.01.2015	10:00	16:00	Learning: Theoretical	Learning AngularJS
Martin	11.01.2015	12:00	14:00	Learning: Practical	Boostrap prototyping
All	12.01.2015	11:30	17:15	Pre-project	Requirement specification
JT	13.01.2015	09:30	12:15	Learning: Practical	Learning AngularJS
Martin	13.01.2015	09:30	12:15	Learning: Theoretical	Learning AngularJS
All	13.01.2015	12:15	14:00	Meeting: Dag	Roles & Groups / Lo-fi presentation / Secure Contracts
All	13.01.2015	14:15	15:00	Meeting: Tom	Meeting 1
All	13.01.2015	15:00	18:00	Learning: Practical	AngularJS
Thomas	14.01.2015	09:30	15:40	Learning: Practical	AngularJS + Pre-project document
Joakim	14.01.2015	09:30	15:40	Learning: Practical	AngularJS + Pre-project document
Martin	14.01.2015	09:30	15:40	Administrartivt	Bachelor Contract / mail about privacy / pre-project
Thomas	14.01.2015	15:40	17:30	Research	AngularJS + Angular Material Design
Thomas	15.01.2015	18:00	22:00	Learning: Practical	Refactoring AngularJS
Thomas	15.01.2015	10:00	13:00	Project website	Bachelor webpage setup (bitbucket) / angular
Martin	15.01.2015	10:00	13:00	Administrartivt	Metting reference[26-27] / legal
Joakim	15.01.2015	10:00	13:00	Pre-project	Project planning
MJ	15.01.2015	13:00	18:00	Pre-project	Writing pre-project
Thomas	15.01.2015	13:00	18:00	Project website	HTML markup / includes..
Martin	16.01.2015	14:30	17:45	Pre-project	Report
Joakim	16.01.2015	16:00	17:45	Pre-project	Report / Software Development methology
Thomas	16.01.2015	14:30	17:45	Project website	home - footer-logos (teknologi / software) + info demo
Martin	19.01.2015	17:00	17:30	Pre-project	Report
Martin	19.01.2015	09:30	10:00	Administrartivt	Legal
All	19.01.2015	10:00	13:00	Pre-project	Project planning document

JT	19.01.2015	13:15	17:50	Project website	Angular file structure and backend scritps
All	20.01.2015	12:30	14:00	Meeting: Dag	Data structure (users / contracts)
All	20.01.2015	14:05	15:00	Meeting: Tom	Meeting 2: pre-project feedback
All	20.01.2015	15:15	17:30	Pre-project	Pre-project correcting
Thomas	21.01.2015	09:30	17:10	Project website	Archive and link is working
Joakim	21.01.2015	10:00	17:10	Pre-project	Report
Martin	21.01.2015	10:00	16:30	Pre-project	Report
Martin	22.01.2015	10:00	16:30	Pre-project	Finished draft, Gradle and Eclipse
Thomas	22.01.2015	10:00	16:30	Project website	styling archive and markup refactoring
Joakim	22.01.2015	10:00	16:30	Project website	Refactoring
Martin	22.01.2015	19:00	20:00	Learning: Practical	Restlet in Eclipse with Gradle
Martin	23.01.2015	11:00	16:45	Learning: Practical	Deploying Restlet applet on Microsoft Azure, Restlet basics
Joakim	23.01.2015	14:30	16:45	Project website	Scrolling
Thomas	23.01.2015	10:00	17:00	Project website	About page and index markup restructuring and css cleanup
Thomas	26.01.2015	10:00	13:00	Project website	Final adjustments
Thomas	26.01.2015	13:15	16:30	Learning: Theoretical	Refresh memory - password protect / restlet +
MJ	26.01.2015	10:00	16:50	Pre-project	Finalizing
All	27.01.2015	12:30	13:20	Meeting: Dag	See meeting reference for this date
All	27.01.2015	14:15	14:30	Meeting: Tom	See meeting reference for this date
Martin	27.01.2015	15:00	17:50	Developing	API
JT	27.01.2015	15:00	17:50	Research	Design patterns / best practises
Joakim	28.01.2015	10:00	18:00	Developing	Angular setup
Martin	28.01.2015	10:00	18:00	Developing	API development (registration)
Thomas	28.01.2015	10:00	18:00	Developing	Dashboard index
Martin	28.01.2015	21:00	22:00	Developing	Commenting code
Joakim	29.01.2015	10:00	17:00	Developing	Setting up angular
Martin	29.01.2015	10:00	17:00	Developing	Completed API for login and registration, testing, documenting
Thomas	29.01.2015	10:00	17:00	Developing	Web index
Martin	30.01.2015	15:00	16:00	Developing	Testing
Joakim	30.01.2015	15:00	17:30	Developing	Angular setup
Thomas	31.01.2015	20:00	23:00	Developing	Dashboard working partial and ngSwitch
All	02.02.2015	10:00	19:00	Developing	Demo demo
All	03.02.2015	12:30	13:00	Meeting: Dag	Sprint one / two
All	03.02.2015	14:15	14:40	Meeting: Tom	Sprint one, report
All	03.02.2015	14:45	17:00	Administrartivt	Task board
MJ	03.02.2015	17:00	18:00	Developing	Login
Thomas	03.02.2015	17:00	18:00	Developing	Index
All	04.02.2015	10:00	12:00	Research	Questionare

Joakim	04.02.2015	12:00	18:00	Developing	Connecting API, session
Martin	04.02.2015	12:00	18:00	Developing	Facebook backend, email notifications
Thomas	04.02.2015	12:00	18:00	Developing	Login and registration
Martin	05.02.2015	10:00	17:45	Developing	Email notification, Facebook login btn integration
Joakim	05.02.2015	10:00	18:00	Developing	Refactoring, cookies
Thomas	05.02.2015	10:00	18:00	Developing	Webapp Refactoring + Index restructuring (header done)
Martin	06.02.2015	14:45	17:30	Developing	Bugfixing (API: Password reset)
Thomas	06.02.2015	15:00	18:30	Developing	Index
Joakim	06.02.2015	14:00	18:00	Developing	Index
Martin	07.02.2015	11:00	12:00	Developing	Sonar, bugfixing, refactoring (API)
Martin	08.02.2015	11:00	13:00	Administrartivt	Data protection authorities, meeting summary
Thomas	09.02.2015	10:00	18:35	Developing	Index + overview mockup
Martin	09.02.2015	10:00	19:00	Developing	API fixes, forgotten password GUI, registration hookup, front page
Joakim	09.02.2015	10:00	19:00	Developing	Session handling, prototyping
All	10.02.2015	12:30	13:20	Meeting: Dag	Dag
All	10.02.2015	14:15	15:15	Meeting: Tom	Tom
JT	10.02.2015	15:15	17:00	Artifact	Glossary
Martin	10.02.2015	15:15	17:40	Developing	API: save user settings
Joakim	11.02.2015	10:00	17:00	Developing	Use-case, wizzard
Martin	11.02.2015	10:00	17:20	Developing	API (user settings, profile picture, available locales)
Thomas	11.02.2015	10:00	18:30	Developing	use-case, wizard + settings module
Martin	12.02.2015	10:00	15:30	Developing	Session handling, API (Categories, store contract)
Martin	12.02.2015	18:50	21:00	Developing	API: modify contract data
Thomas	12.02.2015	10:00	18:00	Developing	settings module + form validating
Martin	13.02.2015	10:30	18:00	Developing	API: Categories, common contract data, JSON serialization of method return values, unit testing
Martin	14.02.2015	12:00	15:00	Developing	API: Attachments
Martin	15.02.2015	11:00	13:00	Developing	API: Unit-tests of attachments (QA)
Thomas	16.02.2015	10:00	19:00	Developing	Settings module, login setup, feedback form
Martin	16.02.2015	10:00	18:00	Developing	Adding contracts (integration)
Joakim	16.02.2015	10:00	17:30	Developing	wizard finalize slide
Martin	17.02.2015	15:20	18:00	Developing	Fixing date picker
Joakim	15.02.2015	18:00	21:00	Developing	wizard finalize slide
Joakim	17.02.2015	10:00	18:30	Developing	wizard attachments
Thomas	17.02.2015	15:20	18:00	Developing	Settings communication to server
All	17.02.2015	14:15	15:00	Meeting: Tom	Scrum + rapport

Martin	18.02.2015	10:00	18:00	Developing	Fixing input fields for adding contracts
Thomas	18.02.2015	10:00	18:00	Developing	Uploading viewing profile picture / gui notes
Joakim	18.02.2015	10:00	18:00	Developing	Wizard attachments uploading
Martin	19.02.2015	10:00	18:00	Developing	Date picker, API (get org. members, cross-domain)
Thomas	19.02.2015	10:00	18:00	Developing	Meeting Dag, settings feedback, wizzard stuff
Martin	20.02.2015	13:00	18:00	Developing	API: Hibernate fixes, storing notifications, get org users, get user type
Joakim	19.02.2015	10:00	18:00	Developing	Attachments frontend
Joakim	20.02.2015	10:00	18:15	Developing	Notifications frontend
Thomas	20.02.2015	10:00	20:00	Developing	wizard time & calendar
Thomas	22.02.2015	19:30	22:30	Developing	wizard time & date solution
Martin	22.02.2015	19:00	20:00	Developing	Add more parties to contract (GUI)
Martin	23.02.2015	10:00	18:20	Developing	Adding more parties, refactoring, note backend, owner
Joakim	23.02.2015	10:30	18:30	Developing	angular finilize
Thomas	23.02.2015	11:20	19:30	Developing	time and date solution / validate (var refactoring)
Thomas	24.02.2015	12:10	18:00	Developing	Validating / refactoring / Meeting with Tom
Thomas	25.02.2015	10:00	19:00	Developing	Wizard validating / feedback / meeting Dag
Martin	24.02.2015	12:10	18:00	Developing	i18n
Martin	25.02.2015	10:00	18:00	Developing	Meeting, bugfixes, i18n
Joakim	24.02.2015	10:00	18:00	Developing	Wizard bugfixing
Joakim	25.02.2015	10:00	18:00	Developing	Wizard bugfixing
Thomas	26.02.2015	10:00	17:00	Developing	Wizard scaling debugging / archive (myContract) markup demo
Martin	27.02.2015	13:30	17:00	Developing	Bugfixing
Thomas	27.02.2015	13:30	17:00	Developing	Mycontracts markup / styling / highlight menu item
Joakim	02.03.2015	10:00	18:30	Developing	Show contract data
Martin	02.03.2015	10:00	19:00	Developing	Refactoring, QA, REST research, Sonar fixes
Thomas	02.03.2015	10:00	19:15	Developing	Refactoring (directive) / Archive styling / main (index) fixes
Martin	03.03.2015	10:30	12:00	Developing	Sonar fixes, research on Restlet validators
Martin	03.03.2015	15:00	18:15	Developing	Advanced notes (API), delete contract
Thomas	03.03.2015	10:30	12:00	Developing	Small fixes, notification dropdown
All	03.03.2015	12:30	13:20	Meeting: Dag	Wizzard & archive (mycontracts)
All	03.03.2015	14:15	15:00	Meeting: Tom	Report / Development phase

Thomas	03.03.2015	15:00	19:00	Developing	Index bugs & refactoring / archive functionality = tabbed searche
Joakim	03.03.2015	15:00	19:00	Developing	Note system
Joakim	04.03.2015	10:00	18:00	Developing	Note system
Thomas	04.03.2015	10:00	18:00	Developing	Filter / archive / "note system" small fixes
Thomas	05.03.2015	10:00	18:00	Developing	Print / fixes / small changes
Martin	05.03.2015	10:00	18:00	Developing	Fixing relation issues, export PDF
Joakim	05.03.2015	10:00	18:00	Developing	edit
Joakim	06.03.2015	16:00	17:30	Developing	Refactoring
Thomas	06.03.2015	16:00	18:00	Developing	Refactoring / highlight on selected contract item.
Martin	06.03.2015	15:00	17:15	Developing	PDF exporting
Martin	07.03.2015	10:00	12:00	Developing	Single-use key
Martin	08.03.2015	10:00	15:00	Developing	Export to XLS, CSV, XLSX
Martin	09.03.2015	10:00	17:30	Developing	i18n of export data, single-use keys, activity log, unit tests
Martin	10.03.2015	14:45	19:10	Developing	Fixes, notification
JT	09.03.2015	10:00	18:00	Developing	Editing and event demo
Thomas	10.03.2015	14:45	20:40	Developing	directive date input
Joakim	10.03.2015	14:45	20:00	Developing	Bug fixing
Martin	10.03.2015	14:45	19:00	Developing	Bug fixing, planning
Martin	11.03.2015	10:00	19:20	Developing	Add/remove org users, basic rights model, category works for org.
Thomas	11.03.2015	10:00	18:30	Developing	Clean up directvie, finished time edit
Thomas	12.03.2015	10:00	20:45	Developing	Single rights / GUI choices / clean up
Martin	12.03.2015	10:00	21:20	Developing	Groups and rights
Joakim	12.03.2015	10:00	21:20	Developing	Group and category management
Martin	13.03.2015	11:30	17:00	Developing	Rights, unit testing, Sonar
Thomas	13.03.2015	12:00	20:30	Developing	Directive on rights table / Single Rights binding (gui - db)
Thomas	14.03.2015	17:30	23:00	Developing	Directive and rights functionality
Thomas	16.03.2015	10:00	21:00	Developing	Manage organization and refactoring
All	17.03.2015	12:00	14:00	Developing	Preparing demo, bugfixes
All	17.03.2015	14:00	15:00	Meeting: Dag	Projet status meting
Thomas	17.03.2015	15:30	17:45	Developing	Implementing sum value in contract / fixed directive
Martin	17.03.2015	15:30	17:45	Developing	Sum (API), research on payment
Thomas	18.03.2015	10:30	18:00	Developing	Next expiring contracts(GUI) / Calendar testing (solution hunting)
Joakim	13.03.2015	15:00	18:00	Developing	Rights
Joakim	15.03.2015	10:00	13:00	Developing	Groups and rights

Joakim	16.03.2015	10:00	18:00	Developing	rights
Joakim	17.03.2015	15:00	18:00	Developing	Refactoring
Martin	18.03.2015	10:00	19:00	Developing	Paypal API integration
Joakim	18.03.2015	10:00	19:00	Developing	Refactoring, route authentication
Thomas	19.03.2015	10:00	20:00	Developing	Calender / nextExp / PayPal
Joakim	19.03.2015	10:00	20:30	Developing	
Thomas	20.03.2015	10:00	17:45	Developing	Change subscription in settings (missing api to work)
Joakim	20.03.2015	15:00	17:45	Developing	Mobile web app
Martin	19.03.2015	10:00	20:00	Developing	Paypal, Windows hybrid app
Martin	20.03.2015	10:00	17:30	Developing	Push notifications (Windows) with push registraiton (API)
Martin	21.03.2015	11:00	15:00	Developing	Push prototype on Android
Martin	22.03.2015	12:00	18:00	Developing	Push registration (API), documentation, refactoring
Thomas	23.03.2015	10:00	19:00	Developing	Finished paypal chaning subscription / myOrg mobile / ngTouch
Thomas	24.03.2015	12:00	18:00	Developing	GUI scaling wizard and category
Martin	23.03.2015	10:00	20:00	Developing	Background tasks for push, facebook integration
Martin	24.03.2015	10:00	18:15	Developing	Facebook integration, picture of contract
Joakim	23.03.2015	10:00	18:00	Developing	Android app
Joakim	24.03.2015	10:00	18:00	Developing	Android facebook
Thomas	25.03.2015	10:00	18:00	Developing	archive contract & export multiple GUI buttson & scaling
Martin	25.03.2015	10:00	19:00	Developing	Picture of contract
Martin	26.03.2015	10:00	18:00	Developing	Windows QA, meeting, export list of contract, report planning
Joakim	26.03.2015	10:00	18:00	Developing	Picture of contract
Thomas	26.03.2015	10:00	18:00	Developing	Mobile friendly modules
Martin	27.03.2015	14:00	17:00	Developing	Report planning, API work
Martin	30.03.2015	10:00	13:00	Developing	API modifications, refactoring
Martin	31.03.2015	10:00	17:00	Developing	Containerless API, tested compatibillity with Spartan
Thomas	30.03.2015	16:00	18:00	Developing	Working multiple export
Thomas	03.04.2015	18:30	21:50	Developing	System module with woking customer list
Martin	03.04.2015	19:00	22:00	None	System architecture
Thomas	04.04.2015	14:30	16:45	Developing	System settings
Thomas	06.04.2015	16:00	20:45	Developing	attachment upload in directive & mobile firendly
Thomas	07.04.2015	14:00	19:00	Developing	Notes mobile friendly
Thomas	09.04.2015	10:00	15:30	Developing	Bug fixes & refactroing

Thomas	09.04.2015	16:15	18:00	None	Gui report & research
Martin	09.04.2015	10:00	17:30	Developing	Bugfixing, refactoring, subscription restrictions
Martin	09.04.2015	12:00	15:00	Developing	QA, bugfixing, refactoring, Sonar issues
Martin	10.04.2015	13:00	19:00	Artifact	Report: Hibernate
Martin	11.04.2015	10:00	16:00	Artifact	Report: Hibernate and Restlet
Martin	12.04.2015	12:00	17:00	Artifact	Restlet design
Martin	13.04.2015	10:30	18:00	Artifact	Internationalization, Gradle, technical memo on pw hashing
Thomas	13.04.2015	10:00	19:00	Developing	Fixing bugs / redirect on homeUser invite
All	14.04.2015	12:30	13:30	Meeting: Dag	
All	14.04.2015	14:15	14:45	Meeting: Tom	
Thomas	14.04.2015	15:00	19:00	Developing	Gui fixes & redirect from Paypal (inprogress)
Thomas	15.04.2015	10:45	18:15	Developing	Refactor paypal redirect and highlight on aside the angular way
Martin	14.04.2015	12:30	18:20	Report	Meetings, report
Martin	15.04.2015	10:00	18:00	Developing	Report (Memo + project management)
Joakim	13.04.2015	10:00	18:30	Developing	Android app
Joakim	14.04.2015	15:00	18:30	Developing	Android app
Joakim	15.04.2015	10:00	18:00	Developing	Android app
Joakim	16.04.2015	11:45	18:00	Developing	Android app
Thomas	16.04.2015	10:15	18:00	Developing	login fixes, support, http get handling
Martin	16.04.2015	11:30	20:30	Report	Report (Project management), deployment on Azure
Joakim	17.04.2015	14:00	18:10	Developing	Android app
Martin	17.04.2015	14:00	18:10	Developing	Optimization
Thomas	21.04.2015	12:30	18:00	Developing	Big fixes, Test cases and 'home-standard-user-category'
Martin	21.04.2015	12:30	18:30	Report	Report writing, bugfixes
Martin	20.04.2015	10:00	18:00	None	Switched database connector manager, fixed onetime key issue
Joakim	20.04.2015	10:00	18:00	Developing	Android app
Joakim	21.04.2015	12:00	17:00	Developing	Refactoring core
Thomas	22.04.2015	10:15	18:00	Developing	Bugfixes, Rapport GUI, Test case, notification adjustments
Joakim	22.04.2015	10:00	18:00	Developing	Refactoring core
Joakim	23.04.2015	10:00	17:00	Developing	Refactoring core + rights management
Martin	23.04.2015	10:00	17:20	Report	Report
Thomas	23.04.2015	10:00	17:40	Developing	Report, css refactor, bug fixes and some todos

Joakim	24.04.2015	16:00	18:00	Developing	subscription restrictions
Martin	24.04.2015	16:00	18:40	Report	Finding images, icons, POE fixes, created API call, sent in report
Thomas	24.04.2015	17:30	19:00	Developing	POE language & scaling bug fix (GUI)
Thomas	25.04.2015	18:00	19:30	Developing	Language POE dropdown + scaling bugs
Thomas	27.04.2015	10:15	18:00	Developing	Bug reporting and fixing, cleanup + commenting + testing
All	28.04.2015	11:00	12:30	Developing	Bugfixes
All	28.04.2015	12:30	15:00	Report	meetings with supervisor and customer
Martin	27.04.2015	10:00	19:00	Developing	i18n
Martin	28.04.2015	15:00	18:00	Report	Project report
Thomas	28.04.2015	15:00	17:50	Developing	Bug fixes, updating involved parties, + some report
Thomas	29.04.2015	10:30	17:40	Report	Testing with real people + report + bug fixes
Martin	29.04.2015	10:30	17:40	Report	Report
Martin	30.04.2015	10:30	17:45	Learning: Practical	Embedded containers on Azure + performance analysis
Thomas	30.04.2015	10:30	18:20	Report	Bug fixes, string reporting, report gui and bootstrap
Martin	01.05.2015	10:30	18:00	Report	Technical memo on performance, planning
Thomas	01.05.2015	10:30	18:00	Report	GUI, target audience + reading (some development refactoring)
Martin	03.05.2015	12:00	15:00	Report	QA
Joakim	26.04.2015	10:00	12:00	Developing	i18n
Joakim	27.04.2015	10:00	15:00	Developing	i18n
Martin	04.05.2015	10:30	18:00	Report	Hybrid apps, list of plugins
Joakim	28.04.2015	10:00	15:00	Developing	i18n
Joakim	29.04.2015	10:00	15:00	Developing	i18n
Joakim	30.04.2015	10:00	18:00	Developing	Refactoring
Joakim	01.05.2015	10:00	18:00	Developing	Testing
Joakim	03.05.2015	10:00	12:00	Report	Angular technology
Joakim	10.04.2015	11:00	18:00	Report	Design Angular
Martin	05.05.2015	12:00	18:30	Report	Readme files, reviewing reports, started on wrap-up
Thomas	05.05.2015	12:00	18:45	Report	GUI, Contract lifecycle, reviewing.
Joakim	06.05.2015	11:00	18:00	Report	Angular
Martin	06.05.2015	11:00	18:00	Report	Ending
Thomas	06.05.2015	11:00	18:20	Report	Contract lifecylce, GUI, and bootstrap
Joakim	07.05.2015	10:00	18:00	Report	Angular Design
Joakim	04.05.2015	10:00	18:00	Report	Angular

Joakim	05.05.2015	10:00	18:00	Report	Angular
Martin	07.05.2015	10:00	18:00	Report	Conclution, description, stitching stuff together
Thomas	07.05.2015	10:00	18:00	Report	Survey, review, gui / contract life cycle
Thomas	08.05.2015	11:00	16:45	Report	GUI feedback & reviewing
Joakim	08.05.2015	13:00	17:00	Report	Grunt & Bower
Martin	08.05.2015	11:00	17:00	Report	Commissioning
Martin	10.05.2015	12:00	15:00	Report	Reviewing
Martin	11.05.2015	10:30	20:45	Report	Meeting, going through report and fixing issues
Thomas	11.05.2015	11:00	20:00	Report	Going through report and fixing issues
Thomas	12.05.2015	12:00	18:00	Report	Report fixing
Martin	12.05.2015	12:00	20:45	Report	Report fixing
Joakim	11.05.2015	12:00	18:00	Report	Report fixing
Joakim	13.05.2015	11:15	18:00	Report	Report fixing
Martin	13.05.2015	10:30	18:00	Report	Report fixing
Thomas	13.05.2015	10:30	18:00	Report	Report fixing
Martin	14.05.2015	10:00	18:00	Report	Report fixing, delivering
Joakim	14.05.2015	10:00	18:00	Report	Report fixing, delivering
Thomas	14.05.2015	10:00	18:00	Report	Report fixing, delivering

C – Meeting summaries

Meeting Summary - ETC

Duration: 1 hour

January 8, 2015	
Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Dag L Solhaug
Next meeting:	January 13, 12:30, ETC

Announcements

The group has successfully obtained the task. Further discussion about the project is required.

Discussion

The discussion during the meeting was centralized around which features the system should support. Concept drawings of the system played a central role during the discussion. Some of the discussed features were:

- Contracts should have attachments in PDF, image or document format that should be received from a scanner or a camera.
- The system should behave equally on different platforms.
- There should be an overview page to make the user get a simple overview of the status of the current contract.
- The system owner should have an overview of active subscriptions.
- Should support Google Analytics and Google AdSense to gather user activity and get revenue from the free-version of the system.
- The system should be structured and simple to use.

Follow-up Points

- Look into how development for Windows 10 apps can be used both on desktop and phone without major adjustments.
- Research optical character recognition (OCR).
- Create lo-fi prototypes of the web-app.
- Suggestion on which subscriptions should be available.

Meeting Summary - ETC
Duration: 1 ½ hour

January 13, 2015	
Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Dag L Solhaug
Next meeting:	January 20, 12:30, ETC

I. Announcements

The group has finished creating lo-fi prototypes of the web site including a concept website, has started creating system requirements and a concept for how contracts should be securely encrypted.

Agenda for the meeting:

- Project contract
- Presentation of lo-fi prototype, concept website
- List of discussion items
- Contract encryption system
- Subscription
- Announcement of system requirements
- Tools to be used
- Summary, next meeting time

Discussion

The customer was happy with the prototypes but felt that the overview needed a better solution.

Other companies should be able to integrate this system with their system, for example, Telenor should be able to publish contracts in our system to their customers if they are using our system. It is important to note that there should be a categorization system so users can easily categorize contracts.

One issue might be to find a way to make private users pay for the system. The group did not see the advertisement as a good enough approach as users are likely to visit the website only a few times each month, which will not generate much web traffic. An alternative approach would be a one-time payment for the mobile application.

Customer was concerned about capacity issues when encrypting attachments to contracts. Therefore, it would be reasonable that this feature should be available for more expensive subscription options. There should be maximum four subscription options, where two should be for private users and the two last should be for organizations/companies.

The customer did not know about any of the frameworks that the group suggested should be used for development of the application. The group did not see PHP as a sufficient language for the development of the application both due to the scale of the application and the required libraries. As the customer stated Java was an alternative, the group saw this as the best approach. However, the customer raised concern around hosting option for Java websites. The group stated that the most major cloud vendors (Google, Microsoft and Amazon) has hosting options for Java web applications. After going through Hibernate, the customer raised a wish for using MariaDB for the database in the system.

Follow-up Points

- Look for a better solution on how to solve the dashboard.
- Continue on system requirements.

Meeting Summary - Supervisor
Duration: 1 hour

January 13, 2015	
Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Tom Røise
Next meeting:	January 20, 14:00, HiG

I. Announcements

Agenda:

- Project contract
- Discuss group rules
- What should be on the pre-project, where is the template that was presented on the last bachelor information lecture?

Discussion

The supervisor raised concern that the group would start writing code that should not be rejected in the final product, at this stage there should only be prototypes.

System requirements should be handed in as an appendix in the pre-project. Prioritize quality above quantity in the reports.

Previous bachelor projects that the group should look more into was discussed.

Should begin setting up the group website; ask IT when this is ready.

Optical text recognition (OTR) should be looked more into when scanning contracts.

Follow-up Points

- Ask IT when we can publish the website.
- Keep OTR in mind, as this should be looked more into at a later point.

Meeting Summary - ETC

Duration: 1 ½ hour

January 20, 2015	
Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Dag L Solhaug
Next meeting:	January 27, 12:30, ETC

I. Announcements

The group has finished writing a draft of the project plan. Research has also been done in which cloud providers has good cloud support. Amazon provides the latest Java 8 platform, while Microsoft provides Java 7 out of the box. It is, however, possible to upload a custom JVM but there are compatibility issues regarding using newer updates where Java 8 update 0 was working fine. Microsoft is aware of this issue and is working on resolving it in addition to making it possible to select other Java versions from the Azure control panel.

The group is also in dialogue with the Norwegian Data Protection Authority as the system may process personal or sensitive personal information. It is unclear whether the project requires a license or if it is sufficient to notify them regarding the system.

A few solutions on how login and contract sharing has been sent to employer.

Project website development has started and should be done shortly.

Agenda for the meeting:

- Norwegian Data Protection Authority
- Cloud solutions
- Discuss contract sharing and groups
- Discuss login for home users and organizations
- Further discussion regarding feature specification

Discussion

The employer was positive to requesting the authorities on declaring whether the project requires a license or just a notification, they got experience with the Data Protection Authority from previous projects. Considers to host the application in Ireland and that the legal aspect should be ok regarding this matter.

The discussion regarding group contracts ended with that each user should be assigned to a group where the group (user group) should have read or read/write-rights to a contract category (context group). The solution that was submitted earlier were not customizable regarding read/write-rights. When a user is given ownership/individual rights to a contract where the contract is not in a category should the user only see that contract and other contracts which the user has sufficient rights to in that category.

The group highlighted the issue regarding Facebook login for organizations, as this may not be a good approach in terms of security. Employer acknowledged the issues but suggested that the difference between home user and organization login should be clearer by having a different background or an entirely different webpage for each login.

Advertisement should gather data from the metadata being displayed on the website (and not from the attachments) and use that data to get ads that are more accurate.

Subscription types were also discussion. The paid home user subscription should remove ads, enable encryption, custom categories and make two-factor authentication possible. Organizations should have a trial version with a low limit on user and contract count. Contract attachment limits would also be reasonable. It would be possible to add advertisement to this version. The paid version for organizations should for example not have limits in terms of contracts or users. Should allow encryption and two-factor authentication. The big issue is to increase the quality on paid organization subscriptions. Another issue is to consider when the system should stop the user from taking an action due to a limit on their subscription.

Employer agreed to that code snippets can be used in the final project report as long as they do not make it possible for others to clone the system. ETC was using Ant as project automation tool but Gradle was an acceptable tool. The employer suggested one-week sprint length as longer sprint length often makes the development team dull and would make the development methodology move closer towards waterfall.

Meeting Summary - HiG
Duration: 1 hour

January 20, 2015

Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Tom Røise
Next meeting:	January 27, 14:30, HiG

I. Announcements

The group is in dialogue with the Norwegian Data Protection Authority. The group is pleased with the current progress on the project.

Agenda for the meeting:

- Norwegian Data Protection Authority
- Task-board
- Feedback on the project plan

Discussion

Having a dedicated room for the bachelor project will become problematic, as the school has already allocated all its rooms and resources. It would be possible to both book a group room continuously and take a picture of the task board after each day. Another way would be to use the current room (G223) and hope that nobody removes the task board. As the group members see that Trello does not fit well enough in terms of features and customization. Jira and Confluence was mentioned as an alternative.

Feedback was given to the draft of the pre project. The problem description should be redefined as the product that the team is going to create is now more clear, compared to the abstract description that was given to the group in the beginning of the project.

Should be clear on where the boundary of the project is as topics such as market analysis and decisions on whether advertisement should be a subject is a decision the employer should make.

Meeting Summary - ETC
Duration: 1 hour

February 3, 2015	
Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Dag L Solhaug
Next meeting:	February 10, 12:30, ETC

I. Announcements

The group got close to completing the goal of the last sprint (see previous meeting summary). Much progress has been made with the API; however, the website has been lagging more behind due to switching and learning a new task runner for building the project. There was lack of documentation under Google Closure and we saw that Grunt was a far better alternative.

We are also going to have a meeting with Eivind (lecturer in ergonomics) and discuss design and eventually creating a user survey.

Agenda for the meeting:

- Scrum retrospective
- Scrum planning

Discussion

Retrospective:

- We need to get the scrum board as soon as possible. Currently waiting for resources from the bookstore.
- Need to get better at communicating with each other, both asking for help and announcing what each member is working on.
- Need to break down larger task in smaller tasks; last sprint had too abstract tasks with two little concrete items.
- Too few breaks.
- Learned and experienced a lot.

Items/tasks for next sprint:

- GUI-setup: 2-3 days
- Email-system (Email verification and forgotten password mechanism): ½ days
- Facebook backend: ½ day
- Facebook login: ½ day
- Session handling: 1 day
- Discussing with the Norwegian Data Protection Authority on whether it is required that the system requires license or if a notification is sufficient: 1-2 hours

Meeting Summary - ETC
Duration: 1 hour

March 10, 2015	
Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Dag L Solhaug
Next meeting:	March 17, 14:00, HiG (Status meeting)

I. Announcements

Most of the items from last sprint have been completed. The group is happy with the progress that has been made.

Agenda for the meeting:

- Scrum retrospective
- Scrum planning

Discussion

It was discussed that the next few sprints is crucial for how many items in the backlog that will be completely implemented. There will be a status meeting together with the supervisor (Tom) where the product will be reviewed on the next meeting.

Retrospective:

- Good work-flow
- Has become better on splitting items into tasks

Items/tasks for next sprint:

- Activity log for recent user activity
- Date modification in edit contract
- Mobile scaling
- Administrate organization members (Adding/removing)
- Administrate categories
- Administrate groups
- Rights between groups, categories and contracts
- Individual rights on contracts

Meeting Summary – Status Meeting

Duration: 1 hour

March 17, 2015

Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Dag L Solhaug, Tom Røise
Next meeting:	March 26, 12:30, ETC/April 7, 14:15 HiG

I. Announcements

The product is starting to get close to feature completeness, the previous sprint was challenging but nearly all of the items have been completed.

Agenda for the meeting:

- Project review
- Scrum retrospective
- Scrum planning

Discussion

A prototype on a Windows hybrid app was demonstrated that was made with Cordova. However, the customer has expressed skepticism regarding maintenance around this as several versions of the web app is generated through a cross-platform compiler tool.

Retrospective:

- Has been working very efficient the last sprint
- Has gotten more done than thought
- Has become better at splitting up items into tasks, it is motivating to get up from the chair and clear the head once progress had been made to a task.

Items/tasks for next sprint:

- Hybrid apps: Push, login, take picture
- Pay with PayPal
- Next expiring contracts + calendar on Overview
- Further work with scaling pages on phone
- Sum on contract

Meeting Summary - ETC

Duration: 1 hour

April 14, 2015

Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Dag L Solhaug
Next meeting:	April 28, 12:30, ETC

I. Announcements

Many of the items/tasks from previous sprint has been completed, there are however still tasks that needs to be carried out. There has not been much time for testing in the last sprint to test the entire product. This day (April 28.) is also the date in the project plan, which the project should be done.

Agenda for the meeting:

- Scrum retrospective
- Scrum planning

Discussion

Retrospective:

- Important to have the option for workarounds for features such as payments, captcha and email activation.
- Having one-week sprints has been fine, has also made the team get a decent workflow for every week.

Although this was according to the project plan, everyone agreed that one extra extended sprint with the length of two weeks that would be sufficient to get the project done. Dag wants the application to be deployed on Azure as of the free subscription tier. There will also be done some user testing in coordination with another bachelor group.

Items/tasks for next sprint:

- Azure deployment
- Complete Android app
- Support page (Point of entry)
- Complete I18n
- Continue refactoring of components around the system
- Rights on components on the page on subscription level and user-type level

Meeting Summary - Supervisor
Duration: 40 minutes

April 14, 2015	
Present:	Martin Storø Nyfløtt, Joakim Andreas Jøreng, Thomas Mellemseter, Tom Røise
Next meeting:	TBA

I. Discussion

We are currently moving towards the final phase of the project and there is still development to be done. A more thorough testing of the system will also be performed towards the end of the project together with user testing that will be coordinated together with another bachelor group.

Feedback on drafts for different segments on the final project report was given. Much of the feedback went to that there was written too much abstract and too little concrete towards our product. Supervisor also felt that we introduced the reader too much to different technologies instead of demonstrating how it was applied to our case.

D – Backlog

#	Type	Title	Description
1	Technical	Web-browser compatibility	Minimum IE9, newer versions of Chrome, Firefox and Safari
2	Functional	Registration: Private	Register a user (name, email, password, subscription type)
3	Functional	Registration: Organization	Register an organization (org-name, email, password, subscription type)
4	Technical	Technology: Frontend	AngularJS (web), Android API 16+, Windows 8.1
5	Technical	Technology: Backend	Hibernate, Restlet
6	Functional	Login	User signs in with email and password, extra token if two-factor is activated/required
7	Functional	Login: Facebook	Private users can log in using Facebook
8	Functional	Manage categories	An org. admin can add new categories in the organization, private users can add new categories to their own account
9	Functional	Categories: Rights	A user has read/modify/admin rights to a contract, either directly or through a group which has assigned rights to a category
10	Functional	Individual rights	An org. admin can add users to a group
11	Functional	Create contract	A user can add a contract to their account
12	Functional	Create contract: category	Auto-completion for category
13	Functional	Create contract: fields	Title/name, created date (automatic), expiration date, second party, (other parties), period (optional), start-date (optional), renewal date (optional), responsible (optional), notes (optional), attachments (optional), notification type (mail, app, calendar), notification time, category (optional)
14	Functional	Create contract: attachment	User can browse or drag and drop an attachment for a contract into a box which will then be uploaded or take a picture if using a mobile device
15	Functional	Create contract: share	User can share a contract with individual users
16	Functional	View contracts: overview	The user can view all the contracts which the user is creator of and is responsible for on the page "My Contracts"

17	Functional	View contracts: group	The user can click on the name of a group which the user is a part of and then get the overview of all the contracts that has been shared in that group
18	Functional	View contracts: shared	The user sees the contracts that is shared with him/her (individually) on the page "shared contracts"
19	Functional	View contracts: search	The user can enter key-words in the search field above the overview to filter contracts and/or specify a date in the date/time picker
20	Functional	View contracts	The user gets a list of contracts, which includes title, expiration date, creation date, and involved parties, should show if the contract has an attachment. Can click on a contract to show more details.
21	Functional	User settings	User can change email and password on the settings page, requires that the user re-enters the password for the account, the new email/password is entered two times
22	Functional	User settings: two-way auth	User can enable two-way auth if the subscription enables this, is prompted to handshake with eg. Google Authenticatior.
24	Functional	User settings: subscription	User can select which type of subscription they want on the subscription page, is prompted to enter PayPal data if this is not stored already
25	Functional	Edit contract	A user can modify the contracts details, shared users, groups and add/remove attachments
26	Functional	Export data from overview	User can export the current overview to PDF, XLS, or print
27	Functional	Overview: next expiring contracts	The user sees a list of the next expiring contracts (name, expiration date/time left, parties)
28	Functional	Overview: expiring contracts on dates	The user can click on the date in a calendar which will show all the expiring contracts in a list below
29	Functional	Overview: events	The user sees a list of the last actions that has taken place in the groups which the user is a member of, for example modified/added/removed contracts
30	Functional	Sign out	User can sign out from the drop down menu when signed in.
31	Functional	Forgotten password	The user can request a password reset by submitting the email address + captcha
32	Functional	Homepage	The homepage lists general features, subscription details, contact information and link to login, registration and contact info

33	Functional	Register: Facebook	Only private users can register using Facebook
34	Functional	Login: Facebook	Only private users can log in using Facebook
35	Functional	Encrypt contracts	Contract attachments and its metadata is encrypted on the server using a public-private key-chain (See Visio illustration) if the subscription enables this.
36	Functional	View subscribed users	The system administrator can see a list of all the subscribed members and their status.
37	Functional	Groups	An org. user is a part of one or more groups, which then is assigned to a category among with its rights.
38	Functional	Organization	Organization can add new users to their organization, account details sent by mail.
39	Technical	Hybrid app	Windows Phone hybrid app
40	Technical	Hybrid app	Android hybrid app
41	Technical	OCR	Images / scanned documents gets their text extracted through OCR

E – Group Rules

<u>Grupperegler</u>

§1 - Fravær fra faste møtetider skal rapporteres til prosjektleder (Martin) minst en dag i forveien.

§2 - Dersom et medlem ikke utfører arbeidsoppgaver skal veileder kontaktes dersom dette gjentar seg.

§3 - Dersom gruppeleder og veileder er enig, kan medlemmer ekskluderes.

§4 - Avgjørelser internt i gruppen avgjøres ved håndsopprekning, men dersom det oppstår store uenigheter er prosjektleder ansvarlig for å få alle til å komme til enighet.

§5 - Gruppeleder har rett til å signere på vegne av gruppa.

§6 - Eventuelle kostnader deles likt på alle gruppemedlemmene.

§7 - Dokumentasjon og kode skal følge retningslinjer og standarder som gruppen har bestemt og bør helst følge standarder og konvensjoner i det programmeringsspråket som er i bruk.

Signatur:

____

____

____

F – Deployment on Microsoft Azure using FTP

This guide explains how to deploy Contract Manager as a web app hosted on Microsoft Azure.

Before deployment, make sure the following files are ready:

- The API compiled as a WAR or a JAR (Note: Remove the .SF files from the JAR archive under the META-INF, otherwise the runtime will throw an exception regarding signed JARs)
- All CSS, JS, HTML for core product and point-of-entry.
- An SQL file for the database (database.sql)
- Tomcat 32-bit zip file for Windows from http://tomcat.apache.org/download-80.cgi (not needed if the API is deployed as a JAR)
- JDK 1.8u0 32-bit installed on a Windows machine from http://www.oracle.com/technetwork/java/javase/downloads/java-archive-javase8-2177648.html#jdk-8-oth-JPR

Setting up MySQL on Azure

Sign into Azure on https://portal.azure.com

In order to create a new database, click the "New" button on the bottom right-hand corner of the screen, click "Data + Storage", click "Azure Marketplace" and enter "MySQL" in the search-field. The select MySQL Database by ClearDB. Click on the create button and enter a database name, e.g. "ctrctmgrdb". Make sure a location close to the desired target audience has been selected. Take note of which location the database is deployed to as this will be the same location for the website. Make sure the Legal Terms has been ticked, read the text and clicked "OK" before clicking "Create".

Once the database has been created, it will appear on the portal dashboard and will automatically open. After this, click properties to get the MySQL hostname, port number, username and password. The database name will be the same as entered in the wizard earlier, in this case "ctrctmgrdb". Then execute the default SQL file database.sql on the SQL database.

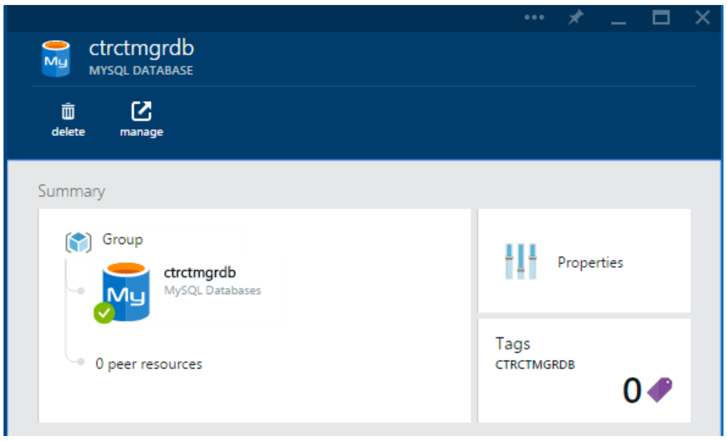
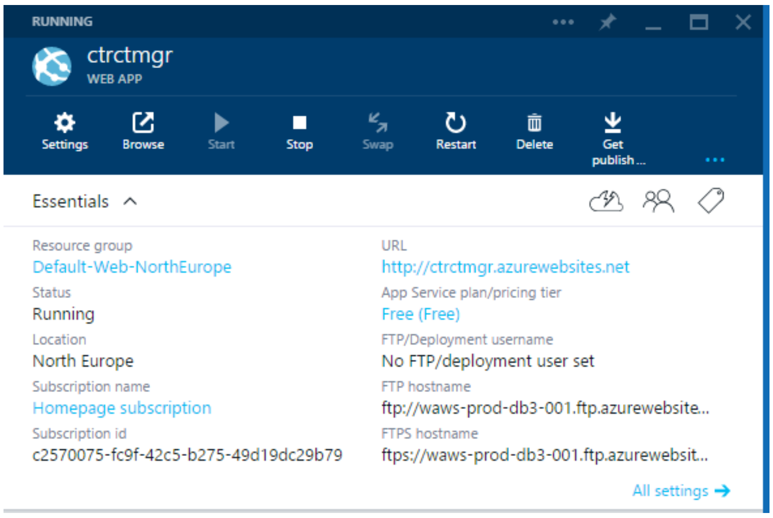


Figure 88 – The created database as seen through the Azure portal.

Setting up an Azure Website

Sign into Azure on https://portal.azure.com

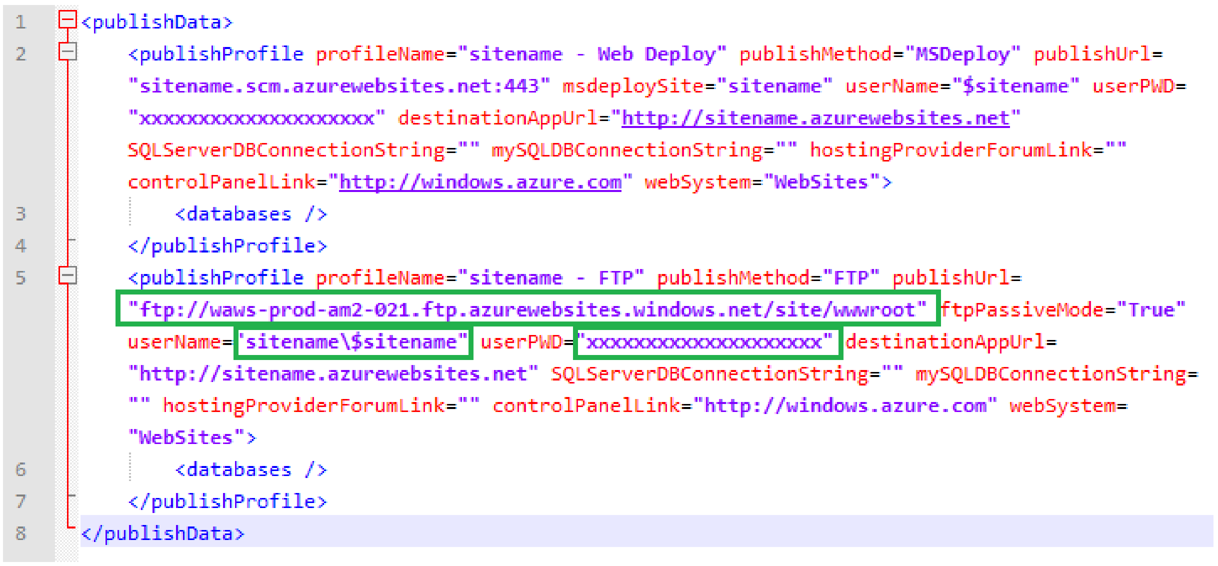
Create a new website by clicking on New -> “Web + Mobile” -> “Web app”. Enter a name for the URL being used for the website, eg. “ctrctmgr”. In order to deploy the website as a free instance, click “Or create new” under “APP SERVICE PLAN”, then click “PRICING TIER”, click “View all” and select “Free” on the bottom of the page.



Resource group
Default-Web-NorthEurope
Status
Running
Location
North Europe
Subscription name
Homepage subscription
Subscription id
c2570075-fc9f-42c5-b275-49d19dc29b79
URL
http://ctrctmgr.azurewebsites.net
App Service plan/pricing tier
Free (Free)
FTP/Deployment username
No FTP/deployment user set
FTP hostname
ftp://waws-prod-db3-001.ftp.azurewebsite...
FTPS hostname
ftps://waws-prod-db3-001.ftp.azurewebsit...

Figure 2 – The created web app through the Azure portal.

Once the website has been created, it will appear on the home screen and automatically open. Click the button “Get publisher profile” next to delete. This will download an XML file containing the FTP username and password. Open the XML file in a text editor such as Notepad++ and find the publishProfile for FTP. The FTP address will be the publishUrl attribute, username is the userName attribute and password is the userPWD attribute.



```xml
<publishData>
    <publishProfile profileName="sitename - Web Deploy" publishMethod="MSDeploy" publishUrl=
    "sitename.scm.azurewebsites.net:443" msdeploySite="sitename" userName="$sitename" userPWD=
    "xxxxxxxxxxxxxxxxxxxx" destinationAppUrl="http://sitename.azurewebsites.net"
    SQLServerDBConnectionString="" mySQLDBConnectionString="" hostingProviderForumLink=""
    controlPanelLink="http://windows.azure.com" webSystem="WebSites">
        <databases />
    </publishProfile>
    <publishProfile profileName="sitename - FTP" publishMethod="FTP" publishUrl=
    "ftp://waws-prod-am2-021.ftp.azurewebsites.windows.net/site/wwwroot" ftpPassiveMode="True"
    userName="sitename\$sitename" userPWD="xxxxxxxxxxxxxxxxxxxx" destinationAppUrl=
    "http://sitename.azurewebsites.net" SQLServerDBConnectionString="" mySQLDBConnectionString=
    "" hostingProviderForumLink="" controlPanelLink="http://windows.azure.com" webSystem=
    "WebSites">
        <databases />
    </publishProfile>
</publishData>
```

Figure 89 – Credentials for FTP deployment where the ftp hostname, username and password is highlighted with green boxes.

Open a FTP client and connect using the hostname, username and password from the deployment profile. Under /site/wwwroot, create a folder “bin”. Under bin, create a folder named “jdk1.8.0”. Upload the content from JDK 1.8 update 0 installed on the Windows machine earlier from “C:\Program Files (x86)\Java\jdk1.8.0” to the newly created folder “jdk1.8.0”.

The API can be deployed either as a JAR file (embedded container) or as a WAR. The difference being the JAR has a slower response time than WAR deployment (a few milliseconds on avg.) but the WAR will take significantly longer to start up.

WAR deployment

Create a folder named “tomcat” under the bin folder. Next, upload the content of the zipped Tomcat folder to the folder named “tomcat”. After this, upload server.xml to

/site/wwwroot/bin/tomcat/conf and replace the existing xml file.

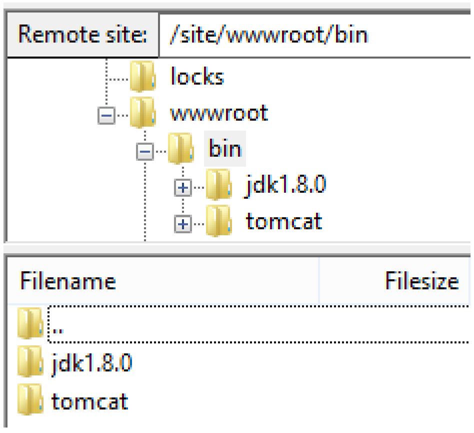


Figure 90 – Tree structure of Tomcat deployment.

It might be a good idea to delete all the folders except ROOT in /site/wwwroot/bin/tomcat/webapps, in order to improve startup speed of Tomcat, it is also possible to enable native libraries for Tomcat. This can be done by moving the file "tcnative-1.dll" in /site/wwwroot/bin/tomcat/bin to \site\wwwroot\bin\jdk1.8.0\bin.

http://wiki.apache.org/tomcat/HowTo/FasterStartUp

Then upload the static content to ROOT under /site/wwwroot/bin/tomcat/webapps.

JAR deployment

Create a folder named "api" under "bin". Upload the JAR file to this folder and create a file called "startup.bat" with the following contents:

```
d:\home\site\wwwroot\bin\jdk1.8.0\bin\java.exe -jar
d:\home\site\wwwroot\bin\api\ContractManagerAPI.jar
no.etc.ctrctmgr.api.JettyMainEntry -Djava.net.preferIPv4Stack=true -o true
```

Upload the static content to the wwwroot folder (/site/wwwroot/).

web.config

At this stage, all the binary files for the platform running Contract Manager has been upload. It is now necessary to configure Azure how to use the binaries.

Open the file "web.config" in a text editor and edit the environment variables required for Contract Manager suffixed by CTRCTMGR. Enter the MySQL username and password from the previously configured database. If the API is deployed as a jar, make sure the processPath attribute is set to " d:\home\site\wwwroot\bin\api\startup.bat"

For more information regarding debugging web.config, see:

http://www.iis.net/learn/extensions/httpplatformhandler/httpplatformhandler-configuration-reference

G – User Tests

Test Case 1

User: Pay3 Pal (q@q.no)

Role: organization administrator

Task 1

Login as an organization with
email: q@q.no
pwd: Hei123

Task 2

Create a new contract category to your organization. Name the category whatever you like.

Task 3

Create a new group to your organization. Name this group 'IMT'.

a. Add the user 'Pay Pal' and assign the category you added in task 1.
b. Let the IMT group get the *Modify right* priviledge to the added category.

Task 4

Add a new contract with the following criterions as a minimum (NB: read all requirements before starting to add the new contract)

1. Add it to the category you created in task 2.
2. Add 'HIG' as a party to the contract
3. The contact started on the 01.april.2015 and lasts for 1 year
4. Write a note to the contract with whatever content.
5. Save the contract

Task 5

Edit the contract you added. Add an attachemnt, and delete your note.

Task 6

Edit a contract that is not active. In other words, undraft the '1234' contract.

Test Case 2

User: Thomas Mellemseter (b@b.org Hei123)

Role: organization administrator

<u>Task 1</u>

Log in as an organization user and change the profile picture.

email: b@b.org

pwd: Hei123

<u>Task 2</u>

Edit the next expiring contract. Edit the name to 'AUDI R8'.

<u>Task 3</u>

Add a new contract with the following requirements as a minimum (NB: read all requirements before starting to add the new contract)

1. Mark the contract as automatically renewal.
2. Add an attchement.
3. Write a note to the contract with whatever content.
4. Do NOT save the contract, and navigate to overview when finished with the above requirements.

<u>Task 4</u>

The contract you added is not vaild. You do NOT want to delete the contract, but archive it instead.

<u>Task 5</u>

Export the 'HIG IMT' contract to .pdf, and add a new note to the contract.

Test case 3

User: Thomas mellemseter (b@b.no Hei123)

Role: Home User

<u>Task 1</u>

Login in as a home user with:
email: b@b.no
password: Hei123

<u>Task 2</u>

Add a contract with the following requirements (Feel free to add other optional input fields)

a. Reference = KJPOS123
b. The contract lasts for 20 months.

<u>Task 3</u>

Share the contract you created in Task 2 with d@d.org

Task 4

Export all EXTRA (category) contracts to one .xls file (HINT: It should be 3 contracts)

Task 5

Delete the contract you created in Task 2.

Questions

1. What was it like to add a new contract, could something be better?
2. What was it like to find a contract you were looking for?
3. What was it like navigating through the app?
4. Other comments, concerns you would like to share?

H – Email Communication with the Norwegian Data Protection Authority (Norwegian)

Fra: juridisk <juridisk@Datatilsynet.no>
Sendt: 12. februar 2015 09:24
Til: martin.nyflott@hig.no
Emne: VS: Vurdering av system ang. konsesjon

Hei.

Vår svartjeneste gir deg kortfattet rådgivning. Vi vil derfor ikke konkludere i saken din, men gi deg råd og veiledning.

Det er i utgangspunktet meldeplikt for behandling av alminnelige personopplysninger, jf. personopplysningsloven §§ 31 og 32, og konsesjonsplikt for behandling av sensitive personopplysninger jf. § 33. Se hva som omfattes av definisjonen sensitive personopplysninger i lovens § 2 nr. 8. (se link til loven her: http://lovdata.no/lov/2000-04-14-31).

Det er imidlertid gjort unntak for konsesjon- og meldeplikt for behandling av sensitive kundeopplysninger dersom den registrerte har samtykket til registreringen og behandlingen av sensitive personopplysninger, og opplysningene er nødvendige for gjennomføringen av en kontraktsforpliktelse, jf. personopplysningsforskriften § 7-14 (se link til forskriften her: http://lovdata.no/forskrift/2000-12-15-1265).

Det er også gjort unntak fra meldeplikten for behandling av alminnelige personopplysninger om kunder, abonnenter og leverandører, dersom personopplysningene behandles som ledd i administrasjon og gjennomføring av kontraktsforpliktelse, jf. forskriften § 7-7.

Oppsummert foreligger det altså verken meldeplikt eller konsesjonsplikt for deres prosjekt, forutsatt at behandlingen av personopplysninger faller innenfor bestemmelsene og beskrivelsen over.

Jeg legger til at all elektronisk behandling av personopplysninger reguleres av personopplysningsloven, og at lovens øvrige bestemmelser derfor vil gjelde for prosjektet. Dere er da pliktige å følge personopplysningslovens regler om hvordan opplysningene oppbevares, hvem som har tilgang til opplysningene, sletterutiner osv. For å gi dere en oversikt over reglene om dette, legger jeg ved en link til vår veileder om informasjonssikkerhet og internkontroll. Se link til veileder her: http://www.datatilsynet.no/Sikkerhet-internkontroll/internkontroll_informasjonssikkerhet/

Håper dette er til hjelp for deg. Hvis noe er uklart, eller du har flere spørsmål, kan du kontakte oss igjen ved å svare på denne e-posten.

Vennlig hilsen

Kristine Hagtvedt Holte
juridisk rådgiver, Datatilsynet
………………………………………………………………………………
Telefon: (+47) 22 39 69 00
Postadresse: Postboks 8177 Dep, 0034 Oslo .
Besøksadresse: Tollbugata 3, Oslo

www.datatilsynet.no
www.personvernbloggen.no

www.twitter.com/datatilsynet
Datatilsynet – i front for retten til selvbestemmelse, integritet og verdighet

Fra: Martin Storø Nyfløtt [mailto:martin.nyflott@hig.no]
Sendt: 8. februar 2015 13:56
Til: Postkasse
Emne: Vurdering av system ang. konsesjon

Hei,

I sammenheng med utviklingen av et IT-system under bacheloroppgaven til gruppen min, ser vi det nødvendig å utføre en vurdering om systemet vårt krever konsesjon eller om det er tilstrekkelig med melding til datatilsynet. Vi hadde derfor satt pris på om dere kunne gjøre en vurdering av vårt system slik at vi eventuelt kan søke om konsesjon før systemet er ferdigutviklet og skal settes i drift.
Systemet er et arkivsystem for kontrakter. Dette systemet skal da brukes av både bedriftsbrukere og privatbrukere og skal være tilgjengelig som en web-applikasjon samt mobilapplikasjon. Hensikten med systemet er at brukere skal kunne legge inn kontrakter og deretter kunne enkelt hente ut kontraktene når det er ønskelig, samt få varsler når kontrakten er i ferden med å gå ut. Kontrakter kan for eksempel være husleie, bredbånd, mobilabonnement, arbeidskontrakt, forsikring, m.m.
Når brukere registrerer seg, lagres fornavn, etternavn, epost-adresse og et passord i databasen. Disse opplysningene er felles for både privat og bedriftsbrukere. Bedriftsbrukere blir i tillegg registrert med bedriftsnavn og organisasjonsnummer.
Kontrakter brukere registrerer i systemet kan deles med andre brukere ved å oppgi hvilke brukere skal kunne lese/endre kontrakten dersom brukeren ønsker dette. Det er opp til brukeren hva slags informasjon som skal legges inn om kontraktene og hvilke kontrakter som skal legges inn i systemet.
Følgende metadata er forskjellige felt brukeren fyller ut når en kontrakt legges inn og blir lagret i systemet:

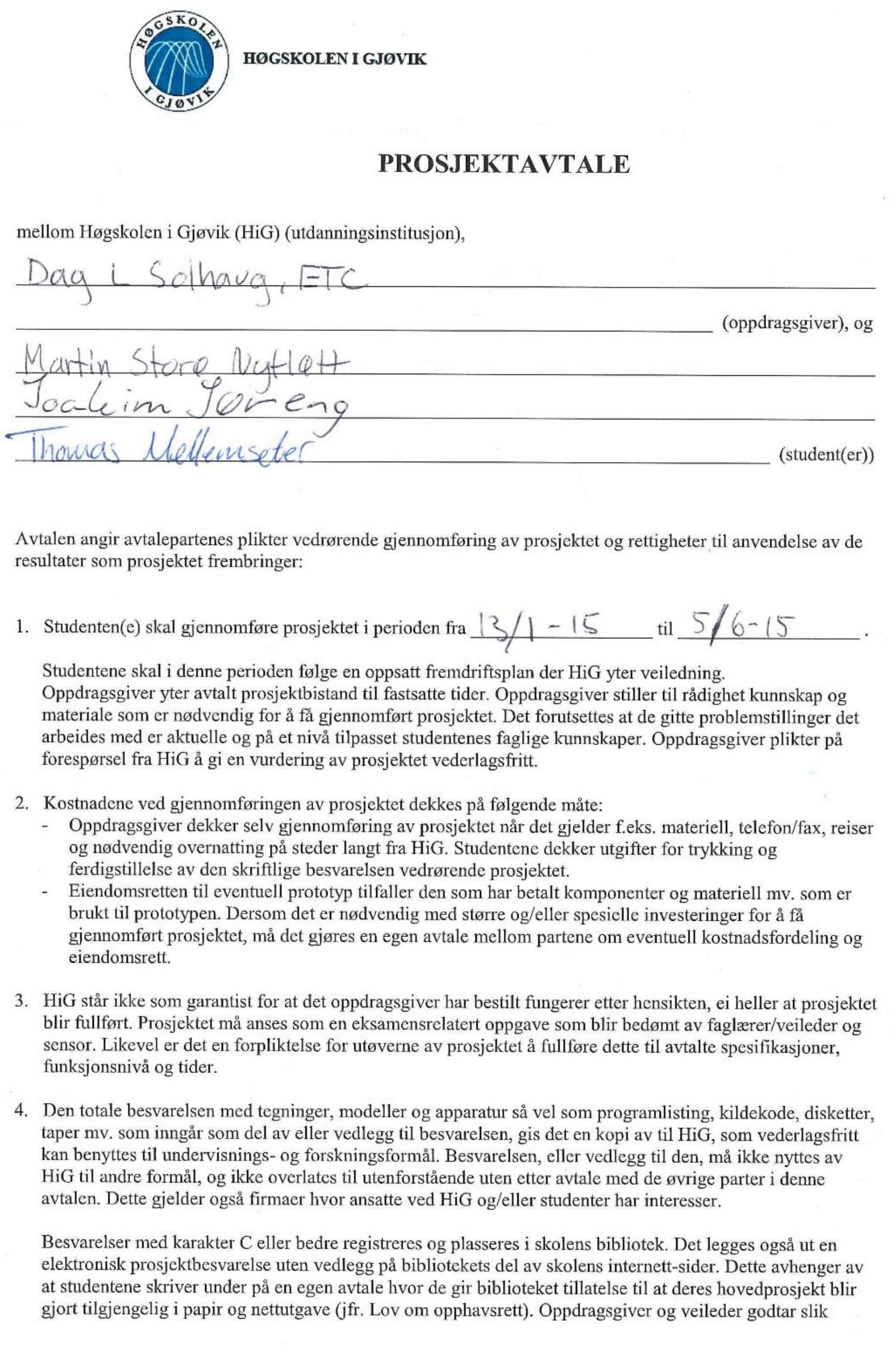
- Tittel på kontrakt
- Navn på personer/bedrifter som er en part i kontrakten
- Tidsrom for når kontrakten er gyldig
- Start/utløpsdato
- Tidspunkt for fornyelse av kontrakt
- Hvilke personer som er ansvarlige for kontrakten
- Tidspunkt for når brukeren skal bli varslet om at kontrakten skal fornyes eller er i ferden med å løpe ut og om denne varslingen skal gå via epost eller om brukeren skal få varsling inne i selve systemet

- Kategori for kontrakt, for eksempel. Forsikring, bank eller mobil. Brukeren kan selv administrere hvilke kategorier som er tilgjengelige.
- Ekstra notater som brukeren ser kan være nyttig å legge inn.

Brukeren kan laste opp selve kontrakten i tillegg til metadataene til kontrakten som kan være et scannet dokument, et bilde av kontrakten med mobiltelefon, eller et Word/OpenOfice/PDF-Dokument. Disse dokumentene blir indeksert og gjort søkbare i systemet for eieren av kontrakten slik at brukeren enkelt skal kunne finne igjen en kontrakt som har blitt lagt inn.
I tillegg kan brukerne abonnere på systemet via PayPal. Betalingsopplysninger brukerne bruker blir lagret og behandlet hos PayPal og ikke i vårt system.
Beskrivelsen over gjelder hvilke opplysninger blir lagret om brukere og hva slags informasjon de kan legge inn i systemet. Dersom noe av dette er uklart hadde vi satt pris på om dere kunne komme tilbake til oss ang. dette slik at vi kan utdype/forklare det som er uklart.

Mvh.
Martin Storø Nyfløtt

I – Project Agreement



HØGSKOLEN I GJØVIK

PROSJEKTAVTALE

mellom Høgskolen i Gjøvik (HiG) (utdanningsinstitusjon),

Dag L Solhaug, ETC

(oppdragsgiver), og

Martin Storø Nyfløtt

Joakim Søreng

Thomas Mellemseter (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 13/1 - 15 til 5/6-15.

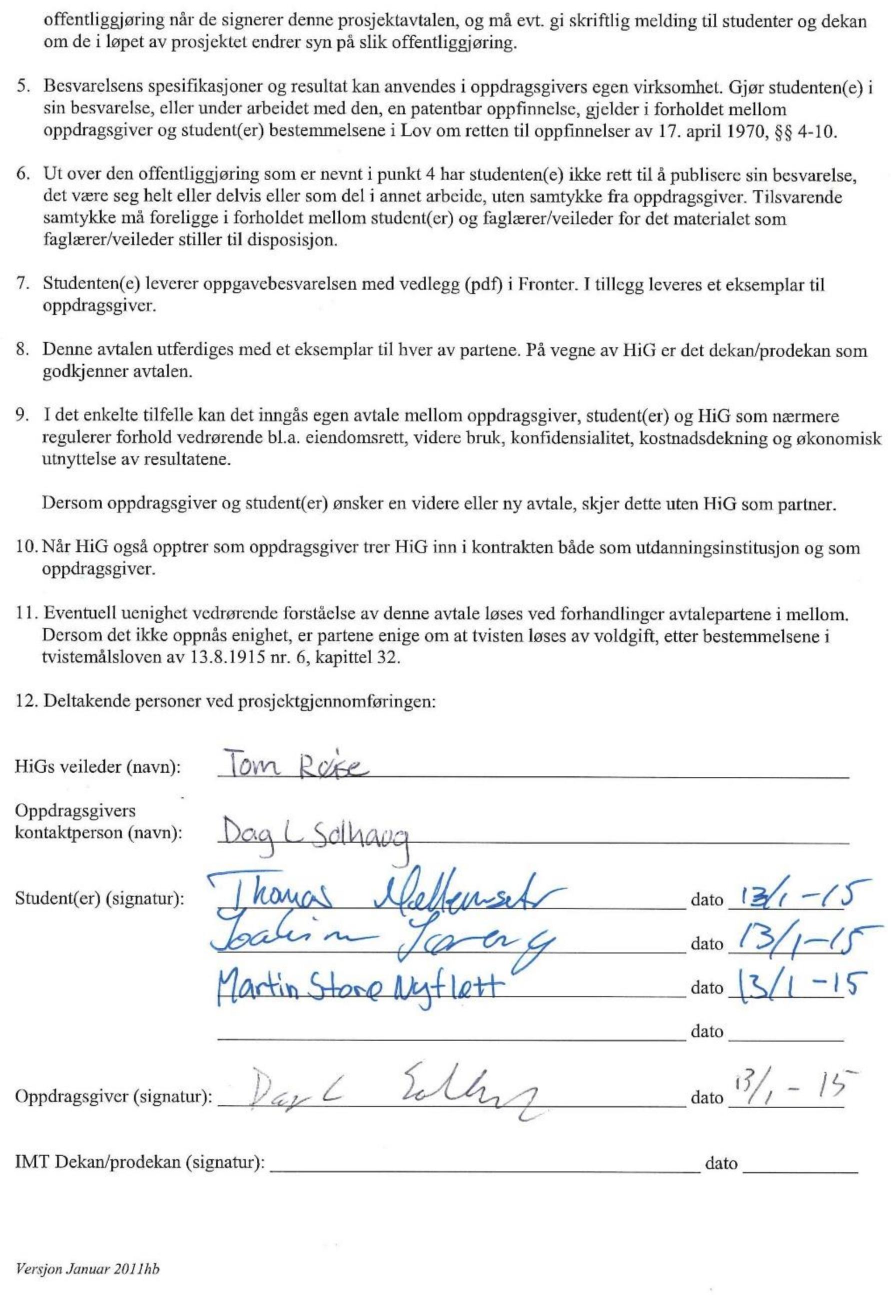
   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der HiG yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger 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): Tom Røise

Oppdragsgivers kontaktperson (navn): Dag L Solhaug

Student(er) (signatur): [illegible] dato 13/1 -15

[illegible] dato 13/1-15

Martin Stone Nyflott dato 13/1 -15

_______________ dato _______

Oppdragsgiver (signatur): [illegible] dato 13/1 - 15

IMT Dekan/prodekan (signatur): _______________ dato _______

Versjon Januar 2011hb