dedication (optional)

Summary

Write your summary here...

Preface

Write your preface here...

Table of Contents

List of Tables

List of Figures

Abbreviations

Symbol = definition

Chapter 1

Introduction

1.1 Equations

To write an equation

```
\begin{eqnarray}\label{eq1}
F = m \times a
\end{eqnarray}
```

This will produce

$$F = m \times a \tag{1.1}$$

To refer to the equation

```
\eqref{eq1}
```

This will produce (1.1).

1.2 Figures

To create a figure

```
\begin{figure}[h!]
  \centering
    \includegraphics[width=0.5\textwidth]{fig/pikachu}
  \caption{Pikachu.}
\label{fig1}
\end{figure}
```

To refer to the figure

```
\textbf{Fig. \ref{fig1}}
```

This will produce **Fig. 1.1**

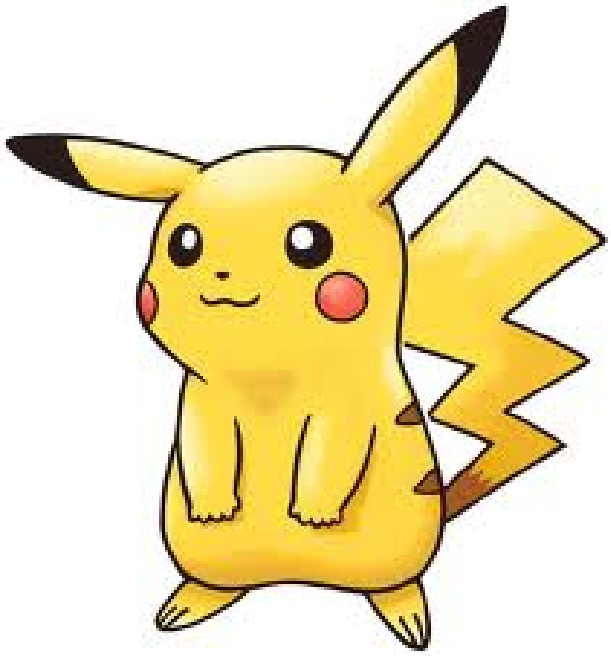

Figure 1.1: Pikachu.

1.3 References

To cite references

```
\cite{1,2,3}
```

or

```
\citep{1,2,3}
```

This will produce: Sarma and Chen (2008); Brouwer and Jansen (2004); Muskat (1937) or (Sarma and Chen, 2008; Brouwer and Jansen, 2004; Muskat, 1937), respectively.

1.4 Tables

To creat a table

```
\begin{table}[!h]
\begin{center}
    \begin{tabular}{ | l | l | l | l |}
    \hline
    \textbf{No.} & \textbf{Data 1} & \textbf{Data 2} \\ \hline
     1 & a1 & b1 \\ \hline
     2 & a2 & b2 \\ \hline
```

```
    \end{tabular}
\end{center}
\caption{Table 1.}
\label{Tab1}
\end{table}
```

This will produce

No.	**Data 1**	**Data 2**
1	a1	b1
2	a2	b2

Table 1.1: Table 1.

To refer to the table

```
\textbf{Table. \ref{Tab1}}
```

This will produce **Table. 1.1**.

Chapter 2

Literature Review

Chapter 3

Basic Theory

Chapter 4

Experiment

Chapter 5

Analysis

Chapter 6

Conclusion

Abstract

This report describes a software development project completed by seven students in the IT2901 Informatics Project II course at Norwegian University of Science and Technology (NTNU), the spring of 2015. The task was given by SINTEF, and it consisted of creating a mobile app which should recommend the user cultural heritage stories from the existing Digitalt fortalt platform. Stories from this platform are recommended to end users based on personalization techniques, which is about taking in context information and feedback from the users in order to make informed choices about what content to present. The report contains a thorough pre-study of other projects based on similar subjects. A full documentation of the development process from beginning to end, including all aspects such as project planning, product specification, implementation, testing, and customer relations can also be found in the report.

Chapter 7

Intro

This chapter introduces the customer, the team, and the projects definition and purpose.

7.1 Stakeholders

7.1.1 Customer

The employer for this research project was SINTEF, in this case represented by Jacqueline Floch and Shanshan Jiang. SINTEF is an independent multidisciplinary research organization within technology, science, social science and medicine. The organization has also provided assignments for the course IT2901 in the past.

7.1.2 Team

Table 1.xx lists the persons in charge of developing the project in this report, and some of their background competencies.

TODO sett inn Table 1.xx team description

Within the team there was a good amount of general experience with web design and computer application design. Some members had experience with making databases, resulting in not having to dedicate extra time for researching this topic. There was a mix of valuable experience between front end and back end development, as well as knowledge about project management and how to relate to various actors such as users and other stakeholders.

7.2 Project description

In the course IT2901 [es20], Informatics Project II, at Norwegian University of Science and Technology (NTNU), the main assignment was to develop a software project for a customer, and was

completed during the spring semester of 2015. The goal of the course was to gain practical experience with the development of a software process for a customer, covering the whole life-cycle of the software project.

The project in this report is named Personalized storytelling. The purpose of this project was to create a cross-platform application (iOS and Android) which would allow users to discover personalized cultural and historical stories based on context-sensitive information and personal interests. The application is a part of the TagCloud [es21] project, which is a project whose aim is enriching cultural and historical experiences through innovative mobile applications. There have been similar projects worked on in the past, one example being stedr, which is an application based on discovering and sharing stories about locations around you (see chapter 3.3.1 for details about stedr). This project was different in the way that it was directly connected to the personalization of stories, and as such, multiple ways of personalization had to be integrated to find a good recommendation for the user.

7.3 Problem description

Even though there is a rich cultural heritage in Norway, many people do not participate in cultural activities. Museums and other cultural institutions have tried to increase interest with innovative exhibitions and tools, but with little luck. The motivation behind this project was to find out how effective personalization is in engaging more people in their cultural heritage. This was done by creating an application that picks out personalized stories based on the user's interests and context, and will thus encourage exploration and find relevant and interesting stories to the user.

The target audience for the application included both those who have an interest in cultural activities, and also those who do not have any interest or experience about this, so as to encourage more of the general population to discover an interest in the subject. A specific target audience of 16-19 year old teenagers were promoted as a possible focus, because of the possibility of encouraging a young audience to develop an interest in cultural heritage.

To summarize, this report details the entire development process of a mobile application, developed for Android and iOS. The application will provide its users with cultural stories in a personalized manner, with the goal of generating more interest in cultural heritage. In the next chapter, a thorough pre-study of all the relevant technologies and methodologies that were considered to accomplish this task will be presented.

Chapter 8

Pre-study

This chapter discusses the research that had to be done, and the choices that the team made in relation to choosing development frameworks and technologies, as well as a development process for the project. The chapter also describes some already existing applications within the same subject as this one, and a study about personalization which was a key aspect for this project.

8.1 Project assumptions and constraints

In this section, assumptions and known constraints for the project are addressed. The list of assumptions are details assumed ahead of the documented project requirements, while the constraints specifies matters which would impact the project.

Assumptions on which the planning of the project was based:

- Access to an API for Digitalt fortalt would be provided by the customer.
- The customer would give access to a server to deploy the back end of the application on.
- The frameworks chosen for development had the features needed.
- The customer would be available for weekly meetings
- Each member of the team would be able to work about 20 hours a week.
- Team would meet at least three times a week.
- Team would follow the set ground rules.

Constraints which the team had to work within throughout the project:

- The deadline for delivering the project was May 30th, and there was no possibility to extend it.
- The front end should be developed for both the iOS and Android platforms.

- The team consisted of 7 people, with little previous experience in mobile app development and no knowledge of personalization algorithms.
- The application is under the Apache license version 2.0 [HM7]. In summary, this grants copyright and patent rights for users to further distribute it, or distribute a modified version using the same license. It should be clear what the eventual modifications are. The source code can also be used as a part of a closed source project.
- No budget to pay for software tools.

8.2 Choice of framework

As one of the requirements from the customer was that the application should be cross-platform (Android and iOS), a hybrid app was found to be the best option. The alternative would have been to create two separate native apps for both platforms. However, this would have been too much work to complete within the deadline, especially as the team had little experience with developing for either of the platforms. A hybrid app is a web app made with HTML and JavaScript wrapped in a native shell so that it can be run like a normal native app. This was a big advantage, as the team already had experience with web design. It also makes it possible to make use of the many tools available for web development, and most of the code can be reused for multiple platforms. Bad performance used to be a disadvantage to the hybrid approach, however mobile hardware has improved significantly in the last few years, so this is not a large issue any longer. The following sections will discuss the advantages and disadvantages of different frameworks that were considered, and explain the final choice for this project.

Table 2.xx below summarizes some of the capabilities and limitations of the various frameworks that were under consideration.

TODO sett inn Table 2.xx Framework comparison

8.2.1 PhoneGap

PhoneGap is an open-source mobile app framework for native packaging [RA2]. What it does is take in a mobile app consisting of HTML, CSS and JavaScript files, and wrap it in a native shell. It can then be deployed to iOS, Android and Windows 8. It also gives easy access to the native features of the phones (geolocation, notifications, storage, etc.) through different APIs. It is not necessary to think about the native SDKs, as the app will be compiled and built with the newest SDK for the platform. It is a very popular framework, and there are many plugins created for it that provide additional functionality.

8.2.2 Ionic

Ionic is an open-source UI framework focused on making it easier to create hybrid mobile apps with a native feel [RA1]. It accomplishes this by offering a foundation to build on and UI components

based on design patterns and best practices found in native apps. The foundation can be built on and customized with additional HTML, CSS and JavaScript. Part of the foundation is AngularJS, which is a JavaScript framework that extends HTML to make dynamic views in web applications. It gives the app a modular architecture, which means that code can more easily be reused for both iOS and Android. A disadvantage is that it is necessary to take time to learn AngularJS in order to take full advantage of Ionic. As Ionic is one of the most popular hybrid mobile frameworks, there exists more learning material about it, and it also has an active user community. Performance is not quite as good on older devices, especially if using large amounts of animations or media.

8.2.3 Appcelerator Titanium

Titanium is a cross-platform Javascript runtime and API framework [RA3]. It currently supports iOS and Android. It offers a JavaScript API which gives access to native UI components and features for the specific platforms, instead of trying to replicate it with CSS or JavaScript like other frameworks. This gives the app a performance advantage, and it is easier to make the interface and interactions feel native. Because the APIs are platform-specific you have to write separate versions of your app for the different platforms. Another disadvantage is that it is difficult to debug as there is no good debugger and Titanium projects cannot be run in Xcode. There would also have been more to learn to be able to use it, as it doesnt use HTML/CSS.

8.2.4 Sencha touch

Sencha touch is a mobile framework with a large number of UI components and an architecture for the front end [RA4]. Instead of enhancing a HTML file, it generates the DOM with JavaScript. It can be used with PhoneGap, but Sencha also has its own native packager. It is harder to learn than Ionic, and the performance is not as good. It supports the platforms iOS, Android, BlackBerry, and Windows Phone.

8.2.5 Conclusion

The framework combination chosen for this project was Ionic and PhoneGap, as this seemed to fit this projects properties and requirements the best. They are both some of the most popular hybrid frameworks and they are a common combination to use. A prototype can quickly be set up with Ionic, and then iteratively customize it to fit the requirements and create a good user experience. One of the non-functional requirements was that it should be easy to extend the app and to reuse parts of it for other apps. This is fulfilled by Ionics modular structure. PhoneGap makes it easier to wrap the code in a native iOS and Android shell. This process could also have been done manually, but it requires some knowledge about the native code languages and SDKs. The many plugins available for PhoneGap should also cover the need for use of native features.

8.3 Software development process

The choice of which development process to use in the project was a central decision to be made. The following sections describe the different models that were under consideration by the team, as well as some advantages and disadvantages of each, which influenced the decision of which one to be used for the project. Table 2.xx below gives a comparison of some aspects of the various processes that were considered relevant for the project.

TODO sett inn Table 2.xx Development process comparison

8.3.1 Waterfall model

The waterfall model is a plan-driven process with well-defined phases. These phases normally include requirement analysis, system design, implementation, testing, and maintenance [es16 P.30-32]. It is necessary to finish one phase before starting the next, and due to this, most planning and decisions need to be made at an early stage in the development. As such it is difficult to respond to changes in the requirements. Another issue with this model is that iterations often involve a significant amount of rework and it is normal to postpone some parts of the iterations in order to continue with the later stages of development. This can lead to errors in the system as well as bad design choices.

8.3.2 Extreme programming

Extreme programming is an agile method focused on pushing out new system versions and functionalities rapidly [es16 P.64-72]. All requirements are written as user scenarios, and before writing the code it is necessary to develop tests for the task. Team members program in pairs and when the code passes all the tests, it can be integrated into the system. It is common for a customer representative to take part in the development and make acceptance tests. New system releases are regularly presented to the customer, and this way it becomes easier to cope with changing requirements. Some of the drawbacks of extreme programming include the lack of overall plans for the project. Several documents such as design details and overall report are left out, and it lacks a solid plan for when to implement the various functionalities.

8.3.3 Scrum

Scrum is a general agile method with focus on managing iterative development rather than specific technical engineering approaches [es16 P.72-74]. It also allows for a rapidly changing development environment and close collaboration between the members of a team. To provide this, scrum makes use of phases called sprints, daily scrum meetings, and several types of charts and logs. One of the main challenges for a scrum team is to choose the right amount of work per sprint so that they dont end up with too little or too much work. Scrum has several similarities with extreme programming, such as high involvement of the customer in the development, as well as continuous testing while implementing new functionality.

8.3.4 Conclusion

For this project, the agile software development methodology scrum was used. The project was not very well defined from the beginning, because the customer was not entirely sure of exactly what they wanted. The team also became more productive, as the deadlines were short and so it was possible to quickly make a simple application which could be tested by users/customer. Also, an agile process was beneficial as it allowed the team to be flexible and rapidly respond to changes in requirements. The customer requested weekly meetings, and it became a logical decision to make use of scrum to have 1-week sprints, so there would be definite progress to show between each customer meeting. Since the team was a group of seven, it was impossible to always work together. Because of this, the regular scrum meetings were beneficial to share and discuss progress.

8.4 Back end

8.4.1 Docker

Docker[EHW2] was made to help automation of application deployment. This happens by providing a virtual operative-system-level abstraction. This means that on a server, it is possible to run several virtual operative systems called docker images, which can easily be deployed to another server. This is beneficial for software development, because it means it is easy to setup identical back ends. The customer used this on their servers, which made using it during development as well, a good choice. A benefit of using docker, is that it can directly access repositories on git. This means that the latest revision is guaranteed to run when starting the back end. However, there are also drawbacks. It is not possible to update files within the docker image currently running without rebuilding it. This means that to update a single line of code, the whole image needs to be rebuilt. This means that while the newest revision is guaranteed to be running, any changes made to the application after the start of that docker image requires manually stopping, rebuilding, and starting the docker image.

8.4.2 Language

8.5 Personalization algorithms

To provide story recommendations in accordance with each users interests the content needs to be personalized. Personalization involves using technology to tailor content, to individual users characteristics or preferences, and to accommodate the differences between individuals. It is a way of meeting the users needs by making interactions faster and easier, which will hopefully increase customer satisfaction and the likelihood of repeat visits. Personalization may be achieved using recommender systems.

8.5.1 Recommender systems

Recommender systems are software tools and techniques that attempts to provide recommendations of items [HM4]. Such systems are simply information filtering systems with the goal of providing suggestions for items to be of use or interest to a user. A few examples of items used in this context are movies, music, books and products in general. Recommender systems typically produce a list of recommendations. The two most common approaches to produce such a list are content-based filtering and collaborative filtering.

8.5.2 Content-based filtering

Content-based filtering methods are used to find similarities between a users preferences and the description of an item [HM5]. These algorithms try to recommend items that are similar to items a user has liked in the past or is looking at in the present. Items that a user likes or has interacted with can be seen as a part of the users profile. Content-based filtering depends on there being much descriptive data available on the items. To find items to recommend, items are compared against a users profile, and recommendations are given based on how well they match the profile. User feedback, usually in the form of rating or a like or dislike button, can be used to assign weights to certain attributes. By using user feedback and weighting it is possible to give more accurate recommendations. [HM4]

8.5.3 Collaborative filtering

Collaborative filtering is based on collecting and comparing information on users behavior, activities or preferences and to recommend items based on a users similarity to other users [HM6]. This approach tries to predict what a user will like based on what similar users have liked. Collaborative filtering assumes that users who have agreed in the past will agree in the future, and that they will like similar items as they liked in the past. These methods often suffer from the problems cold start and sparsity. Collaborative filtering often requires a large amount of existing data on users to be able to make accurate recommendations. The cold start problem is the absence of such data at the beginning of a project. The sparsity problem is that collaborative systems are dependent on having many active users to properly distribute ratings across all the items in the system. However, most active users have only rated a few items in the overall database, which means that even the most popular items have very few ratings. The greatest strength of these techniques is that they are independent of any documented representation, e.g. textual descriptions and subject-tags, of the objects being recommended and work well for objects that are difficult to define such as music and movies. [HM4]

8.6 Existing solutions

Among the previous work there has been developed applications through the TagCloud project. This section evaluate two of those applications, namely stedr and Cooltura. Both of these applications presents stories regarding cultural heritage. Since this project also included personalization,

an evaluation was additionally performed on the application Magic Tate Ball. This application was chosen because the customer mentioned this as a possible inspiration for the current project.

These three applications were evaluated using the following criteria:

- Content. Does the application provide satisfying content or is something lacking?
- Usability. Usability concerns how easy it is for the user to accomplish a task. (see section XX.XX (non-functional requirements) for a more elaborate definition). The evaluation here draw on Jacob Nielsens ten usability heuristics as defined in [AS3].
- Personalization. To what extent does the application provide the user with the opportunity for individualized content?

8.6.1 stedr

The stedr application was developed by students as a prototype to test some research hypothesis. Content was not the primary focus in the application and thus consists mainly of stories related to places in Trondheim. Each place in the application is composed of a main picture and some brief general info about the place plus stories, pictures and sounds. Most of the places have one or more stories related, but pictures and sounds are less common. A possible problem in the application is finding the place the user is looking for. This is not easy unless the user have prior knowledge of the geographical area the desired place is located in. The reason for this is that the applications main view consist of a map with markers representing each available place, but in order to know which place each marker represents the user have to click on the marker. If the user doesnt know where for instance Nidarosdomen is located in Trondheim, they may have to click on all markers on the map before being able to navigate to the Nidarosdomen place view.

Our application does not use a map, so the most relevant views are the place view and the story view. The place view consist of a main picture and three tabs for different ways to explore this story as seen in Figure 2.XX. When changing the tabs, the content below the tabs changes. To see all the content, one might have to scroll down. The main picture is static when scrolling, occupying the upper half of the screen. This is also the case when viewing a specific story in the story view. Nielsens heuristic of aesthetic and minimalist design states that: Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.[AS3] It might be argued that when scrolling for content below the tabs or reading the story in the story view, the main pictures importance in the dialogue is diminished and should not compete as strongly with the other information as it does now.

TODO sett inn Figure 2.XX: A screenshot from stedr showing the available options for one place

Other than finding information, the application also provides the user with the opportunity to add content themselves. Unfortunately this is a rather cumbersome process, as adding stories has to be done through Digitalt fortalt, adding pictures has to be done through Instagram and adding sound has to be done through Soundcloud. According to Nielsen, it is important to avoid making

the user remember information from one dialogue to another. When adding pictures and sounds, specific words and/or tags has to be added on Instagram and Soundcloud. The user doesnt have to remember these words since the application provides the opportunity to add them to the clipboard after clicking Add picture or Add sound, from where it can be fetched when adding the picture or sound. However, the use of the clipboard can be problematic because it hides the content from the user. It may not be obvious to all users how to paste the content into Instagram or Soundcloud, and such users might be forced to remember the words or abandon the action altogether.

The application provides a help site which gives a thorough guide to the user on how to accomplish these tasks. This is in accordance with Nielsens heuristic of help and documentation. The application also helps the user when errors occur, for instance by offering the user a refresh opportunity when the map does not load the places. The language used in the application is both user-centered and consistent, avoiding misunderstandings and making it easy for the user to understand what different actions entails.

The application does not provide any personalization feature, which is the main difference between stedr and our application. The customers motivation for creating an entirely new application instead of expanding stedr was mainly because they wished to test personalization systems on users, and thus needed an application focused mainly around the personalization aspect. In addition, it became possible to access a larger number of stories than stedr currently does.

8.6.2 Cooltura

Cooltura is another application developed under the TagCloud umbrella. The version of Cooltura evaluated here is a demo version, so more developed version of the application may include more functionality and address some of the issues discussed here.

The content is somewhat similar to stedr, with respect to the places and the stories available. In fact, when clicking to view stories in Cooltura one is directed to the stedr app, however not to the relevant story but the main view. Because of this, the user might have to remember what place it was visiting in Cooltura to see the stories connected to the place in stedr. While it might seem trivial to remember this, the user might be interrupted somewhere in the process and to complete it later it have to either remember the place or start the process all over again. As Nielsen has pointed out, a system should try not to make the user remember information, but rather recognize it. A better solution would therefore be to direct the user to the relevant place view in stedr, so that when the user later opens stedr it will open on the chosen place view and the user will recognize the action it was performing. Cooltura provides a more thorough description of each place, but other than that the content is the same as in stedr. Cooltura does not use a map like stedr, but instead a list of available locations. This makes it far easier for the user to navigate to the right place, especially since the selection is quite limited.

TODO sett inn Figure 2.XX: A screenshot of Cooltura showing recommended places TODO sett inn Figure 2.XX: A screenshot of Cooltura showing the view for one particular place

When viewing a specific place on Cooltura the user gets a view with a main picture and some text describing the place as seen in figure 2.XX. Like in stedr the user can scroll down to see more content, but unlike stedr the main picture is not static, that is, the user see less and less of it as it scrolls down. This is more in accordance with Nielsens point of aesthetic and minimalist design, where less relevant information fills up less space in the user interface. The same is true for the view for a specific tourist attraction.

The application does not provide any help site, but the possible user actions are quite few and similar (most of them are about navigating to the desired place), so this might not be a problem.

It appears that the application intends to make use of personalization, but this is not implemented at the time of writing this report[AS4]. The Anbefalte steder-view in figure 2.XX now list every place added in the application (which is three different places), but the heading suggests that personalization is planned for.

8.6.3 Magic Tate Ball

Magic Tate Ball is an application that presents the user with an artwork based on the input parameters: date, time-of-day, geographical location, live weather data and ambient noise levels [AS5]. The content in the application is the artwork, a description of the artwork and an explanation of why the artwork was chosen. It is unknown how many artworks there are, but the information given on each artwork seems adequate.

The main task in the application is to be presented with an artwork. This is done by shaking the phone or by clicking a button in the center of the magic ball, see figure 2.XX for a screenshot of this. Another task is to browse all artworks presented to the user, which is done by swiping or clicking on arrows. Providing these two options makes it easy to for different types of users, those accustomed to swiping and those who are not. When the application is processing to come up with an artwork to present to the user, it shows the user what the input parameters are. This is in accordance with Nielsens heuristics on visibility of system status.

TODO sett inn Figure 2.XX: A screenshot of the main view of Magic Tate Ball

Personalization is the main selling point of Magic Tate Ball. The application uses the five input parameters described above to give the user some control of what content is presented. The user can turn each of these parameters on and off. What the user cannot do however, is to specify preferences regarding art. To that extent, the user has no direct control over what artwork it will be presented, because it is hard to know what to expect when an input parameter for instance is 10 sunny weather. Even though the application provides an explanation to why an artwork was chosen, how it was chosen remains in the dark (e.g. how to know which input parameter was emphasized in the personalization).

Chapter 9

Project management

This chapter describes how the team delegated work tasks and responsibilities, and also how the work was broken down into manageable tasks. The chapter also describes time management and quality assurance for the project.

9.1 Scrum team and roles

The role delegation in the team is detailed in table 3.xx below. The delegation of roles was primarily based on personal interest and motivation. The tasks were divided up into main responsibility areas for back end and front end before assigning people to each one. However, this was only a guideline for main responsibilities, and the group members had to be flexible and work on some tasks outside of their main areas.

TODO sett inn Table 3.xx role delegation

9.2 Work breakdown structure

A work breakdown structure is a decomposition of the project and its goal is to break down each part of the development process into manageable parts to ease the planning and execution of the development. Each element in the diagram can be a product, data, service or a combination of the three. One of the benefits of detailing a project this way appears when doing cost estimation and scheduling the team around the project (i.e. should ease the project planning and help allocate the teams resources).

The work breakdown structure should show a hierarchical decomposition of the project phases and its components. Each main phase is at a top-level and will outline the generic parts of the software development processes. The way the WBS (Work Breakdown Structure) is developed is by starting with the end objective and subdividing each main part into manageable components in terms of size, complexity and duration. Each sub objective is to follow the 80 hour rule. This

means that a subtask is not to exceed 80 hours in magnitude. The WBS for this project is shown below in figure 3.xx.

TODO sett inn Figure 3.xx Work breakdown structure

9.3 Project milestone plan

TODO: put the Gantt and burndown chart here and describe the time/sprint planning.

Milestones are used as tools in project management to give the team some clear and specific goals to work towards as the project timeline moves ahead. There are several milestones throughout the project, some are large milestones like; alpha and beta versions of the software. There are also milestones related to the project report, like the midterm submitting. and final delivery. Milestones can add some value to the project scheduling when used in the right manner and when setting realistic goals. Components that are important for the milestone plan are; key dates, key deadlines and external deliveries. The team used a combined Gantt chart with milestones noted for better visualization, and to better allocate resources for meeting the milestone goals.

The planned deliveries to the customer is the following dates.

- 20.02.15 First prototype on paper
- 27.02.15 Second prototype presented in proto.io[KF2]
- 17.03.15 First working software(alpha-version)
- 10.04.15 Second working software(beta-version)
- 01.05.15 Final product

9.4 Quality assurance

According to Sommerville quality assurance is the definition of processes and standards that should lead to high-quality products and the introduction of quality processes into the manufacturing process [AS6]. In large systems, designed to be used in a long term perspective, quality documentation is important. However, in this project a small system was developed and Sommerville notes that a more informal approach can then be applied, focusing on establishing a quality culture [AS6] within the development team.

Therefore, this section will describe three features considered important by the group to establish such a quality culture, and thus improving the quality of the product, namely group interaction, version controlling, and interaction with the customer. Risk management and testing are also important aspects of quality assurance, and these are discussed in separate chapters(see chapter XX for risk management and chapter XX for testing).

9.4.1 Group interaction

As was noted in section XX.XX(Pre-study - Scrum) an important part of the scrum methodology is the close collaboration between the team members. Scrum provide some events to enhance this collaboration, for instance the daily scrum meetings, and some artifacts, such as the sprint backlog. These features of scrum was used by the group and created a framework for the process development. However, scrum does not define how the group should interact, and the group interaction consisted of more than the methods provided by scrum, for instance when doing sessions of collaborative work.

In order to ensure that the scrum events and the interaction external to these events would create the desired quality culture, the team discussed and agreed upon some basic rules of engagement for the project. These rules specified how the team should create a quality process in order to create a quality product, for instance by setting ground rules for communication between the group members. The tools described in section XX.XX (Tools-communication tools) were used to facilitate the implementation of the rules. In addition, meeting minutes from every meeting was made so that every group member would be aware of the status of the project even if they were not present at the meeting.

The division of the group described in section XX.XX (Project management - Scrum team and roles) meant that a member of the front end part would have more detailed information about what other front end developers were doing than what individual members of the back end part were doing, and vice versa. However, when important decisions were to be made in one part of the project or important problems had to be solved, both parts would be involved in the discussion, even if the decision did not affect their part of the project directly. An example of such a decision was the choice of colours to use in the user interface.

9.4.2 Git and version controlling

The code for this project is hosted by GitHub, a tool described in section XX.XX (Tools chapter). GitHub uses the version control system git, which makes development easier by allowing multiple local branches and thereby giving users the opportunity to try out code before committing them to the master branch. The master branch is the main branch and should only include stable code. The group chose to create two repositories, one for the back end part of the project and one for the front end part since the group also was divided in this way. Doing it this way made keeping track of branches and issues on Github clearer since these often would be at a level of detail only relevant to the developers in that part of the project.

9.4.3 Customer interaction

In section XX.XX(Pre-study-scrum) it was noted that the project was not clearly defined at the start. This meant that good customer interaction was critical, so that the group and the customer would be in agreement of what was expected of the end product. Communication with the customer was done by weekly meetings (some weeks were skipped in the later part of the project period because

there were no issues to discuss), mail and by a shared Dropbox folder. In addition, the customer had access to the code on GitHub.

A couple of days before the weekly meetings, the group would add a meeting agenda to the Dropbox folder. This was done to improve the structure of the meeting and to give the participants time to consider the issues on the agenda, which in turn should increase the benefits of the meeting and increase the likelihood of making decisions. Making decisions and coming to an agreement with the customer on key issues was considered important to push the project forward. After the meeting the group would add a minute of the meeting to Dropbox, so that the customer would know if all the participants had a similar understanding of the issues discussed and the decisions made. Communication by mail was mainly used to rescheduling of meetings and by the customer to give additional information to the group.

Chapter 10

Risk management

This chapter details the risk management of the project, which includes planning and handling all the various potential risks to the project.

The risk analysis below contains a list of possible occurrences that could be harmful to the project. Provided for each risk is a short description, an estimated likelihood that the risk will happen, an estimated impact to the project if it happens, the importance of the risk, a preventive action to try to avoid the problem and a remedial action if the problem were to occur. Likelihood and impact estimates were rated on a scale from 1 to 9, with 9 being the highest, and the importance was calculated by multiplying likelihood with impact. The risk list was updated regularly and sorted by the importance value, thus the risk to be most aware of at each stage of the development process was at the top of the list.

TODO sett inn Table 4.xx Risk list

Chapter 11

Requirements specification

This chapter describe the requirements for the application. The chapter is divided into two sections: one section describing the functional requirements, another that describes the non-functional requirements.

11.1 Functional requirements

The requirements document for this project can be found in appendix A. This document contains a description of the functional requirements. The requirements were elicited and agreed upon with the customer in meetings and formalized in the requirements document. In early customer meetings the functionalities of the application were discussed informally. The group wrote a requirements document which was discussed at subsequent meetings. In these meetings each proposed requirement was refined and given a priority. The priorities were assigned using a high-medium-low scale. This priority has guided the sequence of progress in the project. In addition, the group elicited its own prioritization where all the requirements were ranked, each with its own unique number ranging from 1 to the total number of requirements. The description of requirements in this chapter is less formal and more high-level than the one found in the requirements document.

11.1.1 Summary of the functional requirements

This section provides an informal summary of the main requirements that were initially proposed. These requirements have been detailed and refined in the use case section, as well as the functional requirements document.

Sign up/Sign in view: The application should in some way be able to identify users, but keep them as anonymous as possible. For research purposes, personal data like age group and gender should be collected.

Preferences/Settings: The user should be able to specify some preferences regarding cultural categories and the use of location from the device.

Main view: Browse recommended stories: The application should provide the user with recommended stories based on the users preferences. The user is provided with three choices on each story: to read it now, to reject it or to save it for later.

Story view: The application should present a chosen story in an clear way that respect the work of the author and resemble the presentation on the Digitalt fortalt website. Every story should also include a link to the corresponding story on Digitalt fortalt. The user should be given the opportunity to rate the story and to tag the story. In addition, the application should provide an explanation of why the story was recommended.

Collected story view: The application should keep a collection of stories that the user have been recommended and that have not been rejected. The user should be able to filter this list based on the status of the story (to-read, read, reviewed) or tags (predefined tags like Favorites or the users own tags).

Notifications: The application should provide the user with the opportunity to set a preferred time it wants to receive a notification about a new recommended story on their device. At the set time, the application should send the notification. In addition, the application should notify the user when a story is missing a rating.

About / help site: The application should include an about site, which explains the context in which the application was created.

Personalization: When recommending stories the application should employ both content-based filtering and collaborative filtering algorithms.

Research: The application should gather information about the use. This data is to be presented to the customer at SINTEF through a simple interface.

11.1.2 Use cases

The use cases in this section give an overview of the interaction with the system. In the requirements document one can find references to the use cases for requirements that involve external interaction.

A use case is a simple scenario that identifies actors involved in an interaction with a system and describe this interaction[AS1]. Actors are entities outside the system interacting with it to accomplish some task. This can be a human using the system or it can be some other system. In eliciting requirements in a software development process, use cases are particularly effective in making it clear what is expected of the system in terms of user interaction. A use case diagram is usually presented with ellipses and stick figures. The ellipses represent the use case and the stick figures represent the actors (even if the actor is an external system). This simple notation is often

complemented by a textual description which provide more detail.

The use cases in this section define interactions between an actor and the application. Actors in these use cases are the user of the system, the device on which the application run, and Digitalt fortalt . The use cases consist of a textual description accompanied by a visual representation in the form of use case diagrams. The textual description follows a template which consists of these items:

- ID: A unique identification for the use case.
- Name: A short text describing the goal of the use case.
- Brief description: This is a more elaborate explanation of the use case than the above.
- Actors: These are the users/systems outside the application interacting with it.
- Priority: A metric describing the priority of this use case. This metric is derived from the functional requirements document, and uses a high-medium-low scale.
- Preconditions: Describe what state the system should be in before the use case can start. Typically, some of the other use cases are already performed to set up the use case.
- Basic flow: This describe the normal flow from preconditions to postconditions in a numbered list.
- Alternate flow: A description of scenarios that differ from the basic flow described above. This includes exceptions and errors. It is also presented as a list, but the numbering in this list refers to the items in the basic flow list. The items in this list do not relate to each other.
- Postconditions: Describe what state the system should be in after the use case is performed.

TODO sett inn use cases U1 - U10

11.2 Non-functional requirements

A general requirement for the project was to use english as language in all parts related to the documentation of the application, while the language in the application would be norwegian. Other general requirements concerned the platforms the application should run on. It was decided that it should run on both Android and iOS, and that the design of the application should approach a native feel as much as possible on these platforms.

To make better decisions at a top-level design perspective, and to make better decisions on a component and implementation level, the team wanted the customer to rank each of the quality attributes below. The basis for the list and its descriptions are sourced in Software Architecture in Practice (REF). The ranked list was helpful in choosing the solutions that were most inline with

the customers needs. Normally there would be a system for measuring and quantifying each of the attributes, but this would broaden the scope and workload of the project and instead this was used as a prioritized list assisting the development, since this was believed by the team to be adequate in terms of the scope and length of the project.

There are many more quality attributes but this list describes the most generic ones, plus some specific attributes that the team believed might suit the project. Also, keep in mind that there are a lot of crossover attributes(e.g.. Maintainability/Testability may increase availability)

- Availability: How important is the reliability of the product. The easy representation to think of this is uptime of the service.
- Interoperability: How important is the ability for the system to work together with other systems. (e.g. making use of specific communication protocols or the use of a specified data format)
- Modifiability: How important and how easy should the product be able to be changed after it is finalized? (e.g. making changes to the UI)
- Performance: How important is time and speed of the system? (e.g. response time for retrieving stories)
- Security: How important is the systems ability to protect data and information from unauthorized access? (e.g. losing personalized data)
- Testability: How important is the ability to set up tests for the system (e.g. setting up automated test for components and parts of the system)
- Usability: How easy is it for the user to accomplish a desired task, and what kind of user support should the system focus on. (e.g. tutorials or hints)
- Monitorability: How important is the ability to monitor how the system while its executing. (e.g. statistics etc.)
- Other: These have been considered not very relevant for the project due to various reasons; variability, portability, development distributability, scalability, deployability, mobility, safety, marketability

(TODO Supplement: diagram non-functional requirement)

Chapter 12

Design and architecture

To get a brief overview of the complete product and its required parts the design and architecture is presented in this chapter. It is not meant to give a complete understanding, but rather an overview on how the different parts of the product work together to give a good user experience.

12.1 Architecture

The overall architectural design of the system was made to achieve a rough mapping of what needed to be done in terms of actual programing. The architecture focuses heavily on interactions between the different instances in the systems without going into the specific details on how this is done. To illustrate the architecture two different views were created. One showing only the components, with another showing the processes in the different components. The overall system structure can be seen in figure 6.XX, this shows the four different components and how they interact. In figure 6.XX a diagram showing the architecture with processes can be found. As seen in the legend, the square boxes represents individual components or processes. The boxes with oval corners represents compound processes or larger parts. These are mainly shown because they give an overview over which processes belong to and will be performed by which part of the system. Double lined boxes are external sources that provide an API. Lastly, arrows indicate information flow.

TODO sett inn Figure 6.xx Diagram of the overall system structure for this project.

It is a difficult task to model a whole system in an accurate way, and while the architecture diagram shows an overlook, it can not give much insight into the complexity of each process. This is further complicated by the fact that in the startup phase of the project there are a great number of unknowns. Both complexity and requirements are subject to change. As such, the diagram should only be used as a guide for understanding the composition of the complete system.

TODO sett inn Figure 6.xx Diagram of the architecture for this project.

12.2 Database design

The database was designed with the goal of facilitating the recommendation of stories to users. The data model underpinning the database is visualized in the ER-diagram in figure 6.XX. User and story are the central entities in this diagram, since the goal of the application is to connect users to stories. Both of these entities has a number of attributes describing the entity. The central relationship in the diagram is the recommendation-relationship, where a user and a story is connected. This connection is described by some additional attributes, such as rating, tag and state.

To make the right connections between user and story, some attributes describing the user and the stories are necessary to store in the database. For instance, in order to make recommendations based on nine predefined categories, every story is mapped from subcategories gathered from Digitalt fortalt to one or more of the nine categories (see chapter XX.XX - design and architecture, category mapping). A user is connected to one or more of these categories by the setting of personal preferences. In addition, the database stores information about the changing state of recommended stories in order to make better recommendations.

Another goal of this project is to provide some research data to the customer (see requirement RXX in appendix XX). To do this, some data about the use of the application is stored. This include registering what actions are taken during a user session. The state entity records changes in the state of stories connected to a user. For research purposes the customer also wished to know the relationship between satisfaction of a story (i.e the rating) and the media format contained in the story.
TODO sett inn Figure 6.XX: ER-diagram showing the data model.

In mapping the ER-diagram to a relational database the algorithm in [AS2, p.270-278] has been used. To ensure a good database design, four informal guidelines in [AS2, p. 487-497] have also been used. These guidelines are:

1. Making sure that the semantics of the attributes is clear in the schema
   This means that a relation schema (table) in the database corresponds to a entity type or a relationship type. The relational database used in this project follows this guideline by making a table for each entity and each relationship in the ER-diagram. The n-ary relationship user recommend story is split into tables according to the mapping algorithm.

2. Reducing the redundant information in tuples
   This means that the tables should contain no insertion, deletion or modification anomalies. Our database is designed to avoid redundant information, so that modification of one attribute only have to be performed in one tuple in one relation schema. This results in quite a few tables, but maintains a consistent database.

3. Reducing the NULL values in tuples
   Some of the tables in our database has the potential for NULL values. For instance, a user may access the application without entering an email address, which means that the user table could be filled with NULL values in the mail-column. An alternative solution would be to

to create a user-mail table, which contain only a userId and the email address for those users who actually have entered a email address. The story table also has the potential for NULL values, for instance if a story lacks a connection to an institution.

Some of the problems associated with NULL values such as the use of aggregate operations and the multiple possible interpretations of what the NULL represents does not apply to our tables. Alternative solutions has not been employed because guideline 1 has taken priority. Story and user are the most important entities in our data model and should not be divided among too many tables since that makes retrieving complete information about them harder.

4. Disallowing the possibility of generating spurious tuples
   Means that tables should be joined on attributes that are appropriately related, that is primary keys and foreign keys. This guideline is followed in the database design in this project.

12.3 User interface

The user interface design was an essential part of the project, as the customer prioritized usability over all other nonfunctional requirements. The design thus went through many iterations by working on a prototype and continually getting feedback from customer and user tests. This feedback loop was important as the customer did not have clear requirements to the application from the beginning, and making everything easy to understand for the user was also challenging. Balsamiq was first used to create a basic wireframe, but as its functionality was limited the next iteration of the design was made using Proto.io. This made it possible to receive better feedback on the flow of the app and not just the views individually.

The following diagram explains the overall flow between all the different views in the application. The blue boxes represent views, while the white ones represent modals placed on top of the view the user came from. The text of the arrows explains what the user clicked in the view the arrow comes from, and the arrow points to which view this action leads to. The functionality of the more complex views will then be explained in further detail.

TODO sett inn Figure 6.xx Diagram of the flow between views in the user interface.

Recommendation view

This view displays the stories that should be the most relevant to the user. The user can browse them by swiping through them or by clicking the left and right arrows. When tapping on a card the detailed view of the story will be displayed. The X in the right corner of each card is used to reject the story, which takes it out of the list of the current recommendations and makes that kind of stories less likely to appear in the future. The bookmark icon in the top bar opens a modal for adding a bookmark.

TODO sett inn Figure 6.xx Recommendation view

Story view

This view displays the chosen story in detail. There is a box which displays the media files associ-

ated with the story. The tabs above it will depend on which media types the story contains. Videos will be displayed by default if there are any, as they can be a major component of the story which should not be hidden in another tab. When tapping on a video or image it will be displayed in fullscreen mode. The user can give feedback on the story either by tapping the stars on the bottom part of the view, or by tapping the star icon in the top bar which will open up a modal. The modal will ask the user to rate the story, and the user can exit it by tapping Ferdig or by tapping outside of the modal. Tapping the bookmark icon will make the same modal as in the recommendation view appear.

TODO sett inn Figure 6.xx Story view

TODO: Screenshots and explanations of other views

12.4 Category mapping

There are 31 subcategories presented at Digitalt fortalt. Each story can have 0 to 31 subcategories attached to it. To achieve a content based filtering the user has to select some interest categories. To make it easier for the user, these subcategories are divided into nine main interests categories. Some of these categories of interest were already predefined in [EHW1], while the rest was changed according to discussions with the customer. The subcategories were put into the category interest field that fit them best, with some subcategories being in several category interests. In table 6.XX this mapping is illustrated. It is obvious that some category interests will have more stories, however, subcategories such as literature contain more stories than most others. In such a way, it was intended to create nine category interests that contain roughly the same amount of stories. Even distribution of stories into category interests was intended, however, when a subset of stories was chosen this intention did not hold true as can be seen in chapter XX. Furthermore, the category interests needed to be distinct while still encompassing all the subcategories.

TODO sett inn Table 6.xx Category mapping

12.5 Reuse of code

Reusing written code can be a great help to quickly have progress when programming. To investigate possibilities of this, the code from stedr was reviewed. However, since stedr used a different framework on the front end and java as the server language reusing code proved difficult. Therefore, the code written was developed without relying on previous work.

12.5.1 Digitalt museums application programming interface

All content related to stories displayed in the application was collected from Digitalt fortalt. Initially, to achieve better results when testing the personalization algorithms, the stories collected are limited to the areas Nord-Trndelag and Sr-Trndelag. The API [HM1] used to retrieve stories belongs to Digitalt museum. This API enables search through data from Digitalt museum, displaying

pictures and provides access to an XML representation of available objects.

Digitalt fortalt is established on the same technical platform as Digitalt museum [HM2]. This makes the integration better between the two and the remaining services in Norvegiana. Norvegiana is a datamodel, database and a web service with the purpose of making cultural heritage information more accessible [HM3]. Example services available in Norvegiana are Digitalt museum, Digitalt fortalt, Arkivportalen and Musikkarkiv.

Chapter 13

Tools

This section briefly describes all the tools used for this project, which includes development tools, communication tools and any additional tools.

13.1 Development tools

Front end

- Ionic [es1] - Ionic is a front end UI framework designed to assist the development of hybrid mobile applications. By using this framework it became easy for the team to speed up the design of the interface and test the application both on computers and devices. More details about Ionic can be found in section X.XX

- PhoneGap (Apache Cordova) [RA2] - PhoneGap is a framework that enables software developers to automatically wrap HTML5, CSS and Javascript code into platform-specific code that can run on devices such as iOS and Android phones and tablets. The Ionic framework is based on using PhoneGap for compiling its code. More details about PhoneGap can be found in section X.XX

- Android Studio [es22] - Android Studio is the official IDE for Android application development, and it was necessary to have this installed in order to develop our application on android devices. It also provided a way to install various plugins that were useful or needed for the project.

- Node.js [es23] - Node.js is a platform built on Chromes JavaScript runtime for easily building fast, scalable network applications. This was another tool that was necessary to install, because PhoneGap is built on it. First released in May of 2009, Node.js has been gaining much popularity as a server-side platform.

- Gulp.js [es24] - Gulp is a build tool that is used as a part of PhoneGap in order to automate many common tasks, such as build processes and plugin handling.

- Sass [es25] - Sass is an extension to CSS which adds functionality such as being able to use variables, nested rules and inline imports. Sass helps keep stylesheets organized, and is fully compatible with regular CSS syntax. Sass is the preferred tool for handling stylesheets in Ionic.
- Proto.io [es2] - This a prototyping framework aimed for mobile apps, and it allowed the team to make very quick and functional prototypes that were used for user testing, both by the team and by the customer. When discussing design solutions, it was much faster and simpler to make revisions to the prototype than it would be to redesign the application itself.
- Balsamiq [es3] - Wireframing and mock-up tool which was used to create early mock-ups of the different user interface views. This was used because it allowed the team to quickly make wireframes for the interfaces, and it gives a more professional look than by just making paper prototypes.
- Icomoon [es4] - This is an application with the purpose of generating an icon font from svg files that you upload to it. You can also resize, adjust positions, and set default pixel sizes for the icons. The application turns all the icons into crisp-looking and easy scalable icons. It also automatically generates the HTML and CSS code which you can use to integrate the code into your own application. In this project, Icomoon was used to make the category icons that show the various categories on each story
- FontForge[es5] - After creating an icon font with Icomoon, FontForge was used to make manual adjustments to the icons themselves. FontForge is an editor with many advanced options for giving icons a smoother look and symmetry.

Back end

- Docker [EHW2] - Docker is an open platform that provides the possibility for system developers to build their application and ship it to another computer or server, which can then run the same application, unchanged. The motivation for using Docker was mainly so the team could run the developed application on SINTEFs own server. More details about Docker is described in section X.XX
- MySQL [es8] - MySQL is one of the most widely used open source databases. It has many advantages when it comes to scalability and flexibility, and it is well suited for many types of application development. More details about this projects database and its design can be found in section X.XX
- Digitalt musem API [es9] - The source of the stories in the application was Digitalt Fortalt, and in order to access these it was necessary to use Digitalt museums API. The API is documented on its website and was incorporated into the projects database. Details about the API can be found in section X.XX
- phpunit[KF2] - Phpunit is a framework for writing and performing unit tests on php code. More details about how phpunit was used in this project can be found in [X.X Test Plan - Unit test]

- Karma[KF3] - Karma is a test runner for Angularjs. Karma has it . With Karma it is possible to write javascript tests and run it in different browsers on both desktops and mobile phones. It is easy to run a test for every integration you make. More details about how Karma was used in this project can be found in [X.X Test Plan - Unit test]

13.2 Communication tools

- Google drive [es10] - Used for creating and sharing documents with the whole group, as well as editing shared documents in real-time. it also made it possible to share all files, whether it was images/diagrams/spreadsheets, or anything else. That was why the team decided to use this as a good solution for sharing all documentation.

- Dropbox [es11] - Used to share documents between the group and the customer. This tool was used upon request by the customer.

- Facebook [es13] - Used for discussions and notifications of various things such as meeting times. This was used because Facebook is something everyone was already familiar with, and something that most of the team members use on a regular basis, so messages would be quickly noticed.

- Trello [es15] - Task management application, this was a handy way to quickly see what needs to be done and what has already been completed, similar to a scrum task board . Because of this, the team found it to be a good tool to use, as it speeds up the process of managing work tasks and gave a better overview of the progress.

13.3 Additional tools

- Github [es12] - Used for making a code repository to be shared by the group while developing the system. The customer requested the use of Github, and it was also used because it seemed like the easiest way to share and implement code, as well as sharing the code with the customer.

- Draw.io [es14] - This tool was the primary way of making diagrams and models, such as use-cases and WBS chart. This tool was chosen for this because it provides a lot of templates for different types of graphs, and as everyone uses the same tool for all diagrams the report achieves a consistent style for every diagram.

- Ganttify [rg1] - Converting Trello boards into Gantt Charts, makes the process of creating a gantt chart and milstone plan easier and faster.

Chapter 14

Implementation

This chapter discusses the details of the system implementation process, both for front end and for back end.

14.1 Front end

14.1.1 User interface

Early on in development, the team discovered several limitations of the Ionic framework. For example when using a list to display stories, it was not possible to swipe the list both left and right. The idea was to swipe one way to add a story to be read later, and swipe the other way to reject a story from the list entirely. Because this proved to be impossible, the views were redesigned into a different solution which was much less based around swiping.

As this is an application for mobile devices, it had to be adapted to work on different screen sizes. The team found that it would most likely be best to target a relatively small screen size and then simply enlarge it for bigger screens. This eliminated the issue of having to compress the components to fit smaller screen sizes and potentially be forced to redesign the whole view to fit small screens.

The applications uses many different icons in various parts of the interface, and these have been the source of much debate and redesign. The icons used to represent categories were not always understood by users, and some categories like local tradition and food were difficult to represent universally with just a single icon. Also the bookmark icon shown in the upper right area of figure 8.xx was confusing to many users, and there was a concern in the team that this icon might not accurately represent that it allows the user to save the story in a collection.

Adopting accurate naming conventions for the different components has also been a considerable issue. Stories can be saved in collections, but these collections have interchangeably been called lists, tags, and bookmarks in the system. Also when asking a user to input their preferred

categories to receive stories from, there has been some confusion because of interchangeably calling these categories for interests, preferences, and categories.

Implementing media, and especially video, has been a challenge in the project. An issue with this has been that playing videos is handled differently on iOS and Android, which had resulted in some bugs that only appear on one platform and not the other. These types of issues have been problematic to fix and has taken up much time to fix. In addition, the videos provided by the Digitalt fortalt website come from different sources. Some of them are Youtube videos, others are Vimeo videos, and there are also other variations. Integrating all these different formats smoothly into the application has been a challenge in itself.

14.1.2 Prototype

The prototype has been through multiple iterations. Early on, it was imagined to have a sort of magic discovery function where a user would for example rub a crystal ball and receive a recommended story. This idea was later discarded because the team decided it would be better usability to present the user with multiple recommended story that they could simply browse through instead.

Another of the early ideas was for the user to receive a daily story or some sort of schedule for being presented with recommended stories. However, due to workload and time constraints, this requirement was heavily down prioritized. The most important parts were the personalization and usability aspects, so receiving notifications seemed like an unnecessary extra feature.

A big issue for the interface design has been the handling of the different media elements (text, pictures, audio, video) and how these should be positioned relative to each other. For a while the team designed the application to have one tab for each of these four elements in the story view, as shown to the left in figure 8.XX. The customer had a concern that this might not be the optimal solution, as a user would for example not be able to read text and view pictures simultaneously. After some discussion, the interface was redesigned so that the text would be persistent, and instead the user could tab between pictures, audio, and video. The resulting design can be seen to the right in figure 8.XX.

TODO sett inn Figure 8.XX: Comparison of the story view in the first and second version of the prototype.

14.2 Back end

This section describe the development of each part of the back end. It aims to give a timeline of the development and explain how and why important decisions were made. The different parts described here are the database, personalization and the communication between front end and back end.

14.2.1 Database

Based on the first version of the functional requirements for the application an initial ER-diagram was made in the middle of february. At this stage the customer and the team had not come to an agreement on a prioritization of the non-functional requirements. This meant that there for instance was not clear how important the performance of the system would be for the customer, an attribute of the system which would influence how much info should be stored for each story in the database versus have much info should be retrieved from Digitalt fortalt every time a user views a story. However, the changes to the initial diagram have been relatively minor. Some of the alterations were based on updated requirements from the customer, while others stem from the group and are optimization of the data model or changes made to facilitate the personalization.

Shortly after the first ER-diagram was made, an SQL-script for creating a database was also made. Since the ER-diagram has undergone changes for quite some time after the first version was made, these changes has also influenced the SQL-script. This means that a change in the data model have lead to more work than just updating the ER-diagram. However, the progress of the project relied on testing against an operational database, for instance regarding the harvesting of stories from Digitalt fortalt.

The customer had early on said that the retrieval of research data about the use of the application was important for them. Initially, the details of this was not properly formulated and the initial data model therefore did not reflect this. After receiving a detailed list from the customer describing the research data to be gathered, alterations - mostly additions - in the data model were made to accommodate this. This concerned storing timestamps for user actions and states of stories.

The team did not decide or understand how to do the personalization until mid March. This decision introduced changes to existing tables and the need to create additional tables and views in the database. Mahout had requirements for the input data, which meant that a view was created to store all the necessary data for collaborative filtering. Using this view made it straightforward to put the desired data from the database into Mahouts data model.

14.2.2 Personalization

The way personalization was implemented was by having the user give their preferences about which types of stories they enjoy, and also by harvesting circumstantial information about the user in order to provide relevant stories. Users of this system are also allowed to give feedback on the recommendation of stories from the database to achieve further personalization. Users receive story recommendations based on other users preferences and recommendations, i.e. collaborative filtering. The system was developed as a client/server architecture, while the database containing the stories was provided by Digitalt fortalt [es19]. The stories are found in a variety of formats such as text, images, audio and video.

14.2.3 Front end - back end communication

Communication between front end and back end was handled using http post requests. AngularJs $http is a core service for reading data from remote servers, which is called every time the application needs to add, retrieve, update or delete data. When an http request is made four fields are set: method, headers, url and data. The method field determines the http request method, which in this application is set to post, and the headers field sets the content type to JSON. The url is the location of the remote server script that handles http requests. In the data field the action to be executed is specified, in addition to any data needed to perform the desired action.

Each http request is managed by the same back end php script. This script decodes the http request, determines which action to perform and executes it. When the script has finished executing, a json response is returned to front end with the desired data.

Chapter 15

Testing

The following subsections will describe the strategies for the testing levels usability test, unit test, integration test and system test.

15.1 Unit testing

The purpose of unit testing is to ensure that every component that are implemented in the system are functional and correct. The group performed unit tests on new components or units of code that was implemented in the system. It was necessary to prioritize what components that should be tested, considering the amount of time given for the project. Testing of the user interface is required to ensure that the user get the right user experience. The recommendation module is a module that produce a result that is vital for the quality for the app. Therefore the unit testing in back end will focus on the recommendation module. Most of the code in back end is communicating with database, front end and Digitalt fortalt. Because of this we will focus on testing this code in the integration testing(X.X).

Roles & Responsibilities
To get a structured testing experience, the team had to delegate responsibility for the units. The roles correspond with the roles that were given at the start of this project (Table 3.xx Role delegation), so the tester would have good knowledge to the code and know how it works.

TODO sett inn Table 9.x shows the delegated responsibilities for testing the user interface.
TODO sett inn Table Table 9.x shows the delegated responsibilities in testing the back end part of the system.

Required before performing the tests
The developers of the user interface needed to install the Angularjs Karma, for performing the unit testing. With this tool they could write scripts for each of the test cases.

The developers of the back end of the system needed to install phpunit, a framework for unit testing in php.

After the tests
The tester investigated the results of the test cases and mended the issues if they appeared during the tests. When the issues in the code were fixed, the tester would run the test again and see if it approved. This was cycle repeated until the code was free of issues and runned the expected way.

Test Cases
The testers created test cases and used these as a guide for performing the tests. The test cases has an ID and describes exactly what the test should do, what input data to use and what is expected to happen when the test is running.

TODO sett inn Table 9.X present the test cases used for the unit testing.

15.2 Integration test

Integration testing is usually executed after unit testing and before system testing. This test was performed after every individual component was tested and integrated with the rest of the system. This is to ensure that the different modules of the system were communicating correctly and that data was moving properly from one module to another.

How to perform the integration test
The testing was performed at modules as soon as they were developed and completed, a so-called incremental testing. This approach made it easier to detect issues in the code early and therefore avoid issues to come up at a later stage in the development process - which could be cost-ineffective. Because of the time limitations and the difficulty with learning a new interface to perform integration testing, the developers decided to perform the integration testing with unit test cases. The unit testing framework was already known to the developers and therefore easier and less time consuming to use. It was possible to run unit tests across the modules and get the wanted results.

Test cases
The test cases were made by first having a closer look at the different modules and the data flows between them. The modules in question are shown in the figure 5.xx of the architecture. for this project. The modules that the integration testing was performed on were front end(User Interface), back end with a general processing module and a personalization module, and the database. Because of the personalization module of our system is considered to be a crucial one, the most important data flows was the users input in the form of preferences and ratings. Also the communication with the database was crucial because the users information about ratings and preferences should be stored properly to get a beneficial recommendation. The test cases are described in the following table 9.x.

TODO sett inn Table 9.x

15.3 System testing

The system testing is executed after the unit test and integration tests. It was to give developers a measure of whether the system met all the goals set for the project. The system test included performing a black box testing of the system, where the test cases was based on the use cases(chapter x.x) and the specified requirements(chapter x.x) defined earlier in this report.

Required before performing the system tests
The unit tests and integration test were required to be executed and approved before the system test could be executed. To execute the tester also needed the final product in the way it was intended to the end users. Therefore the application in question should be accessible on a mobile device before performing the test.

The main areas to be tested
The test cases was to cover all the use cases and requirements specified for this project. The main areas are as follows:

- Create user
- Log in
- Set initial settings
- Browse recommended stories
- Add stories to list
- View story
- Rate story
- View list

How are the test to be performed?
In this test one of the developers were executing the test. Because it is a black box test, the tester would do the test cases with no access to the code. the tester went through all of the test cases one by one and performed the test cases manually. Due to the time limits of this project the team were not able to write scripts to perform the test cases.

After the test
After the system test suit is performed and the results are documented, the testers should mend the issues in the system if they appeared during the test. After the mending the test suit was executed again. This cycle was repeated until there was no issues left in the system and all test cases got the expected result.

When the system test is performed and finished, the testers evaluated the tests results and then decided if the system as a whole fulfilled all goals for the project.(See chapter X.X). The test results should, if done in the expected manner, help the developers of this project to verify and validate if the application meets all the requirements.

Test Cases
Each test case has a test identifier and an approach for the tester, and a description of what was intended to happen when the test case was performed. The tester will be referred to as the user. Some of the test cases have a dependability of other tests. If an issue is detected in one test case, it might cause issues in its dependent test cases. table 9.3.1 and 9.3.2 are presenting two of the test cases that were used. The whole test case document are in Appendix C in tables C1.-C.10.

TODO sett inn Table 9.3.1
TODO sett inn Table 9.3.2

15.4 Customer acceptance test

Customer acceptance test (CAT) will be executed during the whole software development life cycle. After a sprint, the customer will test the product, evaluate and bring feedback. In the early stages of the project process this contains testing of the prototypes. When working software is delivered to the customer after a sprint, the customer use their own real input data to test the behaviour of the system. This kind of testing might reveal a different result than from a regular unit or system testing, when the data could be more realistic when the customer defines it. The customer brings feedback either in meetings or through email. The planned delivery dates are presented in 3.3 Project milestone plan.

15.5 Usability testing

The user testing was performed by the front end developers. The preliminary work for the user test also included doing an analysis of the requirements, and used these as a base for making a list of scenarios. The scenarios set the grounds for the test cases. IEEE standard 610 defines a test case as; A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. [KF1] The test case included several test steps that the user followed, and the testers observed how the user reacted in every test case. After each test finished there was a discussion with follow-up questions the user had to answer to get a better insight into what was problematic and what was easily understandable. For further information on this see the framework for the usability testing in Appendix XX(Usability test template).

15.6 Results

This section includes the results from the tests described in sections X.1 through X.4.

15.6.1 Unit test results

Table X.X shows the results we gained from performing the unit test cases. Here are the test cases referred to with their Test case ID. Test Case description and input data are described in the test plan(Chapter X.X).

Detected issues in the system
[TODO: Short description of the main issues found in the system, and how they were handled]
Summary
[TODO: Short description of how the test was, did it give us help to improvement of the code]

15.6.2 Integration test

Detected Issues
Summary

15.6.3 System test

[TODO: results]
Detected Issues
Summary

15.6.4 Usability testing

[TODO: results]
Summary

15.6.5 Customer acceptance test

As described in section 9.4 Customer Acceptance test, this test was performed after every product delivery. The following tables display the customer feedback of every delivery. The tables describes general comments and issues the customer detected and wants to improve.

TODO sett inn Table 9.X shows the date and feedback for the first paper prototype. The feedback information was given to the developers in a customer meeting, and this is a summary from that meeting.
TODO sett inn Table 9.X shows the date and feedback for the first paper prototype. The feedback information was given to the developers in an email sent from the customer. The text is a summary of the email.
TODO sett inn Table 9.X shows the date and feedback for the first paper prototype. The feedback information was given to the developers in an email sent from the customer. The text is a summary of the email.

TODO sett inn Table 9.X .shows the date and feedback for the second working software. The feedback information was given to the developers in a customer meeting.
TODO sett inn Table 9.X.

Chapter 16

Evaluation

16.1 Product quality

16.2 Development process

16.3 Project management

16.4 Team

16.5 Customer interaction

Chapter 17

Conclusion and future outlook

Chapter 18

Bibliography

[es20] http://www.ntnu.edu/studies/courses/IT2901#tab=omEmnet (30.01.2014)
[es21] http://www.tagcloudproject.eu/
[es19] http://digitaltfortalt.no/
[RA1] http://ionicframework.com/blog/where-does-the-ionic-framework-fit-in/
[RA2] http://phonegap.com/about/faq/
[RA3] http://forumone.com/insights/what-titanium-appcelerator-really-and-how-it-works/
[RA4] http://www.sencha.com/products/touch/features/
[AS3] http://www.nngroup.com/articles/ten-usability-heuristics/ (11.03.2015)
[AS4] http://www.tagcloudproject.eu/?p=1390
[AS5] http://www.tate.org.uk/context-comment/apps/magic-tate-ball
[es16] [es17] [es18] Sommerville, Software Engineering, 9th ed.
[AS1] Sommerville, Software Engineering, 9th ed. 106-107, 124-125.
[AS2] Elmasri, Ramez & Shamkant B. Navathe, Database Systems: Models, languages, design, and application programming 6th ed..270-278, 487-497
[es1] http://ionicframework.com/
[es2] https://proto.io/
[es3] https://balsamiq.com/
[es4] https://icomoon.io/
[es5] http://fontforge.sourceforge.net/
[es8] http://www.mysql.com/
[es9] http://api.digitaltmuseum.no/
[es10] https://www.google.com/drive/
[es11] https://www.dropbox.com/
[es12] https://github.com/
[es13] https://www.facebook.com/
[es14] https://www.draw.io/
[es15] https://trello.com/
[es22] http://developer.android.com/tools/studio/index.html
[es23] https://nodejs.org/
[es24] https://github.com/gulpjs/gulp/tree/master/docs

[es25] http://sass-lang.com/
[AS6] Sommerville, Software Engineering, 9th ed., 652-654
[EHW1] TAG CLOUD dokument
[EHW2] https://www.docker.com/
[HM1] http://api.digitaltmuseum.no
[HM2] http://digitaltfortalt.no/info/about
[HM3] http://data.norge.no/data/norsk-kulturr%C3%A5d/norvegiana-api
[HM4] http://www.inf.unibz.it/ ricci/papers/intro-rec-sys-handbook.pdf
[HM5] http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf
[HM6] http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf
[HM7] http://www.apache.org/licenses/LICENSE-2.0
RG1 http://www.gantt-chart.com/
[KF1] 2003. What is a good test case? : Cem Kaner, J.D., Ph.D, Florida Institute of Technology. Department of Computer Sciences
[KF2] phpunit framweork : https://phpunit.de/ (18.03.2015)
[KF3] http://karma-runner.github.io/0.12/index.html (18.03.2015)
[KF4] https://proto.io/ (18.03.2015)

Bibliography

Brouwer, D. R., Jansen, J. D., 2004. Dynamic optimization of waterflooding with smart wells using optimal control theory. SPE Journal 9 (4), 391–402.

Muskat, M., 1937. Flow of Homogeneous Fluids. McGraw Hill.

Sarma, P., Chen, W. H., 2008. Applications of optimal control theory for efficient production optimization of realistic reservoirs. In: Proceedings of the International Petroleum Technology Conference. Kuala Lumpur, Malaysia.

Appendix

Appendix A: Functional requirements
TODO sett inn Appendix A: Functional requirements

Appendix B: Status report example
B.1 Status report example
Status report week 6

1 Introduction
This week has mostly been spent organising and making decisions that will impact the whole project.

2 Progress summary
The decisions that have been made will decide how the work is distributed in the coming weeks. Work has been made on defining goals and milestones. Furthermore, some of the tools to complete the given tasks have been found.

3 Open / closed problems
Closed problems:

- A cross-platform framework have been chosen.
- A rough estimate of what needs doing, how long it will take and when it is due has been performed in the form of a product backlog.
- A list of functional requirements have been compiled after a discussion with the customer. Use case diagrams and scenarios have been made.
- Justification on some of the choices made so far have been written for the report:
  - SCRUM
  - Framework
- Complete a WBS chart.
- A rules of engagement have been signed, this helps solidify what is expected of every member in the group.

Open problems:
No specific ongoing problem at the end of this week.

Choosing a cross-platform framework was a difficult process for various reasons. There is not much experience in the group using such tools. Additionally there was an internal debate about what is expected from the customer and what does the team expect the end product to look like. This was discussed in light of the the constraints imposed from various aspects. However, the team members now feel confident that an appropriate tool have been chosen and are aware of some limitations this leads to.

Understanding properly what the customer wants and prioritizes has also been a focus this week. While this sounds easy enough, the technical details are often lost in communication. This is something that will require constant feedback and monitoring so that the project stays on track in regards to what is desired by the customer.

4 Planned work for next period

- Familiarization with the chosen framework.
- Familiarization with digitaltfortalt.no API.
- Creating a design prototype is a goal, this will unify the group and make sure all members are working towards the same goal. Furthermore, this will explore what options there are and highlight any basic flaws in design.

Appendix C: System test cases
TODO sett inn Appendix C alle tabeller

Appendix D: Test results
TODO sett inn Appendix D alle tabeller