dedication (optional)

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 problem to solve in this project was to develop an application that could give its users personalized story suggestions, based on their interests and with high concern for usability of the application. The application also had the requirement of being able to recommend stories based on what other users like, i.e. collaborative filtering. The motivation for creating such an application was to increase the interest in cultural heritage, and to accomplish this by using innovative technologies to encourage its users to find and read stories that they would find interesting.

To create this application, it was necessary to spend time studying other applications already existing that had similar functionality to what we were trying to do. Much of the focus in the early stages were on refining the requirements with the customer, and planning out how the interface would look like. Several rounds of user testing and prototyping were done in order to detect problems and optimize the interface as much as possible.

The resulting application was able to give smart recommendations of stories based on interests and collaborative factors. It was able to present the stories to its users in an esthetically pleasing manner, as evaluated by the users and customer. It also contains functionality for giving feedback on stories, as well as functionality for saving and managing user-defined collections of stories. In conclusion, the application achieved what it was meant to do, which was to serve as a tool for testing whether this kind of personalization may lead to an increased interest in cultural heritage. The scope of the application was not massive, as it was mainly designed to be used by a controlled group of users for testing purposes. The team in charge of the application and this report feel that the application meets this goal both functionally and esthetically.

Preface

Write your preface here...

Table of Contents

List of Tables

List of Figures

Abbreviations

Symbol = definition

Chapter 1

Introduction

This chapter introduces the customer, the team, and the project's definition and purpose.

1.1 Stakeholders

1.1.1 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.

1.1.2 Team

Table 1.1 lists the persons in charge of developing the project in this report, and some of their background competencies.

Table 1.1: Team description

Name	Competencies
Kjersti Fagerholt	HTML, CSS, Javascript, Java, PHP, SQL, Python.
Roar Gjøvaag	.
Ragnhild Krogh	HTML, CSS, Javascript, Java, Python, responsive web design
Espen Strømjordet	HTML, CSS, Javascript, Java, UI Design experience
Audun A Sæther	HTML, CSS, Javascript, Java, PHP, SQL
Hanne Marie Trelease	Java, PHP, Javascript, HTML, CSS, SQL
Eivind Halmøy Wolden	CSS, Javascript, Java, HTML, PHP, SQL

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.

1.2 Project description

In the course IT2901 [**?**], 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 [**?**] project, which is a project whose aim is enriching cultural and historical experiences through innovative mobile applications. In this project multiple ways of personalization had to be integrated to find good recommendations for the user.

1.3 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.

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 description of the requirements for the project will be presented.

Chapter 2

Requirements specification

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

2.1 Functional requirements

The requirements document for this project can be found in **appendix C**. 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.

2.1.1 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. As agreed with the customer, storing the email address is adequate. 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 user's 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.

List view: The application should keep lists of stories. Lists are created based on the status of the story (to-read, read) or the user's 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 site: The application should include an about site, which explains the context in which the application was created.

Quick tour: An introduction to the application should be given to new users, and also be available through the menu

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. The customer provided a list of this, and some details can be found in **Section 6.2**

2.1.2 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 [1, p.106-107]. 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). Include arrows means that the included use case is required to accomplish the use case the arrow originates from, while the extends arrow describe a use case that might be performed after the main use case. 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.

2.2 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

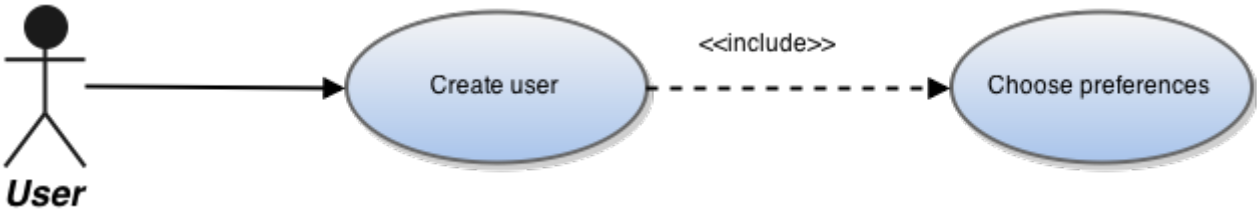


Table 2.1: U1. Create profile

ID	U1
Name	Create recoverable profile.
Brief description	Enter mail to register.
Actors	User
Priority	High
Preconditions	Application installed
Basic flow	1. User clicks on register new user 2. User fills in mail into a registration form 3. Input validated 4. User finishes the registration and the syestem saves the user ID
Alternate flow	1. User skips registration and starts using the system 2. The system stores the new user by an ID internally
Postconditions	User is created and saved by the system

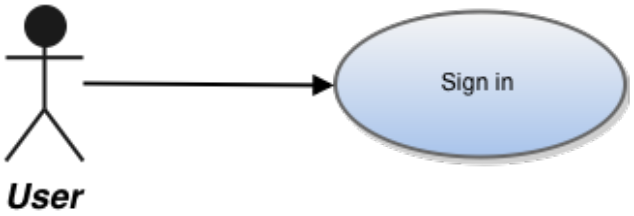


Table 2.2: U2. Sign in

ID	U2
Name	Sign in.
Brief description	Enter email address to sign in.
Actors	User
Priority	High
Preconditions	User has already registered
Basic flow	1. User clicks on sign in 2. User fills in mail into form 3. System checks email address. User id returned 4. Main view displayed
Alternate flow	1. User not found 2. Error message displayed, go to 2.
Postconditions	User is logged in to the system

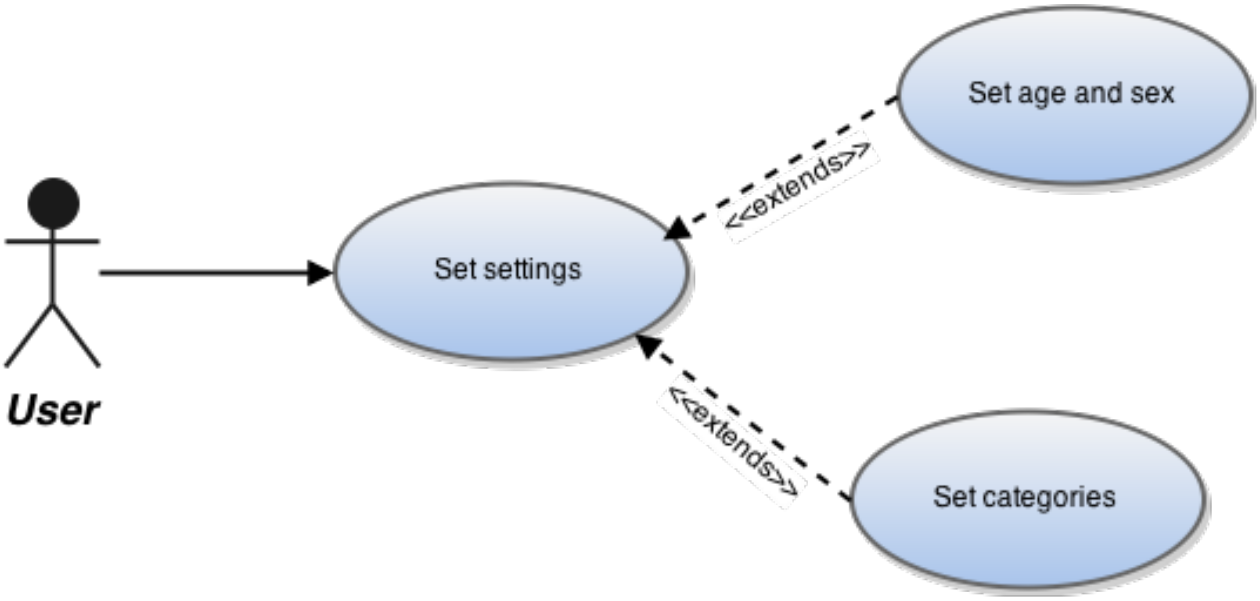


Table 2.3: U3. Set initial settings

ID	U3
Name	Set initial settings.
Brief description	Specify user information to be used for receiving stories.
Actors	User
Priority	High
Preconditions	Application installed
Basic flow	1. User is initially asked to fill a form about preferences by the system (a) Personal info: age group and sex (b) Cultural category preferences 2. The system saves information about the user's preferences
Alternate flow	1. User clicks on settings in order to change preferences
Postconditions	The system has information about the user in order to provide personalized story recommendations.

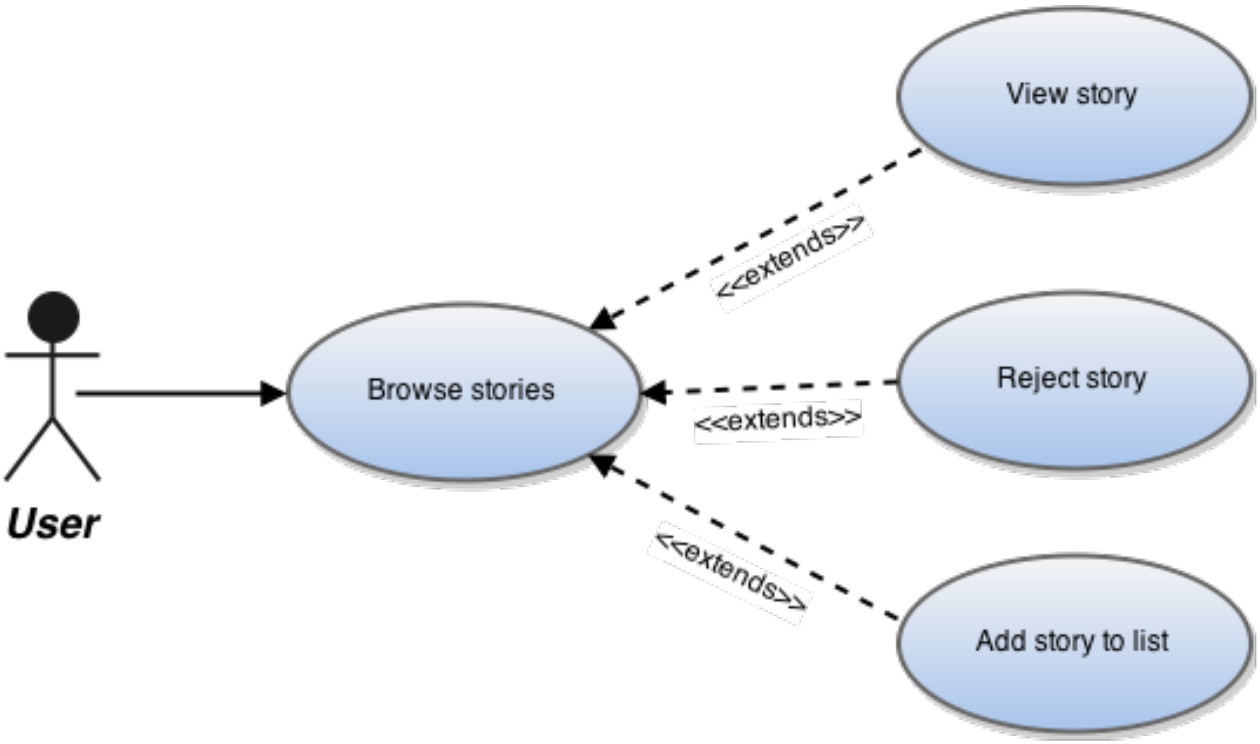


Table 2.4: U4. Browse recommended stories

ID	U4
Name	Browse recommended stories.
Brief description	User is shown a list of recommended stories to choose from.
Actors	User
Priority	High
Preconditions	Preferences already set
Basic flow	1. User is shown a list of recommended stories 2. User does one of the three following actions on a story: (a) Choose to read the story now (b) Reject the story (c) Add story to list 3. If option (b) or (c) is chosen the user is shown a recommendation list without the story
Alternate flow	1. If the user chose to read the story now, the story - and not the updated recommendation list - will be shown.
Postconditions	The system has stored information about the choice of the user.

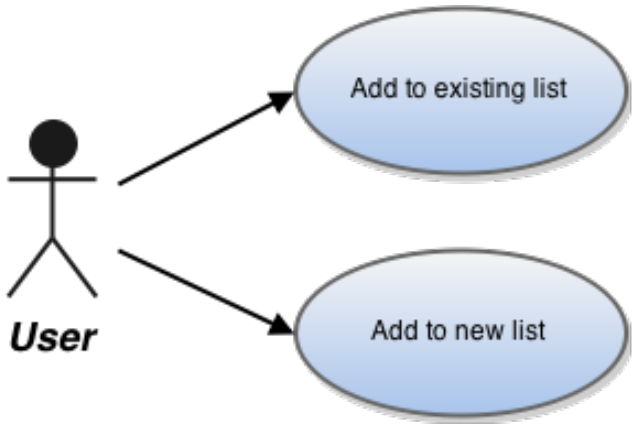


Table 2.5: U5. Add story to list

ID	U5
Name	Add story to list.
Brief description	Put a story in to-read list or user-defined list.
Actors	User
Priority	Medium
Preconditions	User is registered and signed in
Basic flow	1. User opens a story to read 2. User clicks the bookmark button in the story and selects the desired collections to put the story in. 3. Story will be put in the selected list
Alternate flow	1. The desired collection does not exist. The user clicks to add new collection.
Postconditions	Story will be added to and/or removed from various lists according to user's actions

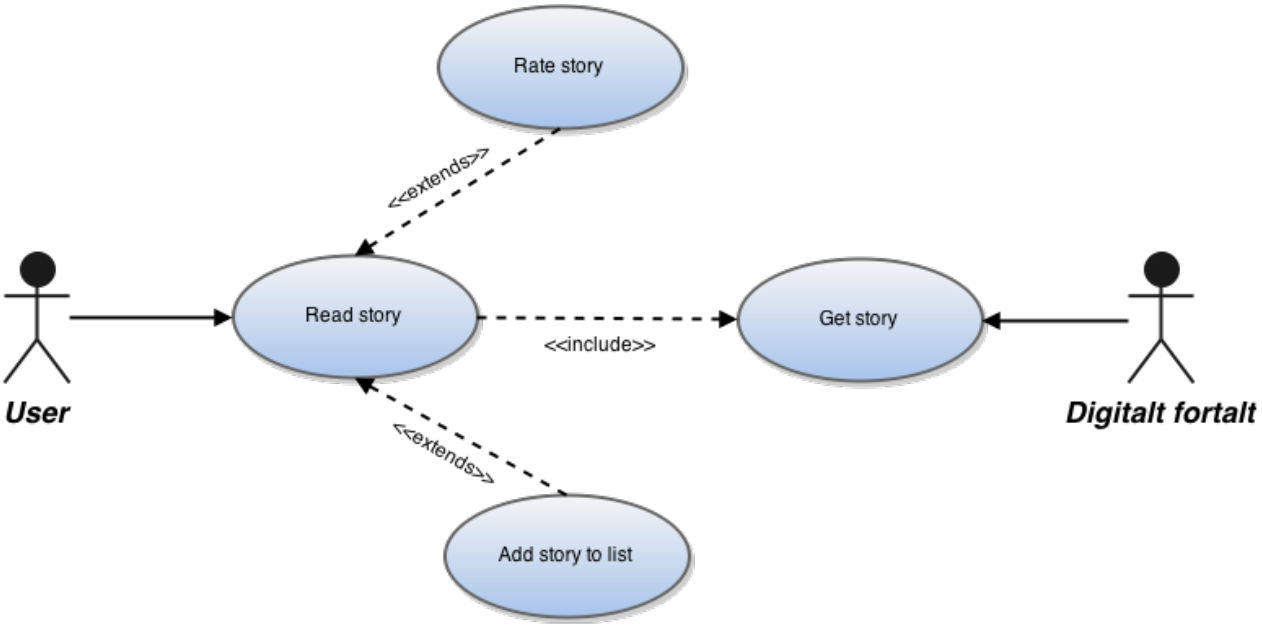


Table 2.6: U6. View story

ID	U6
Name	View story.
Brief description	Display media (text, images, video, sound clips) related to the story selected.
Actors	User, Digitalt fortalt
Priority	High
Preconditions	The story is in the list of recommended stories or favorites
Basic flow	1. User selects story 2. Display: (a) Available media files (b) Link to story on Digitalt fortalt (c) Explanation of why the story was recommended 3. User selects media type (text default)
Alternate flow	
Postconditions	The story is shown according to the user preferences

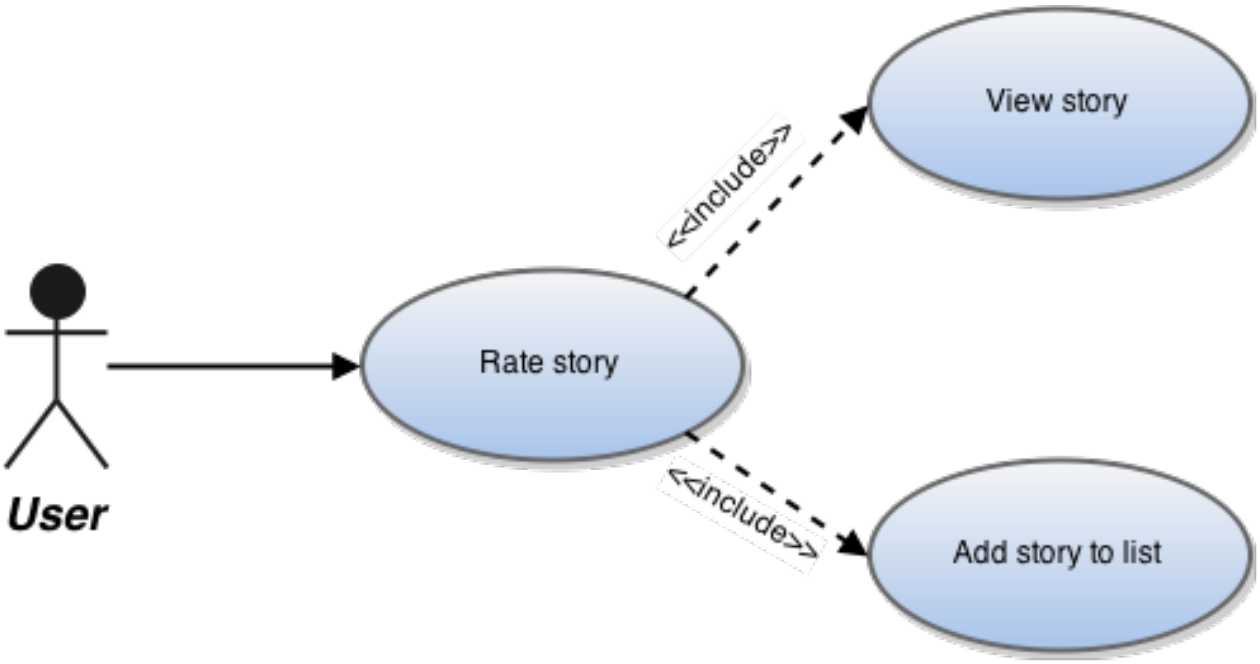


Table 2.7: U7. Rate story

ID	U7
Name	Give feedback / rating.
Brief description	Give a rating on a story.
Actors	User
Priority	High
Preconditions	User has opened a story for reading
Basic flow	1. After reading the story, the user clicks on a rating to give feedback on the story 2. The system saves the rating 3. Story will be put in read list
Alternate flow	1. User does not give a rating on the story before closing it 2. The system reminds the user to rate the story at a later time
Postconditions	The rating of the story from the user is saved

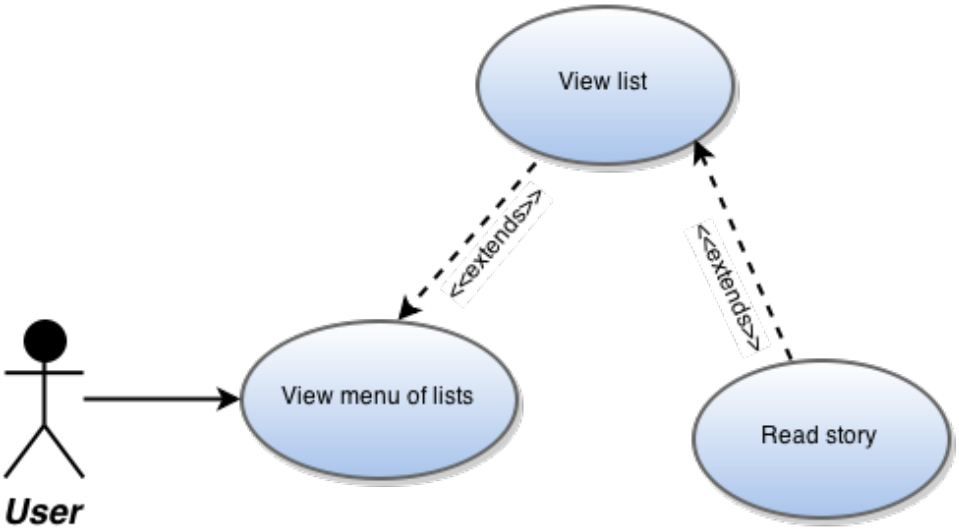


Table 2.8: U8. View list

ID	U8
Name	View list.
Brief description	View list of collected stories (favorites,to-read,tags defined by user).
Actors	User
Priority	Medium
Preconditions	User is registered and signed in
Basic flow	1. User selects list from menu 2. List of stories shown
Alternate flow	1. No stories found 2. Display message “No stories found”
Postconditions	User is shown all of the stories collected

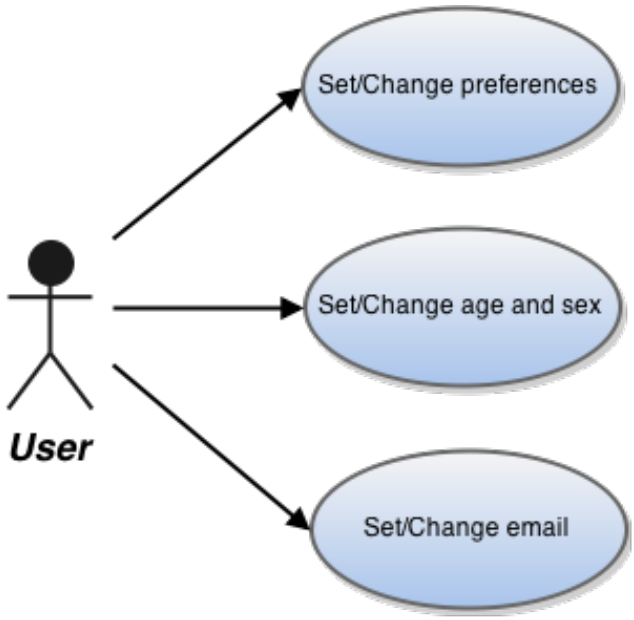


Table 2.9: U9. Specify settings

ID	U9
Name	Specify settings.
Brief description	Specify and change system settings.
Actors	User
Priority	High
Preconditions	User is signed in
Basic flow	1. While logged in, user clicks on settings 2. User edits the data in settings (a) Category preferences (b) Age group / sex (c) User e-mail 3. The system saves the new settings
Alternate flow	
Postconditions	Settings are changed and saved

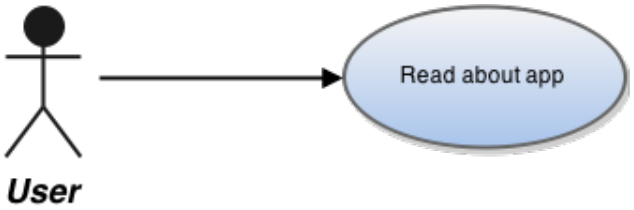


Table 2.10: U10. Read about app

ID	U10
Name	Read about app.
Brief description	Basic info about the application.
Actors	User
Priority	High
Preconditions	User is signed in
Basic flow	1. While logged in, user clicks on settings 2. User clicks on read about app and is presented with information about the app
Alternate flow	
Postconditions	

Practice (REF). The ranked list was helpful in choosing the solutions that were most inline with the customer's 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 system's 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 3

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.

3.1 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 as seen in **Appendix A**.

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 [**?**]. 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.

3.2 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 3.1 below summarizes some of the capabilities and limitations of the various frameworks that were under consideration.

3.2.1 PhoneGap

PhoneGap is an open-source mobile app framework for native packaging [**?**]. 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.

3.2.2 Ionic

Ionic is an open-source UI framework focused on making it easier to create hybrid mobile apps with a native feel [**?**]. 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

Table 3.1: Framework comparison

	PhoneGap w/ Ionic	**Appcelerator Titanium**	**PhoneGap w/ Sencha Touch**
Can write a single code that runs on both iOS and Android?	Yes	No	Yes
Access to native components	Yes	Yes	Yes
Expected ease of learning and use	Relatively easy	Difficult	Somewhat difficult
Performance of created application	Good, AngularJS also provides a performance boost	Very good, as it provides direct access to native components	Acceptable, but not as performance- focused as Ionic.
Debugging	Easy	Difficult	Easy
Programming language used	HTML5, CSS, Javascript with AngularJS	Javascript	HTML5, CSS, Javascript

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.

3.2.3 Appcelerator Titanium

Titanium is a cross-platform Javascript runtime and API framework [**?**]. 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 does not use HTML/CSS.

3.2.4 Sencha touch

Sencha touch is a mobile framework with a large number of UI components and an architecture for the front end [**?**]. Instead of enhancing an HTML file, it generates the document object model (DOM) with JavaScript. The DOM is an interface which creates a representation of the HTML code and allows the developer to manipulate the HTML elements. Sencha touch can be used with PhoneGap, but 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.

3.2.5 Conclusion

The framework combination chosen for this project was Ionic and PhoneGap, as this seemed to fit this project's 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 application and to reuse parts of it for other apps. This is fulfilled by Ionic's 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.

3.3 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 3.2** below gives a comparison of some aspects of the various processes that were considered relevant for the project.

Table 3.2: Development process comparison

	Scrum	**Extreme Programming**	**Waterfall**
Type of methodology	Agile	Agile	Plan-driven
Responds well to changes in requirements?	Yes	Yes	No
Amount of planning needed	Moderate	Very little	Very much
Level of detail for project plan	Moderate	Low	High
Customer involvement	High	High	Low
Frequency of testing	Continuously	Continuously	Only near the end
Iteration length	Short	Short	Long

3.3.1 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 [1, 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.

3.3.2 Extreme programming

Extreme programming is an agile method focused on pushing out new system versions and functionalities rapidly [1, 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.

3.3.3 Scrum

Scrum is a general agile method with focus on managing iterative development rather than specific technical engineering approaches [1, 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 do not 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.

3.3.4 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. Scrum would be helpful in this regard as it would allow the group to quickly make a simple application which could be tested by the users and 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.

3.4 Personalization

To provide story recommendations in accordance with each user's 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 user's 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.

3.4.1 Recommender systems

Recommender systems are software tools and techniques that attempts to provide recommendations of items [**?**]. 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.

3.4.2 Content-based filtering

Content-based filtering methods are used to find similarities between a user's preferences and the description of an item [**?**]. 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 user's 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 user's 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. [**?**]

3.4.3 Collaborative filtering

Collaborative filtering is based on collecting and comparing information on users' behavior, activities or preferences and to recommend items based on a user's similarity to other users [**?**]. 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.[**?**]

Table 3.3: Summary of the main findings in the evaluation of existing solutions

	stedr	**Cooltura**	**Magic Tate Ball**
Content	Stories related to places in Trondheim	Stories from three places, including Trondheim	Artworks and some information about them
Usability	- Requires knowledge of the location to find places on map - The picture occupies much of the space in the place and story view - Good help site - Clear and consistent language	- List of locations is easier to navigate than a map and does not require knowledge of the exact location of the places - Picture is dynamic and makes room for the relevant information to the user when scrolling - Lacks a help site	- Two options for starting the and for browsing the artworks accommodates different user groups - Visibility of system status when processing
Personalization	Does not provide any personalization feature	Not implemented in the tested version	Provide personalization using different input parameters.

3.5 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 2.2** for a more elaborate definition). The evaluation here draw on Jacob Nielsen's ten usability heuristics as defined in [**?**].
- Personalization. To what extent does the application provide the user with the opportunity for individualized content?

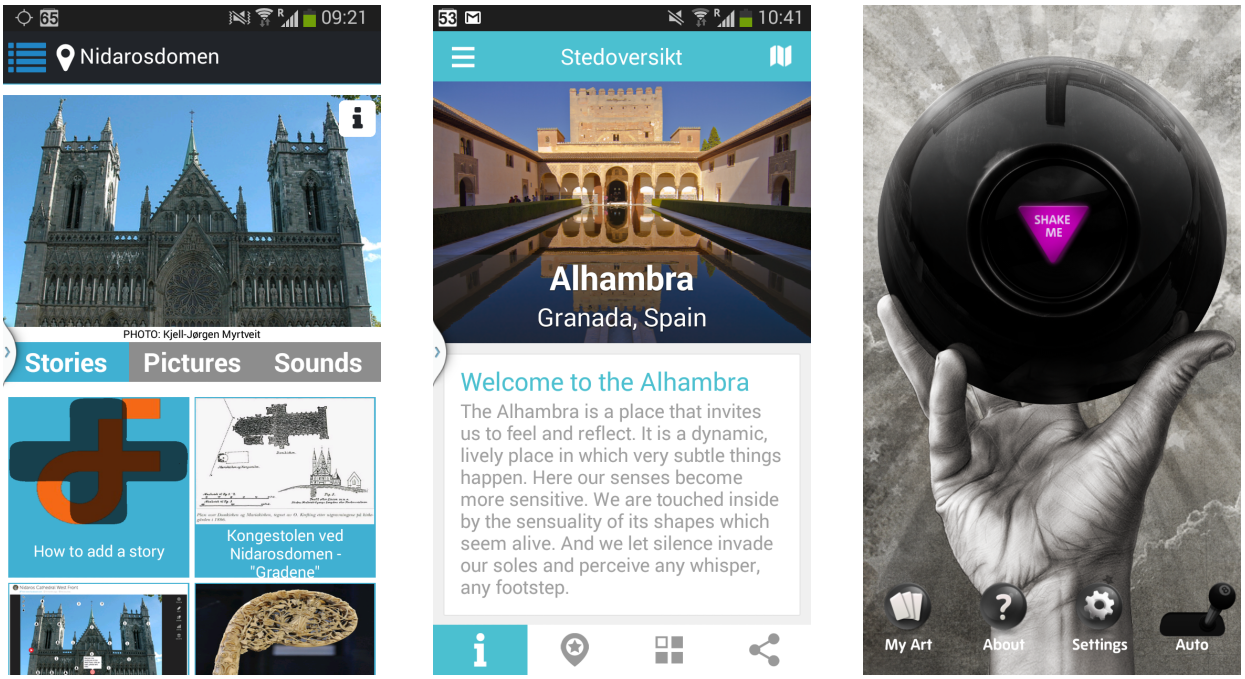
The main findings in the evaluation are summarized in **Table 3.3**.

3.5.1 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. A possible problem in the application is finding the place the user is looking for since the main view consist of a map with markers representing each available place. In order to know which place each marker represents the user have to click on the marker, which means that the user might click on multiple markers do find the desired place.

The place view in stedr can be seen in **Figure 3.1a**. 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, which is also the case when viewing a specific story in the story view. It might be argued that Nielsen's heuristic of aesthetic and minimalist design in case of scrolling is violated as the picture's importance in the dialogue is less important than the space it occupies suggest.



(a) A screenshot from stedr showing the available options for one place

(b) A screenshot of Cooltura showing the view for one particular place

(c) A screenshot of the main view of Magic Tate Ball

Figure 3.1: Place views in stedr and Cooltura and the main view of Magic Tate Ball

The application provides a help site which gives a thorough guide to the user on how to accomplish tasks. This is in accordance with Nielsen's 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 customer's 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.

3.5.2 Cooltura

Cooltura is another application developed under the TagCloud umbrella. The version of Cooltura evaluated here is a demo version, so more developed versions 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. 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. 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 3.1b**. 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 Nielsen's 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 [**?**]. The "Anbefalte steder"-view in Cooltura now list every place added in the application (which is three different places), but the heading suggests that personalization is planned for.

3.5.3 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 [**?**]. The content in the application is the artwork, a description of the artwork and an explanation of why the artwork was chosen. Personalization is the main selling point of Magic Tate Ball. The application uses the input parameters to give the user some control of what content is presented as the user can turn each of these parameters on and off. 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).

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 3.1c** 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 Nielsen's heuristics on visibility of system status.

Chapter 4

Tools

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

4.1 Development tools

4.1.1 Front end

- Ionic [**?**] - 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 3.2.2**

- PhoneGap (Apache Cordova) [**?**] - 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 3.2.1**

- Android Studio [**?**] - 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 [**?**] - Node.js is a platform built on Chrome's 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 [**?**] - 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 [**?**] - 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 [**?**] - 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 [**?**] - 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 [**?**] - 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 [**?**] - 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.

4.1.2 Back end

- Docker [**?**] - 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 SINTEF's own server. More details about Docker is described in **Section 6.3**

- MySQL [**?**] - 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 project's database and its design can be found in **Section 6.2**

- Digitalt musem API [**?**] - The source of the stories in the application was Digitalt Fortalt, and in order to access these it was necessary to use Digitalt museum's API. The API is documented on its website and was incorporated into the project's database. Details about the API can be found in **Section 6.9.1**

- phpunit [**?**] - 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 **Section 8.1**

- Karma [**?**] - 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 **Section 8.1**

- JUnit [**?**] - JUnit is a framework for doing unit testing in Java.

- DBUnit [**?**] - DBUnit is an extension of JUnit that can be used to test methods accessing a database. Among its advantages is the ability to put the database in a known state before each test, which gives the tester control over what to expect as results of tests that change the contents of the database.

- Mahout [**?**] - Mahout is a project that provides scalable machine learning algorithms. Mahout is primarily focused on algorithms for collaborative filtering, clustering and classification. How Mahout has been used in this project is described in **section 6.7**.

Mahout krev
man har Ap
Commons M
Guava-librar
og slf4j. Sk
disse nevnes

4.2 Communication tools

- Google drive [**?**] - 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 [**?**] - Used to share documents between the group and the customer. This tool was used upon request by the customer.

- Facebook [**?**] - 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 [**?**] - 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.

4.3 Additional tools

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

Chapter 5

Project management

The following chapter describes planning the progress of the project, managing resources, controlling potential risks and assure quality of the project. The project management includes a delegation of work tasks and responsibilities, risk management, a work breakdown, time management and quality assurance.

5.1 Risk management

Risk management includes planning and handling all the various potential risks to the project.

The risk analysis 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 is 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. Risks were updated regularly and changes were made when we discovered new issues that needed to be managed.

Table 5.1 describes four of the most important risks in the risk analysis. The complete risk analysis is located in the **Appendix B**.

Table 5.1: Risk list example

Description	**Likelihood(1-9)**	**Impact(1-9)**	**Importance (Likelihood * Impact)**	**Preventive action**	**Remedial action**

The group does not receive quality feedback from supervisor	6	6	36	Be prepared for supervisor meetings. Prepare concrete questions and discuss issues with supervisor	Ask qualified aquaintances to read and give feedback on the report.
Project complexity / Project too difficult	5	5	25	Do not plan many complicated tasks	Downgrade demands
Poor communication within the group leading to misunderstandings and doubts about the progress of the project	4	6	24	Write meeting minutes to document decisions. Have frequent meetings where every team member explains what they have done and what they are planning to do	Make a group decision to solve the misunderstanding

5.2 Scrum team and roles

The role delegation in the team is detailed in **Table 5.2**. 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.

5.3 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 team's 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

Table 5.2: Role delegation

Role	Responsible	Details
Product Owner	Jacqueline Floch (SINTEF)	
Main back end staff	Hanne Marie, Eivind, Kjersti, Audun	
Interaction between server modules	Hanne Marie	This involves making the different server components communicate and work together, such as the database, personalization module, Docker file, etc.
Personalization	Eivind	Involves the procedure for taking a user's context information and preferences, and based on this choose which stories to present to the user. Also involves collaborative filtering to select stories for a user based on the preferences of other users.
Test responsible	Kjersti	Involves creating a test plan for each kind of test (unit, usability, integration, etc.) and documenting the results, as well as assuring test quality and that the tests are actually conducted properly.
Database Manager	Audun	Involves setting up and managing the database. Manage which elements to save in the database and how to present the data upon request.
Main Front end staff	Ragnhild, Roar, Espen	
iOS responsible	Ragnhild	Involves developing and giving the app a look and design that feels intuitive and follows design conventions according to iOS systems. This is both on a technical and design level.
Android responsible	Roar	Involves developing and giving the app a look and design that feels intuitive and follows design conventions according to Android systems. This is both on a technical and design level.
Framework responsible	Espen	Involves developing various parts of the UI, and researching the framework to ensure that the front end design can be made and is made according to the team's and customer's expectations.
Additional roles	Roar, Kjersti, Espen	
Scrum Master	Roar	Project leader, responsible for arranging meetings, delegating work tasks, and overseeing the general progress of the project. Facilitate the communication and cooperation between the group members.
Customer relations	Kjersti	This includes all communication with the customer, as well as other authorities that are a part of the project.
Report management	Espen	This involves overseeing the report work and ensuring that all components of the report are in place. Deliver the report within deadlines and make changes as needed and in accordance with advice from supervisor.

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 **Fig 5.1**

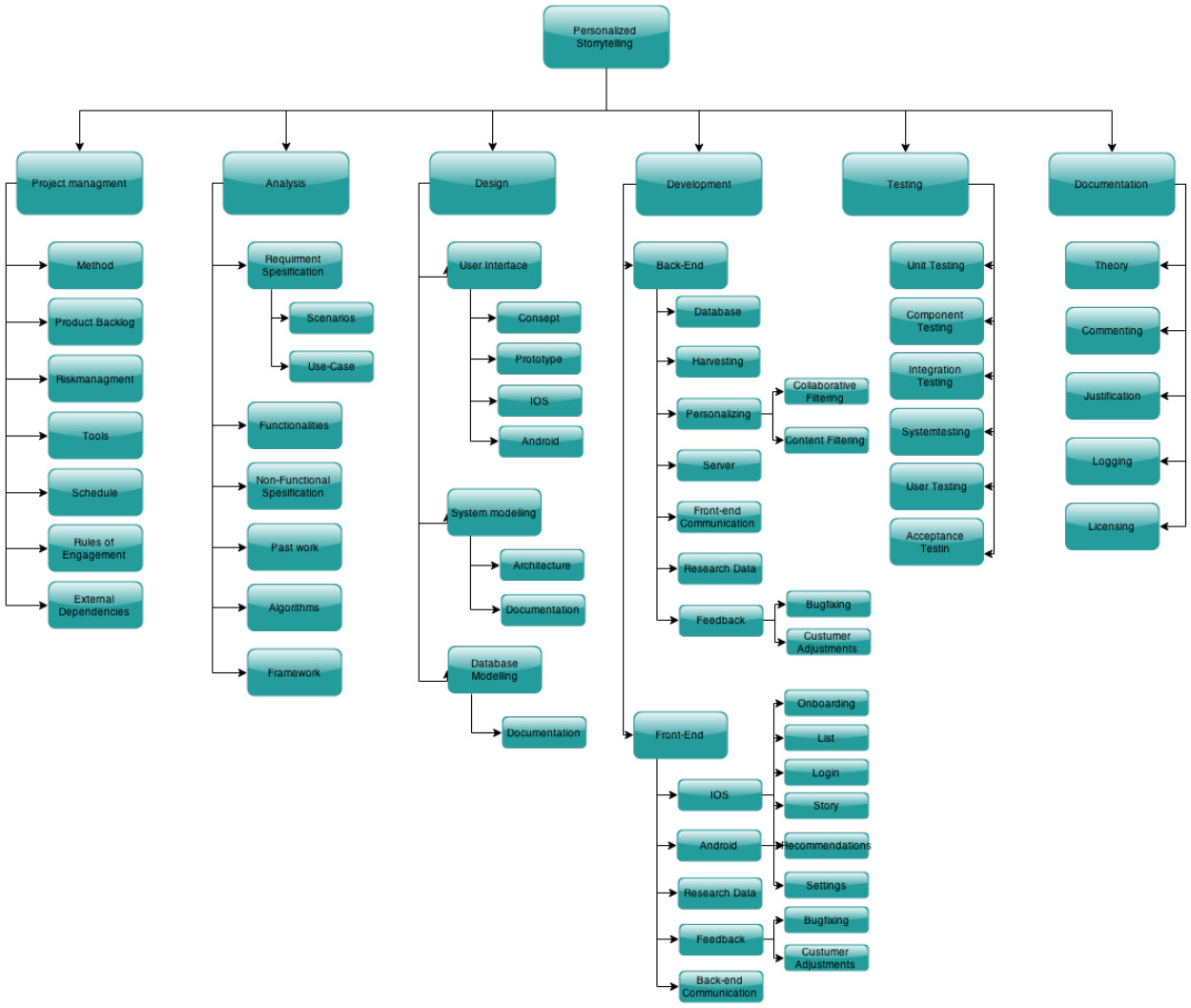


Figure 5.1: Work breakdown structure

5.4 Project milestone plan

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 [**?**]
- 17.03.15 First working software(alpha-version)
- 10.04.15 Second working software(beta-version)
- 01.05.15 Final product

forklar hvor
man kom fr
til dette?, le
hele gantt ti
appendix?

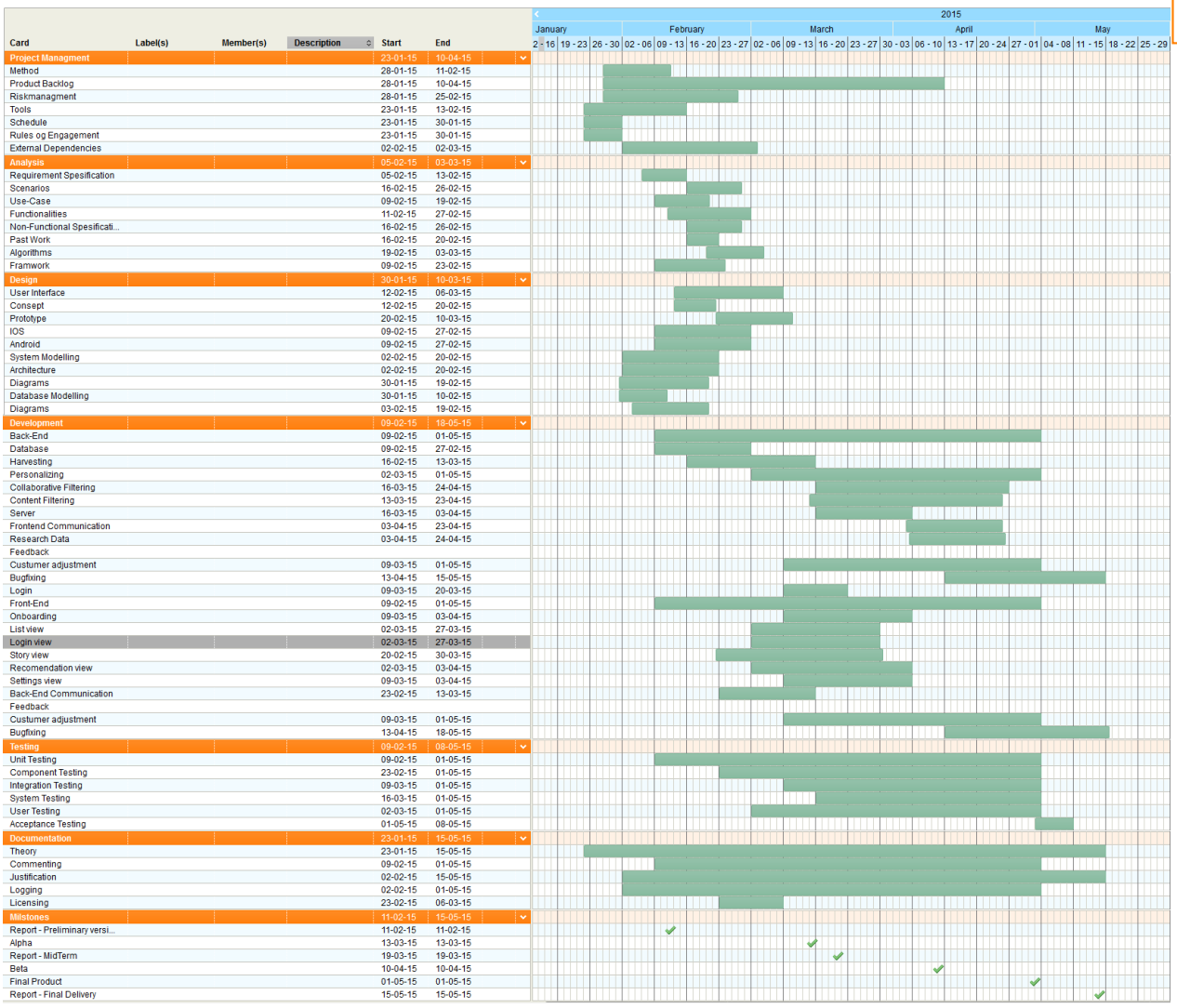


Figure 5.2: Work breakdown structure

5.5 Burn down

s not intro-
? at least
this chap-

Burn down chart is a visual representation for the outstanding work(aka.backlog) relative to the projects remaining time. Its visual representation is outlined in a graph where the x-axis represents the timeline where each point on the graph is a one sprint and the y-axis represents the estimated hours needed to finish the project. This is useful to have better understanding of the rate that the project i moving and it gives indications where or not one will meet the projects estimates.

vi
her?
idiacomputing.com/pub/BetterSoftware-
Charts.pdf,
joel.inpointform.net/software-
opment/burn-
-charts-
al-simple-
project-
ng/

Our estimates are done on the basis of the course no of specialization hours that is said to be about 20 hours pr person

Figure 5.3 Burn down estimate done at the start of the project.

ble sagt i
sning. Men
tet står det
her?, har
mail til
ca

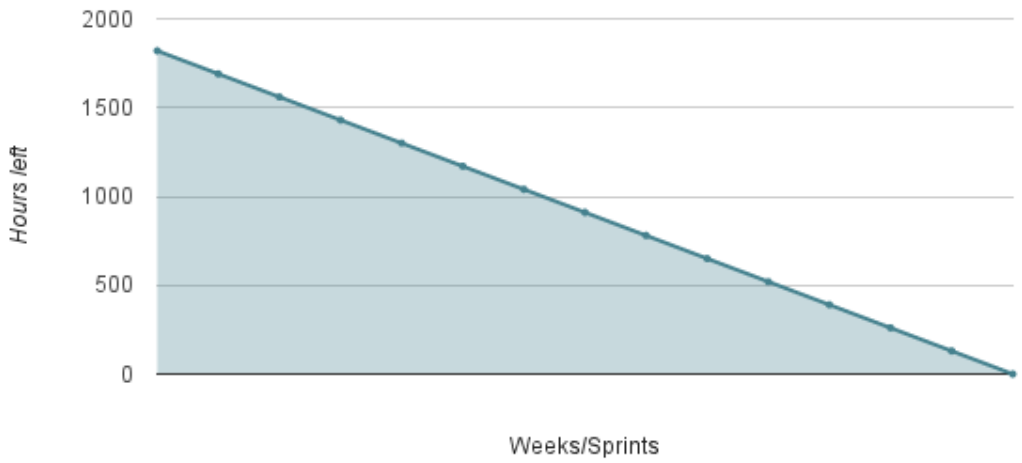


Figure 5.3: Burn down chart

5.6 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” [1, p.652]. 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” [1, p.654] within the development team.

Therefore, this section will describe four 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, code quality and interaction with the customer. Risk management and testing

are also considered important aspects of quality assurance, and these are discussed in separate parts(see **Section 5.1** for risk management and **Chapter 8** for testing).

5.6.1 Group interaction

As was noted in **Section 3.3.3** 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.

This is not i
duced? at le
not in this c
ter

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 (see **Appendix A**). 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 4.2** 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 5.2** 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.

5.6.2 Git and version controlling

The code for this project is hosted by GitHub, a tool described in **Section 4.3**. 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.

5.6.3 Code quality

In order to ease the cooperation on the code for the project, the group followed some general guidelines. These guidelines were:

- Use descriptive names on variables, classes, folders and other elements

- Comment code
- Divide the code into logical units
- Make sure the code units are not too large

5.6.4 Customer interaction

In **Section 3.3.3** 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 are by their request watchers of 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 6

Design and architecture

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

6.1 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 programming. 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, and the other showing the processes in the different components. The overall system structure can be seen in **Fig 6.1**, this shows the four different components and how they interact. As shown in this diagram, the system is created as a client-server model, where the mobile application constitutes the client, which communicates with the PHP backend server. The server is connected to a database, and Digitalt fortalt provides an API to retrieve stories from.

In **Fig 6.2** a diagram showing the architecture with processes can be found. As seen in the legend, the square boxes represent individual components or processes. The boxes with oval corners represent compound processes or larger parts. These are mainly shown because they give an overview of which processes belong to and will be performed by which part of the system. Double lined boxes are external sources which provide an API. Lastly, arrows indicate information flow.

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.

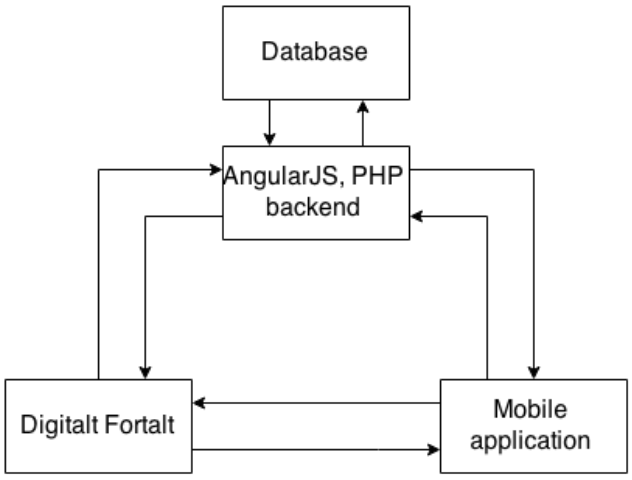


Figure 6.1: Diagram of the overall system structure for this project.

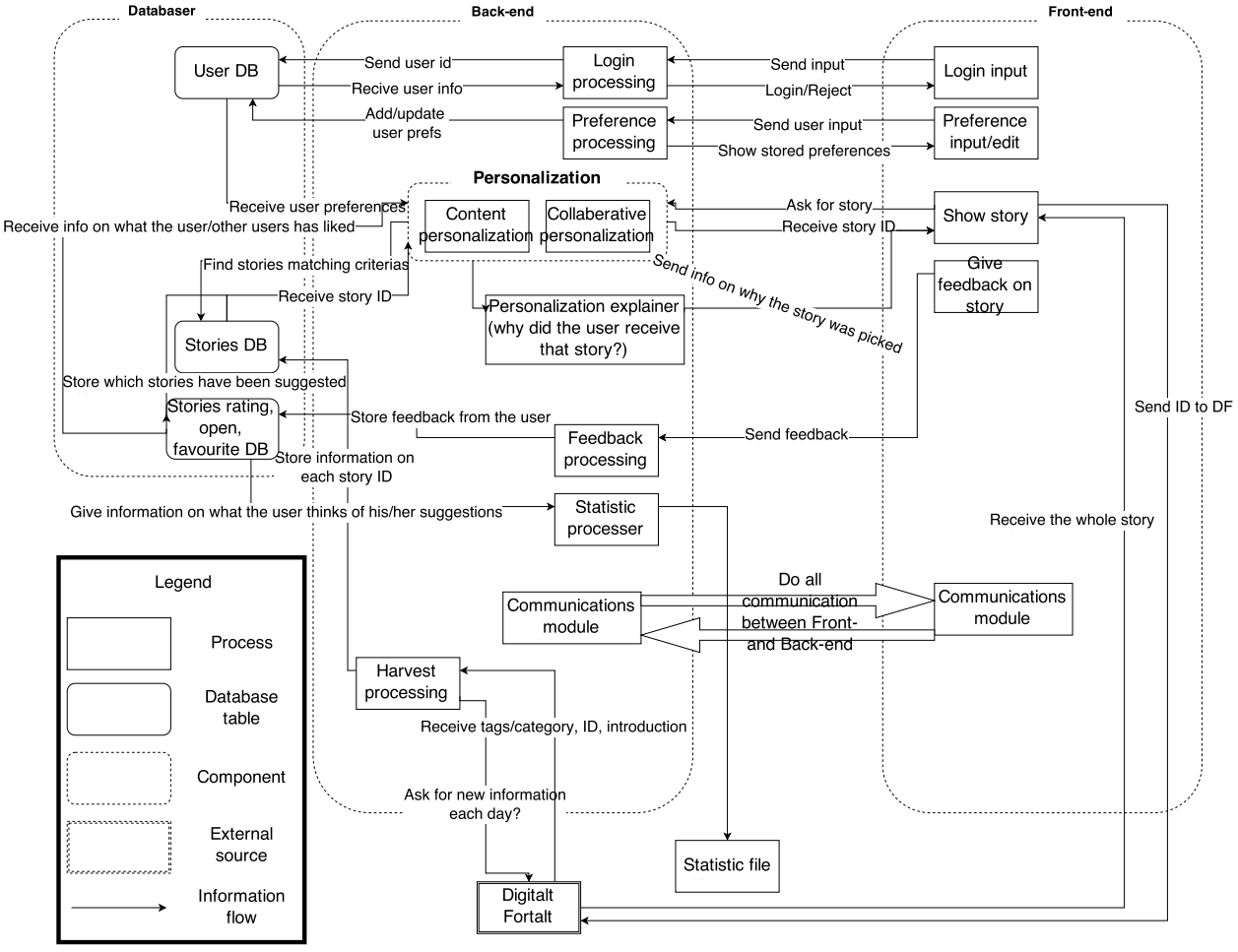


Figure 6.2: Diagram of the architecture for this project.

6.2 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 **Fig 6.3**. 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 **Section 6.6**). 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 was to provide some research data to the customer. To do this, some data about the use of the application is stored. This includes 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.

6.3 Docker

Docker [**?**] 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 a good choice as well. 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.

6.4 front end structure

The front end of the application was designed by using Ionic and AngularJS, and this section will describe how a system made in AngularJS is structured.

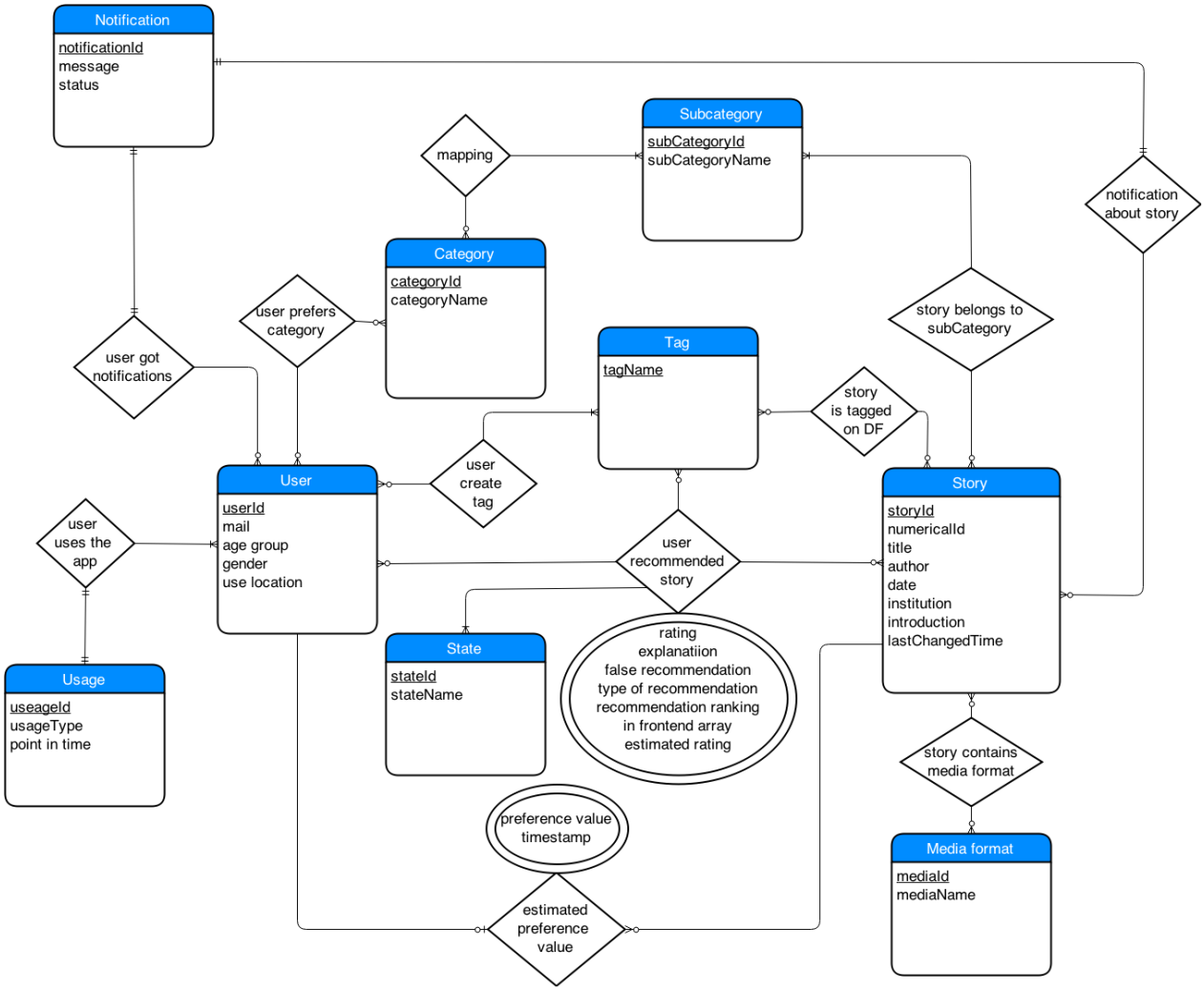


Figure 6.3: ER-diagram showing the data model.

AngularJS provides templates to create systems based on a Model-View-Controller(MVC) architecture, and also provides a variety of components to assist in easing the programming and speeding up the application. It is essentially another layer of abstraction above writing regular Javascript. These are some of the main concepts that are used to create an Angular application:

Model

Manages all the data that can be interacted with by the user. The model also keeps the views up to date and can be manipulated by controllers.

View

The interfaces that the user can see. This means some form of visual representation of the model.

Controller

The logic and behavior of the views. The controllers also make changes the model.

Directive
Special functionality applied to the DOM elements, you can create your own directives as well as use the numerous ones provided by AngularJS itself. In ”Vettu hva?”, several of these were used, such as directives that call on some function when DOM elements are clicked on, or directives that show or hide parts of the DOM based on some condition.

Module
A module encapsulates a part of the application. Instead of having a single ”main” function that holds the application together, Angular applications normally have multiple modules that work together. The benefit of this is that the application is decomposed into logical parts that can be reused, loaded, and tested in any order. In ”Vettu hva?”, each controller is its own module. The part of the program that communicates with the back end is also encapsulated in a module. Furthermore, there is a module that starts up the application and binds together views and controller into different states.

Scope
The scope contains all the elements that the application currently has access too. It can be viewed as a container that stores the current model, and so if a controller or directive is going to modify or access the model, this has to be done by using the scope.

Service
A service contains ”global” logic that is accessible to the entire application. In ”Vettu hva?”, this is used in the module that contains the communication with the back end. This module has three different services, which respectively gives the application access to user data, story data, and requests from front end to back end.

6.5 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 **Fig 6.4** 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 explain 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.

Recommendation view
This view displays the stories that should be the most relevant to the user. The user can

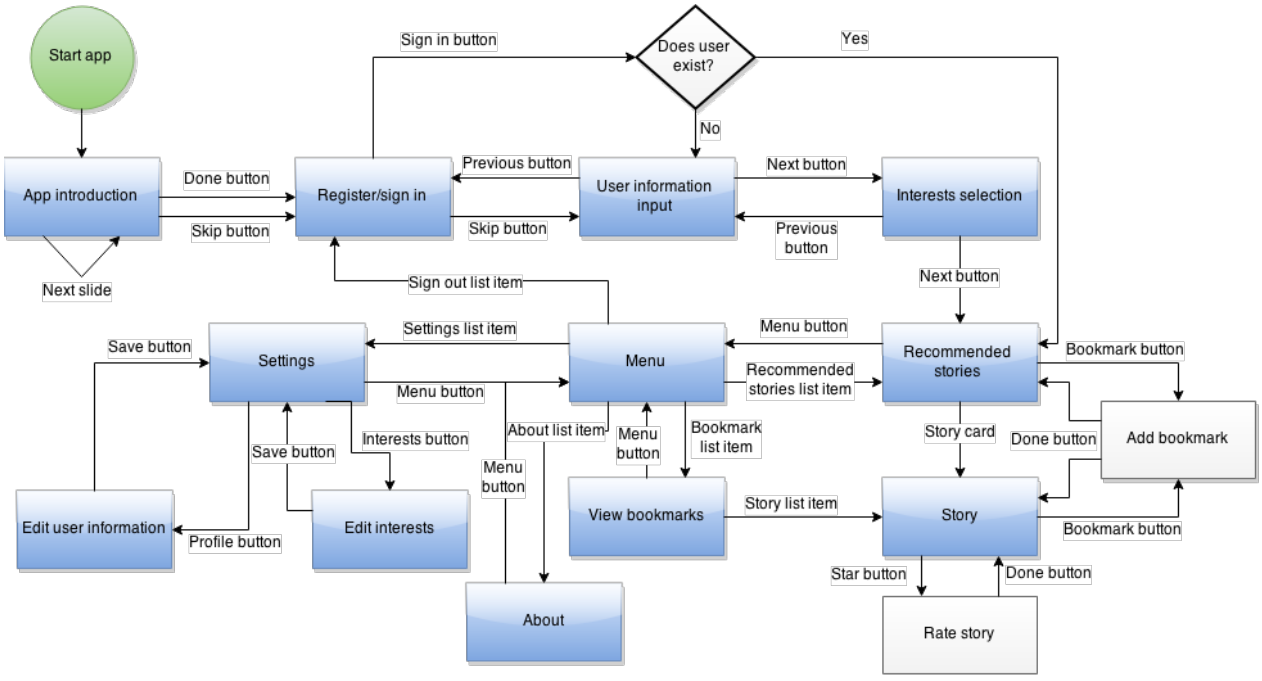


Figure 6.4: Diagram of the flow between views in the user interface.

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 this kind of stories less likely to appear in the future. The bookmark icon in the top bar opens a modal for adding a bookmark.

Story view

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

nshots and
nations of
views

6.6 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 categories of interest.

Figure 6.5: Recommendation view

To make it easier for the user, these subcategories are divided into nine main interest categories. Some of these categories 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 ??** this mapping is illustrated. It is obvious that some category interests will have more stories, and some 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.

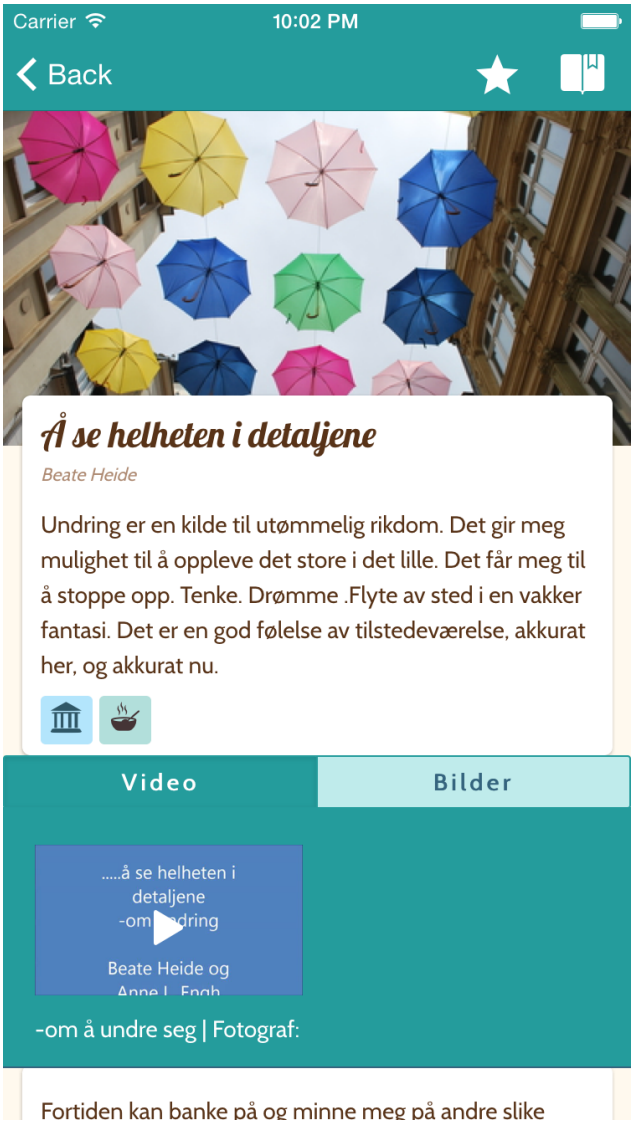


Figure 6.6: Story view

6.7 Personalization

Users receive recommended stories by means of content-based and collaborative filtering, described in **Section 3.4**. The system gathers information about the stories and the users, and feed this information into the recommender engine Mahout which produces a list of recommendations for the desired user. The details of Mahouts inner workings will not be presented here. What will be described is how the input data delivered to Mahout is gathered and produced by our application, what methods of Mahout is used and when they are used, and how the resulting list of recommendations is dealt with.

Both content-based and collaborative filtering rely on giving a numerical value that express how

Table 6.1: Category mapping performed to facilitate, and simplify content based filtering. Each category is assigned to one or more interests.

Archeology	Arkeologi og forminne
Architecture	Arkitektur
Art and design	Bildekunst, dans, design og formgjeving, film, fotografi, media, teater
History	Historie, historie og geografi, kulturminne, sjøfart og kystkultur, språkhistorie
Local traditions and food	Bunader og folkedrakter, dans, fiske og fiskeindustri, fleirkultur og minoritetar, Hordaland, kultur og samfunn, kulturminne, musikk, Rallarvegen, samer, sjøfart og kystkultur, språk, tradisjons- mat og drikke
Literature	Litteratur, teikneseriar
Music	Musikk
Nature and adventure	Fiske og fiskeindustri, naturhistorie, sport og friluftsliv
Science and technology	Fiske og fiskeindustri, fotografi, ”kjøretøy, bil og motor, veitransport”, media, ”natur, teknikk og næring”, ”teknikk, industri og bergverk”, skip- og båtbygging

much a user likes a story. We call this a preference value. In our application this value is computed by combining several measures representing different types of user feedback. These are: the rating a user have given to a story, the categories the user has expressed an interest in, the number of times a story has been recommended to the user, the number of times the story has been opened and the number of times a story has been put in the to-be-read list. Weights are assigned to the measures to differentiate between what impact they should have on the preference value. Rating is considered the most important user feedback, since this value is given to a story by direct user action. The other measures are either less connected to a specific story or more intangible. When computing recommendations, preference values for all stories for the user in question are calculated first.

Each time Mahout is run, up to ten recommendations are inserted in the database. How these are chosen depends on a number of factors. Since a requirement from the customer was to include false recommendations, one of the recommendations is always picked from the lower parts of the recommendation rankings. The other stories are picked from the top of the rankings. Depending on two factors some top recommendations may be skipped for a given run. The first of these factors is that rated stories are not recommended again to the user (only an issue for content-based filtering). The other one is whether the list of recommendations should be added to the existing list of recommendations browsed by the user in the recommendation view or not. When adding to the existing list, recommendations should not be repeated.

Mahout uses a data model and a similarity model to compute recommendations. The data model is a collection of triplets, where each triplet consists of a user ID, a story ID and a preference value. The selection of which preference values to include is different in content-based and collaborative filtering. The similarity model is also different for content-based and collaborative filtering. A

description of these differences follows in the subsequent sections.

6.7.1 Content-based filtering

When doing content-based filtering, the input data model consists of a collection of all the preference values for the user we are making recommendations for. This means that the size of the model will be equal to the number of stories harvested from Digitalt fortalt. To make recommendations Mahout also need measures describing how similar stories are. Similarities between stories are computed based on the subcategories found in the second column in **Table ??** using the cosine similarity formula:

$$similarity = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \cdot ||\vec{y}||} \tag{6.1}$$

The dot product in the numerator is in our system equivalent to the number of common subcategories for two stories, while the denominator is the product of the square roots of the number of subcategories for each of the two stories. If two stories have exactly the same subcategories the similarity value will be 1, and if they do not share any subcategory the value will be 0.

These similarities are put in a CSV file at the time of harvesting. This file is read when the recommendation is run and the values are put in the appropriate Mahout objects. Mahout then uses the data model and the similarity model with similarity values for all possible pairs of stories to produce a ranked list of recommendations using item based recommending.

6.7.2 Collaborative filtering

Collaborative filtering uses a data model with multiple users. Only the stories a user has rated will be included in the data model since this is the most precise user feedback. In our application collaborative filtering is done by combining two different methods provided by Mahout, namely item based recommendations and user based recommendations. The item based approach creates a similarity model based on similarity between items in the data model using Log Likelihood, while the user based approach creates a similarity model based on similarity between users in the data model using Pearson correlation. To prevent dissimilar users from affecting the recommendations in the user based approach, a threshold value for the similarity is set. Two lists of recommendations are produced and then merged to get one list of ranked recommendations.

As was noted in **Section 3.4** collaborative filtering requires a large amount of data to make recommendations. Since only rated stories are included, collaborative filtering cannot be run when a new user starts using the application as the user will not be part of the data model. The criteria for using collaborative filtering needed to be met is that the user has rated at least ten stories which at least ten other users also have rated.

6.8 Back end overview

The back end implementation is divided into five sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of the application. The value of such a separation is that it simplifies the development and maintenance of the application code. By splitting the code into separate parts the system is more logically structured, and cleaner code is achieved. The result is that e.g. SQL related code is maintained by one part of the system, while for instance HTTP processing is executed by another. Below is a description of the different back end sections.

Skal dette v
med? I så f
designarchit
eller implem
tion?

Models
: Consisting of storyModel, userModel and preferenceValue. The models are used to temporarily hold information about a story, a user and a user's preference value for a story to be utilized by other files. Information is either retrieved from the database, sent from front end, harvested from Digitalt museum's API or a combination of two. These models also contain formatting methods, which makes it possible to return story or user information to front end.

Database
: This section contains dbStory, dbUser, dbHelper and harvesting. The db classes are responsible for accessing the database. dbStory contains methods for adding or retrieving story related information from the database and dbUser is responsible for user related information. dbHelper consists of more general methods and is the class which establish a connection with the database. The harvesting script is responsible for collecting all database stories from Digitalt museums API and adding stories to or updating stories in the database.

Skrive om h
vesting og r
erere til det

Personalization
: Consisting of computePreferences and runRecommender. This section computes preference values for each Digitalt fortalt story in the database for each user. runRecommender is also responsible for initializing the Mahout recommendation engine when a user's preference values have been calculated.

Recommender
: This section consists of the java code and recommender.jar file which make up the Mahout recommender (see **Section 6.7**).

Requests
: Contains the controller script, which receives and handles front end HTTP requests and returns JSON responses (see **Section 6.10**).

Figure 6.7 depicts the different back end classes introduced in the previous paragraphs and give an overview of their dependencies.

6.9 Reuse of code

Reusing written code can be a great help to quickly achieve progress when programming. To investigate possibilities of this, the code from stedr was reviewed. However, since stedr used a

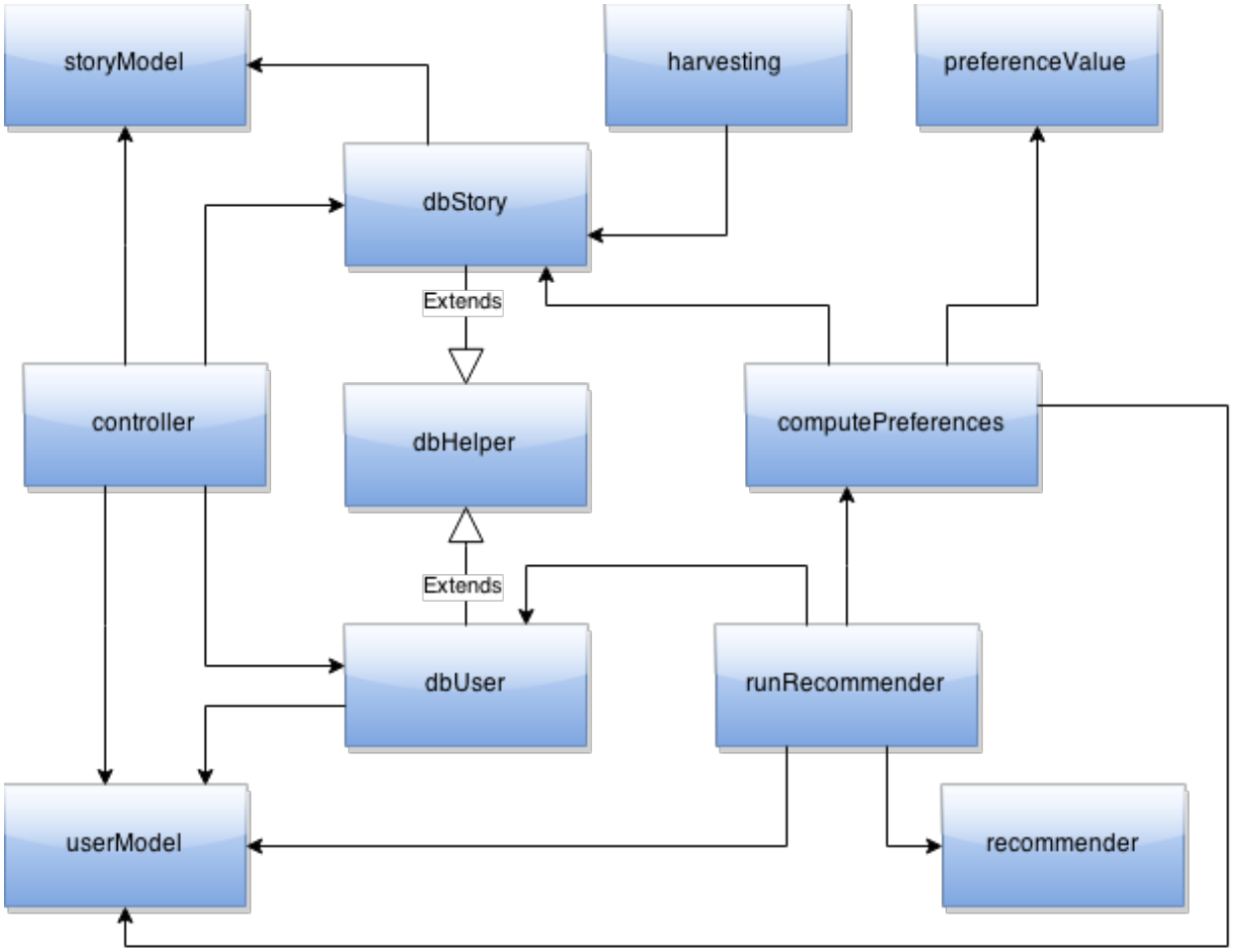


Figure 6.7: Class diagram of the overall back end structure

different framework on the front end and Java as the server language, reusing code proved difficult. Therefore, the front end code was developed without relying on previous work.

6.9.1 Digitalt museum's 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-Trøndelag and Sør-Trøndelag. The API [**?**] 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 [**?**]. This

makes the integration better between the two and the remaining services in Norvegiana. Norvegiana is a data model, database and a web service with the purpose of making cultural heritage information more accessible [**?**]. Example services available in Norvegiana are Digitalt museum, Digitalt fortalt, Arkivportalen and Musikkarkiv.

6.10 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 7

Implementation

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

7.1 Project Progression

Project progression is derived from the activity plan , meeting abstracts and Trello archive and will only briefly summarize what was done at each sprint.

refapp:activi

refapp:meeti

Blir dette re
eret til? + le
inn ref her.
det noe om
lange sprinte
tidliger? når
slutter of sta
klokkeslett

Sprint 1 - 23.01.15 - 30.01.15

The first week mainly consisted of getting familiar with each other and the assignment. As soon as the introductions were done, we talked about our personal goals to set the expectation as a group. We formalized some rules of engagement**??** for the group to sign, and also delegated responsibilities. Later we established a meeting agenda with the customer and met them for the first time.

Sprint 2 - 30.01.15 - 06.02.15

In sprint 2 we began planning and formalizing the requirements. Large sections of work like sketching a WBS **??**, generating a product backlog **??**, and making use cases were done this week. Other design work like the architecture and UI mock-ups were also started on.

Sprint 3 - 06.02.15 - 13.02.15

This was the first week of development. We started testing the API 6.9.1 and finalized some paper prototypes for user testing, and also completed the design for database and architecture. Research towards the personalization, the front end framework 3.2.2 and differences between the two operating systems was also done.

Referanse ti
pirprototype

ref:personali

Sprint 4 - 13.02.15 - 20.02.15

Analyzing the past work 3.5.1 to see if would fit our needs, continuing the research towards collaborative filtering 3.4. Also learning docker 6.3. Mapping the Digitalt fortalt categories to our own and conducting and evaluating the paper prototype tests as well. During this sprint the database code was also starting to take shape.

ref crossplat
OS

ref

Sprint 5 - 20.02.15 - 27.02.15

Sprint 6 - 27.02.15 - 06.03.15

Sprint 7 - 06.03.15 - 13.03.15

Sprint 8 - 13.03.15 - 20.03.15

Sprint 9 - 20.03.15 - 27.03.15

Easter - 27.03.15 - 07.04.15
The application was tested under informal conditions on family and friends, a so-called ”in the wild” test.

Sprint 10 - 08.04.15 - 17.04.15

Sprint 11 - 17.04.15 - 24.04.15

Sprint 12 - 24.04.15 - 01.05.15

Sprint 13 - 01.05.15 - 08.05.15

Sprint 14 - 08.05.15 - 15.05.15

Past 15.05.15
Work was conducted after the last sprint(sprint 14) this was mainly fixing crucial bugs in the application, finalizing the report, as well as carry out an acceptance test with the customer.

ceptance

7.2 Front end

In this section will be an elaboration of some challenges and limitations during the creation of the user interface, both concerning the design and the implementation aspects of the process.

7.2.1 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 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. However, this was not stated until around midway in the project timeline. This fact made it difficult to consider target audience when making design decisions.

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. 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 considerable challenge as well.

7.2.2 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 and customer realized it would be better usability to present the user with multiple recommended stories 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 **Fig 7.1**.

7.3 Back end

This section describes 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 and the personalization.

7.3.1 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 it was for instance 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 how 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 stemmed from the group and were 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 have also influenced the SQL script. This means that a change in the data

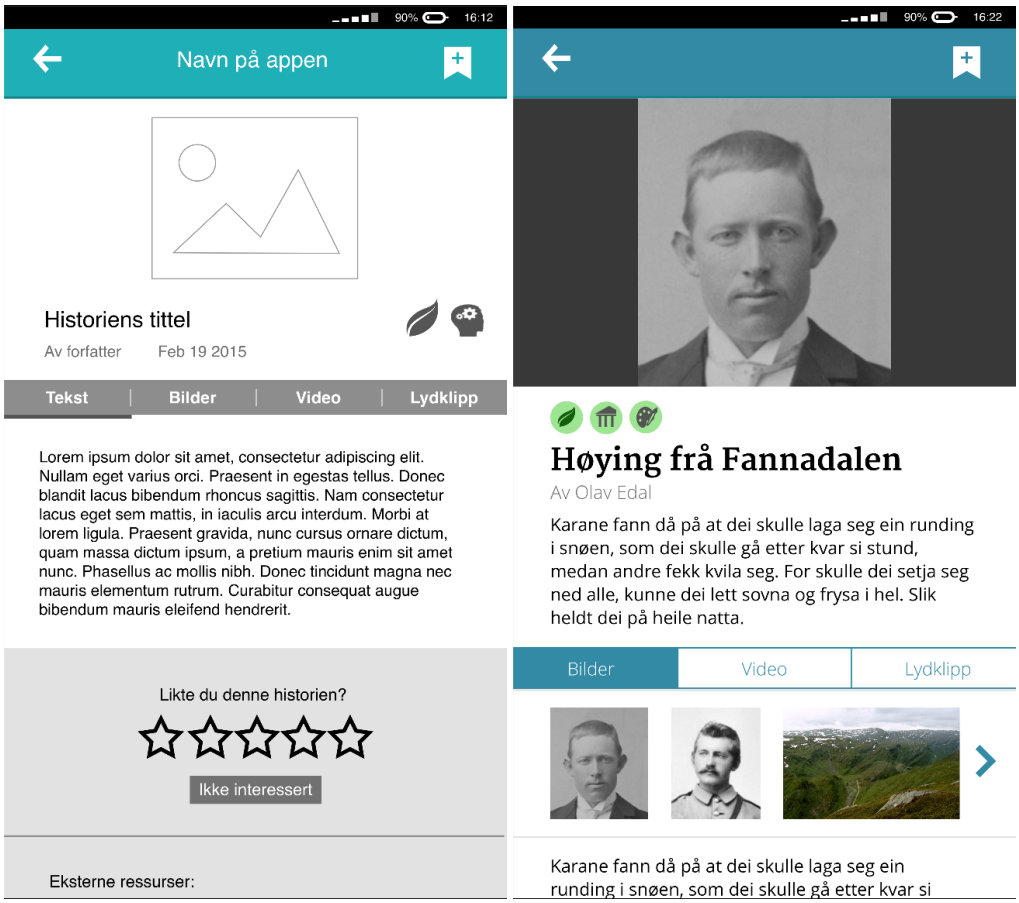


Figure 7.1: Comparison of the story view in the first and second versions of the prototype.

model has 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 relational database was mapped from the ER-diagram using the algorithm in [**?** , p.270-278]. Some of the tables in the database has the potential for NULL values, but this was accepted as it was considered more important to keep the mapping from a entity to a table. In addition, the database already had quite a few tables and further splitting would make the extraction of information more costly, with numerous join operations.

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.

7.3.2 Personalization

At the start of the project much time was devoted to understanding content-based and collaborative filtering and how the theoretical descriptions of these techniques could be turned into a practical implementation in our application. When the back end part of the group had a better understanding on how this could be done, an important decision was whether to use open source recommender engines or to implement the algorithms ourselves. After studying the math and complexity involved in writing our own implementation it became clear that the best solution would be to use existing recommender engines, even if this would require some adjustments to already written code.

As the project was nearly halfway through when the decision to use an open source recommender engine was made, the team did not find the time to do a thorough review of the many different alternatives. This increased the probability of choosing the wrong tool to work with and violated the stated preventive action in the risk list (see **Appendix B**). However, the customer suggested three different recommender engines that might be worth looking into, one of which was Mahout. In choosing a tool the customer had done some research on, the group found that the risk of making a wrong choice decreased somewhat. In addition, Mahout presented its features in a way that resembled the groups research on filtering algorithms and it was therefore reason to believe that getting started with Mahout would not require much additional research. Of particular importance was the fact that Mahout offered an explanation on how to do content-based filtering using their recommender engine. Most such engines provided functionality for doing collaborative filtering,

but it was not always clear how content-based filtering could be done.

Mahouts website provided some guiding on how to build a recommender which made it possible to use the tool without thorough knowledge of how the different methods worked. Most of the work in the beginning of doing the personalization thus centered around how to gather and produce data in a format acceptable to Mahout and how to treat the output recommendations. Content-based filtering was implemented first, as this was the customer's wish and because the first recommendations presented to a user would always be content-based. Since Mahout does not provide methods for finding similarities between items based on their attributes, the group had to implement this. There exist a vast number of functions to compute similarity between objects. The group did some research on different similarity measures, but the literature did not provide a clear-cut answer to which measure was best fitted to our data. Cosine similarity was chosen for its simplicity and because a story's subcategories lent itself well to be represented as a vector.

The customer requested the use of both item based and user based collaborative filtering. There was some doubt in the group as to whether the produced recommendations from the two methods could be combined, but it was found that this was possible as both take the same data model as input. When doing user based collaborative filtering, a threshold value had to be set to tell the recommender engine which users should affect the recommendations. This concerns the similarity found between users. It was difficult to set this value as the computation of the similarity is done by Mahout and because the group did not know what threshold value would be reasonable to set. The value is set by means of trial and error, looking at the resulting recommendations produced using different values, and by looking at example use of this method.

7.3.3 Language

PHP

PHP is a server-side scripting language that is especially suited for web development produced by The PHP Group [**?**]. It is a general-purpose scripting language often used to provide dynamic content from a web server to a client. During the pre-study period several reasons for choosing PHP as the main back end language were presented. Firstly, all back end team members were experienced in the use of PHP. Secondly, as inexperienced users of Docker, more documentation existed on how to implement PHP with Docker than with the other language alternatives considered. Lastly, it was easily combined with the HTTP protocol used by front end to request and receive data (see **Section 6.10**).

DO WE HA
TO WRITE
LANGUAG
ALTERNAT
IF I WRITE
THIS?

Java

Java is a very popular general-purpose object oriented programming language developed by Sun Microsystems and later Oracle Corporation [**?**]. Java was later in the project chosen as a secondary back end language. This was because Mahout, which was used to implement the recommender engine (see **Section 6.7**), is written in Java.

Chapter 8

Testing

The following subsections describes the strategies for the testing levels unit test, integration test, system test, costumer acceptance test and usability test. The unit, integration and system tests were done in a iterative manner. After the test suits were performed and the results were documented, the testers mended 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. The issues that were discovered during all the tests are described in every testing level section.

8.1 Unit testing

The purpose of unit testing is to ensure that every piece of code that are implemented in the system are functional and correct. The group performed unit tests on new units of code before implementing them in the system. It was necessary to prioritize what units that should be tested, considering the amount of time given for the project. The units in this testing was php, java and javascript classes. Therefore it was necessary to use different unit testing tools. Backend code was tested using phpunit framework [**?**], and JUnit [**?**] and the extension DbUnit [**?**]. The user interface code was tested by using a Angularjs unit testing tool, called Karma [**?**]. The classes that were tested is listed up under the unit column in tables 8.1, 8.2 and 8.3. The classes that were left out in the documented testing were tested manually by the developers. If issues were detected during the unit tests, the developers would mend the issues and run the tests again. This cycle was repeated until there was no issues left and all test cases got the expected result.

8.1.1 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 (see **Section 5.2**), so the tester would have good knowledge to the code and know how it works. The delegated responsibilities is presented in tables 8.1, 8.2 and 8.3.

Table 8.1: shows the delegated responsibilities in testing the back end part of the system written in php code.

PHP unit testing		
Unit ID	**Unit**	**Responsible**
U1	Database Helper	Kjersti
U2	Database Story	Kjersti
U3	Database User	Eivind
U4	User Model	Eivind
U5	Compute Preference Value	Kjersti

Table 8.2: shows the delegated responsibilities for the back end part of the system written in java code.

JUnit and DbUnit testing		
Unit ID	**Unit**	**Responsible**
U6	Recommendation	Audun
U7	Database connection with Java	Audun

Table 8.3: shows the delegated responsibilities for testing the user interface.

Angularjs Karma testing		
Unit ID	**Unit**	**Responsible**
U8	UI Login	Ragnhild
U9	UI Story View	Ragnhild
U10	UI Setting	Roar

8.1.2 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. The test cases are presented in Appendix E.

8.1.3 Detected and mended issues

This section includes some of the most important issues that were discovered the unit testing, and how they were solved.

In the database communication there was some changes done during the development which made it easier to handle the code. One example was to make the database connection return an array

where the array keys matched the column names. This way made it easier to access the different values returned from the database. Another issue that was discovered with the media format per story that was not stored correctly in the database, because of an error during the harvesting of the stories from Digitalt Fortalt. These types of issues were solves with several checks to see if the information was in the right format or was not null.

During the tests of the first version of the recommender it was running with the time of 11-12 seconds. This was improved by increasing the efficiency of the communication with the database. The recommender was more efficient when the insertion of preference values and recommendations was done with one insert statement to the database in stead of respectively 167 and 10 statements. Previously the recommender fetched a whole table from the database with the preference values. By fetching a preference value to a specific user helped the recommendation module to run faster too.

8.2 Integration test

Integration testing were performed after unit testing and before system testing. After the back end and front end code was testet and integrated, the integration testing started and focused on that these two modules communicated correctly and that the data was moving between them in the right manner. 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. All the tests was executing by simulating http requests from the UI and check that the back end gives the correct response to the specific http request. If issues were detected during the integration tests, the developers would mend the issues and run the tests again. This cycle was repeated until there was no issues left and all test cases got the expected result.

8.2.1 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 **Fig ??** 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 **Table F.1**.

8.2.2 Detected and mended issues

The following paragaph includes some of the most important issues that were discovered the integration testing, and how they were solved.

The issue that was considered to be the most time consuming one, was to update a user in the database. This issue was detected in the test case I.3. When a user was updated in the UI in the application, the unchanged information that should still be there, was deleted. This was tried to be fixed several times with methods that was not adequate. The problem was eventually solved with fetching all information about a user in the database, update the fields in the user model that had been changed, and then insert all the information in the database again.

Other minor issues that were handled were syntax-issues, redundant code that created unnecessary confusion and missing table attributes in the database.
Changes in the code where made to obtain better structure in the code, such as dividing long code files into smaller ones and to make sure functions returns a response if an error occur.

Due to a misunderstanding between frontend and backend developers, the database returned names of category preference and story category, and not ids.

8.3 System testing

The system testing were perfomed after the unit test and integration tests. The test gave the 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(**Section 2.1.2**) and the specified requirements(**Section 2.1.1**) defined earlier in this report.
In this test one of the developers were executing the test. Because it is a black box test, the tester executed 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. If issues were detected during the integration tests, the developers would mend the issues and the tests was runned again. This cycle was repeated until there was no issues left and all test cases got the expected result. Due to the time limits of this project the team were not able to write scripts to perform the test cases.

When the system test was performed, the testers evaluated the tests results and then decided if the system as a whole fulfilled all goals for the project. If issues were detected, the developers would mend them and run the tests again. This cycle was repeated until the all the test cases got the expected result. The test should, if done in the expected manner, help the developers of this project to verify and validate if the application meets all the requirements.

8.3.1 Test cases

The test cases cover the use cases in 2.1.2 and requirements in C.1. 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 8.4** and **Table 8.5** are presenting two of the test cases that were used. The whole test case document are in Appendix G.

Table 8.4: System test case for creating a recoverable profile.

Test ID	T1
Test Item	Create recoverable profile
Approach	The user locate and press the "register user" button in the app. Applies the email in the correct format.. The response is valid and the user gets feedback.
Input data	"newuser@example.com"
Expected results	The user writes the correct email address and get the correct feedback from the system: "Kontakter server" and will be directed to the startup page.
Testing task	1. Click "create user"-button. 2. Apply email address to the email input field 3. Receive feedback feedback from the system 4. Check email inbox to se if the correct mail from the system was received
Depends on tests	NaN
Pass/Fail	Passed

Table 8.5: System test case for login with email registration

Test ID	T2
Test Item	Log in with email registration
Approach	The user locate the login-button and applies the registrated email and obtain access to the system and the profile connected to this email address .
Input data	valid email: "user@example.com",example invalid email: "mail@example"
Expected results	• The first time the user have logged in System Response: Choose preferences-view should appear. • The user have done this process before System Response: "Vennligst vent mens vi finner historier vi tror du vil like" and direct the user to the view with the recommended stories. • The user types an email with wrong email format System Response: "Ikke en gyldig adresse"
Testing task	1. Navigate to the login view 2. Apply email address to the email input field 3. Receive response from system
Depends on tests	T1
Pass/Fail	Passed

8.3.2 Detected and mended issues

Found repeating recommended stories. This was solved by checking if the frontend array and the top ten recommendations and the ratings done by the user. Picture description, categories ? Sound clips not working. Log in and out, log in again with a different user - Gives the previous users

Fyll ut hvor
de forskjelli
bugsene ble

recommendations. Use a long time to load recommendations - or it doesnt show up at all. Can not scroll down if you are scrolling by touching the frame of video, picture and sound Cant remove bookmark list the user made. Icons are different in different views. Cant scroll if you have many bookmark lists.

8.4 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 brought feedback either in meetings or through email. The planned delivery dates are presented in **Section 5.4** Project milestone plan.

Table 8.6: Customer Acceptance Test - First paper prototype

Delivery	First paper prototype
Date	20.02.15
Comments	Intuitive interface. The selection of categories is good and fast. Category icon in the listview looks very good.
Issues	• It should have a description of why the user have to sign in by email and give the user the option to choose age group and gender, • The customer thinks it is easier to click something than to drag a icon from one place to another. • The customer prefer one story per view when the user browse recommended stories, the swiping from one story to another should be explained. • Customer want a to-be-rated list and a to-read list. To have a trash is confusing. It is okay to not prioritize the notifications in the app.

Table 8.7: Customer Acceptance Test - Second prototype presented in prototyping tool

Delivery	Second prototype presented in prototyping tool
Date	27.02.15
Comments	The customer is overall pleased with the prototype, but they have some constructive comments. There are some confusing icons, some lack of consistent terminology, missing introduction for the app, suggestions for other text for buttons and headlines.
Issues	• Overlap between not interested and one star. Remove the not interested. • The author of a story expects that the story is presented the way he/her made it. It would be more correct to have the elements of the story together, in accordance with the authors intention.The elements of every story is now separated with the tabs in the storyview.
Suggestions	• Have a number connected to the rating stars. • It is interesting for the customer to know if the user prefer picture, video or sound. The system should log this for every user. • It is important to collect the information about the user of the system(age, gender, preferences), want to make it hard for the user to skip this step. Profile information such as age and gender can not be changed after the specification is once set by the user. • Have a little text that appear when you hover the category icons, or apply a function where you can press a button and reveal the descriptions of the categories. • Have the option to share the saved stories on social media. The customer have given this a low priority.

Table 8.8: Customer Acceptance Test -First working software

Delivery	First working software
Date	17.03.15

Comments	The customer likes the user interface of this version and says that it is not necessary to add more functionality, except for the concept view that are not yet implemented. The customer thinks that fetching the stories from Digitalt Fortalt is working fast enough.
Issues	• Missing a concept description for the application • There are some stories that do not have categories connected to them. • These stories should be included in the collaborative filtering.

Table 8.9: Customer Acceptance Test - Second working software

Delivery	Second working software
Date	20.04.15
Comments	The customer is satisfied with the appearance and structure in the story view.

Issues	• There are only 3 stories in the recommendation view When you choose interests in the setup of the application - the interests are not marked when you click on them. • When choosing a gender in the setup of the application, the selection is not stored in the settings view The customer discovered some stories have mismatched icons connected to them. • Stories that include a sound clip, does not view this immediately. The sound clip is located inside a tab, and the user would have to click on this will reveal it. • Some stories include video clips that are not playing. • A user have to sign in every time to visit the app, the user is not remembered. • A story that are read are not automatically stored in the 'Read List'. • Uncertain about the cross in the corner of a story in 'recommended stories view'. The use of this button could mean two things. Either that the user is not interested in this story, or that the user have read this story and just want to close it for now. • The application does not at any time give the user new recommended stories. Missing some kind of feedback when a user has changed the interests in settings. The system should let the user know that it is trying to get new recommended stories based on the new interests. • Some stories do not have a picture attached to it. The customer is suggesting a default picture for these stories.

Table 8.10: Customer Acceptance Test - Final product

Delivery	Final product
Date	01.05.15

Comments	Customer reported positive feedback and expressed contentment towards the final product. The last meeting was spent on going through the requirements list to see if the product met al requirements. There was some comments during the review. R18 is not conducted but the customer sees no problem with this and is satisfied with how the system looks now. R22: The information about the app is now hidden and should be moved to a different location in the menu.
Issues	

8.5 Usability testing

The user testing was performed by the front end developers. The preliminary work for the user test included doing an analysis of the requirements, and used these as a base for making several test cases. A test case included several test steps that the user followed, and the tester 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).

8.5.1 Introduction

The user testing was performed over several days where the first test was conducted the in on the 21th and 22th. of february 2015. This was an early paper prototype while the second session was with the revised prototype on the digital devices (see Proto.io), this was carried out on feb. 28th and march 1th. All the tests were performed in accordance with the guidelines and tips provided in the book: Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition) – 2009

8.5.2 Test Cases

- Create a user
    1. Login
    2. Set personal data
    3. Add at least 2 categories
- Read a story
    1. Browse the suggestions

2. Add a story to ”Les senere”
3. Read another story and look at images

- Find and delete
  1. Find the ”Les senere” collection
  2. Delete a story from the collection
  3. Read another story from the list
- Change settings
  1. Navigate to settings
  2. Change preferences

8.5.3 Test users

table needs
tion and refe
it at some p

Gender	Age	Application usage	Other
Female	26	Medium	Reads some
Male	32	High	Does not read much
Female	51	Low	Reads alot
Male	31	High	Reads some

The test subjects was in advance asked how much and how many applications they use on the handheld devices. They were also asked if and how much reading they see themselves generally doing.

Low - 1-2 applications/ 1-2 times a day. Medium - 2-4 applications/ 2-4 times a day. High - more the 5 applications/ more than 5 times a day.

8.5.4 Summary

Time set aside for each test was about 20 min for the test and with about 10 minutes for some follow-up questions. Both observations during the test and the answers provided in the follow-up is implemented in the view feedback and/or the general changes.

To summarize the tests; each user had his or her own problems with the application but on the whole the user is able to do the task they are asked to complete. So considering the scenarios and

use-cases the application is translating and making the user understand what its functionalities are. There are of course some major inadequate parts, but this is as intended for the test and we are aware of the missing parts.

When it comes to the intuitive understanding, the application has some work to be done. This is covered on a per view basis. And since the user group is not narrow enough we go to design for everybody and this will act as a constraint for the UI.

The parts where users are confused are mentioned in the views section.

Paths are on the whole followed by the majority of the testers with some deviation, but this is expected since the prototype is a bit unclear on its formulation on some of the views. But to conclude, the users are almost without issues following the scenarios to the point.

8.5.5 title

Table 8.11: Usability Test

View	**Desired next step**	**Comments**
1. Intro/Tutorial	2	• "Do I swipe?" • This view is very lacking in content • Are the user able to navigate both ways at the start • It's nice to have a introduction to the application Changes: Add animation/intro

2. Login	3	• Several users don't want to login, they want to test the application first. • "Am I supposed to verify mail after entering?" • The text over Email is not clear about what its for. (Need new wording) • The text was too small • Mail verification? Changes: • Change wording • Make the "Hopp over" button a small link and encourage users to user the "Logg inn" • Add a "?" for hints
3. Profil	4	• Should separate this from the personalization, some user thought this was part of the recommendation. • Is this research data or for the application Changes: • Explain why this is needed and if this is related to the personalizing. • Make the default NONE selected • Add/Change a category to "Hemmelig/Ønsker ikke dele"

4. Interests	5	• "What is this" • This would need an explanation • Can one choose more the one. Changes: • Change the wording: "Velg kategorier"
5.Media format	5	Text¿Image¿Sound¿Video • The sound icon was interpreted as music • Some users did not understand this; it was not the icons what how this affected the stories • several users reported the this was a bit confusing Changes: • Removing this view; because is not adding much to the application and user tests uncovering a lot of misunderstanding about its function and meaning.

6. Recommendations	6	• The read later is very misunderstood, they don't seem to understand that this is placed in a list. And what happen when I press this again • Missing hint to the swipe right/left • The user feels a bit dumped into this view • "What can I touch?" Changes: • Remove button and add a bookmark icon at the top right of the card. • Adding cards at either side to show that there is more. (or adding a card deck to incentive a swipe) • Add a loading animation, Simulating the "magic" that is finding stories for the user. • Add/Change title: Anbefalte Fortellinger/Historier • Make it clear the the card is touchable (ie. make the text fade out in the card) • Add and function to remove the card.

7. Story View	7	• Bookmark icon was not understood by some. • Everything else ok. • “Where do I touch” • A bit of confusion about what the “ikke interessert” button does. Changes: • Add a back button • Change name of button “Ikke interessert” • Make a better link to the icons (Add color etc.)
8.Story view(images and videos)	8	• Expects full screen when touching Changes: NON
Story bookmark	9	• “Can I change the name of the lists?” Changes: None
9.Menu	10	• “What is utforsk?” Changes • Change name of “Utforsk” - Make is clear that this is where the main function/personalisation happens • Skill “Utforsk” og innstillinger (ie. old font)

10. List	10 >12	• Some users is a bit unsure about how to delete a story. • Want to swipe both direction Changes: • Add a hint to show the one can swipe • Add swipe both ways?
11.Settings	4	• This look exactly the same Changes: • Remove the page indicator?

General changes/implementation before next usertest:

Add navigation between the setup views make the user also able to navigate back Add finalized icons Add the intended color scheme

Chapter 9

Evaluation

This chapter describes the final evaluation and reflection by the team for different aspects of the project. This includes positive and negative sides, what worked well and what could have been done differently.

9.1 Product quality

9.2 Development process

We used the agile development methodology Scrum, which has helped our work. The frequent meetings were efficient to coordinate our work and share progress, which was a necessity in a group of 7 members. Scrum is also well suited to respond to requirement changes, which was relevant in this case as the requirements for our application had to be reevaluated and changed several times over the course of the project.

9.3 Project management

We could have estimated time better, as it was significantly off and some tasks proved bigger or smaller than we initially thought. It caused moments of stress when we spent too long on some tasks and didn't delegate enough time, which meant we had to down-prioritize other tasks.

It was hard to use the Scrum sprints effectively. The sprints didn't always have concrete endings, and sometimes it was hard to decide what tasks to work on next. Occasionally we ended up just planning to research instead of specific tasks. We could have done more planning in between sprints and specified the tasks more clearly.

The role distribution has worked well. Every person performed well in their tasks and we finished the tasks that we wanted to. By dividing up the tasks to small packages, it was possible for

each person to work on a specific task at a time, while not having to worry about other factors.

The requirements have been subject to many changes, and because of this, Scrum has worked quite well for us. The process has allowed us to adapt and change the application as needed to reflect the changes in the requirements.

9.4 Team

We had meetings often and kept each other up to date about status, and maintained good communication in general. We agreed about most things and we set ground rules at the beginning for how to work together. We were able to coordinate with each other so we could work independently and still have minimal problems. Everyone contributed and took initiative to get work done. Everyone was quick to point out issues and we took initiative to solve problems as fast as possible.

On the other hand, we divided up the work tasks, for example into front end and back end, and we stuck with it that way. This is not a problem in itself, but it meant that the back end staff did not know much about how the front end was implemented, and front end staff did not know much about how the back end worked. The level of individuality has been maybe too high at times because of this.

Sometimes the creativity stagnated in the group, even though it wasn't necessarily anyone's fault. We had to go back and forth a lot to come up with some solutions. Examples of this were coming up with the name of the application, and also figuring out creative ways to make the user interface user-friendly. On the positive side, the members in the group have been able to come up with good solutions when coding, that have worked well in the end. By criticizing and testing a lot, we have been able to reach good solutions for the implementation. It has been easy to bounce ideas off each other and make decisions as a team.

Cooperation inside each group (front end and back end) has been very good. The members of front end worked well together, and the members of the back end also worked well together. The response from other group members when one person had a trouble has also been very good, and issues has been resolved quickly because of the tight and efficient communication.

Communication between front end and back end staff could have been better. One group did not always know much about what the other did until very late in the project. Even though we did have meetings to share progress, when it came to the code specifics, each group was largely unaware of how the other group had implemented it. The team members could also have been better at communicating when they would not be available for meetings, or show up late. Many meetings were unnecessarily delayed because of this.

In our group we had similar competencies, but different levels of experience. It strengthened our group in the way that some members has special experiences that were useful to perform certain tasks. For example, Docker and Linux experience, as well as database knowledge and the front end

framework used, which some members were able to get much experience in by working with the framework and tools.

9.5 Customer interaction

The customer meetings have been frequent (one meeting every week) and this has been useful. We had many things to discuss, and issues we had to reach an agreement on with the customer. The communication was sometimes difficult, because we did not always understand what the customer wanted, and they did not always understand our thoughts and ideas on how to solve problems.

It has been useful for us to prepare for the customer meetings by writing an agenda before each meeting, and discuss in the group what we wanted to talk about with the customer. On the other hand, we could have been better at coming to an agreement inside the group about issues before each meeting. Occasionally we went to a meeting and team members gave ideas and opinions that not all the other team members understood or agreed with.

We could also have been better at clearly stating our ideas about our solutions to the customer. There has been several incidents where the customer did not understand or agree with our solutions, which cost us time as we had to reevaluate parts of the design.

9.6 Limitations

9.7 Lessons learned

We have learned that it is important to plan the work for each iteration properly. This ended up being a bit confusing when we did not always know what tasks to work on for each sprint.

It is important to evaluate the requirements often, and make sure that we have understood them properly. They need to be referred to often while implementing the application, and they need to be assessed and discussed during customer meetings so that they are understood the same way by the group and the customer.

We should have researched how to implement the personalization techniques earlier. This was started very late in the process and took up a lot of time. The reason it was delayed to begin with was because we wanted other functionality to be implemented first, but in retrospect it should have been started on earlier than we did. The necessary level of research needed for this was underestimated.

Chapter 10

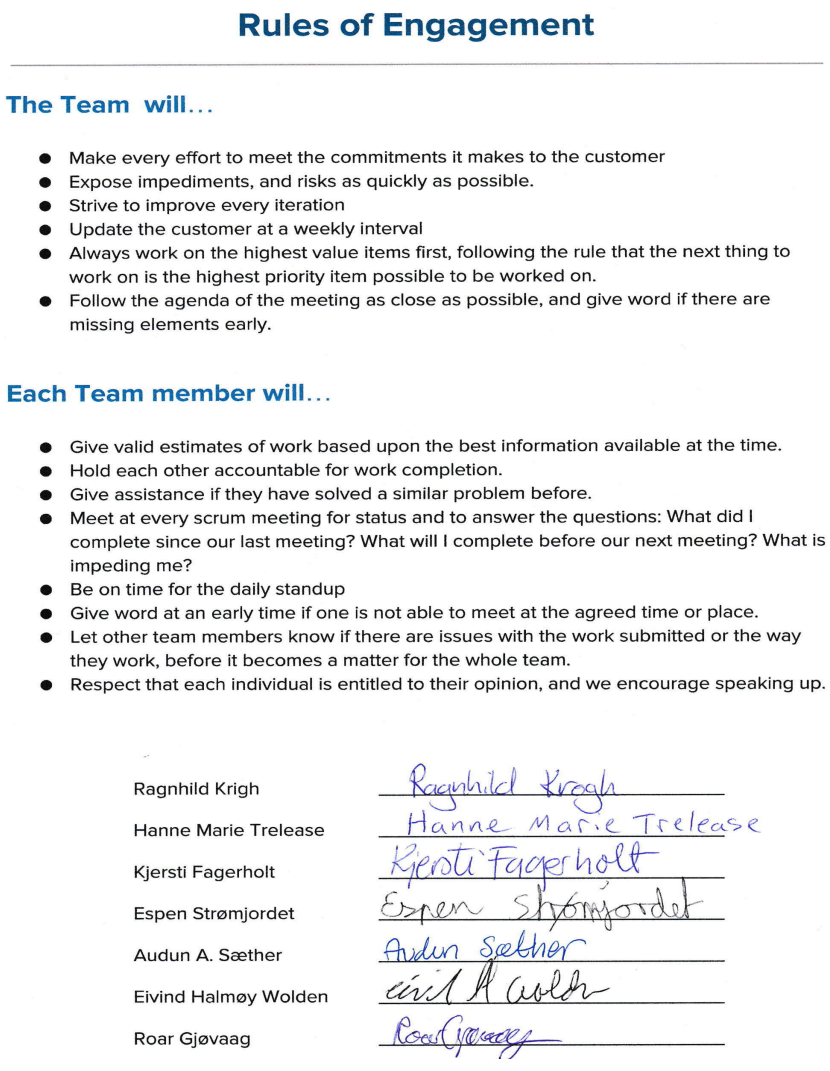
Conclusion and future outlook

Bibliography

[1] Ian Sommerville. *Software Engineering*. Pearson, 9th edition.

Appendix A

Rules of engagement



Rules of Engagement

The Team will...

- Make every effort to meet the commitments it makes to the customer
- Expose impediments, and risks as quickly as possible.
- Strive to improve every iteration
- Update the customer at a weekly interval
- Always work on the highest value items first, following the rule that the next thing to work on is the highest priority item possible to be worked on.
- Follow the agenda of the meeting as close as possible, and give word if there are missing elements early.

Each Team member will...

- Give valid estimates of work based upon the best information available at the time.
- Hold each other accountable for work completion.
- Give assistance if they have solved a similar problem before.
- Meet at every scrum meeting for status and to answer the questions: What did I complete since our last meeting? What will I complete before our next meeting? What is impeding me?
- Be on time for the daily standup
- Give word at an early time if one is not able to meet at the agreed time or place.
- Let other team members know if there are issues with the work submitted or the way they work, before it becomes a matter for the whole team.
- Respect that each individual is entitled to their opinion, and we encourage speaking up.

Ragnhild Krigh — Ragnhild Krogh
Hanne Marie Trelease — Hanne Marie Trelease
Kjersti Fagerholt — Kjersti Fagerholt
Espen Strømjordet — Espen Strømjordet
Audun A. Sæther — Audun Sæther
Eivind Halmøy Wolden — Eivind H Wolden
Roar Gjøvaag — Roar Gjøvaag

Figure A.1: Rules of engagement document

Appendix B

Risk list

Table B.1: Risk list

Description	**Likelihood (1-9)**	**Impact (1-9)**	**Importance (Likelihood * Impact)**	**Preventive action**	**Remedial action**
Underestimate the time planned to use for assignments	8	8	64	Estimate a little higher. continuous meetings. Continuous status update on tasks	Extra work hours and help each other. Have a clear prioritization of tasks so that some less important tasks can be delayed if needed.
The group does not deliver updated information on the report and cannot maximize the quality of the feedback obtained from the supervisor	6	7	42	Always make sure the report meet the demands upon delivery	Ask concrete questions to the supervisor or other competent acquaintances of the group members

Table B.1: Risk list

Description	**Likelihood (1-9)**	**Impact (1-9)**	**Importance (Likelihood * Impact)**	**Preventive action**	**Remedial action**
An issue in the code that is not understood or can not be fixed.	5	8	40	Comment on the code, and talk with each other about the work done on the code	Get help from the supervisor or other people involved.
The group does not receive quality feedback from supervisor	6	6	36	Be prepared for supervisor meetings. Prepare concrete questions and discuss issues with supervisor	Ask qualified acquaintances to read and give feedback on the report.
Project complexity / Project too difficult	5	5	25	Do not plan many complicated tasks	Downgrade demands
Poor communication with the customer leading to misunderstandings and doubts about the progress of the project	4	6	24	Be well prepared for meetings by establishing an agenda for the meeting and sharing it with the customer beforehand. Establish a good customer relationship. Send email to the customer for clarification	Send email to the customer for clarification. Cooperate with customer to reprioritize tasks

Table B.1: Risk list

Description	**Likelihood (1-9)**	**Impact (1-9)**	**Importance (Likelihood * Impact)**	**Preventive action**	**Remedial action**
Poor communication within the group leading to misunderstandings and doubts about the progress of the project	4	6	24	Write meeting minutes to document decisions. Have frequent meetings where every team member explains what they have done and what they are planning to do.	Make a group decision to solve the misunderstanding
Wrong choice of development tools	4	6	24	Do thorough research before deciding which tools to work with	If early in project, consider changing tools
Absence of group member(s) over a period of time	6	4	24	Every member of the group should be aware of what all members are working on, so that they can step in and take over the absent person's tasks	When the progress is halted by a person's absence, other group members should take over the tasks needed for further progress
The workload is distorted. Some members of the group work too much, while others work too little.	5	4	20	Continuous meetings, after every meeting the team members delegate assignments. The leader of the meeting also have to make sure that everyone gets approximately the same amount of work.	Redelegate work tasks within the group.

Table B.1: Risk list

Description	**Likelihood (1-9)**	**Impact (1-9)**	**Importance (Likelihood * Impact)**	**Preventive action**	**Remedial action**
Poor choice of programming language. It becomes difficult to produce the product before deadline	2	9	18	Good research and discussion in the group. Do not choose a language that some people in the group do not have experience with	Reevaluate non-functional requirements with the customer
Customer changes requirements	6	3	18	Constant communication with customer	Use an agile development process to better adapt to changes
Data loss	2	8	16	Local copy and regular backups	Restore latest available backup
Disagreement within the group on key issues in the project	3	5	15	Establish ground rules regarding how to discuss key issues and how to make decisions final. Make democratic decisions	If the disagreement cannot be solved, one may involve the supervisor
Personal conflicts between group members	2	7	14	Establish ground rules for the team so that emerging conflicts are solved as early as possible	First try to solve the conflict between the group members in question. If this doesn't work, involve the whole group. A last resort would be to involve the supervisor

Table B.1: Risk list

Description	**Likelihood (1-9)**	**Impact (1-9)**	**Importance (Likelihood * Impact)**	**Preventive action**	**Remedial action**
Product doesn't meet requirements	2	7	14	Request product feedback from customer	Assess which changes must be made and prioritize the most important parts to change
Overestimate the time planned to use for assignments	3	4	12	Investigate the assignments so it is clear what they include, and how much effort it would take to perform them.	When the assignment is done - long before the time, use the extra time on the more time consuming assignments.
Missing deadlines	1	8	8	Have frequent meetings and plan well	Do extra work hours to finish the work as quickly as possible
Product is not user-friendly	3	2	6	Perform usability tests	Assess which changes must be made and prioritize the most important parts to change
Technical issues(server down)	2	3	6		Use the backup

Appendix C

Requirements

C.1 Requirements

Table C.1: The functional requirements listed up and prioritized by the customers wishes

ID	Name	Description	Priority	Use Case Ref.	Comments
General					
R1	Language	The documentation should be written in english. The application will be written in norwegian and the stories from Digitalt fortalt will appear as they are, most of them in norwegian.	H		
R2	Cross-platform design	The application should run on both Android and iOS.	H		
R3	Cross-platform design	The application design should appear similar on both Android and iOS.	H		
Sign up /Sign in view					

R4	User recovery	The application should provide the opportunity for the user to enter email address, which then becomes the user identifier in the system from the user's point of view.	H	U1,U2	This means that the user can access the profile from different devices.
R5	Anonymous sign in	The application should provide the option to enter the application without registering a user by mail address.	H	U1	Device remembers user. User got id in database, but not mail
R6	Demo view	It should be possible to run the application mode in a demo view where the system can't identify the user	L		
R7	Personal info	The application should obtain some personal info about the user, such as age group and gender.	H	U3	Only for research purposes
Preferences/Settings					
R8	Initial specification of preferences	The user should be asked to set preferences in the startup process.	H	U3	
R8A	Set category preference	The user should be asked to enter a number of preferred categories.	H	U3	
R8B	Set location preference	The user should be asked to specify a preferred location.	L	U3	
R8C	Set notification preferences	The user should be asked to set some preferences about notifications	L	U3	

R9	Changing preferences	The application should provide a settings view where the user can change the preferences.	H	U9	
Main view: Browse recommended stories					
R10	Show list of recommended stories	The application should provide the user with a list of recommended stories based on the set preferences. The stories is presented by a picture and a short text harvested from Digitalt fortalt.	H	U4	
R11	Recommend story outside user's preferences	The application should once in a while recommend a story outside the user comfort zone, i.e. a story that the recommender algorithm does not pick out.	M		Purpose: To broaden the user's horizon. The purpose is not to test the algorithm
R12	Make decision about story	The application should provide the user with three options regarding each recommended story: to choose to read the story now, to reject the story or to save the story for later	H	U4	
Story view					
R13	Present story	The application should present the chosen story in a specific story view.The presentation of the story should be in accordance with the presentation on Digitalt fortalt.	H	U6	

R14	Give feedback/rating on story	The application should provide the user with the opportunity to rate the story. The rating is in the form of a star system with 5 stars.	H	U7	
R15	Tag story / Add story to list	The user should be given the opportunity to connect a story to tags. The tags could be predefined by the system, like ”Les senere” or defined by the user	M	U5	
R16	Link to Digitalt fortalt	Every story should include a link to the corresponding story on Digitalt fortalt.	H	U6	
R17	Explain why a story was recommended	The application should provide an explanation why a given story was recommended.	H		Could just be a general statement like: ”Other users who liked similar stories to you, also liked this one”
List view					
R18	Show list	The application should show a list of collected stories. The stories is presented by a picture and a short text.	M	U8	
R19	Choose list	The application should provide the user with the opportunity to choose different lists. Lists include: to-read,read and user-defined lists	M	U8	
Notifications					

R20	Notifications outside the application	The application should send a notification to the user's device at the time specified in the preferences	L		In agreement with the customer this requirement will not be met
R21	Notifications inside the application	The application should create a notification after a defined amount time to remind the user of stories that have been read but not rated	L		In agreement with the customer this requirement will not be met
About app					
R22	About the application	The application should include an about section, which should include basic info about the project. This include references to TAGCLOUD.	H	U10	
Quick tour					
R23A	Quick tour at start up	The application should provide a new user with an explanation of the application at start up.	H		
R23B	Quick tour in menu	The application should provide the opportunity to revisit the quick tour via the menu.	L		
Personalization					
R24	Use content-based filtering	The application should use content-based filtering to recommend stories to user initially. This should in particular be based on category preferences.	H		Implement this before R25.

R25	Use collaborative filtering	The application should collaborative filtering to recommend stories when the user base is large enough for the algorithm to be effective.	H		
Research					
R26	Gather data to SINTEF for research	The application should gather and store in a file information about the use. This include frequency of use, success rate of recommendations and perhaps other things.			Detailed list of what this included provided in mail from the customer. The customer will view this data through the database

C.2 Quality attributes

Table C.2: The quality attributes used in this project. Prioritized by the customer

Quality attribute	**Description**	**Priority**	**Optionally; Why?**
Usability	How easy is it for the user to accomplish a desired task and what kind of user support should the system focus on. (eg. tutorials or hints)	1	
Monitorability	How important is the ability to monitor how the system while its executing. (eg. statistics ect.)	2	
Modifiability	How important and how easy should the product be able to be changed after it‘s finalized? (eg. making changes to the UI)	3	
Performance	How important is time and speed of the system? (eg. response time for fetching stories)	4	
Interoperability	How important is the ability for the system to work together with other systems. (eg. making use of specific communication protocols or the use of a specified data format)B8	5	
Availability	How important is the reliability of the product.The easy representation to think of this is uptime of the service.	6	
Security	How important is the systems ability to protect data and information from unauthorized access. (eg. losing personalized data)	7	
Testability	How important is the ability to set up tests for the system (eg. setting up automated test for components and parts of the system)	8	

Appendix D

Status report example

Status report week 6

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

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.

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.

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 E

Unit test cases

Table E.1: Here presented by a testId, description of how the test should be perfomed, what input data to use and expected results.

Test case ID	**Description**	**Input data**	**Expected results**	**Result**
Unit 1: Database Helper				
UN1.1	Update one rating value in the database	Random userId, random storyId, updateValue: 2	The function should return true when running the database request.	Pass
UN1.2	Update rating values in the database	Random userId, random storyId, update Value : 5	The function should return true when running the database request	Pass
UN1.3	Update new tag that user have created	Random userId, tagName: 'NyTestTag2'	The function should return true when running the database request	Pass
UN1.4	Update a users tag which is connected to a story	Random userId, Random storyId, tagName: 'Les senere'	The function should return true when running the database request	Pass
UN1.5	Update a users action of rejecting a story	Random userId, Random storyId	The function should return true when running the database request	Pass

UN1.6	Update a story as recommended	Random userId, Random storyId	The function should return true when running the database request	Pass
UN1.7	Get rating on a story done by a user	Random userId, Random storyId	Should return row with the rating presented as a integer	Pass
UN1.8	Get the predefined tags connected to a user	Random userId	Should return row with the tags ”Les senere” and ”Les” which all users in the system is connected to. If this user has self made tags, they should be included	Pass
UN1.9	Get all the tags connected to a user	Specified userId: 105	Should return row with the tags ”Les senere” and ”Les” which all users in the system is connected to, included the added tag ’NyTestTag’	Pass
UN1.10	Get the tags connected to a story and a user	Specified userId: 105, specified storyId: ’DF.1295’	Should return row with the tag ’NyTestTag’	Pass
UN1.11	Get stories with timestamp	-	Should return rows with 167 storyId and a timestamp which indicate when the story was last changed	Pass
UN1.12	Get subcategories to a specific story	Random storyId	Should return a row with an array of subcategory Ids	Pass

UN1.13	Get story information	Specified userId: 105, specified storyId: 'DF.5220'	Should return row with an array with the keys: userId, storyId, explanation, rating, false_recommend, recommended_ranking, in_frontend_array, estimated_Rating	Pass
Unit 2: Database Story				
UN2.1	Fetch Story	Specified storyId: 'DF.1812', specified userId: 105	Should return row with an array of the categories to the story and tags and tagName that are connected to the userId	Pass
UN2.2	Get recommended stories for a user	random userId	Should return 10 rows with stories, and they should include: userId, storyId, recommended_ranking, explanation, false_recommend, title, introduction, author, categories and mediaId	Pass
UN2.3	Get list of stories which is tagged by a user	Specified userId: 103, tagName: 'NyTestTag'	Should return rows with stories, and they should include: storyId, title, author, introduction, date, tagName, categories and mediaId	Pass
UN2.4	Get subcategories per story	-	Should return 167 stories with numericalId and subcategory ids	Pass
UN2.5	Get all stories with storyId, numericalId and categories	-	Should return 167 rows with storyId, numericalId and categories	Pass

UN2.6	Get states per story	Specified userId: 258, specified storyId: 'DF.1600'	Should return a row with stateId, numTimesRecorded and latestStateTime	Pass
Unit 3: Database User				
UN3.1	Add user	example mail: example@example.com	Should return a userId that is not null	Pass
UN3.2	Get user from Id	UserId: generated new Id.	Should return a userId from the database which is equal the userId input	Pass
UN3.3	Update user mail	UserId: generated new id, email: newmail@example.com	Should return a email from the database which is equal the email input	Pass
UN3.4	Update user age	UserId: generated new Id , age group: 0	Should return a user model from the database, which includes the age group that is equal to the age group input.	Pass
UN3.5	Update user categories	UserId: generated new id, category preference: [2]	Should return a row with user model from the database with the category preference that is equal to the category preference input.	Pass
UN3.6	Get user from email	UserId: generated new id, email:54@example.com	Should return a user model from the database, which includes the mail that is equal to the email input.	Pass
UN3.7	Get user categories	UserId: generated new id, category preferences: [1,3,5,7,9]	Should return the categories from the database that is equal to the category preferences input	Pass

UN3.8	Get user mail from Id	UserId: generated new id	Should return a user model from the database,which includes the email that is equal the email input.	Pass
Unit 4: User Model				
UN4.1	Initiate User	userId:1, mail: example@example.com	Should create a user model with the correct input. The getters should return the values that matches with the input values	Pass
UN4.2	Set all user details	userId:1, mail: example@example.com,gender 0, age group: 1, use of loc. : 0 , category preference: [1,3,5,7,9]	Should add the correct user details to the user model. The getters should return the values that matches with the input values	Pass
UN4.2	Print All	userId:1, mail: example@example.com,gender 0, age group: 1, use of loc. : 0 , category preference: [1,3,5,7,9]	The printAll function should return a string with all the correct attributes and values that matches the input values .	Pass
Unit 5: Compute Preference Value				
UN5.1	Compute preference value for all stories for a user	Random userId	Should return 167 rows from the database, and every row should include: storyId, userId, numericalId and preferenceValue	Pass
UN5.2	Compute preference value of a story for a user	Random userId , specific 'DF.1098'	Should return an array with userId, storyId, numericalId and the preference value	Pass

UN5.3	Compute preference value	Specific storyId	Should return the preference value which is the type double	Pass
Unit 5: Run Recommender				
Unit 6: Recommendation				
UN6.1	Run recommender	Random userId		Pass
UN6.2	New content based recommendation	Input: setUp.xml - a data model with userIds and connected preference values.	The recommender should return the correct number of recommendations when it should create new ones	Pass
UN6.3	Add content based recommendation	Input: setUp.xml- a data model with userIds and connected preference values.	The recommender should return the correct number of recommendations when its adding recommendations to existing ones	Pass
Unit 7: Database connection with Java				
UN7.1	Insert recommendation	userId: 1, storyId: Df.1098, DF.1501, explanation: 0, false_rec. :0, ranking: 3,4, estimatedValue: 0,0	Expected table in XML-representation(insert-expected.xml) is the same as the actual from the database	Pass
UN7.2	Insert Update	userId: 1, storyIds: DF.1709,DF.1849, explanation:“updated”, false_rec. : 1,0, typeoffrec. :1,1, ranking: 3,4 estimated-Value: 4.5, 2.5	Expected table in XML-representation(insertUpdate-expected.xml) is the same as the actual from the databas	Pass

UN7.3	Delete recommendations	UserId: 1	Expected table in XML representation(delete-expected.xml) is the same table returned from DB	Pass
UN7.4	Rated	UserIds: 3,5 numericalId: 1812,1901	The getRated function should fetch the correct stories. The input rating should be the same as the returned ratings	Pass
UN7.5	Test frontend array	Numerical Ids: 1849, 1901	Get stories from frontend array should match the fronted array in XML format(setUp.xml)	Pass
UN7.6	Create explanation	Numerical Ids: 1098,1115,1501	The create explanation function should return a string with the correct storyIds and their titles.	Pass
Unit 8: UI-Log in				
UN8.1	Log in with an already existing user - Check if response is correct	existinguser@ mail.no	The user should get access and be directed to the recommendation view(main view)	Pass
UN8.2	Log in with a non-existing user - Check if response is correct	“newemail@ gmail.com”	The user should get access and be directed to the setup view.	Pass
UN8.3	Login a user with the wrong email format - Check if response message is correct	“newemail”	The user should not get access and get a textual response from the system that the format of the input is wrong.	Pass
Unit 9: UI-Story View				

UN9.1	If story contains sound clip or video, check if these are presented properly	- story with video -story with sound clip	Should show the presence sound clip and video in the tabs. Should be able to handle the different video types, and be able to play them.	Pass
UN9.2	Give a rating - Check if the stars change color and response message are visible to the user.		Stars should change color when user performs rating, and the user should get a response message from the system which says that this story has been rated.	Pass
UN9.3	Give the story a new bookmark with a name -add this -Check if response message is visible to the user	"new bookmark"	When a new bookmark is made and the story is stored here, the user should get response message about this.	Pass
Unit 10: Settings				
UN10.1	Update profile with the wrong email format	updateemail	The user should get a response message from the system that the input was not valid and that the user should try again.	Pass
UN10.2	Update profile with a email that already exists in the system	"alreadyexistingemail @email.com"	The user should get a response message from the system that the input was not valid and that the user should try again.	Pass

Appendix F

Integration test cases

Table F.1: Integration Test Cases

Test case ID	**Description**	**Input data**	**Expected results**	**Result**
I.1	Simulate a http request which will log in a user for the first time with an email adress Call getUserFromEmail to the database	email: 'testnr16@example.com, requestType: addUser	The http request should return successfull message included the newly created userId. The database function should return a row with the userId, mail, age_group, gender and use_of_location	Pass
I.2	Simulate a http request which will log in a user for the first time without an email adress. /newline Call getUserFromId to the database	email:null, request type:addUser	The http request should return a successful message included the newly created userId The database function should return the usermodel containing userId, mail, age_group, gender and user_of_location.	Pass

I.3	Simulate a http request which will update a users profile. Call getUserFromId to the database. Check if the updates where correct	email: 'testMail23@example.com', request type: updateUser,	The http request should return a successfull message. The database function should return a usermodel where the email match the input.	Pass
I.4	Simulate a http request that will create a new test user without an email, Simulate another http request which will update email to this test user, with an email that already exists in the system Call the getUserFromId function to the database. Check if the user is not updated	email: 'testMail23@example.com'	The http request should return a failure message. The database function should return an usermodel where the email is null.	Pass
I.5	Simulate a http request that will create a new test user without an email, Simulate another http request which will update email to this test user Call the getUserFromId function to the database. Check if the user is updated		The http request should return a successfull message. The database function should return an usermodel where the email matches the input.	Pass
I.6	Simulate a http request that will create a new test user with an email Simulate another http request that will return the user model from the users email Check if the returned data has all attributes required, and that the data match with the input data	email: 'getUserFromEmailTest@example.com", request type: addUser/getUserFromEmail	The http request should return a usermodel with the attributes userId, email, age_group, gender, use_of_location and with the data which match the input data.	Pass

I.7	Simulate a http request that will create a new test user without an email Simulate another http request that will return the user model from the users id Check if the returned data has all attributes required, and that the data match with the input data	email: null, request type: addUser/getUserFromId	The http request should return a usermodel with the attributes userId, email, age_group, gender, use_of_location and with the data which match the input data.	Pass
I.8	Simulate a http request that will create a new test user without an email. Call the getNumberOfRatingsDoneByThisUser to the database Simulate another http request which will connect a rating to a story for this user. Call the getNumberOfRatingsDoneByThisUser to the database again Check if another rating is registered	email: null, request type: rating, storyId: 'DF.52201, random userId	The http request should return a usermodel with the attributes userId, email, age_group, gender, use_of_location and with the data which match the input data.	Pass
I.9	Simulate a http request that will create a new test user without an email. Simulate another http request where the user rejects a story. Check if the rejection is stored	email: null, request type: rejectStory		Pass

I.10	Simulate a http request that will create a new test user without an email. Simulate a http request that will create a new tag for this user Simulate a http request that will return the list of tags for this user Check if the new tag has been added	request type: addUser/addNewTag/getList tagName: 'newTag1'	The returned list should only include the one tag that where created.	Pass
I.11	Simulate a http request that will create a new test user without an email. Simulate a http request that will create a new tag for this user Simulate a http request that will return the list of stories contained in this taglist Check if the new tag has been added	request type: addUser,tagStory,getList tagName: 'newTag1', storyId: 'DF.5223'	The returned list should have the recently added story in the top of the list. The list should have the following attributes connected to every story: id, title, description, false_recommend, explanation, picture, thumbnail, categories, mediaType, author, data	Pass
I.11	Simulate a http request that will create a new tag for a user. Simulate a http request that will return all the tags connected to this user Check if the tag was stored Simulate a http request that will remove the new tag that were created earlier Simulate another http request that will return all the tags connected to this user Check if the new tag has been removed	Specified userId: 105, request type: addnewTag, getAllLists, removeTag, tagName: 'tagToBeRemoved', storyId: 'DF.6081'	The returned list should have the recently added story in the top of the list. The list should have the following attributes connected to every story: id, title, description, false_recommend, explanation, picture, thumbnail, categories, mediaType, author, data	Pass

I.12	Simulate a http request that will return the list connected to a tag for an user	Specified userId: 105, request type: getList, tagName: 'Les senere'	Should return a list of the stories connected to this tag. The stories should have the following attributes: id, title, description, false_recommend, explanation, picture, thumbnail, categories, mediaType, author, date	Pass
I.13	Simulate a http request that will return all the tags connected to a user	Specified userId: 105, request type: getList, tagName: 'Les senere'	Should return a list of the tags. The tags should have the attributes text and checked.	Pass

Appendix G

System test cases

Table G.1: System test case for creating a recoverable profile.

Test ID	T1
Test Item	Create recoverable profile
Approach	The user locate and press the “register user” button in the app. Applies the email in the correct format.. The response is valid and the user gets feedback.
Input data	“newuser@example.com”
Expected results	The user writes the correct email address and get the correct feedback from the system: ”Kontakter server” and will be directed to the startup page.
Testing task	1. Click “create user”-button. 2. Apply email address to the email input field 3. Receive feedback feedback from the system 4. Check email inbox to se if the correct mail from the system was received
Depends on tests	NaN
Pass/Fail	Passed

Table G.2: System test case for login with email registration

Test ID	T2
Test Item	Log in with email registration
Approach	The user locate the login-button and applies the registrated email and obtain access to the system and the profile connected to this email address .
Input data	valid email: “user@example.com”,example invalid email: “mail@example”
Expected results	• The first time the user have logged in System Response: Choose preferences-view should appear. • The user have done this process before System Response: ”Vennligst vent mens vi finner historier vi tror du vil like” and direct the user to the view with the recommended stories. • The user types an email with wrong email format System Response: ”Ikke en gyldig adresse”
Testing task	1. Navigate to the login view 2. Apply email address to the email input field 3. Receive response from system
Depends on tests	T1
Pass/Fail	Passed

Table G.3: System test case for initial settings

Test ID	T3
Test Item	Set initial settings
Approach	The user is logged in to the system for the first time. The user will choose age group and gender in the inital settings that will appear the frist time the user is logged on to the system. After this the user will be asked to choose cultural category preferences.
Item pass/Fail criteria	
Input data	User action: click the buttons for the age group, gender and interests, and next buttons
Expected results	The user will click the buttons for age group, gender, and interests. The buttons will change color when clicked on. The user navigates to next view by clicking the next button and will obtain access to the system. If the user do not select interests the user will get a response for the system that it is necessary.
Testing task	1. Start app 2. Click the correct age group and gender. 3. Navigate to the next page 4. Navigate to the next page without selecting interests 5. Receive feedback from system 6. Select two interests 7. Navigate to the next page
Depends on tests	NaN
Pass/Fail	Passed

Table G.4: System test case for browsing recommended stories

Test ID	T4
Test Item	Browse recommended stories
Approach	The user is shown a list of recommended stories. The user will click on the first story. The story will be viewed and the user closes it.
Item pass/Fail criteria	
Input data	User action: click arrow
Expected results	The view should show a list with recommended stories. The view of the stories will include a title, story picture or default picture, an introduction, category icons, media icons and a explanation of why this story is recommended to the user. The view should show 10 more recommendations when the user have browes through the 10 first stories in the list. When the user clicks a story the full story should appear in a full screen, including text and potential pictures, videos and sound clips.
Testing task	1. Click on a recommended story 2. Locate back-button and close the story 3. Click on the next recommended story 4. Navigate
Depends on tests	T1,T2
Pass/Fail	Passed

Table G.5: System test case for adding a story to list

Test ID	T5
Test Item	Add story to list
Approach	The user navigate to a story and gives the story a rating. The user clicks the bookmark button in the story, and gives a name to a new list of stories.
Input data	User action: Click on a star, Give name to a new list of stories: ex. “My Favorite Stories”
Expected results	The story that was rated with stars are automatically put in the “read” list. The story should be stored in the new list of “My Favorite Stories” and the user have access to this by navigating to the menu, and then to “Bokmerker”.
Testing task	1. Click on a story in the recommendation view 2. Click on the star icon to in the right upper corner 3. Click on one of the stars in the rating view to give rate. 4. Click out from the view. 5. Click the bookmark button in the story 6. Click the plus icon 7. Type in a ”My Favorite Stories” and click out of the view. 8. Navigate to the lists via the menu 9. Click on ”My Favorite Stories” 10. Click on the first story in the list 11. Navigates to the “read” list 12. Click on the first story in the list
Depends on tests	NaN
Pass/Fail	Passed

Table G.6: System test case for giving rating

Test ID	T6
Test Item	Give a rating
Approach	The user gives a story a rating by clicking the stars. The user navigates to bookmark lists and checks if the story was stored in the “read” list
Input data	User action: clicks on story, navigated to rating view, clicks on a star.
Expected results	The star buttons that were pressed will change color. The system should store the rating for that story. The next time the user clicks on this story - the yellow stars will show the users rating.
Testing task	1. Click on a story in the recommendation view 2. Click on the star icon to in the right upper corner 3. Click on one of the stars in the rating view to give rate. 4. Click out from the view. 5. Click on the star icon again 6. Close rating view
Depends on tests	T1,T2
Pass/Fail	Passed

Table G.7: System test case for specifying settings

Test ID	T7
Test Item	Specify settings
Approach	The user navigates to Settings via the sidebar menu. The user then adds preferences and change the permission to use location.
Input data	User action: click the buttons for the age group, gender and interests.
Expected results	The user will click the buttons for age group, gender, and interests. The buttons will change color when clicked on. The user navigates to next view by clicking the next button and the system will give the response: ”Vennligst vent mens vi finner historier vi tror du vil like”. The system saves the users preferences and updates the recommended stories list. The list of stories should now be updated.
Testing task	1. Navigate to settings 2. Navigate to preferences 3. Add another preferences 4. Check the recommended stories list to see if it is updated
Depends on tests	NaN
Pass/Fail	Passed