



NTNU – Trondheim
Norwegian University of
Science and Technology

Bachelor's Project
Information Science

STEDR

SINTEF Storytelling

Hallvard Jore Christensen
Jon-Andre Brurberg
Jørgen Rugelsjøen Wikdahl
Tor Barstad
Vegard Storm
Øyvind Hellenes

May, 2014

Supervisor: Mohsen Anvaari

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

Sem Sælands vei 7-9
7491 Trondheim, Norway

Preface

This report is the documented result of the course "IT2901 - Informatikk prosjektarbeid II" in the spring of 2014. Our group consisted of six NTNU students from the Department of Computer and Information Science.

We would like to thank our contact person at SINTEF, Jacqueline Floch. Throughout this semester, she have been incredibly supportive and helpful in all aspects. She is truly a spirited person with an astounding dedication to this project. It has been a lot of fun working with her. Additionally, we would also like to thank our supervisor Moshen Anvaari for giving us great feedback and being a cool guy in general.

Trondheim, 29th of May, 2014

Øyvind Hellenes

Vegard Storm

Tor Barstad

Jørgen Rugelsjøen Wikdahl

Jon-Andre Brurberg

Hallvard Jore Christensen

Abstract

Through this project SINTEF wanted to further develop a multi-platform application for handheld devices called Stedr. The aim of the project was to add more important functionality to the application, increase usability and generally improving it in all aspects. In this report we will describe the whole process from preliminary work and planning to the process and then present the final product with test-results. The application is written in Titanium Studio and Play Framework using mainly XML, Java and JavaScript. The application makes use of many different APIs and frameworks to make use of existing services to reduce necessary maintenance. This was requested by the customer. During the course of the project we experienced and overcame many challenges, but in the end we reached all of our goals and ended up with a result we are very happy with.

Contents

	Page
1 Introduction	1
1.1 The subject: IT2901	1
1.2 About Stedr	1
1.3 Stakeholders	3
1.3.1 The team	3
1.3.2 Customer	4
1.3.3 Course Staff	4
1.4 Report Structure	5
2 Pre-study	6
2.1 Solution today	6
2.1.1 Existing functionality	6
2.1.2 Limitations	7
2.1.3 App evaluation	7
2.2 Survey	8
2.3 Tools and technologies	9
2.3.1 APIs	9
2.4 Similar products	10
3 Project organization	12
3.1 Responsibility Areas	12
3.2 Process model	12
3.3 Development Environment	13
3.4 Scrum Planning	14
3.5 Work Breakdown Structure	14
4 System Requirements Specification	16
4.1 Purpose	16
4.2 Intended audience and reading suggestions	16
4.3 References	16
4.4 Product perspective	16
4.5 User classes and characteristics	16
4.6 Product functions	17
4.7 Operating environment	18
4.8 Design and Implementation Constraints	18
4.9 User Documentation	19
4.10 Assumptions and Dependencies	19
4.11 System Features	19
4.12 Product Quality	30

4.12.1 Compatibility	30
4.12.2 Performance Efficiency	30
4.12.3 Reliability	31
4.12.4 Portability	32
5 Architecture	34
5.1 Back-end	34
5.2 Front-end	34
5.3 Process View	35
5.4 Use Case	35
5.5 Sequence	39
6 Implementation	41
6.1 Backlog	41
6.2 Sprints	45
7 Testing	50
7.1 Testing Procedure	50
7.2 Test Cases	50
7.3 Test Execution	59
7.3.1 System Testing	59
7.3.2 Acceptance Testing	59
7.3.3 NFR testing	63
8 Evaluation	69
8.1 Process	69
8.2 Project Management	69
8.3 Communication	70
8.4 Project planning	70
8.5 Problems and difficulty	71
8.6 Lesson learned	72
8.7 Technical evaluation and recommendations	73
8.8 Conclusion	73
9 Attachments	74
9.1 Class diagrams	74
9.2 Screenshots	78
Appendix A Developers Guide	80
A.1 Back-end	80
A.2 Front-end	87
Appendix B Test log	95

Appendix C Status Report Example	96
C.1 Status Report	96
C.2 Activity Plan	98
C.3 Risk Analysis	100
Appendix D Meeting Example	104
D.1 Group Meeting	104
D.2 Customer Meeting	107
D.3 Supervisor Meeting	109
Appendix E Other	111
E.1 Mock-ups	111

List of Figures

1	The Group Members	3
2	A simple overview of the architecture	6
3	KNappen screenshots	11
4	Work Breakdown Structure	15
5	A simple technical overview of the architecture	17
6	Deployment View	34
7	miniature class diagram: back-end	35
8	miniature class diagram: front-end	35
9	Process View	36
10	Map View (Home)	37
11	Menu View	37
12	Place Screen	38
13	Get Stories	39
14	Get Collections	40
15	Class diagram for front-end part 1	75
16	Class diagram for front-end part 2	76
17	Class diagram for back-end	77

List of Tables

1	The Team	3
2	Customer	4
3	Course Staff	4
4	System Feature: Find Place on Map	20
5	System Feature: Open Menu	21
6	System Feature: Search for a Location	22
7	System Feature: Refresh Map	23
8	System Feature: Go to Location	24
9	System Feature: Open Views	25
10	System Feature: Load Content	26
11	System Feature: Collection	27
12	System Feature: Upload Content	28
13	System Feature: Get Help and Info	29
14	Test Case: Get Places	51
15	Test Case: Open Menu	52
16	Test Case: Views	53
17	Test Case: Load Content	54
18	Test Case: Collection View	55
19	Test Case: Collect Map View	56
20	Test Case: Gallery	57
21	Test Case: Upload Content	58
22	System Testing: Validity	59
23	Compatibility: Interoperability	63
24	Performance Efficiency: Publishing New Content	64
25	Maturity Testing	66
26	Fault Tolerance Testing	67
27	Recoverability Testing	68
28	Portability Testing	68
29	Example Test Log: T-F7	95

1 | Introduction

This report is written as a bachelor project by computer science students at NTNU. The project revolves around upgrading and expanding features of a multi platform app called "Stedr" which is currently in beta. Stedr's purpose is to enable people to share their stories about places around the world. This can be anything from a famous attractions to just an ordinary building Trondheim. The contributors will be able to share stories and media through external services like Digitalt Fortalt, Flickr, Instagram and SoundCloud. With the application, users can view other peoples stories and images to help them explore a certain place.

1.1 The subject: IT2901

IT2901: prosjektarbeid i informatikk is the name of the course this project is a part of. Through this project the students will work in groups on a specific project within the scope of computer science. The institute will recommend tasks for the students to choose from. From here the students work self-reliantly under the supervision of employees at the institute. (*Text based on the information gathered from the study guide.*) The main purpose of the project is for the students to acquire practical experience with the software engineering process. Through working with a real customer in a the process, the students will gain valuable experience to prepare them for the working life.

1.2 About Stedr

The app "Stedr" was created in a collaboration between a group of Computer Science students from NTNU and Jacqueline Floch from SINTEF.

Stedr is an app that connects places and stories. It combines the formal history with the social media experience. The latter will also help to create a network effect. We see Stedr as the first step towards a national effort for documenting narratives of places. Take the statue of Olav Tryggvason for instance. Here you can write a story about Olav, the building process, or something on the debate about removing it. That being said, Stedr is made to experience - not for creating content. To create a story takes time. It requires finding sources, put together materials, editing etc. This is not something that can be performed easily on a smartphone.

Ideally we wanted to create a social network for cultural heritage - a kind of GoGoBot for cultural heritage - but this is a much more extensive project. The question is also who would drive such a platform? Kulrurådet offers a platform called Digitalt Fortalt for stories. This is unprecedented and no other countries in Europe offers something

like this. But unfortunately, Digitalt Fortalt has many limitations and it is a bit old fashioned. Europeana is an alternative that has support for user generated content and might thus be used in Stedr in the future.

The main goal of Stedr is to engage people more in the cultural heritage by

1. Giving them easy access to stories related to a cultural heritage.
2. Providing different narratives for people who have different interests like history, art, sports, music etc.
3. Utilizing network effects to increase awareness about places.

Only about half of Europe's population visited a cultural venue in 2010. This is not a satisfying statistic and hopefully in time, Stedr will help to improve this by making people aware of interesting places. A place in Stedr is a point of culture heritage. All places have several stories associated with them. That means there is a potential for providing various stories from each place.

To this date, there are many books related to locations, buildings and art in Trondheim, but several of these books are no longer available in book stores. It is also very inconvenient to walk around with books all the time. Is not it amazing that we do not have access to all this information on the mobile in 2013? Documentation and dissemination of culture in the countryside is of course challenging. There are countless places, many of which lie outside the responsibility of institutions. Associations and individual enthusiasts have helped to gather information and document the places, but the results are fragmented.

So the first step with Stedr is to create a great cultural user experience. To provide the opportunity to discover new places. The second step would be to engage people to tell about places around them.

1.3 Stakeholders

In this subsection we will present the main people involved with the project.

1.3.1 The team



Figure 1 – The Group Members

The team consists of six students all taking a Bachelor degree in Informatics at The Norwegian University of Science and Technology (NTNU). In our group we have a great variation in areas of expertise and knowledge which helped us greatly during the course of the project. Having expertise in many different areas we could help each other and share knowledge across the group to make everybody suited to different tasks. The importance of this project made the whole group very motivated to work for a good result. We are:

Hallvard Jore Christensen	hallvarc@stud.ntnu.no
Jon-André Brurberg	jonandbr@stud.ntnu.no
Jørgen Rugelsjøen Wikdahl	jorgenrw@stud.ntnu.no
Tor Barstad	torob@stud.ntnu.no
Vegard Storm	vegs@stud.ntnu.no
Øyvind Hellenes	oyvihell@stud.ntnu.no

1.3.2 Customer

Our customer is SINTEF (The Foundation for Scientific and Industrial Research). They are the largest independent research organisation in Scandinavia. The organisation was established at the Norwegian Institute of Technology (NTH) in Trondheim in 1950 and expanded rapidly in the following years.

Jaqueline Floc'h	<i>Project Manager</i> Jacqueline.Floch@sintef.no Our main contact at SINTEF and coordinator of the project from the customers side. She is also the primary driver for the app this project evolves around.
Babak Farshchian	<i>Interim Project Manager</i> Babak.Farshchian@sintef.no After some unfortunate events made Jacqueline unavailable in the very beginning, Babak took over for a period of time until Jacqueline could return as our main contact.

1.3.3 Course Staff

Mohsen Anvaari	<i>Supervisor</i> mohsena@idi.ntnu.no Supervises our group during the project, giving feedback and support.
Monica Divitini	<i>Course co-ordinator</i> divitini@idi.ntnu.no Is the course co-ordinator and in charge of the subject.

1.4 Report Structure

Chapter 1

The introduction chapter. Presenting the course, project and the different people involved in the project.

Chapter 2

Chapter describing the pre-study phase of the project. Since this project is based on an existing project with a prototype product, this phase was important for us. This process is documented in this chapter summing up all our research.

Chapter 3

This chapter describes the basis of our project with the main focus on the projects' structure. This includes how both the group and the project have been organized.

Chapter 4

This is the Software Requirements Specification (SRS) for the new version of "Stedr", and may be read both as a single chapter in this report or a stand-alone document. The SRS describes the behaviour of the system consisting of a detailed description with supplementation of diagrams and tables.

Chapter 5

In this chapter we are presenting the system focusing on the architectural part.

Chapter 6

Here you will be presented with the whole implementation process, containing all the sprint iterations.

Chapter 7

This chapter focuses on the testing phase of the project. Acceptance-, case- and non-functional requirements testing are among what is on the agenda in this section.

Chapter 8

A detailed evaluation of both planning, process and finally a conclusion.

Chapter 9

Attachments.

After these chapters the appendices follows, containing other important documents and charts.

2 | Pre-study

2.1 Solution today

2.1.1 Existing functionality

Since the application already is considered a working prototype, we will provide a list which gives a description for the functionality. Working functionality is in this report defined as the functionality that is implemented in the front-end or back-end. If something is implemented back-end it has to be used front-end. A more detailed technical description is found in the architecture-section.

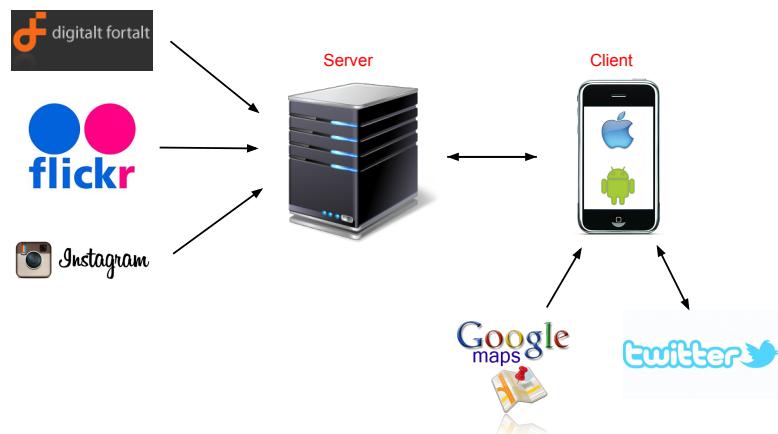


Figure 2 – A simple overview of the architecture

- Browse a map and zoom in and out.
- Load **places**.
- Click on a place in the map and access **stories** from Digitalt Fortalt.
- Get social media related to a place from the content providers Instagram and Twitter.
- Go to a users exact position on a map.

- Search for a location in the map.

2.1.2 Limitations

There are some limitations to the system that needs to be further developed, and some that probably would require total architectural review of the project to be fixed. Our task is to continue the development of the application. An overview of the features we are going to improve are discussed in the requirements section 4.

2.1.3 App evaluation

After the first meetings we concluded that the best way to proceed is to evaluate the existing system, to uncover potential issues and flaws. Therefore we decided that everyone should individually do a usability test when exploring the app for the first time. Here is a short summary of all our reports:

The first major issue most of us experienced is how slow the app loads, with no feedback that something is actually working in the background. Another thought that generally comes to mind is *I don't understand the apps function when opening it without prior knowledge*. What happens is that you get a map with tags you could click on, making you believe the function of the app is to use it when visiting a town (not Trondheim in particular) and want to explore historical monuments and get Wikipedia facts. When clicking on tags you get to the location-specific page, with content from Instagram. Some of us found this social feature not to have very obvious intentions. The social integration seemed out of place. You would want actual and useful information about places to be displayed first. One could also get confused when suddenly a lot of Instagram photos with random people suddenly appear. Our first impression is that this social aspect have to offer something more interesting to be relevant at this stage.

There is a nice touch with a slight gradient in the bottom indicating that you can scroll down for more content. When the phone is turned (switched to landscape mode) there are issues however. There is no scroll function here limiting the content available and the images are cropped. Also, if there is a lot of text added in the Instagram feed, it disappears along with the hash-tags. The way to collect images from Instagram to the application also seems to be sub optimal when irrelevant content are displayed. When tweeting there are no limitations on length, which causes problems when trying to post tweets over 140 characters.

These are some of the feedback extracted from the individual tests.

2.2 Survey

One of the most important research we did during the pre-study was to conduct interviews about Stedr. In these interviews we let six everyday people, use the current version of Stedr and paid close attention to how they used it. The subjects was both male and female ranging between 18-30 in age. The main purpose of this interview session was to get feedback on proposed features from our customer.

Firstly, after the test subject had some time to play with the app. We asked asked some questions about social media integrations, and this is the results:

Can you see yourself tweeting about a place from Stedr?

Yes: 0 No: 3 Don't know: 3

What about Instagram?

Yes: 0 No: 3 Don't know: 3

What about SoundCloud?

Yes: 0 No: 4 Don't know: 2

Additionally, we asked about Wikipedia. This was a proposition from us.

Would you like the app better if Wikipedia was integrated?

Yes: 6 No: 0 Don't know: 0

In retrospect, we see that even though people were positive to this, it doesn't really fit into what Stedr is about. Because of this, we abounded this feature late on.

When asked if they can envision using the app in the future, this was their responses:

- "Yes, if the app can also show patios."
- "Unfortunately, I don't use social media that much, but if the app was more historically oriented, I would be intrigued to use it."
- "Yes totally!"
- "It must be better than Google Maps for me to bother using it. I want the social features, but only for contributing, not for looking at what other people write. I would also use twitter with the app, but only if there was pre defined tweets."
- "Maybe, if I knew about it."

- "Yes, it sounds like a good idea."

The general consensus was very positive about Stedr. Everyone liked the idea, but some were sceptical in regards to the social features that was purposed.

In response to these results, our customer thought this survey was interesting, but it did not change her mind about trying out these features, in part because she had previously organized focus groups with other results. We respect this decision of course, but feel that it was right of us to do this research regardless.

2.3 Tools and technologies

2.3.1 APIs

There was a wish from the customer that we should use other existing services as much as possible instead of having our own database and a comprehensive back-end. The result of this is that the project will be dependent on many different APIs to function properly. As a result of this we spent a lot of time researching different APIs. The existing system had already Norvegiana, Flickr, Instagram and Twitter, though some of them needed a fresh up. In addition, our customer had ideas to expand functionality which meant more APIs. We did research on a lot of different APIs like Google Places, Wikipedia, NRK but did not end up using any of them after consulting with our customer.

Digitalt Fortalt and Norvegiana. After a evaluation in the pre-study phase we decided to switch the main API of application for better performance. Here is a short summary of why:

In current condition, Stedr uses an API called Norvegiana. This API is basically a collection of all public APIs that can be used in collecting different data. The major pros using Norvegiana is the portion of information accessible from different sources including *Statsarkivet*, *Digitalt Fortalt*, *Digitalt Museum* etc. There are many filtering options making Norvegiana able to return all the information needed.

There has been some complaints about how the current service works. Norvegiana is an interface to the dissemination system, not the production system. Which means that when a story is created, one has to wait for the Art Council Norway to perform an export from production to dissemination. This means that stories are not available through Norvegiana at once after they are produced. In some case we had to wait a few weeks. This is not acceptable with respect to our goal of increasing user participation and enhancing usability. We think this problem might be solved by switching completely to the Digitalt Fortalt API, which should not be too much trouble considering the existing code only makes simple calls to the API trough Norvegiana. It is worth

noting that Digitalt Fortalt is originally meant for Norwegian cultural heritage, which might cause problems in a potential international expansion.

The functionality in the APIs, for our usage, are about the same. Where both come short is that both strictly speaking work like databases that you only can retrieve information from, making them impossible to use for posting new content. This has to be solved in another way.

2.4 Similar products

As a part of this project we have reviewed apps that come close to our own in regards to purpose, target groups and functionality. One of the apps we looked at is Trondheim Guide.

One of the best features of Trondheim Guide is virtual reality, where you can look into the screen and see what you have in front of you. This can for instance be used to see the distance to the nearest cultural heritage or attractions. You can also have your own journal, where you can add places you have been to. This is tracked with time and date. There is also a very nice feature where you can make your own postcard, and share it with friends. For each place you visit you can check in with both comments and picture, this is saved in your log as well. Although Trondheim Guide is first and foremost an tourist app, it can also be used to discover various cultural heritages around Trondheim.

Another app that overlaps a lot with ours is KNappen. KNappen is an aesthetically pleasing, well-designed app with a user-friendly GUI. Like our own app it is focused around places. You will find images, routes, augmented reality, audio, video and informative and historical texts related to places. Users can also see the Wikipedia article about the place and they can see related places. They have a map view that is somewhat similar the one in our app, where icons are used to represent the different places.

We think Stedr could benefit from implementing some of the features from these two apps. Especially virtual reality from Trondheim Guide would be a great addition to the app. The user experience from KNappen is also something Stedr could take some inspiration from



Figure 3 – KNappen screenshots

3 | Project organization

3.1 Responsibility Areas

Good delegation of responsibilities helps so that someone at all times have an overview over what tasks needs to be done in specific areas. This also makes it easier to estimate workloads and delegate tasks during the group meetings. It is important to note that even though there are specific responsibility areas, all the group members will be able to get practical experience in all of the project areas. Time spent in different areas will be distributed individually according to the responsibility areas.

Øyvind Hellenes Scrum master Øyvind was selected as the scrum master because of his leadership qualities and because he early on took an interest in the organizational part of the project.

Jon-André Brurberg Project leader Jon-André is the driving force in this team and hence, he is also the project leader. Additionally, Jon Andre showed interest in the documentation so he is also responsible for this.

Tor Økland Barstad Technical coordinator Tor, with his competence and knowledge of programming, is our technical coordinator and will thus oversee the code and functions as a technical supervisor for the group.

Jørgen Rugelsjøen Wikedahl Testing manager Jørgen is responsible for testing the application to make sure the application has as few bugs as possible.

Hallvard Jore Christensen Report coordinator Hallvard have the main responsibility of managing the report. This is because of his experience with L^AT_EX and report documentation in general.

Vegard Storm Usability manager Vegard showed interest in making sure the application is as user friendly as possible and will manage that aspect of the project.

3.2 Process model

For the process model we chose to use the Scrum framework. This was the most natural choice for us among the agile methods since it is a system we all have experience with through previous projects. There are many advantages working with Scrum. It gives clear priority for features and deadlines, which will allow us to focus more of our energy on other vital tasks. This approach promotes communication and transparency. All the team members always knows what is going on through the product backlog. With the backlog cards, the whole production team is also involved with the overall time estimate, which makes it fairly accurate and controllable.

We considered a few other methods as well, like Kanban and XP, but came to the conclusion that Scrum was the system for us. This was due to the many structured rules of Scrum which brings order, but still allows us some freedom during the development of the project.

With Scrum, we will work in iterations called “Sprints” which typically have a duration of a week or two. We also strive towards making these sprints incremental. By doing this the model is designed, implemented and tested incrementally, feature by feature, until the project is finished. The advantage here is that for every sprint we have a working product to show for, which is a good reference to have, both for ourselves and the customer.

Since we already have a working prototype from the very beginning for us to further develop, there is some obvious phase partitions. The first consists mainly of assessing the current version of the product and defining the path ahead before we start the actual programming. This will be done through thorough dialogue and discussion with SINTERTEF, to give us a unison idea of where the product is headed. User evaluation is also important in this phase, both internally and externally within the target user group. And of course technology and framework selection. After this comprehensive planning, the actual coding phase can begin. The sprints will be a big part in this, and since we are working incrementally, so will the testing. .

Prototypes through a digital mock-up will be important in the planning phase. We have chosen to use Balsamiq for this, which will mean we will have an interactive prototype mock-up to show the customer. This makes it easier to have something concrete/“physical” as a reference.

3.3 Development Environment

Since our project is based on further developing on an existing product, there is an advantage in using the same main framework as the previous developers. We decided to use Titanium to easily develop a multi-platform app, but we also took a close look at other options (like PhoneGap) and compared them meticulously in their most critical aspects. With the Titanium framework we use the Titanium SDK which is based on eclipse but tailored for mobile development. For sharing code, Git was our system of choice, mainly because we were already familiar with it, and know it has all the functionality we could need throughout the project. Other documents and files, like notes, summaries etc. we decided to share through a dedicated Google Drive and Dropbox folder, because each has its own advantages in different aspects. As SCRUM service we first choose Agilefant, but later decided to just use spreadsheets instead.

This project obviously involves working with a big set of APIs, like social media, dictionary and other media services. These will play a great part of the development and introduce other frameworks we will have to account for. For communication we often

use mail and chat-services, but we prefer more “personal” forms of communication like a video chat through Skype, phone calls and/or ideally, meetings in person.

3.4 Scrum Planning

In our version of Scrum, we focused primarily on three key points. The first one is process evaluation. In meetings, we always discuss and evaluate what we have done since last time. This is vital for both ensuring good process quality and team motivation. We focused on writing good summaries from meetings such that we later can go back and learn from what we have done.

The second point is product improvement. This concerns both design and code. We believe that in having a constant dialogue, where we talk about what direction we should take the app, we can maximize the potential of the project. The customer is of course highly involved in this part. Each week we meet with our customer to discuss function, and design choices. In these meetings we can suggest new functions and features to our customer.

Last but not least is testing. We did a lot of planning to make sure both our functional and non-functional requirements would be up to date with our customers vision. Writing and documenting usability and acceptance tests helps us making sure the product quality is satisfactory. Additionally we planned and performed user tests to identify user needs.

3.5 Work Breakdown Structure

This Work Breakdown Structure is an overview of what we have done and how our work is distributed between packages. It is updated to match our end result. We have given percentages to each task to show how much work have been put into it.

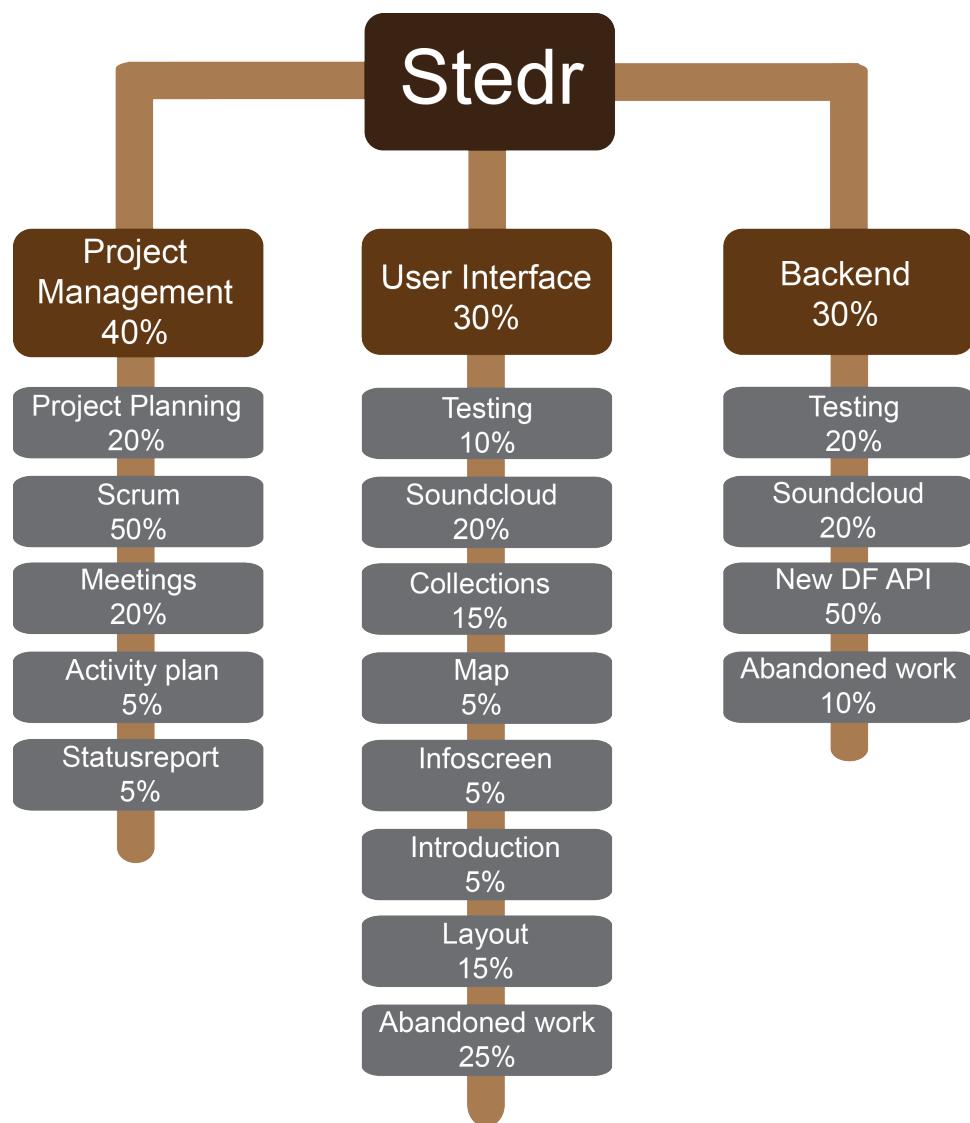


Figure 4 – Work Breakdown Structure

4 | System Requirements Specification

4.1 Purpose

This is the software requirement specification for the new version of Stedr. This concerns both the back-end system and also the front-end that shows the content to the user. Here the traditional architectural terms back-end - and front-end are used, but there are some subtleties to this term, as the front-end itself is managing a content service of its own.

4.2 Intended audience and reading suggestions

Intended readers for this document are current and future developers, and the customer. The reader should also be noted that the SRS both can be read as a stand-alone document to get an overview of the rationalization behind the development process, but that it also is a part of the project report as a whole.

4.3 References

The software requirements are based on the standard as provided by ISO/IEC:25010 **25010** and also the models that can be found in this report's section for architecture and modelling. References to the ISO-standard and other literature are found at the end of the project report under references.

4.4 Product perspective

Originally Stedr is a product developed by students at NTNU as a part of the subject TDT4290, and this application will form a basis for our continued development. The state of the existing application is considered to be a working prototype, and to some degree it is an application that is built up with a traditional server-client architecture. A simple technical overview of the system is provided below.

4.5 User classes and characteristics

The users of the program is mainly divide into two categories. One of these groups is the primary user group which are interacting with the smartphone application, front-end and to-see-content. A typical primary user is an high school student which is introduced

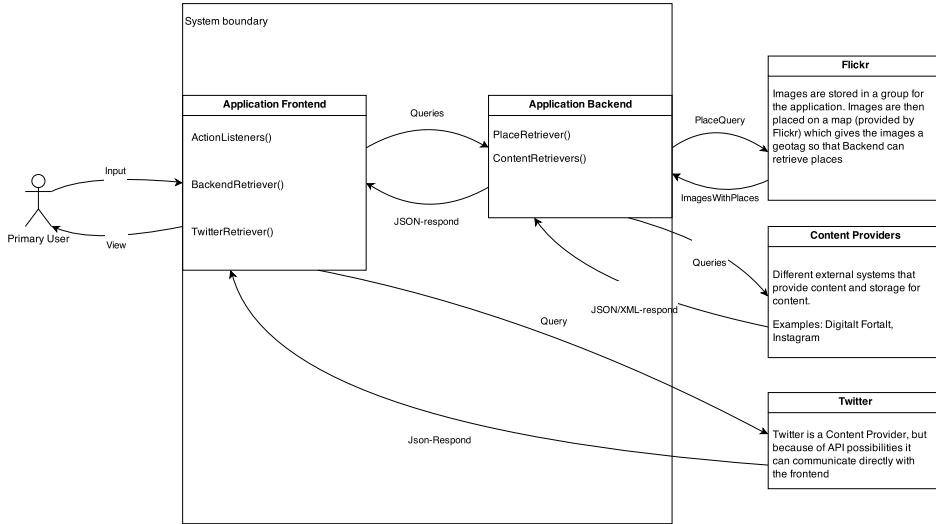


Figure 5 – A simple technical overview of the architecture

to the program in the context of cultural heritage awareness. As mentioned earlier, the broad goal of the application is to make people aware about cultural heritage. If that goal is fulfilled, some primary users of the application will hopefully transit over to become a content provider.

A content provider has the possibility to interact with the system directly, but also he or she can choose to interact with the system more indirectly. This varying degree of interaction will hopefully lower the threshold for users transiting from content consumer to content provider, which is the overall goal of the application.

Another secondary user is the maintainer-administrator. The maintainer-administrator will use a special set of tools to approve creation of the systems places, but these tools are provided by the external system Flickr. Our internal system is communicating with the external system so that applications relevant with information is sent from the external to the internal system. In the end the external system is stand-alone and can not be controlled directly from the internal system.

4.6 Product functions

The main features of the program for different user categories are presented as a bullet list below. All of the provided functions by the system are available to every user without the need of registration or approval by the systems maintainer-administrator. The only exception is related to adding new places. In addition the user may need to register on external systems to make use of those features.

Primary user

- See places on a world map
- Navigate the map
- Select places on the map, and look at stories to the related place.
- Select places on the map, and look at pictures to the related place.
- Select places on the map, and find sounds to the related place.
- Select places related to pre-defined themes (i.e: art in Trondheim).
- Make posts to a social medium.

Secondary user, content provider

- Create stories and pinpoint them so that they appear in relation to a place.
- Create sound and pinpoint them so that they appear in relation to a place.
- Create pictures and pinpoint them so that they appear in relation to a place

Secondary user, maintainer-administrator

- Approve content providers so they can create places.

4.7 Operating environment

The front-end application of the system is a smartphone application which aims to run on the two major smartphone platforms Android (2.2 and above) and iOS. Because of difficulties with developing towards the iOS platform without equipment from Apple, the goal is to get the application to run on a unspecific versions of iOS. This is to make sure that the full implementation of the application will also run on iOS. The back-end of the system, or server, should run as a cloud-based platform provided by Heroku, as the case was for the existing version of the server. Since that service now is unavailable the new version of the back-end will run as a new service and thus replacing the former one.

4.8 Design and Implementation Constraints

Because of the nature of the project as a part of a course, there will be few constraints regarding the development of the system. Because there already exists functionality it's natural to constraint the system to make use of the existing code and technologies. Reimplementing them with new code or technologies are allowed, but since time is a limited resource (approximately 20 hours per member per week) it is important to be

time effective. That time effectivity and available workload is also to be seen as a design and implementation constraint. Apart from a private GitHub account, the project is to be done without funding so for the deploying a free service has to be chosen.

Since the customer is a professional organisation, it is also important that the system behaves correctly according to licensing. The system itself is to be open source under the Berkley Software Distribution license version 3 (BSD-3). It is also important that the system handles licensing from external systems correctly, and only shares legal content.

4.9 User Documentation

Documentation to system users will be provided in the application itself. This documentation has to easily be editable by the maintainer-administrator. In addition to the user documentation there will be provided documentation for developers as an appendix in this report, and code documentation in the code itself and in the GitHub-repository.

4.10 Assumptions and Dependencies

An important assumption in the development of the system is that the former system delivers the functionality which is stated in the feature list given by the customer. A copy of this list can be found in the appendix.

Another important assumption, is that the external systems that were implemented in the earlier systems still is functional. This is also a dependency, because changes in those external systems will make internal system malfunctioned. This can be seen as a large drawback in the system, but as the back-end is to be kept to a minimal, external sources have to be used for content storage and content providing.

4.11 System Features

SF-1	
Name	Find place on map
Priority	H
Goal	To browse the map to find a given place
Actors	Primary User
Preconditions	<ul style="list-style-type: none"> 1. The home screen is displays 2. The internal system and external systems are running 3. The device has a internet connection
Stimulus-Response	<ul style="list-style-type: none"> 1. The home screen is displays 2. The internal system and external systems are running 3. The device has a internet connection
Alternate Flow	2a The place does not exist and is not shown on the map
Functional Requirement	A user should be able to access and browse a map, with places as pinpoints at their respective geographical location. The pinpoints should contain the picture and information found on Flickr. Group places close to each other in one icon on map.
Related Use Cases	1,3
Dependencies	none

Table 4 – System Feature: Find Place on Map

SF-2	
Name	Open menu
Priority	H
Goal	Open the drawer menu
Actors	Primary User
Preconditions	1. 2,3 4 A screen with the menu button
Stimulus-Response	1. The user clicks the menu button 2. The menu opens
Alternate Flow	1a The user clicks the menu button, and the menu is already open 2a The menu closes
Functional Requirement	A button with the possibility to open the menu should always be presented to the user, so that the user easily can navigate the application.
Related Use Cases	1,2
Dependencies	none

Table 5 – System Feature: Open Menu

SF-3	
Name	Search for a location
Priority	M
Goal	Go to a location on the map
Actors	Primary User
Preconditions	1. 1,2,3
Stimulus-Response	1. The user searches for a location with the search bar in the map view. 2. The map navigates to the location
Alternate Flow	2a Location is not found and is not navigated to.
Functional Requirement	A search bar related to the map should be presented to the user, so the user can search for locations (independent of places) to see if there are any stories at that place.
Related Use Cases	1
Dependencies	none

Table 6 – System Feature: Search for a Location

SF-4	
Name	Refresh map
Priority	H
Goal	Update the map with content.
Actors	Primary User
Preconditions	1. 1,2,3
Stimulus-Response	1. The user clicks the update button. 2. The map refreshes and show new places
Alternate Flow	2a No new places are found, so no places are added to the map.
Functional Requirement	The user should be presented with a button that makes requests for new places with content when pushed. This function should also be done automatically so that new content is sent to the user within 5 minutes after it is added.
Related Use Cases	1
Dependencies	none

Table 7 – System Feature: Refresh Map

SF-5	
Name	Go to location
Priority	H
Goal	Go to users location.
Actors	Primary User
Preconditions	1. 1,2,3
Stimulus-Response	1. The user clicks the GPS button. 2. The map zooms to the users location.
Alternate Flow	2a GPS not available so it can not go to the users location.
Functional Requirement	Since the user has the possibility to navigate the map freely, it should also be possible to quickly navigate to places relevant (in context of location) to him/her.
Related Use Cases	1
Dependencies	none

Table 8 – System Feature: Go to Location

SF-6	
Name	Open views
Priority	H
Goal	Open views and see the content related to that specific view
Actors	Primary User
Preconditions	1. 1,2,3
Stimulus-Response	1. The user clicks on a view 2. The user changes views at will 3. Content
Alternate Flow	1a If the user clicks a button for the already chosen view, nothing should happen.
Functional Requirement	For navigation in the place view, the user should be presented with different buttons (or tabs) so that the user easily can navigate between content and still have an overview of what types of content the application provides. Preview picture gallery when places are grouped together. Add description about place, own tab for sound. Be able to show place location on map from story. Be able to filter stories by tag, author, institution video/no video. preview stories by sound from SoundCloud
Related Use Cases	3
Dependencies	none

Table 9 – System Feature: Open Views

SF-7	
Name	Load content
Priority	H
Goal	Content is loaded from the external systems
Actors	Internal System
Preconditions	1. 1,2,3
Stimulus-Response	<ul style="list-style-type: none"> 1. Access the server as done in the previous version of the system 2. Provide input to the server "placeId=" 3. Content is loaded and a JSON-object is replied by the server
Alternate Flow	<p>1a If the user clicks a button for the already chosen view, nothing should happen.</p>
Functional Requirement	The API for DF has to be changed, without changing the behaviour of the response from the server. In addition to this the server will respond with a new container for the audio content. Other content should be handled as normal. Retrieve collection from DF based on hash-tag and location. Retrieve stories in a collection from DF based on tags. Open info retrieved from SoundCloud based on hash-tags or location. Retrieve information from Instagram based on Hash-tags. Be able to get tinyUrls to different content.
Related Use Cases	Null
Dependencies	none

Table 10 – System Feature: Load Content

SF-8	
Name	Collection
Priority	H
Goal	Get all places related to a theme.
Actors	Primary User
Preconditions	1. 1,2,3
Stimulus-Response	<ul style="list-style-type: none"> 1. Access the menu bar. 2. Click on the Collections-button 3. Choose a collection 4. Collections view is opened 5. Change to map view 6. Places related to the collections is shown on map
Alternate Flow	3a No Collections are available
Functional Requirement	A container called Collections are to be implemented. Collections. Allow switching between map-related and collection related functionality. Display picture, title and description about a collection. Have a storyListView. Preview stories in collection story list. Open story in collection list. Places on map view with icon for each story in collection. Preview a place for story on map.
Related Use Cases	3
Dependencies	none

Table 11 – System Feature: Collection

SF-9	
Name	Upload content
Priority	M
Goal	Upload content
Actors	Primary User
Preconditions	<ul style="list-style-type: none"> 1. 1,2,3 5 The user has an account at the content provider he or she is trying to upload to. 6 Places related to the collections is shown on map
Stimulus-Response	<ul style="list-style-type: none"> 1. Access the tabs for different views 2. Click the add-button in the views.
Alternate Flow	3a No Collections are available
Functional Requirement	The user should have the possibility to add content so that. Add picture directly from stedr. ask the user for login-credentials the first time, then store locally for continued access. A similar approach for SoundCloud. Have relevant hash-tags copied to clipboard. Be able to comment and like pictures on Instagram.
Related Use Cases	Null
Dependencies	none

Table 12 – System Feature: Upload Content

SF-10	
Name	Get help and info
Priority	H
Goal	Be informed
Actors	Primary User
Preconditions	1. 1,2,3
Stimulus-Response	<ul style="list-style-type: none"> 1. Access the drawer menu 2. Click the help button. 3. Select the option for what help you need
Alternate Flow	
Functional Requirement	Introduction for first users. Help available at any time.
Related Use Cases	Null
Dependencies	none

Table 13 – System Feature: Get Help and Info

4.12 Product Quality

Guided by ISO:25010, meetings with our supervisor and the feature list given to us by the customer the product qualities that are important for the project is compatibility, performance efficiency, reliability and portability.

4.12.1 Compatibility

There are a lot of information processing done by the back-end, and the information is derived from multiple sources and in different formats. It is therefore important that the back-end processes this information into a standardized format. Since there already exists code, new additions should be compatible with the old code.

Co-existence Since the architecture behind the system relies on a clients (V) and a server (MC), it has to be possible for the clients to share the resources from the server. This means that several instances of clients should be able to be functional at the same time.

The test for co-existence will be accessing the server at the same time with multiple clients. This can be done in combination with the test for maturity under Reliability.

Interoperability All information sent from the server should follow the same format (JSON) so that the front-end only will have to communicate with the back-end using the same format. This means that the back-end will have to convert XML from some of the external systems into JSON. With this approach the front-end will also remain separated from the back-end, which will make it easier to exchange parts of the GUI later on as it is

For valid interoperability it is necessary that all response from the back-end to the front-end is in the JSON-format.

4.12.2 Performance Efficiency

Even though the system isn't a part of a critical operation, the new and improved system will have performance efficiency as an important model of quality. The reasoning behind this is that the customer wanted to give users an experience of a more responsive system. In addition to this the server has to utilize minimal resources, as it is going to be ran in a resource restricted environment.

Time behaviour As of now the time to load new content from the content providers to the application is slow and random. Because of this there are no exact estimation on the time used to pull content from Digitalt Fortalt and Instagram, but the application should use no more than *300 seconds* to pull new content. This fairly high time frame is due to processing times at the external APIs. 300 seconds is therefore the time limit for the system to have pulled new stories/pictures from Digitalt Fortalt/Instagram. The time limitations set for the back-end to respond to requests is set to *120 seconds* for places (with an empty cache) and *15 seconds* for places (with a full cache), stories, images and sounds. Unrelated to the goal of performance issue; the user should be informed when the application is processing information.

Resource utilization Requirements related to resources utilized by the application when performing it's tasks, are already met by the prototype. The new version of the application are bound also by these goals, which means that new functions must maintain minimal usage. Specifically the back-end is bound by the resources provided by the free service of the Heroku Cloud Platform. One of the implications of this is the cache that cannot be written to disk as the file-system on the disk is read-only. Without writing capabilities or database access, the back-end will have problems improving the time behaviour for place-loading when the cache is empty.

Capacity Regarding capacity used by the the application, there should be an improvement. Because the application is to be used on the go where there may not be any WiFi-hotspots, the application should restrain itself to download content that is unrelated to where the user is. Because of the varied content types, it is hard to set a defined limit in how much content (in terms of megabytes) the application should download. The limitations given to the application will therefore be set by the equation:

$$\text{Bound} = \text{Content from Digitalt Fortalt} + 5 \times \text{Content from Twitter} + 10 \times \text{Picture from Flickr} + 5 \times \text{Picture from Instagram} \quad (1)$$

4.12.3 Reliability

Since the application is going to be online without a team responsible for the technical maintenance, the server should be operative as long as the external content providers are feeding it with content.

Maturity Because of the early versioning of the application, the aspect of maturity is not important for this application. Users of the application are few, and they know what

the capabilities of the application is. This means that a user follows a rigid pattern and within that pattern, the probability to execute faults is almost non-existing. Functionality outside that pattern is not supported and thereby it's impossible to execute mistakes.

Testing of this should be done by simulating a high number of calls towards the server. The simulations should be done as threads as this will better simulate a real-life situation which is not done sequentially.

Availability An important characteristic of the application is that it has to be available just as often as a professional service. This means that under normal circumstances, the uptime of the back-end and front should be 90 % or above.

Testing of this is done over at least a week while monitoring the uptime of the server.

$$1 - \frac{\text{Downtime in hrs}}{\text{Days running} \times 24\text{hours}} \times 100 = \text{uptime\%}$$

Fault tolerance Whenever faults are occurring, it is crucial that the back-end has implemented services so that it can recover without the need of a maintainer. Because of the relative simplicity of the back-end, the server should restart itself within *180 seconds*.

Testing for this should be done by deactivating components and then monitor if a system is to be considered functional. A functional system is a system which serves places and stories on a map.

Recoverability Since most of the system will be ran without a maintainer, the system should recover quickly on faults by its own.

Recoverability-testing is done together with testing the fault tolerance, by monitoring the time the system uses to become operational after critical components are disabled and reactivated. Because of the existing architecture it is already known that the system will fail when certain components are deactivated so a fail-proof system is not theoretically possible.

4.12.4 Portability

One of the main reasons for using the Titanium-framework is that it is possible to code and compile it down to native applications for a number of smartphone environments. Because of iOS developer requirements, the former application has unfortunately never been tested on iOS, which is a goal set by the customer.

Adaptability One main constraint for obtaining a high grade of adaptability is to not use platform specific GUI elements front-end. This should lead to an identical codebase

front-end, which can compile down to all the Titanium-supported platforms. With this method it is also possible to ensure some degree of compatibility against new platform versions, since the mobile platforms usually have support for older platforms. Of course this is not satisfied by the system, but by the developers of a given mobile platform. Therefore this support is not guaranteed.

Installability Since the system is supposed to be a native application, it should be possible to install the application from platform-specific repositories (i.e: Play and iOS-store). Since the program is still under development this won't be accomplished though, and the requirement is to deliver a file which can be installed on phones meant for development (debugging enabled).

Replaceability As above, the system should update through a platform specific service. Since the system isn't a part of the official repositories yet, replacement of the application should also be through debugging enabled devices.

Note: Since the program builds down to native applications under 15MB, it should be a fairly small job to make the program available on the different platform-specific repositories.

5 | Architecture

The current architecture of the application is as shown in figure 3.5. We have a back-end written in Java that retrieves information from services like Digitalt Fortalt, Flickr and Instagram. Digitalt Fortalt is where all the stories are obtained from. Flickr holds all the locations, and the pictures are taken from Instagram based on tags. The information is stored on the server and can now be used by the client, which holds the front-end of the application that is being developed on Appcelerator Titanium, using mainly JavaScript and XML. Twitter is integrated directly into the front-end and does not have to go through the server. In addition, we have also added support for Soundcloud.

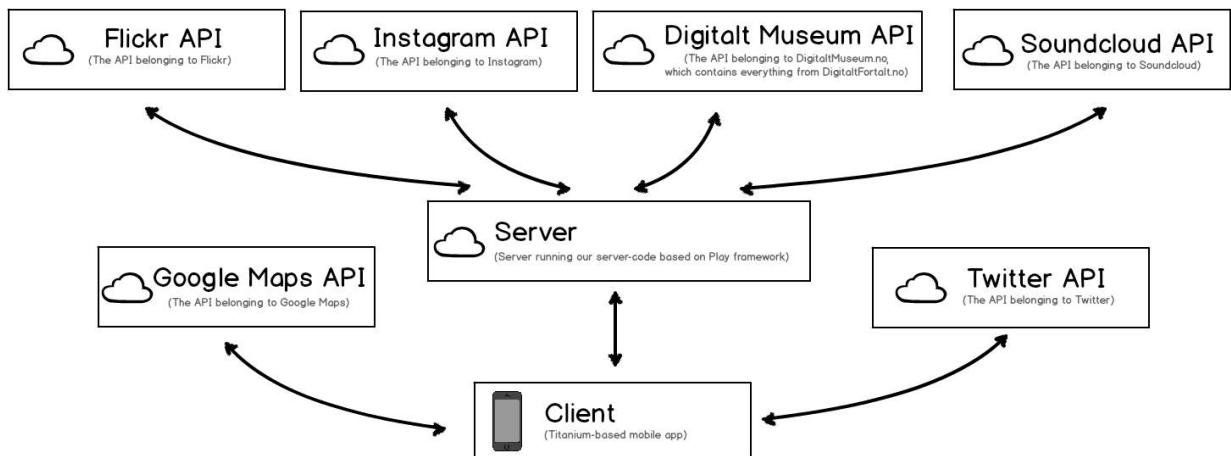


Figure 6 – Deployment View

5.1 Back-end

The Back-end is written in Java and mainly retrieves data from external APIs and save it on the server so that it can be used by the application.

5.2 Front-end

The front-end of the application is an interface to let the user enter, manipulate and view data. It is the part of the application that is being interpreted on the users own device, and is based on XML, TSS and JavaScript for design and functionality.

Every window in the application has a JavaScript-, TSS- and XML-file associated with it. A window can contain various views that can each have different event listeners.

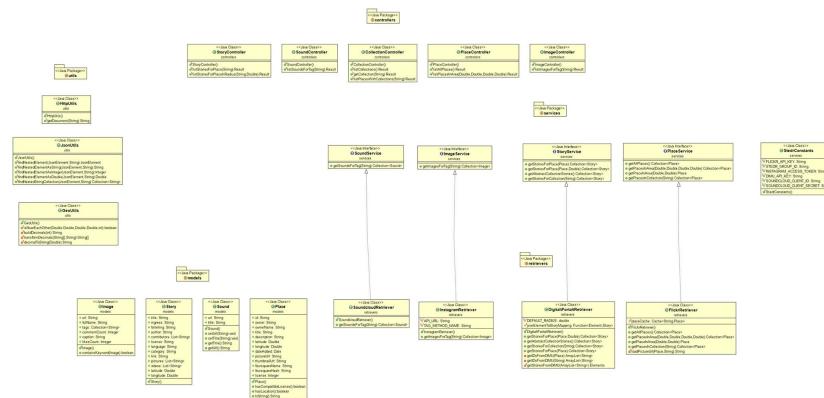


Figure 7 – Full scale class diagram can be found in Attachments 9.1

What the user sees depends on the window currently open and its associated XML, TSS and JavaScript files. What happens when interacting with a view depends on the event-listeners attached to that particular view. Interactions can be purely visual or it can trigger core functionalities. For example the refresh button on the map window has an event-listener attached to it so that when the user clicks it, it will attempt to fetch the locations from the server and plot them on the map. It will also animate the refresh icon to spin, giving the user feedback when the click was registered.

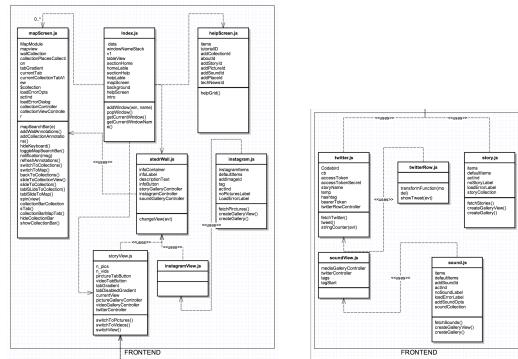


Figure 8 – Full scale class diagram can be found in Attachments 9.1

5.3 Process View

5.4 Use Case

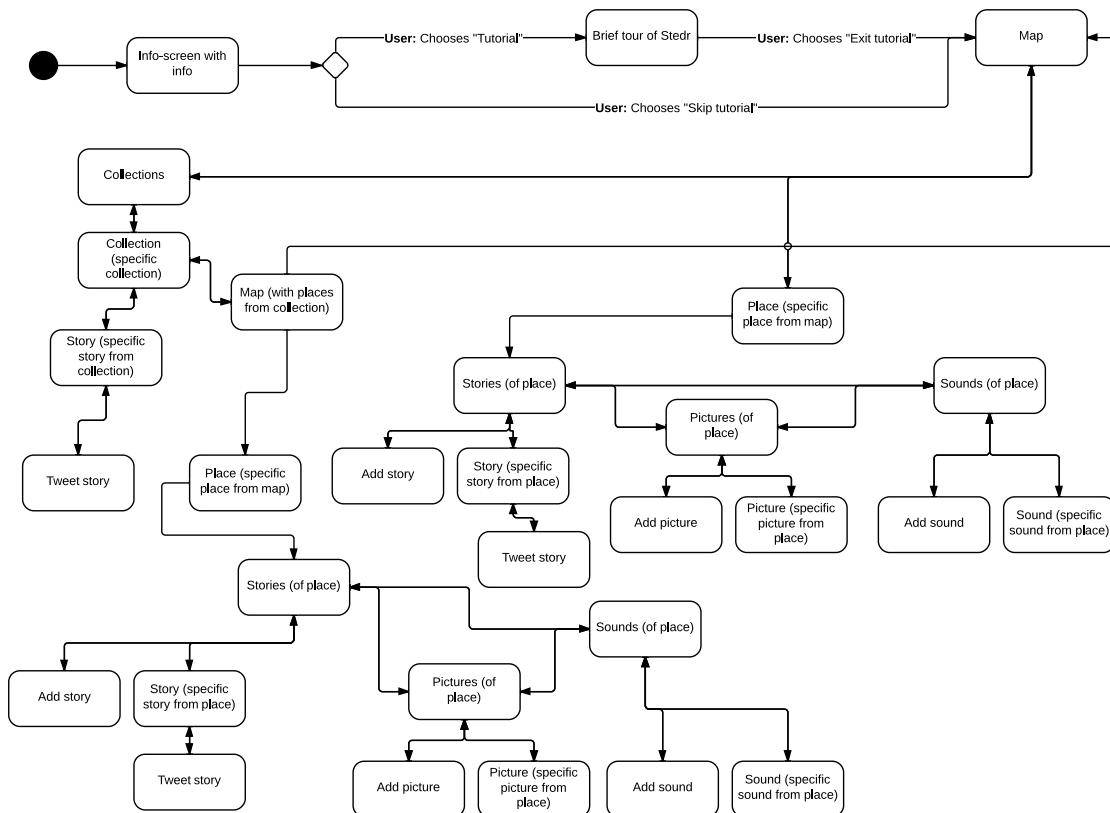


Figure 9 – Process View

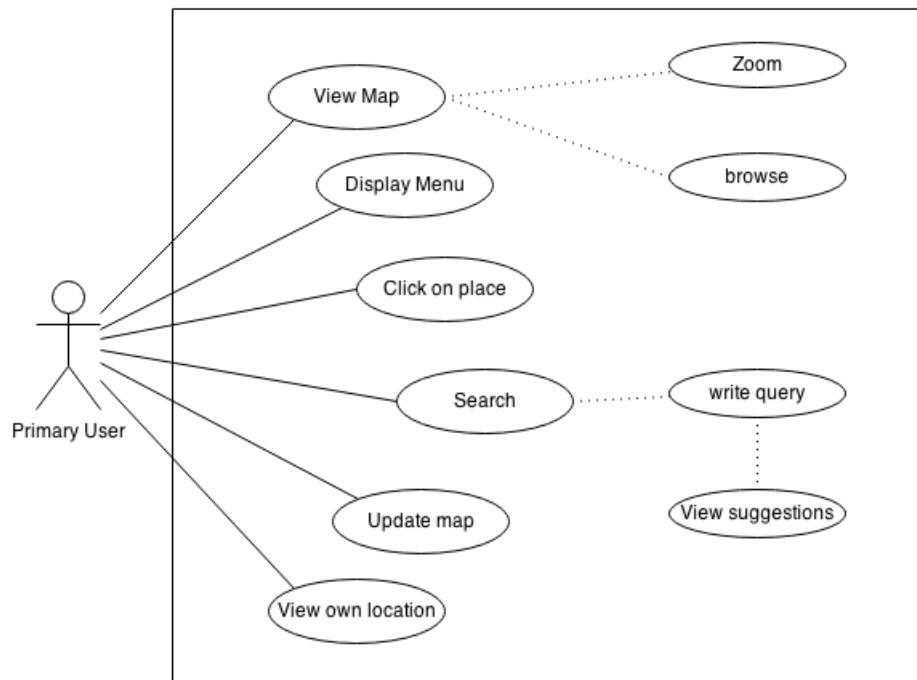


Figure 10 – Map View (Home)

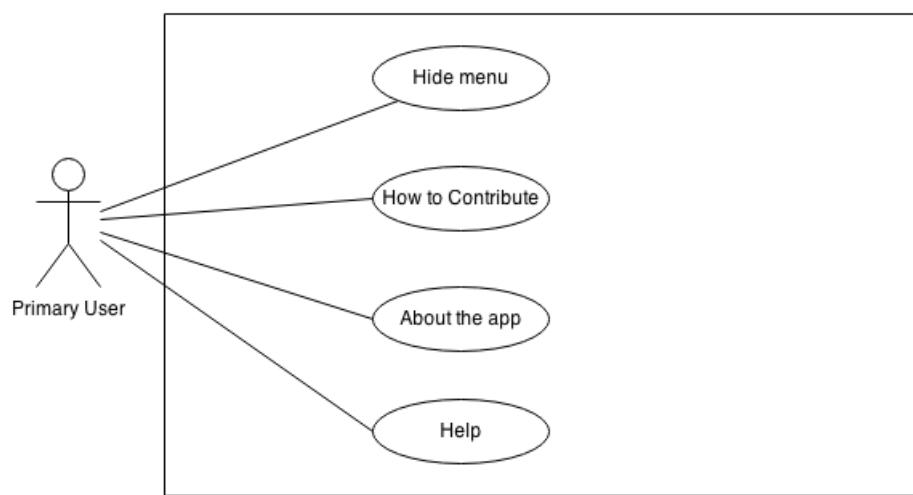


Figure 11 – Menu View

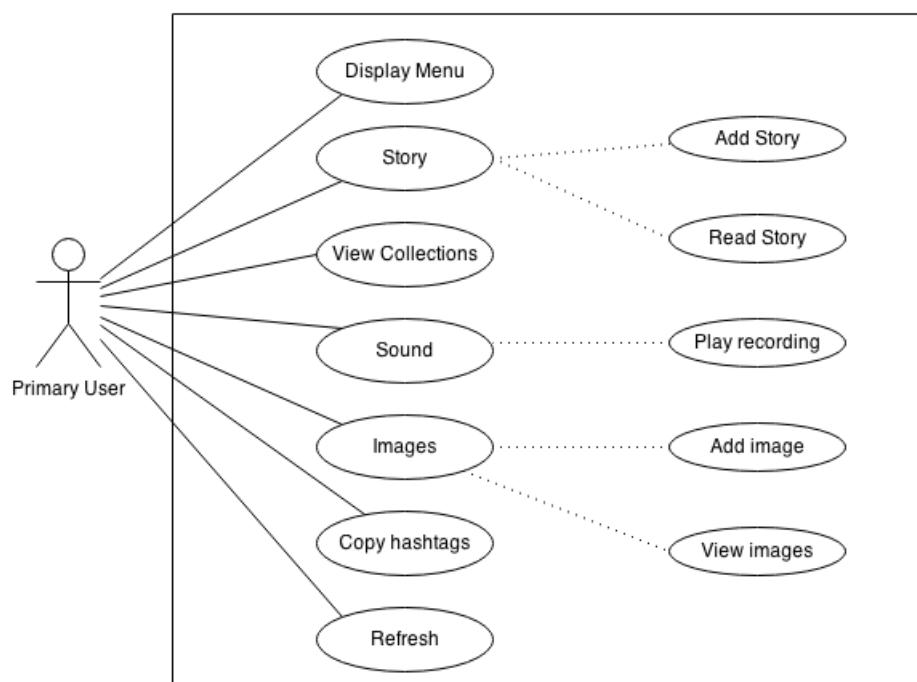


Figure 12 – Place Screen

5.5 Sequence

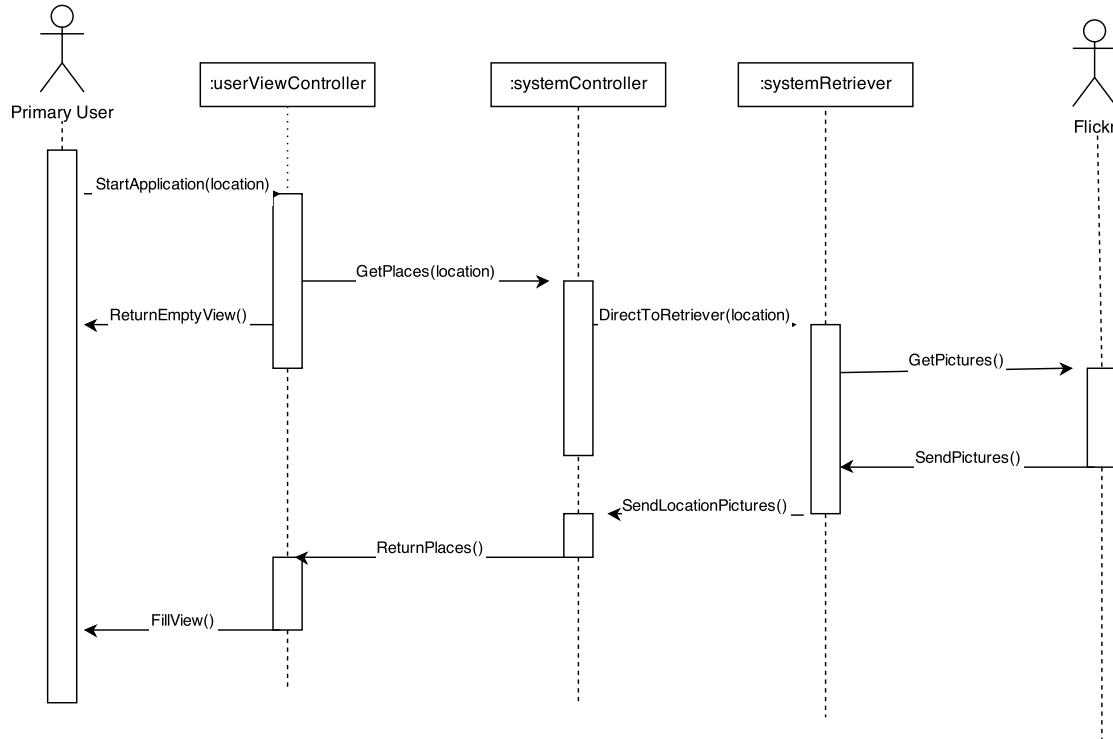


Figure 13 – Get Stories

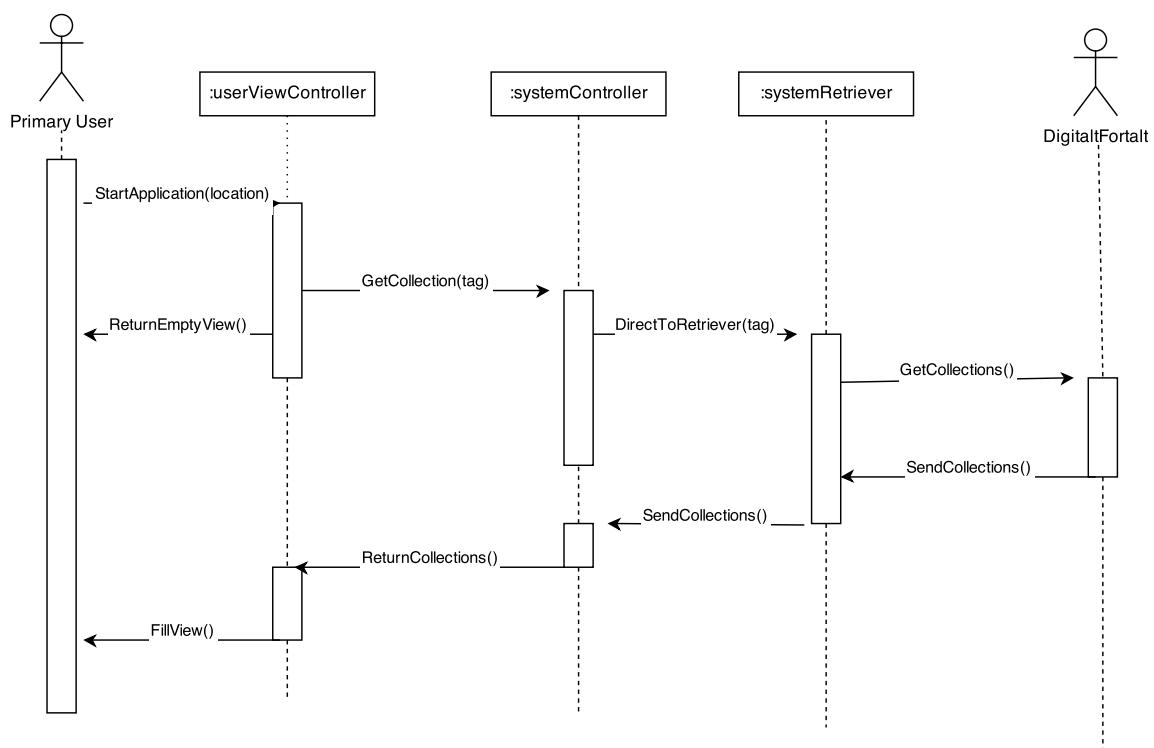


Figure 14 – Get Collections

6 | Implementation

6.1 Backlog

Our backlog changed a lot during the project. About halfway in, we was introduced to many new requirements from our customer. We received a list where she had written down all her requirements for the project and prioritized them. This list was used during the acceptance testing and can be found in section 7.3 ordered by priority. This introduction also altered the priorities a bit, but was helpful as it made our customers vision clearer to us.

ID	Tempo	As	I want...	Such as...	Notes	Priority
1	UI	User	To browse map, zoom in and out.	Watch where there are events of interest.		1
2	UI	User	Click on event in map	users can view stories and pictures for that specific event		1
3	UI	User	Watch pictures of the place	To identify it. And get another view on it.		1
4	UI	User	To have an easy-to-use menu		Slider	1
5	UI	Developer	Let the user's have an easy user interface	user experience is the best it can be.		1
6	UI	User	be able to switch between map-related and collection-related functionality	A new View or activity		1
7	UI	User	view picture/video gallery from DF		Make clear that several pictures are available	1
8	UI	User	read text from DF		keep url, remove tags, show institution (aditionally to author)	1
9	UI	User	Share my experiences and knowledge about a place	To give something back to the community		2
11	UI	User	Search for stories	read and learn		2
12	UI	User	Get my exact point where i am, in map.	I can know which attractions are near		2
13	UI	User	Share a place on social media	To recomend to my friends		2
14	UI	User	Read a description about a place		Retrieve description from flickr, add under picture.	2
15	UI	User	Have a own tab for sounds			2
16	UI	User	record stories for the app.	SoundCloud integration	Be able to add sounds, preview stories from soundcloud and play sounds in the app.	2
17	UI	User	Read other peoples experiences of a sight.	collect and compare experience of a sight		3
18	UI	User	Search for places in map	the map takes us to the specific place the user search for.	Google maps	3

ID	Temp	As	I want...	Such as...	Notes	Priority
19	UI	User	See the place location on map		This requirement is added wrt collections. It should be easy to get the location for a story related to a collection when startin from the "collection - story list view" (see below)	3
20	UI	User	get written feedback when no data is available	sounds, collections, stories, images		3
21	UI	User	Get a collection from DF based on hashtags and location			3
22	UI	User	View information about a collection	picture, title and decription	Preview of stories on collection story list, View information about a collection, open story in collection story list (click on previewed information), view on map	3
23	UI	User	Have a "story list view"			3
24	UI	User	preview information about stories related to the place			3
25	UI	User	Discover new sights			4
26	UI	User	seperate between professional and informal stories	DigitalFortalt vs intstagram/twitter		4
27	UI	User	open information from DF in browser			4
28	UI	User	Get others' experiences and reactions from social media	for fun		6
29	UI	User	Ability to share stories on other social media	twitter/instagram		6
30	UI	User	Like and comment on pictures		After logged in with intagram of course. Check with API, if possible.	6
31	UI	User	get a picture gallery retrieved from other databases as well	trondheimbilder.no		7
32	UI	User	Get route description	get shortest path	Google maps API	8
33	UI	User	"Streetview" a place	Digital discovering	Link the places-koordinates to Google streetview.	10
34	UI	User	Have places close to each other group together		be shoen a gallery of photos when places are grouped together	10

ID	Type	As	I want...	Such as...	Notes	Priority
1	IM	User	Se stories from DigitalFortalt as soon as they are submitted.			1
2	IM	User	Read dictionary/wiki input about a place	Get historical unpersonal facts	IE wikipedia	1
3	IM	Developer	Retrieve information from Digital Fortalt based on geo-location		New API from DF	1
4	IM	User	get infromation from Instagram based on hashtag		Two hashtags?	1
5	IM	User	add hashtags to clipboard from a place (picture view)			1
6	IM	User	Mark a spot on the map as a new place	To start a contribution		2
7	IM	User	Listen to and record sound for a given place.			2
8	IM	Developer	Retriveve information from SoundCloud based on #hashtag or location		Only when attribute is creative common	2
9	IM	User	Get tinyurl to data	stories!, places, (link to picture in Flickr, instagram message etc.)		2
11	IM	User	Find relevant hashtags for a place	Make sure my social media posts gets picked up by stedr		6
12	IM	User	Add picture directly from the app		Instagram API (store credentials to external site on local device, not server)	6
13	IM	User	Add sound directly from the app		SoundCloud API (store credentials to external site on local device, not server)	6
14	IM	Developer	be able to filter story by tag, author, institution, video/no video			7
1	INFO	User	Have help information available at any time		information shoul be structured so that elements of information can be reused.	1

6.2 Sprints

In our version of Scrum we had two week sprints composed of entries from the sprint backlog. Scrum was conducted like described in section 3.4. All the sprints are documented in detail in the sheet below.

6 IMPLEMENTATION

6.2 Sprints

Sprint 1															
Stories	Tasks	Day 1 (17.2)	2	3	4	5	6	7	8	9	10	11	12	13	14
IM6 - Mark a spot on the map as a new place	Tag a place on the map.	40 (estimated time remaining)	40	35	35	35	35	30	35	30	20	20	20	20	15
	Create new place object	10	10	10	10	10	10	10	10	10	10	10	10	10	10
UI4 - To have an easy-to-use menu	Slide from map to menu.	10	15	15	10	10	10	10	10	5	0				
	Make the slider look nice	20	20	20	15	15	15	15	15	10	10	5	5	0	
	Create link between buttons and functions.	10	10	10	5	0									
IM2 - Read dictionary/wiki input about a place	Find solutions for getting data from wikipedia	30	30	30	40	40	40	30	30	30	20	20	20	20	20
	Implement in UI.	20	20	20	20	20	20	20	20	20	20	20	20	20	20
	Implement parser.	40	40	40	40	40	40	40	40	40	40	40	40	40	40
IM1 - See stories from DigitaltFortalt as soon as they are submitted.	Research the new API and find out if its worth replacing the old one.	15	15	15	15	15	15	15	15	15	10	5	5	0	
	Make the GUI more understandable.	15	15	15	15	15	15	15	15	10	5	5	0		

Hours	Total	Organizational	IM6	UI4	IM2	IM1									
Øyvind	41	25					16								
Hallvard	40	15	17	5	3										
Tor	36	9				14	13								
Jon-Andre	43	20	5				18								
Jørgen	37	26		11											
Vegard	42	11			25	6									
Total	239	106	22	41	39	31									

Sprint 2															
Stories	Tasks	Day 1 (3.3)	2	3	4	5	6	7	8	9	10	11	12	13	14
IM7 - Listen to and record sound for a given place.	Inform when no sound is available	1 (estimated time remaining)	1	1	1	1	1	1	1	1	1	1	1	0	
	Show add sound icon in blogg	1	1	1	1	1	0								
	Play sounds from soundcloud	10	10	10	10	5	5	5	5	5	5	0			
IM1 - See stories from DigitaltFortalt as soon as they are submitted.	Implement the new API for digitalt fortalt	20	20	30	35	30	30	25	25	25	20	15	5	5	0
IM6 - Mark a spot on the map as a new place	Mark a spot on the map as a new place	20	15	15	10	15	15	15	15	15	15	15	15	15	15
UI3 - Watch pictures of the place	See picture from Instagram	15	25	25	25	15	10	10	10	10	10	5	0		
	Create symbol for watching several pictures	5	5	5	5	0									
	Make a sliding function to watch several pictures	10	10	10	5	0									
UI17 - Read other peoples experiences of a sight.	Read twitter messages inside stedr	40	30	30	30	30	30	10	10	0					
UI18 - Search for places in map	Make Search field in the navigation top bar	20	20	20	20	20	20	30	30	30	20	20	20	10	0

6 IMPLEMENTATION

6.2 Sprints

UI32 - Get route description	Get route description	20	20 20 20 20 20 20 20 20 20 20 10 10 10 10												
UI33 - "Streetview" a place	Use the google API's function to get a streetview	30	30 30 30 30 20 20 20 20 20 20 20 20 20 20												
Hours	Total	Organizational	IM:1	7	6	UI:3	17	18	32	33					
Øyvind	38	19				3	16								
Hallvard	40	8			7	10	3			3	9				
Tor	40	7					14	17			2				
Jon-Andre	45	15	5	7				18							
Jørgen	41	5	2	11					14	9					
Vegard	43	5		9	10	6			8	5					
Total	247	59	7	34	23	39			35	25	14	11			
Sprint 3															
Stories	Tasks	Day 1 (17.3)	2	3	4	5	6	7	8	9	10	11	12	13	14
IM3 - Retrieve information from Digitalt Fortalt based on geo-location	Implement geo-location from digitalt fortalt's API	10 (estimated time remaining)	10	10	10	10	5	0							
INFO-1: Have help information available at any time	Make a tab for help information in the navigation drawer	3	3	3	3	0									
	Make a blogg	5	5	5	5	5	5	5	1	1	1	1	0		
	Make several blogg post about help information needed in stedr application	5	5	5	5	5	5	5	5	5	5	5	5	0	
UI16 - SoundCloud Integration	Integrate soundcloud API in stedr	20	20	20	40	40	40	40	30	30	30	20	15	5	0
	Check with the test data, that we can listen to sounds inside the application	5	5	5	5	5	5	5	5	5	5	5	5	5	0
UI21 - Get a collection from DF based on hashtags and location	Implement Digitalt fortalt's API to receive collection based on hashtags and location	20	20	20	10	10	3	3	3	0					
UI8 - Read text from DF	Be able to read text from digitalt fortalt's API	5	5	5	5	5	10	0							
UI-12 Get my exact location in map.	Get exact location through a button using the Google maps API	10	5	0											
UI34 - Group places close to each other	Be able to group places, based on location and how close they are to each other	30	30	30	30	20	20	20	20	20	10	0			
IM4 - Get infromation from Instagram based on hashtag	Review the Instagram retrieval method, getting pictures based on hashtags and changing it.	15	15	15	20	10	5	5	5	0					
Hours	Total	Organizational	IM:3	4	UI:8	12	16	21	34	IN:1					
Øyvind	43	23	47							20					
Hallvard	41	13		7		2		7		3	9				
Tor	41	4			13				18		6				
Jon-Andre	45	13	5	10	8	7	2								
Jørgen	39	5					11		14	9					

6 IMPLEMENTATION

6.2 Sprints

Vegard	42	4		9		9		8	5	7	
Total	251	62	12	32	10	34	40	11	34	16	

Sprint 4				Day 1 (31.3)	2	3	4	5	6	7	8	9	10	11	12	13	14
Stories	Tasks																
UI2 - Click on event in map	Implement event listeners	15 (estimated time remaining)	10	15	15	5	3	3	0								
UI6 - Be able to switch between map-related and collection-related functionality	Implement tab view in the home screen so you can switch between collection and map related functionality	5	5	0													
UI11 - Search for stories	Implement search field, so you can search for stories inside the application	20	20	20	40	30	30	30	30	15	5	0					
UI13 - Share a place on social media	Implement shared technology, so you can share places on social media	20	20	20	20	20	20	10	10	10	15	5	5	1	0		
UI14 - Read a description about a place	Implement an information button on inside story view.	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	0
	Retrieve descriptions from digitalt fortalt's API	10	10	10	10	5	5	5	5	5	5	5	5	5	5	0	
UI15 - Have a own tab for sounds	Implement new tab for sound view.	5	5	1	1	1	1	1	1	1	3	0					
UI25 - Discover new sights	Be able to discover new sights using the map-related functionality	30	30	30	20	20	20	20	20	20	20	20	20	20	20	10	0
UI27 - Open information from DF in browser	Be able to open digitalt fortalt in a browser view.	10	10	10	10	15	15	10	0								
IM13 - Add sound directly from the app	Implement sound recording functionality, using the Soundcloud API. So that you can add sound directly from the application	30	30	30	30	20	20	20	20	20	20	20	20	20	20	20	0
IM14 - Filter story by tag, author, institution, video/no video	Be able to filter stories, by tag, author, institution, video and no video.	20	30	20	20	20	20	20	20	20	20	15	15	15	15	15	15

Hours	Total	Organizational	IM13	14	UI:2	6	11	13	14	15	25	27	
Øyvind	38	19		10	5		4						
Hallvard	39	11		8	10	2			3	5			
Tor	40	7				10		18			5		
Jon-Andre	46	13				10		10	7	4		2	
Jørgen	44	6							14		15	9	
Vegard	46	3						9	15	5	7	7	
Total	253	59		18	15	22	4	37	25	18	10	27	18

6 IMPLEMENTATION

6.2 Sprints

	View video in stories from digitalt fortalt	5	5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0
UI9 - Share my experiences and knowledge about a place	Implement share functionality.	30	20 20 20 20 20 20 20 20 10 10 10 5 0
	Test the share functionality	5	5 5 5 5 5 5 5 5 5 5 5 5 5 5 0
UI20 - Get feedback when no data is available	Implement "no story available icon"	3	0
	Implement "no picture available icon"	3	0
	Write logic	10	10 10 3 0
UI22 - View information about a collection	Based on digitalt fortalt's API you can view information about collections	10	20 20 10 0
UI23 - Have a "story list view"	Implement Square view over stories, inside the collection tab.	5	5 5 5 5 5 5 5 5 5 5 5 5 5 5 0
	Retrieve stories based on filtering in the collection view.	5	5 15 15 15 15 15 15 15 15 15 15 10 1 0
UI28 - Get social media experiences and reactions from other people	Be able to review and create expirience through the Twitter API	20	15 20 10 0
UI30 - Like and comment on pictures	Implement like and comment button.	3	3 3 3 3 3 3 3 3 0
	Implement functionality responding to the like and share button with the Instagram API.	15	15 15 15 15 5 1 1 0
IM5 - Add hashtags to clipboard from a place (picture view)	Implement add to clipboard button, picture view. So you can easily post a picture on Instagram and later review it inside the stedr application.	20	20 20 20 20 20 20 20 20 20 10 5 0
IM8 - Retriveve information from SoundCloud based on #hashtag or location	Retriveve information based on hashtag or laction, from the soundcloud API.	20	20 15 15 15 15 15 15 15 15 10 5 1 0
IM11 - Find relevant hashtags for a place	Be able to retrieve relevant hashtags for a place of interest.	10	5 1 1 1 0
IM12 - Add picture directly from the app	Implement camera function inside the app.	15	15 15 15 15 30 30 30 30 30 20 20 20 20
	Be able to post picture inside the application, so that it can be reviewed later on.	20	20 20 20 20 20 20 20 20 20 20 20 20 20 20

Hours	Total	Organizational	IM:5	8	11	12	UI:7	9	20	22	23	28	30
Øyvind	48	30		3	5				10				
Hallvard	46	20	8		2	8	3	5					
Tor	45	7		13	10	10						5	
Jon-Andre	44	5	11				7	9			10	2	
Jørgen	40	2 49							14	15	9		
Vegard	43	3						15	5	4	9	7	
Total	266	67	19	16	10	17	8		25	23	20	19	24

7 | Testing

System testing or software testing, falls into something that is called “Black-box testing”. This is a method of software testing, that investigates the functionality of the application, eg. what it does. It will not interfere with its internal structure or workings.

7.1 Testing Procedure

When you are about to conduct a test, you find a test-person. Then you tell them what the software is supposed to do and further give them the Test cases[7.2]. It is very important for the test subjects to think loud so we can get maximum feedback from the testing.

7.2 Test Cases

Test cases are built around the specifications and requirements of the application. What the application is supposed to do.

Get Places	
ID	T-F1
Requirements	SF-1
Feature	Places are shown on the map
Preconditions	<ul style="list-style-type: none"> 1. Flickr is up 2. The Flickr group contains photos with locations 3. Application is installed on device 4. Device is connected to the internet
Test Description	<ul style="list-style-type: none"> 1. Open the application 2. Wait for 30 seconds 3. Click on a pinpoint 4. Zoom out to a world view
Expected result	The map should show some clickable pinpoints. When clicked the pinpoints should open a little box containing a thumbnail picture and small text provided by the Flickr Stedr group. When zoomed out new places should be loaded according to what the user see.
Pass/Fail criteria	The test is considered a pass if the expected result happens. The last step that need to be passed is that the place at Grenada is shown. If there are any inconsistencies with the expected result, the test should be considered a fail.
Severity	High

Table 14 – Test Case: Get Places

Open menu	
ID	T-F2
Requirements	SF-2
Feature	Drawer menu with options is opened.
Preconditions	1,2,3,4
Test Description	<ol style="list-style-type: none"> 1. Click on the menu button 2. Click on all of the icons in the menu 3. Click on the menu again
Expected result	When the menu button is pressed, a drawer menu should open. All of the icons in the drawer menu is also buttons and when clicked again, the menu button should close the drawer menu.
Pass/Fail criteria	The test is considered a pass if the menu button opens and closes a drawer menu. Also, all of the icons should
Severity	High

Table 15 – Test Case: Open Menu

Views	
ID	T-F3
Requirements	SF-6
Feature	All the views are accessible
Preconditions	1,2,3,4 T-F1 Get places
Test Description	<ol style="list-style-type: none"> 1. Click on a pinpoint 2. Click on the small window that appears 3. Click on one of the buttons <i>Images, Sound, Story</i> 4. Dependent on the previous step, click on the buttons not yet pushed 5. Click the menu button 6. Click home
Expected result	<p>Which view that is selected is shown to the user by being in a different colour than the two other buttons. If the button for the selected view is touched, nothing should happen.</p> <p>For every button representing a non-selected view, the user should be taken to the view as indicated by the button text.</p>
Pass/Fail criteria	<p>The test is passed if the button:</p> <p>Image - Takes you to the image view Story - Takes you to the story view Sound - Takes you to the sound view.</p> <p>The selected view has a unclickable button in a different colour representing the selected view.</p> <p>Considered a fail if there are any inconsistencies with the criteria above.</p>
Severity	High

Table 16 – Test Case: Views

Load Content	
ID	T-F4
Requirements	SF-7
Feature	Content is loaded for the places
Preconditions	T-F3 Views
Test Description	<ol style="list-style-type: none"> 1. Click on a pinpoint(not Camera Obscura) 2. Click on the description 3. Go through the views as in T-F3 3a Click on all of the titles on the story 3b Click on two random images 3c Click on a sound
Expected result	The places should be loaded with relevant and accessible content from all of the content providers.. If some content-types are not provided for the specific place, the content type should be loaded but indicate that it is empty.
Pass/Fail criteria	The test is considered a pass if the expected result happens. If there are any inconsistencies with the expected result, the test should be considered a fail.
Severity	High

Table 17 – Test Case: Load Content

Collection view	
ID	T-F5
Requirements	SF-8
Feature	Show a view with the stories related to a collection
Preconditions	T-F2 Open menu T-F4 Load Content 5 It exist a collection
Test Description	1. Press the menu button 2. Press the Collection button 3. Press a collection
Expected result	When the collection button is pressed a new view should open with the list of stories related to the collection.
Pass/Fail criteria	The test is considered a pass if it is possible to open the menu and access a collection with a list of stories.
Severity	Medium

Table 18 – Test Case: Collection View

Collection map view	
ID	T-F6
Requirements	SF-8
Feature	Show places related to a collection as pinpoints in a map
Preconditions	T-F2 Collection View
Test Description	<ol style="list-style-type: none"> 1. Press the menu button 2. Press the Collection button 3. Press a collection 4. Press the <i>show on map</i>-button
Expected result	When the “show on map”-button is clicked, a map view should open with related places showed as pinpoints. Pinpoints not related to the collection should not be placed on the map.
Pass/Fail criteria	The test is considered a pass if all places related to a collection is exclusively shown in a map view.
Severity	Medium

Table 19 – Test Case: Collect Map View

Gallery	
ID	T-F7
Requirements	
Feature	Gallery function
Preconditions	<p>T-F4 Load Content The application is in a place with a story where there are multiple images to the story.</p>
Test Description	<ol style="list-style-type: none"> 1. Press the story title 2. If there are more pictures related to a story, press the arrows
Expected result	When accessing stories with multiple pictures as content, arrows indicating the possibility to go through picture files should appear. When pressed new images should replace the old picture.
Pass/Fail criteria	The test is considered a pass if the expected result happens. If there are any inconsistencies with the expected result, the test should be considered a fail.
Severity	Low

Table 20 – Test Case: Gallery

Upload Content	
ID	T-F8
Requirements	SF-9
Feature	Content can be uploaded to Instagram, Twitter and SoundCloud
Preconditions	<p>T-F4 Load Content</p> <p>6 Successfully connected to the content (not story provider) providers</p>
Test Description	<ol style="list-style-type: none"> 1. Click on a pinpoint 2. Click on the description 3. Go through the views as in T-F3 4. Upload textual content to Twitter 5. Upload picture to Instagram 6. Upload sound to SoundCloud
Expected result	The places should be loaded with relevant and accessible content from all of the content providers.. If some content-types are not provided for the specific place, the content type should be loaded but indicate that it is empty.
Pass/Fail criteria	The test is considered a pass if the expected result happens. If there are any inconsistencies with the expected result, the test should be considered a fail.
Severity	Medium

Table 21 – Test Case: Upload Content

7.3 Test Execution

7.3.1 System Testing

During the final stages of the development, it was performed a system test after integrating all of the developed components. This was an important step done before the acceptance testing, to see that all of the components developed by different team members and the former development team worked together. Since we did not have the resources it was impossible to let an external team do the testing, but to emulate some sort of independence to the system the tests were done by a group member who had been less involved in the development of the client and never tried the client on a smartphone. This of course is a weakness in the testing, since the tester had a general knowledge of the system worked and also a basic idea of the client from using the system on a smartphone emulator.

A summary of the test results is shown in the table below, and an example from the test log can be found in the appendix B

ID	Verdict	Comment
T-F1	Unknown	Valid with full cache, not valid when empty cache
T-F2	Pass	None
T-F3	Pass	None
T-F4	Pass	None
T-F5	Pass	None
T-F6	Pass	None
T-F7	Pass	None
T-F6	Fail	This feature was declared infeasible

Table 22 – System Testing: Validity

7.3.2 Acceptance Testing

Acceptance testing is one of the last levels of the software testing process. The purpose of such testing is to evaluate the system's compliance with the given requirements to check whether it is acceptable for delivery. Hence the name acceptance testing.

On May 14. during one of our last meetings we sat down with Jacqueline Floch for the final acceptance test of our system. This acceptance test included further developed features and improvements of the graphical user interface.

For the main test we had a feature based acceptance test where we went through the requirement list. Not to say that usability is not as important, but our customer had received the latest build almost every week in the later stages of the project. With this, the

customer have conducted usability acceptance testing regularly throughout the project, giving us feedback. Because of this we did not have the same need for a usability based user acceptance test, which we know is the norm, but Jaqueline Floch had the app tested by four of her colleagues, which all gave positive feedback. If there had been any crucial shortcomings it probably would have been pointed out.

The test was conducted by going through the requirement list sorted by priority and individually receiving a verdict on acceptance. The individual requirement either reviewed Pass, fail, outdated or cancel if the requirement was withdrawn. The latter was the case on a lot of the requirements with lower priority since we had little time and the customer wanted us to focus on the more important tasks.

The result of the acceptance test are listed below:

Verdict	Task	Priority
	<i>Overall</i>	<i>max 10</i>
pass	Use English for user interface, code documentation and report	10
not tested	Maintain multi-platform consistency	10
pass	Maintain consistency with UI design with App developed in autumn	10
pass	Maintain a minimalist design	10
pass	Avoid using platform specific UI elements	10
pass	introduction for first users	10
pass	Provide a new user view (activity) or revise the current map view to allow switching between map-related and collection-related functionality	10
pass	Retrieve information from Digital Fortalt based on geo-location; use new API til DF and check storie are available at once	10
pass	Check for places east from Greenwich	10
pass	Functionality to add #hashtag to clipboard.	10
pass	Show picture/video gallery from DF. Make clear that several pictures are available	10
pass	Show text from DF: make sure URL are kept; remove tags: show institution (additianally to author)	10
pass	Licencing constraint set by DF	9
pass	Refine and update the refresh feature	9
	Retrieve information from SoundCloud based on #hashtag - Only retrieve when attribution creative common	9
pass	Show an "add picture icon(first in list)	9
pass	Show a "no picture" icon when no picture is available	9
pass	Preview stories: show sound icon from SoundCloud and title (square format)	9
pass	Retrieve collection from DF based on tag #stedr_collection and location	8
pass	Retrieve description (and picture thumbnail) about collection from DF	8
pass	Story view layout: Use similar layout as for "place information" except that 1) pictures from DF - not from Flickr 2) stories are child in collection 3) add is a different text	8
pass	Story view: retrieve stories for collectiosn based on tag	8
pass	Map view: Show icons from each place of stpry in collection	8
pass	Preview a story in collection	8
pass	Open a story in collection	8
pass	Add tab for sounds	8
pass	Show a "no picture" icon when no picture is available	8
pass	Show a "no story" icon when no story is available	8
pass	Add description about place	7
pass	Open as a normal story - not in a browser view (relatert til add story)	7
pass	Add tinyurl to story (i.e. Link to story in DF to tweet)	6
not realizable	Add tinyurl to place (i.e. Link to picture in Flickr to Instagram message)	6
pass	Help information available at any time	5
cancel	Add information directly from stedr	5
cancel	Ask the user for logging in Instagram (first time)	5
cancel	Store Instagram credentials on mobile - not on server	5
cancel	Ass information directly from stedr	5
cancel	Ask the user for logging in SoundCloud (first time)	5
cancel	Store SoundCloud credentials on mobile - not on server	5
cancel	Add like	5
cancel	Add comment	5
cancel	Filter story by tag, author, institution, video/no video.	3
cancel	Show picture gallery retrieving pictures from other databases, e.g. Trondheimbilder.no	3
cancel	Group places close to each other; one icon per place	1
cancel	Show picture gallery when several places	1

Verdict	Task	Priority
	<i>Overall</i>	<i>max 10</i>
pass	Layout for enhanced readability (improve usability)	
pass	Retrieve information from Instagram based on #hashtag	

As you can see, one requirement are missing a verdict. This is because this was not testable since we waited for a response from SoundCloud at the time. This was later implemented and are now working as it should.

7.3.3 NFR testing

It is important for the project that our result meets the projects main non-functional requirements, described in the “Product quality” section of the SRS chapter (4.12), for it to be considered a success.

Compatibility *Co-existence* With the result found under Reliability - maturity, it is clear that the system can handle that multiple client share the same environment. It is worth to notice that the testing program makes use of threads to be more realistic regarding parallel requests to the server.

Interoperability Since the requirement for interoperability was said to be fulfilled as long as the system followed a standardized method of communication over a known and widely used protocol, the testing consisted of looking at the objects sent from the server and acknowledge that they were valid JSON-objects.

#	Content	JSON
1	Places	Yes
2	Stories	Yes
3	Images	Yes
4	Collections	Yes
5	Sound	Yes

Table 23 – Compatibility: Interoperability

Performance Efficiency We have greatly improved the core of the system to boost the efficiency of the application. This should cause the application to use no more than 300 seconds to pull new content from the APIs. The efficiency was tested by adding new content and recording 10 times with different content posted at different times. By measuring the individual response times and calculating the average result we will get a rough estimate.

Since much of the test content had relatively consistent results we did not think more testing was needed. The Instagram images appeared almost instantly (1 second) consistently, since we completed the tests manually, we could not measure finer times. The twitter content were a little slower as expected, but did also appear consistently at a

#	Content	Time a day	Result
1	Instagram	10:55	1 sec
2	Instagram	14:40	1 sec
3	Instagram	14:45	1 sec
4	Tweet	14:16	19 sec
5	Tweet	14:18	20 sec
6	Tweet	10:40	20 sec
7	Story	15:26	133 sec
8	Story	16:15	10 sec
9	Story	11:00	104 sec
10	Story	15:48	135 sec
11	Story	16:03	107 sec
13	Story	19:01	132 sec
14	SoundCloud	18:51	5 sec
15	SoundCloud	19:15	7 sec
16	SoundCloud	19:20	6 sec

Table 24 – Performance Efficiency: Publishing New Content

reasonable time averaging in just under 20 seconds. The SoundCloud publishing also happened pretty quick averaging 6 seconds As for the stories we published through Digitalt Fortalt, the times were very inconsistent. In three of the tests (# 7, #10 and #12), which was the longest ones, the content had to be uploaded to DF as well as being published. This probably added to the time. All the other tests were performed with the content already uploaded, but we still recorded a massive variation in results ranging from 10 seconds (#8) to 105 seconds (#9).

Average:

$$\frac{133 + 10 + 104 + 135 + 107 + 132}{6} = 103.5 \quad (2)$$

With all results clocking in under 150 seconds, and our goal being under 300 seconds, we are very pleased with the results and can happily see our app passing this test. Considering that the original app needed multiple days for the results to appear, it is needless to say that there have been a massive improvement.

Another important part of the performance efficiency is the application's data usage. Blowing the users' data limit and potentially taking a large part in increasing their phone-bill is something we want to avoid. To avoid this we have implemented a data usage restraint. This restraint is set through equation (1) in section 4.12.

We tested the data usage to make sure our application met our standard. We measured this roaming unconnected to any WiFi-hotspots while running the app.

Quick session

Opening map: 55.74 kB

Entering place with only one story: + 6.07 kB

Browsing session

Browsing map over Trondheim: 1.64 MB

Entering place with 6 stories and 2 Instagram pictures: + 0.02 MB

We found these results to be pretty reasonable. The app does not download more content than necessary, and the results seems really consistent compared to the experiences we have had using the application.

Reliability Testing of components was done by feeding test cases to the components. When accepted, these new components was added to a running server instance that acted as a beta. The beta server was deployed at the 9th of April and has since been monitored to record uptime and errors. Shortly after Easter the old server was changed with the new one so that the customer could use new features, but also so that it was possible to monitor how the server performed under normal usage.

A shortcoming to the test-data below is that each time new features was deployed to the server, the server was restarted. Because of this, the longest interval the server has been running uninterrupted is approximately one week.

Maturity

The testing of maturity was done by analysing the data recorded in comparison to the feedback we got from the customer. Also, there was done an additional test where specific task were completed. Since maturity considers the system under normal use, the additional test was not a stress test in terms of using the system until it broke. Rather it was using the system at a relatively high frequency and noting the number of incidents.

To simulate users we made a testing program which made a total of 500 requests over a short period of time (< 5minutes). This test was done eight times over a single day, but with random rest periods for the server. If the server returned an error this was noted.

The test results clearly shows that there is a problem when the server has been inactive for more than an hour. This is because the server keeps a cache of places for an hour, before it deletes the cache. After the cache is emptied the server waits for a request before it again repopulates the cache. Also, when the server requests Flickr for new places it throws a time-out error. The time-out error doesn't have an effect for the user other than that the application is a bit slow if the user is the first to use the system after the cache has been emptied.

Availability

Since the customer started using the new server (23rd of April) the server has been functional continuous, except from the 6th to the 7th of May. This gives that the system

#	Time since last test	Number of errors	Notes
1	NA	1	Places do not load. This also disables all other content retrievers
2	30 minutes	0	Stories do not load. Does not affect other parts of the system
3	60 minutes	1	Instagram pictures do not load. Does not affect other parts of the system
4	10 minutes	0	Sounds do not load. Does not affect other parts of the system
5	120 minutes	1	Total systems fail
6	60 minutes	1	Places do not load. This also disables all other content retrievers
7	10 minutes	0	Stories do not load. Does not affect other parts of the system
8	60 minutes	1	Instagram pictures do not load. Does not affect other parts of the system

Table 25 – Maturity Testing

has had an uptime under presumably normal use of:

$$1 - \frac{24\text{hrs}}{23\text{days} \times 24\text{hours}} \times 100 = 95.652\%$$

Fault tolerance

The systems fail noted under the availability test was due to a global error on one of the systems external system. From this it is clear that the system depends totally on some of its components and thereby external systems to provide the required functions. To determine how many critical components that exists in the systems, every component in the system was disabled to see the effect on the system. In addition, another part of the test considered interrupting connections to external systems.

Note that to be considered functional, the system had to show places and load stories to the end-user.

This describes a major drawback of the reliability of the system, out of the four retrievers there are two single points of failure. In addition, the system also depends on three of five external systems to function properly. It is also important to note that if any of the external systems make radical changes so the system components becomes outdated, this will also lead to a non-functional system. Because of this the system cannot say to satisfy the requirement for fault tolerance.

Recoverability

Testing for recoverability was done together with the testing for fault tolerance, by tak-

#	Disabled Component	Functional system	Notes
1	Flickr Retriever	No	Places do not load. This also disables all other content retrievers
2	Story Retriever	No	Stories do not load. Does not affect other parts of the system
3	Picture Retriever	Yes	Instagram pictures do not load. Does not affect other parts of the system
4	Sound Retriever	Yes	Sounds do not load. Does not affect other parts of the system
5	External component (Heroku)	No	Total systems fail
6	External component (Flickr)	No	Places do not load. This also disables all other content retrievers
7	External component (Digitalt Museum)	No	Stories do not load. Does not affect other parts of the system
8	External component (Instagram)	Yes	Instagram pictures do not load. Does not affect other parts of the system
9	External component (Soundcloud)	Yes	Sounds do not load. Does not affect other parts of the system

Table 26 – Fault Tolerance Testing

ing the time on how long it took for the system to be functional after re-enabling an external or internal components. Since the system isn't a critical system, the timing requirement was set to five minutes, so to pass the recoverability requirement the system should function normally within five minutes of the component re-enabling.

#	Disabled Component	Functional system after five minutes	Notes
1	Flickr Retriever	Yes	< 5minutes
2	Story Retriever	Yes	< 5minutes
3	Picture Retriever	Yes	< 5minutes
4	Sound Retriever	Yes	< 5minutes
5	External component (Heroku)	Yes	< 5minutes
6	External component (Flickr)	No	< 60minutes
7	External component (Digitalt Museum)	Yes	< 5minutes
8	External component (Instagram)	Yes	< 5minutes
9	External component (SoundCloud)	Yes	< 5minutes

Table 27 – Recoverability Testing

Because of a cache used to store places provided by Flickr, the system uses up to 60 minutes to refresh the cache. This means that errors so that Flickr doesn't respond to requests, leaves the system with an empty cache and thereby no places. This will again lead to a failure of the FlickrRetriever and the system itself as describes under the Fault Tolerance Test.

Portability The multiplatform aspect of the project has played a major part in the choice of environment and frameworks. What we want to achieve is a reasonable consistency through different platforms and versions. We have throughout the development process tested it with many different virtual devices on different settings, but the most valuable ones are the ones performed on the physical devices at our disposal. As previously mentioned, we have unfortunately not had it tested on any iOS devices.

#	Platform (version)	Device	Notes
1	Android (4.4.2 KitKat)	LG Nexus 5 (3 different devices)	Works fine
2	Android (4.1 Jelly Bean)	Samsung Galaxy SII	Works fine
3	Android (4.3 Jelly Bean)	Samsung Galaxy Nexus	Works fine
4	Android (4.2 Jelly Bean)	HTC One	Works fine
5	Android (4.0 Ice Cream Sandwich)	Huawei Mediapad tablet	Works fine
6	Android (4.3 Jelly Bean)	Sony Xperia Z1	Works fine

Table 28 – Portability Testing

8 | Evaluation

8.1 Process

With our process model, Scrum, we found that following it by the book, became very troublesome. Therefore we decided to make some modification to the original model. The main problem with using Scrum by the letter is that we are all students, and this project only counts for half of the semesters study points. This means we all have different schedules, and thus making it difficult to have daily meetings. End meetings, or retrospective meetings is also something we have not prioritized much.

In Scrum it is common to use something called planning poker when deciding how long tasks should take. This essentially means that everyone “votes” on how time consuming they think a given task will be. We found this to be a little unnecessary because its usually so imprecise, and have therefore chosen to just let the persons responsible give their judgements to save time.

8.2 Project Management

One of the main challenges for us concerning project management was the changes made in regards to features during the first part of the project. At the beginning our main customer contact was sick, and the replacement contact was much less acquainted with the project, and from these talks we got the impression that it was wished that we had our own proposals of features, and we were encouraged to start working in the direction we thought most likely to benefit the vision the customer had for the app. When our main customer contact became well again we understood that her vision on the app differed from the impression we previously had gotten in some significant respects, and that she had a clearer idea about what features should be included in the app than our previous customer contact. We take self criticism for not being more careful in the beginning in regards to starting to work on features that we were not sure of, to be prioritized by the customer. It also took a while to come to a common understanding with our customer of the requirements. A while into the project, in order to help us make up for a slow start, the customer wrote a detailed, unambiguous and prioritized list of requirements, including about 50 features. This was a breakthrough in our communication with the customer that helped us reach a common understanding of both the large picture and the details.

8.3 Communication

We feel that the communication internally on the team have been good. By using different channels like Facebook, mail, and phone, we were able to stay in touch throughout the project even though we couldn't always be physically present on the school.

With the customer much of the communication has been good and effective, but parts of it could have been handled better if we were to start over again, and was obviously halted in the beginning by our main customer contact being sick. Due to this sick leave, our communication had to go through a third party. At first, it didn't seem like a big problem, but when we later discovered how this had led to major misunderstandings, it ended up hurting us more than we first thought it would.

About three weeks into the project, we started to have direct communication with our customer over Skype once a week. These meetings didn't work well for us. It made it hard for everybody to engage in the conversations. Although we always paid attention and were careful to write summaries from these meetings, the customer's visions didn't get through to us somehow. We thought we had a clear picture of what our customer wanted, but it turned out to be wrong.

It was not really until we started meeting our customer in person, on March the 5th, that we understood how much we had been talking past each other. There had been lacking clarity in messages between us and since both parts thought they understood each other, no measures were made. Now, after an almost four hour meeting with our customer in person, we finally came to a common understanding of what purpose the app had. This also meant that we suddenly was far behind schedule since we had to completely redo the product backlog based on the new feature list. Had such a list been provided to us from the beginning, much time would have been saved.

That being said, we also take self criticism. There was for instance one incident where a team members didn't pay enough attention on a meeting. This resulted in him working almost 20 hours with trying to integrate the SoundCloud UI experience into Stedr. Our customer had previously stated that it was enough for us to just make a simple API call to the SoundCloud servers and retrieve sounds based on title. Additionally, our customer felt that we sometimes were slow to answer emails. This is regrettable, and we tried to improve in this regard after she mentioned this.

8.4 Project planning

Our pre-study and project planning was really thorough, but in retrospective, much of it ended up being a waste. We spent a lot of time doing research and user tests, trying to figure out what direction we should take the app in. Because of the misunderstanding explained in Communication, we thought we had much more freedom than we actually had. We had the idea that we could just get creative and play with the app as we saw fit.

We do not have any problems working with predefined feature lists and requirements, but then it should be clear from the start that we in fact have these restraints.

In the first half of the project, these were some of the features we were working on that we later dismissed because 1) they didn't fit into what the app is about, and 2) we misunderstood some of the requirements.

- Adding new places from the app
- Wikipedia integration - Possibility to see wiki entries related to the place you are visiting.
- Adding full SoundCloud experience in the app.
- Attaching NRK archive footage to a place.
- General design overhaul. (Dismissed mock-ups can be found in E.1)

8.5 Problems and difficulty

The development of this app have been a bit bumpy for us. We have stumbled upon surprisingly many problems from our risk analysis. One could of course question our preventive abilities, but we still feel we took the right precautions. Some things are just left to luck.

From our risk analysis, this is some of the more important problems we came across in this project.

- Communication failure - Between the team and the customer as explained in Communication.
- Major requirements change - For us, the new feature list meant we had to change the priorities of the requirements a lot.
- Technical difficulties - We had huge trouble setting up the development environment for titanium framework. Two team members didn't even get it to work at all.
- Unavailability - Two team members spent 2 weeks in China which reduced our capacity right before Easter.
- Lack of Competence - Combined we had zero experience with some of the technologies used in this project beforehand.
- Sickness - As explained above.
- Equipment Failure - One team member had to send his computer to service for a total of 6 weeks. Two others had to replace their android device.

This was also the first time for all of us to take over a project halfway to further develop it. This meant that we didn't have the option to choose technologies based on our strengths since its already chosen by the previous group. Actually, none in our group had any experience with any of the technologies used. This was very unfortunate since we have to spend a lot of time to learn new frameworks. Additionally, you have the aspect of understanding all the code that had been written.

8.6 Lesson learned

In terms of management, we have learned some important lessons.

Firstly is the importance of clear milestones. In the beginning we had very unclear goals and this affected the group. It was later solved when we got a prioritized list 7.3.2 from our customer. Working without a clear focus can be challenging for the team members.

Also, clarity internally. If we had been a bit more clear on responsibilities within the group, the development process would have gone more smoothly.

Lastly, we cannot overemphasize this: Communication is a determining factor for a projects success that we should not underestimate the importance of. Of course we already knew this coming into the project, but in practice it is easy to lose focus. Because of a simple misunderstandings we spent two weeks working on the user interface and other irrelevant features when we should have prioritised integrating APIs instead. So clarity on what the customer want is important. We also learned that Skype meetings or talking over the phone is not sufficient for good communication.

On the technical side, when it came to development platforms, we initially didn't have much to say, since the previous group had already made the decision to use Titanium front-end and Play back-end.

In the aftermath of the project, our general consensus is that Titanium is a sub-optimal framework for this kind of project. Stedr is an app that uses many different APIs and integrating these in a good way turned out to be hard. This is because Titanium provides a kind of half-breed between native and web development. Most APIs are made for either native or web and does not translate well into the Titanium environment, thus making it difficult to use. Just setting up the work environment turned out to be quite a task, especially on some of our team members machines.

It would, in our opinion, been better to work native or just use pure web-standard (HTML5, CSS3 and JavaScript).

8.7 Technical evaluation and recommendations

After reviewing the test cases the system seems to satisfy the requirements (and features) set by the customer, except for the non-functional requirement related to reliability.

The failing of the reliability is caused by the server, because of the lack of storage and the servers dependency of external systems. Especially the external system used for storage of the Places (Flickr) has been a problem, this is because the external system uses very long time to respond to our internal system's requests. This problem will also probably magnify in size as more Places are added to Flickr, which in turn will lead to an even greater delay of the response sent by the external system.

To enhance the systems reliability the customer should evaluate the possibility to add a database back-end. This would make it possible for the system to store its own information so that the only dependency and single point of failure would have been the server itself. A requirement for this, is that the customer has to take editorial responsibility for the content stored on the server which the customer at the time of this project wasn't willing to do.

Another recommendation for further work is that the option to remove the back-end as a whole. As of now the system roughly is a Model View Controller-application, where the smartphone application acts as the view. The computational restrictions is therefore back-end, but the back-end is limited to run as a cloud service with limited resources as it should be free of charges. A consequence is that it becomes natural to ask if the processing done back-end, couldn't be done more efficient at the end-users smartphone. Combining the removal of a back-end with local storage, would also lead to the application no longer require continuous Internet access.

Both of these recommendations implicates that some parts of, or the whole system, needs to be rewritten. Careful evaluation of the options needs to be done before eventually deciding if the reliability of the system isn't good enough.

8.8 Conclusion

Even though we were both unfortunate and a little careless in the beginning, we managed to pull ourself together and produce a result we are really proud of. All that is left now, is to hope our customer feels the same about our work, and like us feels that we had a working relationship that on balance was pleasant, fruitful and successful. We have learned incredibly much from this project since we have stumbled across a fair amount of unlikely, but yet realistic problems that can occur in every working project. We feel that this valuable experience will help us to improve greatly for future projects.

9 | Attachments

9.1 Class diagrams

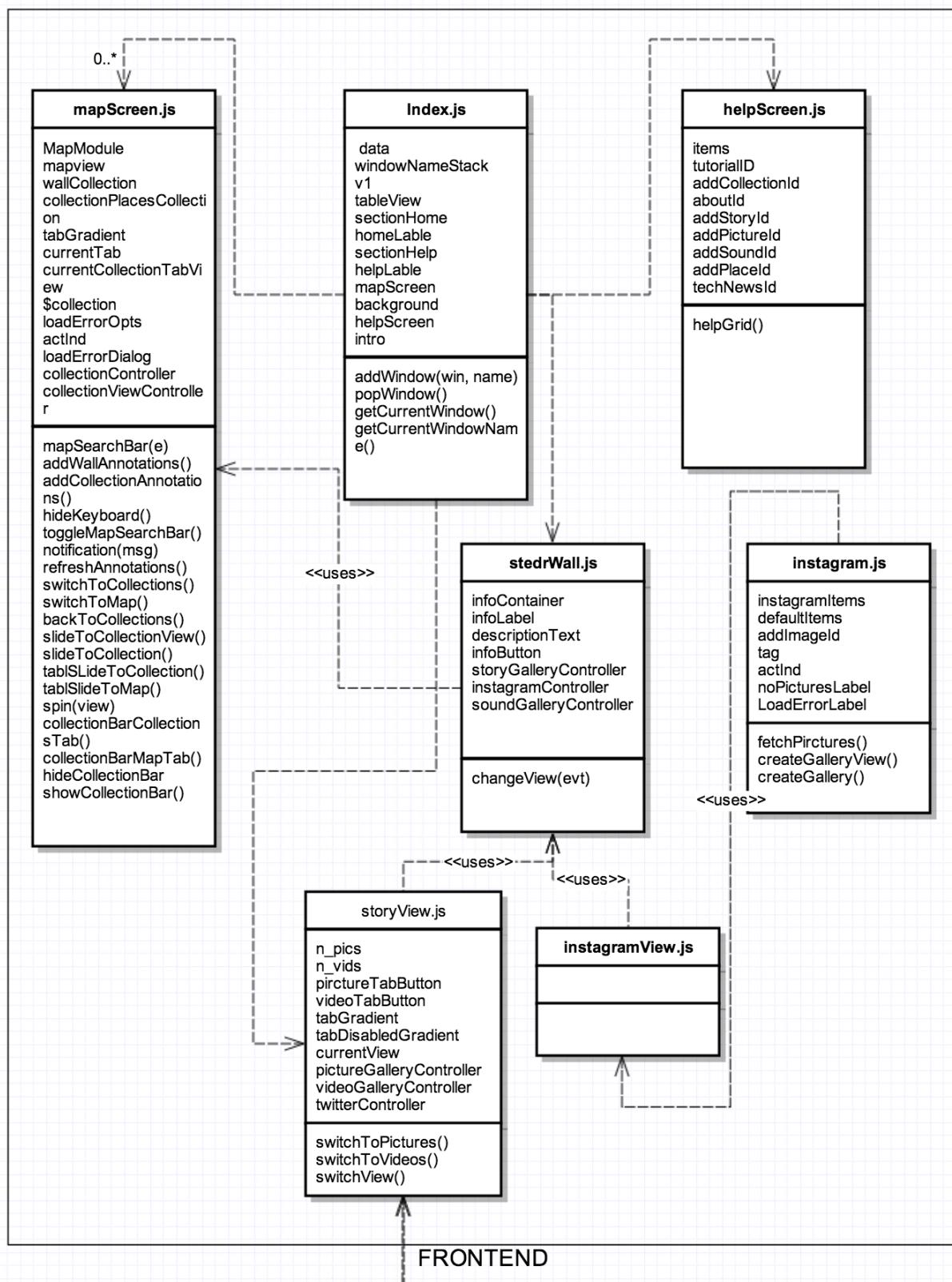


Figure 15 – Class diagram for front-end part 1

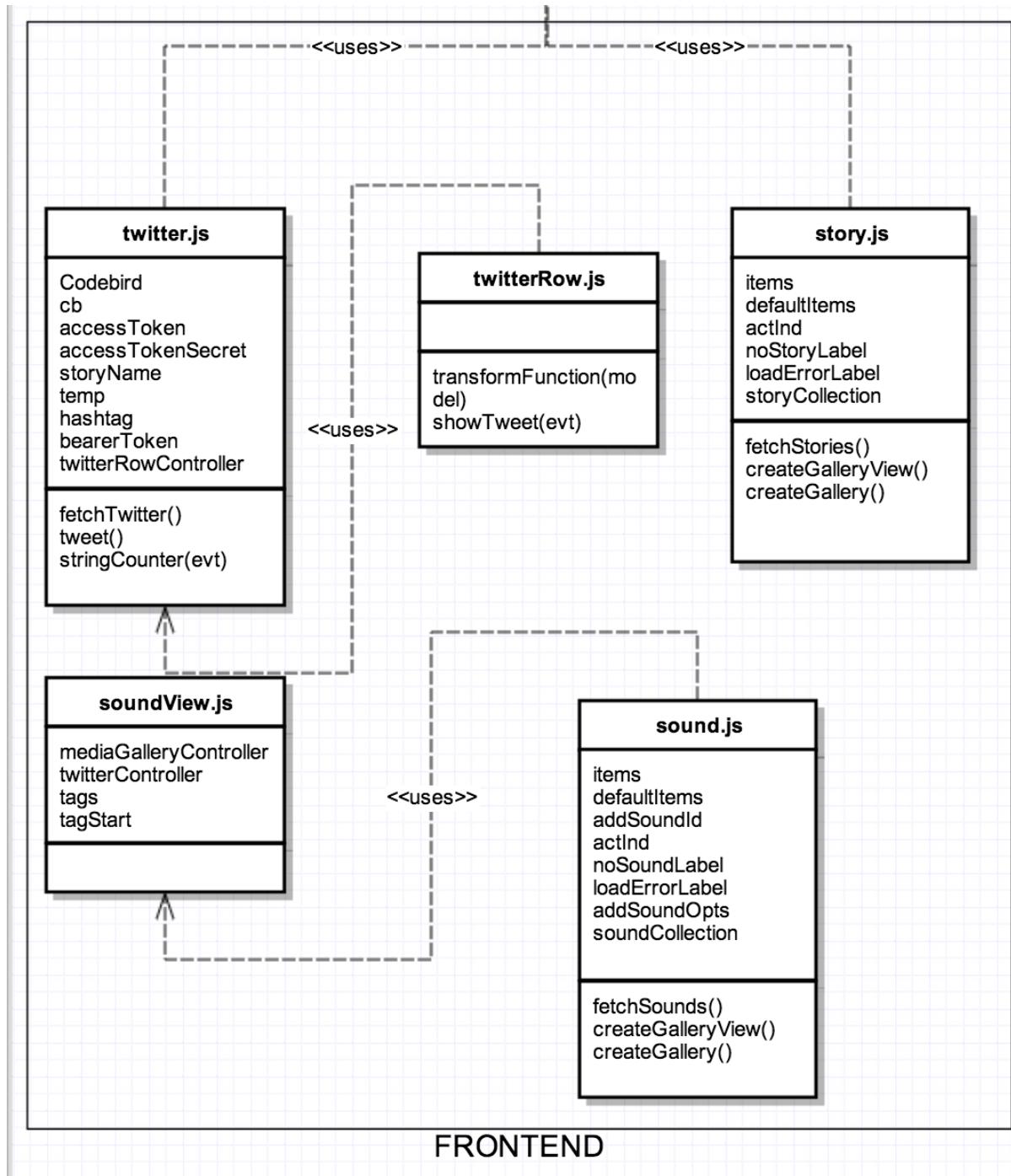


Figure 16 – Class diagram for front-end part 2

9 ATTACHMENTS

9.1 Class diagrams

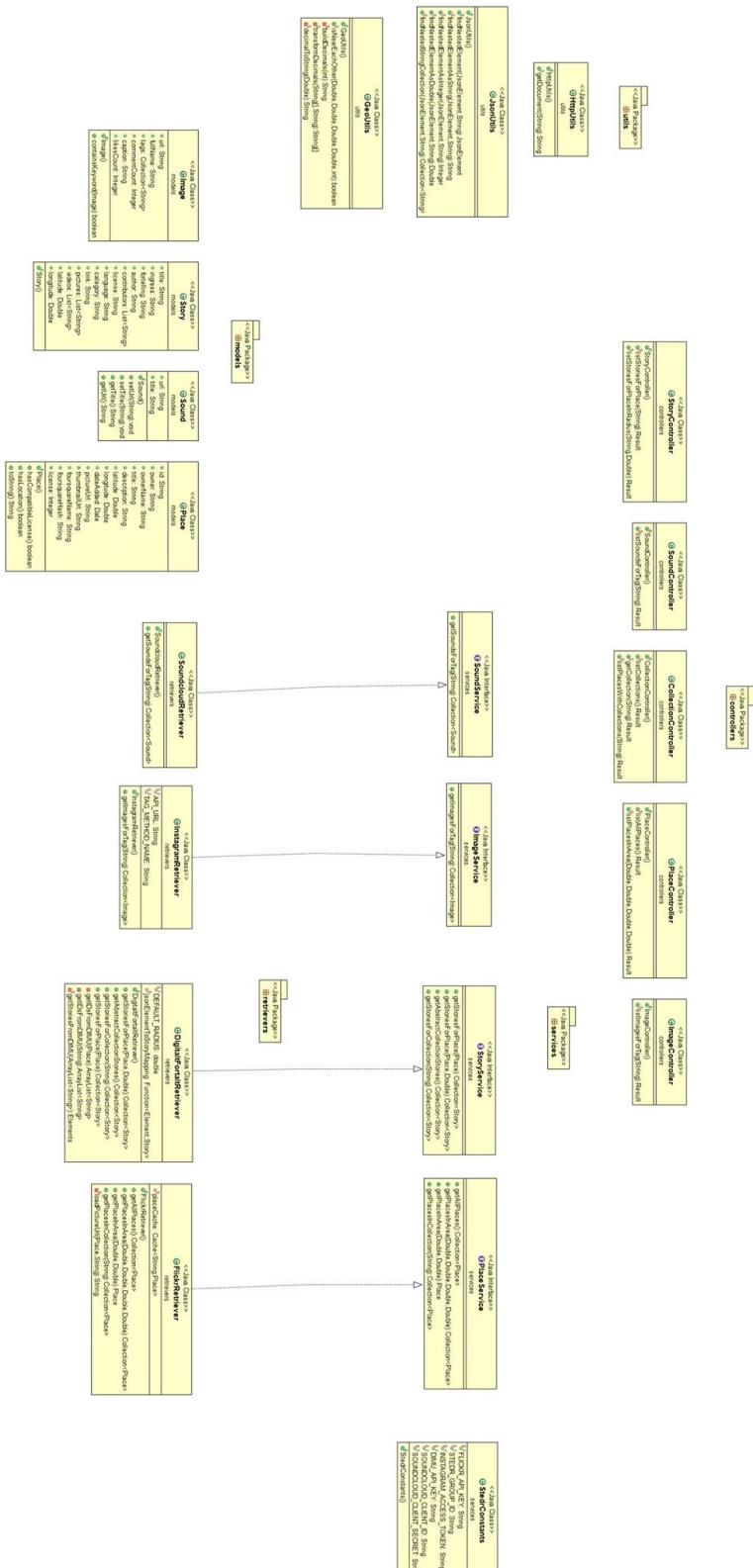
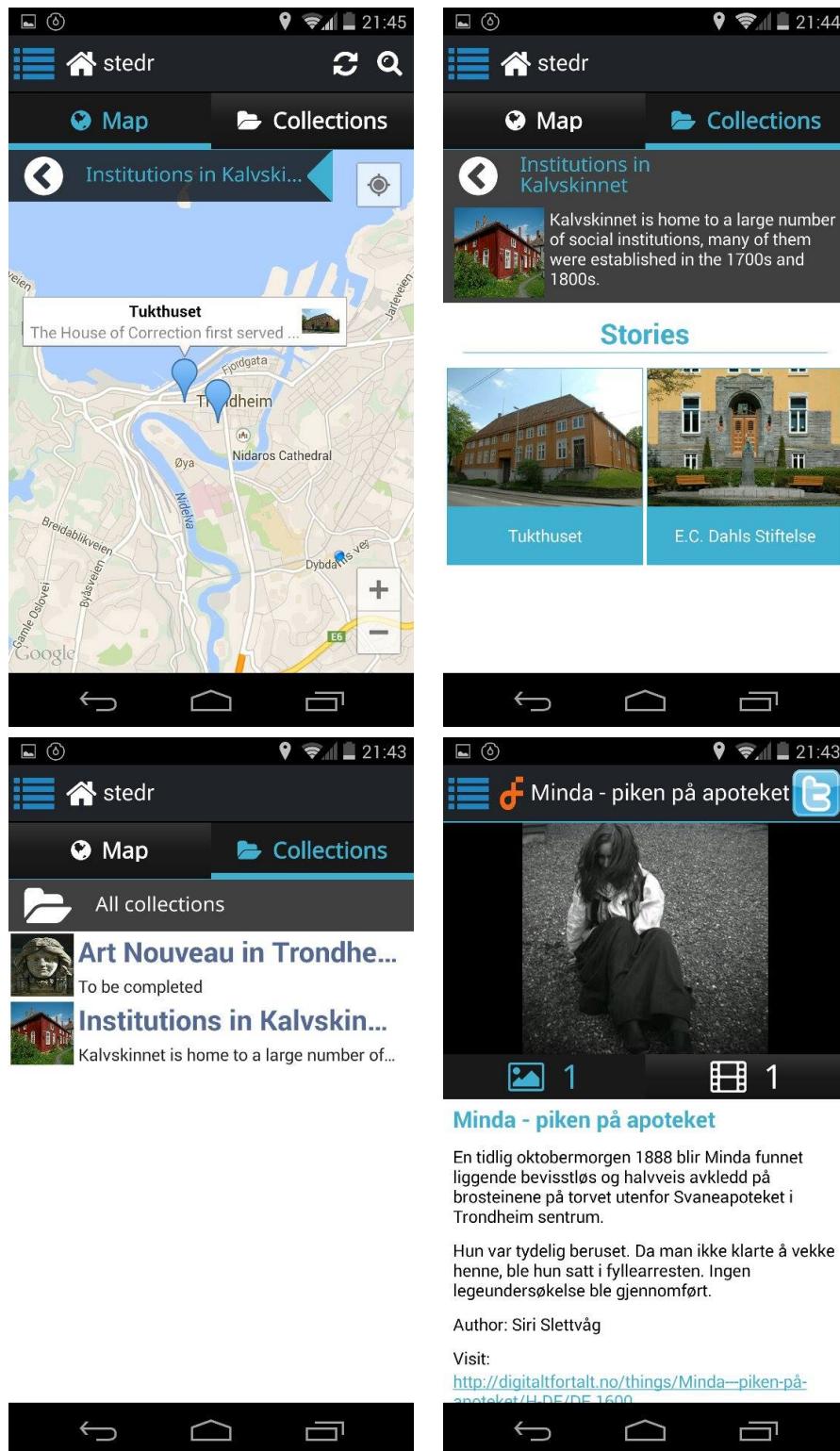
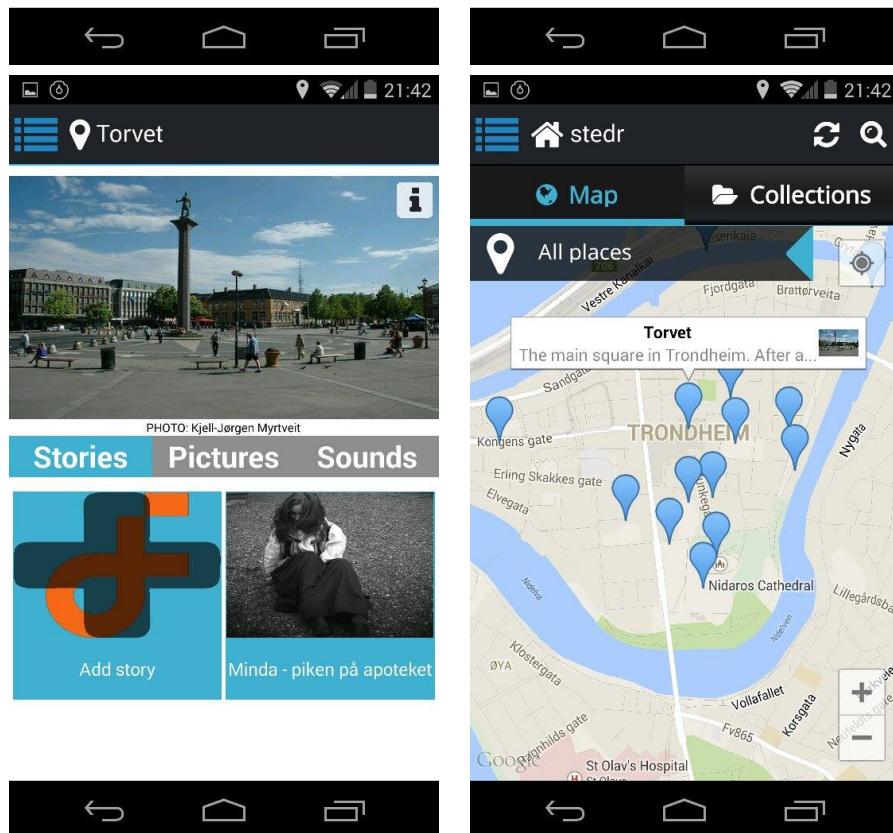
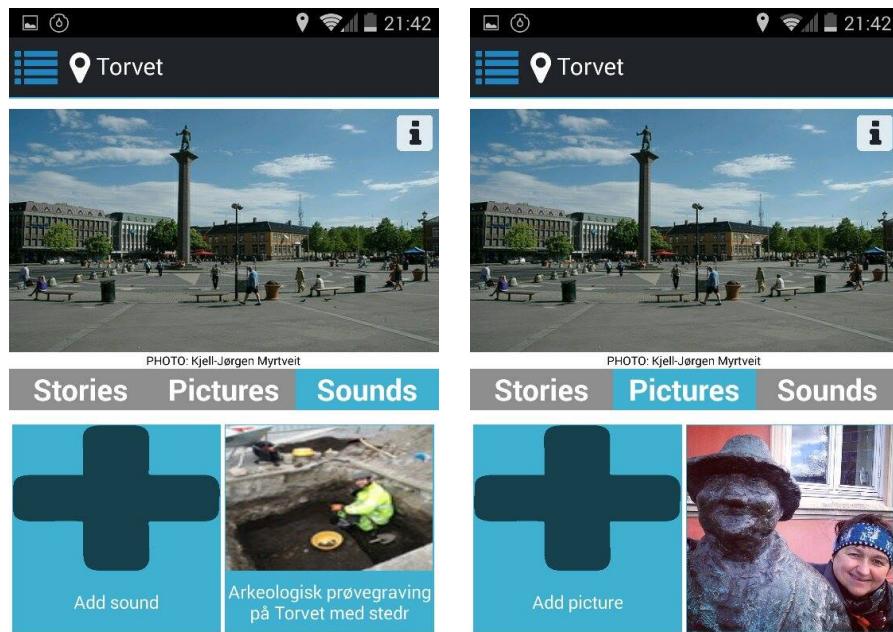


Figure 17 – Class diagram for back-end

9.2 Screenshots





A | Developers Guide

This introductory developers guide, aims to make a developer able to set up Stedr so that the developer gets an overview of how the system works and also how the developer can get started programming. The first part of the guide is about the back-end while the second part is about the front-end. Both of the guides needs to be completed to get the example program running, and the back-end part has to be completed before the front-end part.

This guide is **not** meant as a tool guide, so some parts of the guide is superficial and it is left to the reader to study the tools closer.

A.1 Back-end

The back-end is a Java program using the Play Framework, and the back-end is deployed to the Heroku, a cloud platform hosting applications as services. The source code itself maintained on a GitHub account provided by SINTEF called TagCloud. Before continuing this means that a couple of prerequisites has to be fulfilled by the developer.

The developer should have:

- Installed an updated version of JDK
- Installed a code editor (Eclipse will be used in the tutorial)
- A working GitHub-account
- Installed git
- Cloned TagCloud/StedR_server with the help of Git from GitHub
- Installed the Typesafe Framework from Play Framework
- An Heroku account

On your computer, open up a terminal of your choice (cmd, bash, ...) and navigate to the folder where you have extracted Typesafe Activator (Play Framework). Depending on your platform type the command which will execute activator. It is possible to use Typesafe Activator with a graphical user interface by passing ui as a parameter. This will look something like:

```

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Jon-Andre>cd workspace

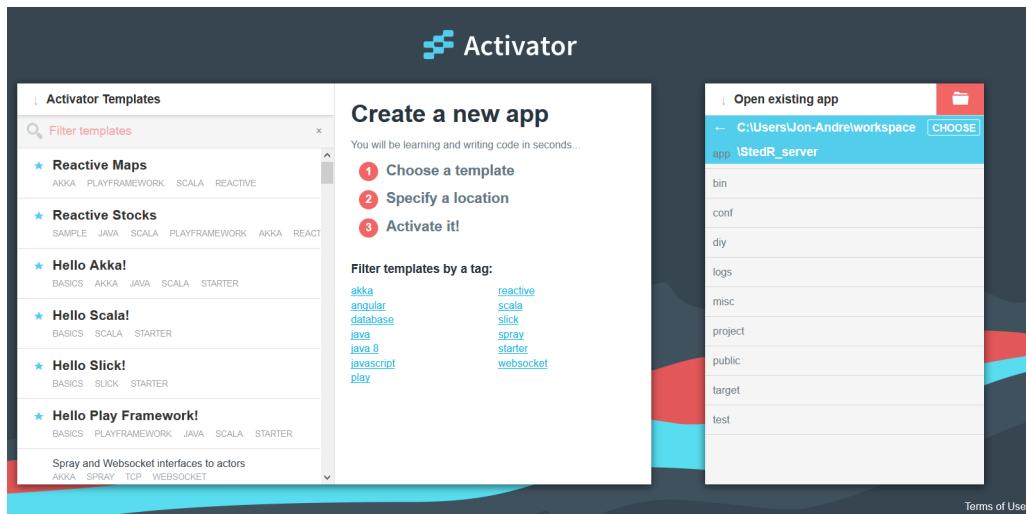
C:\Users\Jon-Andre\workspace>cd activator-1.0.13

C:\Users\Jon-Andre\workspace\activator-1.0.13>activator ui

```

The graphical user interface should then open automatically in a browser view. If this doesn't happen check the terminal for error messages.

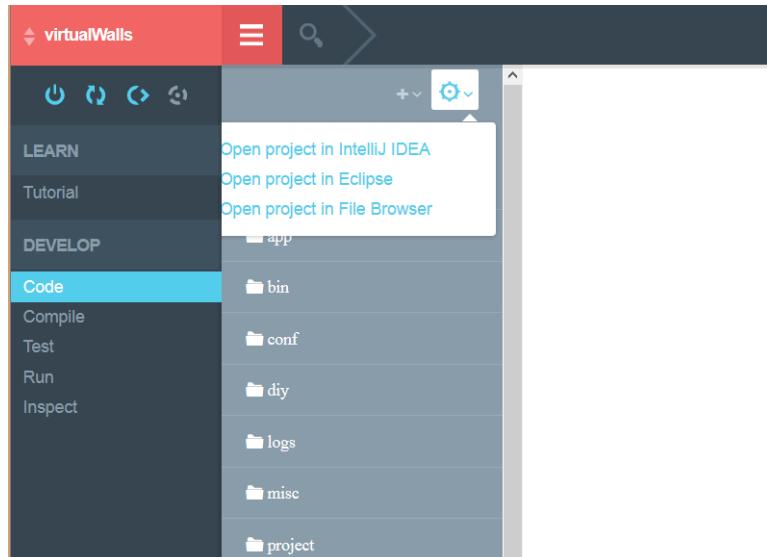
In the right sidebar navigate to the folder where you have cloned StedR_server from GitHub. Click choose. Now the program is starting to compile, and the server will try to run as a local instance on localhost port 9000.



Notice that Typesafe Activator itself is running on port 8888. During the compiling of the system problems may occur. Often this is related to a mismatch between Typesafe and Java, for example an updated version of Java and an outdated version of Typesafe often leads to issues. If the program is compiled successfully, something like this should appear at localhost:

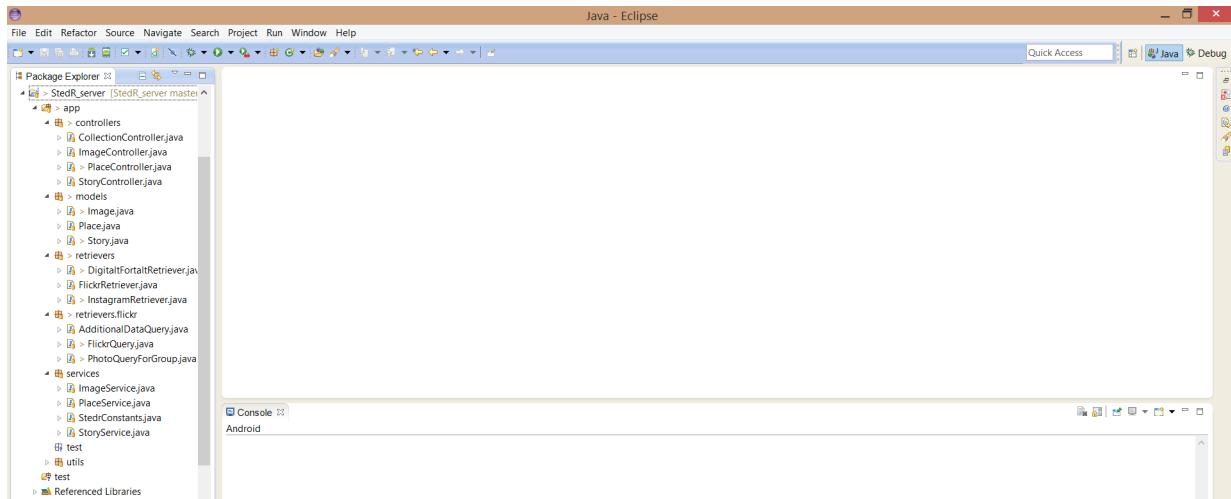
Route	Description
1 GET /places.json	controllers.PlaceController.listAllPlaces()
2 GET /places_in_area.json	controllers.PlaceController.listPlacesInArea(startLatitude:Double, startLongitude:Double)
3 GET /stories.json	controllers.StoryController.listStoriesForPlace(placeId:String)
4 GET /stories_in_radius.json	controllers.StoryController.listStoriesForPlaceInRadius(placeId:String, radius:Double)
5 GET /listCollections.json	controllers.CollectionController.listCollections()
6 GET /getCollection.json	controllers.CollectionController.getCollection(tag:String)
7 GET /places_in_collection.json	controllers.CollectionController.listPlacesWithCollections(tag:String)
8 GET /images.json	controllers.ImageController.listImagesForTag(tag:String)
9 GET /assets/\$file<.+>	controllers.Assets.at(path:String = "/public", file:String)

If you already have tried importing the folder with the source code to an editor like Eclipse, you may have noticed a lot of errors appears. To import the program and its dependencies as a project: In the left sidebar of Typesafe, click on Code, then the gear-icon. Here you can choose between IntelliJ and Eclipse, and Typesafe will then generate project files and guide you through how to open the program as a project.

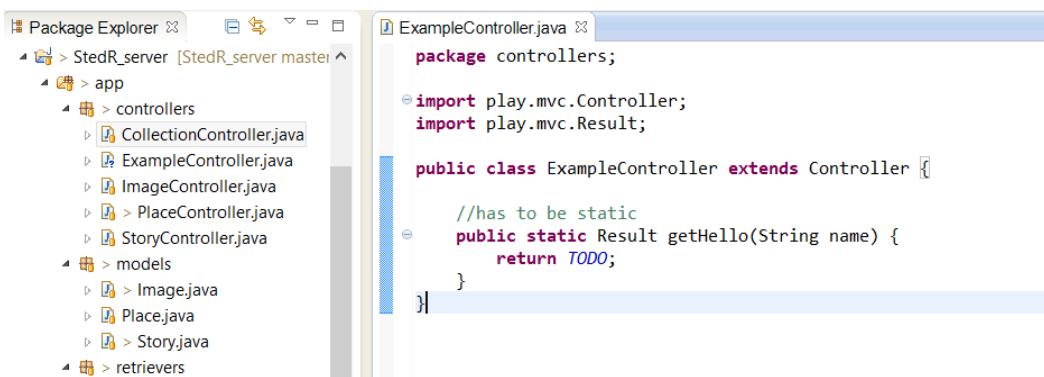


Now you should have a running instance of the server locally, and you're also ready to code.

In Eclipse you will get an overview of the different source files and source packages. In the `Controllers` package you will find controllers that take care of identifying queries (sent as an URL), creating Retrievers that process the queries and at last returning a response to the query. All of the Controllers are written by former Stedr-developers, but all of the Controllers extends a Controller from the Play Framework.



Now, let's create a controller for our example application:



We ask for a parameter called `name` which naturally is the name you want to be displayed in the smartphone-application. To pass a parameter you have to edit the file called `routes` in the conf-folder, the passing is done directly from the smartphone application.

```

Java - StedR_server/conf/routes - Eclipse
File Edit Refactor Navigate Search Project Run Window Help
Package Explorer ExampleController.java routes
# Routes
# This file defines all application routes (Higher priority routes first)
# ~~~~~

#Example
GET /hello.json controllers.ExampleController.getHello(name: String)

# Places
GET /places.json controllers.PlaceController.listAllPlaces()
GET /places_in_area.json controllers.PlaceController.listPlacesInArea(startLatitude: Double,
GET /stories.json controllers.StoryController.listStoriesForPlace(placeId: String)
GET /stories_in_radius.json controllers.StoryController.listStoriesForPlaceInRadius(placeId: String)

# Collection
GET /listCollections.json controllers.CollectionController.listCollections()
GET /getCollection.json controllers.CollectionController.getCollection(tag: String)
GET /places_in_collection.json controllers.CollectionController.listPlacesWithCollections(tag: String)

# Images
<

```

It would now have been possible to create a retriever directly, but we won't do that. In the services folder there are three files ending with Service.java. These files are interfaces, and the reasoning behind them is that it should be easy to add or change services. As of now stories are provided by Digitalt Fortalt, so we have a DigitaltFortaltRetriever.java which implements StoryService.java. That way we can change the content provider to Wikipedia by creating a new retriever WikipediaRetriever.java which implements StoryRetriever.java. A lot of code would then have to be written in order to get the fictional WikipediaRetriever.java functional. Back to the example we will therefore create an ExampleService.

The screenshot shows the Eclipse IDE interface with the title bar "Java - StedR_server/app/services/ExampleService.java". The menu bar includes File, Edit, Refactor, Source, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, and Find. The Package Explorer view on the left shows the project structure under "StedR_server [StedR_server master]". The "app" package contains controllers (CollectionController.java, ExampleController.java, ImageController.java, PlaceController.java, StoryController.java), models (Image.java, Place.java, Story.java), retrievers (DigitalFlickrRetriever.java, FlickrRetriever.java, InstagramRetriever.java), and services (ExampleService.java, ImageService.java, PlaceService.java, StedrConstants.java, StoryService.java). The "routes" view shows routes for CollectionController, ExampleController, ImageController, PlaceController, and StoryController. The "ExampleController.java" editor tab is active, displaying the following Java code:

```
package services;

public interface ExampleService {
    public String getHello(String name);
}
```

Now we're ready to implement the HelloRetriever.java

The screenshot shows the Eclipse IDE interface with the title bar "Java - StedR_server/app/ExampleController.java". The menu bar includes File, Edit, Refactor, Source, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations like Open, Save, Cut, Copy, Paste, and Find. The Package Explorer view on the left shows the project structure under "StedR_server [StedR_server master]". The "app" package contains controllers (CollectionController.java, ExampleController.java, ImageController.java, PlaceController.java, StoryController.java), models (Image.java, Place.java, Story.java), retrievers (DigitalFlickrRetriever.java, ExampleRetriever.java, FlickrRetriever.java, InstagramRetriever.java), and services (ExampleService.java, ImageService.java, PlaceService.java, StedrConstants.java, StoryService.java). The "routes" view shows routes for CollectionController, ExampleController, ImageController, PlaceController, and StoryController. The "ExampleController.java" editor tab is active, displaying the following Java code:

```
package retrievers;

import services.ExampleService;

public class ExampleRetriever implements ExampleService{

    @Override
    public String getHello(String name) {
        return "Hello " + name;
    }
}
```

After the HelloRetriever.java the last thing that needs to be completed is the ExampleController which was created in the beginning of the example

```

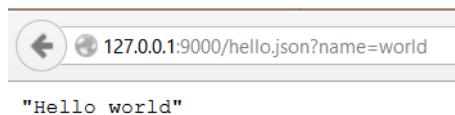
Java - StedR_server/app/controllers/ExampleController.java
File Edit Refactor Source Navigate Search Project Run Window Help
Package Explorer routes ImageService.java ExampleService.java ExampleRetriever.java
ExampleController.java
package controllers;
import java.util.Collection;
public class ExampleController extends Controller {
    //has to be static
    public static Result getHello(String name) {
        ExampleService exampleService = new ExampleRetriever();
        String example = exampleService.getHello(name);
        return ok(Json.toJson(example));
    }
}

```

This concludes in a server which will respond with Hello World if World is passed as a parameter. To see this, the system has to be recompiled in TypeSafe Activator. After recompiling open localhost port 9000, there you should see:

Route	Description
1 GET /hello.json	controllers.ExampleController.getHello(name:String)
2 GET /places.json	controllers.PlaceController.listAllPlaces()
3 GET /places_in_area.json	controllers.PlaceController.listPlacesInArea(startLatitude:Double, endLatitude:Double, startLongitude:Double, endLongitude:Double)
4 GET /stories.json	controllers.StoryController.listStoriesForPlace(placeId:Long)
5 GET /stories_in_radius.json	controllers.StoryController.listStoriesForPlaceInRadius(placeId:Long, radius:Double)
6 GET /listCollections.json	controllers.CollectionController.listCollections()
7 GET /getCollection.json	controllers.CollectionController.getCollection(tag:String)
8 GET /places_in_collection.json	controllers.CollectionController.listPlacesWithCollection(collectionId:Long)
9 GET /images.json	controllers.ImageController.listImagesForTag(tag:String)
10 GET /assets/\$file<.+>	controllers.Assets.at(path:String = "/public", file:String)

If this is correct, <http://127.0.0.1:9000/hello.json?name=world> should give:



Normally you would commit this to the git-repository on GitHub, and if the new functionality is to be a part of the running server it should also be committed or deployed to Heroku. This is done by pushing the server commit to `git@heroku.com:stedr-beta.git`.

Note that in order to push to the server directly, you will need contributor access to stedr-beta.herokuapp.com. If this isn't available or provided upon request you can create a new Heroku instance, but then the request URL destinations have to be changed in the smartphone application.

Note that the first server commit to a new Heroku instance is so slow that you can get an error message regarding time-out when compiling. At the time of writing we are not sure if this error is ours or Herokus, because it may seem like the error is related to Herokus slug compiler. Heroku needs to install OpenJDK on their instance the first time the server is pushed to Heroku, OpenJDK is fairly large so this would also explain that the server's slug is fairly large. The error was fixed by adding a system.properties-file under app in the project folder, with the value java.runtime.version=1.7. This file should now be pulled from GitHub when cloning the project.

Remember to add the API-keys to the `StedrConstants.java`. These must ONLY be added to private repositories, and should really only be added to a deployment branch of the project. Also, since almost all of them are owned by members of former dev-teams they may become invalid without notice:

```
// this apikey belongs to: chrisfro@stud.ntnu.no
public static final String FLICKR_API_KEY =
    "cd04f142470e7de7c992b3a3b140f636";
public static final String STEDR_GROUP_ID =
    "2297124%40N25"; // escaped
this access token belongs to: knut.nerga@gmail.com
public static final String INSTAGRAM_ACCESS_TOKEN =
    "623771306.1fb234f.09aa9355cc8e469f8839d18385f719d5";
// this access token belongs to: Jacqueline Floch
public static final String DIMU_API_KEY =
    "h_LUmzB SAC9CqsDzsuzgg";
// these access tokens belongs to: Tor Barstad
public static final String SOUNDCLOUD_CLIENT_ID =
    "737095f8e223d83af9b88a9b48d90ea9";
public static final String SOUNDCLOUD_CLIENT_SECRET =
    "c559b568bfa50272ff18bcb27a87fa65";
```

A.2 Front-end

The front-end is developed using Appcelerator Titanium. Similar to the back-end, the source code itself is maintained on a GitHub account provided by SINTEF called Tag-Cloud. Before continuing this means that a couple of prerequisites has to be fulfilled by the developer.

The developer should have:

- A working GitHub-account
- A working Appcelerator-account
- Installed git
- Cloned TagCloud/VirtualWall with the help of Git from GitHub.
- An Android or iOS device

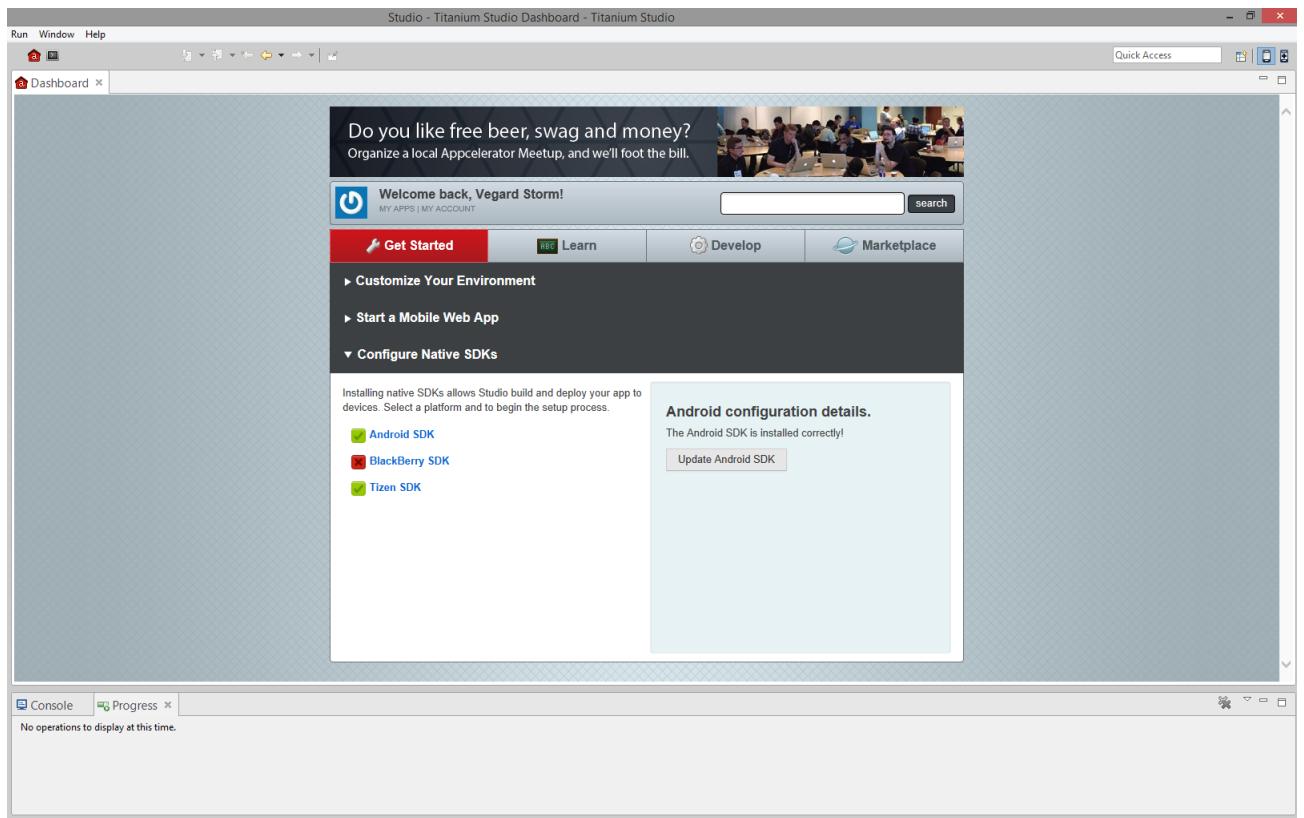
Note that you need a to install the sync software for your device and make sure you have USB debugging enabled. It is also important to note that you need a Mac in order to build the program for iOS devices. Android or iOS device is not required, since it is possible to use an emulator.

Download Appcelerator Titanium.

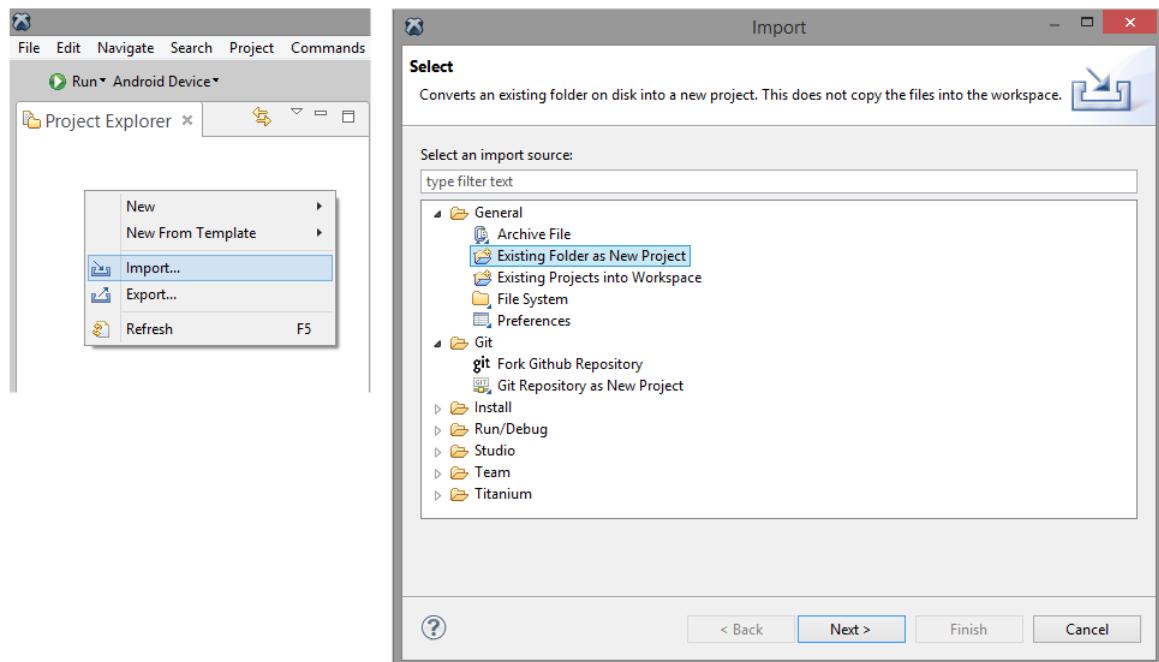
Install titanium by following the installation wizard. When titanium is finished installing, open it. Titanium will ask you to choose a workspace location, where your projects will be stored. You then have to log in with your Appcelerator-account.

Titanium will then most likely attempt to install software development kit (SDK) for Android. If it does not, you can do so via the dashboard. Click the “Get Started” tab and scroll to “Configure Native SDKs”, click Android SDK and make sure it is updated. The Tizen and Blackberry development kits are not necessary for this application.

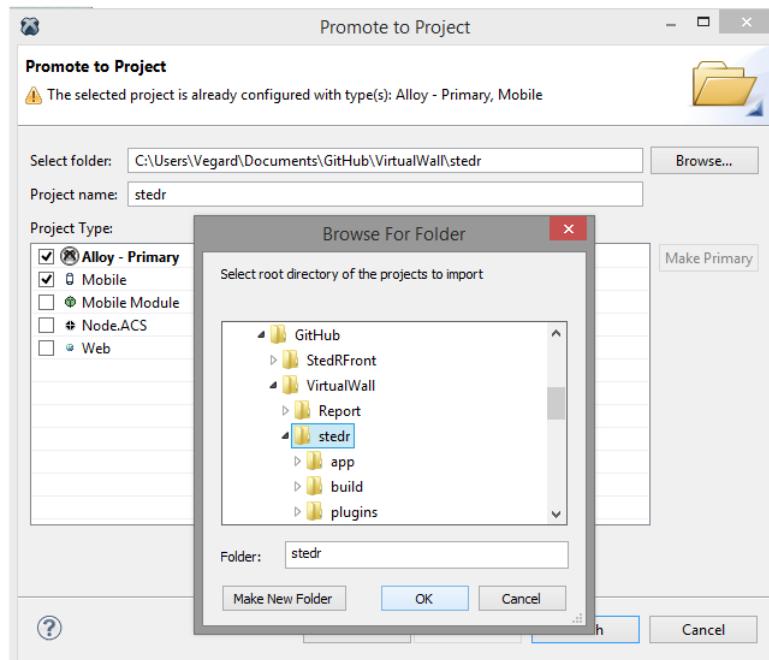
If you have a Mac, you should be able to install the iOS SDK as well. You need this if you are testing the application on an iOS device.



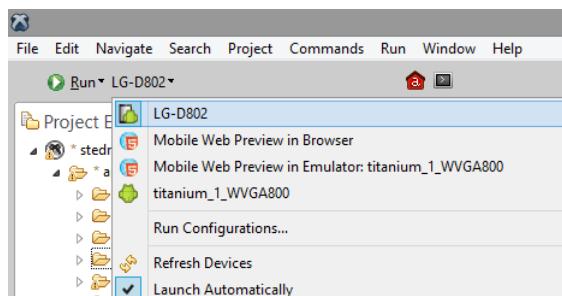
Now we are ready to import the Stedr project. 1. Right click in the project explorer and choose “Import”. 2. Choose the “Existing Folder as New Project”-option under “General” and press “Next”.



3. Click browse and choose the folder where you downloaded Stedr and press “OK”. Example: C:/Users/Vegard/Documents/GitHub/VirtualWall/stedr 4. Under “Project type” make sure Alloy is checked as primary and mobile is checked (This should happen automatically) and press “Finish” if everything is in order.



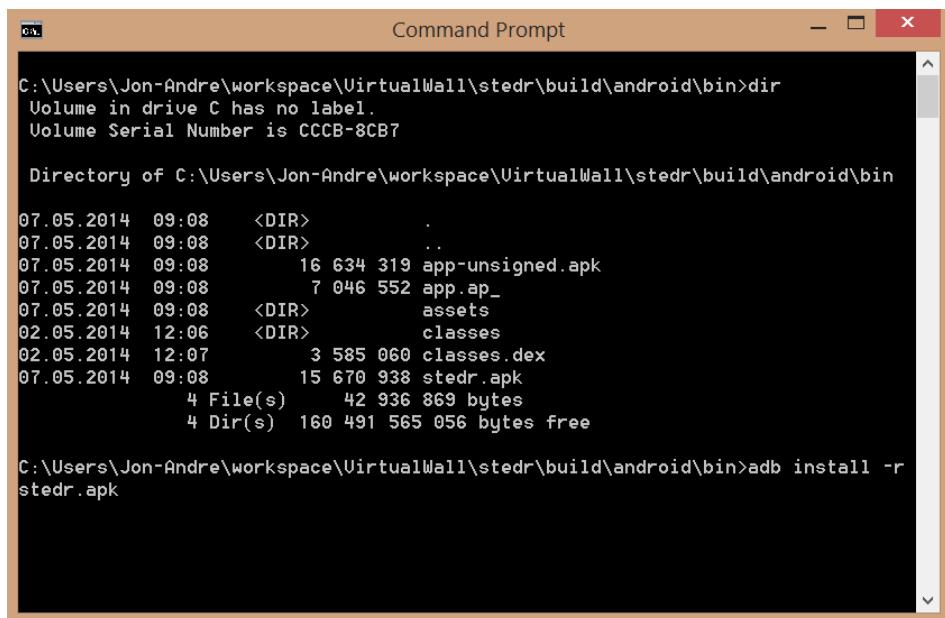
We are now ready to code or run the project. It might be beneficial to try to run the program on your device or an emulator before you start coding. If you have already done what is explained in the prerequisites and connected your device to the PC via USB, your phone should show up in the list of devices as displayed below. If it does, select it and click “Run”. However if it does not, make sure you have the synchronization software for your device installed, and that you have “USB debugging” enabled in your phone settings. If you have issues finding the “USB debugging” option on your phone, it may be because you do not have access to developer settings. The easiest way to figure out how to get access is to google “how to access developer options on *your device*”.



Emulator: It is possible to run the program on the official Android emulator, but it has some performance issues. An alternative emulator is the Genymotion. Note that this emulator has some restrictions when asking for a free license, so make sure not to break their term of service.

Genymotion does not support Google Apps (i.e Google Map Service) which is needed to run stedr. To install Google Apps on Genymotion, see this screencast. Before continuing it is also practical to add adb to the environmental path, so that adb is accessible directly in the terminal. On Windows you can locate adb.exe under the folder platform-tools where you have installed the Android SDK.

With Genymotion started (the emulated device should be running) enter adb install -r stedr.apk. It is easiest to do this from the builder folder where, but you can probably send the location of the apk as a parameter. Notice the -r parameter, this is for reinstalling an apk which becomes necessary to use if the apk already is installed at the emulated device.



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window contains the following text:

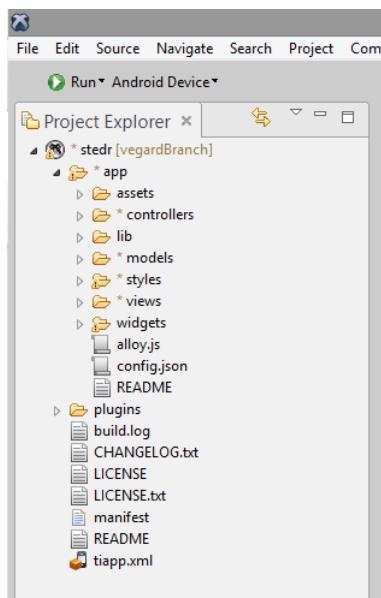
```
C:\Users\Jon-Andre\workspace\VirtualWall\stedr\build\android\bin>dir
Volume in drive C has no label.
Volume Serial Number is CCCB-8CB7

Directory of C:\Users\Jon-Andre\workspace\VirtualWall\stedr\build\android\bin

07.05.2014  09:08    <DIR>      .
07.05.2014  09:08    <DIR>      ..
07.05.2014  09:08           16 634 319 app-unsigned.apk
07.05.2014  09:08            7 046 552 app.ap_
07.05.2014  09:08    <DIR>      assets
02.05.2014  12:06    <DIR>      classes
02.05.2014  12:07           3 585 060 classes.dex
07.05.2014  09:08           15 670 938 stedr.apk
               4 File(s)   42 936 869 bytes
               4 Dir(s)  160 491 565 056 bytes free

C:\Users\Jon-Andre\workspace\VirtualWall\stedr\build\android\bin>adb install -r
stedr.apk
```

Structure in Titanium alloy: if you explore your newly added Stedr project in the project explorer you will notice that under the app folder there are a number of subfolders.



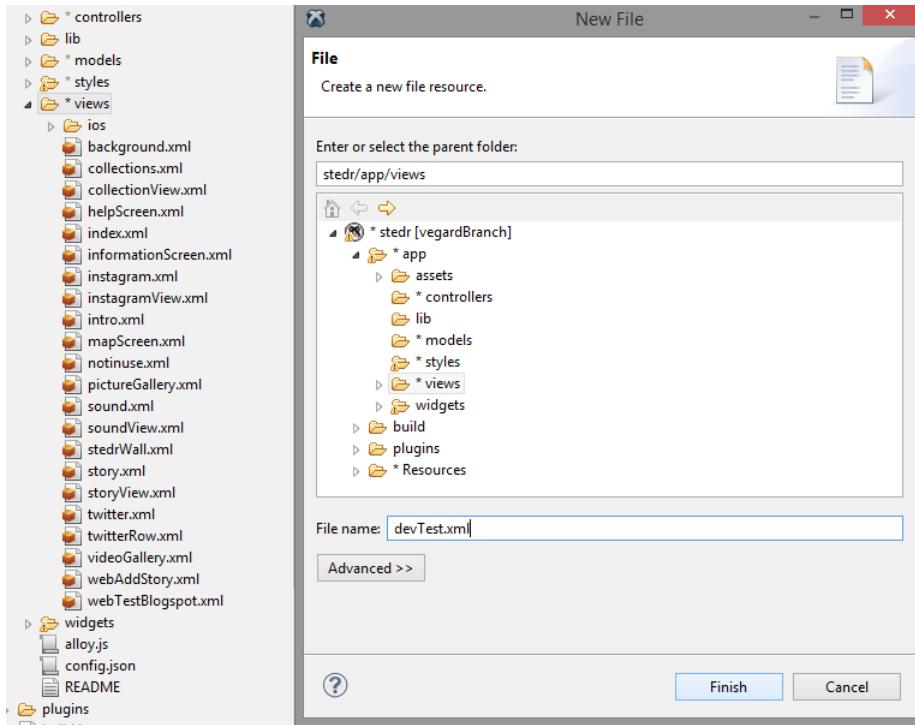
functionality and design.

Additionally we have the assets folder, where media like pictures are stored; the lib folder, where you store relevant imported libraries; the models folder, where you can create models that are useful when for instance converting a string into an object; the widgets folder, where imported pre-made UI-elements are stored.

Coding Example In this example we assume that you have completed the back-end example and uploaded it to the server stedr-beta.herokuapp.com.

We will be creating a view that contains a label with the value that we retrieve from the server stedr-beta.herokuapp.com/hello.json?name=world. If done right, the text "Hello World" should be displayed on the screen when opening this view.

1. Begin by creating a new devTest.xml-file in the "views"-folder, a devTest.tss-file in the "styles"-folder and a devTest.js-file in the "controllers"-folder.



2. Creating the XML-file for the view. This is the basic structure of the UI elements. You can also add some properties to the different elements.

```
*devTest.xml *
1<Alloy>
2    <!-- This is our main view. You can have multiple views inside another view.
3    In this example we will just make a simple view that fills the entire screen and has a white background -->
4    <View id="devTest" class="container" width="100%" height="100%" backgroundColor="white">
5        <!-- Here we can add elements we want inside our main view devTest.
6        For this example we just want a label -->
7        <Label id="testLabel"/>
8    </View> <!-- This closes out the view. -->
9</Alloy>
```

2. Creating the TSS-file for the view. The purpose of this file is mainly to stylize UI-elements that are used more than once in the XML-file. Even though a TSS-file is not necessary for a simple view like this and can be done easily in either the XML- or the JavaScript-file example, here is how you would stylize the label using TSS:

```
*devTest.xml *devTest.tss *
1 //This stylizes every element in the XML-file with the ID "testLabel"
2 "#testLabel" : {
3     color: "black",
4     font : {
5         fontFamily: 'Helvetica',
6         fontSize : '36dp'
7     }
8 }
```

3. Creating the JavaScript-file. This is where we retrieve the information from the server, as well as set the text of the label.

```

1 // We want to retrieve the information stored in "stedr-beta.herokuapp.com/hello.json?name=world" and add it to a the testLabel
2 // "world" is the parameter
3 var parameter= "world";
4 var url = "stedr-beta.herokuapp.com/hello.json?name="+parameter;
5
6 var client = Ti.Network.createHTTPClient({
7     onload: function() {
8         var response=this.responseText;
9         // This is just to see the information in the console aswell
10        Ti.API.info("Received text: " + response);
11        // Here we set the text of the testLabel to the resone text
12        $testLabel.setText(response);
13    },
14    // function called when an error occurs, including a timeout
15    onerror : function(e) {
16        Ti.API.debug(e.error);
17    },
18    timeout : 5000 // in milliseconds
19 });
20
21 // Prepare the connection.
22 client.open("GET", url);
23 // Send the request.
24 client.send();

```

Now all we have to do is open this view where we want it. For test purposes i will in this example set it to open automatically when you open the app. To do this, we have to go into the index.js file under controllers. Every window/view opens or closes through this class because this is where the window stack is. To make sure the test view is shown automatically, scroll down to the bottom of index.js and add it right above \$index.open() as shown below:

```

193     //sets the opened variable to "true"
194     Ti.App.Properties.setString("appLaunch", JSON.stringify({opened:true}));
195 };
196
197 // First you create a controller for the 'devTest' view
198 var devTest = Alloy.createController('devTest').getView();
199 // Then you add the view to the view-stack by calling the "addWindow" function
200 addWindow(devTest, "test");
201
202 $index.open();
203
204 // Decides if you are allowed to flip the view using phone orientation etc.
205 $index.orientationModes = [Titanium.UI.PORTRAIT, Titanium.UI.UPSIDE_PORTRAIT];
206

```

Now if you save and run you should see something like this when you start the application on your phone:



Hello world

We have now gone over the basics of how to create a view as well as retrieving information from the server. To further understand how to develop Stedr using Titanium, looking at and understanding the previous code is essential. The documentation available on the Appcelerator website is also extremely helpful when learning about Titanium:

B | Test log

Test ID	T-F7
Test Responsible	Tor Barstad
Tester	Jon-Andre Brurberg
Date and time	25.05.2014, 11:00
Preconditions	Satisfied
Properties	The reasoning behind this test, is to check the gallery function related to stories are working. Also this gives a quick overview over the story and the layout.
Steps	<ol style="list-style-type: none">1. The tester clicks on a story title from the place to enter the story screen.2. The screen appears, displaying an image. The user swipes to the side resulting with the image being replaced with the next in the gallery on screen.3. Steps 1 and 2 were repeated on different stories and places to check for consistency.
Result	Pass
Validity	Pass
Comments	There are not implemented arrows as described in the text but swiping or clicking to the side will display more pictures, this feature is very intuitive making the arrows obsolete.

Table 29 – Example Test Log: T-F7

C | Status Report Example

C.1 Status Report

Summary Status Report

1. Introduction

The last week has been pretty busy with a lot of progress in terms of development. The last customer meeting turned out to be very constructive and the customer seemed to be satisfied with the recent development of the system.

2. Progress summary

Updated Activity Plan (8)

Milestone/Gantt

3. Open / Closed problems

The last week has almost been problem free, and the progress has been pretty good. Regarding the continued development there has been a delay, because of some content which have not been made available to the group from the customer.

In the following week it will be difficult to keep up the momentum from last week, as Easter is approaching (unavailability because of travelling) and a couple of the group members are in China for a school excursion. The group members that still are available will therefore focus on few assignments which they are familiar with from earlier.

4. Planned work for next period

New Activity Plan (9)

5. Updated risk analysis

Risk analysis

C.2 Activity Plan

ID	Package	Activity	Resources	Planned work (hrs)	Start	Finish	Actual Work (hrs)	Status (% completed)	Comment
1	Development	Sound recording	Øyvind	15	10.03.2014	23.03.2014	15	50	
2	Systems Engineering	New communication standard	Tor	20	10.03.2014	23.03.2014		10	60
3	Systems Engineering	Collections, possibilities	Vegard	5	10.03.2014	23.03.2014		10	30
4	Development	Integrate new DF-API	Jon-Andre	10	05.03.2014	23.03.2014	9	75	
5	Development	Soundcloud integration	Vegard	10	10.03.2014	23.03.2014	7	75	
6	Project Management	Customer Meeting	All	18	19.03.2014	19.03.2014	0	Canceled	
7	Project Management	Presentation	Hallvard	5	19.03.2014	19.03.2014	20	100	
8	Management	Status Report	Vegard	1	23.03.2014	23.03.2014	2	100	
9	Project Management	Supervisor Meeting	All	3	17.03.2014	17.03.2014	4	100	
10	Project Management	Peer evaluation	All	20	17.03.2014	23.03.2014	25	100	

C.3 Risk Analysis

Problem	Description	Likelihood (1-9)	Impact (1-9)	Importance (Likelihood * Impact)	Preventive action	Remedial Action	
Lack of abstraction in product	The existing solution does not have possibilities to add features following templates	6	9	54	None	Standardize communication, and also add a standard for content	
Communication Loss	The groups' communication with each other is not satisfactory. Group don't establish good communication with the customer and supervisor.	3	7	21	Actively establish communication and reach out to the parties regularly.	Talk with the group about the communication, and try to get a good understanding of the problem. Establish communication media, so the group can talk with each other.	
Change requests	Change requests that does not meet the requirements of the product	3	7	21	Have well defined requirements specification and implement it iteratively.	Reach out to the customer and ask what they think about the request changes.	
Technical difficulties	Some problems may turn up to be very hard to solve. This can in turn lead to delays and frustration. And may sometimes be very time consuming.	5	4	20	Regularly have technical discussions with the group, that way the hard problems can be handled by the group as a whole.	If the problem is to hard, try to get help from other groups. Also evaluate if the problem can be handled differently.	
Workstation are noisy	The workstation is filled with people who make a lot of sound, so the developers team can't concentrate to the fullest.	5	4	20	Can preorder room, so we get our own workstation to work on.	Order room/move to a private facility, and move the whole developers team there if the noise are that bad.	
Failing to do planned work	Members of the group fails to do scheduled work due to falling behind in other subjects or matters not related to the project.	9	2	18	Good scheduling habits. Sit down every week and see what's planned to do in the project the following week. Coordinate against what you have to do in other subjects.	Make up for lost work during weekends or other available time slots	
Insufficient product	Developing a product that does not meet the requirements of the customer	2	9	18	Good and continuous communication with the customer. Work with an agile development structure such as Scrum		
API change	Changing the general API because lack of functionality.	2	9	18	Sufficient research about API before implementing it into the project.	Either drop the functionality that is missing, or start developing with the new API if there's time and the priority is high enough.	
Different app views	Customer and developers have different views of the apps' purpose and functions.	3	6	18	Have regular meetings, inform and discuss all changes to project scope, goals and features.	Discuss with customer and find middle ground.	
Scope	The amount of features requested are beyond what the development team can deliver in time	6	3	18	Be specific with the customer how much time we have, and explain deeply how much time it takes to develop a single feature	Discuss what are the necessary features that must be in the product, and flush out what is the least necessary.	
Unavailability	Group members are unavailable because of different projects, which makes it difficult to complete the planned work load	4	4	16	Get an overview over when group members are available, and then plan how to either work more before the time of the unavailability or plan to catch up the missed work at a later stage	Other group members have to make up for others unavailability	

Problem	Description	Likelihood (1-9)	Impact (1-9)	Importance (Likelihood * Impact)	Preventive action	Remedial Action	
Lack of competence	The developers don't have enough competence about the given software the project requires.	8	2	16	Meet every day, do workgroups together and learn by failing.	Talk with other members of the group, and hear if they have the competence. This will prevent hours of searching, when you can listen what the other members have to say. And direct you on the right path for the competence you need.	
Hardware communication	Not possible to use the dev. environment to make use of some of the hardware (camera, microphone)	4	4	16	Try to keep the application relevant to what it's actually meant to do, focus on the core areas.	Access the hardware externally through APIs	
Loss of work (DUPLICATE)	Disk failures or losing equipment that causes project related work to disappear	2	8	16	Establish good backup habits. Have the group share the code (using git etc) and use cloud services for document storage	Talk with other group members, hear if they have it on a local hard drive. If not rollback to the latest stable version, where there is least of loss.	
Software issues	Not all group members can install necessary software properly on their own devices. Software not functioning properly on device.	5	3	15	Research software before taking use of it. Install software together as a group.	Work together in small groups with the task, or reassign another work to the individual.	
Missing deadlines	Some work may take longer time than expected, this may cause delays later on in the project.	3	5	15	Have a steady and disciplined workflow and plan ahead. Overestimate work rather than underestimate.	All members meet and plan what is to be done, and assign it right away. This way the project can be delivered as soon as possible.	
Customer turnover disruption	A key contact in SINTEF leaves the company, putting the project in a unclear state	2	7	14	Good communication. Multiple contacts with knowledge of the project	Quickly contact the customer and discuss how to proceed and how it's affected	
Sickness	Group members or other crucial personell gets sick	4	3	12	Have regular updates about the progress of the work being done, and don't make important task rely completely on one person without a backup plan. Stay healthy.	Talk to the person about the individual tasks, how much he can handle, and distribute the work the member can't complete.	
Group members falling out.	Members doesn't show for meetings, or goes of the grid without notice.	2	6	12	Good communication and agree on a schedule that suits everyone.	Take action at once, and make inquiries to why the member didn't show.	
Uneven workload	Uneven distribution of workload	6	2	12	Stay updated on the tasks given and work put in, then distribute new work accordingly.	Have intervention and discuss the workload, try to redistribute to make up for the differences.	
Conflict over changes	Group members not in agreement over supposed changes in group management, work, responsibility etc.	3	4	12	Have an open dialog.	Discuss in group and decide as a democracy.	
Late for meeting	Members of the group are late for meetings with group/customer and supervisor	6	2	12	Good communication and agree on a schedule that suits everyone.	Take action at once, make inquiries to why the member came late and make it clear to the individual that this is not acceptable.	

Problem	Description	Likelihood (1-9)	Impact (1-9)	Importance (Likelihood * Impact)	Preventive action	Remedial Action	
Documents customer/supervisor meeting	Lacking the sufficient documents for the meeting with the customer used for presenting changes, mockups or reports about fieldwork etc.	2	6	12	Have the documents stored in the cloud so you can access it where ever you go. With your respective smartphone/tablet and pc's,	Discuss what you remember and try to make the best out of the meeting.	
Equipment failure	Computers and other dependable devices malfunctions.	4	2	12	Keep documents and code in the cloud so you can work from another device if your primary device malfunction.	Get replacement as soon as possible.	
Application on mobile device	Problems installing application from the used framework on mobile devices.	3	4	12	Make sufficient research about the framework we are suppose to use in front of the project. So we have as little problems as possible to install on mobile devices.	Use google and try to find people who have had the same problems as you are having. And try to do the same as they have done before you.	
Document sharing failed	Authorization of documents sharing is not complete, people don't have access to the groups documents.	2	4	8	Give all the authorization they need for the documents to be shared. So all can view, edit and share documents.	Find out where the problem lies, so everyone can get authorization for the given documents and folders.	
Lack of software	Lack of software necessary for the development process.	1	3	3	Talk about what software is required for the development of the product. Ask the customer for this software or funds to acquire it in good time.	Ask the customer immediately for the required software, so the development progress don't have any major delays.	

D | Meeting Example

D.1 Group Meeting

Here follows an example of notes one of our meetings. The summary from the meetings are written in Norwegian, and translating them for the report was not something we prioritized.

Til stede: Alle

Agenda:

- Oppsummering fra forrige gang
- Gjort siden sist
- Evaluering av stedr
- Preliminary report
- Diverse
- Til neste gang

Oppsummering fra forrige gang:

Forrige gang ble vi enige om å i hovedsak se nærmere på 'stedr' og evaluere appen. Siden sist har vi også hatt møte med supervisor og Sintef, det foreligger ikke noe referat fra Sintef-møtet enda. På grunn av sykdom var dette møtet med Babak og ikke den opprinnelige kunden Jacqueline.

Gjort siden sist:

Øyvind: Mock-up, og alt fra lista.

Hallvard: Titanium, APIer, RISK.

Jon-Andre: Titanium og oppsett mot stedr, skrevet en liten evaluering av stedr, agilefant, PHP/symfony, jobbet litt (for lite) på RISK-dokumentet.

Tor: Mange oppgaver viste seg overflødige, kommer tilbake

Vegard: Titanium, LaTeX, sharedLaTeX.

Jørgen: God evaluering av stedr, RISK

Evaluering av stedr: Diverse UI-bugs. F. eks kan man ikke rotere telefonen. Ikke noen scroll-funksjon på 'pictures'. I overkant mye scroll enkelte steder, man kan scrolle forbi slutten av teksten. Twitter tillater ikke mer enn 140 ord, men det gjør appen. Er det noe poeng å tweete fra stedr, eller skal man bare hente inn? Jørgen har prøvd å få inn et bilde fra instagram, men dette dukker ikke opp i stedr etter 12 timer. Måten stedr henter inn historier (vha. hashtags) gjør at det blir mye irrelevant informasjon. Autentisering opp mot Twitter er rart, hva er stedr homepage. Vi liker mye av designet. Hva er identiteten/poenget til appen? For utdypninger se eget dokument.

Preliminary report:

Risk-list er nesten ferdig. Jørgen, Jon og Øyvind skal møtes på lørdag for å jobbe og fullføre midterm.

Diverse:

For nå tar vi det med ro i LaTeX, bruker Google Docs i første omgang. Tor og Hallvard har sett litt på skytjenester (sky-backend), noe som kan virke interessant.

Til neste gang:

Øyvind: Prelim-rapport, fikse kodekopier

Hallvard: APIer, balsamering

Jon-Andre: Prelim-rapport, referat fra Sintef

Tor: Se på skybackend

Vegard: APIer, prøve å få kildekoden til å fungere i Titanium

Jørgen: Prelim-rapport

D.2 Customer Meeting

Here is an example of the summary of a meeting with our customer. Again the text is in Norwegian.

Videomøte med Jacqueline 19.02.2014

Til stede fra gruppe: Øyvind, Jørgen og Jon-Andre
Fra kunde: Jacqueline Floch

Gruppa har fått mye informasjon, men hva synes Sintef er viktigst?

1. APIet til Digitalt Fortalt
2. Stable bilder til samme sted.
3. Gjøre appen bedre og forbedre integrasjonen mot eksisterende APIer.
4. Skape interesse gjennom sosiale medier (link til fortelling f. eks)
5. Filtrering
6. Koble til andre databaser (Soundcloud)

Vil beholde så mye som mulig av det som allerede finnes i appen. Det ble brukt mye tid på design i høst, så dette bør ikke prioriteres nå.

Filtrering: Når et sted blir hentet kan man f. eks få en liste med tags. Filtrering basert på brukerprofil (og generelt) vil for øyeblikket være litt problematisk grunnet få fortellinger.

DF har link til Wikipedia.

Hvis vi har forslag til forandring både backend/frontend så kan dette gjennomføres.

Informerer om den lille brukerundersøkelsen vi har hatt, og at vi har et inntrykk av at den sosiale delen.

Flickr API: Et sted er definert som et bilde, geolokasjon, bilde er delt innenfor en gruppe. Gruppe-APIet til Flickr er ikke optimalt, gruppe er brukt for å gjøre søket enklere. Backend henter alle bilder og informasjon fra den gruppa.

Gruppa har et inntrykk av at appen ved førstegangsbruk er litt vanskelig, kanskje det bør være innlagt en sidemeny.

En litt abstrakt utfordring er: Hva er et sted, og hvor stort er et sted?

Notiser:

- Det APIet som ble brukt i høst fungerer ikke nå
- Gruppa bør sjekke ut Trondheim Byguide

D.3 Supervisor Meeting

Here are some notes from one of our meetings with the our supervisor Mohsen Anvaari. We found these meetings to be a great resource during our project, giving us constructive criticism and advice. This specific meeting occurred 18.02.2014.

Meeting with Supervisor 18.02.2014

He thought the report was generally good, but some small things were missing:

In the introduction we should have given a short introduction to the customer.

He had some issues with the structure of the report:

The term Software Engineering is too broad. Should rather be split up in sections like Architecture, Design etc. An alternative would be to simply split the structure in to Sprint 1, 2 etc.

Time organization should just be called Project Planning.

Never use "things" in the report. Be more specific.

"GUI and APIs" - For what? Specify that we will work on Stedrs GUI and API.

Remember to give an brief description on what the application is for. Explain the usage.

Process is fine.

Time-plan and architecture is fine, but needs to be more detailed for the next version.

Some of the Non-functional-reqs is functional. ISO standard. Chose 3 or 4. How we tackled it later. Diary is functional.

Linking is superimportant!

Risk analysis is very good.

E | Other

E.1 Mock-ups

In the beginning of the project we did made mockups to describe our ideas and thoughts. These was later dismissed as the project progressed, but we implemented ideas from them later on, so even though they are not directly relevant to the final project, we feel they are relevant enough for the subject, in terms of how time were spent etc, to include them in the report.

