Virtual Walls - Walls that tell us stories

Odd Fredrik Rogstad
Christian Frøystad
Simon Stastny
Knut Nergård

Abstract

> The assignment relates to the EU-IST project TAG CLOUD. TAG CLOUD will develop innovative digital solutions with the aim to increase the engagement of people in culture and cultural heritage. The main aspects that will be investigated are: new interaction interfaces with cultural artefacts, user as a contributor and personalized information. In Norway, the focus is "culture in the landscape", i.e. the discovery of cultural memories that we meet when walking around in cities, in villages and in the nature. [1, p. 47]

The aim of this particular project is to develop a concept and a prototype that will make information about cultural heritage more accessible and fun! This will be accomplished by creating a mobile application that lets users consume content from cultural databases in an exploratory format.

Contents

Abstract **iii**

1 Introduction **1**

2 Project plan **3**
2.1 Project directive . . . 3
2.1.1 Project name . . . 3
2.1.2 Project sponsor . . . 3
2.1.3 Partners . . . 3
2.1.4 Background for the project . . . 4
2.1.5 Project goal . . . 4
2.2 Tools . . . 5
2.2.1 Hardware . . . 5
2.2.2 Software . . . 5
2.2.3 Collaboration and communication . . . 5
2.2.4 Allocated time . . . 7
2.3 Detailed plan . . . 7
2.3.1 Phases . . . 7
2.3.2 Activities . . . 8
2.3.3 Milestones . . . 8
2.3.4 Gantt diagram . . . 9
2.4 Risks . . . 9
2.5 Project organization . . . 11
2.5.1 Customer contact . . . 12
2.5.2 Scrum master . . . 12
2.5.3 Secretary . . . 12
2.5.4 QA manager . . . 12
2.5.5 Test leader . . . 12
2.5.6 Project manager . . . 12
2.6 Quality Assurance . . . 12
2.6.1 Routines . . . 13
2.6.2 Customer meeting . . . 13
2.6.3 Advisor meeting . . . 13
2.6.4 Group meeting . . . 14
2.7 Templates and procedures . . . 14
2.7.1 Document templates . . . 14
2.7.2 Code standards . . . 14
2.7.3 Version control procedures . . . 22

3 Preliminary study **23**
3.1 Current situation . . . 23
3.1.1 Cultural Heritage today . . . 23
3.1.2 Existing systems . . . 24
3.1.3 Edge dominant systems . . . 28
3.2 Existing mobile applications . . . 29
3.2.1 Kulturminnesøk app . . . 29
3.2.2 Kulturarv app . . . 31
3.2.3 1001 Stories of Denmark . . . 32
3.2.4 Conclusion . . . 34
3.3 Work methodology . . . 34
3.3.1 Waterfall . . . 35
3.3.2 SCRUM . . . 35
3.3.3 Chosen work methodology . . . 36

4 Requirements **37**
4.1 Introduction . . . 37
4.1.1 Purpose . . . 37
4.1.2 Scope . . . 37
4.1.3 Definitions, acronyms and abbreviations . . . 38
4.1.4 Overview . . . 38
4.2 Overall description . . . 38
4.2.1 Product perspective . . . 38
4.2.2 User Interfaces . . . 39
4.2.3 Software interfaces . . . 40
4.2.4 Communication interfaces . . . 40
4.2.5 Product functions . . . 41
4.2.6 User characteristics . . . 41
4.2.7 Constraints . . . 41
4.3 Specific requirements . . . 41
4.3.1 External interfaces . . . 41
4.3.2 Functions . . . 42
4.3.3 Software system attributes . . . 43
4.4 Use case diagrams . . . 44

5 Architecture **47**
5.1 Architectural drivers . . . 47
5.1.1 Maintainability . . . 47
5.1.2 Portability . . . 47
5.1.3 Division between logic and presentation . . . 47
5.1.4 Short development time . . . 47
5.2 Stakeholders and concerns . . . 47
5.2.1 Developers . . . 47
5.2.2 Customer . . . 48
5.2.3 End users . . . 48
5.2.4 Course staff . . . 48
5.3 Selection of Architectural views . . . 48
5.3.1 Logical view . . . 48
5.3.2 Process view . . . 48
5.3.3 Development view . . . 48

5.4 Architectural Tactics . . . 49
5.4.1 Modifiability . . . 49
5.4.2 Usability . . . 49
5.4.3 Testability . . . 49
5.5 Architecture and design patterns . . . 49
5.5.1 Client . . . 50
5.5.2 Server . . . 50
5.6 Views . . . 50
5.6.1 Server . . . 50
5.6.2 Client . . . 51
5.7 Architectural rationale . . . 51

6 Testing **53**
6.1 Test plan . . . 53
6.1.1 Test plan identifier . . . 53
6.1.2 Introduction . . . 53
6.1.3 Test items . . . 53
6.1.4 Features to be tested . . . 54
6.1.5 Features not to be tested . . . 54
6.1.6 Approach . . . 54
6.1.7 Item pass and fail criteria . . . 54
6.1.8 Test deliverables . . . 54
6.1.9 Environmental needs . . . 54
6.1.10 Responsibles . . . 55
6.1.11 Staffing and training needs . . . 55
6.1.12 Schedule . . . 56
6.1.13 Risks and contingencies . . . 56
6.1.14 Approvals . . . 56
6.2 Test cases . . . 56
6.3 Test results and evaluation . . . 56

7 Phases **57**
7.1 Planning . . . 57
7.2 Sprint 1 . . . 57
7.2.1 Introduction . . . 57
7.2.2 Requirements . . . 57
7.2.3 Pre study . . . 57
7.2.4 Implementation . . . 63
7.2.5 Testing . . . 63
7.2.6 Review . . . 63
7.3 Sprint 2 . . . 63
7.3.1 Introduction . . . 63
7.3.2 Sprint backlog . . . 63
7.3.3 Requirements . . . 63
7.3.4 Pre study . . . 63
7.3.5 Implementation . . . 65
7.3.6 Testing . . . 66
7.3.7 Review . . . 66
7.4 Sprint 3 . . . 67
7.4.1 Introduction . . . 67

7.4.2 Requirements . . . 67
7.4.3 Pre study . . . 67
7.4.4 Implementation . . . 68
7.4.5 Testing . . . 69
7.4.6 Review . . . 69
7.4.7 Phase evaluation . . . 69
7.5 Sprint 4 . . . 69
7.5.1 Introduction . . . 69
7.5.2 Requirements . . . 69
7.5.3 Pre study . . . 69
7.5.4 Implementation . . . 69
7.5.5 Testing . . . 69
7.5.6 Review . . . 69
7.5.7 Phase evaluation . . . 69
7.6 Sprint 5 . . . 69
7.6.1 Introduction . . . 69
7.6.2 Requirements . . . 69
7.6.3 Pre study . . . 69
7.6.4 Implementation . . . 69
7.6.5 Testing . . . 69
7.6.6 Review . . . 69
7.6.7 Phase evaluation . . . 69
7.7 Finalization . . . 69

8 Evaluation **71**

A Assignment **75**

B Timesheets **77**

C Documentation **79**

List of Figures

2.1 Screenshot of Trello 6
2.2 Overview of project 9

3.1 Kulturminnesøk, Norway 24
3.2 Kulturminnesøk, Trondheim 25
3.3 Kulturminnesøk, info display 25
3.4 Kulturminnesøk, more info 26
3.5 Digitalt fortalt, front page 26
3.6 Digitalt fortalt, search result for Trondheim 27
3.7 Digitalt fortalt, Regalierommet, with media slider and text 28
3.8 The metropolis 28
3.9 Kulturminnesøk, map function 30
3.10 Kulturminnesøk, list function 30
3.11 Kulturminnesøk, augmented reality function 31
3.12 Kulturarv, map function 31
3.13 Kulturarv, augmented reality function 32
3.14 1001 Stories of Denmark, list function 32
3.15 1001 Stories of Denmark, map function 33
3.16 1001 Stories of Denmark, a sight 33
3.17 1001 Stories of Denmark,timeline function 34
3.18 The Waterfall process [10] 35
3.19 The SCRUM process 36

4.1 Overview of interaction 39
4.2 Use case diagram for Users 44
4.3 Use case diagram for Walls 44
4.4 Use case diagram for Stories 45

5.1 Server - Logical view 50
5.2 Server - Process view 51
5.3 Server - Development view 51

7.1 Menus, dedicated vs. hidden side-menu 60
7.2 Temperature pins, with filtering 60
7.3 Simple wall, with tabs to navigate. Nidarosdomen is used as an example 61
7.4 Wall using multiple data sources, with slide gestures to navigate. Nidarosdomen is used as an example 62
7.5 Mercator cylindrical map projection basics. [29] 67

List of Tables

2.1 Time sheet per phase 9
2.2 R1 9
2.3 R2 10
2.4 R3 10
2.5 R4 10
2.6 R5 11
2.7 R6 11
2.8 Group organization 11

4.1 Definitions, acronyms and abbreviations 38
4.2 Users 42
4.3 Walls 42
4.4 Users 43

6.1 Roles allocation for testing tasks. 55
6.2 Roles allocation for testing tasks. 55
6.3 Test tasks schedule 56
6.4 Test tasks approvals 56

7.1 Backlog for sprint 2. 63
7.2 API for Sync in Alloy[25] 65

Chapter 1

Introduction

This report describes the planning and execution of the project "Virtual Walls - Walls that tell us stories" (later in this report only referred to as Virtual Walls or the system) on behalf of eTrøndelag and Sintef ICT.

The aim of the project is to communicate and engage people in cultural heritage. The assigned project is part of the course TDT4290 - Customer driven project at NTNU, and also an ongoing project at Sintef ICT on behalf of eTrøndelag and the EU initiated project TAG-CLOUD.

Today most places of cultural or historical significance is mapped and has a rather large amount of connected data. This information might be hard to gain access to, written in a form more suited for scholars than the average man, or simply not written down at all.

The system will help the user discover, consume and share information about places, buildings, persons, events, and other historically significant information. The user can add new information, improve existing information, comment on information or walls, make new walls and share a wall on social media.

Both the walls and the information will be categorized using taxonomy so that the user might filter the sort of information he is looking for.

The goal of the project is to make information about cultural heritage more accessible, understandable and fun for the common man.

Chapter 2

Project plan

This chapter will give an overview of how the work is to be structured and the most important milestones.

2.1 Project directive

The project directive describes the background and mandate of the project.

2.1.1 Project name

The customer has given the project the name “Virtual Walls - walls that tells us stories”. The project will, in this report, be referred to as “Virtual Walls” or the system. The name is derived from “Væggen” in Copenhagen which was the inspiration of this project.

2.1.2 Project sponsor

The sponsor of the project is eTrøndelag together with SINTEF ICT. eTrøndelag is the division at Sør-Trøndelag county that focuses on ICT. Their responsibility lies in ICT strategy, digital communication and cooperation, infrastructure and digital cultural development. The division has four advisors.

SINTEF ICT is the ICT unit at SINTEF, which is the largest independent research organization in Scandinavia. SINTEF is a non-commercial organization which means that all the profits goes into new research, equipment and development of staff. The organization has about 2100 employees.

2.1.3 Partners

Customer

Name: Eistein Guldseth
Role: Advisor Digital communication and Interaction, STFK
Mobile: +47 414 74 826
E-mail: eistein.guldseth@stfk.no

Name: Jacqueline Floch
Role: Senior scientist at SINTEF ICT

Mobile: +47 930 08 536
Fax: +47 73 59 43 02
E-mail: jacqueline.floch@sintef.no

Advisor

Name: Reidar Conradi
Role: Professor, NTNU
Mobile: +47 91 89 70 29
Phone: +47 73 59 34 44
E-mail: conradi@idi.ntnu.no

Group

Name: Odd Fredrik Rogstad
Mobile: +47 99 10 73 27
E-mail: oddfredrikrogstad@gmail.com

Name: Christian Frøystad
Mobile: +47 45 21 70 66
E-mail: chrisfro@stud.ntnu.no

Name: Simon Stastny
Mobile: +47 45 16 62 98
E-mail: simonst@stud.ntnu.no

Name: Knut Nergård
Mobile: +47 41 25 67 20
E-mail: knut.nerga@gmail.com

2.1.4 Background for the project

Today, if you want information about cultural heritage you will have to access many different sources, and the information might be less than accessible. The joint European project TAG CLOUD wishes to change this, and also bring together content provided by experts and individuals. TAG CLOUD is to develop innovative digital solutions to engage people in cultural heritage by making the information more accessible and involving the user in the content production.

In connection with this project eTrøndelag has decided, together with SINTEF ICT, to explore the possibility of using Virtual Walls to spread information about cultural heritage in Trondheim.

During the last 5 - 6 years, smart phones has become more and more common. In the later years mobile Internet has also become somewhat common. This has made this project possible, as the users do not need to carry an ordinary computer around.

2.1.5 Project goal

The goal of the project is producing a prototype of the system that might be experimented with and tested in different contexts. The prototype should be accompanied by excellent documentation, both for users and developers.

2.2 Tools

This section describes the tools used to communicate, develop and implement the system.

2.2.1 Hardware

2.2.2 Software

2.2.3 Collaboration and communication

This section describes the tools used to collaborate and communicate throughout the project.

Google Docs

Google docs is a freeware web-based office suite that enables multiple members to collaborate on documents, spreadsheets, presentations, drawings etc. from any computer with Internet access. Google docs uses Google Drive, which is a file storage and synchronization service. That means that two or more persons can work on the same document simultaneously, at different geographical places.

We chose to use Google Docs because everyone already had used this with great success on previous projects. It is also fairly simple to use, so no training were needed. Since Google Docs has some limitations regarding appearance and navigation in documents, we used LaTeX to produce the final document.

LaTeX

LaTeX is a document markup language and document preparation system for the TeX typesetting program. LaTeX enables us to make large, professional looking documents without all the hassle regarding table of content, placing of pictures and diagrams, cross-references, making tables etc..

Some group members had some experience with the language, but everyone else was eager to learn and try it out.

Dropbox

Is a file hosting service that offers cloud storage and file synchronization, much like Google Drive. We only used Dropbox as an collaboration tool with the customer, as in sharing files, etc.. We found that Google Drive, with the possibility to use the embedded office suite, Google Docs, were the best alternative as an collaboration tool for document writing.

Facebook

Facebook is an online social networking service used by 1 110 million users worldwide [investor.fb.com/releasedetail.cfm?ReleaseID=761090]. We started a private group, "TDT4290 - CDP - Virtual Wall", and used this to everything from planning meetings, having scrum stand ups to discuss about the project.

At first we had some problems with time-sensitive matters, such as revoking meetings, where some did not see the revoking post on facebook. Therefore we started using email for such matters.

Email

Email were used for communicating with both the customer and advisor. Within the group it was used for time sensitive matters.

GitHub

GitHub is a web-based hosting service for software development project that use the Git revision control system, which is a distributed revision control tool (DRCS). Since Git is distributed every developer has their own local repository, independent of the centralized repository, that allows developer to work in offline mode and each of the local repositories acts as a backup. This was used during the implementation of the system.

Everyone had used GitHub on other projects, but almost every member needed a refresher

Trello

Trello is a free web-based project management application that uses the paradigm kanaban. Kanaban, a Japanese word that roughly mens "card" or "signboard", is a method made famous by the car manufacturer Toyota in the 1980s for supply management.

Kanaban is fairly simple, a project is represented by a board, the board contains lists, that corresponds to a workflow. Lists contains cards that represents tasks.

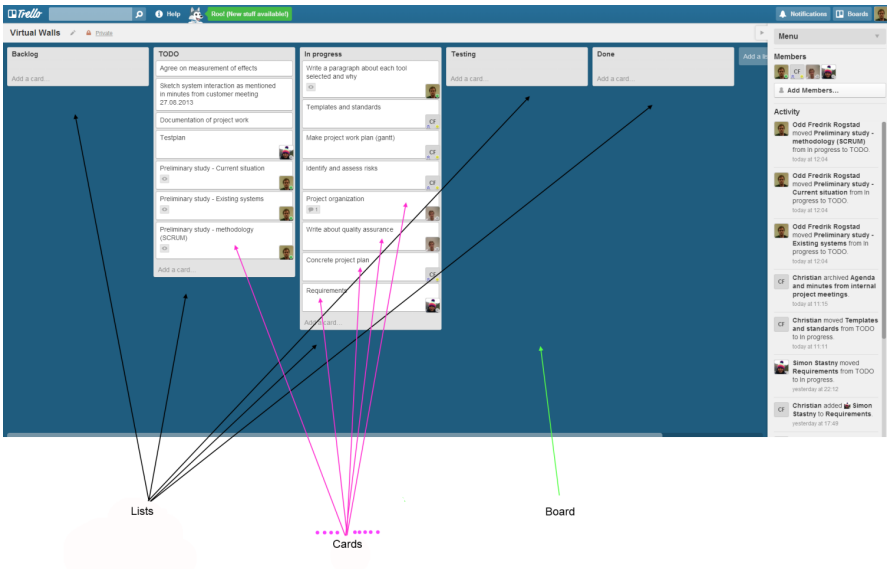

Figure 2.1: Screenshot of Trello

Figure 2.1 shows how we used it for document work. There are several lists; "TODO", "In Progress", "Testing" and "Done", which represents the workflow. We started by populating the "TODO"-list, with all the appendices that were needed. Then we divided the different tasks between the group members, and started working. When a person worked on a task, then the corresponding card would be in the "In progress" list, when the task were finished, it was moved to the "Testing" list, for quality assurance. If it was approved, it was moved to "Done" list, if not, back in the "In Progress" list.

2.2.4 Allocated time

The course the project is part of amounts to 15 points of credit, which indicates that each member of the group should work for about 25 hours on the project each week. Given that the group consists of 4 members, and that the project run over 13 weeks, we get a total of allocated time that amounts to 25 x 4 x 13 = 1300 hours.

Due to the diversity of the group, one in a full time job, the others attending different classes, the group has few meetings in person each week. As such, every group member is free to plan his own work each week, but has to meet at the weekly group meeting. As a result of this the group has only a few items on its weekly agenda: internal group meeting, meeting with advisor and meeting with customer.

2.3 Detailed plan

The goal of this chapter is to indicate how the project was executed. It was written as a guide on beforehand, but has been revised during the project lifetime in order to provide a better level of detail.

2.3.1 Phases

Here follows a short summary of the contents, duration and expected outcome of each phase.

Planning and preliminary study

Duration: 21.08.2013 - 11.09.2013
Expected outcome: Preliminary first chapters of report, requirements and test plan.

Sprint 1

Duration: 11.09.2013 - 25.09.2013
Expected outcome: Architectural documentation, refined requirements and test plan, and minimal working prototype. Complete phase document.

Sprint 2

Duration: 25.09.2013 - 09.10.2013
Expected outcome: Refined requirements and test plan, working prototype with developer documentation. Also complete phase document

Sprint 3

Duration: 09.10.2013 - 23.10.2013
Expected outcome: Refined requirements and test plan, working prototype with even better and more complete developer documentation. Complete phase document.

Sprint 4

Duration: 23.10.2013 - 06.10.2013
Expected outcome: Refined requirements and test plan, working prototype with excellent developer documentation.

Finalization

Duration: 06.11.2013 - 21.11.2013
Expected outcome: Excellent project report, presentation and prototype accompanied by user and developer documentation. Phase focuses primarily on finishing report and presentation, but also polishing the documentation.

2.3.2 Activities

Every week some repeating meetings will take place.

Customer meeting

Each week a status is given to the customer, as well as clarifying anything being unclear at that time. Backlog is prioritized every second week before the start of the next sprint.

Advisor meeting

This meeting is to report status and get valuable feedback and advice from the advisor.

Internal group meeting

Since it is not possible for the group to work together during normal work hours, this internal group meeting serves as the planning meeting for the coming week.

2.3.3 Milestones

Pre-delivery of report

Due: 14.10.2013 (08.10.2013 electronically to advisor)
Delivery of the chapters Abstract, Introduction, Pre-study and the one containing choice of development model. The outline of the report should also be finalized in the table of contents. The outline does not need to be complete, but should give a good indication as to what each sections is to discuss.

Production of reports

Due: 20.11.2013
The report is to be copied and bound in four copies no later than 20.11.2013. Preferably earlier.

Final delivery and presentation

Due: 21.11.2013
Presentation for customer, advisor and censor. After the presentation, a copy of the report should be given to the customer and three copies should be delivered to the information desk at IDI. A PDF version should also be sent by email to anreala@idi.ntnu.no.

All copies is to be accompanied with a CD that contain the implementation, documentation and a PDF version of the report. The content of this CD will be documented with a “Readme.txt” file, as specified in the compendium.

2.3.4 Gantt diagram

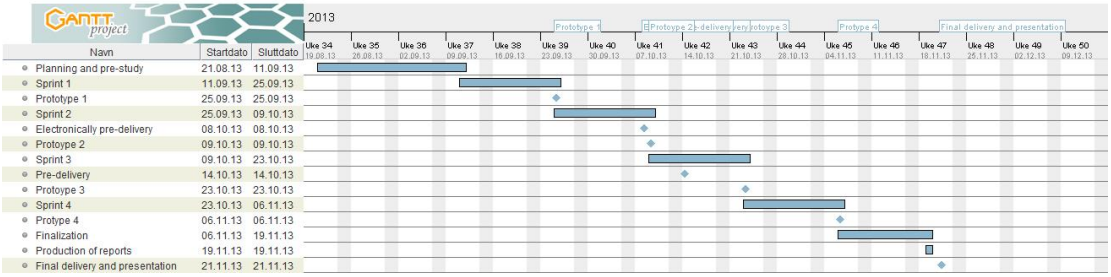

Figure 2.2: Overview of project

Activity Planning	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Finalization	
Lectures	60	4	4	4	4	4
Admin	25	20	20	20	20	50
Planning	50	8	8	8	8	50
Pre-Study	100	20	20	10	10	10
Requirements	60	15	15	10	10	0
Design	5	30	30	15	15	0
Implementation	0	60	60	50	50	10
Testing	0	22	22	50	50	38
Documentation	0	21	21	33	33	38
Sum	300	200	200	200	200	200

Table 2.1: Time sheet per phase

2.4 Risks

Nr	R1
Activity	All
Risk factor	Group member having a full time job in addition to being 75 % student
Consequences	H: The quality and scope of the project might suffer
Probability	M
Strategy and actions	Monitor situation closely. Have clear assignments and deals
Deadline	Continuously
Responsible	Project leader

Table 2.2: R1

Nr	R2
Activity	All
Risk factor	Disease
Consequences	H: The available work power might be reduced and tasks delayed.
Probability	M
Strategy and actions	All tasks to be entered into task management system. When sick, the group should be informed and if necessary tasks assigned.
Deadline	Continuously
Responsible	Everyone

Table 2.3: R2

Nr	R3
Activity	All
Risk factor	Lack of responsibility
Consequences	H: Other members must do the work originally assigned to some other member
Probability	M
Strategy and actions	Make understandable tasks with clear deadlines. Other members might take some of the workload, stalling other parts of the project. If need be, the course staff should be notified.
Deadline	Continuously
Responsible	Everyone

Table 2.4: R3

Nr	R4
Activity	Cooperation - All
Risk factor	Internal conflicts
Consequences	M: As the social environment will suffer, it will drain more energy from the people involved. This will in turn most likely reduce quality of the work done.
Probability	M
Strategy and actions	Resolve disagreements before they turn into conflicts and respect that others might have a different opinion.
Deadline	Continuously
Responsible	Everyone

Table 2.5: R4

Nr	R5
Activity	Communication with customer
Risk factor	Miscommunication
Consequences	H: The solution delivered might not be the solution asked for
Probability	M
Strategy and actions	Keep regularly and often in touch with customer. Write minutes, and get them approved by customer. Work in increments, and demo the current state of the prototype often.
Deadline	Continuously
Responsible	Customer contact and secretary

Table 2.6: R5

Nr	R6
Activity	All
Risk factor	Distributed work
Consequences	H: Efficiency might lower as could quality
Probability	M
Strategy and actions	Daily updates on the current work. Check each others work. Tasks with clear deadlines
Deadline	Continuously
Responsible	Everyone

Table 2.7: R6

2.5 Project organization

The following section describes how the team was structured in the project. We go through what roles team members are allocated to, what the roles entails and finally the time schedule for the team.

Role	Member
Customer contact	Odd Fredrik Mørch Rogstad
Scrum master	Odd Fredrik Mørch Rogstad
Secretary	Christian Frøystad
QA manager	Simon Stastny
Test manager	Simon Stastny
Project manager	Knut Nergård

Table 2.8: Group organization

Following, we list the different roles in the team, and what the tasks that comes with the positions is. This is to be seen as guidelines, the individuals are not necessarily expected to do all the work related to the position by themselves. They are however responsible for making sure it is being done.

2.5.1 Customer contact

The customer contact is responsible for the activities involving the customer. He is to arrange meetings with customer, keep the customer informed and generally serve as the contact person for the team in regards to the customer.

2.5.2 Scrum master

The scrum master is responsible for making sure the sprints progress as planned. If needed he will make alterations to the plans to better fit the current status.

2.5.3 Secretary

The main responsibility for the secretary is to get what is being discussed down in writing. He takes notes during the meetings the team attends, and writes minutes of the meetings which is then distributed to the team .

2.5.4 QA manager

The project has several rules to ensure good quality both in documentation, interaction with customer and advisor, and in work done by the team. The QA manager is responsible for making sure the team lives up to these rules.

2.5.5 Test leader

The test leader is responsible for the activities around testing. This includes the writing of tests, ensuring that they cover the requirements the customer has set, the execution of tests, and handling the results from the tests, figuring out what has to be done as a result of the test.

2.5.6 Project manager

The project manager is responsible for keeping the project on track. He should know the status of the project at any time, and know what has to be done to make progress. He ensures that everyone in the team has something to do and keeps track of the hours spent from the individuals in the team. Finally he is responsible for keeping a positive work environment in the team, working to solve conflicts if they may arise.

2.6 Quality Assurance

Quality assurance is an important part of any project. Without any sort of standards and routines, the collaboration of the group will suffer major setbacks once you need to combine the individual works in the group to the final product, as well as general confusion and/or uncertainty. To ensure our product turns out the best possible way and that the organization was pleasing, we started early setting up rules for communication within the group as well as external communication with customer and supervisor. Routines of the method of work was set up, and standardized templates were created.

2.6.1 Routines

Having a group with widely different time schedules, we quickly figured out that a more decentralized work structure would be in order. By doing this, we further increased the importance of good routines for work sharing and communication, as the group meetings were scarce and errors or misunderstandings might exist for a prolonged time if they first surfaced. To create a good channel for communication and document sharing within the group and externally, several steps were taken. Our main communication tool internally was a facebook project group. Here, internal messages, meeting reminders, questions and answers with more was posted. We also agreed using a mailing list in addition for more short-time updates, like meetings being canceled. This was seen as a necessary step after an unfortunate experience were not everyone noticed the cancellation message in time.

To handle sharing of individual works, a shared google disk was used. The access was only given to the group members, as this was not the platform we used to share files with the customer and advisor. The google disk was structured into several subfolders. Any document that is to be shared with externals, that being customer or advisor, is first stored in the respective customer and advisor folders, edited if needed, and approved by the group before they are sent to the externals in the selected way of communication. Files to be shared within the group had it's own folders as well. Lastly, each member has a private folder to note down hours spent or to store files not ready to be fully shared yet.

To share documents with externals, the group has taken use of dropbox and mail. Documents that has been approved by the group is sent either by mail or added to the shared dropbox folder where the externals can access them.

Even though our system of working in separate locations works to a certain degree, we still needed some meetings to make sure the group was somewhat synchronized and kept on the right track both according to the customer as well as internal work. To manage this we set up meetings for the group. Once a week, we arranged a customer meeting, an advisor meeting and a group meeting.

2.6.2 Customer meeting

We held weekly meetings with the customer. The meeting day and time was scheduled on a weekly basis, and more or less bound to Tuesdays or Wednesdays. The goal with these meetings was to establish a common understanding on how the system should be, and find the right requirements to agree upon. To make sure we are in an understanding with the customer, we take notes and create minutes from the meeting. This is in turn sent back to the customer, who either approves them or brings back feedback on what was misunderstood for the next weeks meeting. If anything happened to be misunderstood, this will be detected and changed within a week, preventing major workloads on the wrong premise. In addition to the minutes, we used action points, which are activities the customer wants us to look into or complete before the next meeting. This could be creating time estimates of user tests, architecture overviews, UI suggestions and more.

2.6.3 Advisor meeting

Once a week, we met with the advisor to discuss how the project was coming along. If the team faced any problems with the project or the team which we could not manage

solving by ourselves, this was the place to get assistance. This could be anything from lack of required knowledge to personal conflicts within the team. The meetings were also held to give the advisor an overview of our situation. Weekly status reports and timesheets were given to help the advisor keeping an overview of the progress of the project.

2.6.4 Group meeting

At least once a week, we held group meetings. The goal with these meetings were to help synchronizing the group, updating all the members on what was done and what was to be done, raise important questions about further progress, or simply having a place to raise questions or concerns that does not fit other places. Although we simultaneously used a shared document were the team members wrote their progress and intended actions, the meetings ensured everyone was being a part of the update. Important decisions also benefited from having all team members come together on a solution, so no one feels left out of the process as well as smart solutions that otherwise might be overlooked is included. In that regard, group meetings were held when as many as possible could attend.

2.7 Templates and procedures

Here the established standards are outlined.

2.7.1 Document templates

The group has established standard documents for agendas for meetings with respectively the customer, the advisor and the group. A template has also been made for writing minutes from each meeting.

A template for the weekly status report has also been made, and can be used with few modifications every week.

For meeting invitations a standard has also been established so that inviting people to meetings should be done in an uniform way and as effectively as possible.

2.7.2 Code standards

The project uses Java for server side programming, and JavaScript for client side programming.

Java

The project follows the coding guidelines of Play! Framework[2], which in turn relies heavely on Java Code Conventions[1].

Whitespace

- Use spaces, not tabs
- Use 4 space characters to indent

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

- A line should not exceed 100 characters
- Lines should not contain trailing spaces.

Symbols

- Always use braces around blocks, even single line blocks.
- The opening brace is always on the same line as the if statement, except when you have multiple arguments.

```
if (arg1 ||
    arg2 < arg1 ||
    arg3)
    {
        doSomething();
    }
```

- When breaking logic, || and && at the end of the line.
- One instruction per line.
- Always put else on its own line.

Function and variable names

- Use US English for function and variable names.
- Use full English words for variable names. Examples: documentProvider but not docProv, feedElement but not fdElt. Variable names reduced to a single letter should not be used, unless in indexes such as in for loops.
- Do not prefix interface types with I, and do not prefix abstract types with Abstract

Comments

- Comments should be in grammatically-correct US English, with full sentences that include punctuation.
- Javadoc comments before classes or methods are not mandatory.
- Don't include empty javadoc. Either the javadoc provides information and it's good to have it, or it doesn't and shouldn't be there.
- Do not leave commented-out code in source files. If the code is no longer used, just delete it. We can always use the Git history to get it back if necessary. Similarly, delete any dead code.

General Java Guidelines

- Do not use raw types: List¡String¿ rather than simply List.
- Do not leave useless import statements in the code.
- Use the Override annotation whenever possible (but not for interface implementations as it breaks Java 5 compatibility).

- Do not ignore exceptions (no try with an empty catch block). If you don't handle a checked exception you can either add a throw to your method declaration (if you believe the exception is worth staying checked) or throw it again as a RuntimeException if it's rare enough so users of your method shouldn't have to care.

```
try {
   // Code with a checked exception that shouldn't be checked
} catch(Exception e) {
   throw new RuntimeException(e);
    // This will propagate to the controller,
    // and the developer will see the exception in his web browser
    // (or it will be logged if in production mode)
}
```

- Prefer enum to int constants.

JavaScript

The coding standard for JavaScript is taken directly from Titaniums coding standard[3].

Avoid the global scope
Putting objects into the global scope can cause various problems:

- Objects placed in the global scope will not be automatically garbage collected. You'll have to manually null global objects to mark them ready for collection.
- It's easy to inadvertently overwrite an object in the global scope, because that variable is accessible so widely within your program.
- The global scope of app.js is not accessible from other contexts or within CommonJS modules. So, you can't just dump variables there so you can access them throughout your app.

For these reasons, avoid defining variables in the global scope. Objects are placed in the global scope when:

- You declare a variable outside of a function or CommonJS module. Using a modular pattern will alleviate this problem.
- You omit the var keyword when declaring a variable (within or outside of a function). So always use var when declaring variables.

Avoid local objects in global event listeners
The following code will cause a memory leak because the locally scoped variables are referenced in a global event listener. This is because the program will need to retain the locally scoped vars in order for the global event listener to use them. The global event listener will also persist until the app exits or the listener is explicitly removed.

```
var someFunction = function() {
    var table = Ti.UI.createTableView(),
        label = Ti.UI.createLabel(),
        view = Ti.UI.createView();
```

```
    Ti.App.addEventListener('bad:move', function(e) {
        table.setData(e.data);
    });

    view.add(table);
    view.add(label);

    return view;
};
```

Global event listeners include those associated with Ti.App, Ti.Geolocation, Ti.Gesture, and so forth. The same problem is possible with non-global event listeners, like those you associate with a UI element. If that UI element remains valid in memory, any event listeners – and the objects they refer to – must also be kept in memory.

The above example is an anti-pattern that will eventually consume the app's available memory. It's important to note that this is a common anti-pattern that developers employ in browser-based environments too, where it causes the same result, so it is not unique to Titanium.

If you need to have a custom event, consider a method / callback that you can invoke later on. For the global events like location, network change, etc. it's highly recommended to place them in app.js. The general rule of thumb is global events handle global objects.

Defer script loading One of the bottlenecks of a Titanium application is JavaScript evaluation. This is particularly the case for Android, although the V8 runtime provides substantial improvements for this issue compared with Rhino. For that reason, to speed the startup and responsiveness of your application, you should avoid loading scripts until they are absolutely needed. As in the following application, which has three windows to be opened in succession on a click (touch) event, note that the dependent JavaScript for each window is not loaded until absolutely necessary.

Lazy script loading in app.js

```
//muse be loaded at launch
var WindowOne = require('ui/WindowOne').WindowOne;

var win1 = new WindowOne();
win1.open();

win1.addEventListener('click', function() {
  //load window two JavaScript when needed...
  var WindowTwo = require('ui/WindowTwo').WindowTwo;
  var win2 = new WindowTwo();
  win2.open();
  win2.addEventListener('click', function() {
    //load window three JavaScript when needed...
    var WindowThree = require('ui/WindowThree').WindowThree;
    var win3 = new WindowTwo();
    win3.open();
  });
```

```
});
```

Or, if you're not using CommonJS but building out a namespace: *Deferred loading to build a namespace*

```
var someNameSpace = function() {
        var API = {
                init: function() {
                        // create your UI here or do whatever
                }
                reset: function() {
                        // null objects, clean up, etc
                }
        };

        // Construct anything you want outside the local 'API' object

        return API;
};
// And to use it
var test = new someNameSpace();
```

Don't Extend Titanium Prototypes
Many users attempt to add to the Ti namespace as a means to persist data across contexts, extend / override native methods, etc. This can sometimes work but is very unreliable for the following reasons:

1. The Titanium end objects are really not true JavaScript objects. They are proxy representations of native operating system components. As such, they are constructed to pass through properties and method invocations. Your extensions could conflict with native functionality or interfere with proper operation of the proxy objects.

2. Sometimes you might be able to store things on the namespace but it's not changeable (i.e. an array stored on the namespace might not be able to be modified - mutable, etc.). Other-times your stored objects will be completely null.

3. Since this isn't an approved way of storing anything, there's no guarantee it will work in future releases of Titanium.

As a rule do not add to, or extend via the prototype, any object or module in the Titanium namespace. If you want to extend a core part of the Titanium API you should build a native module to accomplish this. If you're just looking for an extendible JS namespace, create your own (i.e. {{var MyApp={} }}). Coding strategies for multiplatform apps

Branching in code is useful when your code will be mostly the same across platforms, but vary here and there. Long blocks of if...then code are difficult to read and maintain. Also, excessive branching will slow your app's execution. If you must use this technique, try to group as much code as you can within a branch and defer loading as much as possible to mitigate the performance penalty of branching.

Using platform-specific JS files is likely to be most useful when your code is mostly different across platforms. This removes long if...then blocks from your main code. Separating platform-specific code reduces the chances of an error that comes from accidentally using the wrong platform's API or property. However, you'll have to remember to apply changes and fixes to each of the platform-specific files. So this approach could increase your work rather than reduce it.

Don't store sensitive data in non-JavaScript files
Your JavaScript files are minified and obfuscated when you build for distribution. Depending on your target platform, they will be further processed and packaged into the compiled or "object-ified" files of your app. However, images, JSON files, SQLite databases, and other files not named with a .js extension are simply packaged as-is with your app's files.

APK and IPA files are essentially Zip files. Their contents can be revealed by any Zip-decompressor. Thus, your non-JavaScript files are accessible to the curious.

You should not include sensitive data in non-JS files. Simply renaming files with a .js extension is not a suitable alternative. Such files might not be supported on device. And, the Titanium build process removes them from the final build. Set local variables to avoid calling native methods

Each time you request the value of a device-related property, Titanium has to query the operating system for the value. For example, if you read from Ti.Platform.osname or Ti.Platform.displayCaps.platformHeight, Titanium must take a "trip across the bridge" to request the value from the operating system. Doing so takes a few cycles and if used too frequently could possibly slow your program. Something like the following would be more efficient:

```
var isAndroid = (Ti.Platform.osname=='android') ? true : false;

if(isAndroid) {
        // do Android specific stuff
} else {
        // do iOS stuff
}
```

Modular components with CommonJS
Appcelerator's primary recommended architecture a modular app architecture constructed with CommonJS modules. In fact, we have a whole Best Practices section devoted to CommonJS Modules in Titanium. CommonJS modules are discrete and independent building blocks, eliminating concerns about global variables and naming conflicts. In our testing, it is a highly performant architecture compared to some other solutions. This pattern is also used by other JavaScript-based environments, such as Node.js.

MyModule.js

```
// variables defined in this file are private
var defaultMessage = "Hello world";

// we make objects, variables, functions available to the
```

```
// calling context by adding them to the exports object
exports.sayHello = function(msg) {
        Ti.API.info('Hello ' + msg);
};

// we can assign other objects, functions, and variables to
// exports and they will be available to the calling context
exports.helloWorld = function() {
        Ti.API.info(defaultMessage);
}
```

app.js

```
var myModule = require('/MyModule');
myModule.sayHello('Kevin');  //console output is "Hello Kevin!"
```

Other architectures are valid and meet the needs of many developers. Which you choose is ultimately up to you and your experiences

Custom objects as components
Another popular pattern is one we teach in our training classes, that of custom objects typically stored within an app-specific namespace hierarchy. This model is flexible and well-suited to rapid deployment projects. It takes advantage of JavaScript's language features. Components are all members of the same global scope, thus sharing data within the app is simple. And when implemented well, this pattern can lead to very readable (and thus maintainable) code.

On the downside, this pattern is less performant than CommonJS modules, especially on Rhino/Android. The rapid nature of this pattern can lead the developer to general, high-level bad practices and developer 'laziness'. Inheritance is vague or even non-existent. And critically, memory management can be difficult as object references can remain after they're no longer needed.

```
// create an object literal to be your app's namespace
var myapp = {};

// the following could be in a separate "ui.js" file and include()'d into you
(function(){
        myapp.ui = {}; // this sub-namespace extends the app's namespace obje

        myapp.ui.createApplicationWindow = function() {
                var win = Ti.UI.createWindow({
                        backgroundColor:'white'
                });

                var header = Ti.UI.createLabel({
                        text: 'My App Heading',
                        top: 10,
                        height:'auto',
                        width:'auto'
```

```
                });
                win.add(header);
                return win;
        };
})();
```

The same could be accomplished without the self-calling function, if you prefer:

```
// create an object literal to be your app's namespace
var myapp = {};

// the following could be in a separate "ui.js" file and include()'d into you
myapp.ui = function() {
        var API = {
                createApplicationWindow: function() {
                        var win = Ti.UI.createWindow({
                                backgroundColor:'white'
                        });

                        var header = Ti.UI.createLabel({
                                text: 'My App Heading',
                                top: 10,
                                height:'auto',
                                width:'auto'
                        });
                        win.add(header);
                        return win;
                }
        }
        return API;
};
```

Classical-based patterns

In general, Appcelerator does not recommend classical-inheritance based models because JavaScript is not a class-based language. For an in-depth look at inheritance patterns in JavaScript, we recommend you read Douglas Crockford's Protypal Inheritance in JavaScript and Classical Inheritance in JavaScript articles.

Classical inheritance is familiar for programmers coming from Java and other class-based languages. They rightly claim that this pattern enforces discipline and logically structured code that is generally easy to read, debug, and document. However, it's our belief that such a pattern confuses the idea of 'classes' and 'objects' in Javascript, forces the programmer to define his or her own inheritance rules, and is slower to implement in a rapid-prototype setting.

The code below is a fragment of this pattern:

```
var SomeUIClass = function() {
        // ----- DEFINE PRIVATE PROPERTIES AND METHODS -----
        var UIGroup = Ti.UI.createView({ zIndex:5 }),
            UIBg = Ti.UI.createView({ borderRadius:10, opacity:0.2, width:150
```

```
            UIInd = Ti.UI.createActivityIndicator({ height:50, width:50, botto

        // ----- DEFINE PUBLIC PROPERTIES AND METHODS -----

        // ----- Public Properties -----
        this.somePublicProp = null;

        // ----- Public Methods -----
        this.display = function() {};
        this.toggle = function(toggle) {};
};
```

2.7.3 Version control procedures

All documents are produced in Google Docs, which incorporates version control for each document. The documents will regularly be backed up on local storage.

All code will be tracked with git and checked in continuously in order to always be consistent.

Chapter 3

Preliminary study

3.1 Current situation

Cultural Heritage is the legacy of physical artifacts and non physical attributes of a group or society that are inherited from past generations. According to [4], cultural Heritage can be divided into three main categories; tangible culture, intangible culture and natural heritage. The system will mainly focus on the "culture in the landscape", i.e. the discovery of cultural memories that we meet when walking around in cities, in villages and in the nature.

An important aspect in cultural heritage is preservation, the task to preserve and protect buildings, objects, landscapes and other important cultural artifacts. One thing is to physically protect these artifacts, but another important aspect is to educate today's generation and help them discover and learn about our cultural heritage to both enrich and preserve our legacy for future generations.

This section will give the reader a better understanding of the current situation in how cultural heritage is preserved today. It will also take a look at existing mobile applications that helps people discover and learn about cultural heritage.

3.1.1 Cultural Heritage today

The most famous organization working with cultural heritage is UNESCO [5]. The United Nations Education, Scientific and Cultural Organization is an agency of the UN (United Nations), working for peace and security through international collaboration in education, science and culture. UNESCO World Heritage Committee is responsible for establishing the famous list of UNESCO World Heritage Sights [6], which today contains 981 properties (September 2013), that includes cultural, natural and mixed properties.

There are seven listings in Norway that the UNESCO World Heritage Committee consider having outstanding universal value, that is;

- Bryggen, Bergen
- Urnes Stave Church, Luster in Sogn and Fjordane
- Røros Mining Town and the Circumference, Røros in Sør-Trøndelag
- Rock Art of Alta, Alta in Finnmark

- Vegaøyan – The Vega Archipelago, Vega in Helgeland
- Struve Geodetic Arc, multiple measuring points in several countries
- West Norwegian Fjords - Geiranger and Nærøyfjord

These are really marvelous pieces of cultural heritage, but is that it? Norway has only seven sights that are considered as cultural heritage and worth a visit? Of course not, our scope is way bigger than that! It is up to the users to decide what cultural heritage is.

3.1.2 Existing systems

In 2004 The Directorate for Cultural Heritage in Norway (Riksantikvaren) launched a database, called Askeladden, over protected cultural heritage sights and cultural environments in Norway. The problem was that it was only available to agencies involved in cultural heritage management, and not to the public. However, in 2009, parts of the database was finally made available through the website Kulturminnesøk (kulturminnesok.no), and today, after a bigger upgrade in 2012, it contains over 150 000 sights.

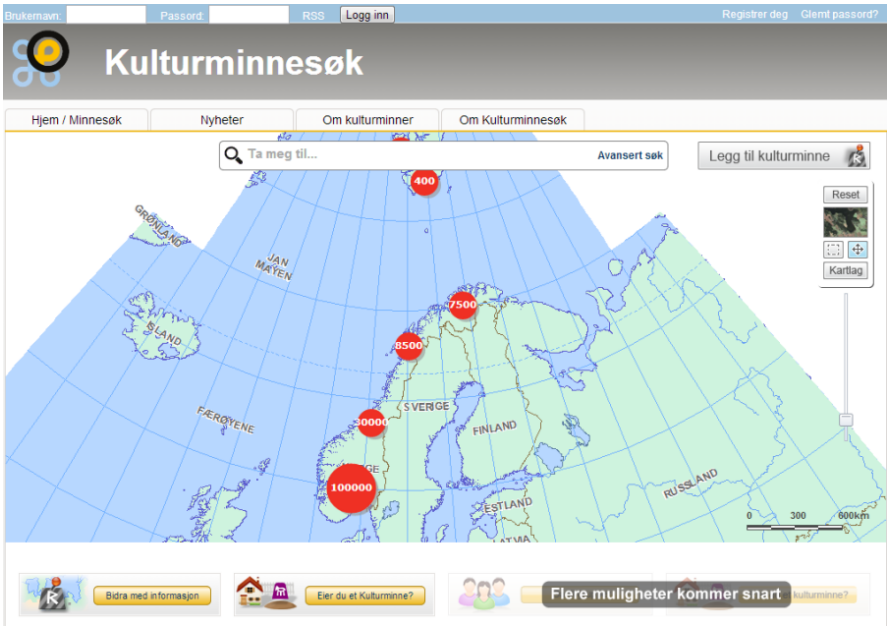

Figure 3.1: Kulturminnesøk, Norway

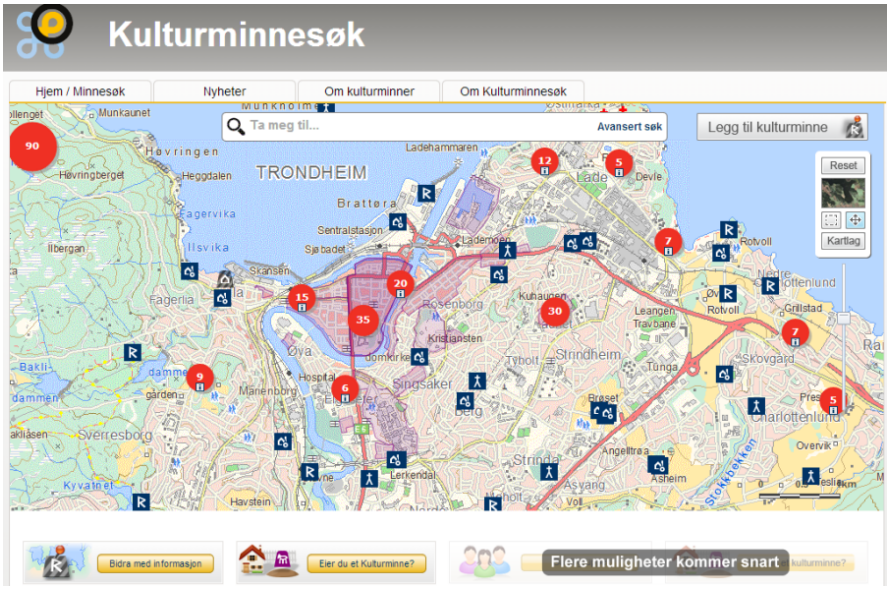

Figure 3.2: Kulturminnesøk, Trondheim

As you can see in Figure 3.1, sights from all over Norway is represented. We can either press the red circles, zoom in manually or search by text to find places we want to take a closer look at. If we zoom in over Trondheim, as Figure 3.2 shows, there are plenty of sights all over the city. Every sight is marked with a symbol, which tells the user if it is a building location, religious place, archaeological memory or technical/industrial memory. If you click on a symbol, a small info display appears, Figure 3.3, that informs the user what type of cultural heritage it is, its conservation status, dating and municipality. If the user wants more info, the user can press the "Mer info" (More info)-text, and a new sight appears, Figure 3.4, where a picture, if available, and a some more info appears. It is also possible to comment and like (as in Facebook) the site.

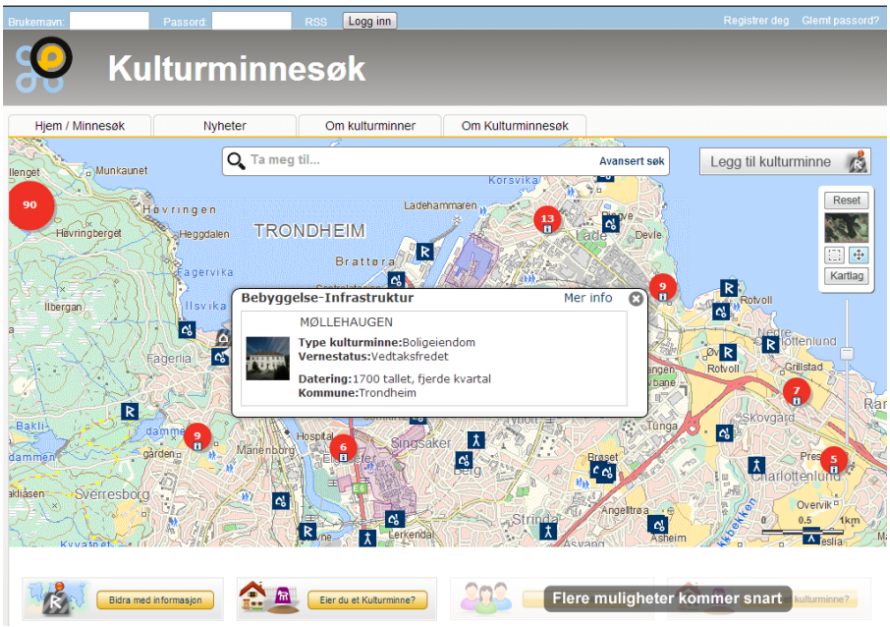

Figure 3.3: Kulturminnesøk, info display

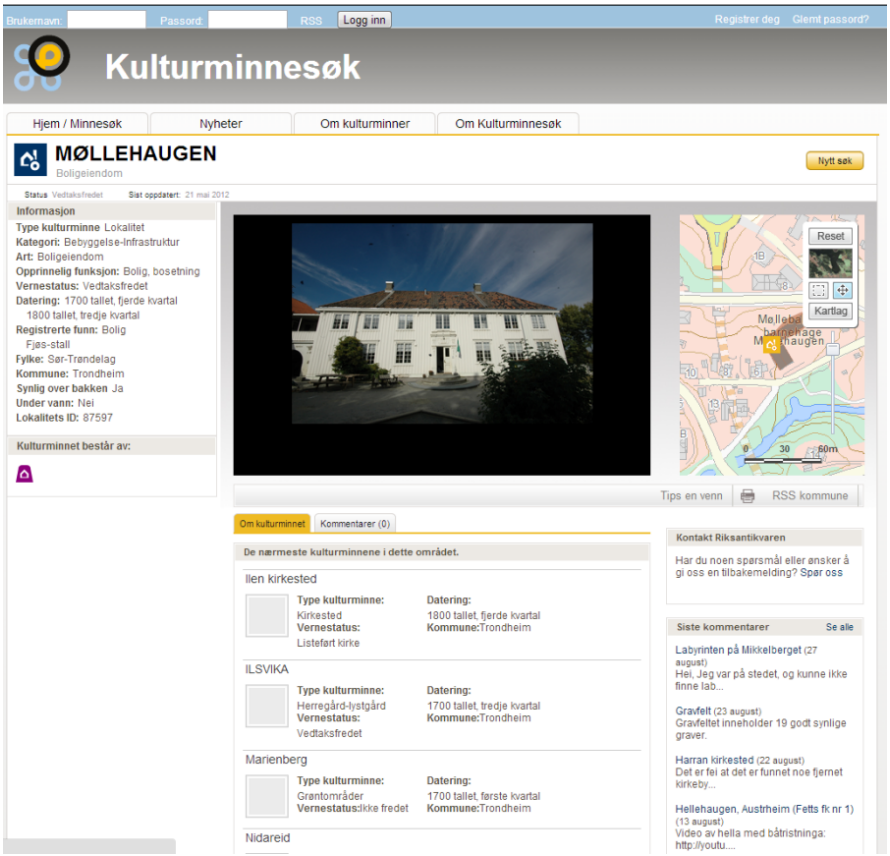

Figure 3.4: Kulturminnesøk, more info

Kulturminnesøk can be considered a partly edge dominant system (will be explained next section), since everyone can register and add their own cultural heritage sight. There is about 1500 sights that users have added. Users have added everything from swimming halls to archaeological memories. We call it just partly edge dominant, since only 1% of the content is added by the public.

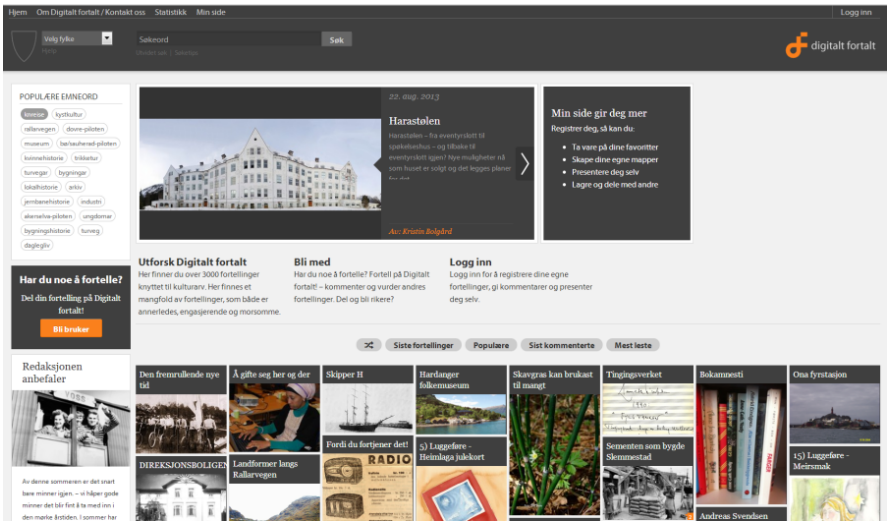

Figure 3.5: Digitalt fortalt, front page

Another edge dominant cultural heritage system is “digitalt fortalt”, or digitally told. Digitalt fortalt is a website where users, cultural institutions or private persons, can register and add stories related to cultural heritage. Users have added stories about everything from clothespins to protected buildings, and today there are over 3000 stories.

To read a story you can either search, by pressing one of the popular tags or simply search from the search field, or browse on the front page, in last told stories, popular stories, last commented stories or among the most read stories. If we search on “Trondheim” in the search field, we get 135 hits. The stories is placed in a grid system, and every story is represented in a little box, with a title, a picture (a play-sign if a sound is attached to the story), author and location. Some stories are also linked to a date or a period. The search result is shown in Figure 3.6.

If we press one of the stories, Regalierommet for instance, shown in Figure 3.7, a page with a media slider and information appears. In this case the story is represented as sound and pictures. The information below the media slider says something about why it was made, and some information about the creators. In other stories there are typically one or more pictures and the story represented with text.

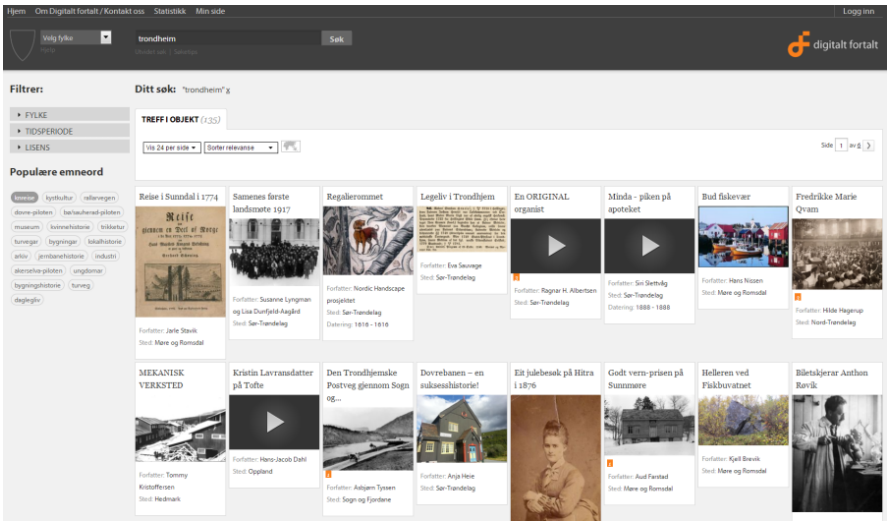

Figure 3.6: Digitalt fortalt, search result for Trondheim

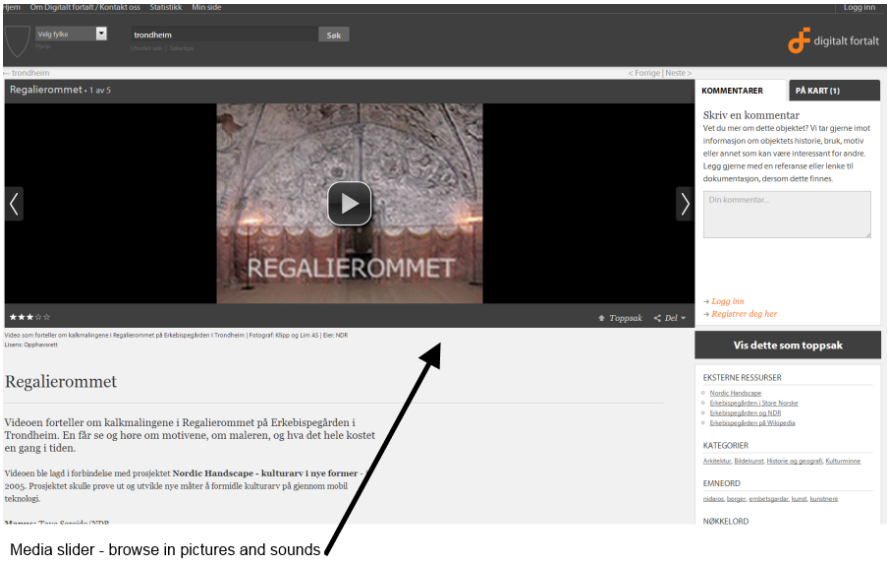

Figure 3.7: Digitalt fortalt, Regalierommet, with media slider and text

3.1.3 Edge dominant systems

An edge dominant system is one that almost entirely depends on input from users. Well known edge dominant systems today are Facebook, Twitter, YouTube, Wikipedia, etc., all of which have created enormous value by their users.

Almost all edge dominant systems today share a common ecosystem structure, called a "Metropolis" structure, by analogy with a city, as shown in Figure 3.8.

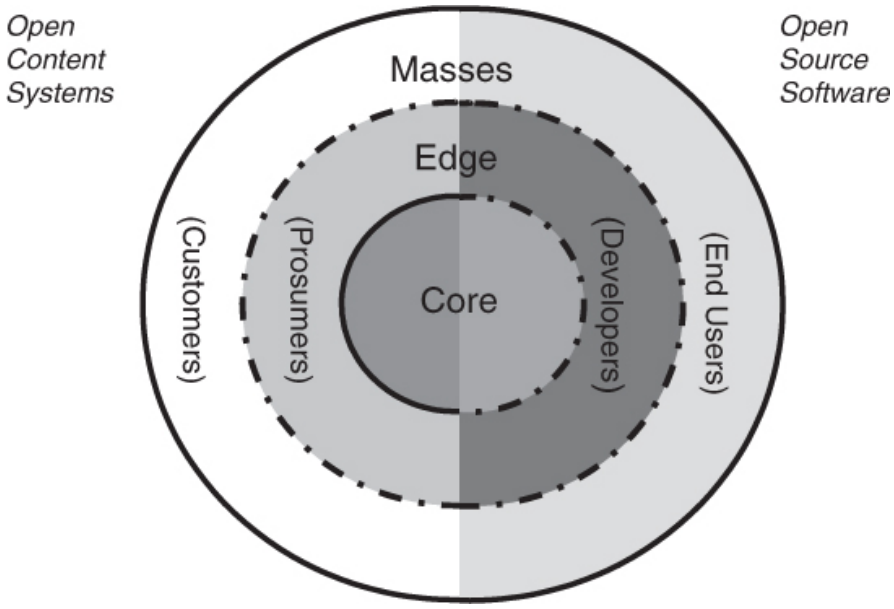

Figure 3.8: The metropolis

The metropolis structure divides the communities of stakeholders in an edge dominant system:

- In the outermost ring we have the masses, consisting of customers and end users. They consume the content made by the metropolis, and contribute requirements.

- The middle ring contains prosumers and developers. These are the people that produces the content for the metropolis. The developers write software for the community to use, and the prosumer produce (and consume) content in the metropolis.

- In the middle we have the core that keeps the metropolis together. This is the software that provides its services through a set of APIs that the prosumers and developers can use to produce new software and content for the masses to use.

3.2 Existing mobile applications

In the previous sections we have learned that there already are already systems that helps the public in finding cultural heritage. In this section we will look at some existing mobile applications which in many ways works quite similar to the systems above. It is important to look at these systems to get an idea of how others have done it before us, both to get inspired and to find weaknesses with this kind of applications. First we will take a look at Kulturminnesøk's mobile application, then at two danish solutions.

3.2.1 Kulturminnesøk app

After using Kultruminnesøk in the web browser we discovered that they also had a mobile application. Not surprisingly, it serves much of the same functions as the web application, but has some drawbacks and advantages. First of all, it has the same map function, Figure 3.9, you can zoom in and out with the well known "pinch-to-zoom" gesture, which allows the user to zoom in or out by moving two fingers further apart or closer together while touching the display. The same symbols as in the web application is used, and we can press them to get more information. In addition to the map you can use a list function where you can see your nearest sights, Figure 3.10. However, the coolest function is the mobile application is in the function "Vis meg", or "Show me"; this is a augmented reality function, where all sights within a certain range appears on the screen, through a camera application, represented by the same symbols in the map, Figure 3.11. A quite big drawback is that there are no pictures in the "More info" display, but the biggest weakness is that the application does not show user-created sights, just the cultural sights from the Askeladden database.

Figure 3.9: Kulturminnesøk, map function

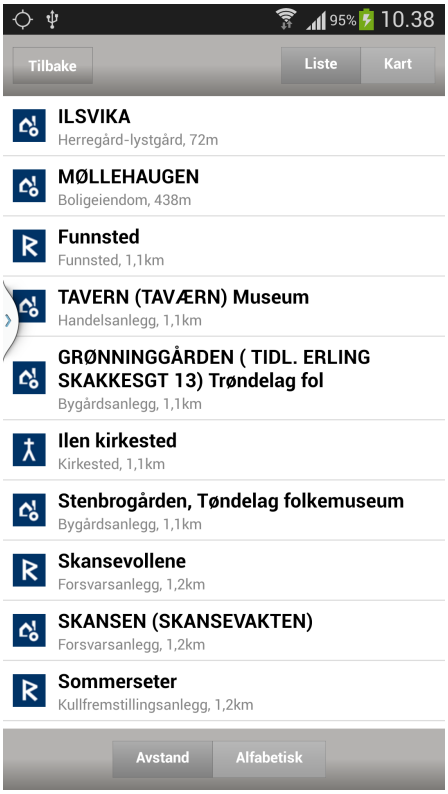

Figure 3.10: Kulturminnesøk, list function

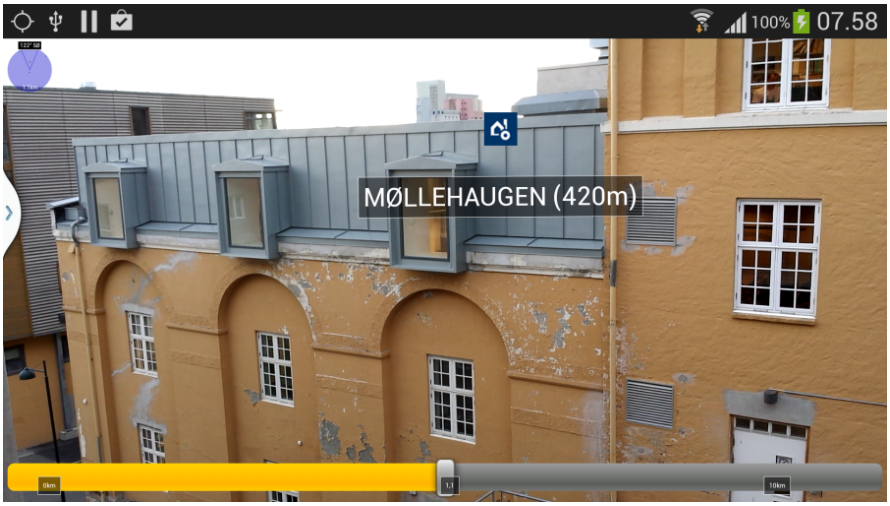

Figure 3.11: Kulturminnesøk, augmented reality function

3.2.2 Kulturarv app

Kulturarv, or Cultural Heritage, is a danish cultural heritage mobile application. It is very similar to the Kultursøk mobile application, it has both a map function, Figure 3.12, and a augmented reality function, Figure 3.12, to show the user where to find protected and preservation-worthy buildings. It uses the danish "Fredede og Bevaringsværdige Bygninger", or "Protected and Preservation-worthy Buildings", (FBB) web service, which appears to be quite similar to the Askeladden database. It sounds like the danish Kulturarv and Kultursøk is pretty much the same thing, but Kulturarv has one more important functions, which makes is much better; Kulturarv blends the media from FBB and other geolocation data sources like Instagram, Flickr, Wikipedia and Twitter. Unfortunately, we are not able to test the application because it seems to only work if you are physically in Denmark, but we have the impression of how it works through Kulturarv's homepage [7]. It is also worth mentioning that the application is an open-source project released under the GNU GPL v3 license.

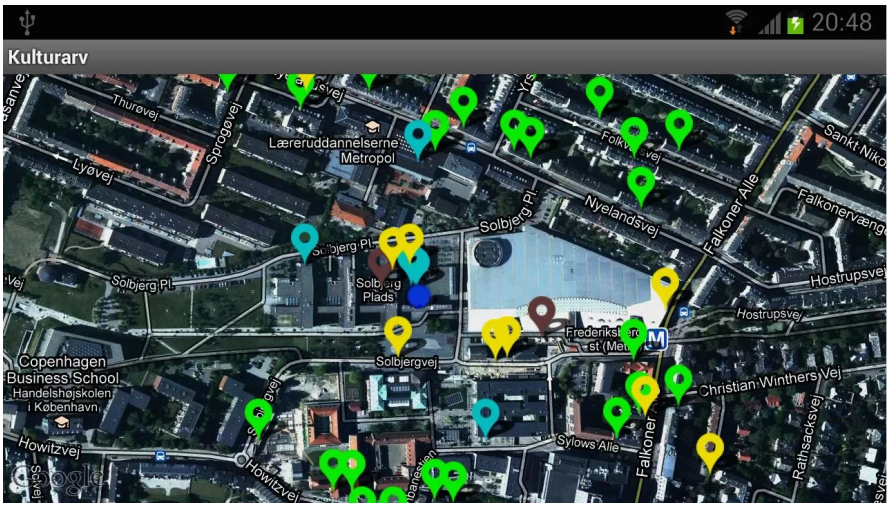

Figure 3.12: Kulturarv, map function

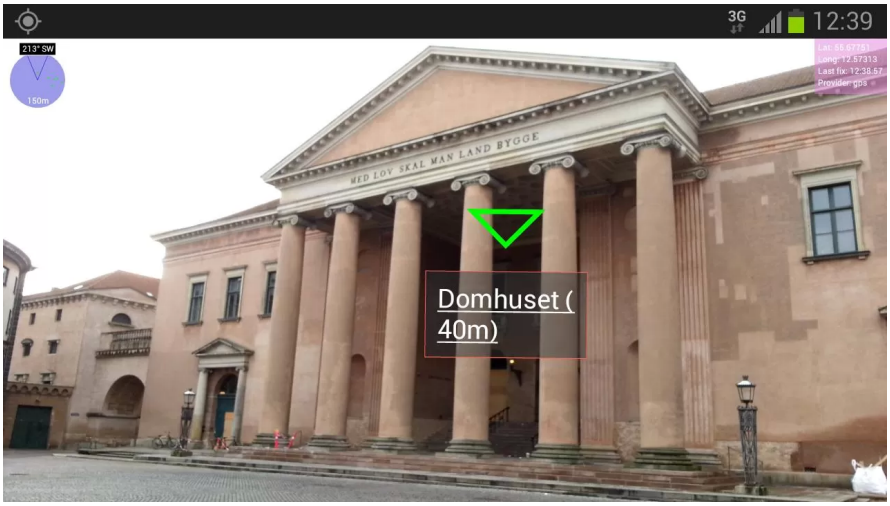

Figure 3.13: Kulturarv, augmented reality function

3.2.3 1001 Stories of Denmark

Again we are looking at a danish application that has many similarities with a Norwegian application, namely Digitalt Fortalt. It is called "1001 Stories of Denmark", and it lets the user read stories about 1001 places in Denmark, including ancient monuments, architectural buildings and sceneries. You can browse sights with either a map or a list, Figure 3.14 and 3.15. When you choose a sight, you can read about the sight in general, read stories about it or look at pictures, Figure 3.16. Another function is that you can get predefined routes or make your own routes, that includes the sights you want to see. For example, you can find a route that is about the famous architect and designer Arne Jacobsen (1902 - 1971), that includes visits to buildings that was designed by Arne Jacobsen. In the same way as the previous application, we can not test it because you need to physically be in Denmark.

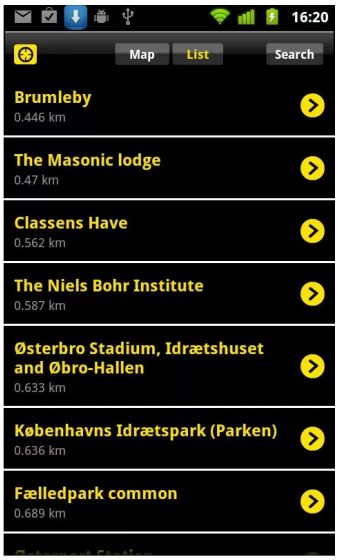

Figure 3.14: 1001 Stories of Denmark, list function

Figure 3.15: 1001 Stories of Denmark, map function

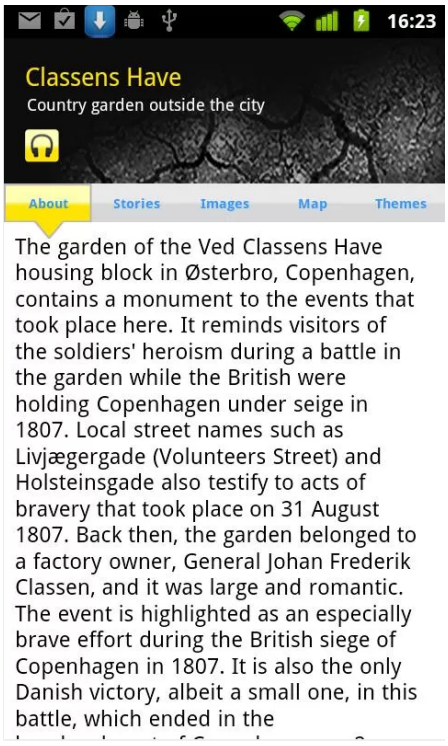

Figure 3.16: 1001 Stories of Denmark, a sight

This mobile application also has an associated website [8], where you have the same functions as in the mobile application. In addition there is a timeline function, where

all the stories with a timestamp appears, Figure 3.17. The website also invites users to create their own profile, that allows them to add and recommend places, add stories, comment on existing stories and to upload images and videos.

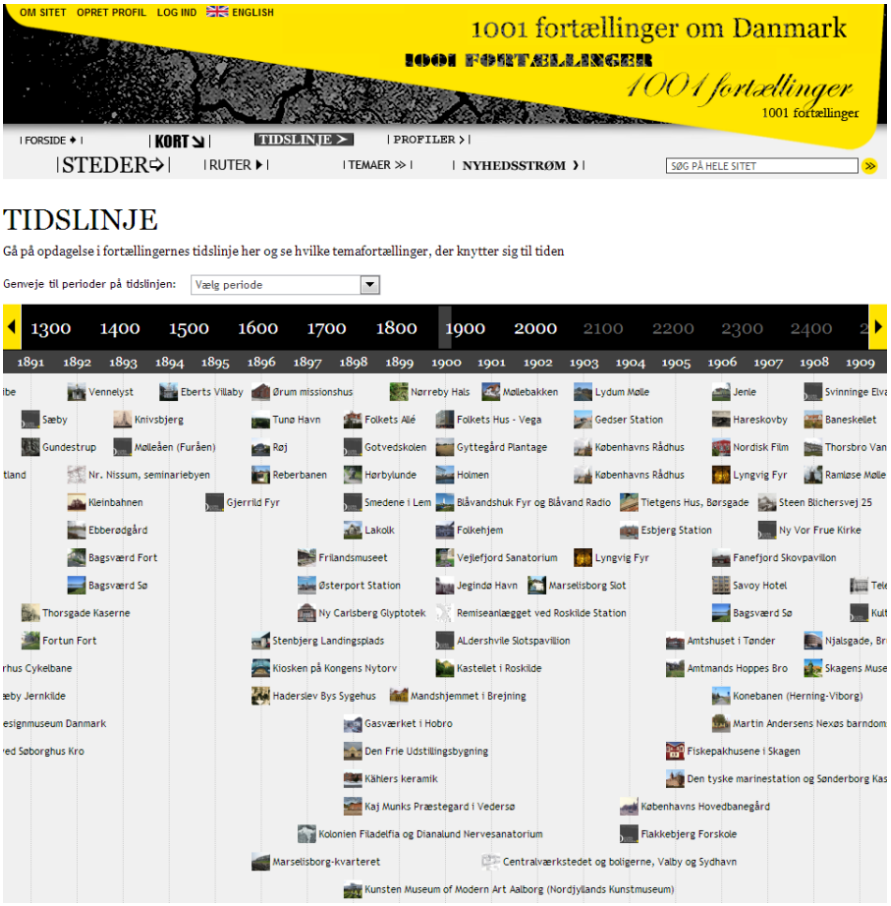

Figure 3.17: 1001 Stories of Denmark,timeline function

3.2.4 Conclusion

All the applications we have looked at are great pieces of technology. It educates today's generation and helps them discover and learn about our cultural heritage. They are all content-rich applications, and they offers the public to contribute. One thing we have noticed is that nearly every sight, story, place or piece of history are pretty thoroughly made. Perhaps we should focus on a more low-threshold application, so that everyone, even the very young, can participate. We see that every applications has their own advantages, and we hope that we can purify some of these and perhaps even come up with some new revolutionary features.

3.3 Work methodology

There are different ways of approaching a project. Two of the more commonly used methods is Waterfall and SCRUM.

3.3.1 Waterfall

The Waterfall model [9, p. 30-32] is derived from general system engineering processes.The work done in a Waterfall model is partitioned in activities. The activities are done sequentially, only when the prior is done, can the consecutive start.

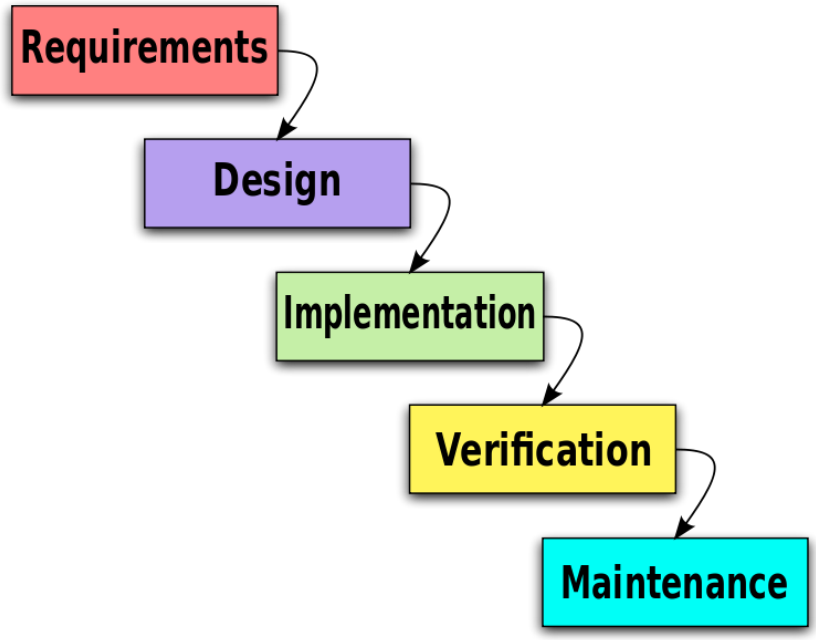

Figure 3.18: The Waterfall process [10]

Every phase produces one or several documents to be passed on to the next phase. These documents needs approval before a phase is considered completed, and the next might start.

Waterfall makes good sense in larger manufacturing environments where certification of every part of the system is required in order to be allowed to manufacture and sell the product. For example the aircraft industry needs detailed plans of the whole aircraft to be certified before manufacturing, and faults could produce devastating effects.

In the original publication about Waterfall for software systems, the author raises some limitations of the model and suggests some project owner involvement.

> For some reason what a software design is going to do is subject to wide interpretation even after previous agreement. It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery. To give the contractor free rein between requirement definition and operation is inviting trouble. [11, p. 335]

3.3.2 SCRUM

In accordance with the statement from Royce above, SCRUM strive to involve the customer throughout the whole development process. This is done by doing only the strictly required planning and specification writing up front, and then working iteratively with the project together with the customer in sprints. According to [9, p. 73] sprint

could last from 2 - 4 weeks and by the end the work is review and result presented for the customer. For each sprint the customer sees the current state of the project and participates in specifying what to do for the next sprint.

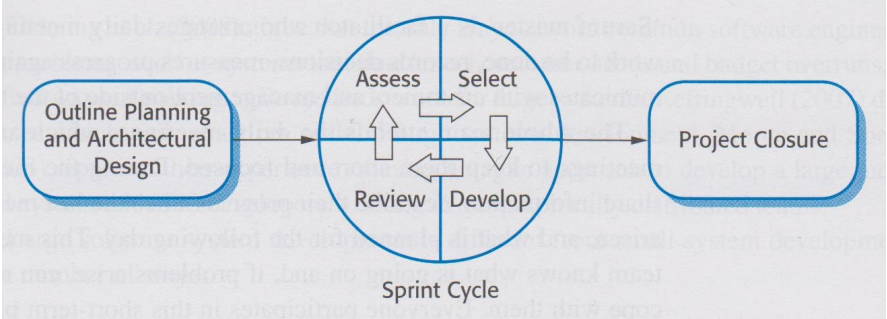

Figure 3.19: The SCRUM process

This use of continuous customer involvement, helps clarifying misunderstandings and correcting errors on both the customers and contractors side during the process. The fact that this might be caught earlier, means that the project is more likely to deliver on time and on budget.

The customer involvement is primarily within prioritizing and adding items to the backlog which is used as the basis for each sprint. The customer is also involved in clarifying items as they are moved from the backlog to the sprint log.

3.3.3 Chosen work methodology

Given the uncertainties surrounding the concrete goal and plan behind the project, it is beneficial to choose SCRUM to retain a close connection with the customer. This will help resolve any misunderstandings or disagreements about project scope and focus areas.

The customer have experience with SCRUM and suggested the methodology to be used. It will also make it easier to produce a good end project, knowing that we are allowed to refine our documents, requirements and such during the lifetime of the project.

Chapter 4

Requirements

This chapter is a System Requirements Specification document in the terms of practice recommended by IEEE 830 [12].

4.1 Introduction

4.1.1 Purpose

The purpose of this document (System Requirements Specification, hereinafter SRS) is to collect and analyze the requirements for VirtualWall software system. The requirements formalization as presented is a result of a joint effort of the customer and the development team. It serves as a description of what customer is expecting the system to be like, it presents the systems features and functionalities.

The intended audience of this document shall be:

1. development team (students)
2. customer (SINTEF ICT, eTrøndelag/Sør-Trøndelag fylkeskommune)
3. course responsibles (advisor, etc.)

Each part of the audience holds different stakes in the project, but all are interested in information contained within the SRS.

4.1.2 Scope

This SRS document deals with the project Virtual walls – walls that tell us stories. This project shall result in developing a tool for working with so called virtual walls, creating and sharing stories on them.

As presented by customer, the VirtualWall software system is going to be a complex system consisting of a number of subsystems including several content databases, a server backend and web application frontend.

In the scope of this year's run of Customer Driven Project course, a prototype of a web application frontend and a basic server backend shall be developed.

4.1.3 Definitions, acronyms and abbreviations

AJAX	Asynchronous JavaScript and XML
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
Virtual Wall	By virtual wall, we understand a virtual space where users can share stories like on a real-life wall. Stories can be textual, as well as featuring interactive media like video, audio, images or hypertext.
Owner	All items (walls, stories) in the system are created by someone. We call this user owner of the underlying entity

Table 4.1: Definitions, acronyms and abbreviations

4.1.4 Overview

The rest of this SRS document describes the proposed product in the means of its desired functionality, characteristics, constraints and perspectives. This general description could be found 4.2. Description of the actual requirements is to be found in 4.3.

4.2 Overall description

4.2.1 Product perspective

As stated above, the whole system will consist of several subsystems such as the frontend, server backend, own database and external content databases. This implies the use of client-server model.

The components developed in this phase of the project are:

1. mobile/web application (frontend)

2. server (backend)

Customer requires that the tool shall be available for PCs, tablets and mobile phones (smartphones). The prototype shall be optimized for a tablet preferably.

The following diagram shows the whole system and how the separate parts work together.

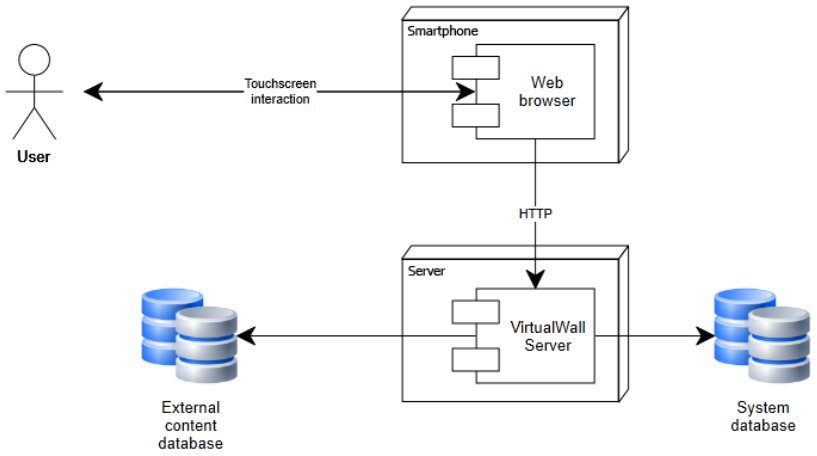

Figure 4.1: Overview of interaction

In the initial phase, the server will not be directly connectex to any external content databases machine, the content from them would be presented as simple hyperlinks leading to the third party website (such as kulturminnesok.no, digitaltfortalt.no).

4.2.2 User Interfaces

Since the frontend of the whole system shall be developed in this project, the description of user interface shall be considerably stressed.

We can characterize the nature of the system as crowdsourced (or edge-dominant, in general), as the content shall be generated and modified by users to some extent. Due to this, there is not going to be any backend for data entry, the only point where the system is exposed to human beings is the web application frontend.

Our key quality attribute, usability, relates tightly to user interfaces. For this reason, all usability-related requirements on user interface are listed in section 4.3.3 as quality attribute requirements.

Web application front end

As user requests that the initial prototype application shall be developed for use on tablets. Typical usage of the system suggests that it will be used in field, which implies using the application on smartphones and tablets.

Both smartphones and tablet have some inherent characteristics that have to be respected when designing the user interface and the means of interacting with it.

The screen sizes of current smartphones vary a lot, with the most phones with screen size from 3.5 inches (9 cm) up to 5.5 inches (14 cm). Screen sizes of tablets vary between values from 9 inches (22 cm) to 11.5 inches (29 cm). In addition, these devices can be used in either landscape or portrait mode.

Such a diversity in screen sizes makes designing the user interface very challenging. On one hand, we want to maximize the screen usage even on large-screen devices, on the

other hand, we must make the application usable even on devices with screen area up to 9 times smaller.

Another characteristic property of those devices is that their interaction mechanisms are somewhat limited.

1. These devices are generally missing a mouse, and thus a mouse cursor. Direct pointing is done by using finger upon touchscreen. This is however less precise, so the user interface must be accustomed to that.
2. Devices usually miss a hardware keyboard. This is surrogated by having a software keyboard on-screen. This however takes up some of the valuable screen real estate. This kind of keyboard is additionally not as usable as a real hardware one, so the user is not tempted to use it as much., therefore use of any hotkeys is out of question.

4.2.3 Software interfaces

We imply that the server and the client interoperate and are developed to work with one another. The required interfaces and communication schema is to be determined later on.

4.2.4 Communication interfaces

Another inherent characteristic of mobile devices (smartphones, tablets, etc.) is the unreliability and speed of network connection. Since this application is a part of a client-server system, the communication link between those is critical.

The main issues with mobile data networks are frequent interruption, big latency and small bitrate. Sometimes the connection can as well be unavailable at all. This results in the data connection being slow and unreliable which has a great significance on the performance of the application and its usability. As developers of the web application, we need to take this into consideration.

There are several techniques how to deal with mentioned issues or at least mitigate their impact.

Unavailability of connection

To deal with frequent interruption and unavailability of connection, an off-line mode of the application could be implemented in the future, which would enable the user to work without a network connection (in a limited way).

This was described by the customer as to be a low priority requirement, not to be implemented in this phase.

Small throughput

To deal with small throughput of the network, the updates of the content in the client application could be implemented using delta updating. This would mean only new and changed data are sent from the server, data that did not change are used from the local storage.

This issue is not to be addressed in this phase of the project and may be implemented later on.

4.2.5 Product functions

Frontend

The key functionality expected in the frontend is management of virtual walls (creating, editing), stories on them (creating, editing) and social interaction (commenting, sharing).

Backend

The key functionality of the server is to support the features of the client by storing the data in the database and retrieving it back. As said above, the content is stored in the system database. The server shall expose an API for accessing the items from database (walls, stories) for access from the frontend.

4.2.6 User characteristics

As this app shall be accessible to open public, we can neither anticipate users' educational level, nor experience, nor technical expertise. We can however assume basic computer literacy.

Documentation from customer features a scenario how this application would be used in class in elementary school. We may anticipate that some of the users can be as young as 10 years for example.

See chapter 4.3.3 for requirements on usability.

4.2.7 Constraints

HTML5 use

Since use on many different devices and platforms is expected and necessary, customer requested that the frontend would be a web application. As the system is crowdsourced and heavily relies on users, it it necessary that it interacts very well with the users and is visually appealing. Also, for reasons stated in 4.2.4, local storage is necessary to enable off-line mode (in the future). This implies the use of cutting-edge technologies for the development.

For this reason, customer suggests that the application will be developed in HTML5.

4.3 Specific requirements

4.3.1 External interfaces

As the application developed is going to be a client/server where the client is a web application, we will use HTTP protocol for data exchange. Data transferred over HTTP in our case will be in HTML and XML or JSON formats. This is to be determined later.

Virtual Wall Client/Server interface

VirtualWall client is a classic web application that runs in a web browser. This implies that is will be downloaded via HTTP as a HTML document.

Once the web application is open in the browser, data transfers with the web server can be done using AJAX to retrieve the data in XML/JSON format from the server.

Format of the messages shall be determined later.

Social networks

Walls and stories shall be easily shared using social networks like Facebook or Twitter. Custom button for sharing/liking shall be available for each story or wall that user wants to like or share.

Sharing in this case means posting a link on user's social network profile.

4.3.2 Functions

The functional requirements are organized according to object categories.

Users

F1	System shall let users to register an account within the system

Table 4.2: Users

Walls

F2.1	System shall enable management of virtual walls
F2.1.1	System shall let registered users create a virtual wall
F2.1.2	System shall let owner delete a virtual wall
F2.1.3	System shall enable owner to add contributors to wall (their stories for this location shall appear on the wall)
F2.2	System shall let users filter and view walls
F2.2.1	System shall let users to filter walls by popularity
F2.2.2	System shall let users to filter walls by tags
F2.2.3	System shall let users to filter walls by metadata (wall owner, location)
F2.2.4	System shall let users to filter walls by story authors
F2.2.5	System shall show user a list of walls that feature his stories
F2.3	System shall let registered users comment on walls
F2.4	System shall let users like and share wall over social networks (facebook, twitter). See 4.3.1 for details

Table 4.3: Walls

Stories

F3.1	System shall enable management of stories on virtual walls
F3.1.1	System shall let registered users add story to a location
F3.1.2	System shall enable the stories to contain text, hyperlinks, video, pictures, audio
F3.2	System shall let users filter and view stories from a particular wall
F3.2.1	System shall let users to filter stories by popularity on a particular wall
F3.2.2	System shall let users to filter stories by tags on a particular wall
F3.2.4	System shall let users to filter stories by metadata (author, media duration) on a particular wall
F3.3	System shall let users comment on stories
F3.3.1	System shall notify story author about new comment
F3.3.2	System shall enable users to flag a comment as a spam
F3.3.3	System shall notify story author about spam-flagged comment
F3.3.4	System shall enable story author to remove a comment
F3.4	System shall let users like and share stories over social networks (facebook, twitter). See 4.3.1 for details
F3.5	System shall show user a list all his stories, and for each story a list of walls featuring the story
F3.6	System shall notify the user when last wall featuring his story was removed

Table 4.4: Users

4.3.3 Software system attributes

In this section, we describe several quality attributes this system shall have.

Maintainability

Customer stated that the system shall be well documented and easily extensible and modifiable. These concerns are related to the quality attribute of maintainability.

Customer requests that the project documentation shall include description of all interfaces, architectural description and sequence diagrams to illustrate how does the system work.

Portability

As customer requested, the front end application shall be developed in order to be working on various devices, ranging from small smartphones to desktop stations.

Quality attribute scenario to evaluate the system shall be determined later.

Usability

From the description of typical usage (mentioned in 4.2.6) and characteristic problems listed in 4.2.2, we came up with several usability scenarios that shall evaluate the usability

quality of the software system. As usability is a big concern in this project, usability scenarios and tests shall be included in the report under Test Procedures Specification.

These scenarios shall evaluate, whether the application suits the needs of the audience in the means of usability. These shall be derived from use cases and requirements.

4.4 Use case diagrams

Same categorization as in Functions.

User registration

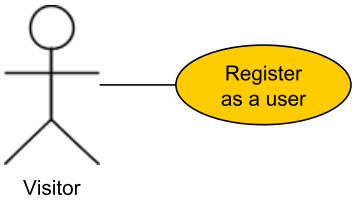

Figure 4.2: Use case diagram for Users

Walls

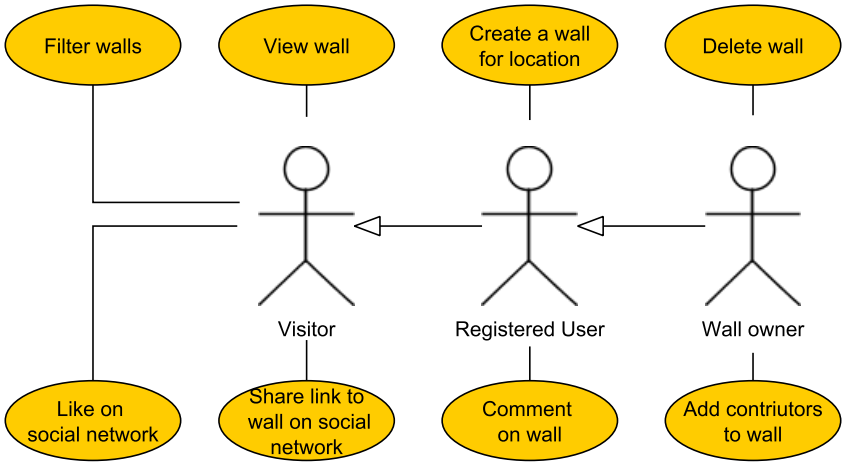

Figure 4.3: Use case diagram for Walls

Stories

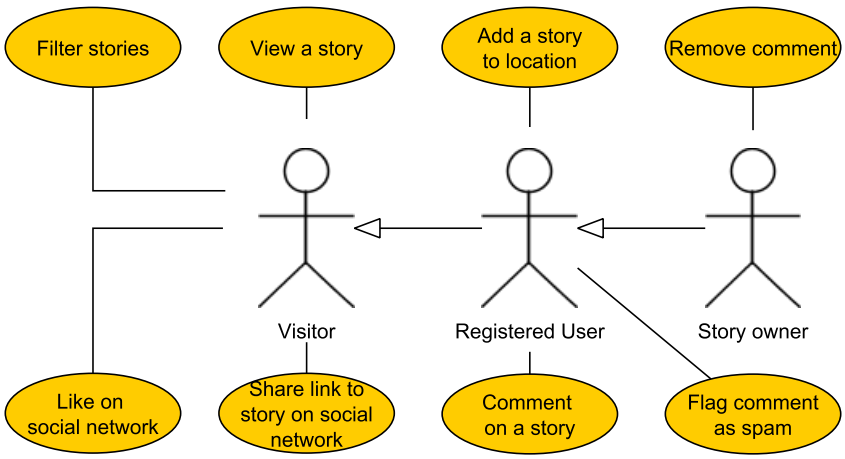

Figure 4.4: Use case diagram for Stories

Chapter 5

Architecture

5.1 Architectural drivers

In the following the document, the following items have been used as the main drivers for the architecture.

5.1.1 Maintainability

A maintainable code base makes for a developable code base. Achieving a maintainable base will make development easier in the short run and maintenance less expensive in the long run. It will also be easier for other teams to continue the development upon the same code base.
Quality requirement: Modifiability

5.1.2 Portability

The customer requested that the system should be usable on different smartphone platforms and also on a normal computer. As such portability is key in order to lighten the workload required to achieve this goal.
Quality requirement: Modifiability

5.1.3 Division between logic and presentation

Division between logic and presentation makes the system more modular and modifiable.
Quality requirement: Usability, Modifiability

5.1.4 Short development time

Given the relatively short duration of this project, the architecture should make for fast development, while not sacrificing quality.

5.2 Stakeholders and concerns

5.2.1 Developers

The architects and developers are concerned about ease and speed of development. As part of this also comes the elegance of the resulting architecture with those in mind that are to continue the development or maintain an operational system.

5.2.2 Customer

The customer is interested in seeing a highly usable system that can run on multiple different devices and be easy to build upon, should they decide to use the prototype as starting point for an actual product.

5.2.3 End users

End users are mostly concerned about the ease of use of the system. This includes the stability and performance as well as the normal usability.

5.2.4 Course staff

The course staff is mostly concerned about the quality of the documentation and its readability.

5.3 Selection of Architectural views

"The 4+1 View Model of Software Architecture" [13] outlines several ways of presenting different aspects of the architecture, useful to different persons. The article uses a variant of Booch syntax to visualize the views, but in this document we will use standard UML diagrams.

This description will focus on the first three views, logical, process and development view.

5.3.1 Logical view

Purpose: The logical view will give us an understanding of how to layout our code in a meaningful way. It will show how classes and modules relates to each other, and as such help us identify common relations and tasks and then generalize them.
Stakeholders: Developers, course staff
Form of description: Class diagram

5.3.2 Process view

Purpose: The process view serves to show how the modules and classes communicates with each other, and explain the system processes.
Stakeholders: Developers, Customer, End users, Course staff
Form of description:

5.3.3 Development view

Purpose: The main matter of concern for the development view is the development process. It serves to show which parts of the system depends upon others.
Stakeholders: Developers
Form of description: Architecture Layer Diagram

5.4 Architectural Tactics

5.4.1 Modifiability

Increase cohesion

By making every part of the system do only one thing, but do it well, it will be easier to change parts of the system or reuse it in other parts.

Decrease coupling

By reducing the coupling the change or modification of parts of the systems will have less impact on the rest of the system. Thus making modifications easier.

Information hiding

Information hiding allows to provide a stable internal API. This will reduce development time, increase testability and also provide better modifiability.

5.4.2 Usability

User initiative

By allowing the user to abort operations, undo or redo the user will feel more comfortable and secure while using the application.

User model

By storing the users credentials, the user will not have to provide the credentials every time he executes an action, like requesting content.

5.4.3 Testability

Limit complexity

By doing what can be done simple, in a simple way, the system will be less complicated to test and more likely than not have less bugs.

Local state

By keeping the state local, the complexity of state machines with multiple participants are avoided. Make central access point stateless, and keep track of the state on the local devices.

5.5 Architecture and design patterns

The overall architecture for the whole system is client-server, where the handheld device or the web browser requests information from the server, which in turn returns it. The advantage of using client-server is that multiple devices might access the same information without needing to download an entire library every time someone updates some information. Using client-server instead of peer-to-peer simplifies keeping the integrity of the information and securing the users data.

5.5.1 Client

On the client Model View Controller will be used in order to separate concerns and keep the code as clean as possible. By dividing user interface, data and logic in different blocks, we increase modifiability and maintainability as well as increasing the speed of development as subdividing tasks will be easier.

The client might even be Model View Presenter (MVP). This uncertainty comes the fact that the technology was too badly documented, so we are looking for alternatives.

HTML5 will function as an adapter that lets us run the application on multiple devices, including Android, iOS and normal web browsers.

5.5.2 Server

The server will be a stateless REST services built on Play Framework. This gives a simple client-server approach where the server consists of some simple layers.

The server will be fully stateless which will help on scalability and availability as well as simplifying modifiability.

5.6 Views

5.6.1 Server

Logical view

The logical view is quite simple due to the fact that Play! Framework is doing most of the heavy lifting. Each logical entity has its own controller/model pair that handles fetching, storing and modifying the relevant information.

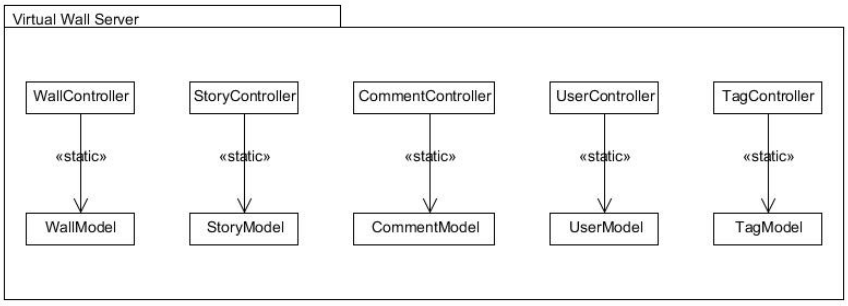

Figure 5.1: Server - Logical view

Process view

The process in the server is quite simple as it is modeled on the way HTTP works, and the way browsers threats HTML. The client sends a request to an URL and the Play! Frameworks main controller delegates the responsibility of delivering data to the correct controller based on the URL requested. If the content received has connections to other types of content on the server, the client will have to send a new request for that content.

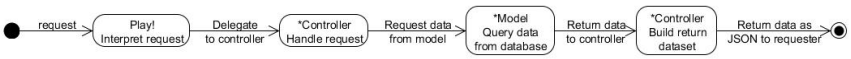

Figure 5.2: Server - Process view

Note that *Controller and *Model means every corresponding pair of models and controllers.

Development view

DATABASE represents all the data. In this layer the work of structuring the data and figuring out the database design is also included.

MODELS represents the classes that will be used to work with the database from the rest of the backend code. The models will abstract away the database, so that it can be changed in its entirety if needed without changing any other code.

CONTROLLERS represents the classes that handles the requests and returns data to the client or sends it to the model for storage in the database.

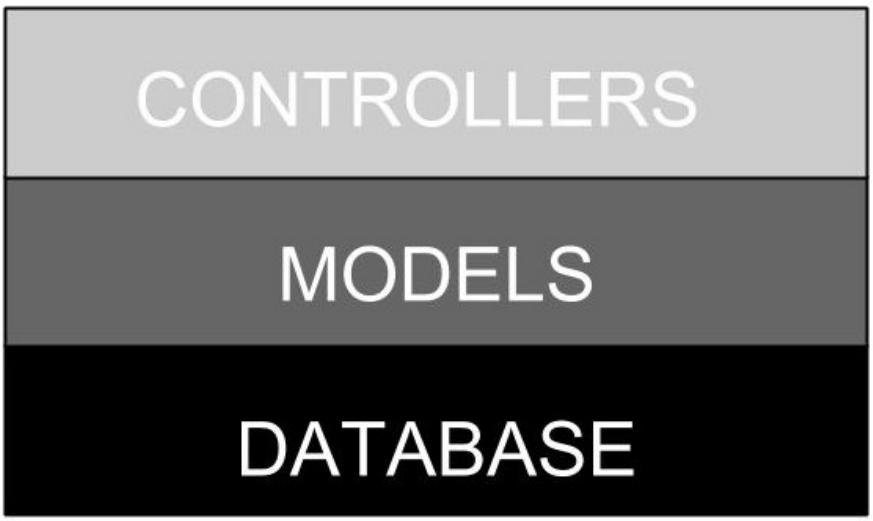

Figure 5.3: Server - Development view

5.6.2 Client

Logical view

Process view

Development view

5.7 Architectural rationale

Chapter 6

Testing

6.1 Test plan

The items and the form and content of the each test document is based on the IEEE Standard 829 [14]. So according to the definition of test plan in the standard: "The test plan prescribes the scope, approach, resources, and schedule of the testing activities. It identifies the items to be tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan".

6.1.1 Test plan identifier

Let this test plan to be identified as "VirtualWall_TESTPLAN_1".

6.1.2 Introduction

The Virtual Wall software system consists of two subsystems:

- backend/server
- frontend/client (mobile/web application)

The backend is the part of the system where all the data is stored. The backend serves the clients with the data on demand.

The client is the part of the system that presents the data to the user and that directly interacts with the user.

6.1.3 Test items

Following items thall be tested:

- The source code of server subsystems will be tested.
- The source code of client subsystems will be tested.
- The user interface of the client subsystem will be tested.

6.1.4 Features to be tested

All functional requirements listed in 4 for this project shall be tested, unless listed as features not to be tested under 6.1.5.

6.1.5 Features not to be tested

There are no items known not to be tested.

6.1.6 Approach

The overall approach is to perform a system test of all features listed under 6.1.4 using tests cases specified in 6.2.

During the development, the source code shall be tested using unit tests prepared by developers themselves. This simplifies the integration by making problems to be found early.

Resulting source code of both subsystems shall be tested using black-box testing. This enables us to abstract from the source code and only focus on the output and outcomes of actions, and compare this with the expected results. These tests are going to be based on test cases specified below and will be performed by black-box testers.

Apart from black-box testing, we shall perform a code inspection to assure desired quality attributes (listed in 5.1) such as modifiability. This shall be done by white-box testers, i.e. members of the development team themselves.

All testing shall be done according to the time plan specified in the schedule table below.

Testing of the user interface shall be done by intended users of the system by means of usability testing.

6.1.7 Item pass and fail criteria

Meeting the preconditions for each test and receiving a corresponding expected result are criteria for successful passing the test. The test is evaluated as failed otherwise.

6.1.8 Test deliverables

The documents to be delivered shall include:

- SRS (System Requirements Specification)
- Test procedures specification (test cases)
- Test logs (test results)
- Test summary report (evaluation)

6.1.9 Environmental needs

Backend subsystem shall be deployed on a server running JRE 1.6 and an instance of PostgreSQL database. The database needs to contain specified schema with structure according to DDL scripts.

Server computer needs to be connected to the Internet. Specified port shall be open for server to listen on and reachable from outer network (not blocked by a firewall, etc).

Device to run the client needs to be equipped with a web browser to open the web application in. The device needs to be connected to the Internet.

6.1.10 Responsibles

The table 6.1.10 maps responsibilities to roles in the scope of testing.

Task	Responsible role
Test management	test manager
Test design	test developers test manager
Test preparation	white-box testers test manager
Test execution	white-box testers black-box testers usability testers
Test evaluation	test manager

Table 6.1: Roles allocation for testing tasks.

The table 6.1.10 maps roles in testing to the members of the development team.

Role	Names of allocated persons
Test manager	Simon Stastny
Test developers	Christian Frøystad Odd Fredrik Rogstad Knut Nergård Simon Stastny
White-box Testers	Christian Frøystad Odd Fredrik Rogstad Knut Nergård Simon Stastny
Black-box Testers	TBD
Usability Testers	TBD

Table 6.2: Roles allocation for testing tasks.

6.1.11 Staffing and training needs

For blackbox testing, the test responsible needs to be instructed about the system workflow in order to be able to follow the steps listed in the test cases.

For code inspection, the person performing it must have a detailed knowledge about the inner workings of the system to be able to discover possible problems and/or draw up improvements of the current solution.

For usability testing, the tester shall have (in the direction of this system) no further knowledge than what is expected to be a minimum knowledge needed for the end user to

use the software . This typically includes user knowledge of computer/smartphone/tablet operations and interaction with the device (direct pointing, keyboard, or some characteristic touch gestures). Tester especially should not have an a priori knowledge about the system workflow, since this is something a typical end user will not have.

6.1.12 Schedule

The table 6.1.12 lists tasks and maps them to people responsible for them and dates due to which they should be finished.

Task	Responsible	Due date
Design test cases for system tests	Simon Stastny	2013-10-31
Design usability tests	TBD	TBD
Perform black-box tests	TBD	TBD
Perform usability tests	TBD	TBD

Table 6.3: Test tasks schedule

In case the functional requirements are a subject to change in the future, the design of black box tests may get delayed to absorb the changes in the requirements specification. The black box test design can only be as much finished as the functional requirement specification is.

6.1.13 Risks and contingencies

As far as we are aware, there are no risks related to testing but those mentioned in 2.4 as general risks for this project.

6.1.14 Approvals

The table 6.1.14 lists approvals of completed tasks.

Task	Approved by	Date
Design test cases for system tests	Simon Stastny	
Design usability tests		
Perform black-box tests		
Perform usability tests		

Table 6.4: Test tasks approvals

6.2 Test cases

6.3 Test results and evaluation

Chapter 7

Phases

7.1 Planning

7.2 Sprint 1

7.2.1 Introduction

The aim of this first sprint is to establish a baseline, in terms of technology but also with regard to the social aspects of executing a sprint as well as the amount of work being manageable within the timespan of the sprint.

7.2.2 Requirements

The server as well as the client should have a base satisfying the architectural drivers listed in 5.1.

7.2.3 Pre study

Server

This is the pre study done for the server in sprint 1.

Extensible Markup Language (XML) vs JavaScript Object Notation (JSON)
XML is a extensible markup language created by the World Wide Web Consortium (W3C). It is made to be both human and machine readable. It consists of a developer defined set of nested elements that may contain data or new elements.

JSON [15] is based on two structures: a collection of name-value pairs, and an ordered list of values. The format is made to be human readable and easily parsed and created by computers. It is based on a subset of the JavaScript programming language, and because the customer has asked for HTML5 to be used for the app JSON makes sense as that will be easier to use together with JavaScript than XML.

Representational State Transfer (REST) vs Simple Object Access Protocol (SOAP)
REST [16] is an architectural style that is stateless at its heart, which leads to better visibility, reliability, and scalability. REST was developed in parallel with HTTP/1.1 and the largest system following this architectural style is the internet (according to

[17]). REST is also a good match for data driven applications as the way you handle a REST services is by utilizing the verbs GET, POST, DELETE, PUT, PATCH.

SOAP is much more flexible than REST, but also much more complex. While REST is primarily data driven through resources, SOAP is logic oriented through operations. SOAP is also a better choice for making stateful web services. All communication with a SOAP service happens through XML.

The simplicity of the REST architectural style, and the fact that it allows the developer to choose the format of exchange makes this our preferred way to go.

Play vs Spring
Two of the well known frameworks when it comes to developing webservices in JAVA is Play! Framework and Spring.

Spring is a very extensive framework with support for quite a few protocols and services. The framework also does a lot of work not necessarily apparent to the developer without a thorough investigation into the documentation.

The Play! Framework is more of a core framework that does the basics, but leaves it to the developer to develop the application in mostly his own style. Different from Spring, Play! Framework also reloads automatically every time a change occurs in the source, and in this way contributes to more effective development.

The extensive documentation and speed of development made us choose Play! Framework for the system.

MySQL vs Postgres
The database system most known to the developers is MySQL as this has been used in several earlier courses. Both systems are highly capable relational databases with good coverage of features and excellent performance. There are, however, particularly one issue that separates them, and make one stand out as our database of choice.

In April 2009 Oracle bought Sun [18], and by that gained control of MySQL. Initially Oracle promised that MySQL would prevail and that it and its coming features would stay open source [19]. The situation today shows that they have not honored that promise, and several different versions of MySQL are availiable [20]. This serves to show that the future of MySQL is uncertain, and as such our choice falls on Postgres.

Client

The client requested use of HTML5 on the client side, so that the prototype could be tested by the whole of the test group without having to hand out phones of a specific type, but rather let the test objects use their own devices.

When it comes to real cross platform HTML5 apps, that is to say no need for native code at all, Apache Cordova is a very useful project. Apache Cordova is what was earlier known as PhoneGap [21]. The code was contributed to The Apache Software Foundation in 2011, at the same time Adobe acquired PhoneGaps creators Nitobi.

The way Cordova works is that it exposes native interfaces through JavaScript libraries, so that local web clients might use native features like gyroscope, accelerometer, etc. When the application is built using web technologies, it is compiled into a native webcontainer. In order to do this, one needs to have access to the platform one compiles for. For

Android, you need the Android SDK. For iOS, you need an Apple computer and Xcode. For Blackberry you need WebWorks SDK.

The need for all this different platforms when building defies some of the intention of using a cross platform framework for developing the app, as it would make the workload and expenses grow quite substantially.

This is where Intel XDK comes in handy. It is a simple IDE for native web apps development with Cordova. It also includes an emulator, so that testing across units is much simplified. When ready to test on phones or build final version for deployment, Intel provides building for all platforms with a simple click in interface.

Intel also provides a fast and native looking HTML5, JavaScript and CSS library with their XDK, but due to very poor documentation for systems wishing to retain the qualities of modifiability and maintainability, the library was rejected.

Investigation into best fitted client side framework will continue into the next sprint.

Interaction and user interface

When the sprint started it was quite unclear how the application would look and feel. Within this sprint there has been two iterations on the user interface.

Version one
First thing done was to come up with what kind of main functionality the application needed;

- Some kind of menu system, to navigate in the application
- A function to browse walls
- A way of displaying walls and stories

Another concern was about the definition of a wall and story, thus what kind of information did it need to display.

Menu
We saw two opportunities, a dedicated menu, typically the start up point for the application, and a more hidden menu, that we'll refer to as a side-menu. The dedicated menu is typically made up with buttons, where the user can access all top-level functions. It is easy and straightforward to implement, it is also well known for most users. A more elegant solution is the hidden side-menu, where the user can access the menu wherever he or she is in the application, with just a sliding gesture from right to left on the screen.

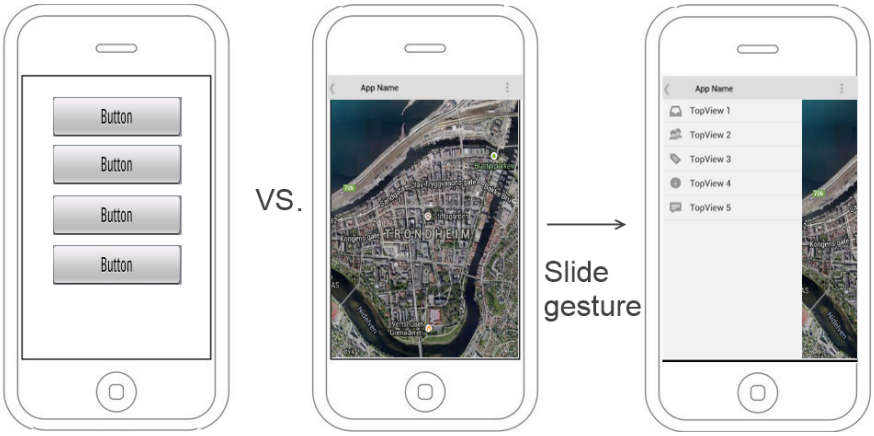

Figure 7.1: Menus, dedicated vs. hidden side-menu

The customer prefered the hidden side menu.

Browsing walls
An important function is to let the user browse and find walls. The most obvious solutions, to use a map and/or a list, were discussed, but no solution was determined.

Another important function is to filter or sort the walls. In a list it is easy to sort the walls on, for instance, popularity or the distance to your location using GPS tracking. There were suggested to use some kind of temperature pins in a map, where the pins shows the user how popular the different places are by using colors. The user could also decide to only see the most popular places by excluding the more cold colored walls, as Figure 7.2 shows. The temperature pins were just one of many suggestion on how to represent walls on a map, others where to use symbols or icons, one colored pins and images.

The customer wanted to focus more on the use of filtering on tags and metadata.

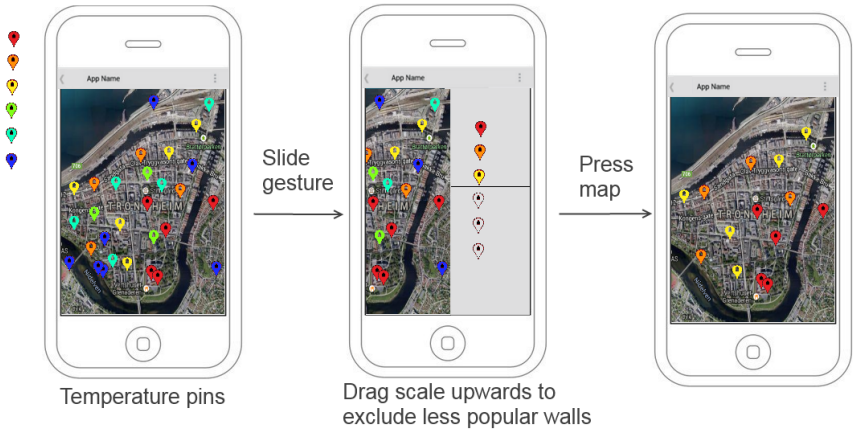

Figure 7.2: Temperature pins, with filtering

Look at walls The main function of the application is to access a wall and browse user created information, that could be stories, pictures, links to other resources, etc.. Two possible solution were suggested; a wall that displayed both user created stories

and information from multiple data sources as in Figure 7.3, for instance, Instagram, Wikipedia, YouTube, Twitter, Google, Flickr, etc., and a more simple version where the focus was on user created data, thus stories and pictures, as in Figure 7.4.

The idea of using multiple data sources came from the danish mobile application, Kulturarv, or Cultural Heritage (both mentioned in 3.2), which blends data from a main database with other geo location data sources like Instagram, Flickr, Wikipedia and Twitter. The customer had not thought about the idea of using multiple data sources, and were therefore quite surprised by the solution. The group and customer agreed that the use of multiple data sources were out of the scope, and decided to focus on the more simple version.

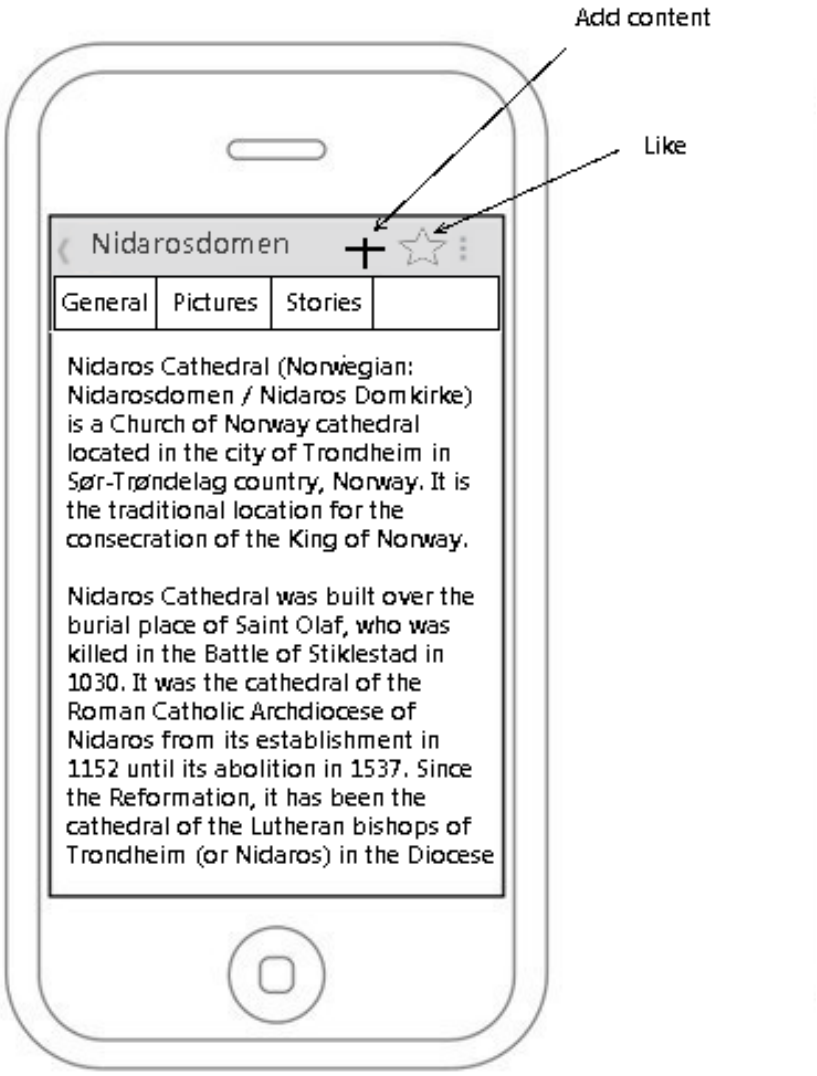

Figure 7.3: Simple wall, with tabs to navigate. Nidarosdomen is used as an example

Figure 7.4: Wall using multiple data sources, with slide gestures to navigate. Nidarosdomen is used as an example

Version two
TBA

7.2.4 Implementation

Server

During this period only the most basic proof of concept was developed for the server. A RESTful access point was made for the developed, with a database. All the basic groundwork was put in place, except for authentication, to develop the rest of the service.

7.2.5 Testing

This sprint no formal testing was done, only functional tests on the server and discussions around the user interface.

7.2.6 Review

We did not manage to finish all the predefined tasks for this sprint. This was partly due to the absence of two group members during most of the sprint period. The delay was also partly due to inexperience with mobile JavaScript frameworks and the lack of real documentation for most of them.

7.3 Sprint 2

7.3.1 Introduction

The goal of the second sprint was initially to build upon the bases put in place by sprint one, but due to a steep learning curve in JavaScript and lack of real documentation for most frameworks this was not the case.

The server has seen development based upon what was done in sprint one, but the client has started from scratch.

7.3.2 Sprint backlog

Item	Outcome
Define the concept of story	Done
Define the main principles of UI	Done
Browse virtual walls	Done in server
Open a virtual wall	Done in server
Basic framework for app with support for communicating with server	Done
Architecture	Partly done

Table 7.1: Backlog for sprint 2.

7.3.3 Requirements

7.3.4 Pre study

Client

The M project
Initially "The M project" [1] came out quite good in comparisons of frameworks for client

[1] http://www.the-m-project.org/

side development for mobile phones. This was due to it being lightweight and what seemed like usable documentation. Its described caracteristica also matched quite good with what we was looking for. Especially the MVC part and the native look and feel of the jQuery Mobile user interface. The MIT license was also very attractive, as it is compatible with the BSD license required for all code produced in this course.

The reasons why The M project was not chosen in the end, surrounds two topics. First, the project has its own build system to help the developers develop more efficiently. Unfortunately this did not work, and produced unrunnable code. One could edit the files after each build in order to make them runnable, but 5 minutes extra per build to clean the code was just not acceptable.

The second problem is of a more serious kind, both for this development team and for those that might follow at a later time. The documentation was vague at best and partly outdated.

Enyo
Enyo[2] was developed by HP for WebOS[22] and is at this time sponsored by LG. It has a very interesting approach to development, one develops features in the app as separate reusable components and glue them together. In that way it feels much more natural to develop in Enyo, and the documentation is excellent. The license, Apache 2.0, is also very attractive as it is compatible with BSD.

The problem with Enyo was that the requirements specified modifiability as one very important factor to take into account. While components can make the app itself modifiable, Enyo does not support MVC in the components and as such one just moves the complexity one level down. When developing components we would still have to handle that lack of ability to structure the code in an efficient way.

Adding support for MVC, or making it easy to use MVC in combination with the rest of the framework, is an ongoing effort at Enyo.

The reason Enyo was not selected is that the MVC support has not yet stabilized, and the documentation is still a bit too thin on the matter.

Sencha Touch
Sencha Touch[3] has excellent documentation and a good community. The problem with Sencha is the incompatible license. Sencha uses the General Public License version 3, which is incompatible with the BSD license.[23]

RhoMobile
RhoMobile is developed by Motorola Solutions[4] to help developers develop software across their different handheld devices. They have also added support for the mobile platforms. This framework has support for developing enterprise applications and has an attractive license in MIT. The reason this framework was not chosen, was because it uses Ruby as well as JavaScript. It was decided that learning one new language would be enough work, if not another one should be added on top.

Titanium
Appcelerator Titanium[5] is a cross platform development framework for mobile and web.

[2] http://enyojs.com/
[3] http://www.sencha.com/products/touch/
[4] http://www.motorolasolutions.com/US-EN/Business+Product+and+Services/Software+and+Applications/RhoMobile+Su
[5] http://www.appcelerator.com/titanium/

The development is done in JavaScript and XML. It is licensed under the Apache Public License version 2. Titanium is a "write once, adapt everywhere" framework[24]. That does not mean that it can't run everywhere without adaption, but that in order to look native it needs adaption. Titanium has an excellent base of documentation and a mature framework supporting easy use of the model view controller architectural pattern.

Titanium was chosen because of its framework, its documentation and the huge community that surrounds it.

7.3.5 Implementation

Server

In this sprint the requirements for the overall workings of the system seemed to be somewhat stable so we designed a data model and implemented it on the server side. This turned out to be a wrong assumption later on.

Client

Sync client
Titanium Alloy is backend agnostic. This means that most of the code does not know, nor need to know, what kind of backend is running. This is accomplished by utilizing an abstraction layer in Alloy called Sync. Sync provides an unified API that can be implemented for each type of backend.
The API has the following methods:

Backbone Method	Sync CRUD Method	Equivalent HTT
Collection.fetch	read	GET
Collection.create (id == null) or Collection.create (id != null)	create or update	POST or PUT
Model.fetch	read	GET
Model.save (id == null) or Model.save (id != null)	create or update	POST or PUT
Model.destroy	delete	DELETE

Table 7.2: API for Sync in Alloy[25]

As the application makes use of REST server, a backend supporting REST was needed. Although a custom backend could have been created, it made more sense to reuse the implementation of others. The napp.alloy.adapter.restapi[6] by Mads Møller was chosen due to its feature completeness and active development.

Models
Models in Titanium Alloy build upon Backbone.js[7] [26]. Therefor the documentation for Backbone.js is relevant in addition to the documentation from Titanium.

A basic model looks like this:

```
exports.definition = {
        config : {
                "columns"           : {
```

[6]https://github.com/viezel/napp.alloy.adapter.restapi
[7]http://backbonejs.org/

```
                "wallId"                : "int",
                "name"                  : "string",
                "description"           : "string",
                "latitude"              : "double",
                "longitude"             : "double",
            },
            "URL"           : "http://stedr.herokuapp.com/walls.json",
            "adapter"       : {
                "type"          : "restapi",
                "collection_name" : "wall",
                "idAttribute" : "wallId"
            }
        },
        extendModel : function(Model) {
            _.extend(Model.prototype, {
                // If adding and editing walls shall be allowed, the
            });
            return Model;
        },
        extendCollection : function(Collection) {
            _.extend(Collection.prototype, {});
            return Collection;
        }
};
```

7.3.6 Testing

Only functional testing was done for this sprint as development went on. No systematic testing was executed.

7.3.7 Review

Once again we would like to have reached a milestone farther ahead at this time, but we have had to spend a lot more time learning new mobile frameworks than we had expected.

Nevertheless, the group has seen increased participation during this sprint, and more group members has become more active. This has lead to us getting more done than in the earlier sprints, and we how this will also mean that we will get even more done in the coming sprints.

Some of our goals for this sprint was only reached for the server. The reason for this is the fact that the group members are more accustomed to Java than JavaScript, and the fact that Play! Framework has better documentation available than the mobile frameworks we have used.

The architecture is only partially done, this is due to the change in mobile framework and requirements. Now that a mobile framework has been chosen and the requirements more stabilized, we hope to have the architecture finished within a relatively short amount of time.

7.4 Sprint 3

7.4.1 Introduction

7.4.2 Requirements

7.4.3 Pre study

Map views, coordinates and projections

One of the main features of the application is to show user location of various virtual walls on a map. Since we decided to use Titanium frameworkk for the development of the mobile application, we are using its own map view (`Titanium.Map.View`) for the display of maps and locations of walls upon them.

The database is now featuring just a few walls, but it is supposed to host much more of them in the future and it would not be optimal for the performance and operational reasons (such as costs of data transfers on mobil internet) to fetch all the walls when user is using the application.

The user is only interested in walls he could actually see on the map, so we decided to take an approach where we fetch the data from the server while filtering it upon geographic coordinates. The map view supplies us with a geogaphical coordinates of the center of the displayed map and with west-to-east distance and noth-to-south distance. This makes it possible to compute the coordinates of the north-west (i.e. top-left) corner of the map view, and the south-east (i.e. bottom-right) corner of the map view. Then the application fetches the data from the server, asking it to filter only those walls, which have coordinates in the rectange defined by those two corners.

It is a well-known fact that planet Earth is not flat. For this reason, various map projections are used to transform the surface of Earth (3D) onto a plane (2D). Google Maps, which provide the map tiles for the map view, are using a variant of so called Mercator [8] projection. This map projection was used in naval navigation for showing constant bearings (*loxodromes*, also known as *rhumb lines*) as straight lines on the map. [27, 28]

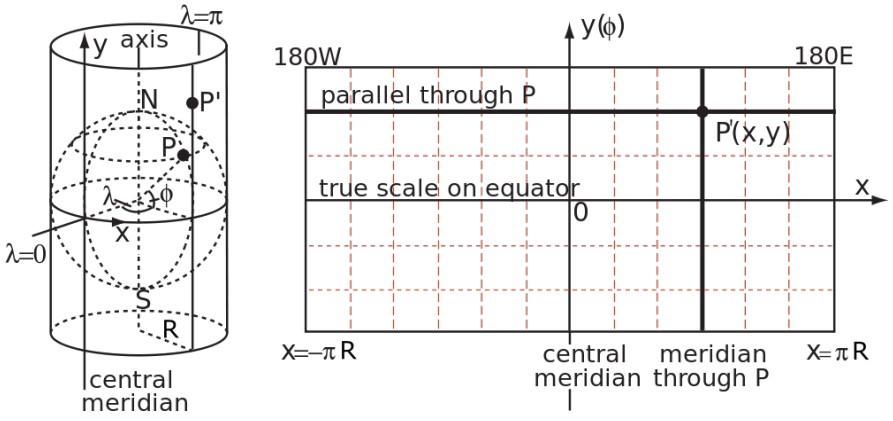

Figure 7.5: Mercator cylindrical map projection basics. [29]

[8] Named after Flemish geographer and cartographer Gerardus Mercator, who firstly used it in 1569 for his map titled *Nova et Aucta Orbis Terrae Descriptio ad Usum Navigantium Emendate Accommodata*, which is Latin for *New and more complete representation of the terrestrial globe properly adapted for use in navigation.*

As seen in 7.5, it is a cylindrical conformal projection, so it preserves local angles (which is actually one of the reasons Google Maps use it), but is unfit for use in high-latitudes as the map extends infinitely North and South (so called *polar exaggeration*). This shall be fine for use until we need to add a wall for a location near to one of the geographic poles (which are streched infinitely along the top and bottom edges of the map). This could be a problem is we want to add a wall for example for the Amundsen–Scott South Pole Station located just few meters from the geographical souch pole and which would and stretch widely east to west. [27, 28]

7.4.4 Implementation

Server

Upon receiving changes proposed by customer, the data model was stashed and reworked to acomodate the proposed changes. As the concept of walls and stories finally settled down, we were able to implement API for retrieving walls based on geographic coordinates and to retrieve stories on walls.

As requested in the last batch of change requests from customer, stories shall be stored in an external database called *Digitalt Fortalt*. This database provides an API for search which we call to retrieve the stories. Our application does not write any data to *Digitalt Fortalt*, stories need to be created manually on their website.

Walls on the other hand shall be stored in out system database. We now store each wall with its geographic coordinates in a database table and filter as described in 7.4.3.

Data stored in our database are retrieved by classes in the `models` package. The models are extending `play.db.ebean.Model` class and use standard JPA[9] annotations.

The data from *Digitalt Fortalt* are retrieved from the API by classes in the `retrievers` package. The data are fetched and parsed by using the *jsoup* Java HTML parser. We found this library to be more usable than *DOM* and *SAX* parser implementations in Java (due to various reasons such as usage of `Iterable` interface, which make the work with collection-like structures much easier).

[9] Java Persistence API

7.4.5 Testing

7.4.6 Review

7.4.7 Phase evaluation

7.5 Sprint 4

7.5.1 Introduction

7.5.2 Requirements

7.5.3 Pre study

7.5.4 Implementation

7.5.5 Testing

7.5.6 Review

7.5.7 Phase evaluation

7.6 Sprint 5

7.6.1 Introduction

7.6.2 Requirements

7.6.3 Pre study

7.6.4 Implementation

7.6.5 Testing

7.6.6 Review

7.6.7 Phase evaluation

7.7 Finalization

Chapter 8

Evaluation

Bibliography

[1] Compendium: Introduction to course TDT4290 Customer Driven Project, Autumn 2013. 2013;Available from: http://www.idi.ntnu.no/emner/tdt4290/docs/TDT4290-compendium-2013.pdf.

[2] Code Style Guide [homepage on the Internet]. Play! Framework Development Team; [cited 2013 Oct 16].

[3] Coding Best Practices [homepage on the Internet]. Appcelerator; [cited 2013 Oct 16].

[4] What is Cultural Heritage [homepage on the Internet]. Culture in Development; [cited 2013 Oct 7]. Available from: http://www.cultureindevelopment.nl/Cultural_Heritage/What_is_Cultural_Heritage.

[5] Introducing UNESCO: what we are [homepage on the Internet]. UNESCO; [cited 2013 Oct 7]. Available from: http://www.unesco.org/new/en/unesco/about-us/who-we-are/introducing-unesco/.

[6] World Heritage List [homepage on the Internet]. UNESCO; [cited 2013 Oct 7]. Available from: http://whc.unesco.org/en/list/.

[7] Kulturarv app [homepage on the Internet]. CodeUnited; [cited 2013 Oct 7]. Available from: http://codeunited.dk/kulturarv/.

[8] 1001 fortællinger om Danmark [homepage on the Internet]. Heritage Agency of Denmark; [cited 2013 Oct 7]. Available from: http://www.kulturarv.dk/1001fortaellinger/.

[9] Sommerville I. Software Engineering. 9th ed. Harlow, England: Addison-Wesley; 2010.

[10] Smith P. Waterfall model diagram [image on the Internet]; [cited 2013 Oct 7]. Available from: http://en.wikipedia.org/wiki/File:Waterfall_model_(1).svg.

[11] Royce WW. Managing the Development of Large Software Systems: Concepts and Techniques. In: Riddle WE, Balzer RM, Kishida K, editors. ICSE. ACM Press; 1987. p. 328–339.

[12] IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998. 1998;Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=720574.

[13] Kruchten P. The 4+1 View Model of Architecture. IEEE Softw. 1995 Nov;12(6):42–50. Available from: http://dx.doi.org/10.1109/52.469759.

[14] IEEE standard for software test documentation. IEEE Std 829-2008. 2008;Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4578383.

[15] JavaScript Object Notation [homepage on the Internet]. The Internet Society; [cited 2013 Oct 7]. Available from: http://json.org/.

[16] Fielding RT. REST: Architectural Styles and the Design of Network-based Software Architectures [Doctoral dissertation]. University of California, Irvine; 2000. Available from: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[17] Representational state transfer [article on the Internet]. Wikipedia, The Free Encyclopedia; [cited 2013 Oct 7]. Available from: http://en.wikipedia.org/wiki/Representational_state_transfer.

[18] Oracle Buys Sun [article on the Internet]. Oracle corporation; [updated 2009 Apr 25; cited 2013 Oct 7]. Available from: http://www.oracle.com/us/corporate/press/018363.

[19] Oracle Makes Commitments to Customers, Developers and Users of MySQL [article on the Internet]. Oracle corporation; [updated 2009 Dec 14; cited 2013 Oct 7]. Available from: http://www.oracle.com/us/corporate/press/042364.

[20] MySQL Editions [homepage on the Internet]. Oracle corporation; [cited 2013 Oct 7]. Available from: www.mysql.com/products/.

[21] PhoneGap [homepage on the Internet]. Adobe Systems Inc; [cited 2013 Oct 7]. Available from: http://phonegap.com/about/.

[22] About Enyo - History [homepage on the Internet]. Enyo Team; [cited 2013 Oct 16]. Available from: http://enyojs.com/about/#History.

[23] The Free-Libre / Open Source Software (FLOSS) License Slide [homepage on the Internet]. David A. Wheeler; [cited 2013 Oct 16].

[24] Supporting Multiple Platforms in a Single Codebase [homepage on the Internet]. Titanium Documentation Team; [cited 2013 Oct 10].

[25] Alloy Sync Adapters and Migrations [homepage on the Internet]. Titanium Documentation Team; [cited 2013 Oct 16].

[26] Alloy Models [homepage on the Internet]. Appcelerator; [cited 2013 Oct 16].

[27] Furuti CA. Projections for Navigators and Radio Operators [homepage on the Internet]; [cited 2013 Oct 7]. Available from: http://www.progonos.com/furuti/MapProj/Normal/ProjNav/projNav.html.

[28] Rankin B. Navigating And Measuring Large Areas [homepage on the Internet]; [cited 2013 Oct 14]. Available from: http://www.radicalcartography.net/?projectionref.

[29] Cylindrical Projection basics [image on the Internet]; [cited 2013 Oct 14]. Available from: http://en.wikipedia.org/wiki/File:Cylindrical_Projection_basics2.svg.

Appendix A

Assignment

Appendix B

Timesheets

Appendix C

Documentation