Twilm
ONELINER ABOUT THE PROJECT HERE

NTNU – Trondheim
Norwegian University of
Science and Technology

SPECIALIZATION PROJECT? =P
October 30, 2013
Team: Fredrik Persen Fostvedt and Martin Havig
Supervisor: Heri Ramampiaro

Acknowledgements

Thanks Mum!

Abstract

Skal brukes til å fange leserens oppmerksomhet Skal summere innholdet av raporten Bør være mellom 1/2 - 1 side lang Bør innholde: kontekst, mål, og hva som er blitt oppnådd. Også hva som er nytt / ny oppdagelse bør være med

Some awesome abstract, example:

This report will give the reader an insight into the details of the design, development and implementation of the task given in the course TDT4290 - Customer Driven Project, taught at NTNU - the Norwegian University of Science and Technology. The customer is Netlight and they have presented the group with the task of breathing new life into the console.

Web-applications these days are leaning against a mouse-controlled, web-fronted design. This has taken away much of the efficiency of power users, who have traditionally used terminal applications on a daily basis, and had the system in their fingers.

A hybrid web-fronted/console design would be a possible solution to this problem: The power user can make use of their full potential through a console whilst the objects are presented in the web-interface.

This is a proof-of-concept task, and all research done will be documented and used to argue for and against the solutions used and not used. Everything from the planning of the project startup and preliminary-study to the complete conclusion is described in this report.

The approach to investigate and solve this problem starts with a thorough study of relevant technologies, and how this can be made possible. The conclusion of this study allows us to create a system which showcases the real potential of our soution. Through this whole process we have a close work-relationship with our customer to ensure his desires and expectations for the project are met, and that our conclusions and findings boosts future research in this field.

Acknowledgments

Some awesome Acknowledgments

Vet ikke hva som er best, acknowledgments før eller etter abstract, tar det l8er

I hoved-kapitlene beskriv selve forskningen, og dens resultater. Inkluder nok teknisk detalj slik at leseren klrer å bedømme nøyaktigheten og originaliteten av arbeidet ditt. Denne beskrivelsen av forskningen bør være den største delen av avhandlingen din, hvordan den er delt inn i kapitler vil avhenge av arbeidet ditt.

Preface

Some awesome preface, example:

This report serves as the primary piece of documentation of our attendance to the TDT4290 Customer Driven Project course, hosted at the Department of Computer and Information Science at the The Norwegian University of Science and Technology, in the autumn of the year 2012.

Our customer was the Netlight consulting company represented by Peder Kongelf.

We would like to thank our supervisor, Zhu Meng, for his feedback on our work, and also Peder Kongelf for presenting us with the opportunity to work on such an interesting project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Context; motivation for the project; problem statement; outline of dissertation.

I introduksjonskapittelet, forklar kort konteksten med og motivasjonen bak arbeidet ditt, forklar problemet som du prøver å løse og forklar hvorfor dette problemet er verdt å løse.

Wonsole is a student project under the TDT4290 course in IDI, NTNU. The project is intended to give all its students experience in a customer guided IT- project and the feel of managing a project in a group. Every phase of a typical IT- project will be covered. This report will serve as documentation of our work. This includes our work progress, the technologies we used, our research findings and so on. The introduction chapter is meant to describe the project, our goals and briefly the involved parties.

1.1 Purpose

1.2 Motivation

1.3 Context

1.4 Project

This is a proof of concept project. The underlying task is to research and develop a system where power users can benefit from a console. The concept aims to ease the workload of a power user who is working with object editing, and to see how the efficiency of a console might prove to improve the work. The power user is usually a user who often works with the system over a longer time, and is in depth familiar with the system. We will research already existing systems of this kind, and look at the possibilities and advantages of such a system in a chosen domain.

We have chosen a library as our domain, and this will be used to explore and test the concept. The library domain is chosen since it possesses potential for the existence of power users and multiple input forms which could be made more efficient through a console. This domain also opens the opportunity to test our system on for instance employees on campus, which is important for the proving of the concept.

Goals

1. Provide extensive documentation and a successful presentation of the end product.
2. Create a working prototype of a system where a scripting console is embedded into a modern web interface. The console should provide access to viewing and modifying the underlying data objects of the system's domain via a Domain Specific Language(DSL).
3. Investigate the ramifications of the added functionality, in terms of usability and technical aspects.

It is important to note that the report is in focus. It will be the cornerstone of the prototype, to not just ease further development, but also to amplify the reasons for the choices we make.

1.5 Project name

Project name is important project identificator. It should summarize main project goal or functionality. In real projects, this is often a trademark, or a name that reflects the name of the company. For our project, the main concern was to create a descriptor that reflected the root concept, namely incorporating a console into a web application.

We held a brainstorming meeting specifically to create a name for the project. Early in the process, we created a list of words that could describe our project functionality or goal. Some keywords:

- Master, Super User
- Console, Terminal, Command Line
- Web Application, GUI
- Internet, Networking
- Text, Keyboard

From these keywords we attempted to compile a list of candidate names:

- Console 2.0
- Wonsole
- Wensole
- Websole
- Werminal
- interCLI

After a discussion and a brief investigation into which names were already taken by other projects, we chose the name *Wonsole*. The project name alone can be a little confusing, so we added the subtitle: *The new web console for power users.*

1.6 Structure of Report

This report describes the development of bringing the scripting experience to the power user. The report is structured after the course of the project, and gives the reader insight into the development of the system.

Chapter	Description
Chapter 1	The introduction chapter introduces the problem to the reader, introduce the members and stakeholders of this project and explain the motivation for doing this project. After reading this chapter, the reader will be left with an overall view of the projects outline and goals.
Chapter 2	The Preliminary Study chapter describes the work and research done, and road taken to the choices we made. This includes technologies, methodologies, testing and version control.
Chapter 3	The Project management chapter deals with how the team will work to reach the desired goal. This includes the structure of the team, the plan for the project, constraints the team and project is put under and quality assurance.
Chapter 4	The Requirements chapter describes the backlog and rationale for this. Use cases for the system is also placed in this chapter together with user stories.
Chapter 5	The Test Plan chapter contains the approach for testing with the overall test plan for the project, and the test scheduling.
Chapter 6	The Architecture chapter explains the structure of the system, how it is put together and why it is put together the way it is.
Chapter 7 - 10	In the sprint chapters the reader can see how the product has developed during the time of the project. This includes planning, architecture, the implementation, testing and evaluation of the sprint.
Chapter 11	Evaluation chapter includes the team's view on the team dynamics, relationship with the customer, issues during the project, the planning and the quality assurance.
Chapter 12	Conclusion chapter sums up the project and describes the findings and reflections around this.
Appendix	The appendix contains tables, templates, the test cases and more.

Table 1.1: Structure of Report

Chapter 2

Preliminary Study

Contents

Review of relevant literature; review of similar software products or tools.

I “survey”-kapittelet presenter en oversikt over tidligere relevant arbeid inkludert artikler og evt. eksisterende produkter. Vurder kritisk styrker og svakheter av tidligere arbeid

This chapter is ment to outline the preliminary study of our project. This includes what our technologies aims to achieve and how we will use them to achieve this. Beyond this, the chapter will show the mythology the team chose to use and why this was a natural choice to go with. Research in software testing methods will be included and the best fit for this project. Since this is a prototype project, a considerable time is put into this part of the project, to assure the team makes good choices when it comes to technologies to use.

2.1 Set to work with (another name perhaps)

Describes the existing stuff from twitter and twitter

2.1.1 Netflix

Dataset

somethingsomething

somethingsomething

2.1.2 Twitter

Twitter-API

somethingsomething

somethingsomething

2.2 Concept

Should maybe be in the intro?

The efficiency of the power user can always be increased. And one way of achieving this efficiency boost is through adding new functionality, but for bigger systems, this functionality adding can prove to be troublesome. Switching between the mouse and the keyboard can add up to be time-consuming in the long run, and also inefficient. With an environment where the user then instead can construct their own functionality, and keep their hands on the keyboard, can add tremendous value to the user experience. This can be made possible with a scripting environment, where the user themselves will be given the opportunity to work through a console on their graphical user interface. This can release them from the mouse and let them work more efficiently on the tasks at hand.

2.3 Similar solutions (Evaluation of State-of-the-art)

In this section, we will discuss similar solutions and their relevance to our project.

If this is not state of the art stuff, maybe separate this and that

Product	Counter-Strike
Concept	This game features a console that offers a wide variety of commands, including: - Changing the game options - Altering the game world - Player actions and cheats - Multiplayer communication and administration
Intended use	In a game, this functionality eases development and debugging, as well as increasing the modability and long-term value for players.
Similar products	Similar consoles also exist in other games, such as Carmageddon TDR 2000.
Relationship to our project	Like our project, the Counter-Strike console allows for using a DSL to work with objects in its given domain. It shows that the console can be a powerful tool that comes at a relatively small development cost compared to designing a GUI with a similar feature set.

Table 2.1: Counter-Strike Console

Figure 2.1: Counter-Strike Console

Product	Blender
Concept	This 3D morming software features a Python console with access to the morm data, animation data, etc. It is also common to extend the program using custom Python scripts.
Intended use	In a creative suite, this type of functionality can be used to perform operations that are not directly supported in the user interface. It is particularly useful for programmatically executing repetitive tasks that can be automatized.
Similar products	Similar solutions also exist in other creative tools, including a Python console in GIMP and Nyquist prompt in Audacity.
Relationship to our project	The Blender console exposes the underlying data structures to the user via an already widespread, powerful scripting language. This enables the users themselves to expand upon the functionality of the program and perform operations that would be prohibitively time-consuming to execute manually.

Table 2.2: Blender Console

Figure 2.2: Blender Console

Product	Firefox
Concept	This web browser features a Web Console where it is possible to execute JavaScript commands with access to the objects on the current page.
Intended use	In a web browser, this is useful for web development, prototyping and debugging for websites that use JavaScript.
Similar products	Similar features also exist in a few other browsers, such as Google Chrome.
Relationship to our project	Like our project, the Firefox Web Console offers access to the objects on a web page. It is possible for us to use the same principle, but for features that are useful for the end user and not just the developer.

Table 2.3: Firefox Web Console

Figure 2.3: Firefox Web console

Product	try.mongodb.org
Concept	This database website features a console where the user can execute commands on a dummy database.
Intended use	On this website, the console serves as an educational and demonstrational tool for people who don't want to invest too much time in learning about the database system.
Similar products	Comparable consoles also exist to allow the user to execute arbitrary queries in database administration tools such as PHPMyAdmin.
Relationship to our project	This console allows the user to execute queries directly on a database, and interestingly it allows input of objects with arbitrary structure. We will require persistent storage of our objects on a server, so this technology may be relevant.

Table 2.4: MongoDB Console

Figure 2.4: MongoDB Console

Product	web-console.org
Concept	This project provides shell access to a server through the browser window over HTTP with support for real-time communication.
Intended use	This system is intended to be used for server administration purposes when HTTP may be the only feasible method of connection.
Similar products	Similar solutions include access to unix Shells via VPN.
Relationship to our project	If, in our project, the server is designed to support user input using a DSL in a terminal window, then a solution like this one may provide the necessary functionality to embed a useful console into a Web UI.

Table 2.5: Web-Console

Figure 2.5: Web- Console

These products serve to illustrate that consoles are still applicable for purposes including software development, debugging, learning, extendability and where there is a high demand for flexibility. They can enhance productivity and provide features that would be prohibitively complex or expensive to implement in a graphical user interface. They also show that consoles do exist on the web, and that this is a field that is worth closer examination.

2.4 Development language and technologies (maybe not needed)

2.4.1 Database

For the system we need some kind of persistent storage, a database which all the clients can talk to to get the latest data in the system and to synchronize data from different clients. This database will be placed on a central server which all the clients can communicate with. For the database we are left with a decision between traditional SQL database or a so called NoSQL database. The differences are explained below.

NoSQL

NoSQL databases are a group of new emerging types of databases that are defined by the fact that they are addressing some of the following points: being non-relational, distributed, open-source and horizontally scalable [8]. NoSQL arose from the need to store large amount of data that do not necessarily follow the same schema, or share all the same attributes. One particular type of NoSQL databases that caught our attention early was the document NoSQL databases. The main characteristic of these databases is their schema- less structure. They also differ from SQL by generally not using a structured query language for data manipulation. They are easy to replicate and they offer simple APIs for the clients to communicate with. They are heavily customized for web- applications, and have gained much popularity in the modern web era. Most document NoSQL databases focus on quick replies to requests, as the query operation is by far the most common in a typical web- application. Because they typically are distributed, NoSQL databases are able to store enormous amounts of data, and they are often used within Big Data [14] applications like Hadoop [13], which recently has become a very popular subject within the computer science community. BASE [9] instead of ACID. Document NoSQL databases seemed like they would fit our project quite well. [7, 12]

SQL

The traditional SQL databases is by far the most common way of storing data in the world today. They store data in columns and tables, and add relationships between these tables. As a result they are referred to as relational databases. They focus on query optimization techniques and most of them use some kind of structured query language. Typically supports 4 basic operations which is select, update, rmete and insert. SQL databases are schema defined and follows the ACID(atomicity, consistency, isolation, durability) principles. [10]

Database Alternatives

Below, some of the database implementation available to us are discussed. We mainly investigated document NoSQL databases and regular SQL databases. The most appealing options are introduced below.

MongoDB

MongoDB is a large scale, high availability, robust system. It is a document NoSQL system, so instead of storing the data in tables as you would in MySQL, the data is stored in a document based fashion through for instance JSON with dynamic schemas. This makes the DB easier scalable horizontally. But the mongoDB still possesses the some of the great properties from relational databases, such as indexes, dynamic queries and updates. With mongoDB it is easy to map objects to programming language data types, which makes the DB easy to work with. The embedded documents and arrays makes the join operations less importance, which result in the ability to make reads and writes faster. JavaScripts can also be included in the queries. This makes mongoDB an efficient and high speed data base for storing objects and fetching to a system. MongoDB also has its own access console, where you can use scripting with Javascript language. [1]

CouchDB

CouchDB stores the data in JSON documents(NoSQL) object-oriented, which can be accessed through HTTP. The data documents can be transformed and data fetched with JavaScript. CouchDB has a lot of built in features which makes web development with CouchDB easier. It scales well through replication and is a consistent system, where CouchDB prioritizes the data of the user. CouchDB is multiversion concurrency control based, this is good for intense versioning, offline databases which resync later and master to master replication. The interface to the database is REST based, which is a useful feature when you are developing a web- application. [4, 5]

MySQL

MySQL is the most popular database in the world of open databases. This is because of its high performance, reliability and ease of use. It should therefore be considered for the question of which database system to use. In opposition of the two database systems described above, MySQL is a relational database. This makes it more troublesome to work, with when it comes to JavaScript, than the other two. It is not as well integrated with JSON and will need parsing to be able to work with the clients. This alone is a hard negative towards MySQL. [3]

Conclusion

We decided to go for CouchDB in this project. We are working in a web domain, which CouchDB was designed for. While developing the console we will be working closely with JavaScript objects, and try to find ways of exposing these to the users. JavaScript objects are easily converted to JSON, the format used to store data in document NoSQL databases like MongoDB and CouchDB. As we only have the need to store the actual objects and a limited amount of relations between them, using CouchDB will ease our work considerably, and allow us to do things which would not be possible with a regular SQL database. CouchDB imposes far less restrictions on how you store your data than traditional SQL does. As long as the data is represented in JSON, you can store pretty much store anything you like, even within the same database. This will give us great flexibility when it comes to adding information to specific objects, and also means that objects of the same type can contain different attributes without us needing to create a new schema or change anything in the database. It also gives the user great flexibility in the sense that they can add any information they would like to the different objects, and we the developers don't have to plan for it at all, the database does all this for us. The fact that the customer suggested to us that we could use a NoSQL database, also tipped the scale in direction of the NoSQL alternatives.

Relational databases is the traditional way to deploy databases and they are in widespread use. In many situations they are extremely useful. However relational databases requires you to morm your data up front, before you save anything to the database. Failing to comply with these restrictions will lead to failure, and it is sometimes difficult to create this kind of morm that actually fits real world data. Even though objects may share common attributes, there are bound to be some attributes that are different. This is difficult to plan for in advance, and its the main reason we will not be using a SQL database for this project. For the database, we originally chose to use MongoDB as our document NoSQL database. This was due to the fact that it was better documented and seemed better suited for the system as we originally planned it. Some of the features CouchDB offered wasn't thought of as necessary for the project at the time. During the implementation process we however came to the conclusion that CouchDB actually was a better fit. This was mostly due to its ability to act as an standalone system on a server without the need for other supporting server technologies like Node.js or ASP.NET. Also, the fact that it automaticly creates a RESTful API to access the database turned out to save us a lot of time. As a result of these discoveries we changed to CouchDb during the third sprint.

2.4.2 Client-side web application technologies

Our main focus will be on the client part of the application, since this is the experimental part of our system, and it is this part that is visible to the user through the web browser. It is therefore important to choose a suitable technology for this.

Adobe Flash

A multimedia platform currently owned by Adobe. Is currently the industry standard for multimedia web applications. It excels at animation and 2D games, but its strong points are not likely to be useful in our project. A separate JavaScript is required to perform communication with a server and the development tools are costly.

Microsoft Silverlight

A rich media application framework developed by Microsoft. Useful for multimedia applications, but likely not beneficial for our project.

Java Applets

A technology that allows a Java AWT/Swing application to be displayed in a browser, backed by a Java Virtual Machine. Has excellent performance compared to other popular client side browser technologies. A signed applet can also communicate with a server using traditional sockets. It is possible to embed a scripting engine(for instance JavaScript). However, development effort may prove to become excessively heavy.

JavaScript

JavaScript is a scripting language supported by all popular web browsers. Has extensive frameworks built around it and allows for rapid development. It is also possible to let the user write commands using JavaScript directly.

Conclusion

As a team, we have extensive experience with Java, and less with the other technologies. However, we all have at least some experience with JavaScript, and we believe that it is the better choice for this project: The DSL can be implemented by allowing the user to perform operations on the JavaScript objects using (a subset of) the JavaScript language itself. There are also excellent tools for transfer and storage of JavaScript objects. Furthermore, communication between JavaScript and HTML elements is easily achieved.

JavaScript Related technologies

jQuery
A JavaScript library that simplifies how to use JavaScript to interact with the webpage, notably selection of Document Object Morm elements.

MooTools
A JavaScript framework that, notably, enhances the Document Object Morm and JavaScript's object oriented programming morm.

Dojo
A JavaScript toolkit offering asynchronous communication, a packaging system and systems for data storage. Intended to ease rapid JavaScript web development.

HTML5
A revision of the HTML standard currently still in development. Notably, it supplies support for multi-

media and more advanced user interface elements. Is commonly used in conjunction with JavaScript.

CSS
Cascading Style Sheets, used to specify a consistent look and feel to a series of HTML documents.

We decided to include jQuery in this project, coupled with HTML5 and CCS. jQuery was included because it is a technology we as a group have extensive experience with. It enabled us to easily select DOM elements, which is helpful when you want to create a dynamic web GUI.

2.4.3 Markup Languages

When communicating between the client and the server we need a data exchange format to represent objects and actions. The format has to be able to serialize and deserialize them on sending and receiving. The two alternatives most commonly in use today for solving this problem is XML (Extensive Markup Language) and JSON (JavaScript Object Notation).

XML

In widespread use in a lot of areas as of today and boasts great support and documentation. Originally meant to be a document markup language, but has over the years been used as a data representation language as well. It is suited to describe complex objects and documents, and it is easy to extend. Generally thought of as the more secure option of the two.

JSON

As the name suggest JSON is serialized JavaScript objects, well suited for web development, and very fast. JSON is easy for JavaScript to parse(we will be using JavaScript on both server and client), and the language has built in support for serializing and evaluating JSON data. Its small, simple, and easy to use. JSON is especially good at representing programming-language objects. It has gained a great deal of popularity in recent years, and it is well documented.

Conclusion

Both languages is well supported in almost all web related libraries, and they are both extensively documented. As security is not a main concern of this project, the fact that XML is more secure will not influence the decision. JSON will be used for this project. It covers the functionality we need, and it is generally thought of as the easier language to use. It is perfectly suited for web- development and JavaScript, which is the domain of this project. In addition the developers have more experience in using JSON than XML. JSON is also used to store objects in most document NoSQL databases, which we will be using for this project.

2.5 Software Testing

2.5.1 Testing Methods

The purpose of software testing is to uncover software bugs in the system and to document that the system meet the requirements and functionality that was agreed upon for the system. Testing can be implemented at any stage in the development process, traditionally it is performed after the requirements have been defined and the implementation is completed. In agile development processes however, the testing is an ongoing process. The chosen development methodology will in most cases govern the type of testing implemented in a given project. [11]

Software testing methods are traditionally divided into white- and black- box testing. They differ mainly in how the test engineer derives test cases.

White- Box Testing

White- box testing focus on the internal structures of a system, and it uses this internal perspective to derive test cases. White- box testing is usually done at unit level, testing specific parts or components of the code. This kind of testing focus on how something is implemented and not necessarily why. Unit testing alone cannot verify the functionality of a piece of software is supposed to exhibit. It can uncover many errors or problems, but it might not detect unimplemented parts of the specification or missing requirements.

Black- Box Testing

Black- box testing handles the software as a black- box, meaning it observes the functionality the system exhibits and not the specifics on how it is implemented. The tester only needs to be aware of what the program is supposed to do, he doesn't need to know the specifics on how the functionality is implemented in the code. Black- box testing is typically performed to check if the functionality of the program is according to the agreed upon requirements, both functional and nonfunctional. Black- box testing is usually done at the component, system and release levels of testing. The main advantage of black- box testing is that no programming knowledge is needed to actually perform the tests. This way you can hire someone outside the development team who has had nothing to do with the implementation of the code to write and perform the tests, and you achieve as little ownership of the code as possible. An argument can be made though that this lack of insight in the specifics of the source code will result in repeated testing of some parts of the code, while other parts could be left untested.

Test Driven Development

The principle behind TDD is to develop the code incrementally, along with test for that increment. You don't move on until the code passes its test. The tests are to be written before you actually implement the new functionality. The process helps programmers clarify their ideas of what a code segment is actually supposed to do. The process is often used in agile development methods. Benefits from TDD include:

- Code coverage, every code segment should be covered at least one test.
- Regression testing, check to see if changes in the code have not introduced new bugs.
- Simplified debugging, when a test fails it should be obvious where the problem lies, no need for a debug tool.
- System documentation, the tests themselves act as a form of documentation that describe what the code should be doing.

[2]

Automated Tests

Automated offers the ability to automatically do regression tests, i.e. testing to uncover if any new code has broken a test that previously passed. If we opt for manual testing regression testing will be very time consuming as every test done so far has to be done over again. With an automated testing framework this job will be a lot easier as you can run a great number of tests in a matter of seconds. Most development languages offers libraries for automated testing.

2.5.2 Testing Levels

Testing can be done at many different levels and in different stages in the development process. Following is the most common partitioning of testing levels and a description on each of them.

Unit Testing

Unit testing aims to check specific components, such as methods and objects. Typically you will be testing objects, and you should provide test coverage of all the features of that object. Its important to choose effective unit test cases, that reflect normal operation and they should show that the specific component works. Abnormal inputs should also be included to check if these are processed correctly.

Component Testing

Tests bigger components of the system, and their interfaces(communication with other components). Made up of several interacting objects. Component testing is mainly a tool to check if component interfaces behaves according to its specification.

System Testing

In a given development project there may be several reusable components that have been developed separately and COTS systems, that has to be integrated with newly developed components. The complete system composing of the different parts is tested at this level. Components developed by different team members or groups may also be integrated and tested at this stage.

Release Testing

Release testing is the process of testing a particular release of the system that is intended for use outside of the development team. Often a separate team that has not been involved in the development perform this testing. These kind of tests should focus on finding bugs in the complete system. The objective is to prove to the customer that the product is good enough. This kind of testing could either be based on the requirements of the system or on user scenarios.

User Testing

This is a stage in the testing process in which users or customers provide feedback and advice. This could be an informal process where end- users experiment with a new system too see if they like it and that it conforms to their specific needs. Testing on end- users is essential for achieving success in a software process as replicating the exact working environment the system will be used in is difficult to achieve during development. The end users can help provide feedback on how specific functionality will work in an actual work environment.

Another form of user testing involves the customer and its called acceptance testing. Its a process where the customer formally tests a system to decide whether or not it should be accepted, where acceptance implies that payment for the system should be made. Acceptance testing is performed after the release testing phase.

2.5.3 Conclusion

The concept of TDD is to develop exhaustive tests that specify the system, and then writing code with the goal of satisfying the tests. This is useful in systems where the key functionality is in the form of program logic that can be verified to conform to the specification. It is difficult to write such exhaustive tests in applications that rely heavily on GUIs, network connections and database systems because of the added complexity and heterogeneousness that these features involve. In addition, our prototype's exact specifications are likely to change during development, requiring large amounts of test rewriting in the case of TDD, because the tests will be more numerous and because the tests must be more strict. As a result we have opted not to use TDD in this project. We will however be utilizing unit tests with an automated testing framework where it is appropriate and will harvest some of the advantages linked to TDD, like the automated regression testing.

We will be writing unit tests throughout the implementation process and run these continuously. These tests will not be included in the report as test cases. The test cases will rather comprise of component and system tests. Component tests will be used to test specific components and their functionality as well as their interaction with other components. System tests will be used on the system as a whole to check if it meets the agreed upon requirements. These test cases will be derived from the requirements. We will also try include some acceptance tests to include the customer in the testing process.

We will be utilizing user testing and involve end users to get feedback on the entire system or specific functions. This testing will mainly be used to get feedback on the domain scripting language and how easy it is to understand and use. Preferably it will be done continuously throughout the development process. It is important to involve users as the goal of this project is to ease their workflow. As we are not working with an existing system thats actually in use, there will not exist any real users of this system with experience using it. We will therefore mainly be using our fellow students as test subjects, as they are readily available and technically competent enough to understand the concept and act as superusers. In addition the customer has stated that it will encourage employees from the entire company to us give feedback if we ask for it. Specific solutions can be sent to the customer representative which in turn will relay it to experts on the area it concerns.

2.6 Evaluation (maybe)

To sum up what we saw after the prestudy

Chapter 3

Requirements

Contents

How requirements were captured; discussion of major requirements (referring to Appendix A for details).

I “Requirement”-kapittelet forklar hvordan du satt opp kravene. Men ikke ha med en fullstendig oversikt over kravene her!

In this chapter, we will describe our initial product backlog and its rationale. The product backlog is in the form of a list of user stories describing the overall requirements for the system. Note that because of the product’s nature, this list would be subject to extensive changes in later sprints; It is merely a starting point.

3.1 Choice of Domain

Our project's problem is a rather general one, not intrinsically tied to any specific domain. However, its existence depends on an actual area of application. For this reason, it has been necessary for us to discover a domain for which to implement our prototype. Our criteria for selecting such a domain were as follows:

- It should be simple enough for test subjects to understand and feel familiar with.
- It should be complicated enough to demonstrate the problem.
- The console has to be useful in the domain, in such a way that it eases the workflow or opens new possibilities
- Object oriented design should be applicable
- There should be potential for the existence of power users capable of using the console

Various domains were considered, including but not limited to banking, warehouses, project management, travel agencies, education, social networking, music management and health care. Ultimately, based on the criteria listed, we decided in consensus to implement our solution for a library system.

3.2 Use cases

As discussed earlier, our domain is a library system. This leaves us with objects such as books, authors, employees and customers. Possible use cases would include:

- Register a new customer
- Register a new employee
- Order a new book
- Add a new book to the system
- List books in the system
- Search for books
- Place a reservation on a book
- Record a borrowing of a book
- Record a return of a book
- Extend a borrowing of a book
- Register the state of a returned book
- Remove a book from the library
- Request that a new book is ordered
- Export inventory
- Find the location of a book
- Generate reports

Figure 3.1: Use Case Diagram - Customer

Figure 3.2: Use Case Diagram - Employee

3.3 User stories

User stories are sentences describing requirements of users and justification of those requirements. They are written in the perspective of the end user. Our customer was unable to supply us with an explicit set of such user stories, so we composed a list of candidate user stories from our common understanding of the system gained through earlier meetings with the customer. The user stories are presented from a few different points of view: The administrator section describes detailed technical requirements. The general section contains stories applicable on general problems. The domain specific section contains user stories that are specific to the library domain. [1]

We proceeded to collectively estimate the difficulty of implementing each user story, as presented in the list below. The list of user stories was reviewed by the customer and approved as an initial product backlog. We would use this list for prioritizing which functionality to implement during the planning meeting for the first sprint.

Administrator point of view

A1 As an administrator, I want to be able to store objects in a persistent database, so that I can migrate data easily. Difficulty level: 5.

A2 As an administrator, I want to reflect the changes in persistent storage back to user sessions within one second, so the users always see the latest updated data. Difficulty level: 6.

A3 As an administrator, I want to be able to work directly with object attributes rather than any form of raw data. Difficulty level: 4.

User point of view - General

G1 As a user, I want my saved actions to be replicated on to the server within one second, so that my actions will not be lost and the client and server is consistent. Difficulty level: 3.5.

G2 As a user, I want my changes to be propagated to other users of the system real- time. Difficulty level: 2.

G3 As a user, I want to be able to see the console and graphical interface at the same time, so that I can use them both side by side simultaneously. Difficulty level: 8.

G4 As a user, I want the changes in console reflect in graphical user interface and likewise, so that I can have overview of the changes I made and I can understand easily, how the system works. In addition this will ensure consistency between the console and graphical interface. Difficulty level: 4.

G5 As a user, I want access to a tutorial, so that I can learn to work with the system easily. Difficulty level: 3.

G6 As a user, I want to be able to display the currently available commands in the console, so I can easily see what I can do with the objects at hand. Difficulty level: 3.

[1] http://scrummethodology.com/scrum-user-stories/

G7 As a user, I want to be able to easily repeat and edit last command, so that I can use it on another object. Difficulty level: 2.

G8 As a user, I want to be able to use batch commands, so that I can work with more than one object at the same time. Difficulty level: 3.

User point of view - Domain specific

D1 As a user, I want to be able to add a new book to the system using both the console and graphical interface. Difficulty level: 1 (reference).

D2 As a user, I want to be able to rmete a book in the system using both the console and graphical interface. Difficulty level: 0.5.

D3 As a user, I want to be able to edit information on a specific book and save these changes using both the console and graphical interface. Difficulty level: 2.

D4 As a user, I want to be able to search for a specific book in the system, so that I can watch the information on it using both the console and graphical interface. Difficulty level: 0.75.

D5 As a user, I want to be able to list all the books currently in the system, so that I easily can get an overview of all the books currently in the library. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D6 As a user, I want to be able to register a new customer in the system, so that customers can be saved in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.

D7 As a user, I want to be able to register when a customer borrows a book, so that the information is stored in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.75.

D8 As a user, I want to be able to edit the information on a specific borrowing of a book, so that any changes can be recorded. This should be possible in both the console and the graphical interface. Difficulty level: 2.5.

D9 As a user, I want to be able to list all the books currently borrowed, so that I can get information on each of them. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D10 As a user, I want to be able to reserve a book for a customer, so that customers can request and reserve certain books. This should be possible in both the console and the graphical interface. Difficulty level: 3.

D11 As a user, I want to be able to list all the reservations currently in the system. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D12 As a user, I want to be able to view reservations on specific books, so that I can see if a specific book is available for borrowing at the moment. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D13 As a user, I want to be able to order new books for the library using both the console and the graphical interface. Difficulty level: 1.5.

3.4 Summary

The envisioned system is a web application with an embedded console, see. The user will be able to choose if they want to use the console or the more traditional web user interface.

In the system, users will be able to manipulate a list of books by adding, removing and editing books, and perform filtering and searching. In addition, the system will have a list of customers, who are able to place reservations on books. Books can also be borrowed by customers.

Chapter 4

Design

Contents

How the product was designed, with discussion of design alternatives (referring to Appendix B for details).

I “Design”-kapittelet diskuter de viktigste funksjonene i ditt design og hvordan det har utviklet seg, fremhev noen nye/ originale funksjoner.

Men ikke ha med design dokumentasjon her!

In this chapter, we will describe our initial product backlog and its rationale. The product backlog is in the form of a list of user stories describing the overall requirements for the system. Note that because of the product’s nature, this list would be subject to extensive changes in later sprints; It is merely a starting point.

4.1 Choice of Domain

Our project's problem is a rather general one, not intrinsically tied to any specific domain. However, its existence depends on an actual area of application. For this reason, it has been necessary for us to discover a domain for which to implement our prototype. Our criteria for selecting such a domain were as follows:

- It should be simple enough for test subjects to understand and feel familiar with.
- It should be complicated enough to demonstrate the problem.
- The console has to be useful in the domain, in such a way that it eases the workflow or opens new possibilities
- Object oriented design should be applicable
- There should be potential for the existence of power users capable of using the console

Various domains were considered, including but not limited to banking, warehouses, project management, travel agencies, education, social networking, music management and health care. Ultimately, based on the criteria listed, we decided in consensus to implement our solution for a library system.

4.2 Use cases

As discussed earlier, our domain is a library system. This leaves us with objects such as books, authors, employees and customers. Possible use cases would include:

- Register a new customer
- Register a new employee
- Order a new book
- Add a new book to the system
- List books in the system
- Search for books
- Place a reservation on a book
- Record a borrowing of a book
- Record a return of a book
- Extend a borrowing of a book
- Register the state of a returned book
- Remove a book from the library
- Request that a new book is ordered
- Export inventory
- Find the location of a book
- Generate reports

Figure 4.1: Use Case Diagram - Customer

Figure 4.2: Use Case Diagram - Employee

4.3 User stories

User stories are sentences describing requirements of users and justification of those requirements. They are written in the perspective of the end user. Our customer was unable to supply us with an explicit set of such user stories, so we composed a list of candidate user stories from our common understanding of the system gained through earlier meetings with the customer. The user stories are presented from a few different points of view: The administrator section describes detailed technical requirements. The general section contains stories applicable on general problems. The domain specific section contains user stories that are specific to the library domain. [1]

We proceeded to collectively estimate the difficulty of implementing each user story, as presented in the list below. The list of user stories was reviewed by the customer and approved as an initial product backlog. We would use this list for prioritizing which functionality to implement during the planning meeting for the first sprint.

Administrator point of view

A1 As an administrator, I want to be able to store objects in a persistent database, so that I can migrate data easily. Difficulty level: 5.

A2 As an administrator, I want to reflect the changes in persistent storage back to user sessions within one second, so the users always see the latest updated data. Difficulty level: 6.

A3 As an administrator, I want to be able to work directly with object attributes rather than any form of raw data. Difficulty level: 4.

User point of view - General

G1 As a user, I want my saved actions to be replicated on to the server within one second, so that my actions will not be lost and the client and server is consistent. Difficulty level: 3.5.

G2 As a user, I want my changes to be propagated to other users of the system real- time. Difficulty level: 2.

G3 As a user, I want to be able to see the console and graphical interface at the same time, so that I can use them both side by side simultaneously. Difficulty level: 8.

G4 As a user, I want the changes in console reflect in graphical user interface and likewise, so that I can have overview of the changes I made and I can understand easily, how the system works. In addition this will ensure consistency between the console and graphical interface. Difficulty level: 4.

G5 As a user, I want access to a tutorial, so that I can learn to work with the system easily. Difficulty level: 3.

G6 As a user, I want to be able to display the currently available commands in the console, so I can easily see what I can do with the objects at hand. Difficulty level: 3.

[1] http://scrummethodology.com/scrum-user-stories/

G7 As a user, I want to be able to easily repeat and edit last command, so that I can use it on another object. Difficulty level: 2.

G8 As a user, I want to be able to use batch commands, so that I can work with more than one object at the same time. Difficulty level: 3.

User point of view - Domain specific

D1 As a user, I want to be able to add a new book to the system using both the console and graphical interface. Difficulty level: 1 (reference).

D2 As a user, I want to be able to rmete a book in the system using both the console and graphical interface. Difficulty level: 0.5.

D3 As a user, I want to be able to edit information on a specific book and save these changes using both the console and graphical interface. Difficulty level: 2.

D4 As a user, I want to be able to search for a specific book in the system, so that I can watch the information on it using both the console and graphical interface. Difficulty level: 0.75.

D5 As a user, I want to be able to list all the books currently in the system, so that I easily can get an overview of all the books currently in the library. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D6 As a user, I want to be able to register a new customer in the system, so that customers can be saved in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.

D7 As a user, I want to be able to register when a customer borrows a book, so that the information is stored in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.75.

D8 As a user, I want to be able to edit the information on a specific borrowing of a book, so that any changes can be recorded. This should be possible in both the console and the graphical interface. Difficulty level: 2.5.

D9 As a user, I want to be able to list all the books currently borrowed, so that I can get information on each of them. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D10 As a user, I want to be able to reserve a book for a customer, so that customers can request and reserve certain books. This should be possible in both the console and the graphical interface. Difficulty level: 3.

D11 As a user, I want to be able to list all the reservations currently in the system. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D12 As a user, I want to be able to view reservations on specific books, so that I can see if a specific book is available for borrowing at the moment. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D13 As a user, I want to be able to order new books for the library using both the console and the graphical interface. Difficulty level: 1.5.

4.4 Summary

The envisioned system is a web application with an embedded console, see Figure . The user will be able to choose if they want to use the console or the more traditional web user interface.

In the system, users will be able to manipulate a list of books by adding, removing and editing books, and perform filtering and searching. In addition, the system will have a list of customers, who are able to place reservations on books. Books can also be borrowed by customers.

Chapter 5

Implementation

Contents

I implementasjonskapittelet diskuter de viktigste algoritmene og datastrukturene, og hvordan de utviklet seg, fremhev noen nye/originale funksjoner. Også diskuter hvordan du har tenkt å utføre testen din (validering og evaluering).

In this chapter, we will describe our initial product backlog and its rationale. The product backlog is in the form of a list of user stories describing the overall requirements for the system. Note that because of the product's nature, this list would be subject to extensive changes in later sprints; It is merely a starting point.

5.1 Choice of Domain

Our project's problem is a rather general one, not intrinsically tied to any specific domain. However, its existence depends on an actual area of application. For this reason, it has been necessary for us to discover a domain for which to implement our prototype. Our criteria for selecting such a domain were as follows:

- It should be simple enough for test subjects to understand and feel familiar with.
- It should be complicated enough to demonstrate the problem.
- The console has to be useful in the domain, in such a way that it eases the workflow or opens new possibilities
- Object oriented design should be applicable
- There should be potential for the existence of power users capable of using the console

Various domains were considered, including but not limited to banking, warehouses, project management, travel agencies, education, social networking, music management and health care. Ultimately, based on the criteria listed, we decided in consensus to implement our solution for a library system.

5.2 Use cases

As discussed earlier, our domain is a library system. This leaves us with objects such as books, authors, employees and customers. Possible use cases would include:

- Register a new customer
- Register a new employee
- Order a new book
- Add a new book to the system
- List books in the system
- Search for books
- Place a reservation on a book
- Record a borrowing of a book
- Record a return of a book
- Extend a borrowing of a book
- Register the state of a returned book
- Remove a book from the library
- Request that a new book is ordered
- Export inventory
- Find the location of a book
- Generate reports

Figure 5.1: Use Case Diagram - Customer

Figure 5.2: Use Case Diagram - Employee

5.3 User stories

User stories are sentences describing requirements of users and justification of those requirements. They are written in the perspective of the end user. Our customer was unable to supply us with an explicit set of such user stories, so we composed a list of candidate user stories from our common understanding of the system gained through earlier meetings with the customer. The user stories are presented from a few different points of view: The administrator section describes detailed technical requirements. The general section contains stories applicable on general problems. The domain specific section contains user stories that are specific to the library domain. [1]

We proceeded to collectively estimate the difficulty of implementing each user story, as presented in the list below. The list of user stories was reviewed by the customer and approved as an initial product backlog. We would use this list for prioritizing which functionality to implement during the planning meeting for the first sprint.

Administrator point of view

A1 As an administrator, I want to be able to store objects in a persistent database, so that I can migrate data easily. Difficulty level: 5.

A2 As an administrator, I want to reflect the changes in persistent storage back to user sessions within one second, so the users always see the latest updated data. Difficulty level: 6.

A3 As an administrator, I want to be able to work directly with object attributes rather than any form of raw data. Difficulty level: 4.

User point of view - General

G1 As a user, I want my saved actions to be replicated on to the server within one second, so that my actions will not be lost and the client and server is consistent. Difficulty level: 3.5.

G2 As a user, I want my changes to be propagated to other users of the system real- time. Difficulty level: 2.

G3 As a user, I want to be able to see the console and graphical interface at the same time, so that I can use them both side by side simultaneously. Difficulty level: 8.

G4 As a user, I want the changes in console reflect in graphical user interface and likewise, so that I can have overview of the changes I made and I can understand easily, how the system works. In addition this will ensure consistency between the console and graphical interface. Difficulty level: 4.

G5 As a user, I want access to a tutorial, so that I can learn to work with the system easily. Difficulty level: 3.

G6 As a user, I want to be able to display the currently available commands in the console, so I can easily see what I can do with the objects at hand. Difficulty level: 3.

[1] http://scrummethodology.com/scrum-user-stories/

G7 As a user, I want to be able to easily repeat and edit last command, so that I can use it on another object. Difficulty level: 2.

G8 As a user, I want to be able to use batch commands, so that I can work with more than one object at the same time. Difficulty level: 3.

User point of view - Domain specific

D1 As a user, I want to be able to add a new book to the system using both the console and graphical interface. Difficulty level: 1 (reference).

D2 As a user, I want to be able to rmete a book in the system using both the console and graphical interface. Difficulty level: 0.5.

D3 As a user, I want to be able to edit information on a specific book and save these changes using both the console and graphical interface. Difficulty level: 2.

D4 As a user, I want to be able to search for a specific book in the system, so that I can watch the information on it using both the console and graphical interface. Difficulty level: 0.75.

D5 As a user, I want to be able to list all the books currently in the system, so that I easily can get an overview of all the books currently in the library. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D6 As a user, I want to be able to register a new customer in the system, so that customers can be saved in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.

D7 As a user, I want to be able to register when a customer borrows a book, so that the information is stored in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.75.

D8 As a user, I want to be able to edit the information on a specific borrowing of a book, so that any changes can be recorded. This should be possible in both the console and the graphical interface. Difficulty level: 2.5.

D9 As a user, I want to be able to list all the books currently borrowed, so that I can get information on each of them. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D10 As a user, I want to be able to reserve a book for a customer, so that customers can request and reserve certain books. This should be possible in both the console and the graphical interface. Difficulty level: 3.

D11 As a user, I want to be able to list all the reservations currently in the system. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D12 As a user, I want to be able to view reservations on specific books, so that I can see if a specific book is available for borrowing at the moment. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D13 As a user, I want to be able to order new books for the library using both the console and the graphical interface. Difficulty level: 1.5.

5.4 Summary

The envisioned system is a web application with an embedded console, see Figure . The user will be able to choose if they want to use the console or the more traditional web user interface.

In the system, users will be able to manipulate a list of books by adding, removing and editing books, and perform filtering and searching. In addition, the system will have a list of customers, who are able to place reservations on books. Books can also be borrowed by customers.

Chapter 6

Evaluation

Contents

This is the final phase of this project, the evaluation. Here it will be discussed why and how the outcome ended up being what it is today. This includes how the team worked together, why it gave the result it did, the cooperation with the customer, and how it was working with an overseeing force. Furthermore, we will discuss the issues met during the project, and how the process we used worked for us.

6.1 Team Dynamics

This section will shed light on how the group functioned, the cooperation and how this affected the work.

Work distribution

We feel like we managed to distribute the work evenly throughout the project and that every group member showed competence in their own fields. The actual work distribution was very dynamic: We agreed on what needed to be done, and then group members had to take individual responsibility to perform work on the tasks they were comfortable with. For our particular group composition we feel that this worked really well and that we were able to rmiver great results. However, we realize that with less enthusiastic team members, this morm may not have worked as well.

Communication

In our team, communication was done over skype and through frequent physical meetings. All team members were available online nearly all the time, making coordination a relatively simple task, since other members could be consulted at any time. Furthermore, our small group size allowed us to get well known with each other, leading to high awareness of the other team members' strengths.

An important point of success in our communication has been conflict resolution. All team members did a good job of bringing up potential points of disagreement so that these could be solved in consensus, or so that temporary solutions could be agreed upon until the matter became more clear.

Effort and estimation

Our actual amount of logged work fell a little short of the expected amount of hours. However, we were able to successfully execute the tasks that we planned at each step and avoided overworking ourselves. We do not think that the shortage of hours in itself had any significant detrimental effect to the project.

The shortage was largely because of our start phase being slow, due to a sense of confusion about how to get started on the project. We were not given any clear requirements nor existing code or examples to work on. We have learned from this and we will be more confident about taking initiative to push our projects forward in the future.

Another point is that our estimates for the implementation time was steadily a bit higher than the actual implementation time. In reality we think this was a healthy trait that could have helped us compensate for additional implementation problems, should they have appeared.

In general, we think our estimates were within reasonable bounds. This is a difficult task that is not likely to be perfect in any way, but we have definitely gained experience in producing more reasonable estimates.

6.2 Customer and project task

Our customer was very happy with the results. He made it clear that we exceeded his expectations, especially with the fact that we implemented more than he asked for. We experienced some communication problems from time to time, including meeting invitations getting lost in the customer's mobile device, but we addressed these issues and were able to keep ourselves and the customer updated on the progress.

From our point of view the customer was available, gave us great feedback and guidance throughout the project. One thing to note is that the mastermind behind the project, Stig Lau, was only available on one occasion. Ideally we would have had more direct contact with him, as we believe this would have served to steer us in the right direction even more quickly. However, our main customer contact, Peder Kongelf, was involved and enthusiastic about the project, made large resources within the company available to us, showed us that the project was important to them. We would not have been able to rmiver such a great end result if not for the great effort and constant feedback from the customer.

As stated in the TDT4290 Compendium(2012), "The target is a simple proof-of-concept in order to research any changes to the API, the potential of the method as well as evaluate the usability of scripting languages/DSL and API."

Initially we were considering basing our implementation on an existing system in order to be better able to assess the actual changes required to implement a console in a system. However, during our meeting with Stig Lau, it was made clear that it was not necessary to base our implementation on an existing web-application.

In the beginning we also wished to involve more users in testing our system's usability to better evaluate this. The customer originally wanted us to start working on this after the end of Sprint 2, but he then requested a change to a more technology-oriented focus as discussed earlier and in the conclusion. Discovering the potential of the method then became the main priority of the project. We believe we have indeed discovered and documented unique potential, and that this prioritization was in line with the customer's request. Yet, given more time it would have been interesting to further investigate the other two points(usability and API changes in existing systems).

The project task was an exciting one, giving us the opportunity to work with something new. The technologies we ended up using were cutting edge and we had fun discovering their true potential. The fact that the task was interesting increased our motivation and efforts. The experiences and knowledge we got from this project is something we will carry with us for years to come.

6.3 Advisor

Our advisor has been a steady aid, with meetings on nearly every week throughout the project. He has been available when we needed him and provided valuable feedback on each step of the way, especially on the writing of the project report.

We could have used this resource to a greater extent, especially with regard to design phase documents.

6.4 Development Process

This section will explain the usage of the chosen development methodology, and the good and bad parts with using this methodology.

Using scrum

The scrum development methodology was chosen early even though the team members were not too familiar with this sort of development process. But during the planning phase it became clear that it was a natural way to go. This was because agile development methodologies have proven to be efficient in prototype projects, where changes are highly likely to happen. It helps reduce the impact of change, which we got to experience first hand for ourselves.

Instead of the traditional daily physical stand-up scrum meetings we chose to keep all members updated through Skype. There we shared experience, troubles and progress on a daily basis, but without the overhead of physically having a meeting. We held a demo presentation for both the customer and the advisor at the end of every sprint, this helped us keeping them both updated on our progress, and opened for feedback. Based on this feedback we could make the needed changes to the system, so our next sprint would improve the system in the right direction. In the end we were able to rmiver a product that we and the customer were very happy with, which in hindsight can make us say that we were right to choose Scrum. It allowed us to adapt to changing requirements, find misdirections, and ultimately to fulfill the expectations from the customer. Without the adaptation we would have solved a problem, but not the problem the customer wanted us to solve.

Good

As mentioned above, Scrum allowed us be flexible and made it easy to change the direction the system was taking. This would have been troublesome without an agile development methodology. Traditional methods such as the Waterfall morm are rigid and designed to include only one design phase, one implementation phase, etc. This would have stopped us from being able to handle the redirection we did between sprint two and three towards a more dynamic system. Scrum let us to scale down in each sprint, focusing on smaller tasks. It also made the involvement of all the members in each aspect of the project easy, since Scrum opened a forum where we shared with each other.

Our execution of the Scrum process was far from perfect, as was to be expected since none of us had any prior experience in using it. But we learned as we went, and feel the experiences we have accumulated in this project will help us in the years to come. Scrum is the development process of choice in most projects concerning IT and we are bound to face this development morm in the future. This project provided us with first hand experience in the importance of constantly following up the customer to be able to rmiver a product that meets their expectations.

Bad

The Scrum process imposes a lot of rules, activities and meetings, which was time consuming. The overhead produced by each meeting, each demo presentation, etc, could in some cases have been better spent towards other activities instead. Without all this overhead we would have gotten more time to focus on the implementation, and maybe include more features in the product.

6.5 Testing

We mainly performed three types of testing in this project, unit testing of the code, test cases that were made for each user story and acceptance testing on each user story with the customer. The test cases were performed towards the end of each sprint, and in multiple cases helped us discover minor bugs or shortcomings of the system. We feel like this gave us sufficient test coverage of the different functionality and parts of the system. According to the test plan we were also supposed to do user and system testing, however this was not done. The reason for this is explained below. All in all we are satisfied with our testing process and the amount of testing we were able to do.

We originally planned to include user testing in this project. In the beginning the project was user-centered focusing on the user experience and usability of the system. It was important for us to involve users in the development process. However as the project developed, it became increasingly technology-centered. The priority shifted to discovering the true potential of the technologies we had chosen. As a result of this change in priorities and the limited time available, user testing was not performed as a part of this project. It is however something we would have liked to do if we were given more time, to test different users response to our system and identify improvements that can be made.

Our system turned out to be something else than what we expected from the beginning. The product in its current state is more a platform for developers than a finished product to be set out in production. Because of this it was difficult to identify what to look for in a complete system test of the final product, we did not posses an extensive list of requirements we could base it on. The different components of the system had already been extensively tested at the end of each sprint, both through test cases for each user story and unit tests of the code. Also it would have been difficult to test for specific functionality as the product is able to do pretty much anything. Although it is a library system it does not make sense to test it as a one, as it was never the intended goal for this project to produce a functioning library system. Taking all of this into account we decided not to perform a complete system test on the final product.

6.6 Issues

Group size

The biggest issue the team encountered was that the team size ended at four. The intended group size for the project was five to seven, which should have produced an extra 325 - 975 work hours, which is a considerable amount of hours. We managed to come out on top of this situation because of a set of responses and effects of having a small group.

Ease of communication

Since we were of such a small size the communication was tight and keeping everyone updated was easy to achieve. This made production efficiency high since there was not done any double work.

The team members

Taking responsibility came more naturally with such a small group as ours. The members stepped up their game and rmivered when it was needed. The general goal of the team was to rmiver a good result. This together with a great chemistry between the members made producing a good result with a small group size achievable.

Changing requirements

Between the second and the third sprint the requirements from the customer changed somewhat. This opened up for changes to the existing system, and made us reconsider the technologies we had used. This led to a system overhaul, and some bigger changes were made, like changing the database technology we currently used, into another. We managed this issue through some precautions we had made.

Risk handling

Since we were dealing with a prototype project, we always had in mind that requirements could change, and therefore had ease of modifiability in the back of our mind, while developing the system.

6.7 Summary

The course has all in all proved to be a positive and valuable experience for us all. For most of us, the experience of working on such a big project in a team was quite new. We experienced how important it was to plan ahead, to distribute the workload, and to collaborate to achieve a common goal. This was also our first experience in working with an external customer, giving us valuable experiences in this type of project. These experiences are sure to prove useful for in the years to come when we participate in similar projects.

We, as a group, feel that we have reached our goal, and rmivered a great product that the customer was very satisfied with. We made our customer happy and exceeded his expectations, and looking back this achievement is something we can be proud of.

Chapter 7

Conclusion

Contents

I evalueringskapittelet beskriv hvordan du vurderer arbeidet ditt. Oppsummer evalueringsresultatene, og bruk dem til å vurdere ditt eget arbeid kritisk. Vær ærlig om eventuelle mangler. Hva betyr resultatene? (Signifikans) I konklusjonen din, beskriv status for ditt arbeid. Oppsummer hva du har oppnådd, sammenlignet med hva du opprinnelig ønsket å oppnå. Relater arbeidet til tidligere relevant arbeid. Foreslår videre arbeid som du tror vil være verdt.

This chapter will describe the final version of our product and what we have found during this project. Also we will discuss the further development of the system and what improvement can be made.

7.1 Final Product

The system we have created is a basic library system, and it allows users to store information on books and authors. It is a web application, meaning it is accessed through a web- browser. It separates itself from regular web- applications in that it incorporates a console. The system consists of two main components, a client part and a server with an associated database. The client part deals with user input while the server provides the clients the ability to persistently store data between user sessions.

Client

A screenshot of the client is shown in Figure 7.1. The client part of the system is implemented using web technologies like JavaScript, jQuery, HTML and CSS. The interface is split in two separate views, one containing the console, and one containing a simple GUI for the application. The GUI part consists of two views, showing either a list of objects, or specifics on one single object. This part of the application offers limited functionality, which is confined to navigation of the application and editing of existing objects.

Figure 7.1: Client Screenshot

The console on the other hand contains a rich set of functionality. The main purpose of the console is to expose the underlying objects of the system, and it allows for the user to modify the properties of these objects. It comes with a list of predefined commands that represents commonly used actions such as add, rmete and navigation between the different databases. The commands offers a layer of abstraction to the users, and makes it easier for inexperienced users to utilize the console. The commands are functions written in JavaScript, and when recognized by the console/parser the console makes a call to these functions. The console is also capable of running any JavaScript code, meaning anything that is not recognized as a command is executed as regular JavaScript in the browser. This gives the user the ability to define their own functions, and extend the functionality of the console.

Server

The database of the system is implemented using CouchDB, which functions as a standalone server system. The CouchDB consists of a set of databases, each holding a group of objects with their associated attributes. Our system contains two databases, one for book objects and one for author objects. The communication between client and server is done through a RESTful api that CouchDB creates automatically. This api contains a rich set of functionality, and allows for actions such as add, rmete, update and filter.

7.2 Chosen Technologies

We spent a lot of time on the pre- study and on research in this project, and we feel it was well worth the effort. Also our customer helped us a lot, and offered great suggestions and guidance throughout the project. The technologies we ended up using were not only new and exciting, but were also a great fit together. JavaScript, jQuery, JSON, HTML and CSS are robust and mature web technologies in use all over the world today. Coupled with CouchDB, which was designed from the ground with web-applications in mind, they make a powerful collection of technologies that work well together.

Probably the most important decision on technology we were left with after the pre- study was whether to go for a traditional database management system like MySQL, or choose a more unconventional NoSQL system. The customer did not specify any requirements concerning the performance or stability of the server in this project, and indicated that the specifics on the server implementation was not that important to them. As a result we wanted it to be as simple as possible. After a detailed look at the needs for this project we ended up choosing a NoSQL database in the form of CouchDB. The reason for this was that CouchDB provided us with many advantages that a traditional relational database could not. Our system is heavily centered around JavaScript objects and their attributes. CouchDB, and other document oriented NoSQL systems, stores data in the form of objects directly in JSON format, which is a simple textual notation for JavaScript objects. This implied a minimal amount of work getting the client and server to communicate. If we had opted to use a traditional SQL system for the database we would have needed more extensive server side technologies, like PHP or ASP.NET to handle the database and to do database-object parsing. All of this was avoided by choosing CouchDB, which is able to work without any extra server software, and automatically creates a RESTful api to interact with the database.

Whereas trying to add undefined attributes to an existing record in a traditional relational database management system would result in an error, CouchDB automatically adds these attributes to the stored object. In essence, it allows us to add any new attribute to the existing objects, as long as it's represented in JSON format. This obviously gives you great flexibility. To do something similar in a regular SQL system you would have to change the schema of the database table every time a new attribute was discovered, or be able to morm every aspect of the objects from the beginning. For non-trivial real world domains, this usually presents a difficult challenge. Also the job of representing complex objects are considerably less than in a relational database. Complex object representation in SQL often requires a tedious process to identify the different tables you need and the relations between them. In CouchDB this job is reduced to just putting the entire JavaScript object directly into the database, which saves a lot of time for the developer. Maybe the best experience from this project was the freedom the technologies gave us. All the technologies were really simple out of the box, and this left us with the ability of choosing exactly which functionality to add. Also the technologies were flexible, meaning there were few limitations to what we could add. This flexibility means that is is easy both for developers and end users to add new functionality to the system. It should be mentioned that to be able to use this system to its maximum potential you need some sort of prior experience with JavaScript. But with this in hand you can accomplish virtually anything.

7.3 Findings

As discussed above, the system allows the user to dynamically define new object types and redefine existing ones, which is a rare, but powerful feature. In non-trivial domains it can be extremely difficult to create good a morm before the system is deployed, and the morm may need to change over time. Traditionally, database redesign would be a much more complicated and expensive process. [6]

As opposed to traditional systems where the developer has to explicitly add new functionality, our console solution features a complete feature set from the start, by means of the scripting language. The job of the developer is then to explicitly restrict unsafe functionality, offer simplifications to the syntax, and create alternative input methods(such as a rich GUI). A key advantage of implementing a console is the drastically reduced development cost of input commands as opposed to design and development of complex GUI elements. Exposing these to the user also poses a problem; Another important advantage of a hybrid console/GUI system is that the graphical part of the application can be simpler and therefore more likely to be user friendly, since the more complex functionality which is only used by power users can be accessed through the console.

A backside to the flexibility of the scripting console is that it poses a challenge when it comes to restricting unwanted functionality in the system. This was not something we focused on in this project, as our efforts were ultimately focused on finding the potential of the console. Our prototype assumes that the users know what they are doing and have no desire to cause problems in the system. However, we believe that this can be solved on a case-by-case basis if the solution is to be developed further. Another potential issue related to this would be security. In cases where this is a concern, ensuring that the user does not perform malicious actions would be of outmost importance. Security was not a point of focus for our project; However, in our solution, all commands are executed on the client side, and only JSON objects are sent to the server. We believe that in most domains, it is viable to implement a more complex server that tests the received objects for consistency to eliminate activity that is harmful to the system.

7.4 Paradigm shift

After reviewing the prototype from the second sprint(Wonsole1), our customer requested large changes to the requirements of the system, and effectively altered the paradigm of our project. Most notably, the requirement of dynamically defined object types made the existing implementation obsolete and forced a complete redesign with no hard coded domain-specific object types. The redesign also included changes to which supported technologies were used.

7.4.1 Technological Changes

We performed extensive changes to the system after the second sprint. The PubNub and Shell technologies turned out to be less useful than we originally thought, and did not add the kind of value we expected. As a result both of these technologies were not included in the final product. Also we were left with a decision on the database system. From the pre- study we found that MongoDB and CouchDB offered much of the same functionality. The decision to go with MongoDB originally was based on the fact that is was better documented, and seemed easier to implement on a server running Node.js. However it now became apparent that the Node.js wasn't needed. Both because the PubNub functionality was dropped and because CouchDB had the ability to work as a standalone system without any extra server software. Also the specific functionality offered by CouchDB turned out to be a better fit for the customer's new vision of the system than what MongoDB offered. Ultimately we decided to drop Node.js and switch to CouchDB. Looking back this was a reasonable decision.

7.4.2 Thought Patterns

The focus of the project shifted from being user-centered to being technology-centered: During the first two sprints, the customer was interested in improving the usability features of the console, as well as the interaction between the GUI and the console. During the last two sprints, the customer expressed a strong desire that we work on finding the potential of the method instead. Because of this, the end result is somewhat different from what we first imagined, and our initial desire to perform extensive user testing was not fulfilled. However, the end product holds a greater potential with its expanded feature set and flexibility.

7.5 Further Development

As a result of new requirements from the customer and new possibilities discovered during the development, we ended up creating something different than what we set out to do. Instead of focusing on a single application incorporating a console, what we actually have created is a tool for developers. Meaning that our product can be picked up by almost anyone with basic experience in programming and used to create an application in almost any domain. So in essence what we have rmivered is a platform for developers to use and not a product in the form of a library system. As a library system our final product is not finished, on the other hand it is nearly finished as a development tool. It functions as a baseline for developers, creating almost endless possibilities. Its flexibility ensures that it can be suited to any domain and to specific needs. In here lies the true potential of our system and this is where future efforts should be focused.

Although the goal of this project ultimately didn't include finishing the library system, it is still something we would like to do. Many of the challenges presented in the library domain holds for a multitude of other domains as well, and if solved it would make our solution more valuable. For the system to be able to serve as a production library system domain specific limitations on the actions available to the user has to be imposed. At the current state of the system it is possible for the user to input harmful code that breaks the system, for example by creating recursive functions that end up being called in an infinite loop. This kind of behavior should not be allowed. At the same time the flexibility the system presents the user is one of its best features, so it is not easy to know where to draw the line. The user should not be limited too much, as that would erase many of the advantages of our current system. How the functionality should be restricted is a topic open for a great deal of discussion, which is beyond the scope of this project. With great power comes great responsibility, meaning that for now we trust the users to not intentionally harm the system.

We feel there are improvements yet to be made to the console, such as increasing its usability. We wanted to include functionality that is available in most standard consoles, like for instance auto completion of commands; This was included in the prototype from Sprint 2, but due to the extensive changes required by the customer, time constraints and the prioritizations requested by the customer, this was not carried over to the latest prototype. This would be work for the future.

Although the performance and scalability of the server was not a concern in this project, it should be mentioned that CouchDB is designed to scale. Meaning that it offers tools to easily replicate the databases to multiple servers, ensuring that users can be fed data from multiple servers instead of one central one. This solution avoids one single point of failure and divides the workload. Also it offers great performance on queries to the database. CouchDB was made with this in mind, since for most web-applications the query operation is by far the most common one. While of course still a major challenge, we think that scaling an hypothetical finished version of our system, would be eased by the fact that we chose CouchDB. Also the performance of the database should persist even with a large amount of simultaneous users.

7.5.1 Other Domains

Our system is not confined to the library domain. As long as you can use domain specific knowledge to create objects and put these into the CouchDB, it can pretty much be used in any domain. The library domain in our current system is in essence only represented through the domain specific objects and their attributes stored in the database. Luckily inserting new objects is a trivial task in CouchDB, the only thing you need to do is to create a new database and start putting objects in it. Another aspect that increases the portability of the system, is that we allow users to create their own functions to fit their specific needs. It is possible to let the end users create the domain specific functionality of the system, as it requires minimal knowledge of JavaScript to be able to define your own functions. Also most of the commands available in the current console are not only confined to the library domain, but are applicable to other domains as well.

7.5.2 Existing Applications

One area which we did not find time to research is the possibility to implement our solution in an existing system. If possible, this would add great value to the solution. One aspect which we imagine would present a challenge, is that our solution relies on the fact that the existing system is already using CouchDB for the database. And as far as we can see, there aren't many systems using this technology as of today. But CouchDB and NoSQL databases in general is an emerging set of technology that we think will only grow in popularity for the years to come. It holds a lot of advantages over more traditional database management systems, especially for applications developed in the web- domain. We feel this will contribute to making our product even more valuable in the future.

7.6 Summary

The objective of this project was to demonstrate the need for scripting in a web- application, showing that it can add value to the user experience. We feel like we have accomplished this objective. We have shown what is possible to achieve with the technologies we had chosen, and what advantages they provided us with compared to more traditional technologies. We have demonstrated the flexibility of our system, which can be suited to almost any need and any domain without changing the core functionality. The system is in no way complete as a library system, but that wasn't really a priority for this project. The rmivery we made ended up being a platform for developers with almost endless possibilities. We feel we have proved that the concept of a console in a web- application is useful, and that was the tasks at hand.

As far as we can see, there exists no similar systems in the world today. We feel we have created something new, something of great value. We have utilized new emerging technologies, with a lot of advantages over more traditional solutions. There is however still some work left to be done before a system based on our work can be used in a production environment. Our system presents a new way of thinking, in that it explicitly restricts functionality instead of explicitly adding functionality. It is way of thinking we think has a lot of potential, yet it requires more research on some of its aspects before we can draw definitive conclusions on its implications. Our product is not finished, but what we have created can be picked up by anyone. With more development and research we think it can be something that holds commercial business value.

Appendix A

Requirements

Appendix B

Design

Appendix C

Test Cases

C.1 Sprint 1

Table C.1: Test Case TID01

Item	Description
Description	Storing objects in a database on the central server
Tester	Øystein Heimark
Preconditions	There needs to be a server running with a connection to a database available
Feature	Test the ability to store objects permanently on the server from the client
Execution steps	1. Open a new client 2. Call the appropriate method for storing a new object with a given set of attributes from the client. 3. List the content of the database and observe if the new object is indeed stored with its correct attributes.
Expected result	The object is stored in the database with the correct attributes

Table C.2: Test Case TID02

Item	Description
Description	Retrieving objects from the database on the central server
Tester	Øystein Heimark
Preconditions	There needs to be a server running with a connection to a database available
Feature	Test the clients ability to retrieve objects from the server
Execution steps	1. Open a new client. 2. Call the appropriate method for retrieving an object. 3. Observe the response from the server.
Expected result	The object is successfully retrieved from the server with the correct attributes

Table C.3: Test Case TID03

Item	Description
Description	Sending real- time messages from server to client
Tester	Øystein Heimark
Preconditions	There needs to be a server able to send messages up and running, and a client ready to receive
Feature	Test the ability to send real- time messages from server to client
Execution steps	1. Open a new client. 2. Send a message from the server with the associated method. 3. Observe the output on the client side.
Expected result	The message will be received by the client and displayed within one second from when the message is sent from the server.

Table C.4: Test Case TID04

Item	Description
Description	Alerting clients that there has been added a book to the central database on the server
Tester	Øystein Heimark
Preconditions	TID03 and either TID05 or TID06 must alredady have passed. The server must be running
Feature	The ability to alert multiple clients that a new book is added to the system real- time
Execution steps	1. Open the application with multiple clients. 2. Add a new book from one of the clients. 3. Observe the output on all the clients
Expected result	All the clients will be alerted within one second that a new book has been added, and the list of books in the client will be updated.

Table C.5: Test Case TID05

Item	Description
Description	Verifying that domain specific objects are available through the console
Tester	Øystein Heimark
Preconditions	A console must be available
Feature	The ability to work directly with domain specific objects and objects attributes
Execution steps	1. Open a console. 2. Create a book object. 3. Change the attribute of the newly created object by command.
Expected result	The user is able to retrieve objects and change their attributes via the console.

Table C.6: Test Case TID06

Item	Description
Description	Verifying that there is a console and graphical interface present on each page
Tester	Øystein Heimark
Preconditions	None
Feature	Simultaneous display of console and graphical interface
Execution steps	1. Open a new instance of the application with a web- client. 2. Observe if there is a graphical interface as well as a console present.
Expected result	Console and graphical interface is present on the same page.

Table C.7: Test Case TID07

Item	Description
Description	Adding a new book to the system with the graphical web- application
Tester	Øystein Heimark
Preconditions	The server with the REST api must be running. A graphical interface must be available.
Feature	The ability to add new books to the system from a client with the graphical web-application
Execution steps	1. Open the application with a web client 2. Add a new book from the web- application on the client. 3. List the books currently on the system and observe if the new book is added.
Expected result	The new book is added to the system and the list of books with the attributes stated in the creation of the book.

Table C.8: Test Case TID08

Item	Description
Description	Adding a new book to the system with the console. A console must be available
Tester	Øystein Heimark
Preconditions	The server with the REST api must be running
Feature	The ability to add new books to the system from a client with the console.
Execution steps	1. Open the application with a web client 2. Add a new book from the console on the client. 3. Observe the list of the books currently on the system and observe if the new book is in this list.
Expected result	The new book is added to the system and the list of books with the attributes stated in the creation of the book.

Table C.9: Test Case TID09

Item	Description
Description	Listing all the books currently in the system using the graphical web- application
Tester	Øystein Heimark
Preconditions	The server with the REST api must be running. There has to be books stored in the database. A graphical interface must be available
Feature	The ability to get an overview of the books currently in the system using the web-application
Execution steps	1. Obtain a list of all the books in the system directly from the central database/server 2. Use the graphical web- application to get a list of all the books in the system 3. Compare the result from step two to the one obtained in step 1, and verify that they contain the same books.
Expected result	The list of books presented in the graphical web- application is identical to the one stored on the central database/server.

Table C.10: Test Case TID10

Item	Description
Description	Listing all the books currently in the system using the console.
Tester	Øystein Heimark
Preconditions	The server with the REST api must be running. There has to be books stored in the database. A console must be available
Feature	The ability to get an overview of the books currently in the system using console.
Execution steps	1. Obtain a list of all the books in the system directly from the central database/server 2. Use the console to get a list of all the books in the system 3. Compare the result from step two to the one obtained in step 1, and verify that they contain the same books.
Expected result	The list of books presented in the console is identical to the one stored on the central database/server.

C.2 Sprint 2

Table C.11: Test Case TID11

Item	Description
Description	Storing objects without a schema in a database on the central server .
Tester	Øystein Heimark
Preconditions	There needs to be a server up and running with a database available
Feature	The ability to store objects with a different attribute set, in the same database.
Execution steps	1. Call the appropriate method for storing a new object with a given set of attributes 2. Call the same method again, but provide an object with a different set of attributes 3. Observe that both objects are stored in the database with the correct attributes.
Expected result	Both objects, with different attributes, are stored in the database.

Table C.12: Test Case TID12

Item	Description
Description	Printing out commands in the console while operating with the GUI.
Tester	Øystein Heimark
Preconditions	None
Feature	For every action made in the GUI the corresponding command in the console should be printed in the console.
Execution steps	1. Open a new client 2. Do a lot of different actions in the GUI. 3. Observe that the correct commands are printed in the console.
Expected result	The correct commands are printed in the console.

Table C.13: Test Case TID13

Item	Description
Description	Showing a popup menu in the console with the available methods and attributes for the object which is currently selected.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to list the methods and attributes of a given object in a popup menu.
Execution steps	1. Open a new client 2. Create a new object. 3. Select that object using the console. 4. Show the popup menu using the corresponding hotkey
Expected result	The correct methods and attributes of the selected object is shown in the popup menu.

Table C.14: Test Case TID14

Item	Description
Description	Selecting an element from the popup menu and insert the selected method or attribute in the console.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to autocomplete methods and attributes selected from the popup menu.
Execution steps	1. Open a new client 2. Select an object using the console. 3. Pick a method or attribute from the popup menu.
Expected result	The selected method or attribute from the popup menu is printed in the console.

Table C.15: Test Case TID15

Item	Description
Description	Highlighting an object in the GUI when it is selected from the console.
Tester	Øystein Heimark
Preconditions	None
Feature	Highlighting a selected object.
Execution steps	1. Open a new client 2. Select an object using the console. 3. Observe the response in the GUI.
Expected result	The selected object should be highlighted in the GUI.

Table C.16: Test Case TID16

Item	Description
Description	Highlighting a group of objects in the GUI when it is selected from the console.
Tester	Øystein Heimark
Preconditions	The system must be capable of selecting multiple objects
Feature	Highlighting groups of objects.
Execution steps	1. Open a new client 2. Select a group of objects. 3. Observe the response in the GUI.
Expected result	The selected objects should be highlighted in the GUI.

Table C.17: Test Case TID17

Item	Description
Description	Cycling through the current selection of objects using a hotkey.
Tester	Øystein Heimark
Preconditions	The system must be capable of selecting multiple objects
Feature	The ability to cycle through the current selection of objects in the console using a hotkey.
Execution steps	1. Open a new client 2. Select a group of objects. 3. Cycle through the objects using the hotkey in the console.
Expected result	The objects in the current selection are made available to the user one by one, in the correct order.

Table C.18: Test Case TID18

Item	Description
Description	Highlighting the selected object while cycling through a selection of objects.
Tester	Øystein Heimark
Preconditions	The system must be capable of selecting multiple objects
Feature	While cycling through a selection of objects in the console the current object will be highlighted in the GUI.
Execution steps	1. Open a new client 2. Select a group of objects. 3. Cycle through the objects using the hotkey in the console. 4. Observe the response in the GUI.
Expected result	The highlighted object in the GUI will be updated as you cycle through the selection.

Table C.19: Test Case TID19

Item	Description
Description	Update a separate section of the GUI when the user clicks on a record, and make the user able to edit the record in this section.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to edit the information on a record from a separate section of the GUI.
Execution steps	1. Open a new client 2. Click on a record. 3. Edit the info on the clicked record in the separate section. 4. Observe the response in the GUI.
Expected result	A separate section of the GUI is updated with the information on the clicked record. When this information is edited the record is updated.

C.3 Sprint 3

Table C.20: Test Case TID20

Item	Description
Description	Changing directory in the application in the console.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to easily change the working directory from the console.
Execution steps	1. Open a new client 2. Call the command for changing directory. 3. Observe the response in the GUI and console.
Expected result	The GUI is updated to represent the specified directory. The objects in this directory is available from the console.

Table C.21: Test Case TID21

Item	Description
Description	Storing objects as local variable.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to extract objects and store these in specified variables in the console.
Execution steps	1. Open a new client 2. Navigate to a directory with objects. 3. Extract an object with a DSL command, and assign it to a variable. 4.Change directory. 5.Print the content of the variable in the console
Expected result	The object which you assigned to the variable is printed.

Table C.22: Test Case TID22

Item	Description
Description	Allow for the use of functions on the objects.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to apply function on single or groups of objects.
Execution steps	1. Open a new client 2. Retrieve a group of objects. 3. Call DSL command for applying a function to a group of objects. 4.Call DSL command and apply a mathematical function on a numerical attribute of the objects.
Expected result	The function is applied correctly to all the objects, and the attributes of the involved objects are updated accordingly.

Table C.23: Test Case TID23

Item	Description
Description	Allow for editing and adding of attributes.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to edit existing attributes or add new ones.
Execution steps	1. Open a new client 2. Navigate to a specific object. 3. Change an attribute, using both the GUI and the console. 4.Add an attribute, using both the GUI and the console. 5.Extract an entire JSON object and add it as a new attribute for the object
Expected result	Existing attributes are correctly updated, and new attributes are correctly added. The JSON object is added as a attribute of the specified object.

Table C.24: Test Case TID24

Item	Description
Description	Update GUI according to changes.
Tester	Øystein Heimark
Preconditions	None
Feature	Whenever a user changes, rmetes or adds an attribute of one or several of the objects, the GUI updates.
Execution steps	1. Open a new client 2. Make changes to single and groups of objects
Expected result	The GUI updates whenever an object changes.

Table C.25: Test Case TID25

Item	Description
Description	Store changes to database.
Tester	Øystein Heimark
Preconditions	None
Feature	The ability to work locally and push changes to the server when the user desires.
Execution steps	1. Open a new client 2. Do a variety of actions in the application. 3. Issues DSL command to store the changes on the server
Expected result	The changes are successfully stored on the server, and is retrievable for other users as well.

C.4 Sprint 4

Table C.26: Test Case TID26

Item	Description
Description	Call specific functions on a group of objects.
Tester	Øystein Heimark
Preconditions	There needs to be objects present in the system
Feature	The ability to call a command that applies a specified function on a group of objects.
Execution steps	1. Open a new client 2. Make a group of books and apply a function that adds 15 percent of the price 3. Make another group of books and apply a function that subtracts 15 percent of the price
Expected result	The books in the first group have their price increased by 15 percent, while the books in the second group have their price decreased by 15 percent.

Table C.27: Test Case TID27

Item	Description
Description	Filtering objects
Tester	Øystein Heimark
Preconditions	There needs to be books present in the system
Feature	The ability to filter the objects on a specified criteria.
Execution steps	1. Open a new client 2. Call filter command to get all books with title “Peer Gynt”.
Expected result	The GUI is updated to to show only the books with title equal to “Peer Gynt”.

Table C.28: Test Case TID28

Item	Description
Description	Adding and rmetion of objects
Tester	Øystein Heimark
Preconditions	There needs to be books present in the system
Feature	The ability to add a new book to the system using, and rmete specific books already in it.
Execution steps	1. Open a new client 2. Call command for adding a new book. 3. Call command for rmeting the newly added book.
Expected result	The book is first added to the system and appears in the list of books. When the rmetion command is issued, it should be removed from the system

Table C.29: Test Case TID29

Item	Description
Description	Creating scripts
Tester	Øystein Heimark
Preconditions	The ability to create custom scripts and to execute them
Feature	The ability to add a new book to the system using, and rmete specific books already in it.
Execution steps	1. Open a new client 2. Create a script that prints the title of all the books with a price smaller than 100, and save it to a variable. 3. Call the variable.
Expected result	The script is stored in the specified variable and executed. The title of all the books with a price smaller than 100 is printed.

Appendix D

Implementation Documentation

D.1 Wonsole1 Objects

Library object specification

```
function removeSelected ()
remove all books that are selected in the visible list, will update the web UI and db.

function listBooks()
List all books into the console.

function removeBookByID(_id)
Remove a book with the given ID. Will update the web UI and db.

function selectAllToggle()
Toggle select all books currently in the visible list. Will update the web UI and db.
This function is coupled with the checkbox for selecting all books in the list.

function query(parameter1, value1, parameter2, value2 ...)
Queries the list of books for an array of books where the
specified book parameters match their respective values.
May be called with an arbitrary even number of arguments.
The values will be interpreted as regular expressions if they are strings.

function generateHTML()
Generate list elements for all books in the system,
at the "BOOKTABLE" element in the HTML document.

function retrieveObjects()
This function retrieves all objects from the server and updates the web UI and db.
Will lock the UI until objects have been received.
```

Book object specification

```
function Book(title, author, id)
Constructor for the Book object. Will add it to the list of Books in LIB.
Should be used with the new keyword.
If id is null, the object will be sent to the server, and the id will be returned.
```

```
This will block the UI and then update it.
Leaving out the id parameter entirely will be interpreted as the id being null.
id should be null when using this from the console or web UI,
but defined in the callback function for retrieving Books.

function saveUpdate()
Update the book on the server, blocking/unblocking and updating the UI in the process.
Should be called after altering the Book object’s variables.

function changeAuthor(newAuthor)
Change the author of the book. Will update the web UI and db.

function toJSON()
Generate a JSON object from this Book.

function changeTitle(newTitle)
Change the name of the book. Will update the web UI and db.

function toggleSelect()
Toggle whether this book is selected.
Will make sure the value of the Book’s checkbox is correct, if it exists.

Remove this book from the system. Will update database and UI.
function remove()

function generateHTML()
Generate HTML element(s) for this book. Will not manipulate the UI by itself.
Returns the DOM HTML element. Should be a table row. Row style will be overridden.
```

D.2 RESTful API Documentation

Base URL

The base URL for REST API is: http://netlight.dlouho.net:9004/api/

Get a list of all books

Description: Returns a list of all the books currently stored in the system

Resource URL: http://netlight.dlouho.net:9004/api/books
HTTP Methods: GET
Response format: json
Parameters: None

Request Example:
GET http://netlight.dlouho.net:9004/api/books

Response:

```
[
    {
        "_id": "506b6445b107d7567a000001",
        "author": "An author",
        "title": "Book1"
    },
    {
        "_id": "506c91a1b107d7567a000004",
        "author": "Another author",
        "title": "Book2"
    }
]
```

Example call in jQuery:

```
$.get(‘http://netlight.dlouho.net:9004/api/books’, function(response){
//Callback function
});
```

Add a book to the database

Description: Adds a book to the database with the supplied parameters. The created book object with a text identifier is returned as a repsonse.

Resource URL: http://netlight.dlouho.net:9004/api/books
HTTP Methods: POST
Response format: json
Parameters: None

Data:

- title(required): The title of the book that is to be added. Example values: "Title", "A Book".
- author(required):The author of the book that is to be added. Example values: "Author", "Another Author".

Request Example:
POST http://netlight.dlouho.net:9004/api/books
POST Data title="Title", author="Author"
Response:

```
[
    {
        "_id": "506b6445b107d7567a000001",
        "author": "Author",
        "title": "Title"
    }
]
```

Example call in jQuery:

```
$.ajax({
  type: ’POST’,
  url: ‘http://netlight.dlouho.net:9004/api/books’,
  data: { author:”Author”, title: ‘‘Title”},
  success: function(response){
   //Add book to local storage
  },
  dataType: ‘json’
});
```

Get a single book by id

Description: Returns a single book, specieifed by the id parameter

Resource URL: http://netlight.dlouho.net:9004/api/books/:id
HTTP Methods: GET
Response format: json

Parameters:

- id(required): This is a text identifier which is used to identify the book in the database. This is created by the database on insertion, and returned to the user. Example value: "506b6445b107d7567a000001"

Request Example:
GET http://netlight.dlouho.net:9004/api/books/506b6445b107d7567a000001

Response:

```
[
    {
        "_id": "506b6445b107d7567a000001",
        "author": "Author",
        "title": "Title"
     }
]
```

Example call in jQuery:

```
$.get('http://netlight.dlouho.net:9004/api/books/506b6445b107d7567a000001', function(response){
//Callback function
});
```

Update a single book by id

Description: Updates a book with the new values, specified by the supplied id parameter. Returns the updated book object.

Resource URL: http://netlight.dlouho.net:9004/api/books/:id
HTTP Methods: PUT
Response format: json
Data format: json
Parameters:

- id(required): This is a text identifier which is used to identify the book in the database. This is created by the database on insertion, and returned to the user. Example value: "506b6445b107d7567a000001"

Data:

- title(required): The title of the book that is to be added. Example values: "Title", "A Book".
- author(required):The author of the book that is to be added. Example values: "Author", "Another Author".

Request Example:
PUT http://netlight.dlouho.net:9004/api/books/506b6445b107d7567a000001
PUT Data: title="NewTitle", author="NewAuthor"
Response:

```
[
    {
        "_id": "506b6445b107d7567a000001",
        "author": "NewAuthor",
        "title": "NewTitle"
     }
]
```

Example call in jQuery:

```
$.ajax({
  type: ’PUT’,
  url: ‘http://netlight.dlouho.net:9004/api/books/506b6445b107d7567a000001’,
  data: { author:’’NewAuthor’’, title: ‘‘NewTitle’’},
  success: function(response){
   //Change book attributes in local storage
  },
  dataType: ‘json’
});
```

rmete a book by id

Description: rmetes a book, specified by the supplied id parameter.

Resource URL: http://netlight.dlouho.net:9004/api/books/:id
HTTP Methods: rmETE
Parameters:

- id(required): This is a text identifier which is used to identify the book in the database. This is created by the database on insertion, and returned to the user. Example value: "506b6445b107d7567a000001"

Request Example:
rmETE http://netlight.dlouho.net:9004/api/books/506b6445b107d7567a000001

Example call in jQuery:

```
$.ajax({
  type: ’rmETE’,
  url: ‘http://netlight.dlouho.net:9004/api/books/5069868335f41ce71a000001’,
  success: function(response){

  },
  dataType: ’json’
});
```

Appendix E

Product Backlogs

This section shows the evolution of the product backlog throughout the sprints.

E.1 Sprint 1

A1 As an administrator, I want to be able to store objects in a persistent database, so that I can migrate data easily.

A2 As an administrator, I want to reflect the changes in persistent storage back to user sessions within one second, so the users always see the latest updated data.

A3 As an administrator, I want to be able to work directly with object attributes rather than any form of raw data.

G1 As a user, I want my saved actions to be replicated on to the server within one second, so that my actions will not be lost and the client and server is consistent

G2 As a user, I want my changes to be propagated to other users of the system real- time

G3 As a user, I want to be able to see the console and graphical interface at the same time, so that I can use them both side by side simultaneously.

G4 As a user, I want the changes in console reflect in graphical user interface and likewise, so that I can have overview of the changes I made and I can understand easily, how the system works. In addition this will ensure consistency between the console and graphical interface.

G5 As a user, I want access to a tutorial, so that I can learn to work with the system easily.

G6 As a user, I want to be able to display the currently available commands in the console, so I can easily see what I can do with the objects at hand.

G7 As a user, I want to be able to easily repeat and edit last command, so that I can use it on another object.

G8 As a user, I want to be able to use batch commands, so that I can work with more than one object at the same time.

D1 As a user, I want to be able to add a new book to the system using both the console and graphical interface.

D2 As a user, I want to be able to rmete a book in the system using both the console and graphical interface.

D3 As a user, I want to be able to edit information on a specific book and save these changes using both the console and graphical interface.

D4 As a user, I want to be able to search for a specific book in the system, so that I can watch the information on it using both the console and graphical interface.

D5 As a user, I want to be able to list all the books currently in the system, so that I easily can get an overview of all the books currently in the library. This should be possible in both the console and the graphical interface.

D6 As a user, I want to be able to register a new customer in the system, so that customers can be saved in the system. This should be possible in both the console and the graphical interface.

D7 As a user, I want to be able to registrate when a customer borrows a book, so that the information is stored in the system. This should be possible in both the console and the graphical interface.

D8 As a user, I want to be able to edit the information on a specific borrowing of a book, so that any changes can be recorded. This should be possible in both the console and the graphical interface.

D9 As a user, I want to be able to list all the books currently borrowed, so that I can get information on each of them. This should be possible in both the console and the graphical interface.

D10 As a user, I want to be able to reserve a book for a customer, so that customers can request and reserve certain books. This should be possible in both the console and the graphical interface.

D11 As a user, I want to be able to list all the reservations currently in the system. This should be possible in both the console and the graphical interface.

D12 As a user, I want to be able to view reservations on specific books, so that I can see if a specific book is available for borrowing at the moment. This should be possible in both the console and the graphical interface.

D13 As a user, I want to be able to order new books for the library using both the console and the graphical interface.

E.2 Sprint 2

A4 As a developer, I want to have a schemaless database, so I don't have to change any code every time the objects change.

G1 As a user, I want my saved actions to be replicated on to the server within one second, so that my actions will not be lost and the client and server is consistent

G4 As a user, I want the changes in console reflect in graphical user interface and likewise, so that I can have overview of the changes I made and I can understand easily, how the system works. In addition this will ensure consistency between the console and graphical interface.

G4a - As a user, I want commands issued in console to update information in graphical UI.

G4b - As a user, I want the actions in graphical UI to print corresponding commands to console.

G5 As a user, I want access to a tutorial, so that I can learn to work with the system easily.

G6 As a user, I want to be able to display the currently available commands in the console, so I can easily see what I can do with the objects at hand.

G9 As a user, I want the system to be able to autocomplete the commands, and display a list of the available commands on a given object that can be chosen.

G10 As a user, I want the graphical user interface to indicate which object or group of object I'm working on.

G10a - A newly created object should also be highlighted in the user interface.

G11 As a user, I want the console to be able to cycle through objects in an array one at the time, and for the currently selected object to be displayed on the gui.

G12 As a user, I want the system to show me details of a record, after I click on a record.

G13 As a user, I want the system to highlight the changes my by other users, while I work with the system.

G14 As a user I want to be able to maintain an uninterrupted workflow in the web UI when changes are performed by other users.

G15 As a user I want to be able to sort the items or find an item, so that i can work with the data effectively

D3 As a user, I want to be able to edit information on a specific book and save these changes using both the console and graphical interface.

D4 As a user, I want to be able to search for a specific book in the system, so that I can watch the information on it using both the console and graphical interface.

D6 As a user, I want to be able to register a new customer in the system, so that customers can be saved in the system. This should be possible in both the console and the graphical interface.

D7 As a user, I want to be able to register when a customer borrows a book, so that the information is stored in the system. This should be possible in both the console and the graphical interface.

D8 As a user, I want to be able to edit the information on a specific borrowing of a book, so that any changes can be recorded. This should be possible in both the console and the graphical interface.

D9 As a user, I want to be able to list all the books currently borrowed, so that I can get information on each of them. This should be possible in both the console and the graphical interface.

D10 As a user, I want to be able to reserve a book for a customer, so that customers can request and reserve certain books. This should be possible in both the console and the graphical interface.

D11 As a user, I want to be able to list all the reservations currently in the system. This should be possible in both the console and the graphical interface.

D12 As a user, I want to be able to view reservations on specific books, so that I can see if a specific book is available for borrowing at the moment. This should be possible in both the console and the graphical interface.

D13 As a user, I want to be able to order new books for the library using both the console and the graphical interface.

E.3 Sprint 3

- G9 - As a user, I want to be able to navigate the application like a directory structure, using simple commands to change the current directory.
- G10 - As a user, I want to be able to store objects in variables in the console, and to be able to use these later on.
- G11 - As a user, I want the content of the GUI to represent the current directory, so that I can easily see which directory I am currently in.
- G12 - As a user, I want to be able to issue a command to see the properties of a specific object both from the console and in the GUI

- G13 - As a user, I want to be able to call a specific function on a group of objects.
- G14 - As a user, I want to be able to perform mathematical operations on numerical attributes of the single or groups of objects.
- G15 - As a user, I want to be able to change current attributes or dynamically add new ones to the objects.
- G16 - As a user, I want to be able to add entire JSON objects as attributes of other objects. For example I want to be able to extract an author object and add this object to multiple book objects.
- G17 - As a user, I want the changes I do on the objects from the console to be replicated to GUI, so that the GUI always presents updated information.
- G18 - As a user, I want to be able to work locally, so that I can save changes to the server when I want to.
- G19 - As a user, I want to be able to filter a list of objects from the console, so that I can view only the books I am interested in.

E.4 Sprint 4

- G13 - As a user, I want to be able to call a specific function on a group of objects.
- G19 - As a user, I want to be able to filter a list of objects from the console, so that I can view only the books I am interested in.
- G20 - As a user, I want to be able to add new objects to the system using a simple syntax.
- G21 - As a user, I want to be able to rmete objects from the system using a simple syntax.
- G22 - As a user, I want to be able to define and execute scripts.

Bibliography

[1] 10gen Inc. MongoDB Introduction. "http://www.mongodb.org/display/DOCS/Introduction", 2012. [Online; accessed 2012-10-12].

[2] Kent Beck. *Test-driven development : by example.* Addison-Wesley, Boston, 2003.

[3] Oracle Corporation. MySQL About. "http://www.mysql.com/about/", 2012. [Online; accessed 2012-10-12].

[4] The Apache Software Foundation. CouchDB about. "http://couchdb.apache.org/", 2012. [Online; accessed 2012-10-12].

[5] The Apache Software Foundation. CouchDB technical overview. "http://wiki.apache.org/couchdb/Technical%20Overview", 2012. [Online; accessed 2012-10-12].

[6] David Kroenke. *Database processing : fundamentals, design, and implementation.* Pearson Prentice Hall, Upper Saddle River, N.J, 2006.

[7] Neil Leavitt. Will nosql databases live up to their promise?

[8] NoSQL. List of NoSQL databases. "http://nosql-database.org", 2012. [Online; accessed 2012-10-10].

[9] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, May 2008.

[10] Raghu Ramakrishnan. *Database management systems.* McGraw-Hill, Boston, 2003.

[11] Ian Sommerville. *Software engineering.* Pearson, Boston, 2011.

[12] Cristof Strauch. Nosql databases.

[13] The Apache Software Foundation. Welcome to Apache Hadoop! "http://hadoop.apache.org/index.pdf", 2012. [Online; accessed 2012-11-10].

[14] Wikipedia. Big data. "http://en.wikipedia.org/wiki/Big_data", 2012. [Online; accessed 2012-11-10].