Wonsole

The new web console for power users

NTNU – Trondheim
Norwegian University of
Science and Technology

netlight

Customer Driven Project
November 20, 2012
Team: Ivo Dlouhy, Martin Havig, Øystein Heimark, Oddvar Hungnes

Abstract

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 - Norwegian University of Science and Technology. Netlight is the customer 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 and web-fronted design. This has taken away the efficiency of a power user, who used a terminal application on a daily basis, and had the system to their fingertips.

A hybrid web-fronted/console design would be a possible solution to this problem. Where the power user can get 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 research done will be documented and use 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 from this study let us put together a system which we test on different kinds of users, such as, librarians, our peers and others. Through this whole process we have a close work-relationship with our customer to ensure the wants and expectations are met, and that our conclusions and findings boosts future research in this field.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

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 General information about project

1.1.1 TDT4290 Customer Driven Project

The project is the making of the course TDT4290 Customer Driven Project. Customer driven project is a course held at NTNU, described in section 1.3.2. Customer driven project is held through one semester, and accounts for 15 credits. This is a mandatory subject for all 4th year students at IDI and aims to give all its students experience in a customer guided IT-project and the feel of managing a project in a group. In this course the students are divided into groups, and each of these groups receives a customer. The customer has put together a task for the group to handle, and to make sure that the product produced throughout the project corresponds to the wishes of the customer, the group and the customer should have a close relationship. Each group also receives an advisor. This advisor will support the group with inputs and issue solution suggestions. The delivery of the course is a report and a product, this will be presented to an examiner. The report is the most important part of the project, and will contain information such as: preliminary study, project management, architecture, conclusion and more. The course will provide realistic experience in both report writing and product development driven by a customer. This will help the students perform better when they are out in real life emplyment situations. [1]

1.2 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 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.

1.3 Stakeholders

The stakeholders in this project is any person or organization which has some interest or is affected by this system's development. They together constructs the different restrictions and goals of the project.

[1] `http://www.idi.ntnu.no/emner/tdt4290/`

1.3.1 The Team

The team's role is, primarily, to meet all requirements presented by the customer and IDI. We are responsible for development of the project. Our interest in the project is to receive experience with new technologies and project management, as well as to receive satisfactory grading. The project was intended for 5-7 students, and the group started out with 5 members, but unfortunately one had to drop the course, which led to a group count of 4. This will, in some way, affect what the team can manage towards what was expected when the project was put together.

1.3.2 NTNU

NTNU (Norwegian University of Science and Technology) has the main responsibility for higher education in Norway. NTNU has a rich and diverse set of educational roads to pursue for instance faculty of architecture, faculty of humanities, faculty of information technology (which is the origin faculty of this course), and many more. There are about 22 000 students at NTNU, and of them about 1800 are exchange students. NTNU's interest in the project is to provide realistic experiences of high quality in order to educate students, who will later be able to perform better in real life employment situations. [2]

IDI

IDI (Department of Computer and Information Science) at NTNU is hosting the Customer Driven Project and will oversee the process and have the final say in determining our grade. IDI is providing an advisor who serves a one-man steering committee and is providing advice and guidance for the group throughout the project. [3]

1.3.3 Netlight

Netlight, our customer, is a Swedish IT- and consulting-firm. Their field of expertise is within IT-management, IT governance, IT-strategy, IT-organisation and IT-research. They deliver independent solutions based on the customers specs. With the broad field of knowledge they can handle whatever tasks presented by their customers. They reach this goal by focusing on competence, creativity and business sense. Peder Kongelf will be our contact person to Netlight. Throughout the project a lot of time will be spent in meetings with him to better understand what is needed to make sure the project is going in the right direction. Netlight's role is to describe and define the requirements of the project. [4]

1.3.4 Contact information

Contact information on the involved members of this project.

[2] http://www.ntnu.no
[3] http://idi.ntnu.no
[4] http://www.netlight.com/en/

Table 1.1: Contact information

Person	Email	Role
Ivo Dlouhy	idlouhy@gmail.com	Team member
Martin Havig	mcmhav@gmail.com	Team member
Øystein Heimark	oystein@heimark.no	Team member
Oddvar Hungnes	mogfen@yahoo.com	Team member
Peder Kongelf (Netlight)	peder.kongelf@gmail.com	The customer
Stig Lau (Netlight)	stig.lau@gmail.com	The customer
Meng Zhu (NTNU)	zhumeng@idi.ntnu.no	The advisor

1.4 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 funcionality 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.*

1.5 Structure of Report

This report describes the development of bringing the scripting experience to the power user. The report is structered after the course of the project, and gives the reader insight into the development of the system. Chapter 1, the introduction chapter, will introduce the problem to the reader, introduce the members and stakeholders of this project and explain the motivation for doing this poject. After reading

this chapter, the reader will be left with an overall view of the projects outline and goals. Chapter 2, the prestudy chapter, Chapter 3, the prestudy chapter, Chapter 4, the prestudy chapter, Chapter 5, the prestudy chapter, Chapter 6, the prestudy chapter, Chapter 7, the prestudy chapter, Chapter 8, the prestudy chapter, Chapter 9, the prestudy chapter, Chapter 10, the prestudy chapter, Chapter 11, the prestudy chapter, Chapter 12, the prestudy chapter,

Chapter 2

Preliminary Study

Contents

This chapter is ment to outline the preliminary study of our project. This includes what our technologies aims to achive and how we will use them to achive this. Beyond this, the chapter will show the methology the team chose too use and why this was a natural choce 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 considrible time is put into this part of the project, to assure the team makes good choices when it comes to thecnologies to use.

2.1 Concept

The efficiency of the power user can always be increased. And one way of achieving this efficency 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 timeconsuming in the long run, and also inefficient. With an envinonment 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 oppurtunity 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.

2.2 Similar solutions

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

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 moddability 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 modelling software features a Python console with access to the model 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

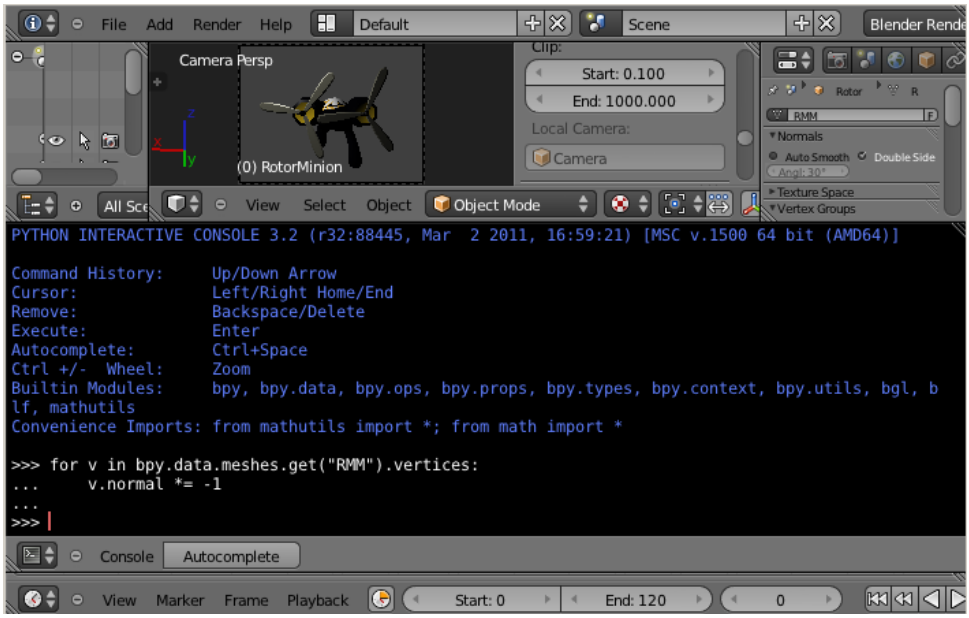

Figure 2.2: Blender Console

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

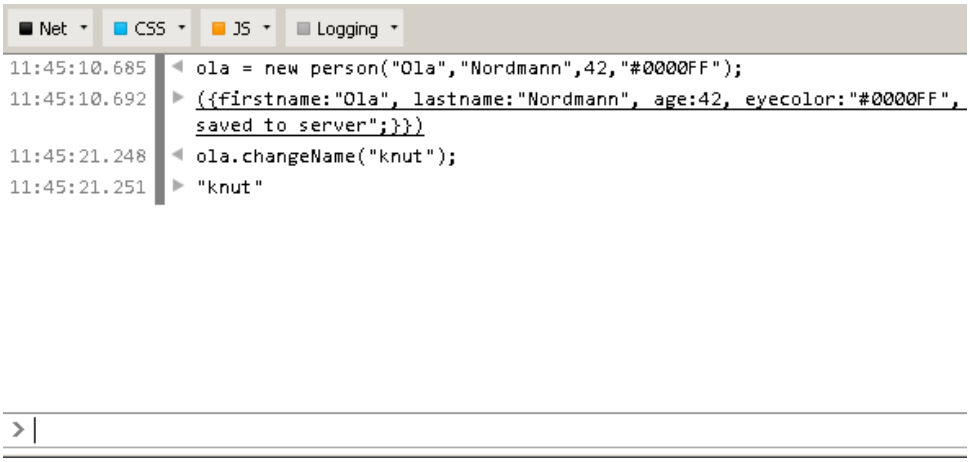

Figure 2.3: Firefox Web console

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

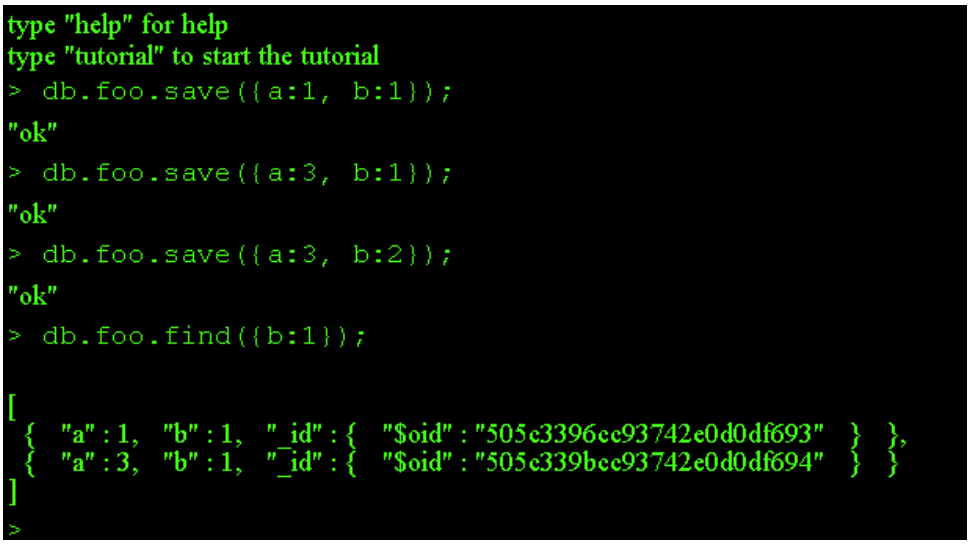

Figure 2.4: MongoDB Console

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

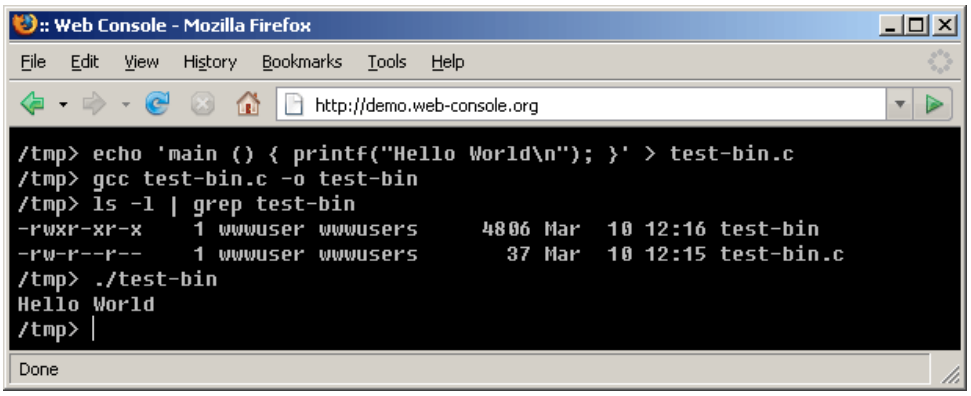

Figure 2.5: Web- Console

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.

2.3 Development language and technologies

2.3.1 Collaboration

Redmine

Redmine is an online project management tool. It is an open source, cross-platform application. If offers role based access control, issue tracking system, Gantt chart display and browsing source code saved on git. We use this application for project planning and creating, assigning and logging issues. It can also be used as an alternative source code browser with its own automatically replicated copy of github repository. Information stored in the system is very important, so we set up automatic backup of the database to our git repository. [1]

Google Calendar

[1] http://www.redmine.org/projects/redmine/wiki

Google Calendar is online time management tool, which enables users to create events in shared calendars. It is also possible to invite people to attend an event. We use it for planning the meetings, and put in important lectures and deadlines. Shared calendar is administered by team members. It will let the members use their favourite mail, whilest still receaving invitations to planned meetings. [2]

Our group Google Calendar is available on page: https://www.google.com/calendar/embed?src=6j792hr87jahd4vpnj636h1

Google Groups

Google Groups is online communication tool. It offers group management and communication through web interface and email. We use it as a mailing list tool. Contact person publishes important news and information. Meeting dates and notes are distributed to all members of group. Customer and advisor can send emails directly to group email address, so that if contact person is not available at the moment, anyone from group can answer.

Our Google Group mailing list address is: ntnu-netlight-project@googlegroups.com [3]

Google Drive

A file storage and synchronization service. Google Drive is the home of Google Docs, which is a set of applications such as document editing, spreadsheet, powerpoint, etc. We use these applications to easy share all documents between all the members of our project. It lets us edit and work with the same documents separately at the same time, and keeps a safe backup in the Google Cloud of our files if some computers where to break down. [4]

Skype

A free P2P/client-server communication program. It lets the users communicate with each other through voice, video, instant messaging and file transferring. The users needs a Skype account to reach other Skype users. This program is a VoIP service which lets us communicate without any extra cost if connected to a network. We mainly use Skype for discussions when the group is not gathered, this also gives us a transcript of what was discussed. All the members possesed a skype account, which made it easy to set up a project group. [5]

GitHub

Web-based hosting service for software development. It uses the Git revision control system. Git focus on speed, and gives the user full revision tracking. GitHub uses this to present different functionalities to the users, such as setting up repositories, forking repositories, writing wikis and setting up web pages for the repositories. This integrates well with the shared code and report writing. [6] [7]

[2] https://www.google.com/calendar
[3] https://groups.google.com/googlegroups/overview.html
[4] http://en.wikipedia.org/wiki/Google_Drive
[5] http://en.wikipedia.org/wiki/Skype
[6] http://en.wikipedia.org/wiki/GitHub
[7] http://en.wikipedia.org/wiki/Git_*software*

LaTeX

LaTex is a document markup language that uses the TeX typesetting program. The main focus of LaTex is to make authors able to focus on the content of what they are writing without worrying about its visual presentation. It is mainly used to transform XML- based documents to PDFs. LaTex is widely used in academia, to produce scientific reports, etc. This, and that the customer suggested it, made it a natural choice for the report.

2.3.2 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 synchronise 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 adressing some of the following points: being non-relational, distributed, open-source and horizontally scalable [13]. 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 customised 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 queries 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 [22] applications like Hadoop [20], which recently has become a very popular subject within the computer science community. BASE [15] instead of ACID. Document NoSQL databases seemed like they would fit our project quite well. [11, 19]

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 optimisation techniques and most of them use some kind of structured query language. Typically supports 4 basic operations which is select, update, delete and insert. SQL databases are schema defined and follows the ACID(atomicity, consistency, isolation, durability) principles.

Database Alternatives

Below, some of the database implementation available to us are discussed. We mainly investigated document NoSQL databases and regular SQL databses. 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 prioritises 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. [5, 6]

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, and should therefore be considered when the question comes to 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 working with the clients. This alone is a hard negative towards MySQL. [4]

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. As a result, instead of explicitly adding functionality, we can add restrictions to allow the user to add functionality within a certain scope. 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 model 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 model 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.

2.3.3 Backend

Node.js

Node.js is a platform designed for server side applications. It is written in JavaScript using Google Chrome's JavaScript runtime called V8. The same runtime is used by the MongoDB database engine, so it shares some of its properties. It can be used for building fast and scalable network applications. Also it is very lightweight and efficient [8].

ASP.NET

ASP.NET is a part of the .NET Framework used for creating web applications and services. The framework is language independent, so you can use any language that .NET offers. It also allows programmers to use similar approach in web applications as in desktop applications. On the other hand is not as lightweight as Node.js [12, 14].

PHP

PHP is a acronym for Hypertext Preprocessor. It is a serverside programming language mainly used for web applications. PHP is easy to use and supports a lot of modules, platforms and database systems. One of the main disadvantages is, that is not as effective for web applications, because it is interpreted and doesnt keep application context [21].

Ruby on Rails

Ruby on Rails is a web application Framework using the MVC, Model View Controller model. It is a Ruby language module for building dynamic web applications. This framework also offers variety of modules, which can extend functionality or simplify tasks. The code is run the same way as Node.js and PHP, they are all interpreted, but Ruby on Rails is more difficult to install [16]

2.3.4 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.

JavaScript

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 Model elements.

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

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-

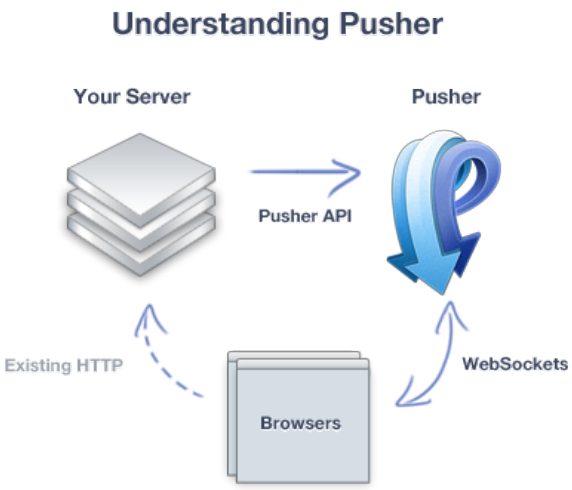

Figure 2.6: Pusher Explained

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.

2.3.5 Synchronization Technologies

We will be adding a library for bi-directional real time communications between the server and the client, to easily detect changes in the objects on both sides and to replicate these changes to the other side lightning fast. This functionality will ensure data consistency between the client and server side. To make these updates fast and to avoid extra work in creating this functionality ourselves, we will employ an external library to get the work done.

Pusher

Pusher is a cloud based system which offers a hosted API. It relies on the use of HTML5 WebSockets, which provides bi-directional communication over a TCP channel. It is a widely used solution which is well documented, and it support for a lot of libraries for both the server and client side. The web-site offers tutorials and extensive documentation on the most popular libraries. Pusher creates channels that can be both listened and published to. If multiple devices are connected to the same channel, they will receive any messages sent to the channel almost simultaneously. Pusher offers a free account(an account is needed to use the system) which offers all the basic functionality we need, with up to 20 connections and 3 million messages per month.

PubNub

Like Pusher, PubNub is a push service hosted in the cloud. It is written entirely in C, which gives it extremely fast performance and enables it to push over 1 million messages a second. It offers great documentation and support for the most popular libraries on both the client and server side and its in widespread

use. Like Pusher it relies on channels for communication which you can subscribe and publish to, and in essence the two solutions work in the same intuitive way. PubNub offers a free account with 1 million messages a month and up to 100 connections.

Other Technologies

We considered other similar alternatives as well, like Socket.io, Vert.x and Akka. But they either lacked support for the technologies we have chosen for the system, or lacked the extensive documentation and widespread use that Pusher and PubNub provides. We were also left with the impression that we would spend more time implementing these services than if we opted for either Pusher or PubNub.

Conclusion

Pusher and PubNub are both great systems that are widely used and they both offer extensive documentation. They cover the specific functionality we need for this project, which is to replicate the changes on both the client and server side. They both offer free accounts with more than enough connections and messages each month to cover our needs. So this decision will come down to our gut feeling. As none of the developers have any experience in using either system, the most important factor for this decision is that the system is well documented and easy to use. During research it became apparent that PubNub is the most widespread solution as of today. If we run into any problems implementing and using the system, it is likely someone has already provided a solution for it. So the decision ultimately fell on using PubNub.

2.3.6 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.

2.4 Development Methodology

2.4.1 Agile vs Waterfall

The waterfall method focus on planning the future in detail. It follows the principle of "Big Design Up Front". It relies on the fact that you are able to report exactly what features that are going to be implemented and tasks are planned for the entire length of the project. It forces you to specify all the requirements early in the development, when you actually know the least about the project and the problems that are to be solved. The rationale behind this is that time spent early on making sure requirements and design are correct saves you much time and effort later. A development team using the waterfall method will only consider to implement the most valuable changes, as changes in this process are time consuming and often requires that completed work is started over. The method places a lot of emphasis on documentation.

Agile methods, as opposed to the predictive methods, are designed to plan for changes in the requirements and features of a project. It emphasises on working code as primary measure of progress, instead of extensive documentation of for example the requirements. Agile methods consists of iterative and incremental steps in the development process, where requirements and solutions evolve through the course of the project. Requirements are bound to change, either because the customer didn't understand the problem in the beginning or because they would like to add new features. Agile methods facilitates the ability to accommodate these changes. Most agile methods includes delivering a working product in incremental stages, and gives the customer something to relate to during the developments process. The CHAOS Manifesto is a survey published by the Standish Group each year and it measures the success of IT- projects. It divides the projects into 3 groups; Success, meaning it completed on time and budget, with all features and functions as specified. Challenged, meaning it completed, but was over cost, over time, and/or lacking all of the features and functions that were originally specified. Failed, meaning the project was abandoned or cancelled at some point and thus became a total loss. As Figure 2.7 illustrates, agile methods although not perfect by any means, more often result in products that are successful(the method used for measuring the success of a project is to some degree debated, but the results serves a purpose nevertheless).

2.4.2 Agile Methods

There exists a lot agile methods for software development, and although all of them follow the basic principles of agile development, they differ in a lot of areas. Following is a detailed description of three different agile methods.

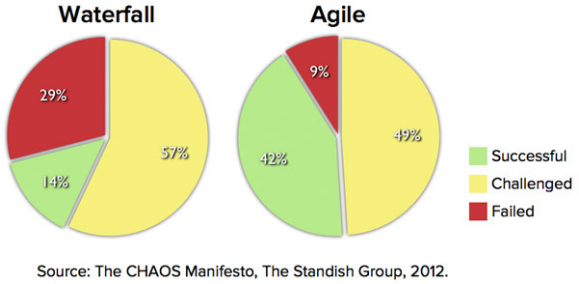

Figure 2.7: Waterfall vs. Agile

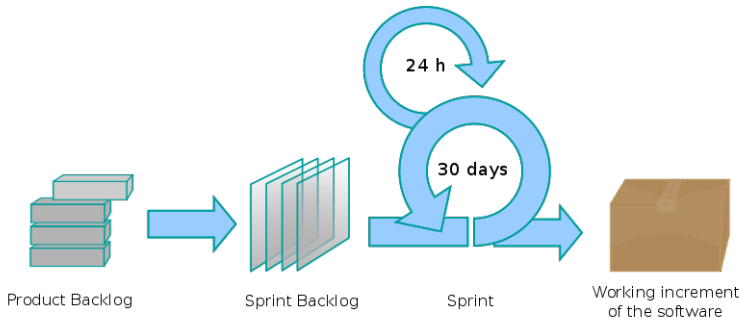

Figure 2.8: Scrum Process

Scrum

Scrum is an iterative, incremental software development model with several short sprints - complete small sets of tasks each sprint.

The Roles:

- The Scrum master, who is responsible for leading the process and to enforce the Scrum rules onto the team. He has to make sure that the development team does not overestimate what they can handle during one sprint. He leads the scrum meetings and enlightens and handles obstacles that may appear.
- The product owner, represents the stakeholders and is the voice of the customer.
- The development team, is responsible for delivering potentially shippable product increments at the end of each Sprint. A development team is made up of 3–9 people with cross-functional skills.

The Sprints:

- Normally last from 7 to 30 days .
- Starts with a planning meeting, where tasks are identified and goals for the sprint is set.
- Product owner tells the team what tasks should be done in the sprint.
- The tasks comes from a prioritized list of requirements called the backlog.
- The team determines what is possible based on this and records this in a sprint backlog.
- The goals should not be changed during the sprint.

The Scrum process is well suited for projects where its difficult to plan too far ahead, where at least some of the aspects of the project are unknown. Its a versatile process which is gives you the ability to handle changes in the requirements and demands from the customer. It allows for the developers to work on different parts of the project at the same time. The design, requirements of the system are not set in stone from the start, and are allowed to evolve during the process. The process delivers unfinished versions at the end of each sprint, which gives the customer a chance to try the system and give continuous feedback to the developers. The Scrum process is somewhat complex, and it will take time to properly learn and execute the method. You also have to decide on what type of Scrum you are going to use, as there exists multiple forms of Scrum. This can prove to be a time consuming process. And even though the team members know Scrum, they will have to learn the version of scrum decided upon, if it turns out to differ from the one they are used to. [8]

[8] `http://en.wikipedia.org/wiki/Scrum_(development)`

DSDM Atern

DSDM(Dynamic System Development Method) is an agile software development method, and it was originally meant to provide some discipline to the Rapid Application Development method. The most recent version was launched in 2007 in an effort to make DSDM tool and technique independent, and its called Atern. DSDM is an iterative and incremental approach that embraces principles of agile development, including continuous user/customer involvement. It enforces you to deliver incremental versions of the product to the customer, where the main criteria of acceptance is that it meets the current business needs of the customer. It follows the principle that it is always better to deliver something "good enough" early than to deliver everything "perfect" in the end.

DSDM as a method fixes costs, quality and time at the beginning of the project. Through a prioritization method called the "MoSCoW Method", with musts, shoulds, coulds and won't haves, it adjusts the scope of the project to meet the given time frame. This allows the development team to focus on the critical functions of the system rather than delivering a perfect system. The method puts a strong focus on actively involving the customer in the development, and continually confirm the solution. The principle-list of DSDM is quite long and complex. For a team that is not experienced in using the method, the process of learning the method will be time consuming. Also, always having to display the progress to the customer can be time consuming and hinder the development effort. Testing is central and shall be done through the whole development process. [9]

Extreme Programming

Extreme programming, hereby referred to as XP, is an agile method designed to reduce cost of changes in requirements by having multiple short development cycles. It includes elements such as pair- programming, extensive code review and unit testing of all the elements of the code. It emphasises frequent communication with the customer and between the developers. The method embraces changes in the requirements of the project, and it doesn't attempt to define a stable set of requirements at the beginning of the project. In XP a representative for the customer is always available on site to answer any questions the developers might have. It also focuses on frequent releases of working code which serves as checkpoints where the customer can add new requirements. XP puts a lot of focus on the code of the project, the advocates of XP argues that the code is the only truly important product of the system development process. XP as a process does not produce a lot of written documents during the development of the project. In XP programmers are expected to assume a unified client viewpoint and focus on coding rather than documentation of compromise objectives and constraints. [10]

2.4.3 Conclusion

If all of the requirements of this project were known in advance and provided by the customer, or the features of the finished product was known and unlikely to change, the waterfall method might be the way to go. But this is a prototype, proof of concept type of project where very little is known about the final product. We were certainly not presented with a finished set of requirements at the beginning of the project, and the requirements we settle on before we start the implementation are also more than likely to change during development. These kind of changes in a waterfall process will be time consuming. Thats why we think that an agile development method will be the best choice for this project.

All of the agile methods described above exhibits properties that will come in useful in this project. They are all based on iterative and incremental development steps, delivering prototypes for the customer to test after each step. This will give the customer a chance to try out unfinished versions of the product and give continuous feedback throughout the project. It can also help the customer to identify new features that they would like to add. They also allows the customer to decide which features to implement in each step, ensuring that the final product will contain the features the customer really need. The agile methods embrace changes in the requirement and provides ways to handle those changes. They also

[9] http://en.wikipedia.org/wiki/DSDM
[10] http://en.wikipedia.org/wiki/Extreme_Programming

encourage tight and continuous communication with the customer, which is important to be able to deliver a product that the customer is satisfied with. Of the three methods the group members are most familiar with the Scrum process. The DSDM method is complex and will take a considerable effort to learn and execute correctly. As none of the group members have used DSDM, and we have a limited amount of time in this project, DSDM is of the table. XP puts a lot of focus on the code, and delivers a minimal set of documents. We are to write an extensive report about the project, and document every part of it, including compromises and assumptions made. The amount of code in this project will be limited and none of the team members have any experience in working with XP. As a result, XP seems like a bad fit for our project. The Scrum process does not put as much focus on documentation as the waterfall process, but we think that the amount of documentation produced during the Scrum process will be satisfactory for the report. It will take time to learn how to execute Scrum properly, but since all of the group members are familiar with the basics of process, we think it won't be too time consuming and worth the effort. The final decision then is to use Scrum as a development method 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. [18]

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.

[3]

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 Code conventions

JSLint

JSLint looks through JavaScript code and detects potentially problematic and unusual syntax in the code to improve the quality of the code.

JavaScript was originally aimed for web pages with small tasks where the Java platform was too heavy to run and develop for. But JavaScript is capable of much more, and has became used in more and larger projects. However, the lenient and flexible syntax useful for small tasks has proven to make JavaScript troublesome for larger projects. JSLint helps the developer in handling this. [9]

JSHint

JSHint is a fork of JSLint which was designed with similar goals of enforcing code quality and avoiding common programming errors, but in a more lenient manner. JSLint is criticized for being a tool that is not first and foremost helpful in preventing bugs or improving readability, but rather a tool that enforces any strict convention as a goal in itself. JSHint attempts to alleviate this, and is more configurable and exhibits less absolutist behaviour. [2]

Comparison and Conclusion

We chose JSHint, this is why. JSLint is a stricter version of JSHint. We started out with JSLint after we had written some lines of code, and saw that it produced closer to a 100 errors, and satisfying the linter would mean to rewrite most of the code. After this discovery we tested the JSHint fork of JSLint, which "make a more configurable version of JSLint—the one that doesn't enforce one particular coding style on its users—" - [11]. JSHint is also much more involved in the JavaScript developer community, and based on this. JSHint did not produce as many errors, and did not require as much effort to satisfy. JSLint enforces its own way of coding, which is not always what you want from a linter. The team switches between different coding languages both inside and outside the project. It is unnecessarily time consuming to get into the mind of Douglas Crockford (maker of JSLint) every time you switch to JavaScript. JavaScript is a dynamic language which lets you write the same code in so many different ways, and the way of JSLint isn't always the way. [12] [13]

2.7 Version Control

2.7.1 Git

Git is a open source project for distribution and version control of source code. Linus Torvalds wanted a source code manager with emphasis on speed, and git was the product of this. Git has the features show in table 2.6.

2.7.2 Subversion

Apache Subversion (SVN) is open source. It is a cross-platform version control tool, used mainly for sharing source code, web pages and documentation. SVN is designed to be central, where you commit to one server. Some of SVN's features are shown in table 2.7.

[11] `http://www.jshint.com/about/`
[12] `http://www.jslint.com/`
[13] `http://www.jshint.com/`

Feature	Description
Branching and merging	Through this the developer can have multiple branches of the system, which can be independent of each other, which makes rollbacks, merging and editing simple and efficient.
Small and fast	Operations are made locally, which makes more efficient than other similar systems such as SVN.
Distributed	The distribution works through cloning of the working repository, this means that every user has a backup of the system in case of a crash. The distributed code can be pushed back to the shared repository with changes, and will update the system for other users to fetch back to their own clone.
Data assurance	The files and commits in the repositories has its own checksum, and can be retrieved through this checksum, this assures that a user cannot fetch anything out of the repository other than what has been put in.
Staging area	Before completing a commit, there is possible to review the changes. This makes it possible to decide upon which files to commit, if not all changes are to be pushed to the main repository.
Free and open source	The user are free to inspect the source code or contribute to the project. Git is under the GPLv2 software license

Table 2.6: Git features

Feature	Description
Branching and tagging are cheap	This can be done in constant time
Merge tracking	SVN lets the user track the changes between lines of development
Standalone server option	The standalon option lets the user run their own server, other then an apached HTTPD server
Interactive conflict resolution	Merging conflictsis made easy through either graphical user interface of the commandline.
Atomic commits	There is no part of the file will be commited if not the whole file is commited.
Free and open source	The user are free to inspect the source code or contribute to the project. SVN is under the Apache License

Table 2.7: Subversion features

2.7.3 Conclusion

Most of the team is fearly familliar with the Git version control tool. It is important that it is not centrelised like the Subversion tool, git aloves the members to easy work offline, since git lets the user always be "online" through their own cloned repository. The git version control tool also alos the team to disribute the reportfiles, images and database-files more easily since git is a "file contetn managment tool". The Git repository is easy to distribute of use through GitHub 2.3.1, and makes the project reachable for all the team members. The mentioned points and the fact that the team is more familiar with Git then SVN made us choose Git as our version control system.

Chapter 3

Project management

Contents

In order to successfully actualize this project, it is necessary to establish a common, high-level understanding of the team, external factors influencing the project, the project itself, and how to organize the work. The project management chapter is meant to document this.

3.1 Team Structure

We are a very small development team of just four team members. In a typical software development project, there are a large number of different roles, so each of us have to be assigned several different roles. We have defined a role for each responsibility that we believe to be important in our process. The person who is assigned a role should have an overview of the progress and have control over which tasks need to be done on their area. The work itself may be broken down into tasks so that these tasks can be executed by anyone from the team. The roles in our team are listed in the table 3.2, overview of the roles is in table 3.1.

Team member	**Roles**
Ivo	Group leader, Customer and Advisor Contact, Scrum Master
Oystein	Test Manager, QA Manager, Weekly Report Manager
Oddvar	GUI Designer, Code Master, Meeting Secretary, Time Keeper
Martin	System Architect, Report Manager

Table 3.1: Team role overview

Table 3.2: Team roles

Role	**Description**	**Assignee**
Team leader	Is responsible for administrative tasks and makes the final decisions.	Ivo
Scrum Master	Shields the development team from external distractions and enforces the Scrum scheme.	Ivo
Customer Contact	Handles communication with the customer. The customer should contact this person regarding general requests, questions and reminders.	Ivo (backup Martin)
Advisor Contact	Handles communication with the advisor. The advisor should contact this person regarding general requests, questions and reminders.	Ivo (backup Martin)
System Architect	Is responsible for the system architecture including distinctions and relations between subsystems and general code design choices.	Martin
Code Master	Overall responsible for code management and structure. Managing branches in Git repository.	Oddvar
GUI Designer	Is responsible for the layout and design of graphical user interfaces.	Oddvar
Test Manager	Is responsible for testing including unit tests, integration tests and usability tests.	Øystein
Report Manager	Is responsible for delegating and overseeing work on the project report.	Martin
Customer Representative	Participates in regular meetings to discuss the progress, project status and future tasks. Represents the customer.	Peder Kongelf
Customer Technical Advisor	May be consulted about technical aspects of the project.	Stig Lau
Advisor	Serves as a one-man steering committee for the project.	Meng Zhu
Meeting Secretary	Is responsible for making sure notes get written and sent after each meeting with the advisor and customer.	Oddvar
Quality Assurance Manager	Responsible for the process of agreeing on QA measures for the project and imposing these on the team	Øystein
Weekly Report Writer	Is responsible for finalizing the weekly report(s) for the advisor and customer, and getting these delivered for approval. Also responsible for meeting agendas and their delivery.	Øystein
Time Keeper	Responsible for making sure that everybody is logging their work, and logging team activities.	Oddvar

3.2 Concrete Project Work Plan

The course staff recommends us to work 25 person-hours per week and student. This project is estimated for 14 weeks. Since we at the moment have 4 group members in our group, the available effort will be $14 * 25 * 4 = 1400$ person hours including own reading, meetings, lectures, and seminars. The customer requested 5-7 students to handle this project, it is regrettably not likely that we will be supplied by one extra group member, so we must expect some more work hours divided on the four of us.

The first 4,5 weeks of the project were used on project planning, pre- study on the domain of the problem, to deduce user stories for the product backlog and to create the overall architecture of the system. The project is a proof of concept type of project, so it was important that we used a considerable amount of time on studying the domain of the problem and to find technologies that would solve the task as best as possible. Our Scrum process consists of 4 sprints, each 2 weeks long. In each sprint we had 200 work hours available, but a part of this will be used on report writing, project management, lectures and meetings. The two first sprint had an estimate of 130 hours available for development tasks, while the two last had an estimate of 140 work hours available. The last week will be used to evaluate the project, finish the report and to prepare the final presentation. The timeline of the project is outlined in Figure 3.1, and the WBS of the project is given in Table 3.4.

3.2.1 Activities

Lectures

Lectures held by the course TDT4290 Customer Driven Project mainly aim to educate the students in the in the art of project management and the different tools used to handle this task in a better way. The lectures can give good insight into the workflow and how to avoid different issues, and should therefore be attended when the lecture involves tools used by the group or tools the group should consider using. After the lectures the group will reflect upon the presented material and if it is usable, or add value to our project, it will be integraded into the project.

3.2.2 Milestones

These are the deliverables and deadlines, that we have to take into account.

- August 21, Project start
- October 14, Pre- Delivery: Deliver a copy of the Abstract, Introduction, the Pre-study and the Choice-of Lifecycle-model chapters to the external examiner (censor) and technical writing teacher. Also deliver the outline of the full report (Table of Contents).
- November 16, Code freeze: Final deadline for the completion of the implementation. The code should not be changed after this date.
- November 22, Final Delivery: Project end. Deliver final report and present and demonstrate the final product at NTNU. Four printed and bound copies of the project report should be delivered, as well as one electronic (PDF) copy.

3.2.3 Sprints

Sprint deadlines: The pre- study, requirements, and testing plan activities should be finished before the start of the first sprint. If this is not the case the number sprints and their deadline might change. The start and end dates of each sprint is listed in Table 3.3

Table 3.3: Sprint Deadlines

Sprint Nr.	Start	End
1	24. September	5. October
2	8. September	19. October
3	22. October	2. November
4	4. November	18. November

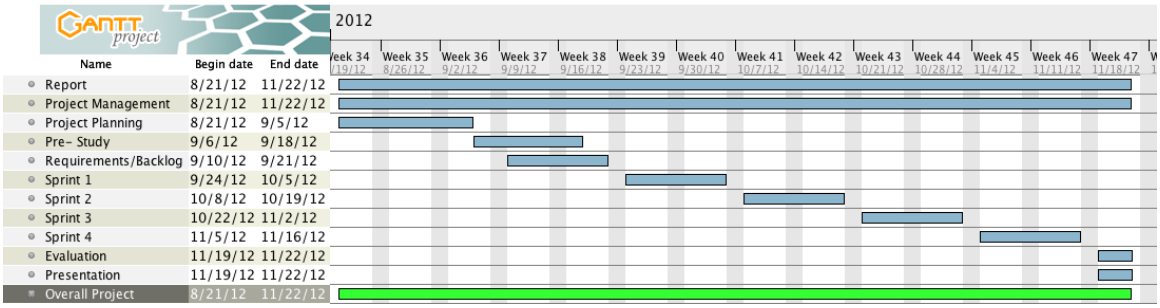

Figure 3.1: Gantt Chart

Table 3.4: Work Breakdown Structure

			Hours	
Task	From date	To date	Est.	Act.
The report	21/08/2012	22/11/2012	300	320
Project Management	21/08/2012	22/11/2012	180	163
Project Planning	21/08/2012	05/09/2012	100	74
Lectures	21/08/2012	05/09/2012	40	50
Pre- Study	06/09/2012	18/09/2012	80	89
Backlog	10/09/2012	21/09/2012	60	47
Architecture	17/09/2012	21/09/2012	40	51
Sprint 1	**24/09/2012**	**05/10/2012**	**130**	**128**
Planning			20	12
Design/Implementation			90	112
Testing			20	16
Sprint 2	**08/10/2012**	**19/10/2012**	**130**	**107.5**
Planning			20	22
Design/Implementation			90	65
Testing			20	20.5
Sprint 3	**22/10/2012**	**02/11/2012**	**140**	**95**
Planning			20	23
Design/Implementation			100	47
Testing			20	15
Sprint 4	**05/11/2012**	**16/11/2012**	**140**	**40**
Planning			20	14
Design/Implementation			100	19
Testing			20	7
Evaluation	19/11/2012	22/11/2012	30	42
Final Presenation	19/11/2012	22/11/2012	30	34
Total			**1400**	**1240.5**

3.3 Constraints

3.3.1 Time

The project must be completed within a timeframe of 13.6 weeks. Customer Driven Project is a 15 study points course, which makes it half a semester worth. The guideline is 24.3 work hours per person, per week. The project will be presented to a sensor on the 22th of November 2012.

3.3.2 Knowledge about the problem

Our problem is not well-explored and the customer cannot give us exact requirements. Thus, this is as much a research project as it is a software development project.

3.3.3 Issues regarding the system

The customer has a huge workforce at their disposal, and it can be introduced to us through the customer. If there would appear a problem regarding the system we can't handle ourselves, we can contact the customer and they can forward the issue to other sections of the corporation for analysis and problem solving. The communication regarding issues of this kind will usually be done per email.

3.3.4 Issues regarding the workflow

The advisor can help us with issues regarding project management and workflow issues. If the group were to be stuck at some point, the advisor can jank the group out of the ditch and set it back on track.

3.4 Quality Assurance

3.4.1 Communication rules

Communication with the customer is usually done by email. The customer will contact the assigned customer contact if anything else is not specified. The assigned customer contact is responsible for relaying any information received from the customer to the other group members as soon as possible. The customer contact is also responsible for sending any information from the group to the customer. Any communication from the customer demanding a reply, will be replied to within 8 hours. If any communication is sent from the group to the customer demanding a reply, this reply should be received by the group within 24 hours.

The same rules apply to the communication with the advisor.

3.4.2 Internal Routines

Each group member is responsible for logging all their work hours in the timekeeping system within 22:00 the same day. All group members are also responsible for checking the Skype group conversation within 22:00 Monday to Thursday, to see if any important information is posted there. If important information has to be delivered after this time, it will be sent by mail. All group members are required to check Redmine each day to check if any new task are assigned to that person.

3.4.3 Deliverables

All deliverables, including source code, documents and meeting notes, will be approved by at least two of the group members before it is sent for approval by either the advisor or the customer. All major phase documents must be approved by all the group members before it is relayed to the advisor for approval.

3.4.4 Meetings

Advisor meetings

Advisor meetings will occur every Thursday 09:15, unless anything else is specified, so the advisor gets updated on the latest work and progress of the project. The day before the meeting the group sends the advisor an agenda for the meeting and a status report of what is done since the last meeting. This will be sent to the advisor within 14:00 the day before the meeting. If this deadline is not sustained, any needed documents will be printed out and brought to the meeting. This lets the advisor enter the meeting prepared, and improve the efficiency of the meeting. Under the meeting issues regarding the work done, plans for next week and project management will be discussed. If there is a sprint-end-week the demo for this sprint will also be shown. Advisor notes will be written during the meetings, compiled and sent back to the advisor within 12:00 the following day.

Customer meetings

Customer meetings will occurs on demand, but usually weekly, and Thursdays 19:15 if there is a sprint-end-week, unless anything else is specified. Meetings will be scheduled at least 48 hours before the meeting starts, and confirmation from the customer should be received at least 24 hours before. An agenda for the meeting and a weekly progress report will be sent to the customer to keep him in the loop of the progress, and get prepared before the meetings. This documentation, including other documents needed for the meeting, will be sent by mail within 24 hours before the meeting. Since the customer is busy during work hours, the meetings are kept after 18:00 on weekdays, and after 12:00 in weekends. They will be kept over Skype, since the customer is located in Oslo. The meetings will be used to get the customer in the loop, clarify uncertainties and redirecting, if the direction of the project taken is not what the customer had in mind. The customer should also be included in the sprint planning meetings to help with use case prioritization. Customer notes will be written during the meetings, compiled and sent back to the customer within 12:00 the following day.

Internal meetings

Internal meetings will occur every Monday 12:15, unless anything else is specified. At this time all the group members are open for meetings. During these meetings the plans for the week will be discussed, workload divided and what was done last week. Notes will be taken and documented for later review. The group meet daily on skype to keep everyone up to date on the project progress and what will be done.

Sprint planning meetings

Sprint planning meetings will be held in the start of every sprint. Which use cases to handle and complete will be discussed here. The tasks will also be discussed with the customer so the customer can help prioritise the use cases and tell us which tasks they want us to complete first. The selected user stories along with their Work Breakdown Structure will be sent to the customer for approval.

Sprint demonstration meetings

Sprint demonstration meetings will be held Thursdays 19:15 in the end of the each sprint, unless anything else is specified. The group and the customer will be attending these meetings. An agenda will be sent to the customer, together with the weekly report and a link to the system, for the customer to test out. The demo will show the system and its new functionality. This lets the customer see the progress, and be able to come with inputs towards how their minds might differ from what has been produced, so the group can get on the right track if that is needed. This lets us make sure that we do not stray too far from the intended system. Since this is a meeting held with the customer the rules from the customer meetings will be held, so customer notes will be written during the meetings, compiled and sent back to the customer.

3.4.5 Version Control

GitHub was selected as the tool for version control in this project. All the relevant work that is produced will be pushed to the repository using git, including all the documents for the report, source code, images, diagrams, and so on. The group members will commit and push their changes on a regular basis, ensuring that the repository is always up to date and available.

3.4.6 Document Templates

The group has created templates for the following documents:

- Weekly status report B.1
- Meeting agenda B.2
- Meeting notes B.3

These templates are listed in the appendices.

3.5 Scrum Process

As explained in the pre- study, for this project we decided to use Scrum as our development methodology. Scrum exists in many shapes and forms, following is a description of how we will carry out the process in this particular project.

Roles

The scrum process contains 3 distinct roles, the Scrum master, the product owner and the development team. The responsibilities of each role is explained in the pre- study. In this project the roles will be filled by

- Scrum master: Ivo
- Product owner: Peder
- Team: Rest of the group, including Ivo

3.5.1 Key Elements

Following is a description of the most important elements of the Scrum process

User stories are a short simple descriptions of a desired feature. User stories are written by the different stakeholders of the project, with particular attention to the end user of the system. For the user stories we will follow a template developed by Mike Cohn, and its written in the form of:

> "As a <type of user>, I want <some goal> so that <some reason>."

The advantage of this form of user stories is that it provides a lot of information in one single sentence. It identifies who the end user is, what the end user wants, and the rationale behind it.

The **product backlog** consists of all the user stories currently derived for the product, and these user stories will be ordered according to their prioritisation from the customer. We will in cooperation with the customer derive the initial user stories for the product, that captures the desired functionality of the system. As the project goes on, new requirements or desired functionality almost certainly will be discovered. New user stories should be created and added to the product backlog. Also the prioritisation of the user stories might change form the customer point of view, and this should be updated in the backlog as a result. The **sprint backlog** contains all the user stories that were picked during the planning meeting. Any user stories remaining at the end of the sprint will carry over in the next one, or put back in the product backlog.

3.5.2 Process

Scrum Planning Meeting

Each sprint will last for two weeks and will start with a sprint planning meeting. In this meeting the work to be done in the following sprint will be discussed and decided on. The customer should be heavily involved in this process, picking user stories from the top of the product backlog, as these have the highest priorities. Each user story should contain an estimate on their workload, and the number of user stories picked depends on these numbers. The scrum planning meeting is in essence a preparation of the sprint backlog.

Daily Meetings

During the sprint daily scrum meetings will be carried out. Ideally should be held the at the same time every day, but in our case this will be difficult as we all have different schedules. We will however set a time for each day that suits all, and these times will be consistent. In the daily meetings the team members will present 3 things to each other:

- What have you done since yesterday?
- What are you planning to do today?
- What issues are you currently facing?

Sprint Demo

At the end of the sprint, a demonstration to the product owner will be held. In this demo we will present what we, the team, has completed during the sprint. We should present something that works, meaning only complete work should be shown to the product owner. The product owner will provide feedback on the finished solutions, and let us know if any changes needs to be made.

Sprint review

The sprint review will be performed by the team following a sprint. In these evaluations we will ask ourselves two main questions, what went well this sprint, and what can be improved in the next sprint.

3.6 Risks

We have tried to identify potential risks early, and document these, by specifying

- The activity during which they may occur and disrupt the workflow
- The risk factor itself; the unfortunate circumstance that might arise
- Impact of the risk, ranked from low(minor inconvenience) to significant(will cause disruption, but we can recover) to critical(must be resolved to prevent project failure)
- Description of the consequences of the risk
- Probability ranked from very low(extremely unlikely) to very high(to be expected).
- Countermeasures to both prevent and recover from the problems
- Any deadline for when the problem must be resolved
- The person responsible for taking action on the risk

Full tables describing our identified risks are listed in the appendices.

3.7 Tool Selection

After detailed study of some tools, we decided to use tools and frameworks listed in table 3.5. In communication and collaboration tools section, the main reason for choosing these tools were last experiences with these - most of us already used these tools in work or in other projects, so we do not have to learn to use something new.

LaTeX software was used to write this report, although most of the team members did not use LaTeX before We decided to use it, because it has a lot of advantages. It is easy to colaborate on source code and use git to track the changes, the result pdf document looks better, than the one produced in other tools, there is automatic referencing of tables, figures and citations. Also the table of contents, tables and figures is automatically generated. Also, using LaTeX was recommended by the customer.

For backend of the system, we wil be using MongoDB in combination with Node.js. Both are extremely effective, scalable and easy to learn. Also they share the same runtime, so the deployment process on the server is simplified. MongoDB and Node.js are newand up-to-date technologies, that are getting more and more used in web applications. NoSQL schemeless MongoDB allows to store any data frontend requires and also the data migration is easy. Node.js can communicate with MongoDB using JavaScrip libraries and offer the data through standard REST interface. Client can retrieve the data from server via HTTP protocol using Ajax with JavaScript and present it to the user. The user interface is written using HTML, CSS and JavaScript.

Part of project	**Technology**
Communication	Skype, Google Groups, Google Calendar
Document colaboration	Google Docs, Google Drive
Code colaboration	Git + Github
Time tracking and mamangement	Redmine
Report	Latex
Backend	Node.js
Database	MongoDB
Communication	PubNub
Client	JavasScript + JQuery, HTML, CSS

Table 3.5: Tools used in project

Chapter 4

Requirements

Contents

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 familliar 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

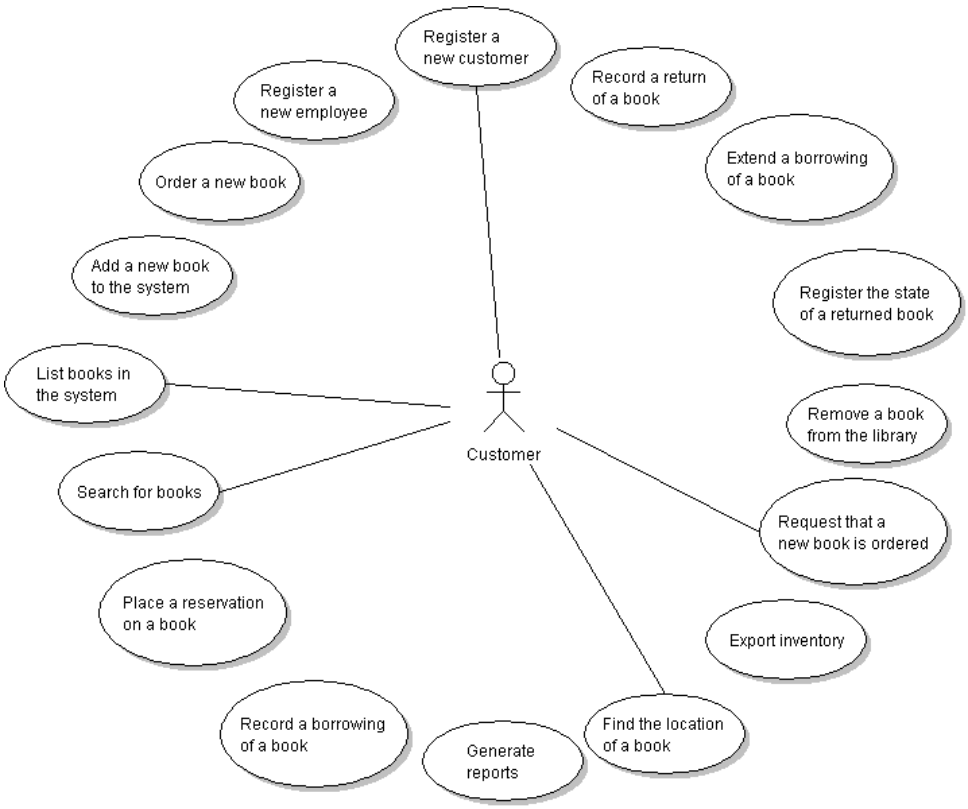

Figure 4.1: Use Case Diagram - Customer

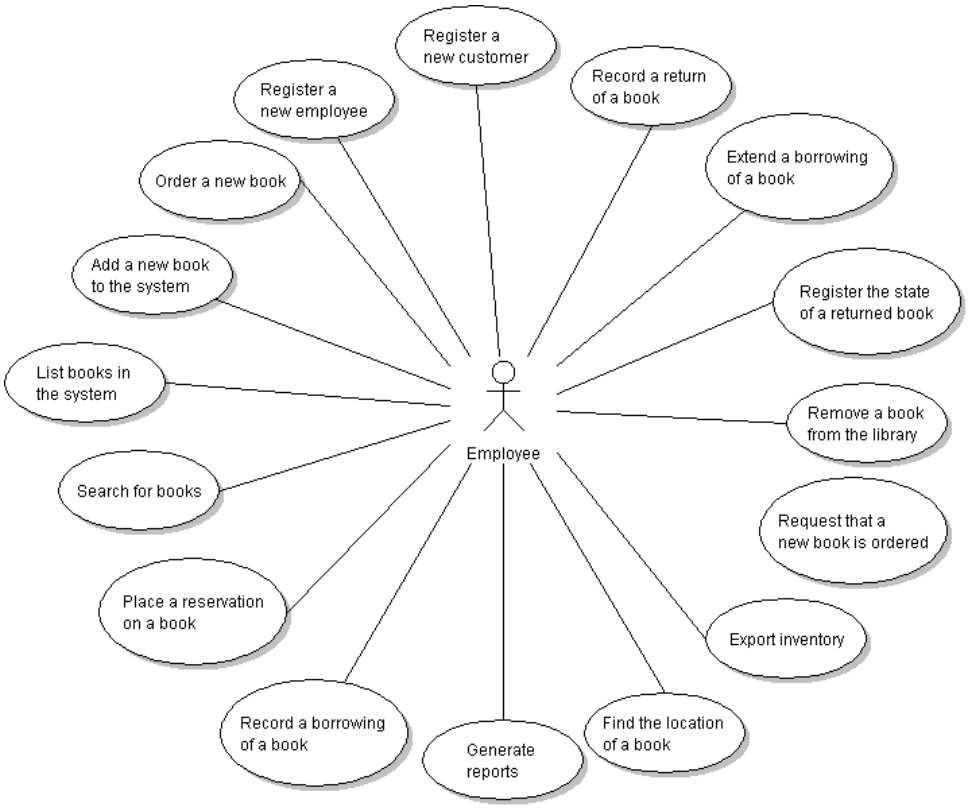

Figure 4.2: Use Case Diagram - Employee

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.

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.

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

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 delete 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 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. 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.

Chapter 5

Test Plan

Contents

This chapter will introduce the overall test plan for this project, and it is based on the IEEE 829 Standard for Software Test Documentation [17]. Some parts of the standard was not deemed to be appropriate for this project and not included as a result. The purpose of this document is to structure the way tests are performed and recorded, assign responsibilities for the testing process as a whole, and to give an outline for the schedule of when the different tests should be performed.

5.1 Testing Approach

The testing methods used for this project is discussed in detail in the pre- study. To sum up, we have opted to utilize both black and white box testing in this project. We will write and run unit tests throughout the implementation process together with an automated test framework. The test cases included in the report will be component, system, user and acceptance tests. If additional requirements are added to the system during the Scrum process, new test cases will be created to test the desired functionality.

5.1.1 Non- Functional Requirements

The aren't many non- functional requirements that are essential in this project. It is a proof of concept type of project and the main goal is to prove that the solutions we come up with will get the job done. As a result, tests for common non- functional requirements like performance and security will not be included in the test cases.

One non- functional requirement worth writing tests for though is usability. The object of the project is to deliver a system that eases the workflow of specific users, and to achieve this we need to develop a system with a large degree of usability. Thats why usability tests will be included in the test cases, in the form of user testing(interviews).

5.1.2 Testing Process Timeline

Figure 5.1 outlines when the different tests will be performed during the Scrum process. Unit testing will be done throughout the development process in every sprint. Every part of the code implemented should have its own unit tests. Component testing will be done as separate components are finished, typically towards the end of the sprints. Acceptance testing with the customer will be performed in the sprint demo at the end of each sprint. At these demos it will be tested whether the agreed upon functionality for that sprint is actually implemented. System testing will be performed when we have an entire system to test, i.e. when we have implemented some parts of each component needed for the entire system to work. This will likely not be the case until the end of the second sprint. User testing will also be performed in the later sprints, when we have a complete system to present to the test users. Release testing will be performed at the end of the last sprint, to see if the agreed upon functionality for the entire project is indeed present in the system.

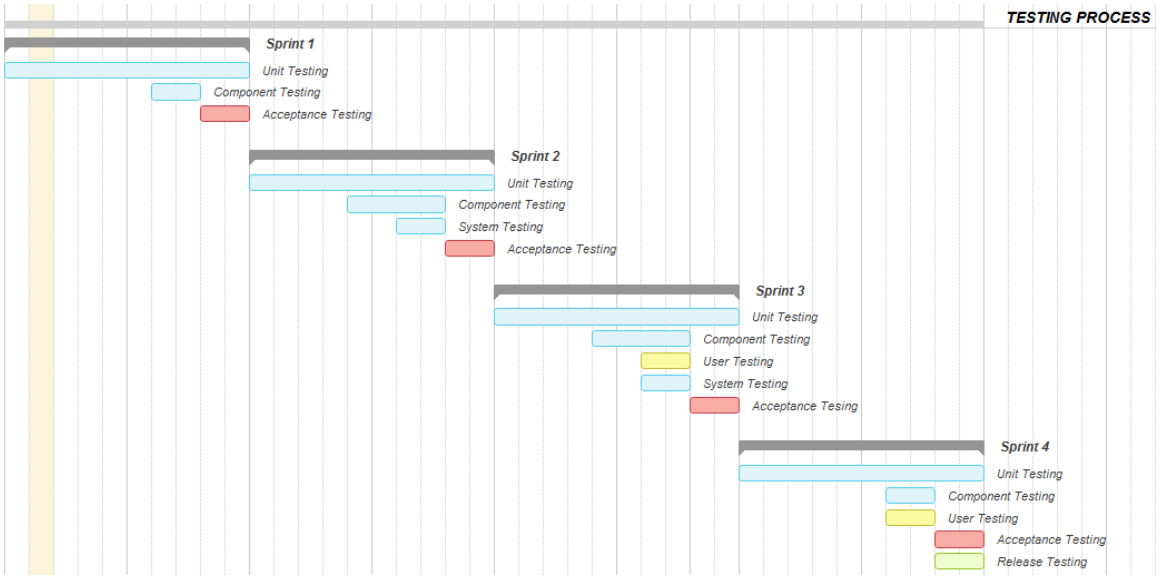

Figure 5.1: Testing Process Timeline

5.2 Templates

The templates stated in Table 5.1 and Table 5.2 will be used to create test cases and to record the results of them.

Table 5.1: Test Case Template

Item	Description
Description	Description of requirement
Tester	The person responsible for performing the specific test
Preconditions	The condition that has to be fulfilled before the execution of this test
Feature	The feature of the system that is to be tested
Execution steps	Steps to be executed in this test case
Expected result	The expected result of the test

Table 5.2: Test Report Template

Item	Description
Test ID	The ID of the given test
Description	Description of the requirement
Tester	The person executing the test
Date	The date the test was performed
Result	The result of the test, success or fail. Comment will be added if deemed needed

5.3 Responsibilities

The test manager is the one with overall responsibility of the quality of the test plan, and that it will be executed according to the schedule. The person writing test cases and recording test results is responsible for adhering to the templates included in this document. Each person is responsible for creating unit tests for their own code, and to run these with the automated test framework during development. This will be done to perform continuous testing of the system, and uncover if any new code break some of the previous tests.

We will mainly use someone different to perform the actual test cases than the person who wrote the specific code segment. This is done to achieve as little ownership of the code as possible on part of the tester. But as we are short on manpower and time we can't guarantee that this is always the case. The test cases will be performed towards the end of each sprint when the relevant functionality is implemented. The test will be performed at a time which allows the developers to fix any uncovered bugs in the system before the sprint is ended.

5.4 Test Criteria

A test is considered passed when it achieves the expected result. A test will be considered to have failed if the result of the test differs from the expected one. If we feel a pass/fail of a specific test needs an explanation this will be noted as a comment in the result.

Chapter 6

Architecture

Contents

The system is to be described in this chapter. This includes the main parts of the system, the architectural drivers, how they are connected together, and their collaboration. The architecture will be described through the 4+1 view model. The party members, or the stakeholders of the system, will have different concerns, so the 4+1 view model will help us describe the system for each of them, and make sure that all expectations are accounted for.

First section presents the Architectural Drivers 6.1 which are the main implications that can affect our system. Second section, Stakeholders 6.2, lists the stakeholders involved in the development of this system as well as their concerns. Third section, the Views 6.3, helps capturing the whole architecture for the different stakeholders to understand. Fourth section, Architectural Tactics 6.4, is used to show how the goals of the system will be reached. Fifth section, Architectural patterns 6.5, will show how the architectural tactics will be supported.

6.1 Architectural drivers

The architectural drivers defines the project and its scope.

- Improving the efficiency in the chosen domain - The traditional web application system is to be improved efficiency-wise, so power users will be able to use less time on more tasks. (non-functional requirement)
- Proving the concept in general - Our solution should be applicable to more general problems than those described by the chosen domain. (business constraint)
- Minimum console functionality - The system's console should have capabilities on par with a traditional web application: Creating objects, editing objects, deleting objects, working with lists of objects. (functional requirement)
- Added flexibility - The console should allow the user to perform tasks that are not possible with a web application of comparable complexity and development cost. (non-functional requirement)
- Console in a web context - The console should operate in a web browser environment. (technical constraint)
- Tolerate changes in requirements - We must use an approach that tolerates iterative changes to the demands from the customer. If new functionality is proposed and deemed likely to make the prototype more useful, it should be implemented asap. (business constraint)
- Delivery on time - The project must be finished on a fixed schedule. (business constraint)
- Academically sound process - The project should be executed using methods that satisfy the academical context. (business constraint)

6.2 Stakeholders

The stakeholders and their concerns when it comes to the system.

- Developer
  - Deadline, the system should be done by the deadline
  - Solve the problem and deliver a system the customer will be satisfied with.
  - Learn new technologies
  - Get experience with project management
- Customer
  - Gets a working system, which satisfies the customers wants
  - Get a usable report on the concept, which satisfies the customers wants
- End user
  - Improve efficiency
  - Easy to use after some training

6.3 Views

The 4+1 view model [10]. Here the views will be described, and how they will look in our architecture. This approach of designing makes the system easier to understand for both the developer, user and the customer, and gives the developers a good overview of what need to be done. Diagrams are based on Martin Fowler's UML [7]

6.3.1 Logical View

Describes the functionality in the system from the end users perspective. The end users will mainly be a power users, wanting to perform object editing tasks efficiently. This view will be described through class, communication and sequence diagram. We had some bigger changes from sprint two to three. The development of the logical view, and the road taken to reach this functionality representation can be observed through the sprint chapters.

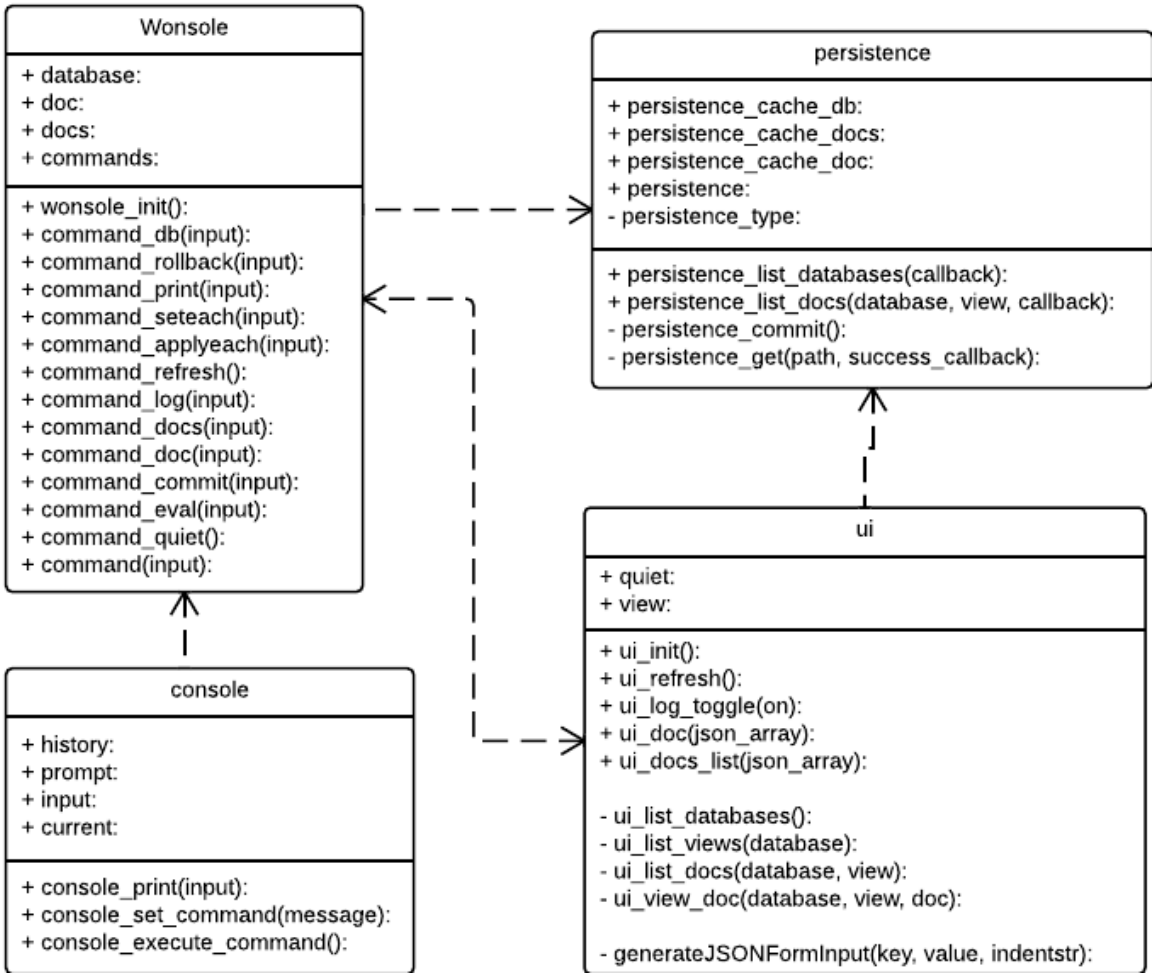

Figure 6.1: Client Class Diagram

Figure 6.1 The client class diagram gives an overview of the class structure of system, and how the classes collaborate. The Wonsole class possesses the commands used by the console to run different actions. Depending on the command, the console's input will lead to an action. This can change the objects in memory, fetch objects from the server, and more. The ui class represent these objects in a readable fashion, and actions to the objects can also here be done. The persistence class lets the user fetch and store objects to the server.

Figure 6.2 The server class diagram shows how the REST api is set up. Since the CouchDB in an integrated REST and server system we only have one class here, which can easily communicate with the server.

SEQUENCE DIAGRAM COMING

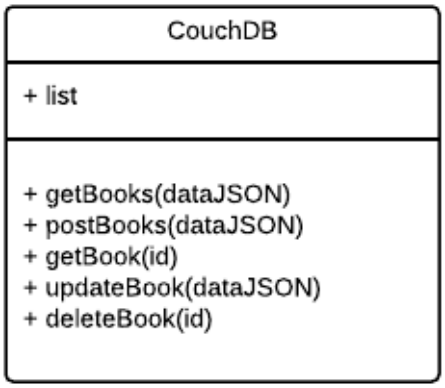

Figure 6.2: Client Class Diagram

6.3.2 Process View

Describes the dynamic aspect of the system, and explains how the different parts of the system will communicate at runtime. Can help pointing out bottlenecks and how to scale the system better. This is described with a activity diagram.

activity diagram COMING.

6.3.3 Physical View

Describes the system from the system engineer's perspective. And explains the physical connections between the software components. Can help to show how changes affect different physical parts of the system, and their impact. Described through a deployment diagram.

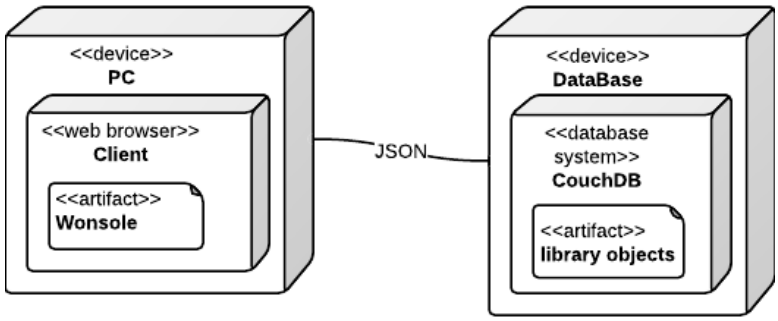

Figure 6.3: Deployment Diagram

Figure 6.3 The structure of the system is divided into two different parts. Here we can see the client can be run on its own computer, and will communicate with the server through JSON. The server is of the type CouchDB and holds the objects for the client to retrieve when wanted.

6.3.4 Development View

Describes the system from the programmer's perspective. This will be described through how the different component parts are separated. Can help dividing the system into smaller parts, which can easily be developed. Component and package diagrams will show this.

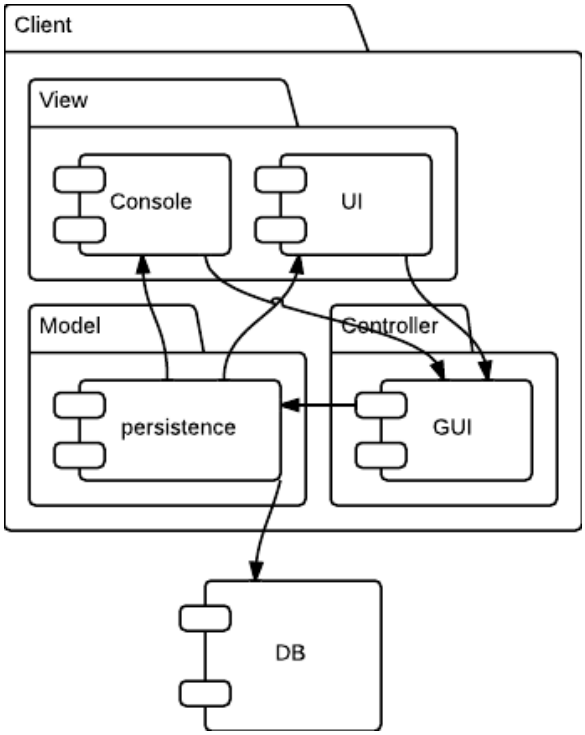

Figure 6.4: Component Diagram

Figure 6.4 These components form a three layered structure. Where the front end (the client) is the upper layer. This layer communicates with the server through the second layer.

6.3.5 Scenarios

Is the + 1 in the 4 + 1 view model. It describes the system through a set of use cases/scenarios. These use cases/scenarios describes how the objects and processes work together. The scenarios can be found in Chapter 4, Requirements.

6.4 Architectural tactics

The architectural tactics are used to describe how different qualities of the system are achieved.

6.4.1 Modifiability

Anticipate changes

Probable changes should be anticipated, and accounted for, so extending the system with new functionality should be possible. This is important since we are dealing with a prototype/proof of concept system. So the customer might change the direction we are going in through out the project, this will lead to changes, and the architecture should be able to bend after these new inputs.

This can be handled with a well defined functionality map, so that the expected system functionalities are accounted for when the system is constructed.

Ripple-effect prevention

It will also be important to make sure that too much does not depend on too much, so it will be possible to avoid potential ripple effects. This happens when one change can expand into the need of changing everything.

This can be prevented by encapsulating information, or constructing interfaces and maintain existing interfaces.

6.5 Architectural patterns

The patterns can help solve different kinds of problems with known solutions used before on similar problems.

6.5.1 Multi-tier

The architecture of the system will be of the multi-tier kind. This means that the presentation, application processing and data management functions will be separated logically. The application (console) will translate the task of the end user, and will be sent to the server through the logical tier.

The console - Translate the task of the end user. The logical tier - Will receive the translated task from the console, and fetch information from the data storage specified by the console, and set it back to the user. The data storage - The last tier, and it will contain the information from the library.

Front end

The frontend is implemented using JavaScript and HTML. This is the presentation tier, and will let the user interact with the objects desired. The JavaScript performs operation and communicates with the server through AJAX calls.

Middle Layer

The middle layer (Logical tier) is responsible for communication between the server and the client. Since the database is schemeless, most objects are allowed to be sent to the backend, so this layer does not do big logical operations.

Backend

This layer (the data tier) stores the information in a database. This information can be sent back through the middle layer to the client on request from the client.

6.5.2 Model-View-Controller

MVC lets us divide the system into three main parts, the model, the view and the controller. The model is the data structure within the application. The controller is the interface provided for the client to do operations. The view will render the model to the client. The client gui and console communication. Since the action made in the console should be reflected in the gui and vice versa, is it natural to use a

model-view-controller pattern. The information in the model will then be separated from the view (the display) and the controller which performs an update on the model.

6.5.3 Rationale

Our architecture is greatly affected by the fact that this project is a prototype project, and that we need to be flexible and able adapt to changes from the outside. This means that we should expect possible changes, and prepare for that. Therefore the modifiability is the tactic we focus on.

MVC was a natural choice for model distribution, so the changes done in the console, could be reflected onto the ui and vice versa.

Chapter 7

Sprint 1

Contents

This chapter will outline the work we did in the first sprint. It explains in detail how we planned the sprint, including which user stories we chose and the architecture we employed to implement these. Details on the implementation on each user story is also included, as well documentation on the testing process. Finally our evaluation of the first sprint is presented.

7.1 Planning

We started the sprint with a sprint planning meeting September 24th. The plan for this sprint was to implement a basic implementation of the system, so that we would have something to show the customer at the end of the sprint. We chose user stories from the backlog that would enable us to this, a client and server offering only the most basic operations.

7.1.1 Duration

This sprint started on September 24th and lasted for two weeks. A customer demo was held at October 4th to show of what we have achieved during the sprint and to ensure that the customer agreed with the implementation.

7.1.2 Sprint Goal

The goal for the first sprint was to get a basic version of the library application working. This included creating a basic GUI with both the graphical web- application and the console side by side, creating a server capable of storing objects and establish communication between the server and client through a basic REST api. The user should be able to use both the web- application and the console to add a new book to the system and to list the books currently in the system. In addition we decided to implement the real- time messaging from the server to the clients to verify that the solution we chose in the pre-study would be up to the task.

7.1.3 Product Backlog

For the first sprint, we used the user stories from the initial requirements as our product backlog. The full product backlog for Sprint 1 can be found in section E.1 in the appendix.

7.1.4 Sprint Backlog

The user stories we chose to include in the sprint backlog with their estimated workload is listed in Table 7.1. We estimated that we have around 160 hours each sprint for working with the actual user stories. The remaining 40 hours would be used for meetings, demonstrations and project management.

Following is some points to discuss the deviations in estimated effort and actual time used.

- We used more time on implementation than we planned for. We are still getting used to the fact that we have to document everything, usually we only expend effort towards the actual coding process. Also we had to spend some time setting up technologies on a server and get them running before we started implementing actual functionality.
- We used less time on design than we planned. Still getting used to designing everything before we code, up front. We plan to do this better the next sprint
- We used quite a lot of time less on testing than we estimated. We anticipated that we had to use a considerable time to correct bugs and errors during the testing, but all the test cases passed on the first attempt. We anticipated that we had to spend more time on getting all the technologies we had chosen to play together nicely through extensive testing, but it turned out that they were a great fit, and that it wasn't too much work to get them working together in the way we intended.
- Once we had implemented the functionality of adding new books, the time used to implement D2 and D5 was considerably reduced, as we could resuse much of the code we had created for D1.

- The PubNub real- time messaging turned out to be easier to implement with Node.js than we expected.
- We ended up spending more time on the Scrum planning meeting than we planned for. In addition we didn't work as many hours that we planned for. As a result we had less time available to implement the user stories. Luckily the it turned out that this time was sufficient to finish all the user stories we planned to implement.

Table 7.1: Sprint 1 Backlog

User Story	Short Description	Hours Est.	Hours Act.
A1	**Store objects in database**	**30**	**40**
	Design	8	6
	Implementation	15	25
	Testing	4	4
	Documentation	3	5
A2	**Send real- time messages**	**20**	**13**
	Design	4	2
	Implementation	10	7
	Testing	4	3
	Documentation	2	1
A3	**Access to domain specific object**	**25**	**22**
	Design	10	5
	Implementation	10	10
	Testing	3	3
	Documentation	2	4
G3	**Graphical web- application and console**	**35**	**38**
	Design	12	10
	Implementation	15	21
	Testing	4	2
	Documentation	4	5
D1	**Add a new book**	**15**	**18**
	Design	3	2
	Implementation	8	12
	Testing	2	2
	Documentation	2	2
D2	**Delete a book**	**15**	**7**
	Design	3	1
	Implementation	8	4
	Testing	2	1
	Documentation	2	1
D5	**List all the books in the system**	**20**	**9**
	Design	5	2
	Implementation	10	5
	Testing	3	1
	Documentation	2	1
	Total:	**160**	**147**

Table 7.2: Sprint 1 Workload

Task	Hours Est.	Hours Act.
Design	45	28
Implementation	76	84
Testing	22	16
Documentation	17	19
Total	**160**	**147**

7.2 Architecture

This section will handle the architecture we used for this sprint. The architecture will be described through class diagram and component diagram.

7.2.1 4+1 view model

The 4+1 view model [10]. Here the views will be described as they are in sprint 1, and how they will look in our architecture.

Logical View

Describes the functionality in the system from the end users perspective.The end users will mainly be power users, wanting to perform object editing tasks efficiently. This view will be described through class, communication and sequence diagram.

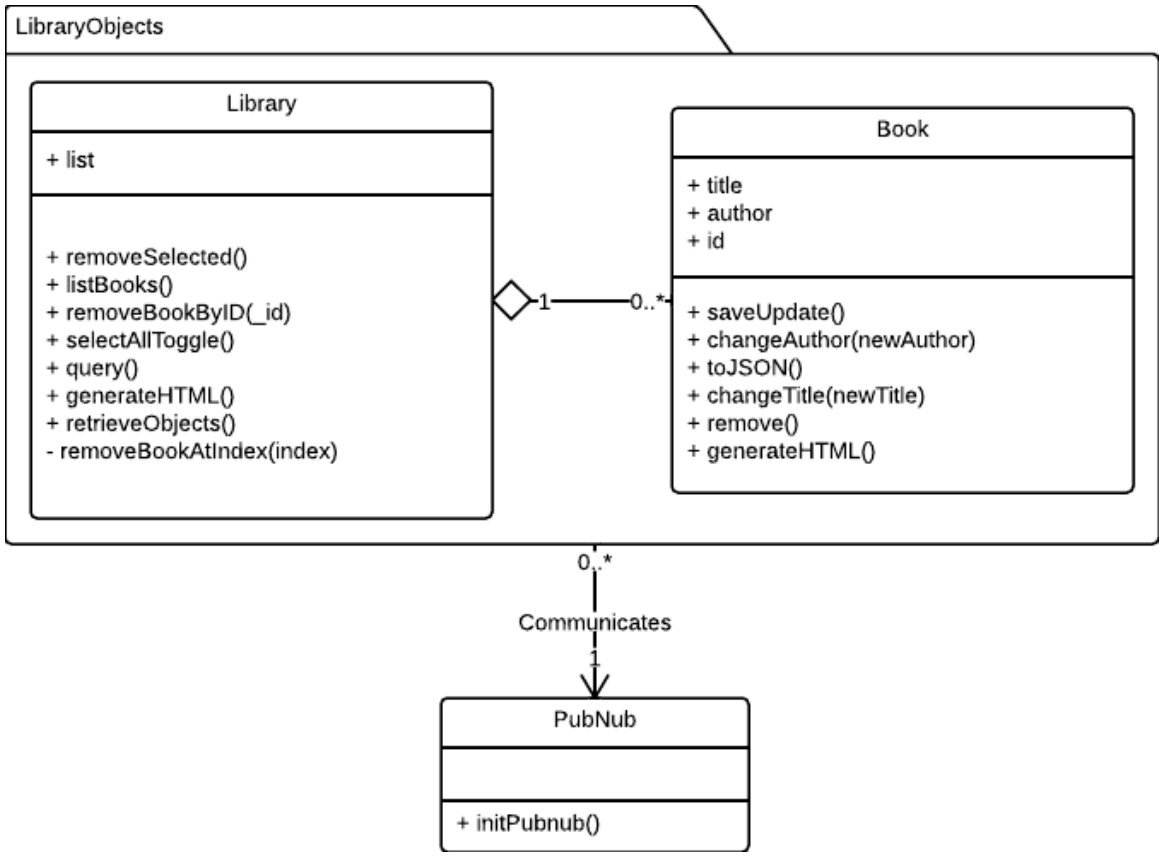

Figure 7.1: Client Class Diagram

Figure 7.1 The client class diagram gives an overview of the class structure of system, and how they collaborate. We can see that the client consists of two separate views, a GUI and a Shell, which is split by a splitpane so the user can se bouth a GUI interface and the commandline interface. These two views can make changes to the library objects, and get reflected back to the other view. The library objects consists of a library filed with books. The changes done to a book is delivered to other clients and the server through PubNub.

Figure 7.2 The server class diagram shows how the REST api is set up, and the communication with the pubnub and db where the library information is stored.

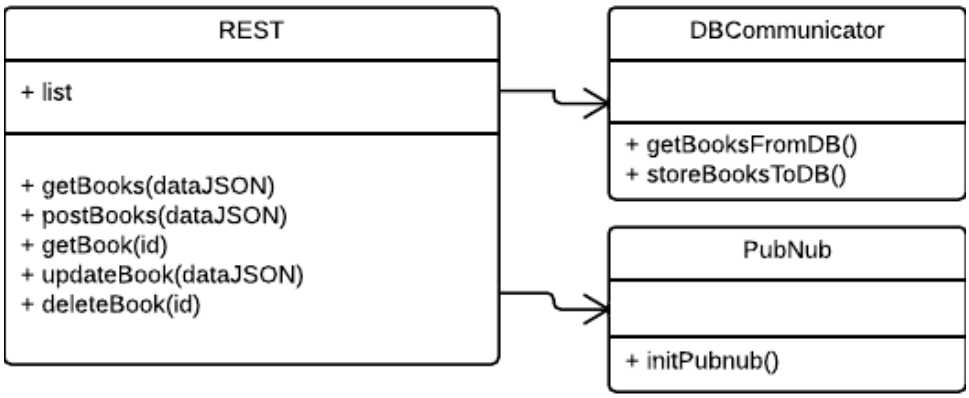

Figure 7.2: Client Class Diagram

Physical View

Describes the system from the system engineer's perspective. And explains the physical connections between the software components. Described through a deployment diagram.

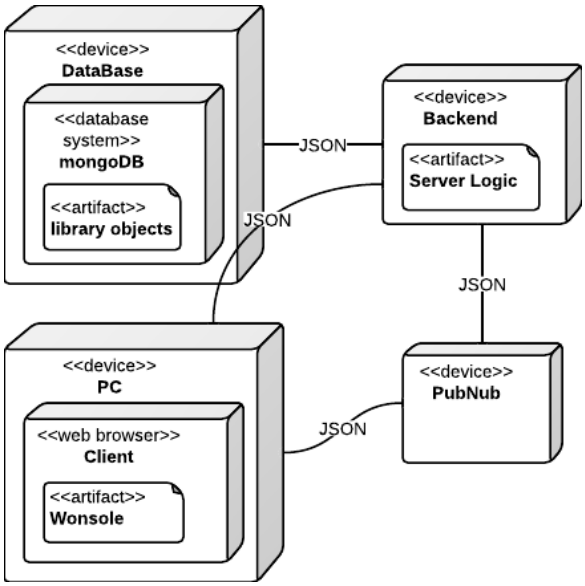

Figure 7.3: Deployment Diagram

Figure 7.3 The structure of the four different parts of the system.

Development View

Describes the system from the programmer's perspective. This will be described through how the different component parts are separated. Component and package diagrams will show this.

Figure 7.4 These components form a three layered structure, and communicate with each other through the neighboring layer.

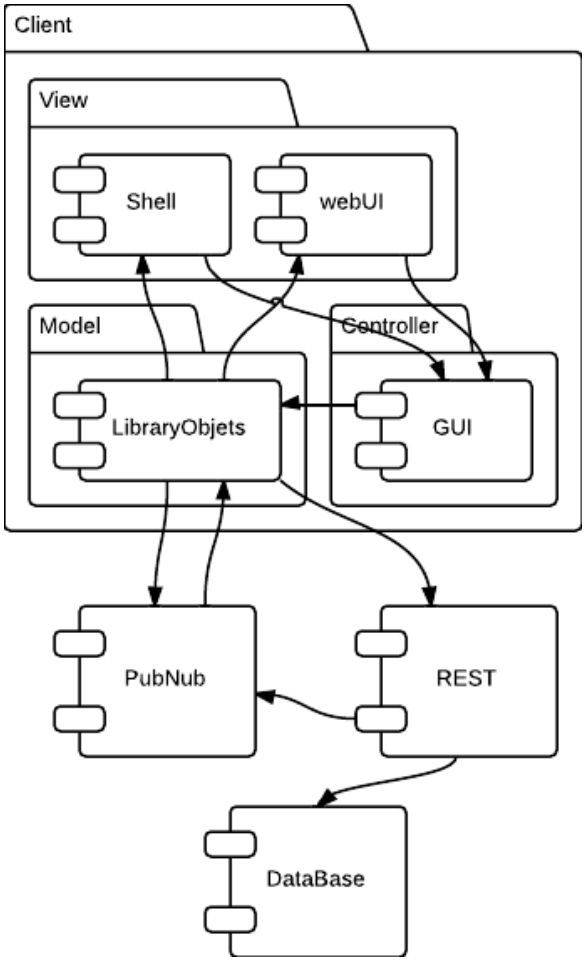

Figure 7.4: Component Diagram

7.3 Implementation

7.3.1 RESTful API

We decided to expose the contents of the database on the central server to the clients through a RESTful API that is served over HTTP. REST is an abbreviation for REpresentational State Transfer, and it basically allows you to get information and perform action equipped only with URLs and standard HTTP methods like GET and POST. The point of REST is having one standard interface for any service. Instead of exposing an interface which has methods, you only expose 4 methods; create, update, read and delete. You use the URL to describe what object you are performing the action on.

A detailed explanation of the RESTful is included in the appendices. In addition example code with jQuery(JavaScript) will be supplied.

7.3.2 Client-side application

See Figure 7.5.

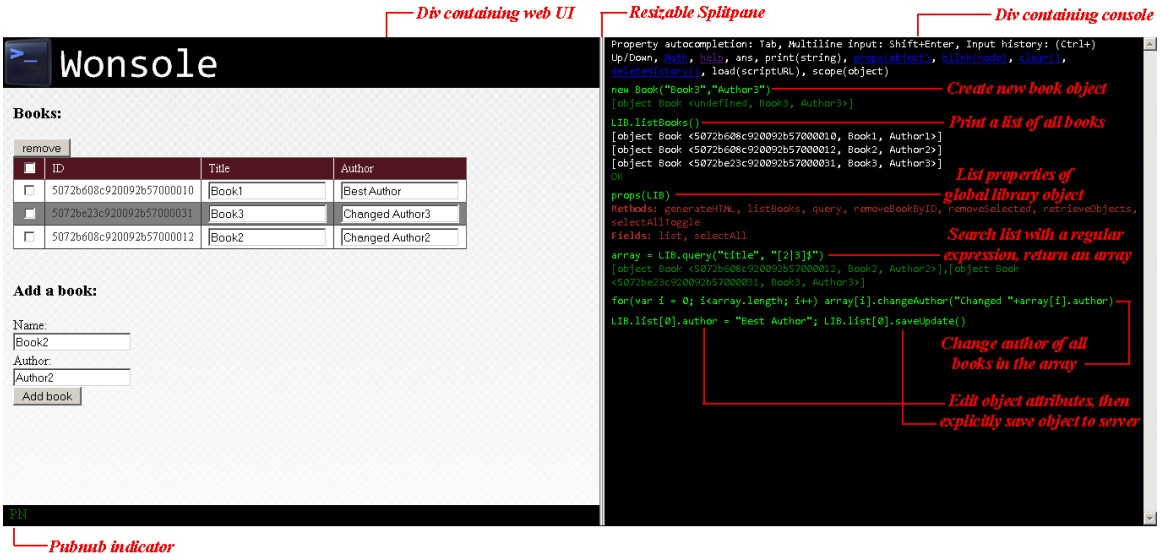

Figure 7.5: Wonsole, Sprint 1

Web UI

The web user interface is a standard HTML/CSS web site. Lists of objects and their information are generated through JavaScript, and UI elements are associated with JavaScript methods to perform changes to the underlying objects.

JavaScript Console

For the console part of the UI, we decided to build off of an existing implementation. We used JavaScript Shell 1.4, which is a command-line interface for JavaScript and DOM. The console will be rather heavily modified to suit the needs of our project. The console current supports a number of useful features, such as listing the variables and methods of an object, suggestions, and a command history. In later sprints, we intend to implement more useful autocompletion, easier navigation of lists of objects as well as better visualization of objects and better integration with the web UI.

The JavaScript Shell is GPL/LGPL/MPL tri-licensed. [1]

In the console, it is possible to execute arbitrary JavaScript code. The intended usage is to perform operations on the system's objects only. To implement this particular solution, it must be possible to access all the relevant data through JavaScript objects, and the objects should have appropriate methods to manipulate and store the data. Preferably, the it should also be possible to alter the contents of the web UI by altering the data in the JavaScript objects.

JavaScript Objects

We are providing a global object containing a list of books, called LIB, which is a monolith instance of the type Library. The full list of books is retrieved and kept updated from the server. For systems with large data sets, it would be possible to implement caching and more selective loading of server data, but this is not directly interesting to our project.

The aforementioned list contains objects of the type Book. These objects contain information about the book, as well as methods to manipulate their data. Using these methods will also lead to the database

[1] `http://www.squarefree.com/shell/`

and web UI to be updated. Detailed documentation of these objects can be found in section D.1 in the appendix.

Pubnub

Pubnub, previously described in the prestudy, was used to transmit a message to all connected clients when data has been changed. For the first sprint, the client dispatches the message to all other clients when it changes a piece of data, and any client must reload the data from the server when receiving such a message. We realize this is a less-than-optimal and not very tidy solution, so it is likely to be improved in later sprints if it proves to be helpful for the project goal.

Simulated synchronous behaviour

For the first sprint, we decided to block user input while the system is communicating with the server, as a quick and safe way to ensure consistency.

jquery.blockUI is a Javascript library we used to achieve this. The intended usage is to simulate synchronous behaviour with AJAX(when necessary) without locking the browser, which is generally unwanted behaviour. [2]

Persistent command history

The console features a command history that is persistent across page changes, for convenience. PersistJS was used to achieve this.

PersistJS is a JavaScript library which handles client-side storage. It attempts to use the most efficient solution available in the browser, and abstracts away the limitations of certain solutions(such as the size limit on cookies). PersistJS is under a MIT license. [3]

Splitpane

The splitpane is used to let the user customize how much of the window to use for the console and the Web UI. This has been made persistent across page changes. The splitpane was created by following a drag and drop tutorial by Luke Breuer. [4]

7.4 Testing

This section will present the tests performed during the first sprint and their result.

7.4.1 Test Results

We performed a total of 10 test cases during this sprint; TID01-10. The results are listed in Table 7.3. The test cases themselves can be found in appendix C.1.

[2] `http://malsup.com/jquery/block/#overview`
[3] `http://pablotron.org/software/persist-js/`
[4] `http://luke.breuer.com/tutorial/javascript-drag-and-drop-tutorial.aspx`

Table 7.3: Sprint 1 Test Results

Item	Description
TestID	**TID01**
Description	Storing objects in a database on the central server
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID02**
Description	Retrieving objects from the database on the central server
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID03**
Description	Sending real- time messages from server to client
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID04**
Description	Alerting clients that there has been added a book to the central database on the server
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID05**
Description	Verifying that domain specific objects are available through the console
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID06**
Description	Verifying that there is a console and a graphical interface present on each page
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID07**
Description	Adding a new book to the system with the graphical web- application
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID08**
Description	Adding a new book to the system with the console
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID09**
Description	Listing all the books currently in the system using the graphical web- application
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success
TestID	**TID10**
Description	Listing all the books currently in the system using the console
Tester	Øystein Heimark
Date	04/10 - 2012
Result	Success

7.4.2 Test Evaluation

All our tests passed on the first attempt, without any complications to speak of. The likely cause of this is that we chose to implement a basic protoype for this sprint, with basic functionality all around. There weren't many comlicated components that had to be implemented, and the interfaces between the components were also pretty straight forward. The fact that we performed all of the tests towards the end of the sprint, when all the different components were completed, may also have been a contributing factor. In doing this we avoided getting errors and failed tests because some components were yet to be implemented.

7.5 Customer Feedback

This sections covers the feedback we got from the customer, both before and after the sprint.

In the beginning of the sprint the customer did not have any objections to the user stories we had chosen to implement, and he liked the goal we had set for the sprint.

After the Scrum demonstration the customer was generally impressed with the result we delivered after the first sprint and he liked the direction we were going in. He had however some feedback on how we presented the system to him. We should try to not think of him as a group member, but as an external person with no prior knowlegde about the specifics about the implementation. We should try to sell him the solutions in a better way.

The customer had a lot of suggestions for the second sprint. He would like us to focus on the console, adding functionality that will be useful for the end user. The domain specific parts of the system is not very interesting to the customer, whats important is to make the console useable. The customer would also like us to split the user stories into smaller sub- tasks, and create a detailed Work Breakdown Structure for the next sprint. We should send this to the customer to get some feedback on the estimates. He also added that we could invite him to short demos during the sprint, to get feedback on specific solutions. On a final note he suggested that we should try to demonstrate the product to our peers, and get some feedback from them.

7.6 Evaluation

This section contains the evaluation of the first sprint, and what we plan to improve for the second sprint. The evaluation was performed during an internal meeting with all group members present.

7.6.1 Review

All in all we were happy with our progress in the first sprint, and the fact that we were able to deliver a functioning basic version of the system. This was the goal we had set at the beginning of this sprint and we were able to finish all the user stories in the sprint backlog. We feel this was in large part due to the fact that we did a good job in dividing tasks among the team members. We were also happy with fact that the communication both within the group and with the customer was good throughout the sprint. The customer was satisfied with the product we were able to deliver, and liked the direction the project was taking. Also we recieved valuable feedback from the customer after the sprint demonstration, with many suggestions for improvements.

Although we were happy with the end result, we felt there was a lot of areas in which we could improve. One of them was the execution of the Scrum process. None of the team members had any previous experince with doing Scrum in an actual project, although we were all familiar with the basic concepts. We failed to properly see the importance of the daily meetings and the organization of these

meetings was not good enough. Some days they were not performed at all, and we feel that this was the main reason we ended up doing some double work. In addition none of the team members had any experience in writing user stories before. As a result some of the user stories were not concise enough and a lot of them were overlapping. This actually resulted in us completing a more user stories than we planned for, including **G2**, **G7** and **G8**, because some of the functionality we implemented was covered by multiple user stories. Although this was a positive surprise we felt we needed to put some effort into rewrite some of the user stories and do a better job on the new ones we created.

We planned to do a lot of design up front before we started this sprint. However we did not succeed in doing this, most of the time was used on implementation. This is a project where the amount of documentation is expected to be extensive, and we were at the time still getting used to that. This also hurt us when it was time to document our work, as we didn't produce much written material during the implementation. As a result we ended up with a lot of unfinished work on the report that had to be done in the second sprint. Also our time tracking routines during the sprint was not good enough, and this led to a lot of extra work when we tried to document the time spent on the different user stories.

Positive Experiences

- Delivered a functional system for the demo.
- Customer was happy with the product and our progress so far.
- We finished all the user stories
- Good work distribution amongst the team members
- Able to include the customer in the planning of the sprint
- Received valuable feedback from customer.

Negative Experiences

- Not enough design up front
- Too little documentation during implementation
- Poor use of time tracking system
- Problems with the burndown chart
- Inexperience in using the scrum process.

7.6.2 Planned Responses

These are the actions we have planned for the next sprint to improve our work process.

Time tracking

We will be using our time tracking system in Redmine more actively, and enforce stricter rules on the team members for logging their work. This will include logging hours before certain timelimits each day and include more detailed description for the work that has been done. This way, documenting the hours spent on each tasks will become a lot easier.

Design Up Front

We will try to design our solutions on every user story prior to the implementation. This way we will hopefully not fall for the temptation of starting on implementation of features without a design to go on. We will also try to document more during the sprint to avoid a lot of work on the report left to do towards the end of the sprint.

Scrum

We will try to improve the overall quality of the Scrum process. The most important task will be to improve the organization of the daily meetings a lot better, and ensure that they happen every day during the sprint. This will improve the coordination between the team members. Also we will fix the burndown chart plugin in Redmine to be able to use this as a guideline during the sprint. We will also put some effort into improving the quality of the user stories and in a larger degree divide these into smaller tasks. This will able us to estimate the workload of each user story more precisly and more easily assign tasks to different persons. Also we will avoid creating user stories with overlapping functionality.

Chapter 8

Sprint 2

Contents

This chapter will outline the work we did in the second sprint. It explains in detail how we planned the sprint, including which user stories we chose and the architecture we employed to implement these. Details on the implementation on each user story is also included, as well documentation on the testing process. Finally our evaluation of the second sprint is presented.

8.1 Planning

8.1.1 Duration

This sprint started on October 8th and lasted for two weeks. A customer demo was held at October 18th to show of what we have achieved during the sprint and to ensure that the customer agreed with the solutions and the direction we were going in.

8.1.2 Sprint Goal

The goal of the second sprint was to increase the functionality of the console. We tried to include user stories that would add functionality that was easy to use and valuable for the end user. The user stories concerned with domain specifics of the system, like implementing support for adding users to the library, will be left untouched in this sprint.

8.1.3 Product Backlog

Between Sprint 1 and Sprint 2, the customer expressed a desire to improve the usability of the application, by adding features such as auto-completion, tabbing through objects, and better integration between the console and GUI. It was not desired that we implement any more domain-specific functionality. In addition, we decided to alter the backend to support arbitrary objects so that it would not be necessary to change the backend code when creating new types of objects in the application.

The user stories completed in Sprint 1 were removed, and the following user stories were added:

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.

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.

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

The full product backlog for sprint 2 can be found in section E.2 in the appendix.

8.1.4 Sprint Backlog

The user stories we chose to include in the sprint backlog with their estimated workload is listed in Table 8.1. As of this sprint we tried to split each user story in the sprint backlog into smaller sub- tasks that could be distributed easily amonst the group members. Each estimate for the subtasks includes design and implementation of the solution. Testing includes writing unit tests as we code, write test cases and to perform them. Documentation includes all the work necessary to document our work on that user story in the report. For this sprint we had 110 hours available for the sprint, exluding work on the report, project management, sprint planning and evaluation. Its a bit less than normal as we decided to put in some extra work on the pre- delivery of the report on October 14th.

The decision to make smaller sub- tasks turned out to be succesful, as we were able to estimate the workload more precisly than we did in the last sprint. The tasks were much more detailed as well, instead of general tasks such as "Design" and "Implementation". We experienced that it was easier to estimate the tasks when they represented something concise, and to achieve this they needed to be small enough.

There is still room for improvement, as the estimates on some of the sub- tasks were somewhat far away from the actual time used. The tasks that we felt were the most difficult to estimate were the bigger ones. We generally spent less time than planned on these, and we feel that the reason for this was that we failed to break these down to small enough packages. Our general feeling is that the smaller and more concise the tasks are, the better we are able to estimate them. For the next sprint we should make an effort to do more of the same, and try to make the tasks as small and concise as possible and avoid big and general ones.

We used less time on the user stories than we planned for this sprint, and a big part of this was due to the pre- delivery of the report. We failed to properly estimate the workload of this delivery, and as a result we had less time available for the user stories. Luckily we were able to finish them using with the time available, mostly because some of the tasks were easier to accomplish than we originally thought. However, this may not be the case in the following sprints. Thats why we feel its important to get the estimates even more precise, so we know exactly how much effort its going to take to achieve the goals we set for the sprints.

Table 8.1: Sprint 2 Backlog

		Hours	
User Story	Short Description	Est.	Act.
A4	**Update to schemaless database**	**10**	**11**
	Change the server implemenation	4	5
	Update REST API calls	2	1.5
	Testing	2	2.5
	Documentation	2	2
G4b	**Reflect actions from the GUI in the console**	**11**	**13.5**
	Print the commands in the console	6	8
	Testing	3	3.5
	Documentation	2	2
G9	**Autocompletion of commands**	**30**	**22.5**
	Add methods to list commands	4	2
	Create popup menu	10	5
	Update elements in menu	4	3
	Make selection of commands possible	3	3.5
	Autocomplete on selection	1	1
	Testing	4	5
	Documentation	4	3
G10	**Indicate the selected objects in GUI**	**26**	**17**
	Design soulution for highlighting elements	2	1
	Create method for highlighting	2	1.5
	Create record for selected objects	4	2
	Respond to changes in selection	8	4
	Highlight newly created objects	2	2
	Testing	4	3.5
	Documentation	4	3
G11	**Cycle through the current selection**	**25**	**18**
	Logic to keep track of selection	5	3.5
	Update this list when selection changes	4	3
	Extract objects when cycling through	7	4
	Update the GUI when selection changes	2	2
	Testing	4	3.5
	Documentation	3	2
G12	**Show details on click**	**8**	**7.5**
	Make ID clickable	1	1
	Update relevant section on click	2	1.5
	Able the user to make changes	3	1.5
	Testing	1	2.5
	Documentation	1	1
	Total:	**110**	**89.5**

8.2 Architecture

The user stories picked out for sprint two did not have much of an impact on the architecture constructed earlier. We therefore kept using the same architecture as in sprint 1.

8.3 Implementation

8.3.1 Autocomplete

The autocomplete is made of a jQuery suggestion menu, a modified JavaScript Shell 1.4 and the available objects in the running system. The suggestion menu is attached to the console textarea in Shell. Shell already possessed the ability to generate suggestions based on the user's input in the console. This information is basically reflected onto the attached suggestion menu, which displays these. Since the system is to be efficient for the selected domain, the available methods and attributes have been cut down, so only the essentials remain, and the user doesn't have to look through unnecessary methods when working.

8.3.2 Object highlighting and detailed editing

It was made possible to click the ID field of book objects in order to highlight their row in the list. This action also filled out the section for adding new books with the information about the given book object. Furthermore, the section for adding new books was enhanced to allow editing of the existing object.

8.3.3 Cycling through objects

Logic was implemented in the console to detect the presence of an array in the user input, and incrementing the index of this array upon pressing the Tab key. The object at the indicated index was then highlighted for editing in the GUI. In order to detect arrays in the textual user input string, the following approach was used:

- If the string contains the characters '(', ')' or '=', terminate. The code later makes use of the eval() function to determine whether the array is valid, and it was deemed unsafe to risk performing function calls(indicated by the presence of parentheses) or assignment statements(indicated by the presence of an equals sign).
- Strip complete or partial occurrences of [index] at the end of the string, where 'index' would be an integer literal. Determine the value of 'index' if it is present, set it to 0 otherwise.
- Evaluate the resulting string and store it in a variable, then test the type of the variable. If the variable is not an array, or if it is an array containing objects that are not book objects, terminate.
- If the book object at the previously determined index in the array is highlighted in the GUI, increment the index by one, wrapping around if necessary.
- Add the [index] construct to the end of the string in the console, and highlight the book object at the current index in the array for editing in the GUI.

8.3.4 Reflection of GUI actions in the console

The GUI elements were altered so that they sent strings containing JavaScript code to the console for execution, and subsequently displaying the results.

In itself, the console already contained support for programmatically executing code and displaying the result. However, to make this feasible, some specific implementations had to be altered so as to not use references that were not accessible from the global scope. For instance, the GUI elements for editing book objects were originally implemented by having the book objects create DOM elements with callbacks directly to the book object which created them. This was changed so that the GUI element instead used an index in the global array of books(which was accessible from the console) in order to obtain a reference to the correct book object.

8.4 Testing

This section will present the tests performed during the second sprint and their result.

8.4.1 Test Results

We performed a total of 9 test cases during this sprint; TID11-19. The results are listed in Table 8.2. The test cases themselves can be found in appendix C.2.

8.4.2 Test Evaluation

Some of our test failed this time. Most of them was due to minor bugs in the system which was easily corrected at once. The remaining failures was due to some missing functionality which was defined by the user stories and where yet to be implemented at the time the testing was performed. However we were able to add this functionality and pass all the tests before the beginning of the third sprint.

Table 8.2: Sprint 2 Test Results

Item	Description
TestID	**TID11**
Description	Storing objects on the server without a schema
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Success
TestID	**TID12**
Description	Printing commands in the console
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Success
TestID	**TID13**
Description	Showing the popup menu
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Failure. The popup menu was displayed, but some of the attributes and methods were missing from the menu
TestID	**TID14**
Description	Selecting elements from the popup menu
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Success
TestID	**TID15**
Description	Highlighting selected objects
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Failure. Objects are not automaticly highlighted when they selected from the console. It is however possible to highlight an object from the console using a method call.
TestID	**TID16**
Description	Highlighting a group of objects
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Success
TestID	**TID17**
Description	Cycling through current selection
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Failure. It is not possible to highlight multiple objects.
TestID	**TID18**
Description	Highlighting while cycling through objects
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Success
TestID	**TID19**
Description	Update separate section on clicking a book
Tester	Øystein Heimark
Date	19/10 - 2012
Result	Failure. It is not possible to edit the title of the book in the separate section

8.5 Customer Feedback

This sections covers the feedback we got from the customer, both before and after the sprint.

At the beginning of the sprint the customer had a lot of suggestions and opinions on what we should implement in this sprint, and provided us some user stories that he wanted us to implement. He suggested that our focus for this sprint should be on improving the functionality of the console, something the group also wanted to do.

At the end of this sprint the customer was happy with the work we presented, but at the same time felt that the system was still missing something. The workflow was not as fluid as it should be, the console needed a x- factor that appealed to the user an made the console more prominent. The focus should be on adding functionality that adds value to the console. At the moment the console and its features are very programmatic. The customer told us that he would meet with other representatives from the company to talk about the direction they wanted the project to take from this point on, and to maybe find the x- factor we were looking for. We would get this feedback for the start of the third sprint.The customer also requested that we started to test our product on our peers, as feedback from users would be extremly valuable at this point of the project. He suggested that we should devote some time in the next sprint to make and perform the user testing, something the group also felt would be a good idea.

8.6 Evaluation

This section contains the evaluation of the second sprint, and what we plan to improve for the third sprint. The evaluation was performed during an internal meeting with all group members present.

8.6.1 Review

Again we were happy with the product we were able to deliver in the sprint. All the user stories in the sprint backlog were completed, and we feel like we reached the goal of extending the functionality of the console. and think that the features we implemented added great value to the console. Again the customer was happy with the end result, although he felt something was missing. This time the customer was even more included in the planning, and helped us in creating some of the user stories. This allowed them to have a greater influnce on what was to be included in this sprint.

We feel like we succeded in addressing some of the issues we faced in the previous sprint regarding our work process. The responses we created after the first sprint turned out to work quite well. Our time logging routines improved significantly, and this eased the work of documenting the workload of each user story. It also helped the team leader to oversee the progress of the sprint and to monitor the work of each of the team members.

As mentioned earlier we succeded in creating a more detailed work breakdown structure, and were able to break each user story into smaller tasks. This helped a lot when we estimated the workload of each user story, and made it easier for multiple team members to work on the same user story at once. The distribution of workload was even better than in the first sprint, and a big reason that we were able to finish all the user stories in time.

The testing proved to be more useful this time, as we uncovered multiple errors in the system and in some cases some missing functionality in some of the user stories. Although this was great and it showed that the testing surved a purpose, the timing of the testing could be improved. In this sprint we performed the test cases at the end of the sprint, and this resulted in errors discovered after the cusotmer demo. Ideally they should have been discovered and corrected before the demonstration to the customer.

Before the sprint we set out to improve the Scrum process, but we only partly succeded. We were

able to organize the daily meetings, but we think there is still some improvements to be made here. We also had some technical issues during the demo which made it difficult for us to perform a proper demonstration of the features we had implemented. The technical aspect of the demo should have been prepared better, and we should always have a backup solution in case we experince problems during the demonstration.

A big downside to this sprint was the fact that we failed to estimate the workload neccessary for the pre- delivery. This ate up a lot more time than expected and as a result we were left with considerably less time for the implementation than we planned for. We were able to adjust and complete all the user stories, but we were lucky considering some of them turned out to be easier than expected.

Positive Experiences

- Much improved time tracking
- Better breakdown of the user stories
- Customer more included in the planning
- Testing proved useful

Negative Experiences

- Can still improve on some parts of the Scrum process
- Failed to estimate the workload of the pre- delivery
- Technical issues during demostration
- Testing performed too late

8.6.2 Planned Responses

These are the actions we have planned for the next sprint to improve our work process.

Testing

In this sprint we experienced how important the testing process can be in regards to finding errors and bugs we would otherwise fail to discover. However we performed the test cases too late to be able to correct them before the customer demonstration. Thats why we in the next sprint will perform the tests earlier, we will then manage to fix the issues in time.

Demo

For the next sprint we should properly prepare for the demo, and make it easier for the customer to follow the workflow. Also we should have a backup solution for how we are going to present our product if we experince technical difficulties again.

Scrum

Put even more effort into creating concise user stories and to break these into small tasks which can easily be distributed among the team members. We feel like this was something which contributed to the success of this sprint, and we would like to improve this even further in the next sprint. Also we aim to put more effort into the daily Scrum meetings and utilize these properly.

Chapter 9

Sprint 3

Contents

This chapter will outline the work we did in the third sprint. It explains in detail how we planned the sprint, including which user stories we chose and the architecture we employed to implement these. Details on the implementation on each user story is also included, as well documentation on the testing process. Finally our evaluation of the third sprint is presented.

9.1 Planning

9.1.1 Duration

This sprint started on October 22th and lasted for two weeks. A customer demo was held at the 1st of November to show of what we had achieved during the sprint and to ensure that the customer agreed with the solutions and the direction we were going in.

9.1.2 Sprint Goal

The goal for this sprint was to redesign the system to accomodate to the new wishes from the customer, and to present a working protoype incorperating new technologies and concepts suggested by the customer.

9.1.3 Product Backlog

Since the end of Sprint 2, our customer representative had presented the system to other representatives of his company. From the current state of the prototype, they had derived a distinct, new set of new requirements that called for a paradigm shift for our project. Our customer emphasized a few features that our system should support:

- Arbitrary mathematical operations - Make it possible to execute arbitrary mathematical operations on the numerical attributes of the objects accessible through the console.
- Schemaless design - Allow the user to insert objects into other objects even though they were not specifically designed with this in mind.
- Navigation of database as a folder structure - Make it possible to navigate the contents of the database as a folder structure, where any type of object can be represented.

Futhermore, the customer no longer wanted us to focus on improving usability features, and instead wanted us to simplify the web UI design. It was also no longer desired that objects were synchronized in real time.

Because real time synchronization was no longer an issue, we decided to no longer use PubNub. To ease the implementation of the folder structure navigation, we decided to adopt **CouchDB** instead of our previous node.js/MongoDB backend. Also, because of the requirement of schemaless objects, and the fact that the previous user interface design was too complex, we decided to do a complete rewrite of the system. For simplicity, the new prototype would not use the JavaScript Shell, jquery.blockUI or PersistJS.

All previous user stories were removed from the product backlog and replaced with new ones:

- 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.

The full product backlog for sprint 3 is listed in section E.3 in the appendix.

9.1.4 Sprint Backlog

The user stories included in the sprint backlog is presented in Table 9.1. As we chose to accomodate the new wishes from the customer we were forced to use a considerable amount of time on reasearch on new technologies and on redesigning the system to utilize these technologies. As a result the amount of time available for implementation of user stories was reduced, and this time we only had 50 hours available for this purpose. From experiences from the previous sprints, we tried to make each user story concise and not too general. We feel this helped us a lot in estimating the user stories more accurately than in previous sprints.

Table 9.1: Sprint 3 Backlog

User Story	Short Description	Hours Est.	Hours Act.
G9	**Navigate the database like a directory**	**6**	**4.5**
	Decide on syntax		
	Detect and parse navigation commands		
	Change directory when command is issued		
	Testing		
	Documentation		
G10	**Store variables in the console**	**5**	**4**
	Make objects in current directory available		
	Ensure persistent storage of the stored objects		
	Testing		
	Documentation		
G11	**GUI represent current directory**	**5**	**4**
	Make viewsfor each directory in CouchDB		
	Call the correct view when changing directory		
	Update the GUI		
	Testing		
	Documentation		
G12	**Expose properties of the objects**	**6**	**8.5**
	Make objects available		
	Make objects in GUI clickable		
	Query database on selection		
	Update GUI		
	Testing		
	Documentation		
G14	**Perform mathematical operations**	**9**	**6**
	Allow user to use mathematical functions		
	Testing		
	Documentation		
G15	**Change or add attributes**	**3**	**4**
	Allow user to edit the attributes		
	Allow user to add new attributes		
	Testing		
	Documentation		
G16	**Allow user to add entire JSON objects**	**2**	**2.5**
	Allow user to assign JSON objects as attributes		
	Testing		
	Documentation		
G17	**Replicate changes to GUI**	**3**	**2**
	Detect changes made from the console		
	Update the GUI accordingly		
	Testing		
	Documentation		
G18	**Allow user to work locally**	**4**	**5**
	Keep track of actions made by user		
	Add command for storing on server		
	Testing		
	Documentation		
	Total:	**50**	**47**

9.2 Architecture

9.2.1 4+1 view model

The 4+1 view model [10]. Here the views will be described as they are in sprint 3, and how they will look in our architecture.

Logical View

Describes the functionality in the system from the end users perspective. The end users will mainly be power users, wanting to perform object editing tasks efficiently. This view will be described through class, communication and sequence diagram.

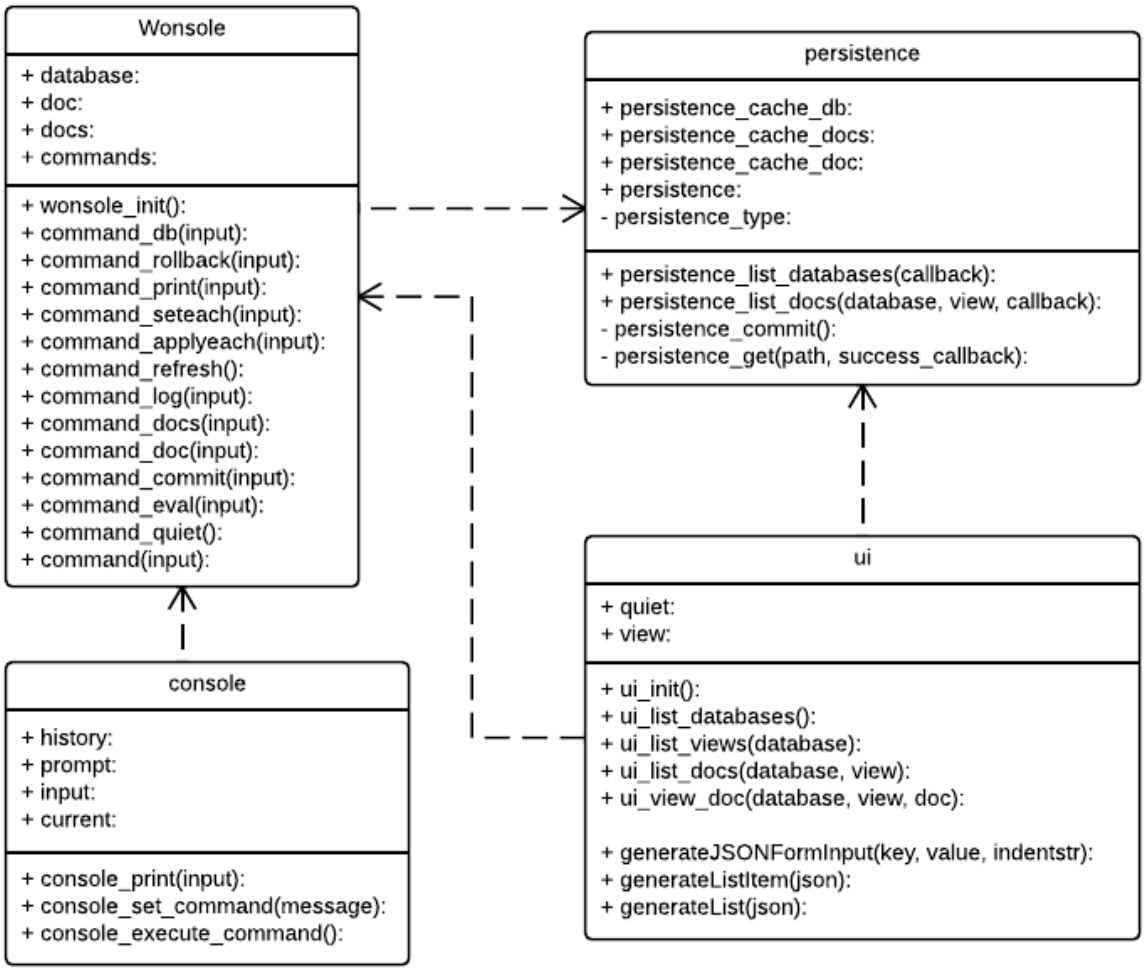

Figure 9.1: Client Class Diagram

Figure 9.1 The client class diagram gives an overview of the class structure of system, and how they collaborate. This sprint we made some changes to the client side to better handle and work with the new database system we introduced in this sprint. We still has a UI/Console view, for the user to work with, but the console and UI now can work more freely onto the database through commands, as seen in the "Wonsole"-class. Since the edited client and new database together made a more dynamic system, the objects the user can work with does not need to be restricted to Book and library objects. The persistence class is instead holding more dynamic versions of these. Changes done in the console is still reflected onto the UI through the Wonsole and persistence classes. The changes are pushed onto

the server through a "commit"-command, instead of an automatic push done by the previously used PubNub.

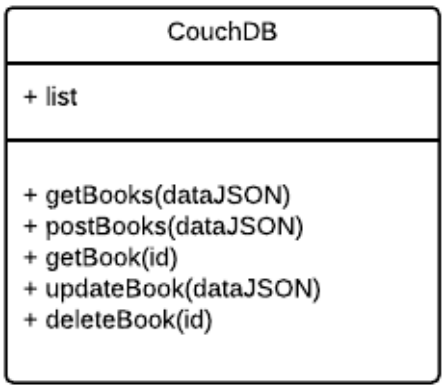

Figure 9.2: Client Class Diagram

Figure 9.2 The server class diagram shows how the REST api is set up. Since the PubNub no longer is a used function, it is removed from the server side also. The new database system is both a REST api and a database system, this renders the separation of REST and DBCommunicator obsolete, and lets us merge it all together into one component.

Physical View

Describes the system from the system engineer's perspective. And explains the physical connections between the software components. Described through a deployment diagram.

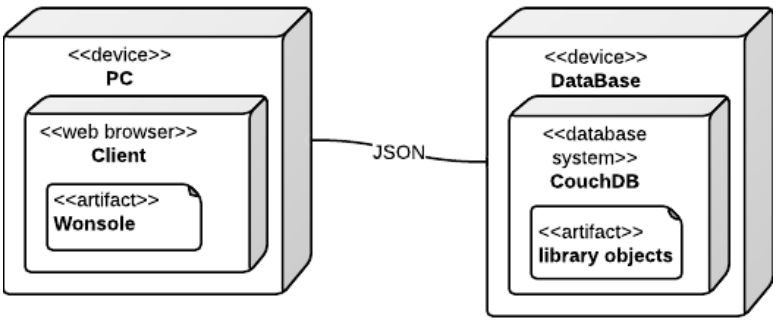

Figure 9.3: Deployment Diagram

Figure 9.3 The structure of the now two different parts of the system, database can be accessed without a middleman, and the PubNub is no longer used.

Development View

Describes the system from the programmer's perspective. This will be described through how the different component parts are separated. Component and package diagrams will show this.

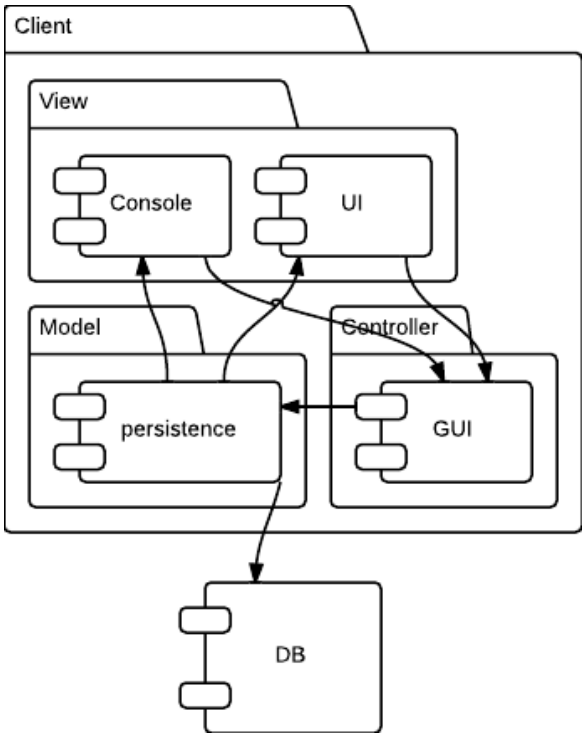

Figure 9.4: Component Diagram

Figure 9.4 These components still form a three layered structure, but the DB component is both the logical tier and the data tier. The components communicates with each other through the neighboring layer.

9.3 Implementation

9.3.1 Database change

The biggest change in sprint 3 is the change of underlying database system. In first two sprints, we used MongoDB as a database system. This was combined with the server written in Node.js, that provided a REST API for database access. Clients used this API to manipulate with the data. Along with the new requirements, customer suggested, that we evaluate choice of the database system and also suggested replacement: CouchDB. Key functionality of CouchDB is, that it provides the complete REST interface and also another great functionality is custom views defined using functions.

New requirements would bring changes for the implemented Node.js server providing REST interface, so we decided to replace the database system and deprecate rest of the server part. We installed CouchDB, created few databases with test data. The REST interface is provided on default, since it is also used for the database management. Only disadvantage of this approach is that all of the application logic is transfered to client, we refer to this as thick client.

9.3.2 Same origin policy

The database system providing the REST API, that is used to retrieve and upload data usually runs on different server or different port. When doing AJAX calls from client directly to database, it is considered as different origin **??**. This problem can be fixed by using JSONP, but this would limit us to use of HTTP GET method. Another problem solution is to modify HTTP headers and set header alowing all origins. We solved this simply by setting up the web server that serves files to proxy the calls to database.

9.3.3 Persistence

Since the database system changed in this sprint, new implementation of communication with database needed to be reimplemented. In earlier sprints, we could design the API ourselves so the implementation was straightforward and simple. Now, we are using built in API of CouchDB that is more complex - so we implemented persistence layer that is an abstract on the data persistence. There are defined few types of persistence such as local connection to database, production server connection to database and demo data - this means, the persistence layer will serve only data for demonstration, without connecting anywhere. Developer can easily set the persistence mode using one variable and the source of data changes automatically. This ensures, that any change in the future is very easy - you just have to implement the API for new database system, binary files, whatever you want and add this option to a set.

9.3.4 Commands

Commands are also brand new in this sprint. We created another set of commands, another simple language on top of JavaScript. These commands allow the functionality of Wonsole to be more accessible - you dont have to implement everything in JavaScript, you can just use the set of predefined commands. In fact, you don not have to use JavaScript at all. See the table **??** for command reference.

9.3.5 Parts of the library

The Wonsole library was divided into 4 main parts in different source files: ui representing functions and callbacks from user interface, persistence - data source (see Persistence subsection), console - implementing functionality of console and main script wonsole.js containing the commands and putting it all togethe r.

9.4 Testing

9.4.1 Test Results

We performed a total of 6 test cases during this sprint; TID20-25. The results are listed in Table 9.2. The test cases themselves can be found in the appendix C.3.

9.4.2 Test Evaluation

We had one failed test this time, which was due to some missing functionality in the GUI. More specificly it is not possible to add new attributes to existing objects from the GUI. We decided not to correct this, as the focus of this project is the console and its functionality. We prioritized to implement more functionality to the console instead. This was a feeling shared by the customer, they wanted us to exhibit what was possible to do in the console. The functionality of the GUI was not important to them.

Table 9.2: Sprint 3 Test Results

Item	Description
TestID	**TID20**
Description	Changing directory in the application in the console.
Tester	Øystein Heimark
Date	02/11 - 2012
Result	Success
TestID	**TID21**
Description	Storing objects as local variable.
Tester	Øystein Heimark
Date	02/11 - 2012
Result	Success
TestID	**TID22**
Description	Allow for the use of functions on the objects.
Tester	Øystein Heimark
Date	02/11 - 2012
Result	Success
TestID	**TID23**
Description	Allow for editing and adding of attributes.
Tester	Øystein Heimark
Date	02/11 - 2012
Result	Failure. It is not possible to add attributes from the GUI
TestID	**TID24**
Description	Update GUI according to changes.
Tester	Øystein Heimark
Date	02/11 - 2012
Result	Success ’
TestID	**TID25**
Description	Store changes to database.
Tester	Øystein Heimark
Date	02/11 - 2012
Result	Success

9.5 Customer Feedback

This sections covers the feedback we got from the customer, both before and after the sprint.

At the end of sprint 2 the customer representative informed us that he would have a meeting with others in the company, to try to identify what our product was missing, a x- factor that would appeal to the users and promote the console. At the Scrum meeting for the third sprint he announced that they had indeed found this x- factor. He wanted us to return to the roots of the project, namely adding scripting to a web- page. We should focus on adding functionality which is impossible or difficult and time consuming to do in a regular GUI.

He wanted us to add functionality that would show of the advantages of the technologies we are using, and how it would not be possible to do this with more traditional technologies. He listed some general use cases he wanted us to implement, and suggested that we looked into another solution for the backend. This was because he felt that this solution was better suited for the use cases he now presented us. He was aware of the time constraints of the project and did not expect us to manage to implement all the functionality he mentioned. But it was important that we could document that it would be possible to implement it with the technologies we are using in this project.

In the middle of the sprint we did a technology preview for the customer. The purpose of this meeting was to ensure that we were going in the right direction. We presented the user stories we had derived from the new requirements and the customer was all in all happy with these. The customer suggested that we should focus on core functionality, features that will stand out during the presentation of the project. We also presented him what we had implemented so far, including an early implementation of the CouchDB system and an early draft of the GUI. He was impressed what we had managed to implement so far, and encouraged us to keep up the good work.

At the end of the sprint we did a sprint demonstration were we presented our results to the customer. He was very happy with our progress, and didn't expect us to be able to get so far so early. For the next sprint he requested that we make a manual for how to use the system(for the users), and a deployment manual for how to set up the system on a new server(for an administrator). He also wanted us to send him a copy of our report, so he could give us some feedback on it.

9.6 Evaluation

This section contains the evaluation of the third sprint, and what we plan to improve for the fourth sprint. The evaluation was performed during an internal meeting with all group members present.

9.6.1 Review

Again we feel we were able to deliver a great product, and we were able to complete all the user stories in the sprint backlog. This sprint posed a different kind of challenge for us, as we decided to switch some of the technologies we were using and to redesign much of the system. This was obviously not according to plan, but we feel we rose to this challenge, adapted well and succeded in making the switch. We found time to research new technologies as well as incorporating these into the design of the system.

The communication with the customer was good throughout this sprint, even better than in the previous sprints. The customer was more included in the planning of this sprint, and he was more specific in what he wanted to be implemented. This helped us a lot when we were creating new user stories. This time we were also able to schedule a meeting with the customer in the middle of the sprint, to allow him a chance to provide some feedback on the prelimanary solutions we had implemented so far. We feel this great communication was reflected by the fact that the customer was very happy with the end result, which exceeded his expectations.

In previous sprints, not enough focus was put on the report and general documentation. To address this

issue we decided to put two team members full time on the report for the last week of this sprint, and it resulted in more work being done with the documentation. We were able to document our work on the research and implementation effectively during the sprint, and also found time to fix some parts of the report according to feedback we recieved from the advisor. All in all we feel this was a successful move.

Despite it beeing a successful sprint in many ways, there was still some aspects of it that was not so good. In this sprint we were too dependent on one person for the implementation. We planned to assign two team members to the implementation, but one of them ended up working on the report instead. Although the person responsible for the implementation did a fantastic job, and we ended up delivering a great product, we should have done a better job in distributing the workload of the implementation. If the one assigned to the implementation somehow would be unable to work the last couple of days, we would have been in big trouble. We should have at least two team members involved in the implementation, to avoid one single point of failure.

We also fell back to our old habits with the time tracking. Our focus was disturbed by the change of direction in the project, and in the middle of it all we forgot to follow up on our routines on time tracking. We also failed to peform the test cases earlier, as we set out to do before the sprint. This may also be credited to the changes. We ended up being very pressed on time towards the end of the sprint and prioritised to complete the implementation instead.

Positive

- We were able to switch technologies without too many complications
- Finished all the user stories on time
- Great contact with the customer throughout the sprint
- Better preparation for customer demonstration
- Was able to create small and precise user stories, which was easier to estimate

Negative

- Too dependent on one person with implementation
- Time tracking, returned to our bad habits from sprint 1
- Failed to perform tests earlier, as we planned.

9.6.2 Planned Responses

These are the actions we have planned for the final sprint to improve our work process.

Workload Distribution

Continue the success of putting two team members full time on the report, as this turned out to be a great measure to ensure the quality of the report. Although there is not a whole lot left to do on the implementation, we should assign two people to the tasks remaining. This to avoid a potensial single point of failure.

Time Tracking

Return to our routines for time tracking, which we were able to do so well in the second sprint.

Chapter 10

Sprint 4

Contents

This chapter will outline the work we did in the fourth and final sprint. It explains in detail how we planned the sprint, including which user stories we chose and the architecture we employed to implement these. Details on the implementation on each user story is also included, as well documentation on the testing process. Finally our evaluation of the last sprint is presented.

10.1 Planning

10.1.1 Duration

This sprint started on November 5th and lasted for two weeks. A final customer demo was held on the 17th of November to show of the final product for the customer.

10.1.2 Sprint Goal

For this sprint a lot of our focus was put on the report and preparation of the final report. At the beginning at the sprint both the team and the customer agreed that the the product was good and only needed minor improvements for the final presentation. So the goal for this sprint was to get as much work done on the report as possible and at the same time start to prepare much of the final presentation. Also we would like to implement small improvements to the system, making it ready for the final presentation.

10.1.3 Product Backlog

For Sprint 4, the customer primarily wanted us to focus on making the prototype more fit for demonstration purposes, and work on documentation, including the report, a user manual and a deployment manual.

The user stories completed in sprint 3 were removed, and the following user stories were added:

- 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 delete objects from the system using a simple syntax.
- G22 - As a user, I want to be able to define and execute scripts.

The full product backlog for sprint 4 can be found in section E.4 in the appendix.

10.1.4 Sprint Backlog

The user stories included in the sprint backlog is presented in Table 10.1. Because the majority of the desired functionality was already implemented, we would spend less time on implementation on this sprint.

Table 10.1: Sprint 4 Backlog

User Story	Short Description	Hours Est.	Hours Act.
G13	**Call function on group of objects**	**8**	**9.5**
	Create mechanism for executing code on an object		
	Create DSL command		
	Testing		
	Documentation		
G19	**Filter a list of objects**	**4**	**4**
	Create a function to filter a list		
	Create DSL command		
	Testing		
	Documentation		
G20	**Add new objects with simple syntax**	**1**	**1**
	Create DSL command		
	Testing		
	Documentation		
G21	**Delete objects with simple syntax**	**1**	**1**
	Create DSL command		
	Update GUI		
	Testing		
	Documentation		
G21	**Define and execute scripts**	**8**	**7**
	Create mechanism for storing and executing scripts		
	Create DSL command		
	Testing		
	Documentation		
	Total:	**22**	**22.5**

10.2 Architecture

The user stories picked out for sprint four did not change much on the architecture from the previous sprint. Some rearanging was done, but nothing major.

10.2.1 4+1 view model

Logical View

Process View

Physical View

Development View

10.3 Implementation

10.4 Testing

10.4.1 Test Results

We performed a total of 4 test cases during this sprint; TID26-29. The results are listed in Table 10.2. The test cases themselves can be found in the appendix C.4.

10.4.2 Test Evaluation

We did not implement too many user stories in this last sprint, so there were not a whole lot of tests to be performed. All the test cases we did passed without any complications to mention.

Table 10.2: Sprint 4 Test Results

Item	Description
TestID	**TID26**
Description	Call specific functions on group of objects.
Tester	Øystein Heimark
Date	15/11 - 2012
Result	Success
TestID	**TID27**
Description	Filtering objects.
Tester	Øystein Heimark
Date	15/11 - 2012
Result	Success
TestID	**TID28**
Description	Adding and deletion of objects.
Tester	Øystein Heimark
Date	15/11 - 2012
Result	Success
TestID	**TID29**
Description	Creating scripts.
Tester	Øystein Heimark
Date	15/11 - 2012
Result	Success

10.5 Customer Feedback

The customer was impressed by the new scripting functionality, which was not explicitly requested by him. He was also happy with the results in general, and aware that the last sprint would not involve major changes.

The customer wanted us to document that the prototype allows a lot of unsafe user input in its current state, since it permits any JavaScript code. Work for the future would be to put more restrictions on what the user is allowed to do, to avoid problems.

Our final prototype does not solve a problem that hasn't been solved before; There are existing systems that have similar feature sets. However, it achieves the goals in a more cost efficient way, both for the software developer(us) and the user, and this is where its strength lies.

10.6 Evaluation

10.6.1 Review

We had a very limited sprint backlog for this sprint. Our focus was instead on the project report and demonstration. Regardless, we feel that we did a good job on getting the prototype completed.

10.6.2 Positive

- Implemented all user stories
- Discovered new possibilities with the console underway
- Implemented more than customer wanted

- Useful feedback on trial presentation with advisor

10.6.3 Negative

- Communication problems with customer (demonstration was unexpectedly postponed one day)
- Presentation only partially complete, still a lot of work to do
- Poor time management led to a lot of work near the end of the sprint

Chapter 11

Evaluation

Contents

In this chapter we will present our internal evaluation of the project and what we have achieved.

11.0.4 Team Dynamics

11.0.5 Customer Relations

11.0.6 Issues

11.0.7 Planning

11.0.8 Quality Assurance

11.0.9 Summary

Chapter 12

Conclusion

Contents

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.

12.1 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 12.1. 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.

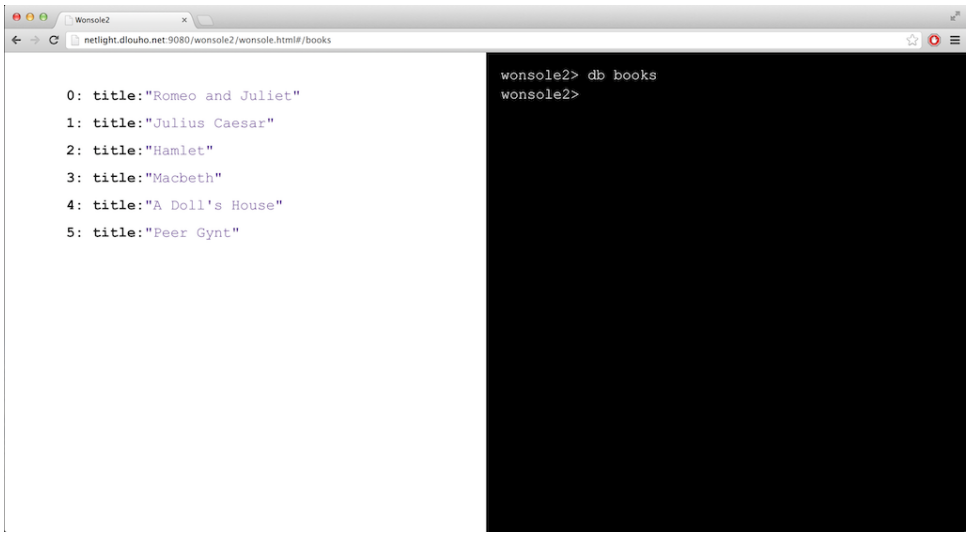

Figure 12.1: Client Screenshot

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, delete and navigation between the different databases. The commands offers a layer of abstraction to the users, and makes it easier for inexperienced users to utilise the console. The commands are functions written in JavaScript, and when recognised 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 recognised 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, delete, update

and filter.

12.2 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 was 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 simplified 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 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 model 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.

12.3 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 model before the system is deployed, and the model may need to change over time. Traditionally, database redesign would be a much more complicated and expensive process.

references: David M. Kroenke's Database Processing Fundamentals, Design, and Implementation (10th Edition) (Chapter Eight) http://www.articlesphere.com/Article/Read-This-Before-You-Redesign-Your-Database-/49329

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.

12.4 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.

12.4.1 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 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 customers 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.

12.4.2 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.

12.5 Further Developement

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 delivered 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 behaviour 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.

12.5.1 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.

12.5.2 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.

12.6 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 delivery 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

Risk Table

#	**Risk**	**Probability**	**Impact**
1	Not getting a fifth party member	M	Significant
2	Obtrusive health/family/personal issues for team members	L	Significant
3	Low morale in team	M	Significant
4	Interfering workload from other activities	H	Minor
5	Miscommunication with customer	M	Critical
6	Changes in customer requirements	M	Significant
7	Errors in project plan	M	Significant
8	Failure of communication in team	M	Critical
9	Failure of time management	H	Critical
10	Errors in workload estimation and distribution	H	Critical
11	Failure of online storage systems and services	L	Significant
12	Failure of personal computers	M	Significant
13	Infeasibility of project as a whole	L	Critical
14	Inability to find potential users and test subjects	M	Significant

Table A.1: Risk overview

Risk #	01
Activity	All
Risk Factor	Not getting a fifth party member
Impact	Significant
Consequence	Increased workload for all remaining party members on all activities
Probability	Medium
Countermeasures	• Contact advisor about the dropped party member, try to get assigned a new member. • Take the missing person into account in planning phase.
Deadline	Intro/Planning (Ultimately in the hands of course staff)
Responsible	Project leader

Table A.2: Risk 01

Risk #	02
Activity	All
Risk Factor	Obtrusive health/family/personal issues for team members
Impact	Significant
Consequence	Increased workload for all remaining party members on all activities
Probability	Low
Countermeasures	• Implement buffers in project plan. • Team members should make their work resumable by another member.
Deadline	None
Responsible	Project leader

Table A.3: Risk 02

Risk #	03
Activity	All
Risk Factor	Low morale in team
Impact	Significant
Consequence	Decreased overall project quality
Probability	Medium
Countermeasures	• Frequent contact between team members • Avoid team members overworking • Focus on general team dynamics advice from advisor
Deadline	None
Responsible	Project leader

Table A.4: Risk 03

Risk #	04
Activity	All
Risk Factor	Interfering workload from other activities
Impact	Low
Consequence	Work on the project is shifted in time, space and responsibility
Probability	Very High
Countermeasures	• Plan ahead with respect to existing schedules • Inform the group of other activities
Deadline	None
Responsible	Project leader

Table A.5: Risk 04

Risk #	05
Activity	All
Risk Factor	Miscommunication with customer
Impact	Critical
Consequence	-The project is not developed as the customer wants it -Work has to be done over
Probability	Very High
Countermeasures	• Weekly customer meetings • Share as much information as possible with customer at all stages
Deadline	None
Responsible	Customer Contact

Table A.6: Risk 05

Risk #	06
Activity	Planning, Requirements, Implementation
Risk Factor	Changes in customer requirements
Impact	Significant
Consequence	Work has to be done over
Probability	Medium
Countermeasures	• Design the prototype with possible modifications in mind. • Try to get information on possible changes from the customer.
Deadline	None
Responsible	Customer Contact

Table A.7: Risk 06

Risk #	07
Activity	Implementation
Risk Factor	Errors in project plan
Impact	Significant
Consequence	Work on the plan and implementation have to be redone
Probability	Medium
Countermeasures	• Review the project plan frequently for consistency • Share plan with customer
Deadline	Planning
Responsible	Project Leader

Table A.8: Risk 07

Risk #	08
Activity	All
Risk Factor	Failure of communication in team
Impact	Critical
Consequence	Failure of unification of the work, uneven workloads, decreased project quality
Probability	Medium
Countermeasures	• Frequent internal meetings • Sharing of work internally
Deadline	None
Responsible	Project Leader

Table A.9: Risk 08

Risk #	09
Activity	All
Risk Factor	Failure of time management
Impact	Critical
Consequence	Parts of project are rushed or not finished in time
Probability	High
Countermeasures	• Put in as much work as possible as early as possible • Implement buffers in project plan
Deadline	None
Responsible	Project Leader

Table A.10: Risk 09

Risk #	10
Activity	All
Risk Factor	Errors in workload estimation and distribution
Impact	Critical
Consequence	Uneven workloads, rushed or unfinished parts of project
Probability	High
Countermeasures	• Implement buffers in project plan • Avoid relying too much on rigid plans • Allow for redistribution of work when necessary
Deadline	Planning
Responsible	Project Leader

Table A.11: Risk 10

Risk #	11
Activity	All
Risk Factor	Failure of online storage systems and services
Impact	Critical
Consequence	Work is lost and has to be recreated
Probability	Low
Countermeasures	• Local backups of data • Know of alternative systems in case of failure
Deadline	None
Responsible	Project Leader

Table A.12: Risk 11

Risk #	12
Activity	All
Risk Factor	Failure of personal computers
Impact	Significant
Consequence	Work may be lost, decreased productivity of team member
Probability	Medium
Countermeasures	• Use primarily online storage systems and keep online backups of everything else • Use university computers if necessary
Deadline	None
Responsible	Individual

Table A.13: Risk 12

Risk #	13
Activity	All
Risk Factor	Infeasibility of project as a whole
Impact	Critical
Consequence	The concept is not a solution to the problem and the prototype is destined to be a failure
Probability	Very Low
Countermeasures	• Extensive preliminary study to uncover this as early as possible
Deadline	Feasibility study
Responsible	Project Leader

Table A.14: Risk 13

Risk #	14
Activity	Planning
Risk Factor	Inability to find potential users and test subjects
Impact	Significant
Consequence	Requirements engineering and prototype testing will be sub-standard unable to provide adequate answers
Probability	Medium
Countermeasures	• Try to get information on potential users from customer • Begin contacting potential users and testers early
Deadline	Testing
Responsible	Project Leader

Table A.15: Risk 14

Appendix B

Templates

B.1 Status Report Template

Weekly Status Report

	Week ??(??th ???. – ??th ???.)
Project	
Project name	**Customer Driven Project – Group 7**
Customer:	**Netlight**
Participants	
Oddvar Hungnes	**Ivo Dlouhy**
Øystein Heimark	**Martin Havig**

Table B.1

1 Summary

...

2 Work done this period

...

Milestones reached this week:

...

2.1 Status of documents

-We currently have the following documents:

* The LaTex report

* User and administrator manual

2.2 Meetings

- We had an internal meeting 12th Nov.

3 Problems and issues

...

4 Planning for the following period

...

4.1 Meetings These are the meetings we have planned (disregarding internal meetings):

- 15th Nov. Advisor meeting
- 15th Nov. Customer demonstration
- 22th Nov. Final demonstration

Week	Activity	Estimate	Actual
n	...	...	...
	...	...	...
		sum	**sum**

Table B.2

B.2 Meeting Agenda Template

Meeting agenda DD/MM

Time: 09:15

Place: ITV 242

Attendees:

On behalf of the group: Øystein Heimark, Martin Havig, Oddvar Hungnes, Ivo Dlouhy

Advisor: Meng Zhu

Agenda

- Go through notes from last meeting
- Discuss the progress report
- ...
- ...
- Discuss work for next sprint

B.3 Meeting Notes Template

Advisor Meeting Notes DD/MM

Time: 09:15

Place: ITV 242

Attendees:

On behalf of the group: Øystein Heimark, Martin Havig, Oddvar Hungnes, Ivo Dlouhy

Advisor: Meng Zhu

Last week's notes

Progress report

Work for the next week

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, deletes 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 obje
Tester	Øystein Heimark
Preconditions	There needs to be objects present in the
Feature	The ability to call a command that applie
Execution steps	1. Open a new client 2. Make a group of books and apply a fu 3. Make another group of books and ap the price
Expected result 15 , while the books in the second group have their price decreased by 15 .	The books in the first group have their pr in the second group have their price decr

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 deletion 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 delete specific books already in it.
Execution steps	1. Open a new client 2. Call command for adding a new book. 3. Call command for deleting the newly added book.
Expected result	The book is first added to the system and appears in the list of books. When the deletion 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 delete 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'
});
```

Delete a book by id

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

Resource URL: http://netlight.dlouho.net:9004/api/books/:id
HTTP Methods: DELETE
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:
DELETE http://netlight.dlouho.net:9004/api/books/506b6445b107d7567a000001

Example call in jQuery:

```
$.ajax({
  type: 'DELETE',
  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 delete 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 delete objects from the system using a simple syntax.
- G22 - As a user, I want to be able to define and execute scripts.

Appendix F

Cheatsheet

Wonsole Cheatsheet

Command	Description	Example usage
`db DATABASE`	opens database DATABASE	`db books`
`docs`	switches view to documents	`docs`
`doc INDEX`	opens document at INDEX	`doc 0`
`commit`	commits the changes to database	`commit`
`rollback`	rollbacks the changes	`rollback`
`refresh`	refreshes the UI	`refresh`
`clear`	clears the console	`clear`
`print`	VARIABLE prints the variable VARIABLE	`print docs`
`quiet`	toggles list verbosity, default: on	`quiet`
`log`	toggles log visibility, default: off	`log`
`remove INDEX`	removes object at index INDEX	`remove 0`
`remove JSON`	removes object specified by JSON	`remove` {"id":1231232}
`add JSON`	adds object specified by JSON	`add \{"title" : "New book"\}`
`seteach ARRAY ATTRIBUTE VALUE`	sets (creates if necessary) ATTRIBUTE in `object in ARRAY to VALUE`	`seteach docs author "Unknown"`
`foreach ARRAY CODE`	executes the CODE for each item of ARRAY	`foreach docs price=price*10`
`filter ARRAY ATTRIBUTE VALUE`	filters the ARRAY with VALUE of ATTRIBUTE	`filter docs title Report`
`script ARRAY`	executes the ARRAY of commands specified by string	`script ["clear", "db books"]`

Appendix G

User Manual

Wonsole

The new web console for power users

User Manual

NTNU – Trondheim
Norwegian University of
Science and Technology

Customer Driven Project
November 20, 2012
Team: Ivo Dlouhy, Martin Havig, Øystein Heimark, Oddvar Hungnes

Contents

List of Figures

Appendix H

Introduction

This document describes how to use the Wonsole application. It contains a user tutorial as well as command refence.

Appendix I

Basics

User interface for wonsole is minimal a simple, so that the funcionality gets presented in a clear way. If you see the main screen in the picture I.1, on the left hand side, there is a GUI - Graphical User Interface - this is where data get presented as graphics. On the right hand side, there is a console window. This is a place where user can input the commands and send the commands to the system by pressing enter.

By default the list of available databases is displayed in GUI and command prompt in console.

Figure I.1: Main screen

Appendix J

Tutorial

J.1 Database: db

The most basic command in Wonsole is the command to open, or switch database. It has this format:

```
db DATABASE
```

Where DATABASE is the name of the database to open. Databases are listed in GUI and they can be opened by clicking at the name too.

After issuing the command, the database contents are displayed in GUI, see picture J.1. Wonsole uses document oriented database, so it is list of documents indexed by numbers. From each document the title attribute is displayed as preview.

Same command can be used to switch the database anywhere in the workflow as in picture J.2.

Note: All uncommited changes are lost when switching the database.

```
0: title:"Romeo and Juliet"
1: title:"Julius Caesar"
2: title:"Hamlet"
3: title:"Macbeth"
4: title:"A Doll's House"
5: title:"Peer Gynt"
```

```
wonsole2> db books
wonsole2>
```

Figure J.1: Database open

```
0: title:"Henrik Ibsen"
1: title:"William Shakespeare"
```

```
wonsole2> db books
wonsole2> db authors
wonsole2>
```

Figure J.2: Database switch

J.2 Presentation: print and view

Command print is available to display data in plaintext instead of using GUI. It's syntax is very simple - it has one attribute: variable or data to print.

```
print DATA
```

Command loads the data, converts it into text and prints in the console as ssen in picture J.3.

Oposite of the print command there is view. It displays the complex variable in the GUI.

```
view VARIABLE
```

Figure J.3: Print

J.3 JavaScript integration

Wonsole includes complete JavaScript language to the command set. If user issues acommand, that is not recognized as the Wonsole command, this input gets evaluated as JavaScript. You can for example issue a helper command, that does some required functionality not supported by Wonsole and than continue using default Wonsole commands, JavaScript language is 100

```
var new = "New variable" //Javascript
print variable
"New variable"
```

This is an example of creating a new JavaScript variable and using it in simple Wonsole command, demonstration on picture J.4.

Note: This is advanced functionality. JavaScript language is very complex and it is the user full responsibility to use it in a safe, suitable way.

```
authors

books
```

```
wonsole2> var variable = "New variable"
wonsole2> print variable
"New variable"
wonsole2>
```

Figure J.4: Javascript integration

J.4 Display: Quiet

When Wonsole displays a list of documents saved in the database (see command db) by default it uses quiet mode. In quiet mode, it displays only object index and value of the title attribute as a object identification as you can see in picture J.5.

```
quiet
```

You can toggle preview mode by using command quiet. When toggled off, all attributes of the object are displayed - but only the simple data types. If the atribute is complex, the Object keyword is displayed instead, see picture J.6.

```
0: title:"Romeo and Juliet"
1: title:"Julius Caesar"
2: title:"Hamlet"
3: title:"Macbeth"
4: title:"A Doll's House"
5: title:"Peer Gynt"
```

```
wonsole2> db books
wonsole2>
```

Figure J.5: Quiet mode on

```
0: title:"Romeo and Juliet",
  price:100,
  language:"en",
  tags:Object,
  stock:5
1: title:"Julius Caesar",
  price:100,
  language:"en",
  tags:Object,
  stock:5
2: title:"Hamlet",
  price:170,
  language:"en",
  tags:Object,
  stock:7
3: title:"Macbeth",
```

```
wonsole2> db books
wonsole2> quiet
wonsole2>
```

Figure J.6: Quiet mode off

J.5 Documents: docs and doc

After opening the database, list of documents is loaded. Is is stored in `docs` variable for you to work with. This variable is an array, so you can index it and get a single document object as seen in picture J.7.

```
db books
docs
[...]
```

Documents displayed in the document list mode are indexed using numbers. You can use these numbers together with `doc` command to open a single document. This command can be also used to switch between documents, you just have to issue it with different index. The opened document is stored inside a `doc` variable as demonstrated in picture J.8.

To change the document, use a simple assignment command. This way you can modify the object freely, you can also add a new attribute. Example is in the picture J.9

```
0: title:"Romeo and Juliet"
1: title:"Julius Caesar"
2: title:"Hamlet"
3: title:"Macbeth"
4: title:"A Doll's House"
5: title:"Peer Gynt"
```

```
wonsole2> db books
wonsole2> print docs
[{"_id":"0b432ce4fb079967bbe5320f33003c50","_rev":"19-
4f3cffa172aca6b413c39865c838bf9c","title":"Romeo and
Juliet","price":100,"language":"en","tags":
["tragedy","pre1800","play"],"stock":5},
{"_id":"0b432ce4fb079967bbe5320f33003fa3","_rev":"19-
c55a2b1dffdeb8426a2fe1079bb06b3c","title":"Julius
Caesar","price":100,"language":"en","tags":
["tragedy","pre1800","play"],"stock":5},
{"_id":"0b432ce4fb079967bbe5320f330049a5","_rev":"17-
f964f928dcafca2802195fc2868c3611","title":"Hamlet","price":1
["tragedy","pre1800","play"],"stock":7},
{"_id":"0b432ce4fb079967bbe5320f330055c0","_rev":"17-
de1c60db46d267b3538c4eac5e33c8e5","title":"Macbeth","price":
["tragedy","pre1800","play","onsale"],"stock":0},
{"_id":"0b432ce4fb079967bbe5320f33006ce0","_rev":"12-
6086f6f8fcff5337083a78839a6fe957","title":"A Doll's
House","year":1879,"tags":
["boring","toolong","post1800","nora"],"language":"no","pric
{"_id":"0b432ce4fb079967bbe5320f33006e34","_rev":"11-
b05670b2552056cdf5eef59afd38d8dd","title":"Peer
Gynt","year":1876,"tags":
["mediumboring","post1800"],"language":"no","price":299,"sto
wonsole2> print docs[0]
{"_id":"0b432ce4fb079967bbe5320f33003c50","_rev":"19-
4f3cffa172aca6b413c39865c838bf9c","title":"Romeo and
Juliet","price":100,"language":"en","tags":
["tragedy","pre1800","play"],"stock":5}
wonsole2>
```

Figure J.7: List of documents

```
Object
  title: Julius Caesar
  price: 100
  language: en
    tags:
      0: tragedy
      1: pre1800
      2: play
  stock: 5
```

```
wonsole2> db books
wonsole2> doc 0
wonsole2> doc 1
wonsole2> print doc
{"_id":"0b432ce4fb079967bbe5320f33003fa3","_rev":"19-
c55a2b1dffdeb8426a2fe1079bb06b3c","title":"Julius
Caesar","price":100,"language":"en","tags":
["tragedy","pre1800","play"],"stock":5}
wonsole2>
```

Figure J.8: Doc object

Figure J.9: Modification of doc

J.6 Transactions: Commit and Rollback

All changes you perform on objects are local, so at any point during the modification process, transaction, you have a option to save all the changes to database using command `commit` or to throw away all local changes and load back the values from database with command `rollback`. This functionality can be used as an undo command, it is up to you, how often you commit the changes.

J.7 Objects: Add and Remove

To add a new object, use an `add` command. It has one parameter, object to add. If you are not sure, just create an empty object. Otherwise you can directly define specific object attributes as in picture J.10. The new object will immediatelly appear in document list, so you can open it and continue working with it, for example add a new attribute as in picture J.10.

```
0: title:"Romeo and Juliet"
1: title:"Julius Caesar"
2: title:"Hamlet"
3: title:"Macbeth"
4: title:"A Doll's House"
5: title:"Peer Gynt"
6:
7: title:"New book"
```

```
wonsole2> db books
wonsole2> add {}
wonsole2> add {"title":"New book"}
wonsole2>
```

Figure J.10: Add

```
Object
  title: New book
  note: Not yet available
```

```
wonsole2> db books
wonsole2> add {"title" : "New book"}
wonsole2> doc 6
wonsole2> doc.note = "Not yet available"
wonsole2>
```

Figure J.11: Add and modify

Appendix K

Administrator Manual

Wonsole

The new web console for power users

Administrator Manual

NTNU – Trondheim
Norwegian University of
Science and Technology

Customer Driven Project
November 20, 2012
Team: Ivo Dlouhy, Martin Havig, Øystein Heimark, Oddvar Hungnes

Contents

Appendix L

Introduction

This document describes deployment and testing process of Wonsole application. All other information about this software are available in detailed project report.

Appendix M

Required software

Some software is required to run the Wonsole.

M.1 Web Server

The client part is a web-application that is server by a web server. Any web server that can serve static files can fulfill this role. In our installation, we will use Apache web server.

M.2 Web Browser

The user needs a generic web browser to view the files and run the scripts. Any browser that supports javascript should work. For example refer to Mozilla Firefox [1] or Google Chrome [2]. System has been tested with Mozilla Firefox 16.0 and Google Chrome 23.0.

M.3 CouchDB

The backend server part is represented by CouchDB database system. No other server software is required, client connects directly to the database system. The system is tested to work with Apache CouchDB version 1.0.3 and 1.2. It should work with any version supporting the Bulk Documents API [3].

[1] http://www.mozilla.org/en-US/firefox
[2] https://www.google.com/chrome
[3] http://wiki.apache.org/couchdb/HTTP_Bulk_Document_API

Appendix N

Installation and configuration

This chapter describes installation and configuration of the tools required to run this software. As long as you can install all the required software, choice of the system is up to you. The reference and only supported system by this manual is be Red Hat Enterprise Linux 6.3.

N.1 CouchDB

Easiest way to install Apache CouchDB is to enable the EPEL - Extra Packages for Enterprise Linux repository. For more information on how to do that please refer to the offictial guide available on `http://fedoraproject.org/wiki/EPEL`. After setup of the repository, couchdb package should be available for install. Install this package with all the dependencies.

```
yum install couchdb
```

After installation, we should setup the CouchDB. The default setup file location is `file:///etc/couchdb/local.ini`. We will be using CouchDB directly as a backend, without a server middleware so we need to set it up to listen on publicly accesible interface and port. The interface bind address can set using variable $bind_address$, universal choice is to set it up as 0.0.0.0 so that the database listens on all available interfaces. The other interesting option is the port, that can be set up using the port variable. CouchDB needs to be available from client side over the network, so choose an interface and port combination, that is allowed on your firewall.

```
port = 5984
bind_address = 0.0.0.0
```

After configuration, start the couchdb service with command

```
service couchdb start
```

You can verify if the service is running using:

```
service couchdb status
```

You should get the feedback similar to

```
couchdb (pid  29915) is running...
```

N.2 Web Server

As a reference web server for this installation, we will use Apache 2.2.15. It can be installed using command

```
yum install httpd
```

By default, the web server is configured to use the directory /var/www/html/ as a root for serving the necessary files. Copy all of the files from folder www from the installation package to folder /var/www/html on your web server. The web server service httpd can be started using command:

```
service httpd start
```

Appendix O

Appendices

O.1 Installation Package Contents

```
wonsole-2.0/
  www/
    css/
    script/
    wonsole.html
  doc
```

Bibliography

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

[2] Anton Kovalyov. Why I forked JSLint to JSHint. "http://anton.kovalyov.net/2011/02/20/why-i-forked-jslint-to-jshint/", 2011. [Online; accessed 2012-10-10].

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

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

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

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

[7] M. Fowler. *UML Distilled, Third Edition*. Addison Wesley, 2004.

[8] Joyent, Inc. Node.js title page. "http://http://nodejs.org/", 2012. [Online; accessed 2012-10-10].

[9] JSLint. What is JSLint? "http://www.jslint.com/lint.html", 2012. [Online; accessed 2012-10-10].

[10] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6):42–50, November 1995.

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

[12] Microsoft. ASP.NET Get Started. "http://www.asp.net/get-started", 2012. [Online; accessed 2012-10-10].

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

[14] Jeffrey Palermo. *ASP.NET MVC in action with MvcContrib, NHibernate, and more*. Manning, Greenwich Conn, 2010.

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

[16] Ruby on Rails. About. "http://rubyonrails.org/", 2012. [Online; accessed 2012-10-10].

[17] Software & Systems Engineering Technical Committee of the IEEE Computer Society. IEEE Standard for Software Test Documentation, 2008.

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

[19] Cristof Strauch. Nosql databases.

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

[21] The PHP Group. What is PHP? "http://www.php.net/manual/en/intro-whatis.php", 2012. [Online; accessed 2012-10-10].

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