Wonsole

The new web console for power users

NTNU – Trondheim
Norwegian University of
Science and Technology

Customer Driven Project
October 10, 2012
Team: Ivo Dlouhy, Martin Havig, Øystein Heimark, Oddvar Hungnes

Abstract

Imagine the old minimachine systems where the end user works in a domain specific application in a black-and-green terminal. She is super efficient, jumping between windows with short cuts and everything is "in her fingers".

This is then replaced with a web-frontend and a mouse. Everything is "wrong", the design makes the usage patterns locked and she gets "mouse sickness" from point-clicking every command.

The task is to modify a web-application and add a scripting-console where the end-user can enter commands into a DSL - similar to the older interface. When commands are run, the results are shown both in the console and the web-interface. The use can still use the mouse and navigate as usual.

The target is a simple proof-of-concept in order to research any changes to the API, the potential of the method as well as evaluate the usability of scripting languages/DSL and API.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this report we will document our work. This includes our work progress, the technologies we used, our research findings and so on. In this intro section we will describe the project, the goals and briefly the involved parties. This chapter contains

- General information about ntnu and netlight
- General information about project
- Contact information on team members
- Goals
- Planned effort
- Schedule of results(Milestones, deliverables, sprint deadlines, etc)

1.1 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. [1]

1.2 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. [2]

1.3 General information about project

The project is the making of the course TDT4290 Customer Driven Project. 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. The customer assign the group a task which makes the project close to normal working life situation.

[1] http://www.ntnu.no
[2] http://www.netlight.com/en/

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

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.

Library is the chosen domain, and this will be used to explore and test the concept. The library domain is chosen since it possesses a power user, multiple forms which might be effictiviced 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.

1.4 Contact information

Contact information on the involved members of this project.

1.5 Goals

1. 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.
2. Investigate the ramifications of the added functionality, in terms of usability and technical aspects.
3. Provide extensive documentation and a successful presentation of the end product.

1.6 Planned Effort

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.

1.7 Schedule of Results

1.7.1 Deliverables

- 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 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.

Table 1.1: 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

1.7.2 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 1.1

Chapter 2

Project management

2.1 Project plan

2.1.1 Tool Selection

2.1.2 Concrete Project Work Plan

2.2 Project name

Project name is important project identificator. It should summarize main project goal or functionality. In real project, is often a trademark, or reflects the name of the company.

In this section, we will describe the process of choosing a name for the project. First of all, we made a list of words, that we can use in the project name. These words describe project funcionality or goal.

Words that can be used in project name: master, console, web, text, keyboard

We used the list of words and brainstorming session on a meeting for compiling a list of project name candidates:

Candidate project names: console 2.0, wonsole, wensole, websole, werminal, interCLI

After discussion we chose the name *Wonsole*. Project name can be sometimes little confusing, so we added the subtitle: *The new web console for power users.*

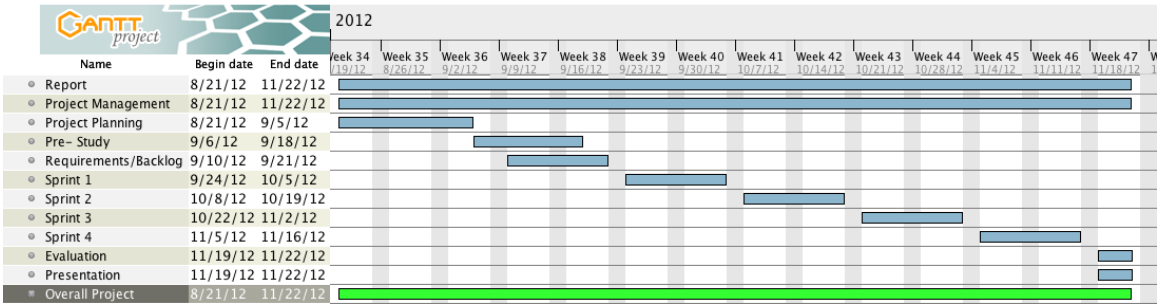


Figure 2.1: Gantt Chart

Table 2.1: Work Breakdown Structure

Task	From date	To date	Hours Est.	Act.
The report	21/08/2012	22/11/2012	300	
Project Management	21/08/2012	22/11/2012	180	
Project Planning	21/08/2012	05/09/2012	100	
Lectures	21/08/2012	05/09/2012	40	
Pre- Study	06/09/2012	18/09/2012	80	
Backlog	10/09/2012	21/09/2012	60	
Architecture	17/09/2012	21/09/2012	40	
Sprint 1	**24/09/2012**	**05/10/2012**	**130**	
Planning			20	
Design/Implementation			90	
Testing			20	
Sprint 2	**08/10/2012**	**19/10/2012**	**130**	
Planning			20	
Design/Implementation			90	
Testing			20	
Sprint 3	**22/10/2012**	**02/11/2012**	**140**	
Planning			20	
Design/Implementation			100	
Testing			20	
Sprint 4	**05/11/2012**	**16/11/2012**	**140**	
Planning			20	
Design/Implementation			100	
Testing			20	
Evaluation	19/11/2012	22/11/2012	30	
Presenation	19/11/2012	21/11/2012	30	
Total			**1400**	

2.3 Team Structure

2.3.1 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		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

2.4 Quality Assurance

2.4.1 Communication rules

To prevent misunderstanding in communication and

meeting	scheduled at least 48 hours before, confirmation within 24 hours before meeting required
documents for the meeting	24 hours before
email communication	contact person directly, google groups is just internal
response from the team	within 8 hours
response from customer	within 24 hours

Table 2.2: Communication rules

#	**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 2.3: Risks 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 2.4: 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 2.5: 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 2.6: 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 2.7: 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 2.8: 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 2.9: 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 2.10: 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 2.11: Risk 08

2.5 Risks

2.6 Meetings

2.7 Lectures

2.8 Issues

2.9 Scrum

2.10 Domain choosing

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 2.12: 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 2.13: 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 2.14: 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 2.15: 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 2.16: 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 2.17: Risk 14

Chapter 3

Preliminary Study

3.1 Concept

3.2 Constraints

3.2.1 Time

The project must be completed within a timeframe of 13.6 weeks. The guideline is 24.3 work hours per person, per week.

3.2.2 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 Feasibility study

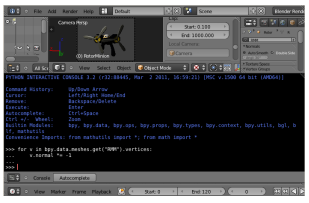
3.4 Similar solutions

3.4.1 Counter-Strike

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

In a game, this functionality eases development and debugging, as well as increasing the moddability and long-term value for players. Similar consoles also exist in other games, such as Carmageddon TDR 2000.

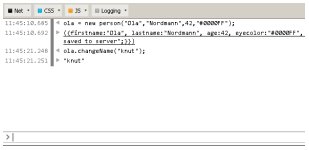
3.4.2 Blender



This 3D modelling software features a Python console with access to the model data, animation data, etc. It can be used to perform operations that are not directly supported in the user interface. It is also common to extend the program using custom Python scripts.

Similar solutions also exist in other creative tools, including a Python console in GIMP and Nyquist prompt in Audacity.

3.4.3 Firefox

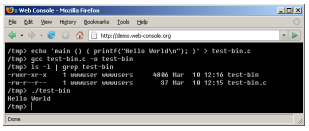


This web browser features a Web Console where it is possible to execute JavaScript commands with access to the objects on the current page. This is useful for web development and debugging. Similar features also exist in a few other browsers, such as Google Chrome.

3.4.4 try.mongodb.org

This database website features a console where the user can execute commands on a dummy database. It is an educational tool for people who don't want to invest too much time in it. Comparable consoles also exist to allow the user to execute arbitrary queries in database administration tools such as PHPMyAdmin.

3.4.5 web-console.org



This project provides shell access to a server through the browser window over HTTP.

It supports real time communication, and is intended to be used for server administration purposes.

3.4.6 Conclusion

This is by no means an exhaustive list, but it serves to illustrate that consoles are still applicable for purposes including software development, debugging, learning, extendability and where there is a high demand for flexibility.

3.5 Version control

3.5.1 git

3.6 Development language and technologies

3.6.1 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 multimedia 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.

3.6.2 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

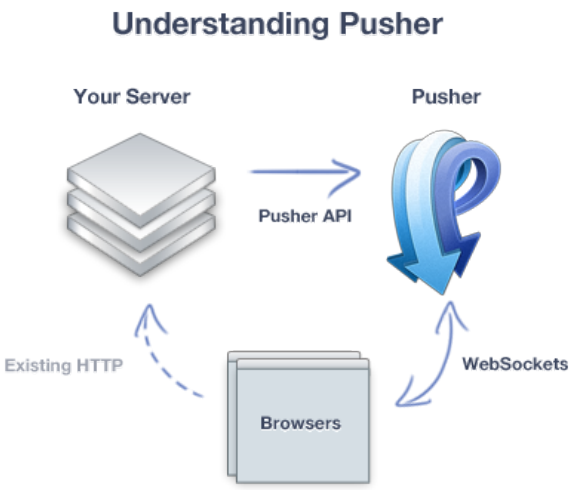


Figure 3.1: Pusher Explained

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.

3.6.3 Markup Languages

When communicating between the client and the server we need a data exchange format to represent objects and actions. The format has to be able to serialize and deserialize them on sending and receiving.

The two alternatives most commonly in use today for solving this problem is XML (Extensive Markup Language) and JSON (JavaScript Object Notation).

XML

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

JSON

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

Conclusion

Both languages is well supported in almost all web related libraries, and they are both extensively documented. As security is not a main concern of this project, the fact that XML is more secure will not influence the decision

JSON will be used for this project. It covers the functionality we need, and it is generally thought of as the easier language to use. It is perfectly suited for web- development and JavaScript, which is the domain of this project. In addition the developers have more experience in using JSON than XML.

3.6.4 Google Drive

3.7 Development Methodology

3.7.1 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.

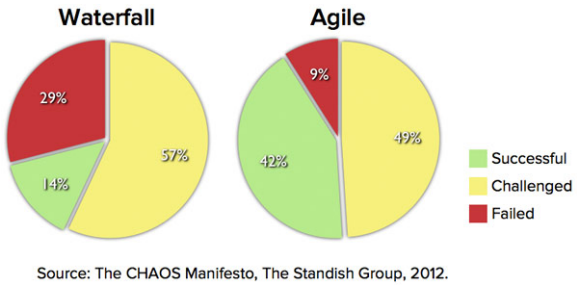


Figure 3.2: Waterfall vs. Agile

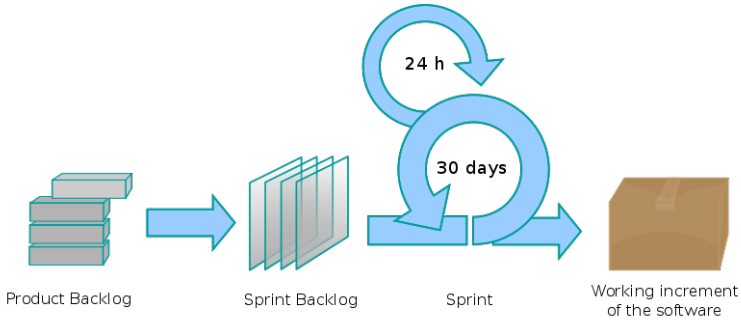


Figure 3.3: Scrum Process

As the Figure 3.3 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).

3.7.2 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.

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. [1]

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. [2]

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

[1] http://en.wikipedia.org/wiki/Scrum_(development)
[2] http://en.wikipedia.org/wiki/DSDM

a unified client viewpoint and focus on coding rather than documentation of compromise objectives and constraints. [3]

3.7.3 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 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.

3.8 Software Testing

3.8.1 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.

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

[3] http://en.wikipedia.org/wiki/Extreme_Programming

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.

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.

3.8.2 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.

3.8.3 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.

3.9 Code conventions

JSLint

Chapter 4

Requirements

4.1 User stories

We compiled list of user stories from a few points of view. Administrator section describes mostly detailed technical requirements. General section contains stories applicable on general problem. Domain section contains domain specific user stories.

4.1.1 Developer point of view (section A)

1. As a developer, I want to be able to store objects in a persistent database, so that I can migrate data easily.
2. As a developer, 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.
3. As a developer, I want to be able to work directly with object attributes rather than any form of raw data.

4.1.2 User point of view - General (section G)

1. 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
2. As a user, I want my changes to be propagated to other users of the system real- time
3. 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.
4. 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.
5. As a user, I want access to a tutorial, so that I can learn to work with the system easily.
6. 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.
7. As a user, I want to be able to easily repeat and edit last command, so that I can use it on another object.
8. 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.

4.1.3 User point of view - Domain specific (section D)

1. As a user, I want to be able to add a new book to the system using both the console and graphical interface.
2. As a user, I want to be able to delete a book in the system using both the console and graphical interface.
3. 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.
4. 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.
5. 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.
6. As a user, I want to be able to registrate 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. As a user, I want to be able to order new books for the library using both the console and the graphical interface.

4.1.4 Planning

4.2 Sequence Diagrams

4.3 Prioritization

4.4 Functional Requirements

4.5 Nonfunctional Requirements

4.6 Test Plan

This chapter will introduce the overall test plan for this project, and it is based on the IEEE 829 Standard for Software Test Documentation[link to reference]. 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.

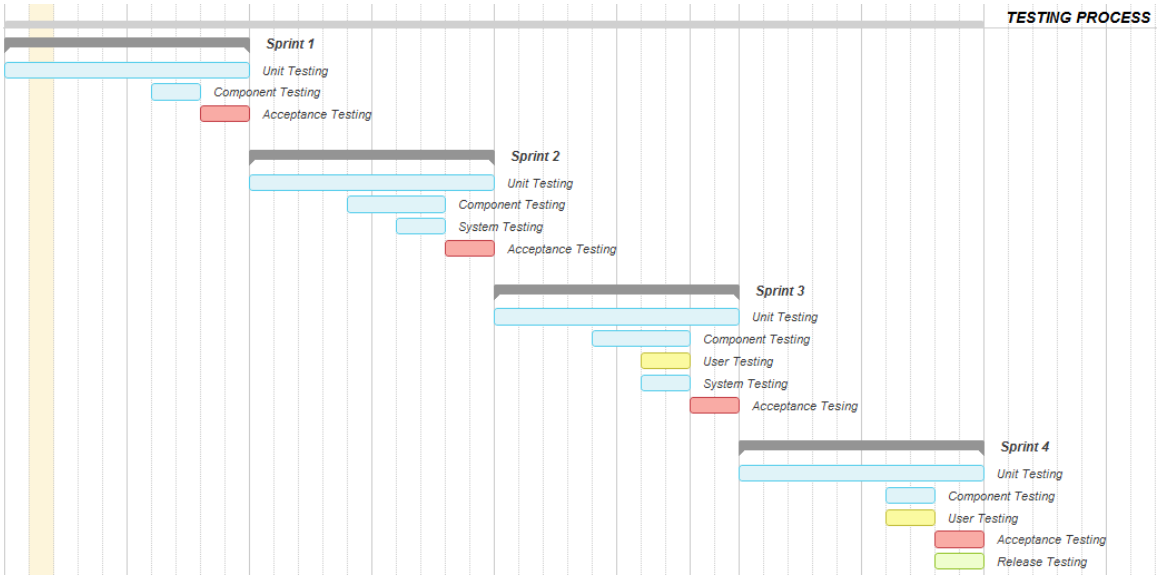


Figure 4.1: Testing Process Timeline

4.6.1 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.

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).

Testing Process Timeline

Figure 4.1 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.

Table 4.1: 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 4.2: 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

4.6.2 Templates

The templates stated in Table 4.1 and Table 4.2 will be used to create test cases and to record the results of them.

4.6.3 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.

4.6.4 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 5

Architecture

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.

5.1 Architectural drivers

The architectural drivers defines the project and its scope.

- Improving the efficiency of the chosen system - The existing system is to improved efficiency vise, so power users will be able to use less time on more tasks.
- Proving the concept - Showing that a system can become more efficient through the use of a console.
- Modifiability - The system will be constructed for a test concept as a prototype, so further development and maintainability should be made easy for outsiders. It should also be possible to edit the system structure under the process of making it without starting from scratch.
- Reach goal from the customer - The customer has a thought of how the system is to look like after we have done our part, we must satisfy these goals and not stray too far from the red thread.

5.2 Stakeholders

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

- Developer
  - 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

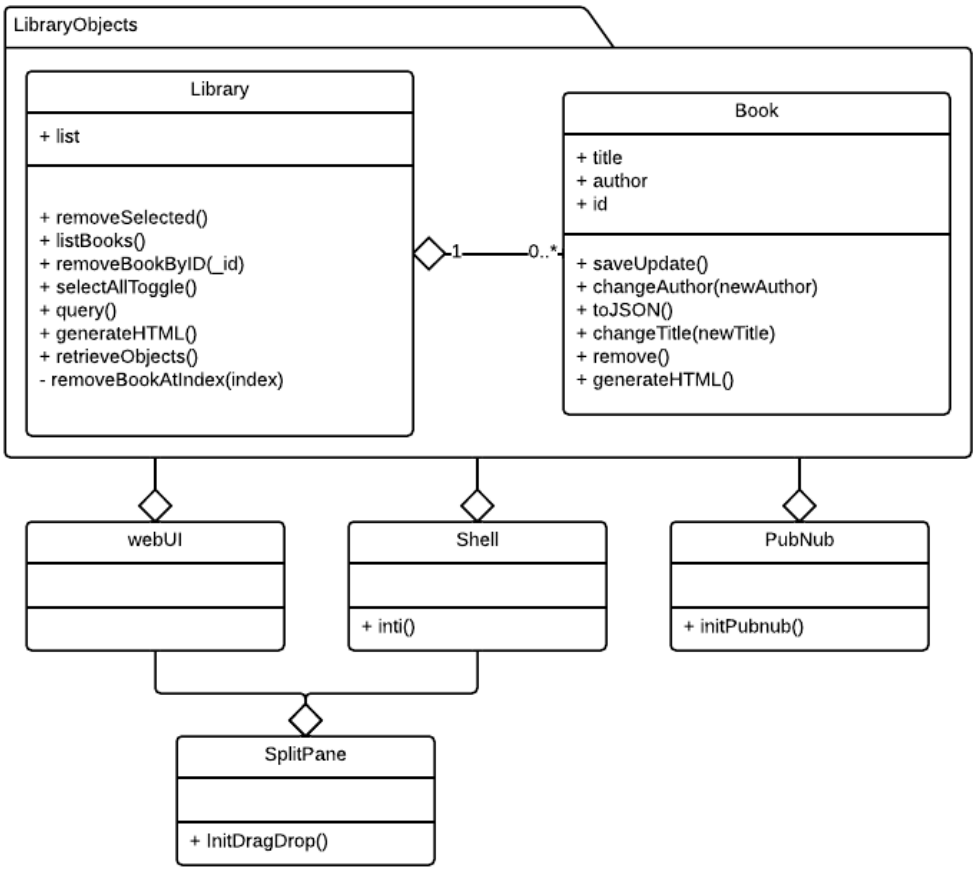


Figure 5.1: Client Class Diagram

5.3 4+1 view model

The 4+1 view model. Here the views will be described, and how they will look in our architecture.

5.3.1 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 6.1 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 6.2 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.

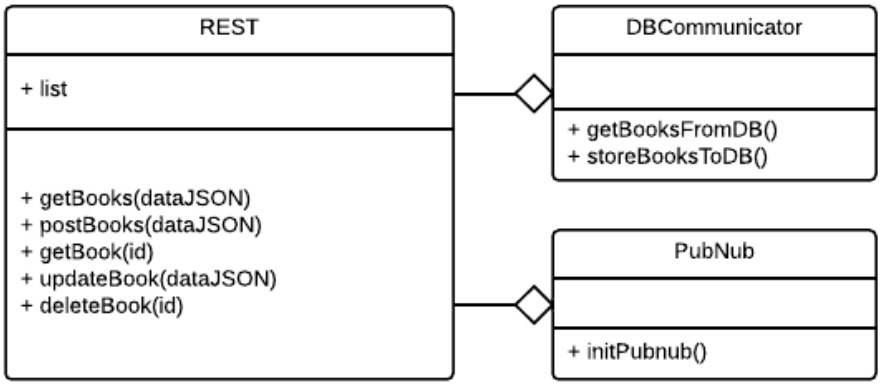


Figure 5.2: Client Class Diagram

5.3.2 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 6.3 These components form a three layered structure, and communicate with each other through the neighboring layer.

5.3.3 Process View

Describes the dynamic aspect of the system, and explains how the different parts of the system will communicate at runtime. This is described with a activity diagram.

The user will ask for an object from the backend, this will be delivered to the client through the communication channel as a json object, the client will interpret this and the user can then edit it through the console, and send it back to the backend.

5.3.4 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 6.4 The structure of the four different parts of the system.

5.4 Tactics

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

5.4.1 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

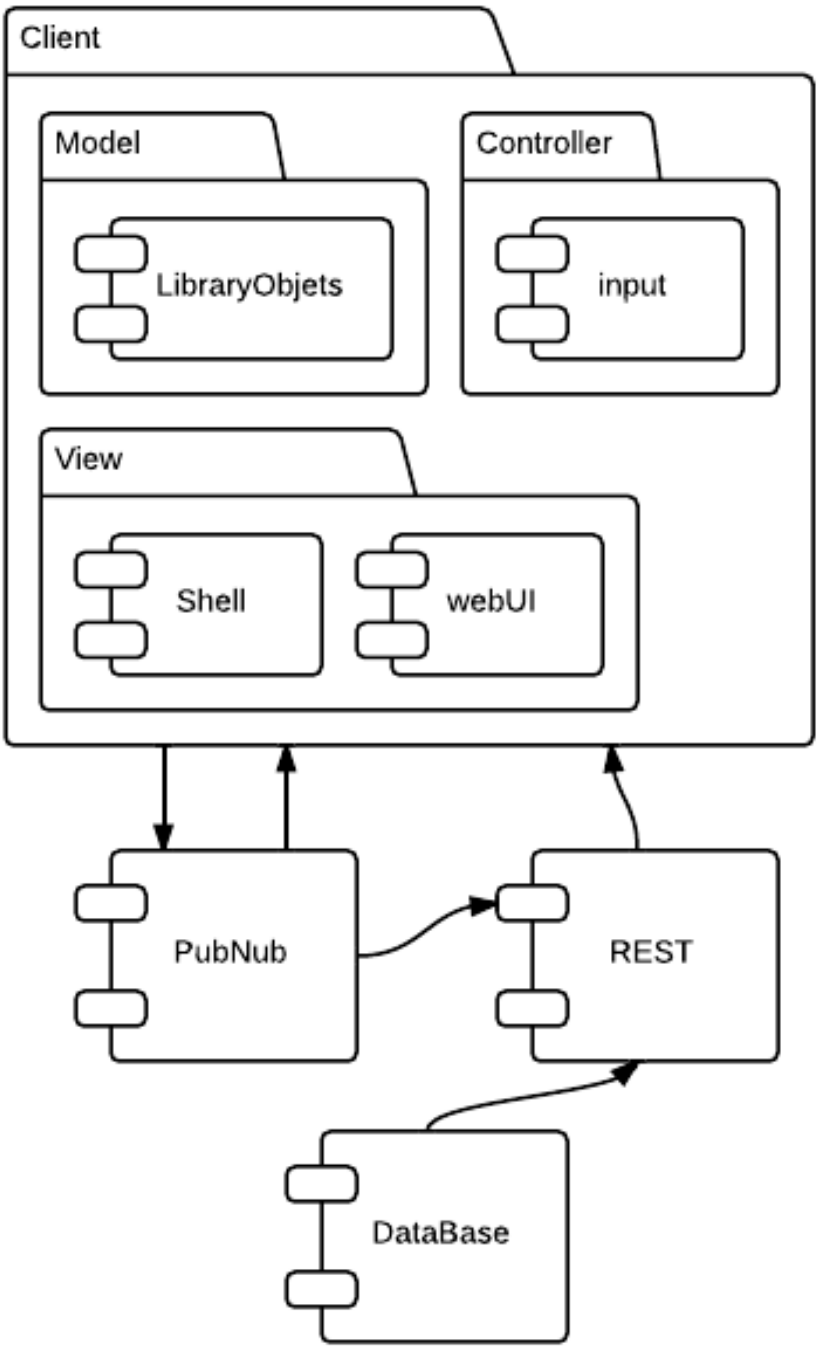


Figure 5.3: Component Diagram

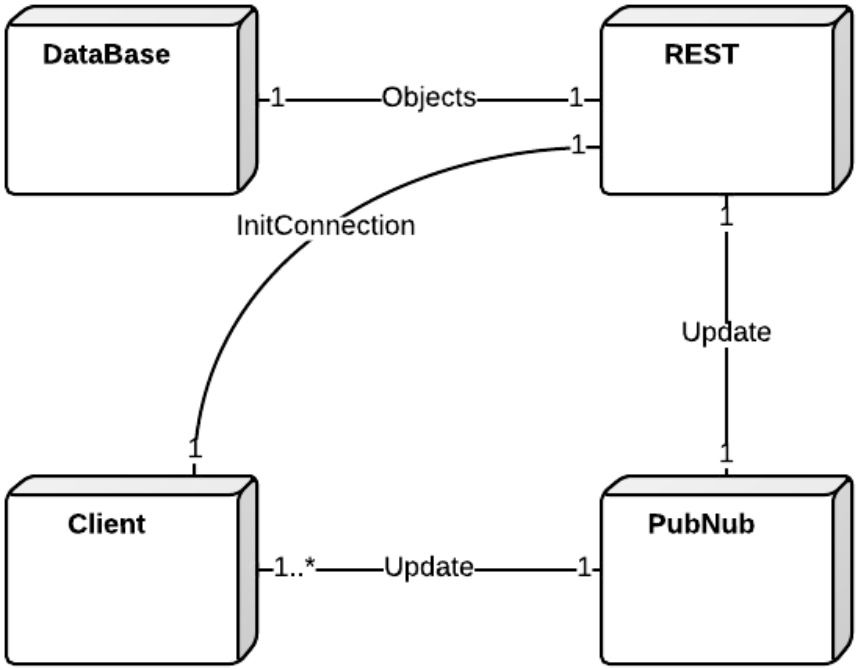


Figure 5.4: Deployment Diagram

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.

5.5 Architectural patterns

The patterns can help solve different kinds of problems with know solutions on new problems.

5.5.1 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

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.

figure of the 3 tiers.

5.5.2 MVC

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.

5.5.3 Rationale

5.6 Database

5.7 GUI

Chapter 6

Sprint 1

6.1 Planning

We started the sprint with a sprint planning meeting September 24th. The plan for this sprint is to implement a basic implementation of the system, so that we 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.

6.1.1 Duration

This sprint started on September 24th and will last for two weeks. A customer demo will be held at October 04th to show of what we have achieved during the sprint and to ensure that the customer agrees with the implementation.

6.1.2 Sprint Goal

The goal for the first sprint is to get a basic version of the library application working. This includes 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.

6.1.3 User stories

The user stories we chose to implement in this sprint with their estimated workload is listed in Table 6.1. We assume that we have around 170 hours a sprint for working with the actual user stories. The remaining 30 hours will 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 just code and hope for the best. 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.

6.2 Architecture

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

6.2.1 4+1 view model

The 4+1 view model. Here the views will be described, 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 6.1 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 6.2 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.

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

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 6.4 The structure of the four different parts of the system.

6.3 Implementation

6.3.1 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

Table 6.1: Sprint 1 User Stories

User Story	Short Description	Hours Est.	Hours Act.
A1	**Store objects in database**	**35**	**44**
	Design	8	6
	Implementation	18	29
	Testing	5	4
	Documentation	4	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**	**40**	**38**
	Design	14	10
	Implementation	18	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:	**170**	**151**

Table 6.2: Sprint 1 Workload

Task	Hours Est.	Hours Act.
Design	47	28
Implementation	82	88
Testing	23	16
Documentation	18	19
Total	**170**	**151**

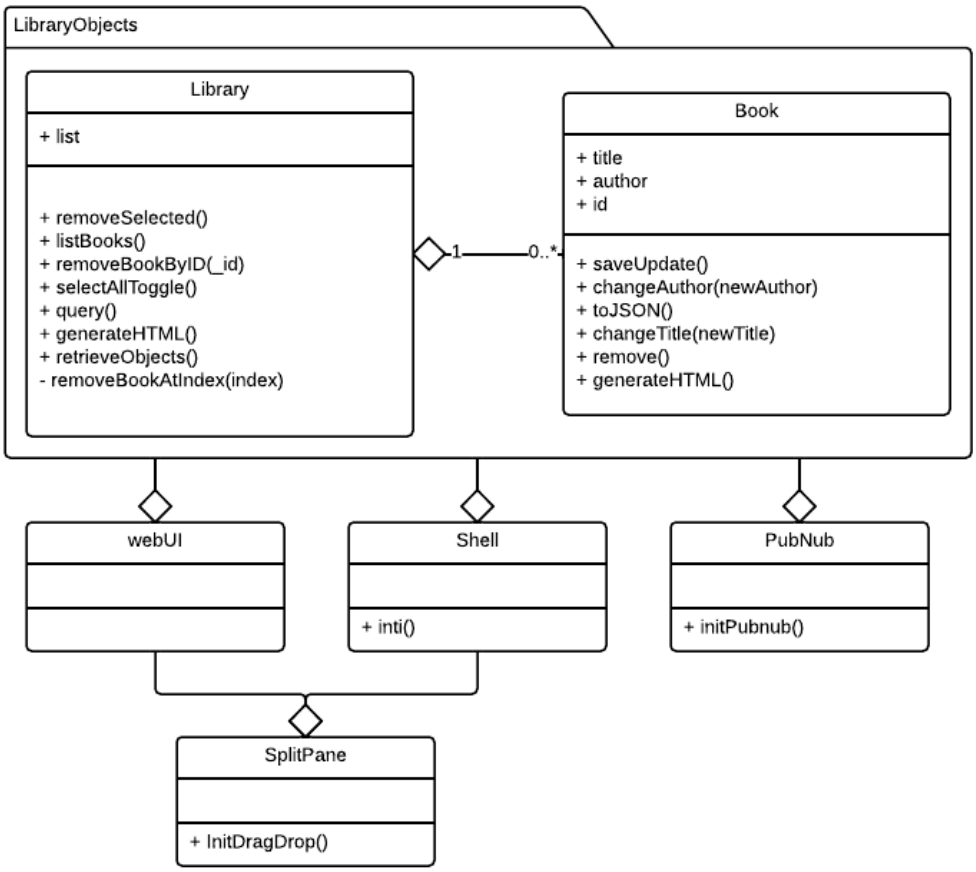


Figure 6.1: Client Class Diagram

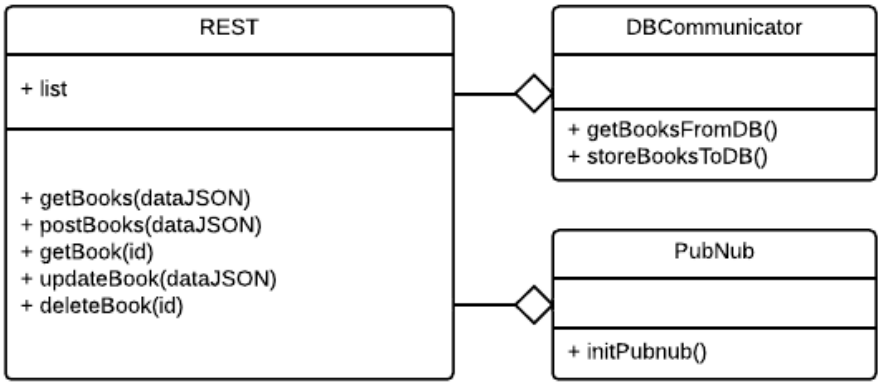


Figure 6.2: Client Class Diagram

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.

Below each resource is explained in detail, in addition example code with jQuery(JavaScript) will be supplied.

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",
```

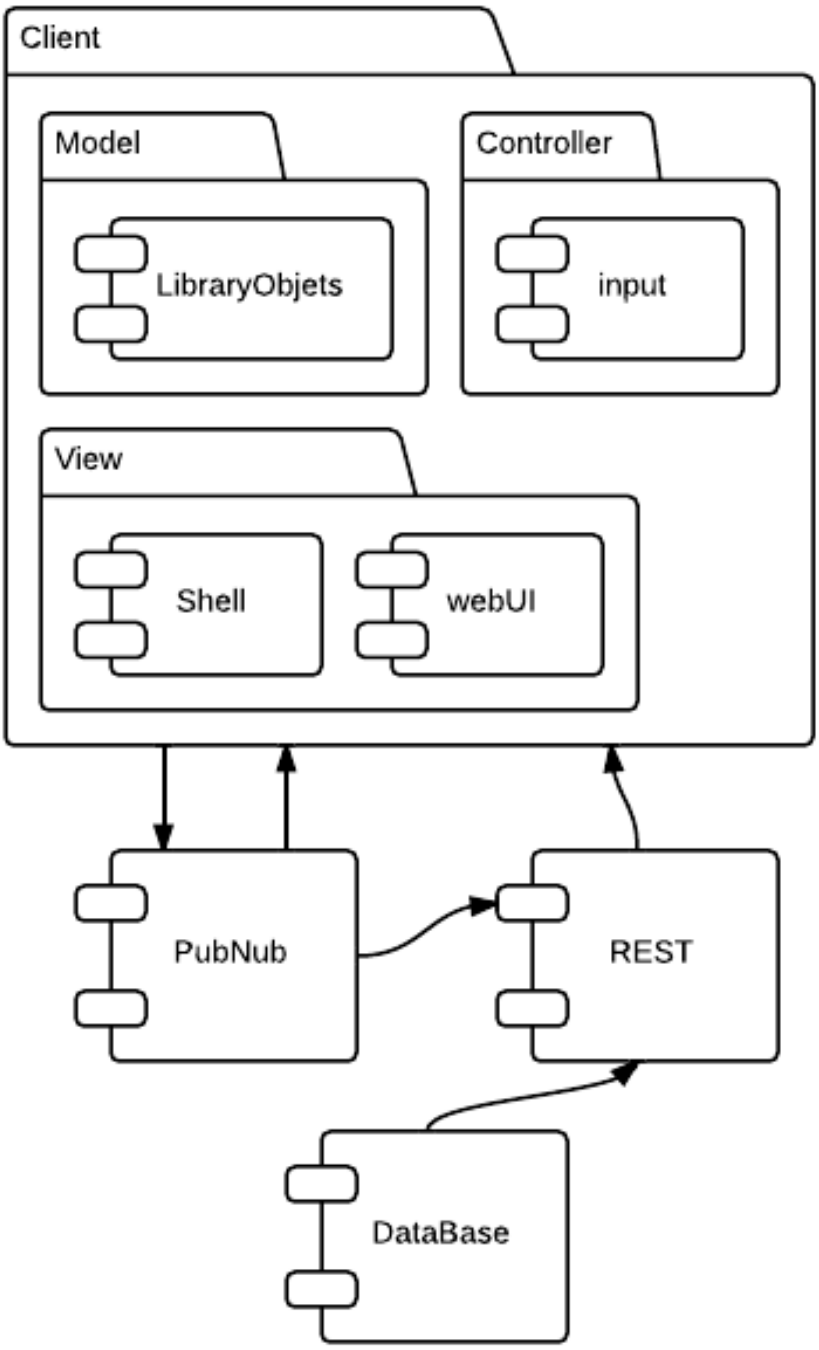


Figure 6.3: Component Diagram

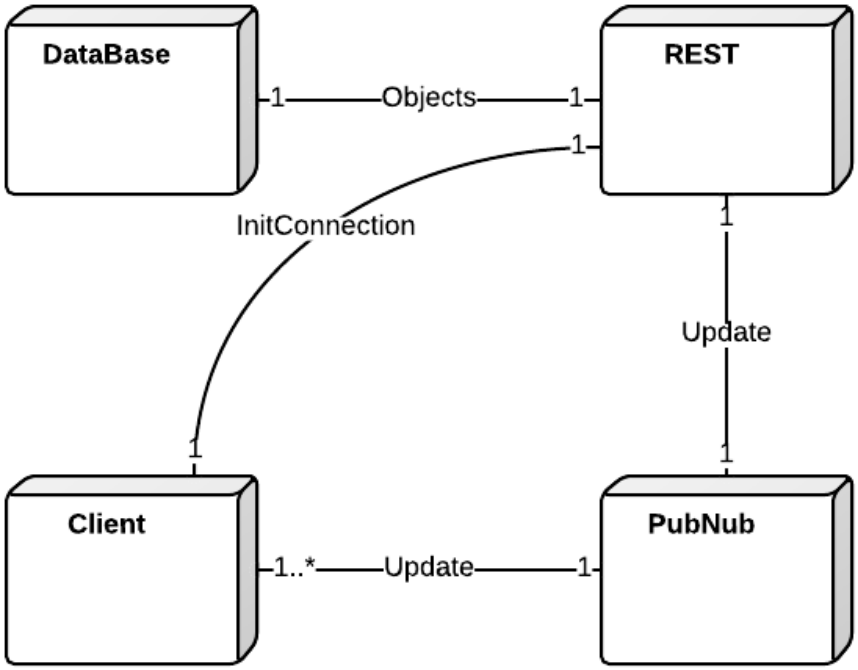


Figure 6.4: Deployment Diagram

```
        "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'
});
```

6.4 Testing

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

6.4.1 Test Results

We performed a total of 10 test cases during this sprint; TID01-10. The results are listed in Table **??**. The test cases themselves can be found in appendix(TBA).

6.4.2 Test Evaluation

All our tests passed on the first attempt, without any comlications 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.

Table 6.3: 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

6.5 Customer Feedback

6.6 Evaluation

6.6.1 Review

6.6.2 Positive

- We delivered a functional system for the demo, with all the promised functionality.
- The customer was happy with the product and our progress so far.
- We finished all the user stories, and even additional ones we didn't plan for
- We divided the work amongst the group members in a good way
- Conducted a good planning meeting on the first Monday, and we got the customer included in the prioritization of user stories.
- We received valuable feedback from customer on the demonstration.

6.6.3 Negative

- We didn't do a lot of design up front, and ended up using more time on implementation than we planned for.
- We weren't good enough to document our work in the report during the sprint. This lead to some work that had to be done after the sprint was finished.
- We are not using the time tracking system properly at the moment. As a result the documentation of work hours was more time consuming than necessary.
- Problems with the burndown chart
- We failed to use the scrum process properly, as we didn't divide the user stories into tasks, and weren't good enough to facilitate daily scrum meetings.

6.6.4 Planned Responses

- Do more design up front, before we start to implemen
- Break down the user stories into smaller tasks, so its easier to estimate the workload and assign them to different persons.
- Facilitate the daily Scrum meetings
- Do more documentation during the sprint, to avoid extra work after the sprint is finished.
- Fix the burndown chart plugin, and start using it as a tool in the development process
- Try to log our work hours more accurate, to make it easier to document them afterwards

Chapter 7

Sprint 2

7.1 Planning

7.1.1 Duration

7.1.2 Sprint Goal

7.1.3 User Stories

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.

110 hours available. Less than normal as we have to focus on pre- delivery of the report on October 14th.

Table 7.1: Sprint 2 User Stories

User Story	Short Description	Hours Est.	Hours Act.
A4	**Update to schemaless database**	**10**	
	Change the server implemenation	4	
	Update REST API calls	2	
	Testing	2	
	Documentation	2	
G4	**Reflect actions from the GUI in the console**	**11**	
	Print the commands in the console	6	
	Testing	3	
	Documentation	2	
G7	**Autocompletion of commands**	**30**	
	Add methods to list commands	4	
	Create popup menu	10	
	Update elements in menu	4	
	Make selection of commands possible	3	
	Autocomplete on selection	1	
	Testing	4	
	Documentation	4	
G10	**Indicate the selected objects in GUI**	**26**	
	Design soulution for highlighting elements	2	
	Create method for highlighting	2	
	Create record for selected objects	4	
	Respond to changes in selection	8	
	Highlight newly created objects	2	
	Testing	4	
	Documentation	4	
G11	**Cycle through the current selection**	**25**	
	Logic to keep track of selection	5	
	Update this list when selection changes	4	
	Extract objects when cycling through	7	
	Update the GUI when selection changes	2	
	Testing	4	
	Documentation	3	
G8	**Show details on click**	**8**	
	Make ID clickable	1	
	Update relevant section on click	2	
	Able the user to make changes	3	
	Testing	1	
	Documentation	1	
	Total:	**110**	

Table 7.2: Sprint 2 Workload

Task	Hours Est.	Hours Act.
Design	43	
Implementation	63	
Testing	19	
Documentation	15	
Total	**140**	

7.2 Architecture

7.2.1 4+1 view model

Logical View

Development View

Process View

Physical View

7.3 Implementation

7.4 Testing

7.4.1 Test Results

7.4.2 Test Evaluation

7.5 Customer Feedback

7.6 Evaluation

7.6.1 Review

7.6.2 Positive

7.6.3 Negative

7.6.4 Planned Responses

Chapter 8

Sprint 3

8.1 Planning

8.1.1 Duration

8.1.2 Sprint Goal

8.1.3 User stories

8.2 Architecture

8.2.1 4+1 view model

Logical View

Development View

Process View

Physical View

8.3 Implementation

8.4 Testing

8.4.1 Test Results

8.4.2 Test Evaluation

8.5 Customer Feedback

8.6 Evaluation

8.6.1 Review

8.6.2 Positive

8.6.3 Negative

8.6.4 Planned Responses

Chapter 9

Sprint 4

9.1 Planning

9.1.1 Duration

9.1.2 Sprint Goal

9.1.3 User stories

9.2 Architecture

9.2.1 4+1 view model

Logical View

Development View

Process View

Physical View

9.3 Implementation

9.4 Testing

9.4.1 Test Results

9.4.2 Test Evaluation

9.5 Customer Feedback

9.6 Evaluation

9.6.1 Review

9.6.2 Positive

9.6.3 Negative

9.6.4 Planned Responses

Chapter 10

Conclusion

Chapter 11

Appendices

11.1 Test Cases

11.1.1 Sprint 1

Table 11.1: 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 11.2: 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 11.3: 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 11.4: 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 11.5: 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 11.6: 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 11.7: 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 11.8: 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 11.9: 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 11.10: 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.