Twilm

ONELINER ABOUT THE PROJECT HERE

NTNU – Trondheim
Norwegian University of
Science and Technology

SPECIALIZATION PROJECT? =P
November 21, 2013
Team: Fredrik Persen Fostvedt and Martin Havig
Supervisor: Heri Ramampiaro

Acknowledgements

Thanks Mum!

Abstract

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

Some awesome abstract, example:

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

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

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

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

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

Acknowledgments

Some awesome Acknowledgments

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

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

Preface

Some awesome preface, example:

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

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

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

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

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

Denne oppgaven dreier seg om filmanbefalinger basert på sosiale media og sosialnettverk. Vi skal ta i utangspunkt i en samling av data om brukere, deres nettverk, samt en samling av filmer med tilhørende "ratings", og utvikle en tekst- og datagruvedriftmetode ("text and data mining method") og/eller maskinlæringsmetode for å produsere gode anbefalinger.

Oppgaven vil kreve at man setter seg inn i metoder for å hente ut kunnskap fra en samling av ustrukturerte data (i.e. Information and knowledge extraction). Det er en stor fordel med en god programmering- og algoritmiskforståelse.

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

1.1 Purpose

1.2 Motivation

1.3 Context

1.4 Project

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

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

Goals

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

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

1.5 Structure of Report

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

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

Table 1.1: Structure of Report

Chapter 2

Preliminary Study

Contents

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

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

This chapter is ment to outline the preliminary study of our project. This includes - what our technologies aims to achieve - how we will use them to

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

2.1 Movie recommendation

This is the prestudy for the movie recommendation part of the system. We will look at existing recommendations systems, datasets to be explored and how to explore this set.

2.1.1 Netflix

Netflix is a on-demand Internet streaming media. It started out as a DVD-rental business, but moved over to a more Internet based business model in 1999 and have from then on had a great success in the subscription-based digital distribution service. To improve customer satisfaction they developed a personalized video-recommendation system called Cinematch. About 60% of the Neflix users select their next movie or TV-show based on this recommendation[————————2————————], so it is important that this recommendation system manages to capture the users movie and TV-show preferences and feed back fitting movies and TV-shows.

Cinematch

This recommendation system is self-updating. It searches the Cinematch database for users who have rated the same movie, determents which of these again have rated a second movie and with this calculates likelihood that users who like the first also likes the second one.

2.1.2 The Dataset

Netflix held a competition where the contest was made to help Netflix improve their movie recommendation system. The team that could beat Netflix's own collaborative filtering system by more than 10% could win one million dollars. To measure the score root-mean-square deviation (RMSE) was used, which is used to measure the difference between the values predicted or estimated and the actual values observed. The RMSE of Cinematch 2.1.1 was at 0,9525.

For this competition Netflix produced a training dataset based on user ratings from their own database. It contains 100 480 507 ratings from 480 189 unique users on 17 770 movies. The data is split up into 17 770 files, one for each movie, where the data is a triplet of the form <user, rating, date-of-rating>. The user-field and rating-field is an integer, while the date-of-rating-field is on the ISO 8601 form, year-month-day. The different movies have different amounts of ratings, and the different users have rated different amounts of movies.

With the training dataset the teams system was supposed to predict ratings on a "qualifying"-dataset, which contained 2 817 131 triplets with only user ids, movie ids and dates. This "qualifying"-datase must be disregarded since the actual ratings for this dataset was only know to the jury of the competition and is nowhere to be found. Instead a probe set is provided with 1 408 385 users

which can be used to remove these from the training set to test predictions based on the test set. This probe set possesses similar statistical values as the qualifying set, so it will give a good prediction of how the result actually would have scored in the competition.

Ratings	100 480 507
Customers	480 189
Movies	17 770
Qualifying set	2 817 131
Probe set	1 408 385

Table 2.1: Dataset statistics

Dataset statistics

To better understand the dataset, some statistics about the dataset was produced.

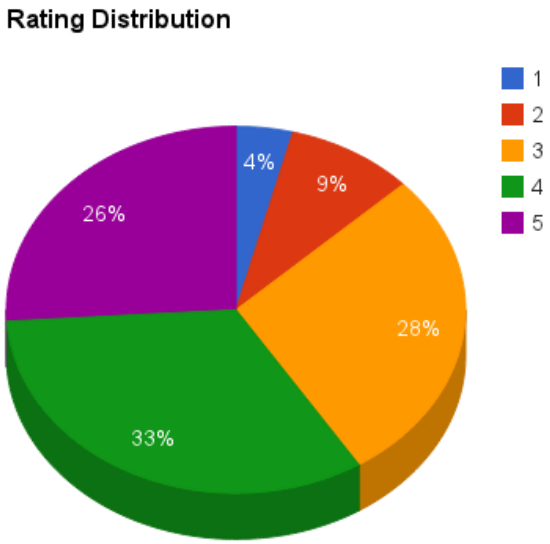

Figure 2.1: Rating distribution

Rating distribution is shown in figure 2.1. 1/3 of the ratings are 4, and rating 1 and 2 only makes up for 13% of the rating distribution. 3 and 5 are quite similarly represented with 28% and 26% respectively. It can from this

be assumed a greater prediction towards the higher ratings, and predictions towards lower ratings should not be done lightly.

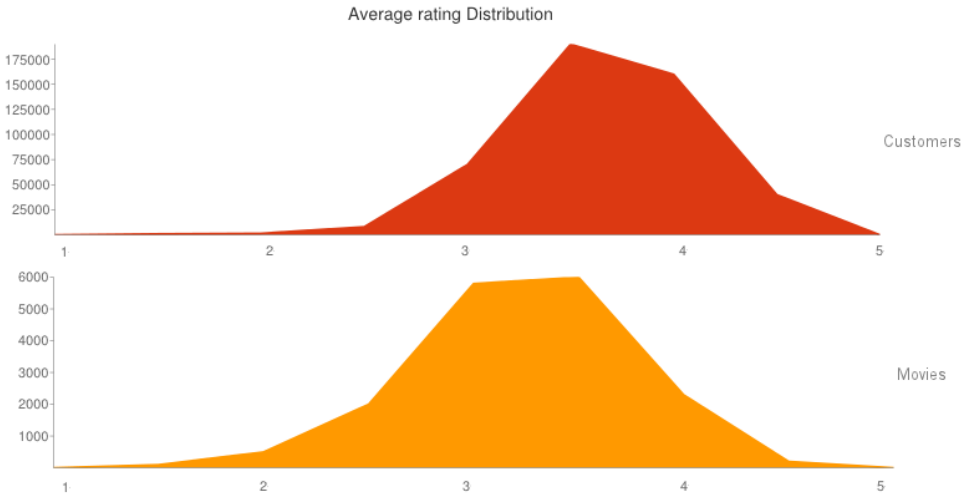

Figure 2.2: Average rating distribution

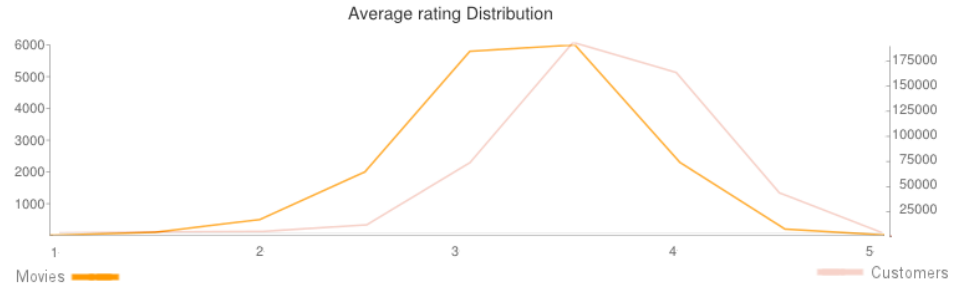

Figure 2.3: Average rating distribution

Average rating distribution is shown in figure 2.2. The upper graph shows average rating distribution of the customers, while the bottom one shows the distribution per movie. Both graphs have quite similar form. Both customers and movies peak at a rating of 3,5, which is natural. The second second highest point is for the customers at 4 and movies at 3, which is interesting to see. TO ANALYSE MORE!

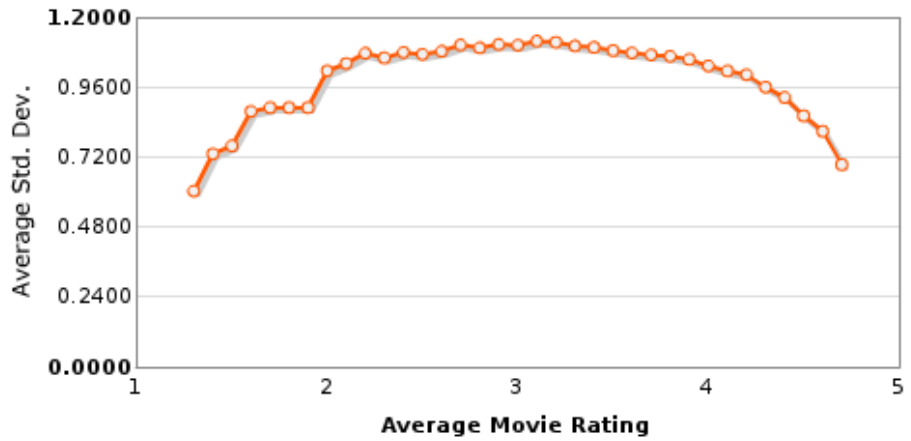

Figure 2.4: Average movie rating distribution

Average movie rating distribution is shown in figure 2.4. Here we see that for the edge ratings (rating 1 or 5) has the least standard deviation, so when a movie is highly rated it usually is rated high, and the same goes for low ratings, while for the middle ratings (2 to 4) we see that it variates more, and quite similarly. So the middle ratings are harder to classify, in other words, when a movie has a rating of 3, the average rating-shift is almost 1.1, so it must be expected that there is a bigger classification hit on these, so the problem will be to fin the right classification.

2.1.3 Classification

TODO: There are two basic problems when analysing very large datasets: - Writing the code correctly. - The performance (i.e. speed at which it runs).

Something about why and which classification alg to use

2.1.4 What the top Netflix-Prize contestants did

BellKor

asdf

Conclusion

2.1.5 Algorithms to solve the problem

There were 51 051 contestants on 41 305 different teams competing in the Netflix-Prize competition, and a lot of these contestants shared their solution

through code repositories such as GitHub (————something about github here?————). Since many of these solution scored very well on the contest it was natural to explore these and use their solution to test potential improvement in collaboration with social media data. Here we explore some of the better solutions and how and why these could be used to our benefit.

https://github.com/n8j/school/tree/master/DataMiningCSEP546/HW1

BellKor's Pragmatic Chaos - RMSE = 0.8567

2.2 Weka

Weka is a machine learning software written in Java. It can be used to visualize data through analyzing it with different kinds of algorithms. The Explorer, a graphical user interface, makes it easy to work with and explore data from different angles. It is a free software under the General Public License []. Weka can handle several data mining tasks, including, data preprocessing, clustering, classification, regression and feature selection. The input data has to be a single flat file or relation, each data point must be described by a fixes number of attributes. The dataset can be accessed trough a SQL database. Can link database tables into a single datatable, which can be used for processing using Weka.

2.2.1 The Explorer

TODO: TABLE OF EXPLORER PROPERTIES? TOO MUCH?

2.2.2 ARFF

Attribute Relationship File Format is the file type used to store data in a database. The structure is as follows:

```
1  @relation 'flat-netflix'
2
3  @attribute assetId string
4  @attribute 14367 numeric
5  @attribute 6050 numeric
6  @attribute 14112 numeric
7
8  @data
9  1658496,?,?,4
10 684232,3,3,?
11 684231,4,?,?
12 1774519,1,?,?
```

Listing 2.1: ARFF Example

Listing 2.1 is an example of how an ARFF file is structured. It startes with a header, containing "@relation", "@attribute"s and the "@data". This header defines the name of the relation, the attributes in the dataset and the data. The "@data"-header contains arrays, where each array has values for the corresponding "@attribute"s values. If a array has no value for a particular attribute it is denoted as a question mark (?).

2.3 Twitter

TODO: Describes the existing stuff from twitter and twitter-api

2.3.1 Twitter-API

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

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

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

2.5 Development language and technologies (maybe not needed)

2.5.1 Database

For the system we need some kind of persistent storage, a database where the harvested data from the Twitter-harvesting is stored so it only is needed to fetch once, and where the data from the Netflix-prize set is stored. This Database will be ran from a personal computer to start with, TODO: SOMETHING ABOUT THE HARVEST IS DONE; OR REFERING TO THE TWITTER CHAPTER, since it is not needed to be constantly running. For the database we are left with a decision between traditional SQL database or a so called NoSQL database. The differences are explained below.

NoSQL

NoSQL databases can be defined as: being non-relational, distributed, open-source and horizontally scalable [8]. NoSQL is meant for storing large amounts of data [14], where all the data does not share the same attributes or schema. They are focusing on eventually consisten(BASE) instead of the SQL ACID [9]. The different kinds of NoSQL databases have different ways of classifying the

Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key-value	high	high	high	none	variable (none)
Column	high	high	moderate	low	minimal
Document	high	variable (high)	high	low	variable (low)
Graph	variable	variable	high	high	graph theory
Relational	variable	variable	low	moderate	relational algebra

Table 2.2: Classification based on feature

database, some of the more used ones are: key-value, column, document, graph and relational.

As we can see from table 2.2 performance, scalability and flexibility are usually rated as high in all the categories. Relational algebra in the database will not be needed for the project to succeed so that category can be disregarded. High flexibility in the database would also be a feature sought after when going for NoSQL, so the column storage can also be eliminated from the equation. TODO: MAYBE SOMETHING MORE HERE lol the tree left are described well here : http://en.wikipedia.org/wiki/NoSQL [7, 12]

SQL

SQL database is the most common way of storing data. The data is stored in columns and tables. The database can make relations between the tables id necessary, hence relational databases. A main focus in SQL databases is queries and optimizing these where there usually exists some king of structured language to fetch the wanted data. The basic operations, which can be found in most SQL databases, are SELECT, UPDATE, REMOVE and INSERT. This kind of database is schema defined and follows the ACID(atomicity, consistency, isolation, durability) principles. [10]

Database Alternatives

TODO WHEN EXPLORED THE THREE NOSQL CATEGORIES LEFT, IF THEY ARE TO BE EXPLORED Below, some of the database implementation available to us are discussed. We mainly investigated document NoSQL databases and regular SQL databases. The most appealing options are introduced below.

MongoDB

MongoDB is a large scale, high availability, robust system. It is a document NoSQL system, so instead of storing the data in tables as you would in MySQL, the data is stored in a document based fashion through for instance JSON with dynamic schemas. This makes the DB easier scalable horizontally. But

the mongoDB still possesses the some of the great properties from relational databases, such as indexes, dynamic queries and updates. With mongoDB it is easy to map objects to programming language data types, which makes the DB easy to work with. The embedded documents and arrays makes the join operations less importance, which result in the ability to make reads and writes faster. JavaScripts can also be included in the queries. This makes mongoDB an efficient and high speed data base for storing objects and fetching to a system. MongoDB also has its own access console, where you can use scripting with Javascript language. [1]

CouchDB

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

MySQL

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

Conclusion

One particular type of NoSQL databases that caught our attention early was the document NoSQL databases. The main characteristic of these databases is their schema- less structure. They also differ from SQL by generally not using a structured query language for data manipulation. They are easy to replicate and they offer simple APIs for the clients to communicate with. They are heavily customized for web- applications, and have gained much popularity in the modern web era. Most document NoSQL databases focus on quick replies to requests, as the query operation is by far the most common in a typical web-application. Because they typically are distributed, NoSQL databases are able to store enormous amounts of data, and they are often used within Big Data [14] applications like Hadoop [13], which recently has become a very popular subject within the computer science community.

TODO: MAKE OUR OWN :p We decided to go for CouchDB in this project.

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

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

2.5.2 Programming languages

TODO something about the languages to use and why! Dependent on: weka twitter api solution used to solve netflix api database to choose future thoughts

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.

JavaScript

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

Conclusion

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

2.6 Software Testing

TODO: CHAPTER NOT THAT NECCESSARY PERHAPS?

2.6.1 Testing Methods

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

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

White- Box Testing

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

Black- Box Testing

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

Test Driven Development

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

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

[2]

Automated Tests

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

2.6.2 Testing Levels

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

Unit Testing

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

Component Testing

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

System Testing

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

Release Testing

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

User Testing

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

in is difficult to achieve during development. The end users can help provide feedback on how specific functionality will work in an actual work environment.

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

2.6.3 Conclusion

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

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

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

2.7 Result Testing

TODO: HERER OR IN THE REQ-CHAP?

2.8 Evaluation (maybe)

TODO: To sum up what we saw after the prestudy

Chapter 3

Requirements

Contents

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

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

3.1 Functional Requirements

- Harvest Twitter
- Find movies in user-tweets
- Find movie-tweets from tweeters the user is following
-
- Supplement netflix-dataset with Twitter-findings
- Test recommendation ability with improved netflix-dataset
- Generate reports
-

3.2 Non Functional Requirements

3.3 Requirements Evaluation

3.4 Prioritized Requirements

Chapter 4

Design

Contents

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

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

Men ikke ha med design dokumentasjon her!

4.1 Domain Concepts

4.2 High-level Architecture

4.3 Twitter Design

4.3.1 The Crawler Package

4.4 Netflix Design

4.4.1 The Data Package

4.4.2 The Recommender Package

Chapter 5

Implementation

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

Chapter 6

Evaluation

Contents

I evalueringskapittelet beskriv hvordan du vurderer arbeidet ditt. Oppsummer evalueringsresultatene, og bruk dem til å vurdere ditt eget arbeid kritisk. Vær ærlig om eventuelle mangler. Hva betyr resultatene? (Signifikans)

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

6.1 Development Process

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

Good

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

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

Bad

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

6.2 Testing

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

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

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

6.3 Issues

Group size

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

Ease of communication

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

The team members

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

Risk handling

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

6.4 Summary

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

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

Chapter 7

Conclusion

Contents

I konklusjonen din, beskriv status for ditt arbeid. Oppsummer hva du har oppnådd, sammenlignet med hva du opprinnelig ønsket å oppnå. Relater arbeidet til tidligere relevant arbeid. Foreslår videre arbeid som du tror vil være verdt.

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

7.1 Final Product

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

7.2 Chosen Technologies

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

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

Whereas trying to add undefined attributes to an existing record in a traditional relational database management system would result in an error, CouchDB automatically adds these attributes to the stored object. In essence, it allows us to add any new attribute to the existing objects, as long as it's represented in JSON format. This obviously gives you great flexibility. To do something similar in a regular SQL system you would have to change the schema of the database table every time a new attribute was discovered, or be able to morm

every aspect of the objects from the beginning. For non-trivial real world domains, this usually presents a difficult challenge. Also the job of representing complex objects are considerably less than in a relational database. Complex object representation in SQL often requires a tedious process to identify the different tables you need and the relations between them. In CouchDB this job is reduced to just putting the entire JavaScript object directly into the database, which saves a lot of time for the developer. Maybe the best experience from this project was the freedom the technologies gave us. All the technologies were really simple out of the box, and this left us with the ability of choosing exactly which functionality to add. Also the technologies were flexible, meaning there were few limitations to what we could add. This flexibility means that is is easy both for developers and end users to add new functionality to the system. It should be mentioned that to be able to use this system to its maximum potential you need some sort of prior experience with JavaScript. But with this in hand you can accomplish virtually anything.

7.3 Findings

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

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

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

server that tests the received objects for consistency to eliminate activity that is harmful to the system.

7.4 Further Development

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

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

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

Although the performance and scalability of the server was not a concern in this project, it should be mentioned that CouchDB is designed to scale. Meaning that it offers tools to easily replicate the databases to multiple servers, ensuring that users can be fed data from multiple servers instead of one central one. This solution avoids one single point of failure and divides the workload. Also it offers

great performance on queries to the database. CouchDB was made with this in mind, since for most web- applications the query operation is by far the most common one. While of course still a major challenge, we think that scaling an hypothetical finished version of our system, would be eased by the fact that we chose CouchDB. Also the performance of the database should persist even with a large amount of simultaneous users.

7.5 Summary

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

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

Appendix A

Requirements

Appendix B

Design

Appendix C

Implementation Documentation

Appendix D

Test Cases

Table D.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

Bibliography

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

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

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

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

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

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

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

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

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

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

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

[12] Cristof Strauch. Nosql databases.

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

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