UNIVERSITY OF SOUTHERN DENMARK

SOFTWARE ENGINEERING 6. SEMESTER

Datamining and its use

Author:
Lasse Bjørn HANSEN
Simon FLENSTED

Supervisor:
Jan Corfixen Sørensen SØRENSEN

A report submitted in fulfillment of the requirements
of Software Engineering 6. semester

at

University of Southern Denmark
TEK

May 10, 2017

"Some quote"

- Gruppe 3

Contents

Chapter 1

Introduction

1.1 Motivation

The amount of data being processed around the Internet and within big systems is continuously increasing. This data should be structured and modelled in a way that makes it easily accessible and easy to work with. Handling large amounts of data the right way can prove to be very useful, not only to the company who possess the data, but also to the end users of a product. To achieve this, the art of data mining is very useful. The company Struct A/S [4] has provided us with a software engineering task of creating product recommendations where data mining will create the foundation. This report will address the use of data mining, how to develop a solution that provides the user with intelligent product recommendations, and makes it possible to maintain current and future data. [1]

1.1.1 Data mining

Data mining has become a big part of modern software engineering. Lots of companies tends to store large amount of data. If the data is analyzed properly and put into use, it can create tremendous value to the company as well as its users. In this case Struct has stored information about users visiting their websites. Previously, this data was stored in an unstructured database and not put to use. By processing the data properly, using data mining, it can be structured in a way that makes it useful to the company e.g. product recommendations.

1.1.2 Product recommendation

If an e-commerce company wants to increase its profit, product recommendation has proven to be very beneficial.[5] This is heavily used by multiple companies including the retail giant Amazon.[6] If you can predict what sorts of products your costumer may find useful, additional sales becomes more frequent. Big data sets, like the one provided by Struct A/S, can make it possible to predict customer needs, if the data is processed properly.

Chapter 2

Problem statement

2.1 Problem description

The initial problem/challenge is given to us by the company Struct A/S and is described as follows:

When launching sites, whether it being regular websites or web shops, a lot of user activity is logged. We therefore have a large amount of data associated with each of our sites but do not currently use it.
In the future we would like to be able to use logged data to generate an insight into the user activity on our site and actively use this data to create a personalized experience for the users.

This project handles the initial analysis of the data, storing it in a scalable way and utilizing the data to create features which add value to the company. The focus of the project is data storing, data mining and recommendation algorithms. These methods are used to implement a final software solution capable of storing, organizing and utilizing current as well as new data about the end users. This allows Struct to easily keep their data updated and receive tailored product recommendations for their users.

2.2 Problem statement

The data we were given is in an unstructured format and can not be put to use as it is. This leads to the following problems - structuring and utilizing the data to create a personalized experience for the users, and making the data easily maintainable.
This leads to the following research questions:

- How can you optimally organize, store and access data in a scalable way?
- How can this data be maintained and updated easily after deployment?
- How can you utilize the organized data to generate tailored product recommendations for the end user?

Chapter 3

Requirements

3.1 Requirements engineering

The requirements of the project are categorized into functional and non-functional requirements. These requirements were derived from the original case given by Struct A/S (Appendix ??), continuously planned meetings with Struct A/S and our supervisor, and as a part of the constant research done during the progress of the project.
The functionality of the final product is developed in order to fulfill the most important aspects of the case, and the occurred requirements from the client meetings.

3.1.1 Functional requirements

The most important features of the system includes delivery of good quality product recommendation and handling of new data. This is very complex features and a lot of requirements must be fulfilled in, order to realize them. The most crucial functional requirements can be seen in table 3.1. These requirements has been the driving force throughout the project. For a complete list of functional requirements, see the git backlog [?].

As we can see from table 3.1, the functional requirements of the final product can be compressed into 12 requirements. This corresponds with the wish of a simple API, that provides good quality product recommendations. A lot of work has therefore been put into developing a complex and reliable algorithm that provides state of the art product recommendations. F01 was the most important requirement and has therefore acted as an ongoing task during the entire development of the product. F02-F06 was secondary requirements as they were not crucial before the recommendation algorithm was implemented. However, once the algorithm was applied to the system, requirement F02-F12 was needed in order to keep the data updated.

3.1.2 Non-functional requirements

The non-functional requirements was described at an early stage, and later clarified at the planned meetings. The functional requirements can be seen in table 3.2.

The non-functional requirements are few, but turns out to be very challenging. NF01 was a choice made based on the fact that the platform Struct is developing on is based on C#. .Net core was chosen because of its high performance and scalable systems, which was needed to realize NF02.[7] NF02 was probably the most challenging requirement to fulfill, however very important since you do not want your website to be slow. At the beginning Struct had a wish that the new database for recommendation data had to be scalable up to billions of records. Because of the denormalized data structure, it was agreed that a No-SQL database would be the right approach. This resulted in requirement NF03. In order to lower the amount of effort needed to integrate the recommendation system, the API had to easily accessible and the data output had to be in a standardized format. This resulted in requirement NF04.

F01	The webshop developer can provide tailored product recommendations to his customers. When the API is provided with information about a visitor, tailored recommendations to the customer must be returned. If the data about the visitor is insufficient to calculate enough tailored recommendations, the most popular products within the last 30 days must be used to present enough recommendations.
F02	The webshop developer can store new behavior data for a visitor in the database. When the API is provided with the required information, new behavior data must be stored in the database.
F03	The webshop developer can store new behavior data for a product in the database. When the API is provided with the required information, new behavior data must be stored in the database.
F04	The webshop developer can store new product groups in the database. When the API is provided with the required information, a new product group must be stored in the database.
F05	The webshop developer can store new visitors in the database. When the API is provided with the required information, a new visitor must be stored in the database.
F06	The webshop developer can store new products in the database. When the API is provided with the required information, a new product must be stored in the database.
F07	The webshop developer can update existing product groups in the database. When the API is provided with the required information, a product group should be updated.
F08	The webshop developer can update existing products in the database. When the API is provided with the required information, a product should be updated.
F09	The webshop developer can update a visitor in the database. When the API is provided with the required information, the visitor should be updated.
F10	The webshop developer can delete existing behavior in the database. When the API is provided with the required information, behavior data should be deleted in order to keep the data up-to-date.
F11	The webshop developer can delete an existing visitor in the database. When the API is provided with the required information, the visitor should be deleted in order to keep the data up-to-date.
F12	The webshop developer can delete an existing product group in the database. When the API is provided with the required information, the product group should be deleted in order to keep the data up-to-date.

TABLE 3.1: Functional requirements

NF01	The API should be developed in C# .NET core.
NF02	Recommendations must be delivered within 40ms.
NF03	The data used for product recommendations should be stored in a fitting scalable No-SQL database.
NF04	The API should be easy accessible through a web service.

TABLE 3.2: Non-functional requirements

Chapter 4

Design

4.1 Conceptual overview of the system

The system is developed as a part of the classic architectural pattern, Model-View-Controller (MVC).(find kilde, evt. wiki) The system itself consists of the Model and Controller part, and lets the client be responsible for the view. In figure 4.1 it is shown how the system is layered. As mentioned, the client is responsible of the view, which in this case is the webshop. Data is sent to the API (Controller) which simply communicates the data to the logic (model) of the system. The logic (model) is responsible for communication with the database, calculations regarding product recommendations, and handling of new incoming data.

4.1.1 Recommendation API (Controller)

The API is split into two Controller-classes, DataController and RecommendationController. If we take a look at figure 4.2, the two controller-classes can be seen. The RecommendationController only consists of one API-call, however this is probably the most interesting call that can be made. Calling the GetRecommendationForVisitor-method with a visitor UID and the desired number of product recommendations will return a String-array consisting of product IDs, which the client can then use to present products in a desirable way to the end-user.
The DataController is used for keeping the data in the database up-to-date. The three Put-methods allows the client to provide new visitor and behavior data to the database. The GetUpdate allow the client to initiate the build of the collaborative filter. Collaborative filtering will be discussed more thoroughly in **??**. GetUpdateVisitorTopProducts() allows the client to initiate the calculations of each visitors most viewed products. At last, CalculateTop20() lets the client initiate the calculations of the most popular products in the last 30 days. All the update-methods is special administrator methods, and should only be accessible to few access-approved staff.

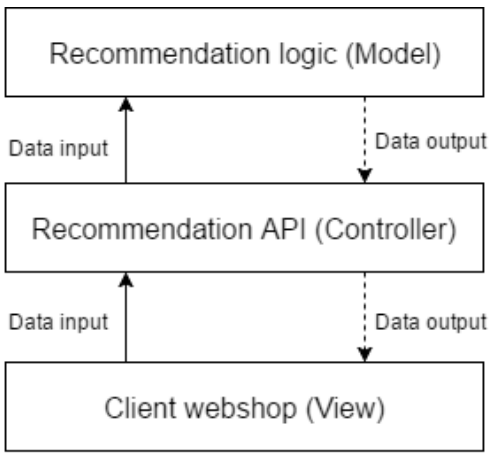

FIGURE 4.1: The Model-View-Controller (MVC) pattern applied on the system

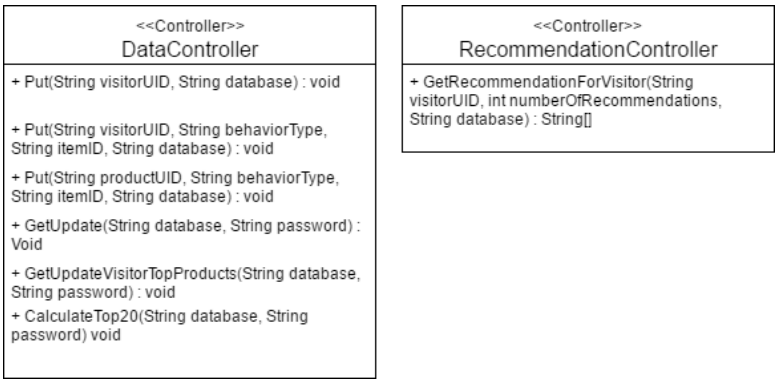

FIGURE 4.2: The two controller-classes in the recommendation API.

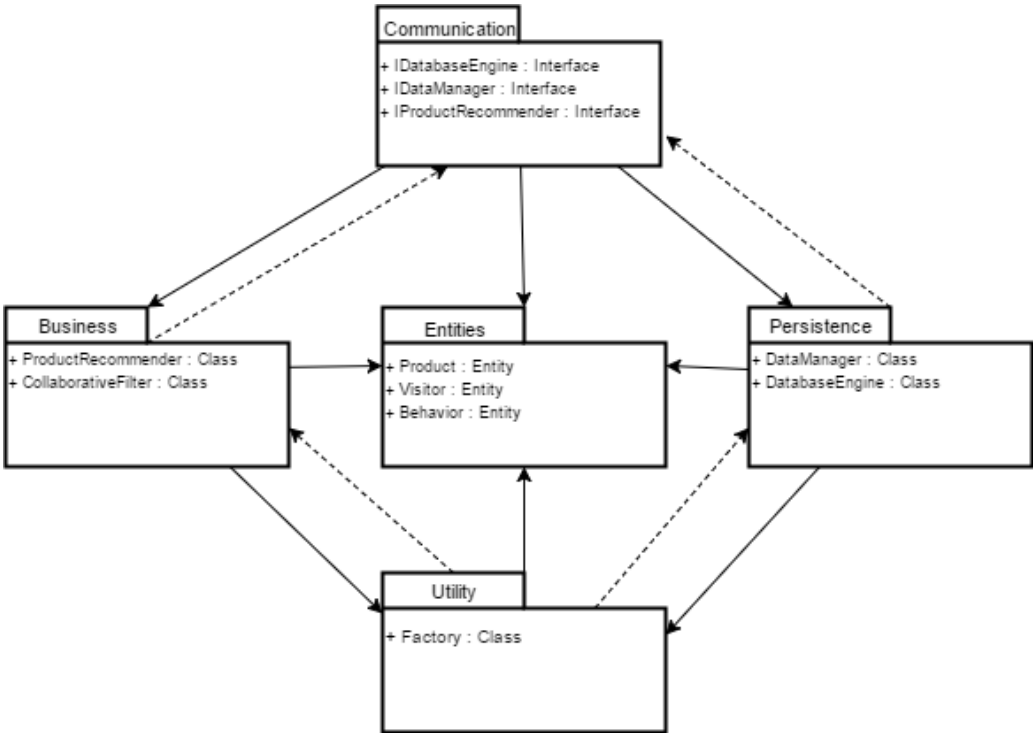

FIGURE 4.3: Package diagram of the model layer

4.1.2 Recommendation logic (Model)

The recommendation logic is where the main operations of the system takes place. The model layer consists of five packages, and is made accessible to the controller layer through three interfaces. A package diagram of the layer can be seen in figure 4.3. All classes used for calculations are placed n the Business package. This is where the product recommendations are calculated before being sent back to the control layer. This is also the package where any offline-calculation is made before it is stored in the database. The persistence package handles all information that needs to be communicated with the database. The Entities and Utility package creates an easier and more manageable way of communicating data around within the model layer. All communication between the Controller-layer and the Model-layer is done through the interfaces seen in the Communication package. These interfaces are implemented by their corresponding classes in the Business and Persistence packages. The implementation of the Model-layer is discussed further in the **??**.

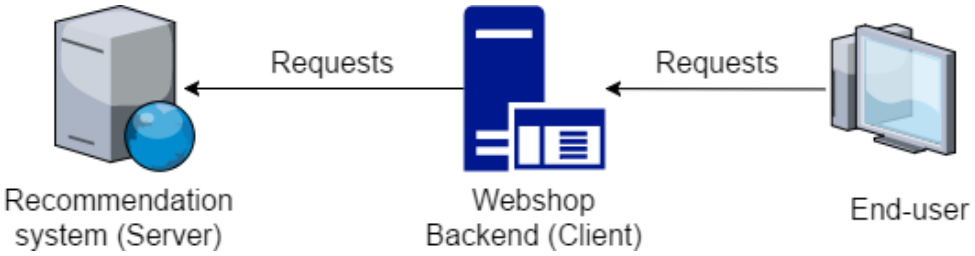

FIGURE 4.4: Package diagram of the model layer

4.2 Client-Server

When put to use, the recommendation system will be distributed and play the server role in a Client-Server model. The system should be considered an application solely for providing product recommendations. This means we have a very thin client, as its only job is to ask for recommendations by sending very little information. In this scenario, the client is the webshop that needs to provide recommendations to one of its users. The client is also able to ask the server to update its database or store new content in the database, but the concept is the same and just as simple as the request for product recommendations. The Client-Server model of the system can be seen in figure 4.4

4.3 Database design

One of the challenges of the project was to come up with a database design that would meet the requirements of the final product.

Bibliography

[1] A. Trivedi. (Feb. 2014). Mapping relational databases and sql to mongodb, [Online]. Available: https://code.tutsplus.com/articles/mapping-relational-databases-and-sql-to-mongodb--net-35650 (visited on 02/23/2017).

[2] M. D. C. Bowen. (Jan. 2017). What is normalized vs. denormalized data?, [Online]. Available: https://www.quora.com/What-is-normalized-vs-denormalized-data (visited on 02/23/2017).

[3] C. Buckler. (Sep. 2015). Sql vs nosql: The differences, [Online]. Available: https://www.sitepoint.com/sql-vs-nosql-differences/ (visited on 02/23/2017).

[4] S. A/S. (May 2017). Struct a/s, [Online]. Available: http://struct.dk// (visited on 05/03/2017).

[5] S. Arora. (May 2017). How to use personalized product recommendations to increase average order value, [Online]. Available: https://www.bigcommerce.com/blog/personalized-product-recommendations/ (visited on 05/03/2017).

[6] J. Mangalindan. (Jul. 2012). Amazon's recommendation secret, [Online]. Available: http://fortune.com/2012/07/30/amazons-recommendation-secret/ (visited on 05/03/2017).

[7] P. C.V.V.A.S.A. C. Rai and M. Wenzel. (Nov. 2016). Choosing between .net core and .net framework for server apps, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/articles/standard/choosing-core-framework-server (visited on 05/10/2017).