Háskólinn í Reykjavík

Hugbúnaðarfræði

T-303-HUGB

Sprint 5

Textual Document

Students:

Emil Örn Kristjánsson

Jóhann Ingi Skúlason

Jón Skeggi Helgason

Ríkharður Friðgeirsson

Steinar Snær Guðjónsson

Valdís Bæringsdóttir

Teacher:

Grischa Liebel

Date:

01.11.2020

Table of contents

1 Introduction

The aim of this project was to create a backend that supports the management and use of polling data for elections. The backend was supposed to be general enough so it can be used for all sorts of frontends for all sorts of polling data. This project followed an agile process, similar to SCRUM, where the project was divided up to five sprints. Each sprint was 2-weeks long and this document is the part of the last sprint. The first sprint was a planning sprint to perceive the system. The next three sprints that followed focused on implementing different parts of the system. This last sprint compiles the whole project and reflects on the work done.

2 The Organisation of the System

The system backend is implemented following a three tier architecture. The top layer or the presentation layer is the user interface. The middle layer of the design is the logic layer that coordinates the application, processes commands, makes logical decisions and performs calculations. The logic layer is also the connection between the top presentation layer and the bottom layer. The bottom layer is the data layer where all information is stored. This layer contains the database or the file system and passes that information to the logic layer that forwards that to the user at the top layer. This implementation names the top layer *Controller*, the middle layer *Service* and the bottom layer *Repository*.

Along side the three tier architecture, the system uses custom made models for important attributes of the system. These models are used in all three tiers of the system. The repository layer of the system reads information from *csv* files and generates data in the format of the models created. This allows the service layer to access the data in an easier and faster way. When the user requests some sort of data from the system at the controller, it asks the service to provide that information. The service then collects that requested information from the repository and sends back up to the controller.

A WebSocket component is implemented as a separate component that when called, interacts with the controller. The WebSocket component listens for requests and when called parses them into commands that are executed by the application.

3 Design Patterns

As mentioned before, the overarching architectural design pattern is the three layer design. Where responsibilities of the system are divided into three layers or tiers. This architecture allows for flexible and reusable applications, by separating the application into tiers we get the option of modifying or adding a specific layer, instead of reworking the entire application.

In each layer we use a bridge pattern in the class implementations, where there is a single class that acts as an interface for calls, the calls are then passed on to the more specialized classes, that handle their specific tasks. The bridge uses encapsulation, aggregation, and can use inheritance to separate responsibilities into different classes. For our application we use this mostly to organize the layers into more discrete parts, creating a more readable

and understandable structure. When used in a larger scale application, this allows for even more modularity in conjunction with the three tier architecture.

Our data models do not match with the required schema from the front-end component, our solution is to use the design pattern Data Transfer Objects. We transform the model data into a DTO that matches the front-end schema. The difference between data transfer objects and business objects or data access objects is that a DTO does not have any behavior except for storage, retrieval, serialization and deserialization of its own data.

4 Future Work

Future work would be first of all to finish the last two remaining *create* functions that the frontend supports. The code reviews stated that the code would be more readable if there were more comments so that would a next step in improving the system code. Another thing mentioned in the code reviews was that the system code was not well enough organized and was divided into many folders and files. Our opinion is that this way it is very organized and easy to change or implement more features in the system. The three tier architecture has proven very useful during the process of implementing this system and is very highly recommended. On the other hand the testing code could be improved and in fact many more aspects of the code might need further testing. Lastly if time and deadlines were not a factor it would have been fun to implement the frontend ourselves. That is often a nice part of programming as then you see more visual what has actually been implemented and what can be done with the created backend.

5 Final words

All together this was a very informative process and a very different and fun approach to assignments than what we have experienced before. All group cooperation went very well and we would like to thank Shalini for being helpful and amazing.