Software design document Trivia webapp

Author: Ante Cetinic Franulovic(4087933) – a.cetinicfranulovic@student.fontys.nl

Contents

[2 Versioning Table 1](#_Toc115792048)

[3 Goal 1](#_Toc115792049)

[4 Proposed solution 1](#_Toc115792050)

[5 Design decisions 2](#_Toc115792051)

[6 Separation of concerns and solid 2](#_Toc115792052)

[7 C4 Diagrams 3](#_Toc115792053)

# Versioning Table

|  |  |  |  |
| --- | --- | --- | --- |
| Author | Version | Changes | Date |
| Ante | 1.0 | Created the document and first 6 chapters | 23.9.2022 |
| Ante | 1.1 | Improved the timeline and added the C4 diagrams | 29.9.2022 |
| Ante | 1.2 | Removed the timeline and improved the separation of concerns. | 4.10.2022 |

# Goal

The goal of this project is to create a webapp that people can use to play quizzes, learn new information on a variety of topics, connect with their friends and have some friendly competition.

# Proposed solution

The main idea of the app is to have users who can play quizzes, add friends and send messages. The app would present you with a number of quizzes on different topics with the ability to search with a keyword. From there you will be able to choose an interesting quiz and be presented with a question and 4 answers (1 of which is correct). If you choose the right answer you get awarded a point and move on, if you choose wrong you just move on. The app would have a profile page where you can see your past quizzes (last 10 you played perhaps) and also showing your lifetime score that would be awarded based on played quizzes. It would have a feature to search for other users and see their profile, from where you would be able to add them as a friend and if accepted send them messages. In the beginning I would allow any user to create a new quiz, however there is a possibility to restrict this to certain users down the line. The app would also contain a admin user who would have more privileges which are not defined yet.

# Design decisions

For the back end of this project Java Spring boot framework will be used for a variety of reasons. Spring boot framework enables dependency injection, so that object can define their own dependencies. Furthermore, it also offers built-in support for some tasks for example: type conversion, exception handling, resource management, events, is autoconfigured with some dependencies which saves a lot of time, it can be used to create a standalone application that runs without an external web server and also offers opinionated approach which is very helpful for someone with no experience in java.

For the front end React will be used as it is more appropriate than angular for small scale applications. Some of the features that React offers are: Virtual DOMS which make the application faster and more responsive, JSX which enables us to easily incorporate JavaScript code into the webapp, Unidirectional data flow which makes debugging easier, React Hooks for easier managing of state logic between components, a big community to search for people who have had the same problems as you in the past. Furthermore, I will also use Material UI library which speeds up the process as it has some premade components (such as buttons, cards, tables) so I don’t have to create everything manually. This will be a significant time save during the project.

For the database I will be using XAMPP and a SQL database as I have experience with that, along with a ready installation. Furthermore, it is a relational database and I believe that joining tables could be useful for things like friends table.

# Separation of concerns and solid

Separation of concerns will be handled firstly by dividing the application into layers backend and frontend. Back end will further be divided into business layer, configuration, controller, domain and persistence, with some of them having additional separation within. I will ensure that every of the classes is limited to specific functionality. In my business layer there are interfaces for each use case so that the controllers can use the functionality without ever interacting with the persistence layer itself or even the implementation of the methods themselves. I will employ a clear hierarchy of interfaces and classes without unnecessary references and implementations.

# C4 Diagrams

Diagram

Description automatically generated

Starting with the C1 diagram, we have 2 types of users. One is a regular user and one is an admin, the difference being that admin can view some information about the users. They both interact with the same software system, however depending on the logged in user, the functionality changes.

Diagram

Description automatically generatedMoving onto the C2, we still have the same users, however now we take a deeper look into the software system. We can separate it into 3 parts. The frontend which is the one that the users interact with, the backend (API application) which is being called from the frontend and actually manages the functionality and finally the database which is where the backend is storing the required quiz and user data.

Diagram, text

Description automatically generated

Furthermore, in the C3 diagram we will take a closer look at the backend (API application) part of the software system. We can divide it into 3 layers. Controller which is the part that is responsible for the REST endpoints. Business layer which is being called from the controllers and contains the use cases and their implementations. Repository layer which is responsible for performing CRUD operations on the database. This ensures the separation of concerns as there is a clear distinction between layers and a what they are accountable for.

Diagram

Description automatically generated

Finally, in the C4 diagram we take a closer look at what these layers look like. Starting from a controller which calls use cases from the business layer. However the controller is not aware of the use case implementation, it is simply calling the interface of the use cases. The use cases then refer to the persistence layer, specifically the repository interface, so they are not aware of the repository implementation either. This will improve on the dependency inversion and Liskov substitution principles as the classes are not aware of the implementations of the interfaces they are using. Interface segregation will be achieved by having a different interface for each use case in order to make sure the only methods that classes depend on are the ones they really need. In order to achieve the single responsibility principle I will have 2 controllers, 1 for quiz related things and 1 for user related things.