Research

Individual Project Semester 6
Fontys, Eindhoven
Veronika Valeva 4090349
31/3/23

What is the best way to keep the different aspects of the system separated ?

Contents

Introduction

When creating software applications, it's important to separate different parts of the application from each other. This means that each part of the application, such as the user interface or the database, should be developed and maintained separately. This approach makes it easier to manage the application and make changes to it. Keeping different parts of the application separated also makes the application more flexible, meaning that it can adapt to changes more easily. It also helps to improve the overall stability of the system, making it less likely to crash or experience issues. Additionally, by separating the different parts of the application, it becomes easier to scale the application as it grows in size or complexity. In this research document, we will explore the best way to keep the different aspects of a quiz game software application separated, using a modular design approach.

Modular design approach

A modular design approach involves breaking down the application into smaller, self-contained modules that each handle a specific aspect of the system. Each module should have well-defined inputs and outputs and should not rely on other modules to function. This makes it easier to modify and update individual modules without affecting the rest of the system. It also makes testing and debugging the application easier since each module can be tested independently. With a modular design approach, it is easier to maintain the system since each module can be updated or modified independently without affecting other modules. Adding new features to the system is easier. New modules can be added to the system without affecting existing modules. . Since each module is self-contained and does not rely on other modules, errors in one module are less likely to affect the rest of the system.

Quiz Game module separation

Front-end

The front-end of the applications will include the user interfaces that users will use to interact with the system. These interfaces have been divided into two distinct components: the game – including game creation and the game itself, and a separate UI for the scoreboard. The reasoning behind the separation of the two components is that they provide different information to the user.

The Game UI will provide features related to gameplay, while the Scoreboard UI will focus on displaying the player's scores and rankings. Keeping the scoreboard in a separate service allows us to scale and maintain it independently from the rest of the game UI. It also helps to ensure that the scoreboard is always available, even if there are issues with other parts of the game UI. This separation of concerns makes it easier to develop and maintain each part of the game, ultimately leading to a better user experience.

Game UI

The Game UI will be an interface used by the users when they authenticate themselves and enter the game. The players are going to be able to set their game preferences and start the game. The technology that is going to be used for this component is yet to be decided.

Scoreboard UI

The Game UI will be an interface used by the users to see their score in all or in a specific question category. Every user will be able to compare his/her score to the ones of the other players. This will be happening in real time and due to this the UI will be rapidly changing every time a player's score changes (gains point). The technology that is going to be used for this component is yet to be decided.

Back-end

The game system is going to be divided into separate services, including a Game Logic Service, Questions & Answer API, Scoreboard Service, and Authentication & Authorization Service. This will make the system easier to manage, more secure, and better suited to handle a large number of users. By dividing the system into separate services, each part can be updated or changed without affecting the others. This will make it easier to fix problems and add new features. Thus, making possible for users to play the quiz game even if the scoreboard service is down. The same goes for the situation where the game logic service goes down (or is under construction) – still the users can use the scoreboard. Overall, splitting the game system into separate parts is a good idea because it will make the system work better and be easier to use.

Game Logic

This back-end server will be responsible for managing the game logic. It will be responsible for the generation of quizzes, the actual game, the answering of questions, and keeping the users' preferences – part of the quiz generation. Initially, the players will play by themselves, however, the server would allow the game logic to expand to multiple people playing together. This would require a real time connection between the back-end and the front-end. For this Client-Server communication the usage of SignalR might be considered. ASP.NET will be used as this back-end framework.

Scoreboard

This back-end server will be responsible for managing the scoreboard logic. It will be responsible for the generation scoreboard information based on different options. The user will be able to see the constantly growing scores of the active players. This will require a real-time connection between the back-end and the front-end. SignalR provides real-time communication between the server and clients, allowing for instant updates to the scoreboard as the game progresses. SignalR can handle large numbers of simultaneous connections, making it suitable for (multiplayer) games with many players. SignalR is tightly integrated with ASP.NET thus ASP.NET is going to be used to build the Scoreboard back-end.

Questions & Answer API

The Questions & Answers Generator will be a back-end server that gathers questions and answers of a specific set of options. Instead of using a custom-made database filled with questions and answers, "The Trivia API" is going to be used. The trivia API can be updated frequently,

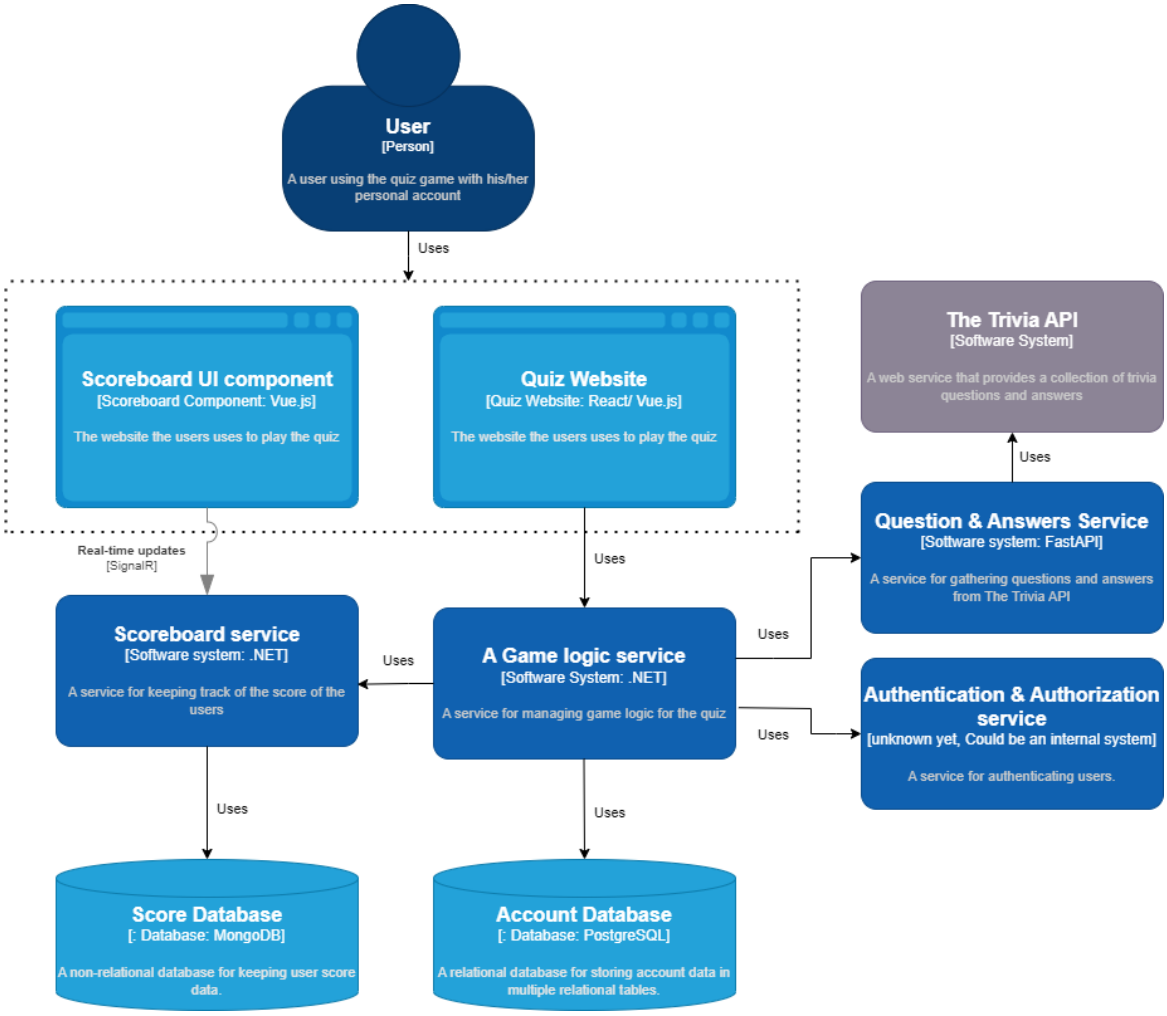
providing the game with the latest and most relevant questions. In contrast, maintaining a database of questions requires constant updating and management. “The Trivia API” provides easy integration through a REST-full API. The API and all data returned from “The Trivia API” are licensed under a Creative Commons Attribution-NonCommercial 4.0 International License. This means that the API and the questions it returns are free for non-commercial use – which fits the scope of this project. FastAPI will be used as a middleware service for separating concerns between the Game Logic and “The Trivia API”. FastAPI is a framework optimized for high performance and speed, making it a great choice for building a service that needs to communicate with a third-party API like "The Trivia API".

Authentication & Authorization

Users are going to have to authenticate themselves before being authorized to play the quiz. By having users login to a game, the game can store user data and preferences, such as their progress in the game, achievements, and settings, and also ensures the security of the game and the user's personal data. It is to be decided how the authentication & authorization is going to be implemented.

Data storing

I have decided to split the game data storing into two separate databases hosted on an Azure server: an Account Database and a Scoreboard NoSQL Database. This decision was made to improve the overall performance and scalability of the game. The Account Database will store user account information, such as usernames and e-mail, game preferences, while the Scoreboard NoSQL Database will store game scores and rankings. By using an Azure SQL server to host the databases, we can ensure that the game experience remains smooth and efficient, even as the user base grows. Keeping these two types of data separate, ensures that there are no performance issues that may impact the game experience. Additionally, using a NoSQL database for the scoreboard data allows for greater flexibility in handling large amounts of data, making it easier to scale and maintain the database as the quiz game grows in popularity.


User
[Person]
A user using the quiz game with his/her personal account
Uses
Scoreboard UI component
[Scoreboard Component: Vue.js]
The website the users uses to play the quiz
Quiz Website
[Quiz Website: React/ Vue.js]
The website the users uses to play the quiz
The Trivia API
[Software System]
A web service that provides a collection of trivia questions and answers
Uses
Question & Answers Service
[Software system: FastAPI]
A service for gathering questions and answers from The Trivia API
Real-time updates
[SignalR]
Uses
Scoreboard service
[Software system: .NET]
A service for keeping track of the score of the users
Uses
A Game logic service
[Software System: .NET]
A service for managing game logic for the quiz
Uses
Authentication & Authorization service
[unknown yet, Could be an internal system]
A service for authenticating users.
Uses
Uses
Uses
Score Database
[: Database: MongoDB]
A non-relational database for keeping user score data.
Account Database
[: Database: PostgreSQL]
A relational database for storing account data in multiple relational tables.