Research

Individual Project Semester 6
Veronika Valeva 4090349
Fontys, Eindhoven
22/3/2023

How to build a Game Logic service?

Table of Contents

Introduction

In the case of my individual project (The Quiz Game) There should be a service that provides the necessary functionalities to control and manage the game's rules and logic. The Game Logic Service's purpose is to centralize the location for the game logic, which makes it easier to manage and maintain. This helps to avoid duplication of logic in different parts of the game, reducing the risk of errors and inconsistencies. It also separates the game's business logic from the user interface and other parts of the game. The service is responsible for managing the handling of game events, determining the correct answers to questions, and calculating scores. It also tracks the progress of each user in the game and manages their game state, such as the questions answered correctly, the score obtained and the game preferences.

Requirements

To build the Game Logic Service I have to consider several factors to ensure that it is scalable, secure, and reliable. The Game Logic Service needs to be designed to handle a large number of players and game instances simultaneously. This requires careful consideration of the service architecture, load balancing, and database performance. The Game Logic Service should be made with safety in mind. This means it should have ways to make sure only authorized users can access it, make sure data is protected while being sent and stored, and protecting against common attacks, such as SQL injection and cross-site scripting. The Game Logic should provide a RESTful API that allows the Game Dashboard client to interact with the service, such as generating and starting a new game, answering a question, and viewing scores. The service needs to be able to call the Q&A service to gather data about question categories and Q&A data sets. It also has to validate given answers, calculate scores, and determine game outcomes. Another factor to consider is including a database to store game data, such as player information, game instances, and game analytics.

Serverless architecture

Serverless architecture is a way to build and run applications without managing servers. Azure Functions can be used to implement the game logic as a series of functions that run in the cloud. Using Azure Functions eliminates the process of choosing a framework for development, learning how to deploy that framework on a server and managing and maintaining those servers overtime. The game logic is broken down into smaller functions that perform specific tasks like gathering questions, validating answers, calculating scores, and determining game outcomes. These functions are triggered by specific events such as user actions or scheduled tasks. Azure Functions provides built-in support for secure authentication and authorization using Azure Active Directory, allowing you to control access to your game logic functions. Azure Functions are highly scalable and can handle a large number of requests, making it easier to scale your game as the number of players increases.

Cloud-based game engine

Cloud-based game engines like Unity or Unreal Engine provide a complete solution for building and deploying games. These game engines include built-in support for game logic, multiplayer features, and other game-related functionality.

To build a game using a cloud-based game engine, one can use the scripting capabilities of the game engine to implement the game logic. For example, a C# or JavaScript script can be written that generates questions, validates answers, calculates scores, and determines game outcomes. The game engine provides a development environment and tools to create and test the scripts. The user interface can be developed by using built-in tools of the game engine, such as menus, buttons, and text fields. These tools allow the designing and creation of the visual elements of the game. Additional scripts can be used to add interactivity to these elements. One advantage of using a cloud-based game engine is that it provides a powerful and comprehensive set of tools for building and deploying games. The game engine handles many of the technical details, such as graphics rendering, audio playback, and network communication, allowing to focus on the game design and logic. Additionally, cloud-based game engines often provide features for multiplayer games, such as matchmaking, game sessions, and real-time communication. However, using a cloud-based game engine also has some drawbacks. The development process can be complex and requires a significant amount of technical expertise. Additionally, cloud-based game engines can be expensive, especially for large-scale games with many users.

Separate service architecture

In a separate service architecture, the game is broken down into smaller, independent services that can communicate with each other. To implement a such an architecture for a game, one can create separate services for game logic, user interface, and data storage. For the game logic service, separate service can be created, which will be responsible for gathering questions, validating answers, calculating scores, and determining game outcomes. This service can expose a RESTful API that other services can call to interact with the game logic. The user interface service, can also be a separate service that is responsible for providing different pages for starting a new game , answering questions, and viewing scores. This service can also expose a RESTful API that other services can call to interact with the user interface. The data storage service will be responsible for handling player profiles, game instances, and scores.

Solution

I have chosen to use the option of separating service architecture. This module of the project will have a ASP.NET Web API service for managing the game logic, a React user interface for accessing the game, generating and playing quizzes, viewing own and overall scores, and an Azure SQL database for storing user information, game information and analytics and scores. Using a .NET Web API service and a React user interface simplifies development and reduces development time. Azure SQL is highly scalable, which means it can handle large amounts of data and traffic without a problem and provides advanced security features such as encryption at rest and in transit, as well as built-in threat detection capabilities. Since I already decided to use Azure SQL, for the upcoming sprints I can easily integrate with other Azure services. ASP.NET API with EF Core provides a strongly typed data access layer that enables you to interact with your database using plain C# - eliminating the need for handwritten SQL queries. EF Core can also be easily integrated with other technologies, such as React and Azure SQL. It is a cross-platform framework that allows you to run your application on Windows, Linux, or macOS, making it a

flexible and versatile choice for your project. The authentication and authorization is yet to be researched.

Deployment

To be added

Continuous Integration and Continuous Delivery

To be added