66

CIS045-3 Distributed Service Architectures – AY 24/25

Student Name: Janith Amantha Lokumanna **Student Number: 2433417**

Student Signature: [illegible] **Date: 16th March 2025**

Please answer the questions below, print this form and bring it to the presentation where it will be checked against reality and signed off by the tutor. Once signed off, scan this form and submit on BREO with all other deliverables (video, code). At this stage, Week 8, (at least) a working prototype should be presented plus some ideas on how to continue with the development.

Software Architecture (Low Coupling / High Cohesion)

How is your code structured? (e.g., components, files, libraries, classes, packages ...)

The game adapts three-tier architecture to separate responsibilities of client, server and database. The game is created using Unity Engine by mainly following OOP concepts in C# on the client side. The client-side code structure follows a modified MVC (Model, View, Controller) approach where Controllers connect Views with Models. View contains C# scripts that are attached to the UI elements of the game, Model contains base classes that are needed throughout the game along with data classes while Controller contains classes that connect Views with Models. The tasks of each C# scripts are made independent by separating them into multiple C# files based on responsibility. By separating tightly coupled C# scripts based on responsibility, it has become possible to reuse the scripts on multiple scenarios without depending on other elements. Also, repetitive code lines are eliminated by creating reusable functions increasing readability and understandability of the code. C# Interfaces are also used to reduce dependencies for functions that needs to be implemented independently. The server side of the game is developed using FastAPI (Python) which is structured by separating functions and classes into separate Python files based on their responsibility to assure the reusage of them and to decrease dependency on individual files assuring high cohesion and low coupling while also making the code clear and understandable.

Event Driven Architectures

What in your code triggers events (e.g., GUI, buttons, timeout ...)?

Since this game utilizes a Graphical User Interface, the main events that manage the flow of the game are Unity Engine's button click (onclick) events. The button onclick events are automatically managed including the handlers by the game engine's button component which lets developers assign functions to onclick events. Furthermore, the core of the game is managed by Keyboard button press events. Utilizing button press events, the game core decides whether the user have pressed the correct key to obtain scores. Most of these events are handled by Unity Engine itself with its event handlers. Additionally, the game includes observer model that follows Event Driven Architecture. The game manages a timer of 10 seconds when letting users answer the problems returned by the banana API. A custom event is created and subscribers to it are assigned. When the timer reaches 0 the event is executed, and the subscriber is invoked as a response to the event making the game over.

Interoperability

Do you use the API https://marcconrad.com/uob/banana/api.php?

If no, where in your code do you demonstrate interoperability? What protocols do you use?

Yes, the game uses banana API to provide extra attempts to players when they run out of attempts. Also, the game follows 3 tier architecture with client, server and database. The client (C#) only directly communicates with server using its socket (IP + port) and endpoints provided by it. The server communicates with the database using its credentials and URL. Client communicates with banana API and FastAPI server while the server communicates with the client and the database. The database only communicates with the sever directly. Interoperability can be seen by all above mentioned scenarios. The C# files of Models are responsible for communication with the server and banana API. They are used to send asynchronous requests to the API endpoints with necessary headers and to capture the returning information. FastAPI endpoint functions are used to capture the requests sent by the client, forward them to the database and to respond the client back with the responses provided by the database.

Virtual Identity

How did you implement virtual identity in your code? (e.g., Passwords, Cookies, IP Numbers ...)

Virtual identity is implemented in the game by creating separate profiles for each player in the game. Each player is authenticated by their email and password which is encrypted. Also, the identity of each player is separately checked using tokens that expires within a time which also enforces more security and confirms users' virtual identity. In every interface of the game, the availability of the token is checked multiple times in every second. The moment the token expires, the game will only let the users access the login interface. A separate leaderboard that is sorted based on the scores each user has obtained is available in the game as well. The leaderboard represents a list of all available players while highlighting the current account a player has logged in from. The current player details are identified using a unique user id that is used to represent each user uniquely, assuring a virtual unique identity to each user.

Any other interesting features:

Sound effects are added to the game to make the user experience of players more entertaining. Problems from the banana API is used to increase the number of attempts a user can have by providing the correct answer to the returned problem. As the game progresses, the falling letters starts falling faster due to an incremental increase in gravity of letter objects making the game more challenging. User passwords are two times encrypted. Once by the client and once by the database enforcing more security. The game core follows singleton design pattern to assure there is only one object is active from the game core, making the program steady. Instead of creating and destroying scenes, panels in one scene are used to represent some UI elements (Login) to reduce the overhead.