Assignment 2

GUI and Communication

In this assignment you will implement a **Graphical User Interface** as a frontend (client) for your Texas Hold'em system. In order to facilitate GUI-server interactions, you will also define a **communication protocol** and implement a supporting **communication layer** in both the server and the GUI client.

Communication

The system needs to handle connections from multiple clients at any given time. Clients should be able to join and play Texas Hold'em games together, as well as send messages to each other (described in the functional requirement section).

Communication Layer

An important aspect of designing a good system is to isolate the main logic of the system (the domain layer). Changing communication protocols, or communication technology (i.e. supporting access to the system from an additional mobile/web platform) should not affect the system's logic in any way. Therefore, classes and components which handle communication between the server and every possible client should be part of the "communication layer" and interact with the domain layer only via the service layer.

Server-initiated Updates

In several usage scenarios of the system, the server needs to pass information to a client not as a direct result of an action performed by this client, e.g. update the client about actions performed by other players in a game of which he is active, send notifications to the client, etc.

This type of communication can be achieved in two ways. The first way is known as "polling", the client queries the server every few seconds to check if there are any new updates. The second possibility is initiating a connection from the server to the client. Each of these options has a different required resource. Be prepared to explain your choices for handling server-initiated updates and the type/amount of information you send.

Note that a client doesn't need to receive all of the updates from the server, only those which are relevant to it. For example, a client should not receive updates from games in which it is not participating.

Graphical User Interface

User interfaces are complex components, characterized by states and external events. As a result, a natural formalism for modeling GUIs is **Hierarchical Statecharts**.

Start by modeling your client using a hierarchical statechart. States and state nesting should be induced by the use cases your system supports.

Implement a client GUI application for your system, based on your model. You may implement either a desktop GUI application (macOS / Windows / Linux) or a mobile GUI application (iOS / Android / Windows Phone). You MAY NOT implement the GUI as a web client[1].

The user interface should support all functional requirements described in the current and the previous assignments. This includes (but not limited to): registration, user sign-in, watching and editing user profiles, watching replays, creating, joining, leaving and playing a Texas Hold'em game, etc. The user interface should allow playing multiple games simultaneously (i.e. displaying multiple interactive windows).

While you are not required to make the application a beautiful work of art, you must observe the usability non-functional requirement defined in Assignment 0. Additionally, the application should be reasonably responsive and fast to react.

*We **strongly** suggest that you do not go overboard when working on the client. Maintain proper workflow and management: define small tasks, estimate time costs, define dependencies between the tasks, and **prioritize properly**.*

Additional Functional Requirements

- **Remove** the use case "Save favourite turns from replays in order to be viewed on later occasions" (originally added in Assignment 0). This includes removing:
    - All implementing code
    - Any supporting tests (unit/integration/acceptance)
    - Relevant models (use cases, diagrams, glossary references, etc.)
- **Remove** the use case "Leagues - the highest ranking user in the system may: Set a default league for new users, set criteria for moving to a new league, move users between leagues". This includes removing:
    - All implementing code
    - Any supporting tests (unit/integration/acceptance)
    - Relevant models (use cases, diagrams, glossary references, etc.)
- New users are given an "unknown" rank, which is calibrated over their first 10 games.

[1] Please note that repackaging a web application as a .apk/.ipa/.appx file does not count as a mobile GUI. It's still a web client...

    - A user with an “unknown” rank may play in any league
- The system redistributes the players among the leagues once a week with the following constraints:
    - Each league contains at least two users
    - All leagues should have the same number of players ±1
    - If there are not enough players to fill all leagues with the constraints, prefer filling the **higher** leagues first (lower leagues are allowed to be empty if the system has very few users).
- Add a single message stream (i.e. chat) to each game instance. All users (both players and spectators) in the game may publish messages to this stream. The following rules must be enforced:
    - Everyone can see player messages
    - Players cannot see spectator messages
    - Players can send **whispers** (private messages) to anyone in the chat
    - Spectators can send whispers only to other spectators (not to players)
    - Whispers are private and can only be seen by the sender and the recipient

Non-Functional Testing

Unlike functional tests (unit, integration, acceptance), the function (:D) of non-functional tests is to help the development team validate the non-functional requirements of the system, i.e. the way the system operates, instead of the system’s behavior.

In addition to the functional tests you must write and maintain, you are required to implement and maintain a collection of system **performance tests**.

Performance testing is a testing practice used to determine how a system performs in terms of responsiveness and stability under various workloads. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage. In the context of this project, performance testing should help you meet various non-functional requirements, such as Speed, Load and Capacity.

Load Testing and Spike Testing are examples of different types of performance testing.

Define and implement Load test scenarios for your system[2]. Your tests should track various aspects of system performance, such as resource use (CPU, RAM and network utilizations) and response time[3]. Additionally, you should make sure to catch and document system crashes which occur due to resource overuse.

[2] There are some free tools available that may assist you with this.
[3] When measuring response time, make sure to differentiate between the network latency and the time the system takes to process the request and respond.

Use the results to estimate the handling capacity and performance of your system on your current hardware under standard operating conditions.

Based on the results of the load tests, estimate a “maximum expected capacity”. Define and implement spike test scenarios to match this estimate.

Required Models and Specifications

- Model your communication components as part of the component diagram
- Model the GUI using hierarchical statechart.
- Model your client as part of the component diagram.
    - There is no need to go into too much detail of the inner workings of the client component
    - No need to create a class diagram for the client
- All diagrams and documents - updated with any changes from this version
- Obviously, your implementation must match your model and vice versa - so keep them synced!

Implementation Guidelines

- The implementation guidelines remain the same in the previous assignment.
- You are required to rotate the roles of Project Manager and of Acceptance Tester.
    - The new project manager must be a team member who has not previously filled the role.
    - Similarly, the new acceptance tester must not have held the role in past iterations.
- In addition of taking over the responsibilities of the previous acceptance tester, the new tester will be in charge of non-functional testing as well.
- Make sure to test the communication between the server and the clients over a network. Use several clients from **multiple devices** to communicate with the server simultaneously. Be prepared to present your system in class via multiple devices.

Deliverables

The project manager will present the following:

- Version summary, including:
    - The status of each requirement: is it implemented or missing? Was it changed in relation to the previous assignment? What was the change?
    - Who did what
- Up-to-date diagrams and specifications

- Up-to-date codebase structure
- Up-to-date functional tests, coverage and statistics
- Non-functional tests and statistics
    - Report on the results and your conclusions
    - Did the tests help you find problems and weaknesses in your implementation?

Each team member must understand all models thoroughly (i.e. diagrams, not use-cases). In addition the team members will present their part in the assignment.

Class Presentation

A team member will prepare a 10-15 minute discussion on the topic of API Design. Read (at least) this article from Microsoft on API design best practices (and any other literature on API design, considerations and practices). Prepare a discussion which covers (at least) the following points:

- Why do we need APIs, what are they for?
- Separation of concerns - Protocol is separate from API is separate from System Internals
- What are REST and RPC? Do you consider one to be better than other in the context of your project? Why?
- Present a set of guidelines / rules to follow when designing an API.
- Find a tool or a set of tools which will assist you with API design and testing. Present and use it in your project.

Please refer to the presentation guidelines at the Assignment page.