Assignment 3

DB and Web Client

In this assignment you will enrich the system with a **persistence layer (database)**, and a **Web based GUI** as a frontend (client) for your Texas Hold'em system.

Functional Requirements

Statistics and general details:

1. Leaderboards. The client can ask for different users leaderboards (top 20 users) sorted according to:
    a. The total gross profit.
    b. The highest cash gain in a game.
    c. The number of games played.

2. User statistics:
    a. The client can ask for a specific user statistics:
        i. Player $ win rate (avg. cash gain per game).
        ii. Player gross profit win rate (avg. gross profit).

3. **No need for the desktop GUI client to support the leaderboard/statistics use cases.**

4. Game chat (relevant to server side only). An additional rule should be enforced regarding the message stream of a game: Messages are sent in a push manner to connected players only.

Non-functional Requirements.

1. **Reliability:**
    a. Information (users, passwords, etc.) **is saved over time**, unless deleted by its data owner (if there is such owner).
    b. Multiple dependent data changes should be done in a transactional manner.
2. **Security:**
    a. Passwords should be saved encrypted in the database.
    b. Passwords and private data should be end-to-end protected (i.e., the 2-way path from the client to the DB will be encrypted and avoid vulnerabilities such as Man-in-the-Middle).
    c. State-of-the-art encryption method should be used for better security (i.e., use the your language built-in / known cryptographic libraries).
    d. Preventing "identity theft", i.e. tricking the server to perform "fold/bet" action for other users.
3. **Web client:**
    a. Some services will be accessible from a web browser as well.
    b. The included services will be: login/logout, leaderboard and player statistics.
    c. No need to support ANY OTHER use cases.
    d. **It's fine if your web client is ugly**, the point is not to learn CSS/javascript at a professional level.

Reliability

The reliability requirement implies the necessity of a persistence component (and a dedicated layer).
Above the obvious notion that the database saves data, it also overloads the application layer with information about managing the database.
There are methods and patterns for separating and connecting the application logic and the persistence layer. We advise you to read Fowler's book - "Patterns of Enterprise Application Architecture" chapter 3.

Class Diagram changes

Saving information over time is called **persistency**. In order to support persistency in a planned manner, decisions must be made at the class diagrams level:

1. Classes whose instances are stored in a database, should be marked as persistent classes, preferably using a label.
2. The **direction and visibility** in the class diagram should be rechecked (the direction describes connectivity, and the visibility describes methods dependencies).

3. For each class C that is persistent, all of its **client classes** (i.e., all classes that are dependent on class C, either explicitly or by a method that receives a type C parameter) must be examined. For each client class of C, **the client methods of C should be marked as clients of persistent classes**. If C1 is a class that is dependent on class C by a connection, then C1 constructors are C client methods.

Mapping Objects to the Database

For persistent classes:

1. You should maintain a separation between the domain layer and the persistence layer.
2. You must choose (and use) a technology that maintains such connection/separation (though there is always a tradeoff you should consider). For more information read the class presentation definition
3. Provide translation from application representation to database representation.
4. When a persistent class object is created, it must be stored in the database as well.
5. Make sure that changes to objects are saved in the database.
6. Make sure that client methods can ask for objects in the database (if the persistent class is not yet loaded into local memory).

Local Memory and DB objects

The system **database integration** raises many problems, which still do not have comprehensive solutions, and therefore we need to find concrete solutions:

1. Handling data duplication between the DB and the local memory.
2. Handling dependencies between database objects, and in-memory objects.
3. Traffic efficiency between local memory and storage (think of objects that are used often and objects that are rarely being used).
4. How the system should behave in case of a database failure? Is there a difference between reading and writing?
5. How to handle local objects in case of a transaction failure?

Tests

1. Provide **tests** for database retention (developer side). The tests should examine the kinds of problems raised in the previous section. The tests must be performed in a regression tests manner (i.e., rerun all previous tests).
2. One of the problems with database testing is **side-effects**. Tests should be designed so that they:
    a. Will not damage an existing database.
    b. Will be reliable.
    c. Will enable tests integration (i.e., it will not be required to “reset” the initial data before each test).

Security

The class presentation for this assignment covers security issues relevant to client-server communication such as wo/man-in-the-middle attacks and identity forgery.

In order to protect your users from such attacks, use an encrypted communication protocol, such as https (against W/MitM attacks) and unique random “session ID”s (against identity forgery):

- Each time a (successful) login request is handled, generate a unique random number. Associate this number with the logged in user in your server and return the number to the client.
- Each request that is not a login request is required to send both a user id and a unique identifier. If the identifier does not match the user id, you have detected an attempted identity theft (or a bug). Return an error.

Disclaimer: *Note that this implementation is far from secure. There are much better (more complex) solutions, but they fall outside the scope of this project. This is a simple, imperfect solution that should be considered as an exercise. Do* ***not*** *use this method as a security measure in any real systems.*

Required Models and Specifications

- Model your persistent components as part of the component diagram
- Model your web 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 assignments.
- 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.

- 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:
    - Unit tests for DB integration (in addition to the other tests).
    - 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

Encrypting the client-server communication and sensitive data in the database is crucial in order to protect the client.

A team member will prepare a 10-15 minute discussion on the topic of wo/man-in-the-middle attacks. Please refer to the following points:

- What is a wo/man-in-the-middle-attack?
- How does it work (general methodology)?
- How does HTTPS help mitigate W/MitM attacks?
- HTTPS doesn't fully protect against W/MitM. An example of this is CSRF attacks. What are some ways to protect users against CSRF attacks?

Please refer to the presentation guidelines at the Assignment page.