Trivial Trivia

Design Documentation  
Prepared by <TEAM>:

* Laxmi Poudel <lp2439[@rit.se](mailto:jdoe@rit.se)>
* Uttam Bhattarai <tb2040[@rit.se](mailto:jdoa@rit.se)>

Summary [2](#_heading=h.30j0zll)

Domain Model 3

System Architecture [4](#_heading=h.3znysh7)

Subsystems 6

DataBase Subsystem 6

User Authentication Subsystem4 9

Command Subsystem 11

Status of the Implementation 17

Appendix 18

# Summary

The Trivial Trivia project began by identifying the functional and non-functional requirements for the game. Functional requirements specify what the game should do, such as allowing the users to create accounts, play the game, view their stats, and add questions to the game. Non-functional requirements specify how the game should work, such as the performance and usability requirements.

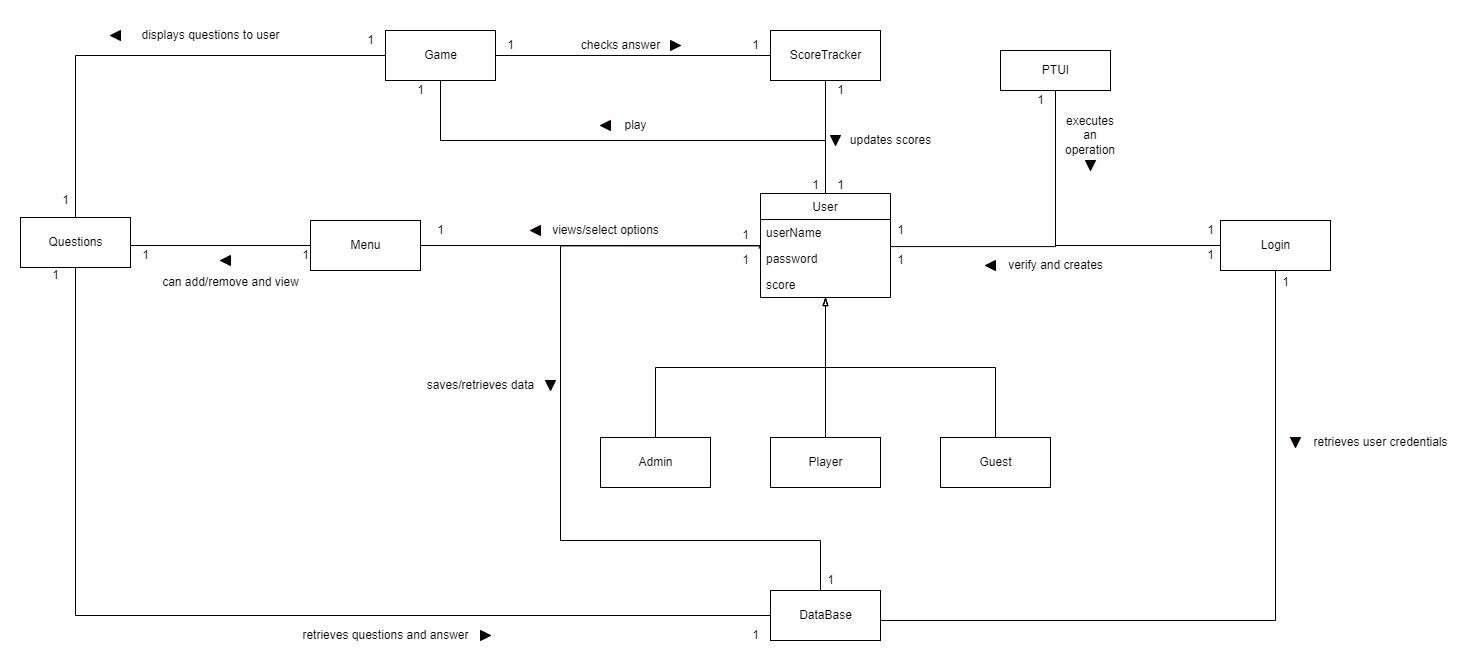
After identifying the requirements, a noun-verb analysis was conducted to determine the necessary classes and patterns for the game. A noun-verb analysis involves identifying the nouns (objects or entities) and verbs (actions or behaviors) in the requirements and using this information to design the classes and methods in the game.

Next, a domain model was created for the game. The domain model is a visual representation of the game's objects, their relationships, and the actions that can be performed on them. The domain model for Trivial Trivia included a database subsystem, a user authentication subsystem, and a command subsystem. The database subsystem contained a database sequence diagram, which showed the interactions between the database and other parts of the system. The user authentication subsystem contained a user authentication sequence diagram, which showed the process of authenticating a user's account. The command subsystem contained various sequence diagrams for different actions, such as playing the game, viewing stats, creating an account, adding questions, exporting questions, and importing player data.

Finally, the information gathered during the project was documented in a design document and GOF (Gang of Four) pattern cards were created for each subsystem. GOF pattern cards are a tool used to document design patterns, which are reusable solutions to common design problems.

All of this work was completed in Sprint 1, which is a time-boxed period during which specific tasks are completed and delivered. In software development, a sprint is typically a few weeks long and is used to break up the project into smaller, more manageable chunks.

# Domain Model

This section provides a domain model for the project. It should follow the guidelines discussed in class and the design project activity sheets. For it to be readable, you may need to turn this page into 

# System Architecture

This section provides a model of the subsystem components that make up the overall software architecture for the project. Draw the subsystems as simple boxes with relationships between them. Provide a narrative that describes the responsibilities of each component and the interfaces that are provided between subsystems.

# Subsystems: Database Subsystem

## Class Structure Diagram

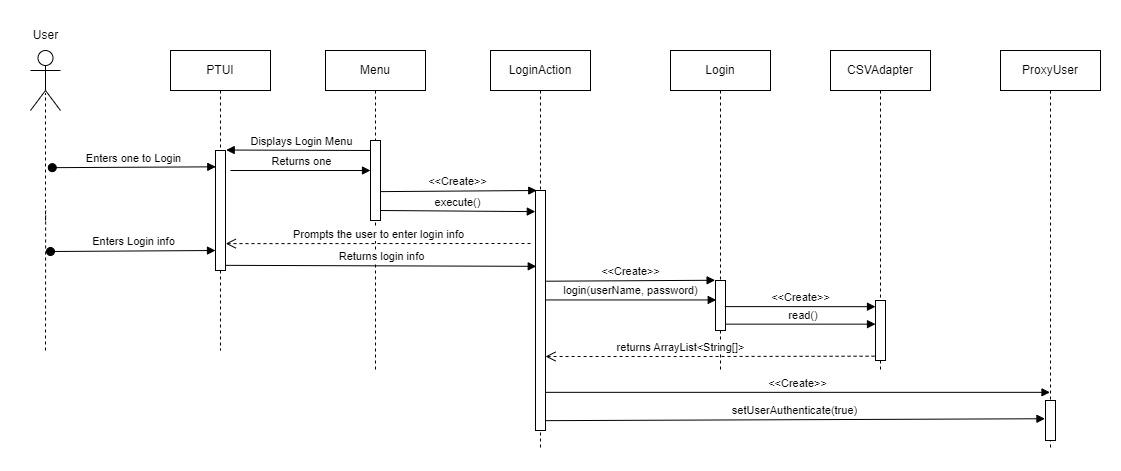
## 

# 

# 

# Sequence Diagram

Sequence Diagram for logging in using Database



| **Name:** Database Subsystem | | | **GoF pattern:** Adapter |
| --- | --- | --- | --- |
| **Participants:** | | | |
| **Class** | **Role in GoF pattern** | **Participant's contribution in the context of the application** | |
| Database | Adapter | Creates a layer of abstraction between the application API and the database API. This interface instance may be aggregated by other classes.The database complies with the open-close concept. | |
| CSVAdapter | ConcreteAdapter | In order to read and write CSV files, this class will make use of the functions provided by the CSV libraries. Additionally, it will implement the Database into operation. | |
| SQLAdapter | ConcreteAdapter | In order to read and write SQL files, this class will make use of the functions provided by the SQL libraries. Additionally, it will implement the Database into operation. | |
| ImportPlayerDataAction | ConcreteComponenet | Imports player data using the adapter pattern. Creates an instance of SQLAdapter for reading the file and an instance of CSVAdapter for writing the file. | |
| ExportPlayerDataAction | ConcreteComponenet | Exports player data using the adapter pattern. Creates an instance of CSVAdapter for reading the file and an instance of SQLAdapter for writing the file. | |
| ImportQuestionAction | ConcreteComponenet | Imports Question using the adapter pattern. Creates an instance of SQLAdapter for reading the file and an instance of CSVAdapter for writing the file. | |
| ExportQuestionAction | ConcreteComponenet | Exports Question using the adapter pattern. Creates an instance of CSVAdapter for reading the file and an instance of SQLAdapter for writing the file. | |
| ViewStatAction | ConcreteComponenet | It creates a new instance of the CSVAdapter, which is then used to read the player stats and display the player's stats. | |
| AddQuestionAction | ConcreteComponenet | User will enter Questions and Answers which will then be converted into an array list to be passed into a CSVAdapter. | |
| LoginAction | ConcreteComponenet | When user enter their login credentials, it calls the login function which then creates an instance of the CSVAdapter which will read then return a array list. | |
| CreateAcountAction | ConcreteComponenet | User will enter username and password which will then be converted into an array list to be passed into a CSVAdapter. | |
| PlayAction | ConcreteComponenet | displays the questions and answer options which are then used to check the user’s answer with the stored answer. | |
| **Deviations from the standard pattern:** | | | |
| **Requirements being covered:** It saves the user's data as well as the game's data, and it also handles the importing and exporting of data. | | | |

# 

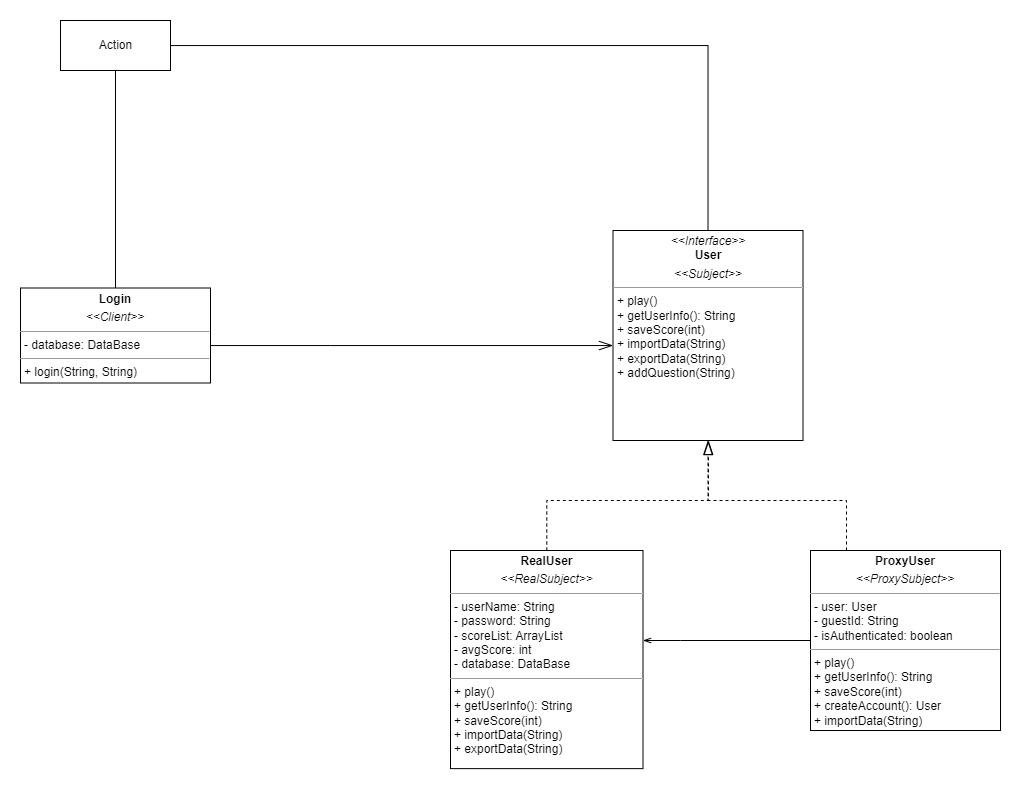
# 

# 

# 

# Subsystem: User Authentication

# Class Structure Diagram

****

## 

## 

## 

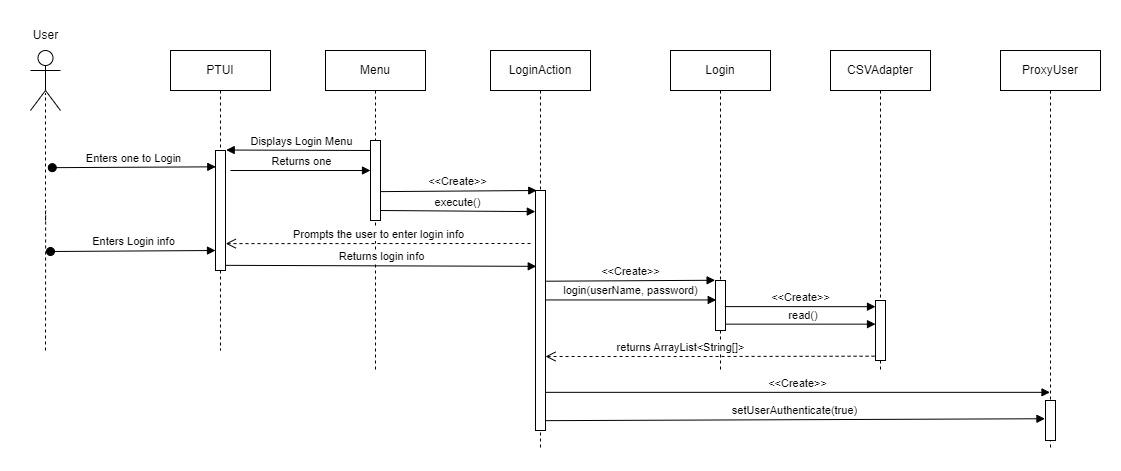
## 

## 

## 

## Sequence Diagram

User logging in



# 

# GoF Card

| **Name: User Authentication Subsystem** | | | **GoF pattern: Proxy** |
| --- | --- | --- | --- |
| **Participants** | | | |
| **Class** | **Role in GoF pattern** | **Participant's contribution in the context of the application** | |
| Login | Client | Creates the object of the user using the concrete proxy. Authenticates the type of user using the information provided and determines the amount of access a specific user has. | |
| User | Subject | Defines the sets of behaviors for the RealUser and ProxyUser. This interface allows RealUser to be used anywhere a ProxyUser is used. All of the concrete proxy implements the behaviors but with different levels of features. | |
| RealUser | Real Subject | Implements the subject interface behaviors and limits the behavior that the real user has. Allows Proxy Subject to access the functions within the real user. It also provides behaviors for users to use it. | |
| ProxyUser | Proxy Subject | Limits and control the access of the RealUser. It maintains a reference (object) of the RealUser that gives access to all the RealUser functions. It checks the level of access a given user have and gives that level of access to the functionality. If a user is authenticated it provides access to the RealUser functions as well. If the user is not authenticated, it can only access guest functionality. | |
| **Deviations from the standard pattern: None** | | | |
| **Requirements being covered: Covers Multiple user type requirements. This pattern allows limiting the functionality access to each type of user. It also covers login requirements as well.** | | | |

# 

# Command Subsystem

## Class Structure Diagram

# 

## 

## 

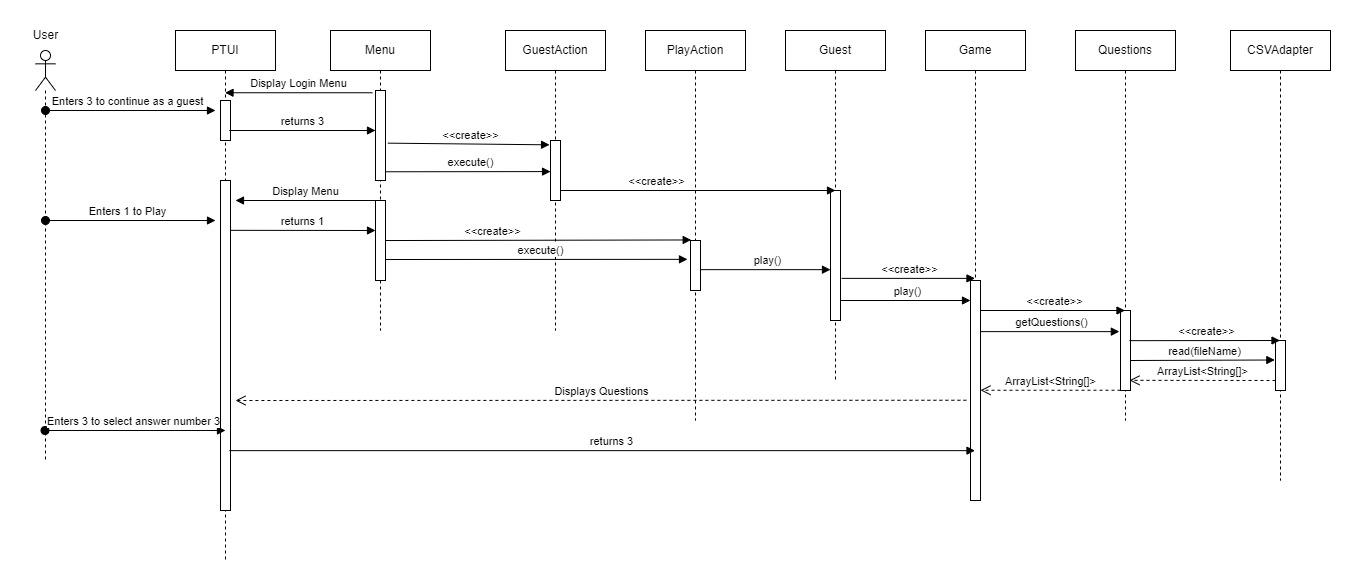
## 

## 

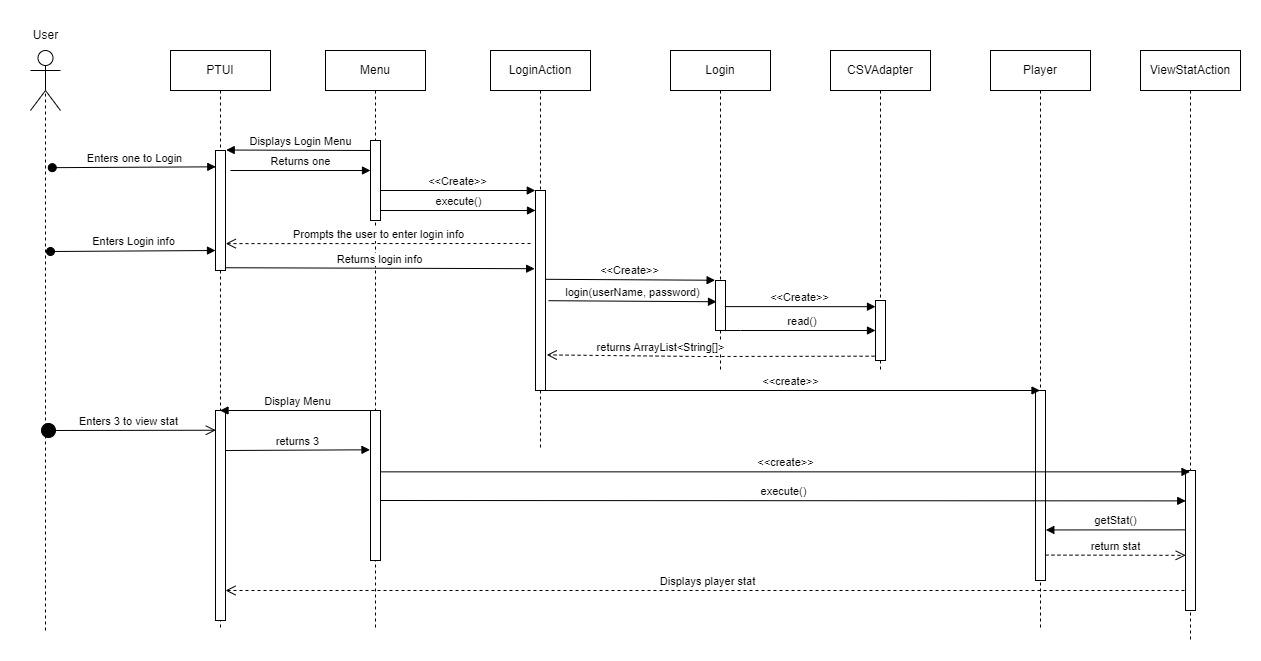
## 

## Sequence Diagram

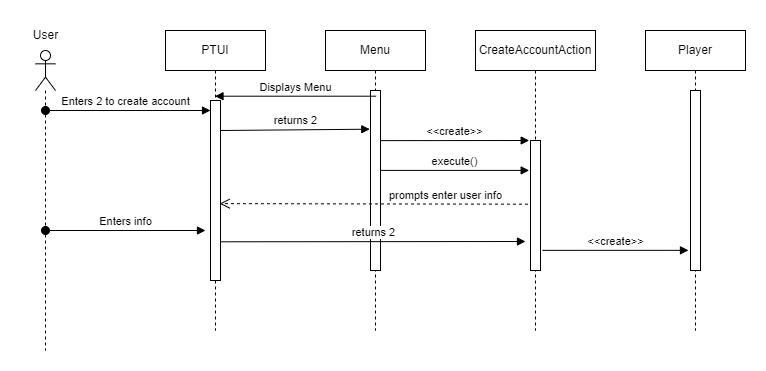
Sequence diagram for the user playing



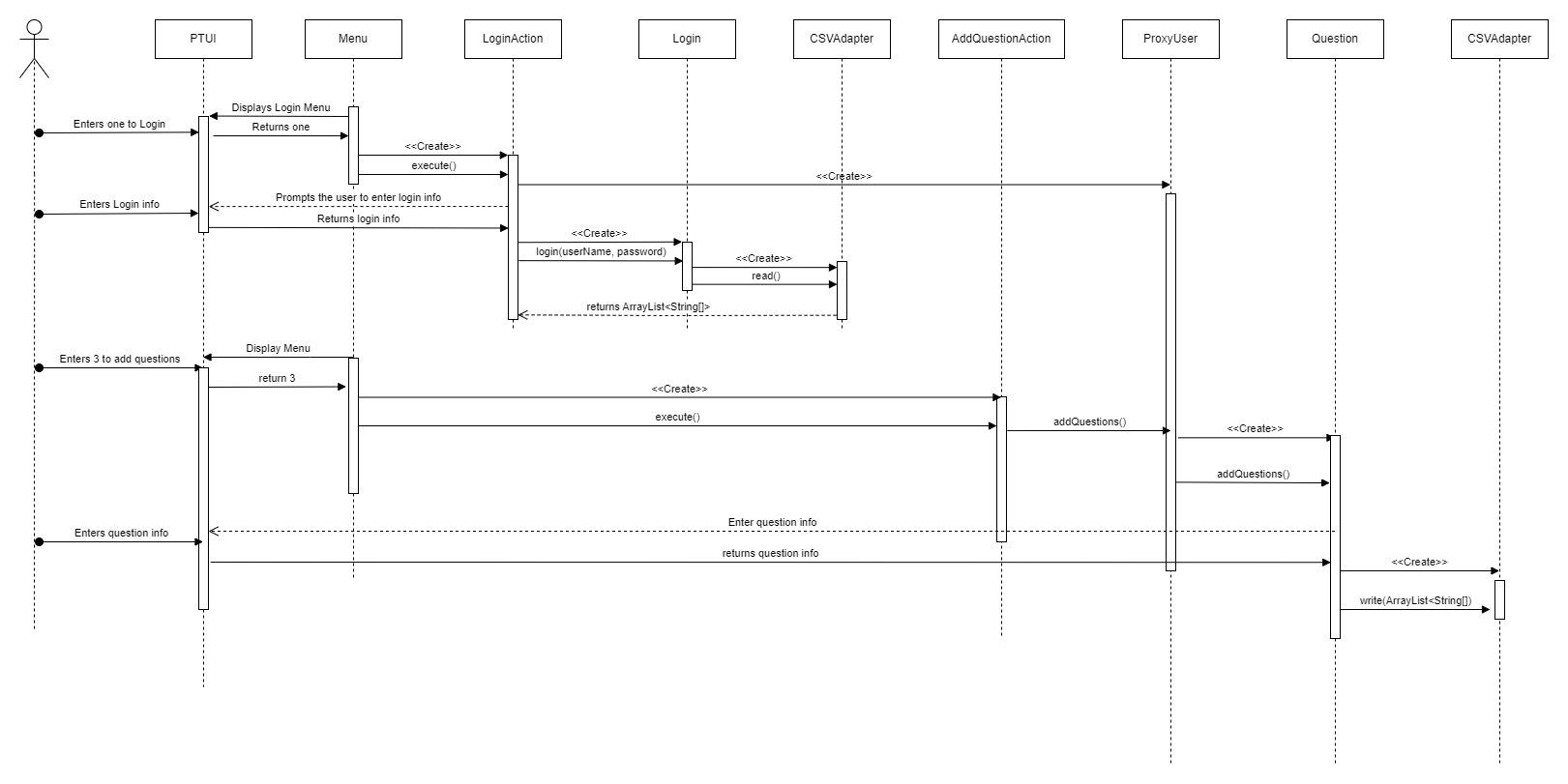
Sequence diagram for the user viewing their stat



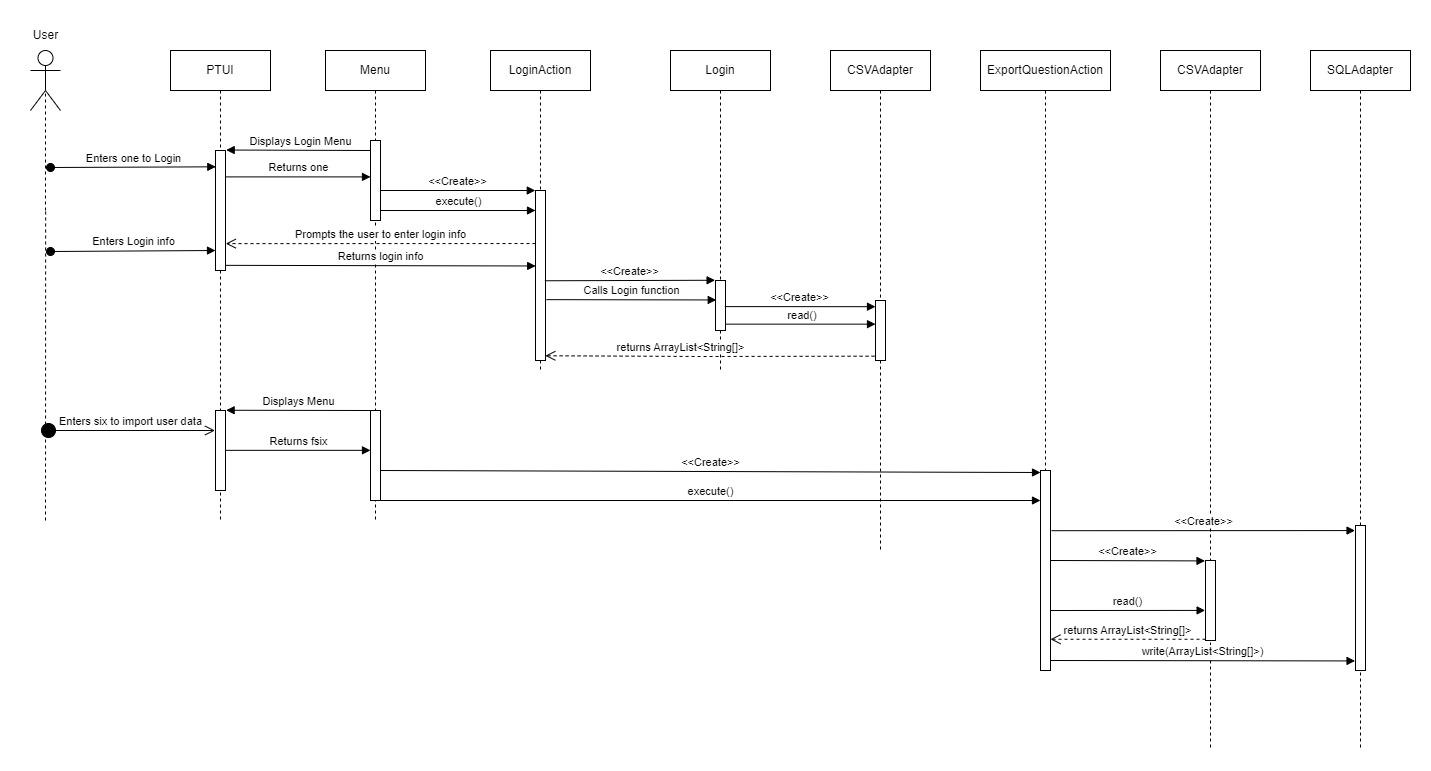
Sequence diagram for user creating their account



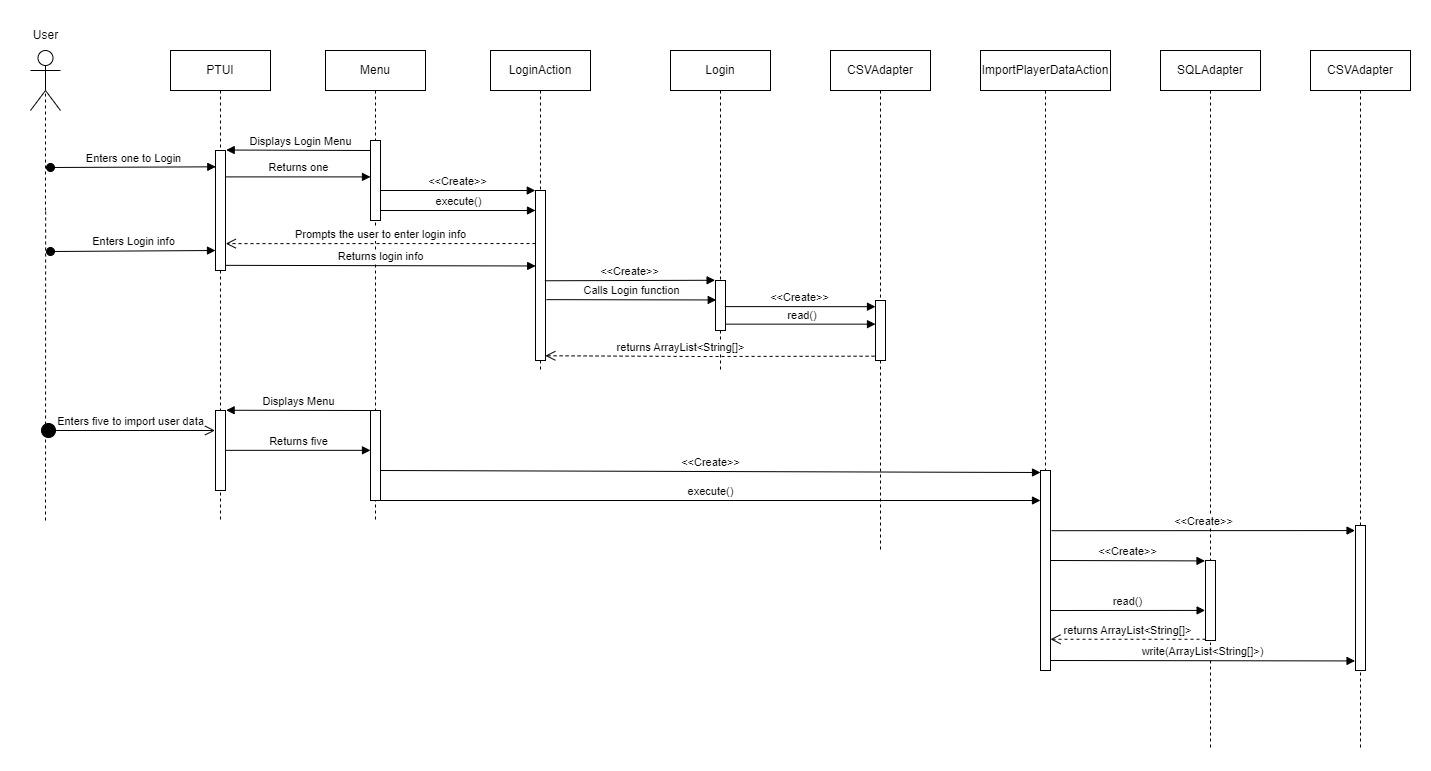
Sequence diagram for user adding question



Sequence diagram for Exporting questions



Sequence diagram for Importing player data



# GoF Card

| **Name: Command Subsystem** | | | **GoF pattern: Command** |
| --- | --- | --- | --- |
| **Participants** | | | |
|  | | | |
| **Class** | **Role in GoF pattern** | **Participant's contribution in the context of the application** | |
| Menu | Invoker | Provides the list of actions the user can perform and depend on the options the user picks it creates the command instance to perform that command. | |
| User | Receiver | A reference of User is provided in the concrete commands. Knows how to perform the operations associated with the commands. Is an information expert for user data as well as play data. Has the ability to save user data. | |
| Action | Command | Defines the interface for user action in the game application. The execute method is invoked each time that the command should perform its related task. | |
| ViewStatAction | Concrete Command | A concrete command that performs a view stat action by calling the method getUserInfo() from the User Class. This function returns User stat as well as other user info which is displayed. | |
| AddQuestionAction | Concrete Command | A concrete command that performs an add question action by calling the RealUser function addQuestion. This will create a reference of the Question class and uses its function to add new questions to the game database. | |
| PlayAction | Concrete Command | A concrete command that allows all types of users to play the quiz game. The user doesn’t have to be authenticated to use this command. This command calls the ProxyUser function play in order to initiate the play sequence. | |
| LoginAction | Concrete Command | A concrete command that allows user to login into their account to save and retrieve their data. This command uses ProxyUser login function to check the user’s credentials (username and password) to authenticate the user. Once the user has been authenticated their level of access increases to the functionality | |
| GuestAction | Concrete Command | A concrete command that creates the guest user or ProxyUser. This command limits the functionality of the user and only allows the user to play or create an account. If they try to perform other actions it displays user has not been authenticated yet. | |
| CreateAccountAction | Concrete Command | A concrete command that allows user to create an account. Once user provides the user info to create the account it saves the data into the database and marks the user as an authenticated user. | |
| ExportQuestionAction | Concrete Command | Exports Question using the adapter pattern. Creates an instance of CSVAdapter for reading the file and an instance of SQLAdapter for writing the file. | |
| ImportQuestionAction | Concrete Command | Imports Question using the adapter pattern. Creates an instance of SQLAdapter for reading the file and an instance of CSVAdapter for writing the file. | |
| ImportPlayerDataAction | Concrete Command | Imports player data using the adapter pattern. Creates an instance of SQLAdapter for reading the file and an instance of CSVAdapter for writing the file. | |
| ExportPlayerDataAction | Concrete Command | Exports player data using the adapter pattern. Creates an instance of CSVAdapter for reading the file and an instance of SQLAdapter for writing the file. | |
| **Deviations from the standard pattern:** | | | |
| **Requirements being covered:** This subsystem covers all the action an authenticated or non-authenticated users can performs. | | | |

# Status of the Implementation

Provide a complete description of the status of your implementation. This should specify all known defects in the system, and indicate requirements that your implementation does not cover.

# Appendix

This section provides fine-grained design details for all of the classes in your design. You will capture this information using the CRC (Class-Responsibilities-Collaborators) card format below.

| **Class:** MyClass1 |  |
| --- | --- |
| **Responsibilities:** ... |  |
| **Collaborators:** ... |  |
| **Users:** ... | **Used by:** ... |
| **Author:** ... |  |