Multiplayer Blackjack Game

*Design Document*

*Revision History*

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Revision** | **Description** | **Author** |
| Fri March 14 | v1 | Worked on determining common application events  Also worked on some basic use cases related to players and dealers | Riley, Martin |
| Sun March 16 | v2 | Added more messages that the server should have.  Worked on a GUI prototype in excalidraw   * Includes **Login Screen (Client), Table Selection Screen, Game Screen** * Did a lot of work in excalidraw working out the flow of the application visually to determine what needs to happen | Riley |
| Tue March 18 | v3 | Went over the excalidraw GUI prototype to fill in some gaps, and extrapolate more niche use cases that aren’t as obvious | Basim, Riley, Martin, Michelle, John |
| Thur March 20 | v4 | Added more use cases for the dealer and the player (stand, blackjack, split, midgame leave, view table, draw card, pay player, shuffle) | Basim, Riley, Michelle, Martin |
| Fri March 21 | v4.1 | No direct changes to this document, but significant work on the supporting Class Diagrams, Use Case Diagrams, and Sequence Diagrams to reflect what we’ve been talking about | Riley |
| Tue March 25 | v4.2 | No direct changes to this document.  Worked on some client GUI UML class diagrams | Riley |
| Mon Apr 7 | v5 | Added Add Funds, View Funds, View Funds History, Withdraw Funds to this document, Added some features, change and fix for Game Flow For Player,Game Flow For Dealer, Login Flow, Access Account Flow in Use Case Diagrams | John |
| Mon Apr 10 | v6 | The design document is officially finished here | Riley, John, Michelle, Martin, Basim |

# Table of Contents

[Table of Contents 4](#_Toc195206570)

[Purpose 8](#_Toc195206571)

[System Architecture 9](#_Toc195206572)

[Database Design 10](#_Toc195206573)

[1. Storage Method 10](#_Toc195206574)

[2. File-Based Data Storage Structure 10](#_Toc195206575)

[Users File (users.csv) 10](#_Toc195206576)

[Game Session File (games.csv) 10](#_Toc195206577)

[Game History File (game\_history.csv) 11](#_Toc195206578)

[Leaderboard File (game\_history.csv) 11](#_Toc195206579)

[3. Data Access & Manipulation 11](#_Toc195206580)

[Reading Data 12](#_Toc195206581)

[Writing Data 12](#_Toc195206582)

[Updating Records 12](#_Toc195206583)

[GUI Design 13](#_Toc195206584)

[1. Client Interface Overview 13](#_Toc195206585)

[2. GUI Layouts & Interactions 13](#_Toc195206586)

[Login & Account Flow 13](#_Toc195206587)

[Bet Window Flow 13](#_Toc195206588)

[Table Lobby (Player) 14](#_Toc195206589)

[Game Table (Player View) 14](#_Toc195206590)

[Game Table (Dealer View) 14](#_Toc195206591)

[3. Features 14](#_Toc195206592)

[Server Design 15](#_Toc195206593)

[1. Architecture Overview 15](#_Toc195206594)

[2. Server Requirements 16](#_Toc195206595)

[Multithreaded 16](#_Toc195206596)

[Socket Based 16](#_Toc195206597)

[Real-Time 16](#_Toc195206598)

[Accurate 16](#_Toc195206599)

[3. Client Requirements 16](#_Toc195206600)

[Multithreaded 16](#_Toc195206601)

[Responsive 16](#_Toc195206602)

[Error Handling 17](#_Toc195206603)

[User-Friendly UI 17](#_Toc195206604)

[Accurate 17](#_Toc195206605)

[Game Initialization 17](#_Toc195206606)

[Player Actions 17](#_Toc195206607)

[Dealer Actions 17](#_Toc195206608)

[Game Resolution 17](#_Toc195206609)

[Betting & Scorekeeping 18](#_Toc195206610)

[User Interface Events 18](#_Toc195206611)

[Create Account 19](#_Toc195206612)

[Login 19](#_Toc195206613)

[Join Table 20](#_Toc195206614)

[Play Game 20](#_Toc195206615)

[Blackjack 21](#_Toc195206616)

[Hit 21](#_Toc195206617)

[Stand 22](#_Toc195206618)

[Split 22](#_Toc195206619)

[View History 23](#_Toc195206620)

[Access Account 23](#_Toc195206621)

[Add Funds 24](#_Toc195206622)

[View Funds 24](#_Toc195206623)

[View Funds History 25](#_Toc195206624)

[Withdraw Funds 25](#_Toc195206625)

[Midgame Leave 26](#_Toc195206626)

[End Game/Quit 26](#_Toc195206627)

[View Leaderboard 27](#_Toc195206628)

[Dealer High Level Activities 28](#_Toc195206629)

[Login 28](#_Toc195206630)

[View Tables 28](#_Toc195206631)

[Create New Table 29](#_Toc195206632)

[Draw Card(Deal Cards) 29](#_Toc195206633)

[Pay player 30](#_Toc195206634)

[Shuffle 30](#_Toc195206635)

[Midgame Leave 31](#_Toc195206636)

[Server Activities 32](#_Toc195206637)

[Startup 32](#_Toc195206638)

[Handle Connections 33](#_Toc195206639)

[Handle Client Requests 33](#_Toc195206640)

[Message Types 34](#_Toc195206641)

[Login Dealer 34](#_Toc195206642)

[Create Account Player 34](#_Toc195206643)

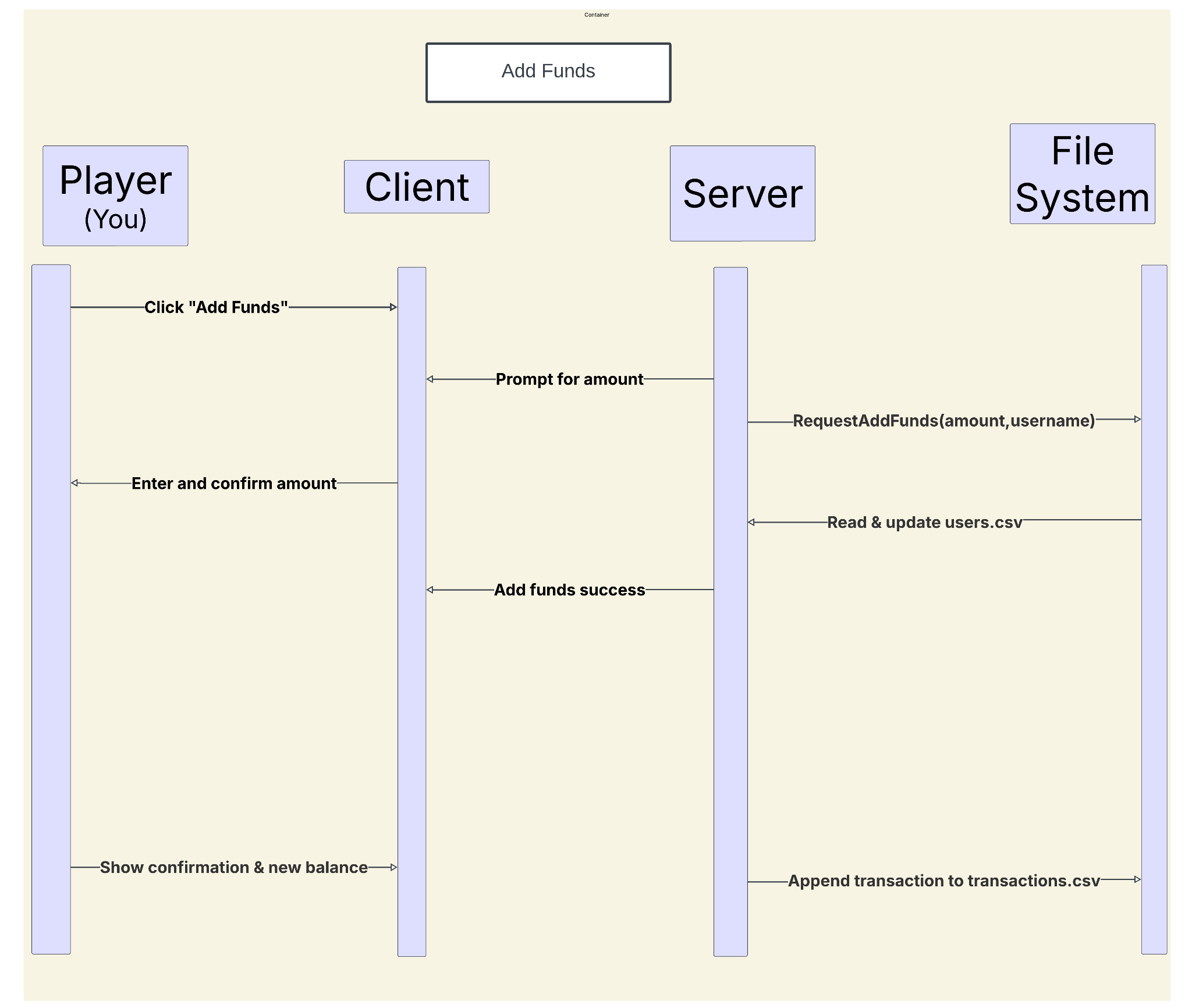
[Join Table 35](#_Toc195206644)

[GUI Prototypes 36](#_Toc195206645)

[UML Class Diagrams 37](#_Toc195206646)

[UML Use Case Diagrams 40](#_Toc195206647)

[UML Sequence Diagrams 43](#_Toc195206648)

[ 43](#_Toc195206649)

[Timeline 48](#_Toc195206650)

[GitHub Repo Link 49](#_Toc195206651)

# Purpose

The purpose of this document is to outline the design and architecture of the Multiplayer Blackjack Game, detailing the core system components, interactions, and implementation strategies. This document serves as a blueprint for developers, ensuring a structured approach to the software’s development and maintenance.

**The key objectives of this document are:**

1. To define the system architecture, including client-server interactions and database structure.
2. To provide details on key functionalities such as account management, multiplayer game logic, and server handling.
3. To describe the message flow and interactions between different components.
4. To ensure security, performance, and scalability by outlining proper design strategies.
5. To establish a foundation for implementation, testing, and deployment of the system.

# System Architecture

The Multiplayer Blackjack Game follows a client-server architecture, where multiple clients (players and dealers) connect to a central game server.

**Key Components:**

1. Client Application: Multithreaded, GUI-based interface for players and dealers to interact with the game.
2. Game Server: Multithreaded, manages game states, processes player actions, and ensures real-time synchronization.
3. Database: Stores user accounts, game history, and leaderboards.
4. Networking: Uses TCP/IP for communication between clients and the server.

# Database Design

## Storage Method

The game will store data in local files rather than a database server.

|  |  |
| --- | --- |
| Storage Type | Use Case |
| CSV | Used for all structured data — like user accounts, game sessions, history, and leaderboards. |
| Text Files | Minimalistic option for logging events |

## File-Based Data Storage Structure

### Users File (users.csv)

Stores player and dealer accounts in a CSV file.

**Operations:**

* Adding a User: Append a new entry to the CSV file.
* Updating Balance: Modify the existing user’s balance.
* Authentication: Read the file and verify username/password.

**Example (users.csv):**

username, password

riley, supersecurepassword

michelle, anothersupersecurepassword

**Example (riley\_stats.csv)**

gameid,betamount,type{0: loss, 1: win, 2: forfeit}

0,130,1

1,140,0

**Example (michelle\_stats.csv)**

gameid,betamount,type{0: loss, 1: win, 2: forfeit}

0,130,1

1,140,0

### Game Session File (games.csv)

Tracks active and completed blackjack games.

**Operations:**

* Start a Game: Add a new row when a dealer starts a game.
* Update Game Status: Modify the status when the game ends.
* Retrieve Game Data: Search for specific games by ID.

**Example:**

game\_id,dealer\_id,table\_id,status,active\_players

101,michelle,5,in\_progress,3

102,riley,2,completed,4

### Game History File (game\_history.csv)

Stores the outcome of completed blackjack games.

**Operations:**

* Log Game Results: Append new game outcomes.
* Retrieve Player History: Search for past games by player name.

**Example:**

game\_id,username,result,bet,earnings

102,martin,win,100,150

102,john,loss,100,0

102,michelle,win,200,300

### Leaderboard File (game\_history.csv)

Stores player rankings based on earnings.

**Operations:**

* Update Player Stats: Modify win/loss count and earnings.
* Retrieve Leaderboard: Sort by total earnings.

**Example:**

username,wins,losses,total\_earnings

basim,5,3,800

riley,7,2,1050

michelle,3,6,300

## Data Access & Manipulation

Since there’s no database being used, the system will handle data manually using file I/O operations.

### Reading Data

* Open the CSV file line by line.
* Split each line by commas to get the fields.
* Use the data (like username or balance) in the program.

### Writing Data

* Open the file in append mode.
* Add a new line with the data you want to save.
* Close the file after writing.

### Updating Records

* Read the entire file into memory.
* Change the specific line or value you want to update.
* Overwrite the file with the updated data.
* Save everything back into the same CSV file.

# GUI Design

## Client Interface Overview

|  |  |
| --- | --- |
| Screen | Purpose |
| Account Screen | Create account and login interface for both players and dealers. |
| Table Selection Screen | Shows available tables (ID, size, dealer/player info), allows join/create options. |
| Game Table Screen | Main gameplay screen: cards, dealer hand, player actions (Hit, Stand, Split), and UI status updates. |
| Dealer Dashboard | Shows active tables, number of dealers, player status, and controls to create tables or request game state info. |
| Other player Leaderboard | Show the ID, name, score,rank, or highScore of other Players |
| Bet Window | Popup window where players will enter their bet before a game starts, and again at the |
|  | end of the game to start a new round. |

## GUI Layouts & Interactions

### Login & Account Flow

* Player launches app -> Login/Create Account
* Dealer launches app -> Login (assumed that they already have an account)
* Username/password sent to server
* Server response determines access to Lobby

### Bet Window Flow

* When it appears:
  + Right after joining a table and before the cards are dealt
  + After a round ends and players chose to continue
* Functionality:
  + Input box or button selection for chip value (e.g., $5, $10, $25, custom)
  + Confirmation button (“Place Bet” or “Continue)
  + If player does not place a bet, they stay seated but skip the round (NOT FINAL)
* Server Communication:
  + Sends bet amount to server
  + Triggers game logic to proceed only after all players have placed bets

### Table Lobby (Player)

* Displays multiple tables: ID, buy-in amount(e.g., $100,500), # of players
* Players select tables -> request sent to server -> response updates UI

### Game Table (Player View)

* Top: Dealer and player cards (circle for each card)
* Middle: Current bet, chip selection (e.g., $1, $10, $100)
* Bottom: Action buttons - Hit, Stand, Split, Leave

### Game Table (Dealer View)

* Displays all players and their cards
* Controls for Shuffle, Deak, End Round, Pay Player
* Ability to kick/ban disruptive players or close the tabl

## Features

|  |  |
| --- | --- |
| Feature | Description |
| Responsive Design | Multithreaded client responds to both server and user events in real time |
| Chip Selection UI | Player selects chips via buttons; server receives total bet amount |
| Dynamic UI Update | UI reflects server responses instantly (new card, updated hand, turn change) |
| Table Status Sync | Client sync table status, chips, and actions every turn |
| Bet Prompt | Bet window enforces the betting phase before starting/continuing a game |

# 

# 

# Server Design

## Architecture Overview

|  |  |
| --- | --- |
| Component | Description |
| Connection Manager | Accepts client socket connections and assigns each to a new thread. |
| Request Handler | Reads messages sent from clients and figures out what they’re asking for (like login, join table, hit, etc). Then it sends the request to the right part of the code to handle it. |
| Game Engine | Runs the actual Blackjack game - deals cards, checks if someone won, handles turns, and updates the game state. |
| Table Manager | Keeps track of all the tables (who’s at which one, open seats, etc). |
| Account Manager | Handles login, account creation,and reads user info from files. |
| History Logger | Records completed games, updates the leaderboard and stores each player;s win/loss history in files for later access.Saves game results and updates the leaderboard so players can see their stats. |

## Server Requirements

### Multithreaded

* The server must be multithreaded to allow for multiple clients to be managed simultaneously.
* Without simultaneous client handling, we can’t have a multiplayer game experience.

### Socket Based

* The server is hosted to the network using sockets.
* This allows clients to discover the open port, connect, and start interacting with the game right away.

### Real-Time

* The server must respond to client requests as they come in, with very little delay.
* This helps make sure gameplay feels smooth and doesn’t lag when people take actions.

### Accurate

* The server makes sure the game state stays correct at all times.
* All clients (players and the dealer) should see the same thing, like who’s turn it is or what cards are on the table.

## Client Requirements

### Multithreaded

* The client should be multithreaded to handle incoming server responses without freezing or blocking the user interface.
* Without multithreading, the client would hang or lag due to constant network communication.

### Responsive

* The client must respond quickly to both player actions (like button clicks) and server messages (like game updates).
* When the server sends updates (like cards dealt, new player joined, etc.), the client should update the UI immediately.

### Error Handling

* The client should detect and correctly handle errors like failed server connections, invalid inputs, or empty responses.
* Users should see a helpful error message (not a crash or unhandled exception).

### User-Friendly UI

* The interface should be intuitive, with clearly labeled buttons and consistent layout
* Player actions (Hit,Stand,Bet,etc.) must be easy to perform and provide feedback.

### Accurate

* The client should accurately reflect the game state based on server responses
* This includes showing cards, chip counts, player turns, and win/loss results in real time

Common Application Events

## Game Initialization

* Players start a new game
* Deck shuffled
* Players and dealers are dealt random cards

## Player Actions

* Hit: Player request another card if they don’t get enough point
* Stand: Players keep current hand
* Double Down: Players double the bet and receives card
* Split: Players splits 2 same-value cards into separate hands
* Ready:Ready for the game and wait other players ready

## Dealer Actions

* Dealer reveals their face-down card
* Draw cards
* Check Card Depend on Rules

## Game Resolution

* Bust: Player or dealer exceeds 21
* Blackjack: A hand with an Ace and a 10-value card wins automatically
* Win/Loss Calculation: Determines if the player wins, loses, or ties based on hand values

## Betting & Scorekeeping

* Players bet before game starts
* Winning are calculated and added to the player’s balance
* For each player if they run out of money which means game is over for them

## User Interface Events

* Button for “Hit”, “Stand”, “Double Down”
* Card animation
* Players image
* Dealer image
* Displaying results and game message

### Create Account

**Relevant Requirements:** Login

**Actor:** Player

**Pre-Conditions:** None

**Post-Conditions:** There should be a new account entry in the “database” file

**Main Flow:**

* Fill in username and password fields
* Submit to server to create the account
* Server responds with status
* Client reacts based on the status

**Alternative Flow:**

**Exceptions:** Fail to write to the database file?

**Related Use Cases: Login, Access Account, View History**

### Login

**Relevant Requirements:** Login

**Actor:** Player, Dealer

**Pre-Conditions:** Account exists in database, valid credentials

**Post-Conditions:** Allows users to access the lobby

**Main Flow:**

* Enter valid username and password
* Attempt to log in.
* Brought to the appropriate game perspective

**Alternative Flow:** Error message (notify the client to try again)

**Exceptions:**

* The server is offline and is not accessible.
* If the username doesn’t exist in the “database”, deny entry
* If the password doesn’t match the password associated with the username, deny entry

**Related Use Cases: Create Account, Join Table, Access Account, Play Game**

### Join Table

**Relevant Requirements:** Login

**Actor:** Player

**Pre-Conditions:** Logged in,

**Post-Conditions:** Allows users to access the lobby

**Main Flow:**

* Enter valid username and password
* Attempt to log in.
* Brought to the appropriate game perspective

**Alternative Flow:** Error message (notify the client to try again)

**Exceptions:**

* The server is offline and is not accessible.
* If the username doesn’t exist in the “database”, deny entry
* If the password doesn’t match the password associated with the username, deny entry

**Related Use Cases: Login, Play Game, Midgame leave/disconnect**

### Play Game

**Relevant Requirements:**

**Actor:** Player and Dealer

**Pre-Conditions:** Must have an account, must be logged in

**Post-Conditions:** After successful login, able to join a game

**Main Flow:**

* Logged in
* Select join game
* Participate in the table game logic

**Alternative Flow:** Insufficient funds.(Min buy in)

**Exceptions:** Server is offline, insufficient funds.

**Related Use Cases:** Hit, Stand, Split

### Blackjack

**Relevant Requirements:**

**Actor:** Player

**Pre-Conditions:** Must have an account, must be logged in, must be in a game, must

get a hand value of 21 (aka blackjack)

**Post-Conditions:** Player gets an immediate payout

**Main Flow:**

* Logged in
* Select join game
* Participate in the table game logic?

**Alternative Flow:** Insufficient funds.(Min buy in)

**Exceptions:** Server is offline, insufficient funds.

**Related Use Cases:** Play game

### Hit

**Relevant Requirements:**

**Actor:** Player

**Pre-Conditions:** Must have an account, must be logged in, must be in a game

**Post-Conditions:** Player should have another card

**Main Flow:**

* Logged in
* Select join game
* Participate in the table game logic

**Alternative Flow:** Insufficient funds.(Min buy in)

**Exceptions:** Server is offline, insufficient funds.

**Related Use Cases:** **Play Game, Stand, Split**

### Stand

**Relevant Requirements:**

**Actor:** Player

**Pre-Conditions:** Must have an account, must be logged in, must be in a game

**Post-Conditions:** Player should end their turn with the same hand

**Main Flow:**

* Logged in
* Select join game
* Participate in the table game logic

**Alternative Flow:** Insufficient funds.(Min buy in)

**Exceptions:** Server is offline, insufficient funds.

**Related Use Cases:** **Hit, Play Game**

### Split

**Relevant Requirements:**

**Actor:** Player

**Pre-Conditions:** Must have an account, must be logged in, must be in a game, must

have a hand of two cards that are identical in “number”

**Post-Conditions:** Player should gain another hand of cards, each hand consisting of a

card from the Player’s previous hand

**Main Flow:**

* Logged in
* Select join game
* Participate in the table game logic

**Alternative Flow:** Insufficient funds.(Min buy in)

**Exceptions:** Server is offline, insufficient funds.

**Related Use Cases:** **Hit, Stand, Play Game**

### View History

**Relevant Requirements:** Login

**Actor:** Player

**Pre-Conditions:** Must have an account, Logged in

**Post-Conditions** After successful login, have access to history

**Main Flow**

* Logged in
* Select view history
* View history
* Download history

**Alternative Flow:** No game data.

**Exceptions**: Server is offline, no game data

**Related Use Cases:** Access Account, Create Account, Game History File (for backend)

### Access Account

**Relevant Requirements:** Login

**Actor:**  Player

**Pre-Conditions:** Logged into an account, not actively playing a game

**Post-Conditions:**  The player should now be able to *view* their account details. No editing.

**Main Flow:** Click button to view account

**Alternative Flow:**  None

**Exceptions:** Server is offline

**Related Use Cases:** Play Game

### Add Funds

**Relevant Requirements:** Access Account

**Actor:**  Player

**Pre-Conditions:** Logged into an account,accessing account section

**Post-Conditions:**  Player's account balance is increased by the added amount.

**Main Flow:**

* Click button to view account
* Select "Add Funds" option
* Enter amount to add
* Select payment method
* Confirm transaction

**Alternative Flow:**  Player cancels transaction before confirming

**Exceptions:** Payment is error, server is offline, player’s cards have not enough

**Related Use Cases:** Access Account, View Funds, View Funds History, Withdraw Funds

### View Funds

**Relevant Requirements:** Access Account

**Actor:**  Player

**Pre-Conditions:** Logged into an account

**Post-Conditions:**  Player views current account balance

**Main Flow:**

* Click button to view account
* Select "View Funds" option
* Current balance is displayed on account dashboard

**Alternative Flow:**  None

**Exceptions:** server is offline

**Related Use Cases:** Access Account, View Funds History, Withdraw Funds

### View Funds History

**Relevant Requirements:** Access Account

**Actor:**  Player

**Pre-Conditions:** Logged into an account, previous transaction

**Post-Conditions:**  Player views history of account transaction

**Main Flow:**

* Click button to view account
* Select "Transaction History" option
* Display list of previous transaction that player withdraw or add

**Alternative Flow:**  Filter transaction by date or type

**Exceptions:** server is offline

**Related Use Cases:** Access Account, View Funds, Withdraw Funds

### Withdraw Funds

**Relevant Requirements:** Access Account

**Actor:**  Player

**Pre-Conditions:** Logged into an account, has balance greater than limit of withdraw

**Post-Conditions:**  Withdraw funds from account for cash or deposit in bank

**Main Flow:**

* Click button to view account
* Select "Withdraw Funds" option
* Enter amount to withdraw
* Select withdrawal method
* Confirm withdrawal

**Alternative Flow:**  Player cancels withdrawal before confirming

**Exceptions:** server is offline, not enough funds, payment processing error

**Related Use Cases:** Access Account, View Funds, View Funds History

### Midgame Leave

**Relevant Requirements:** Login

**Actor:**  Player

**Pre-Conditions:** Logged into an account, actively playing a game

**Post-Conditions:**  Player loses bet, game continues for all other players

**Main Flow:**

**Alternative Flow:**  None

**Exceptions:**

**Related Use Cases:** Play Game

### End Game/Quit

**Relevant Requirements:** Safe Disconnect

**Actor:**  Player or Dealer

**Pre-Conditions:** User is logged in

**Post-Conditions:**

* Cleanly disconnects from server, updates logs if needed

**Main Flow:**

* Click “Quit” from client menu
* Server disconnects client
* Safe exit

**Alternative Flow:**

**Exceptions:**

* Server already closed
* Force quit (crash)

**Related Use Cases:** Midgame Leave, Play Game

### 

### View Leaderboard

**Relevant Requirements:** Login

**Actor:**  Player

**Pre-Conditions:** Logged into a player account,access table

**Post-Conditions:**

* Leaderboard is displayed ID, name, rank, score of other players.

**Main Flow:**

* Select “Leaderboard” from menu
* Access Table
* Show the ID, name, rank, score of other players.
* Request data from server
* Display in table format

**Alternative Flow:**  No player data available

**Exceptions:**

**Related Use Cases:** Play Game

## Dealer High Level Activities

### Login

**Relevant Requirements:** Login

**Actor:**  Dealer

**Pre-Conditions:**

* Server running
* Logged into an account
* Not actively playing a game (?)

**Post-Conditions:**  The dealer can now create tables, and participate in games

**Main Flow:**

* Send login request to server
* Wait until server verifies login

**Alternative Flow:**  None

**Exceptions:** Server is offline

**Related Use Cases:** Play Game

### View Tables

**Relevant Requirements:** Login

**Actor:**  Dealer

**Pre-Conditions:**

* Connected to server
* Logged into an account
* Not actively playing a game
* Active games are being played

**Post-Conditions:**

**Main Flow:**

* Login
* Window showing the players in each table.

**Alternative Flow:**  None

**Exceptions:** Server is offline

**Related Use Cases:** Play Game

### Create New Table

**Relevant Requirements:** Login

**Actor:**  Dealer

**Pre-Conditions:**

* Connected to server
* Logged into a dealer account
* Not actively playing a game

**Post-Conditions:**

* New table on server
* Clients can see the new table and join it

**Main Flow:**

* Click button to view account
* Send request for player list
* Wait for the player list
* Open window showing the players in the table

**Alternative Flow:**  None

**Exceptions:** Server is offline

**Related Use Cases:** Login

### Draw Card(Deal Cards)

**Relevant Requirements:** Login

**Actor:**  Dealer

**Pre-Conditions:**

* Connected to server
* Logged into a dealer account
* Actively playing a game

**Post-Conditions:**

* The active user (dealer or player) should gain a new card from the deck

**Main Flow:**

* All bets are placed.
* Deal cards to each player and dealer

**Alternative Flow:**

* Player decides to hit.

**Exceptions:** Server is offline

**Related Use Cases:** Play Game

### 

### Pay player

**Relevant Requirements:** Login

**Actor:**  Dealer

**Pre-Conditions:**

* Connected to server
* Logged into a dealer account
* Actively playing a game
* Player gets blackjack
* Dealer bust

**Post-Conditions:**

* Player receives the bet amount or 3:2 if the player gets blackjack.

**Main Flow:**

* Game is being played
* Player gets blackjack or dealer bust
* Dealer pays Player proper amount.

**Alternative Flow:**  None

**Exceptions:** Server is offline

**Related Use Cases:** Play Game

### Shuffle

**Relevant Requirements:** Login

**Actor:**  Dealer

**Pre-Conditions:**

* Connected to server
* Logged into a dealer account
* Actively playing a game
* Reached threshold

**Post-Conditions:**

* Shoe is shuffled

**Main Flow:**

* Game is being played
* Reach threshold to shuffle cards

**Alternative Flow:**  None

**Exceptions:** Server is offline

**Related Use Cases:** Play Game

### Midgame Leave

**Relevant Requirements:** Login

**Actor:**  Dealer

**Pre-Conditions:** Logged into a dealer account, actively dealing a game

**Post-Conditions:**

* The active table immediately closes, without penalty to any of the players

**Main Flow:**

* Dealer Login -> Dealer Create Table -> Dealer Start Game -> Dealer Leave Table -> Confirmation Window

**Alternative Flow:**  None

**Exceptions:**

**Related Use Cases:** Play Game

## Server Activities

### Startup

**Relevant Requirements:** ….

**Actor:**  Server

**Pre-Conditions:** Port and ip must be open

**Post-Conditions:**  Clients should now be able to launch and connect to the server

**Main Flow:**

**Alternative Flow:**  None

**Exceptions:**

* Server socket can’t open, maybe it's already running?

**Related Use Cases:** N/A

### Handle Connections

**Relevant Requirements:** ….

**Actor:**  Server

**Pre-Conditions:** Server must be started

**Post-Conditions:**  Client should be handled

**Main Flow:**

* Client starts up, and connects to the server socket
* Server must keep track of this connection in separate thread and continue servicing it for the duration of it’s connection

**Alternative Flow:**  None

**Exceptions:**

* Client doesn’t connect properly

**Related Use Cases:** Startup

### 

### 

### Handle Client Requests

**Relevant Requirements:** ….

**Actor:**  Server

**Pre-Conditions:**

* Client must be connected
* Client must have sent a request

**Post-Conditions:**

* Server must have handled the request and responded with a status
* Server must be ready for more requests

**Main Flow:**

**Alternative Flow:**  None

**Exceptions:**

* Request is not recognized as a valid signature

**Related Use Cases:** Handle Connections

### Message Types

#### Login Dealer

**Relevant Requirements:** ….

**Actor:**  Server/Client/Dealer

**Pre-Conditions:**

* Client must be connected
* Client must have sent a request

**Post-Conditions:**

* Server must have handled the request and responded with a status
* Server must be ready for more requests

**Main Flow:** Open Client -> Login -> Submit

**Alternative Flow:**  Open Client -> Login -> Login

**Exceptions:**

* Request is not recognized as a valid signature

**Related Use Cases:** Handle Connections, Handle Client Requests

**Notes:**

* The Login Dealer request is sent to the server by the client
* The Login Dealer response is sent to the client by the server when receiving the request

#### Create Account Player

**Relevant Requirements:** ….

**Actor:**  Server/Client, Player

**Pre-Conditions:**

* Client must be connected
* Client must click “create account” button

**Post-Conditions:**

* Server must have handled the request and responded with a status
* The account details should be added to the player login file
* Server must be ready for more requests

**Main Flow:** Open Client -> Create Account -> Submit -> Login

**Alternative Flow:**  None

**Exceptions:**

* Credentials already exist in player login file

**Related Use Cases:** Handle Connections, Handle Client Requests

**Notes:**

* The Create Account Player request is sent to the server by the dealer client
* The Create Account Player response is sent back to the client by the server when receiving the request

### Join Table

**Relevant Requirements:** ….

**Actor:**  Server/Client

**Pre-Conditions:**

* Client must be logged
* Client must be trying to join a table and specify which tables

**Post-Conditions:**

* Server must have handled the request and responded with a status
* Server must be ready for more requests

**Main Flow:**

**Alternative Flow:**  None

**Exceptions:**

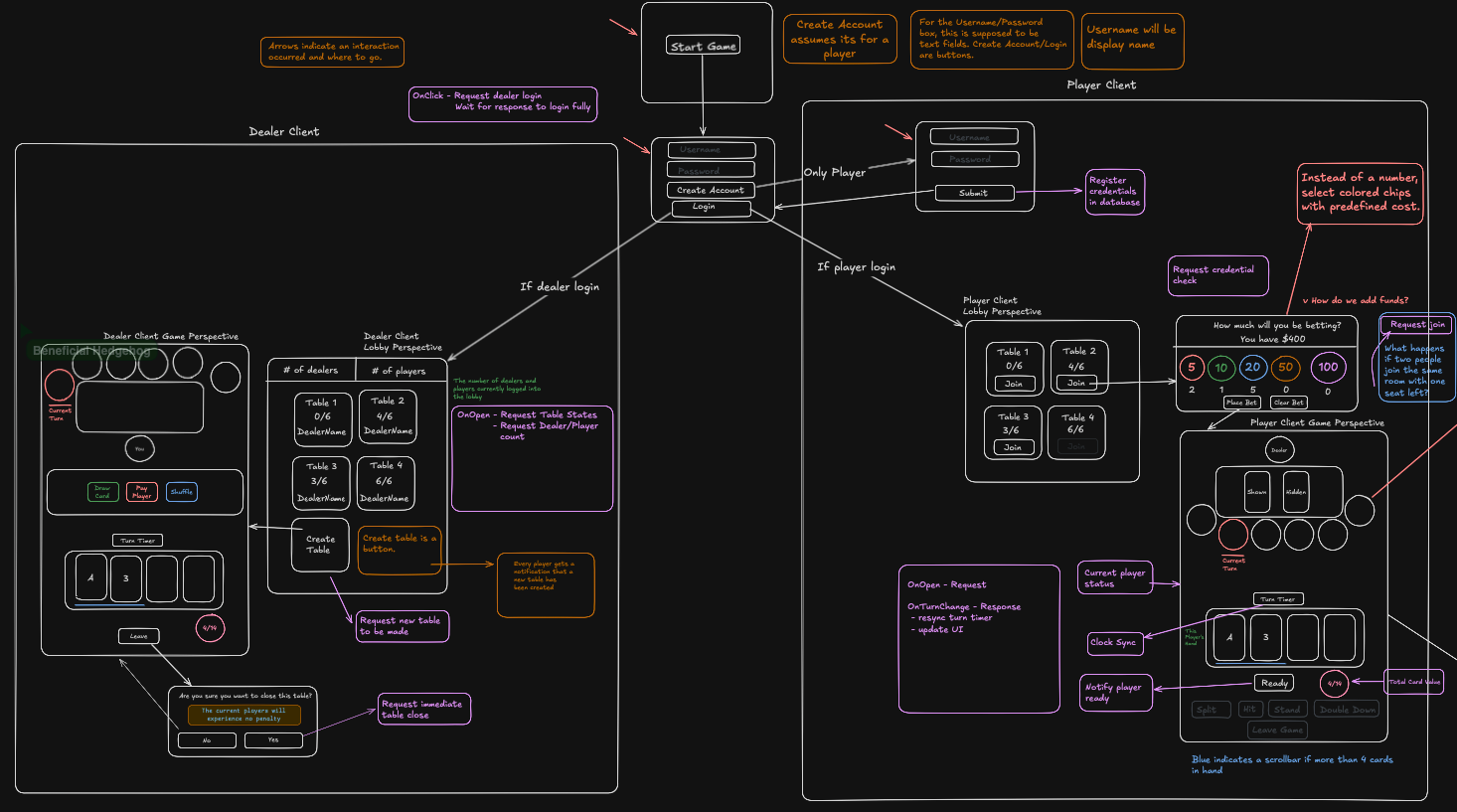
* Request is not recognized as a valid signature

**Related Use Cases:** Handle Connections, Handle Client Requests

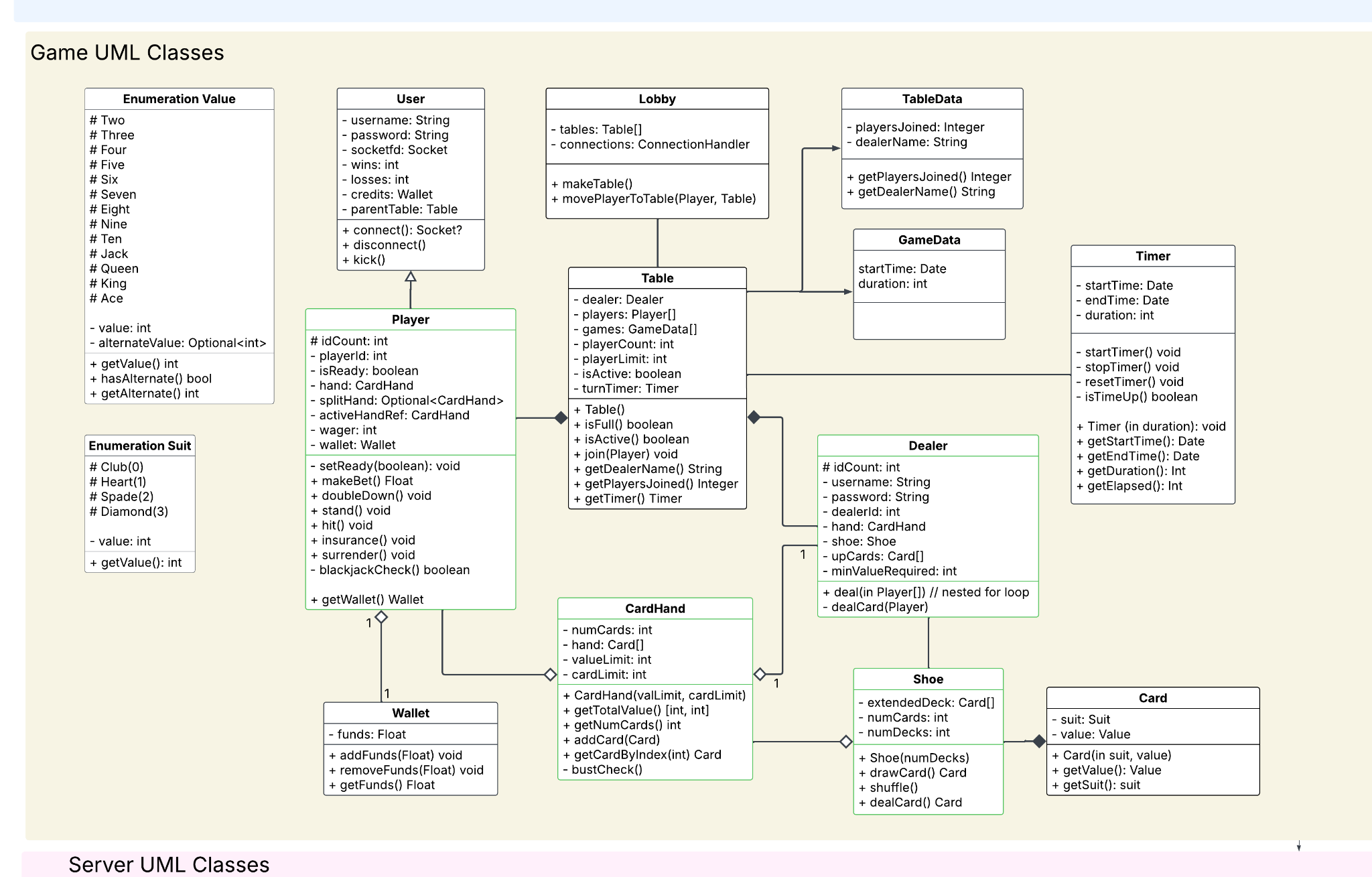
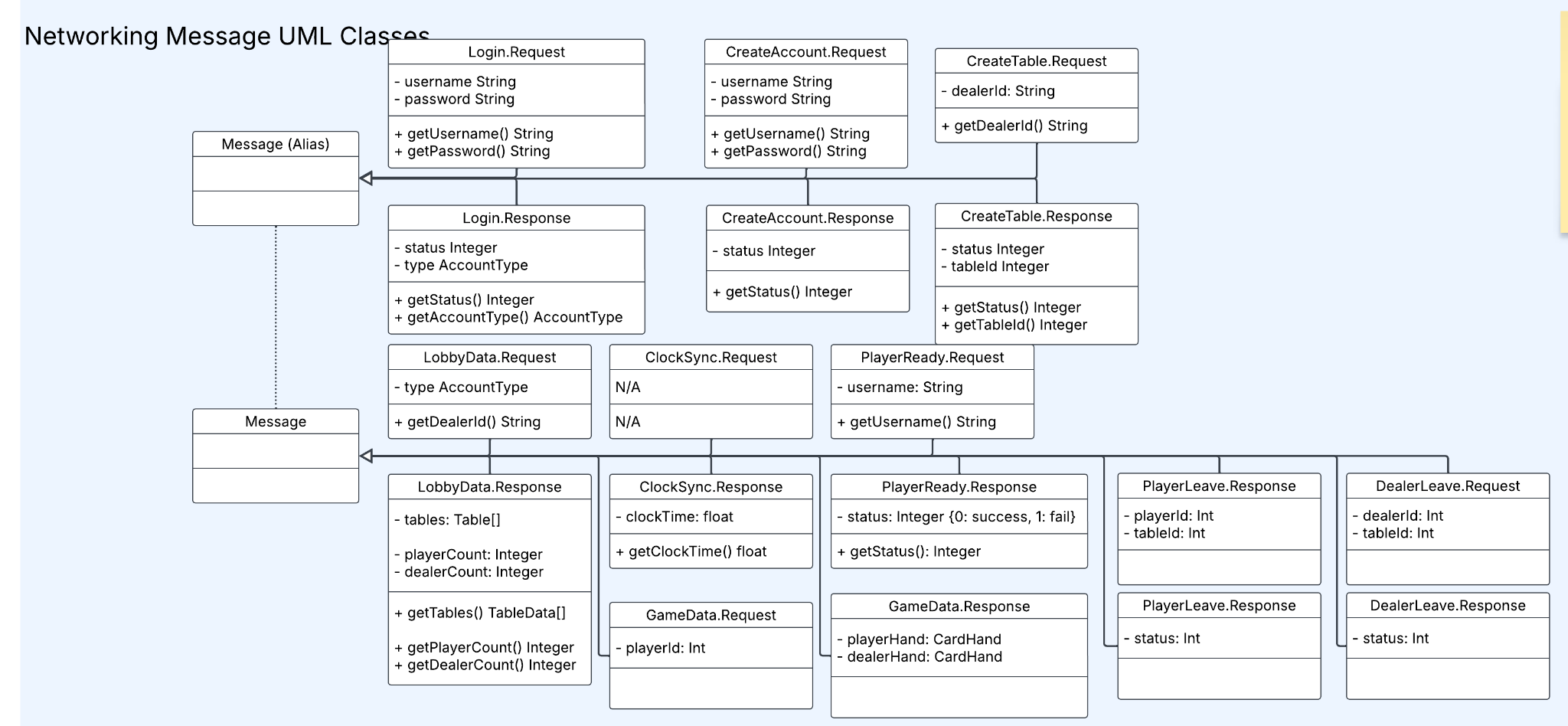
#### 

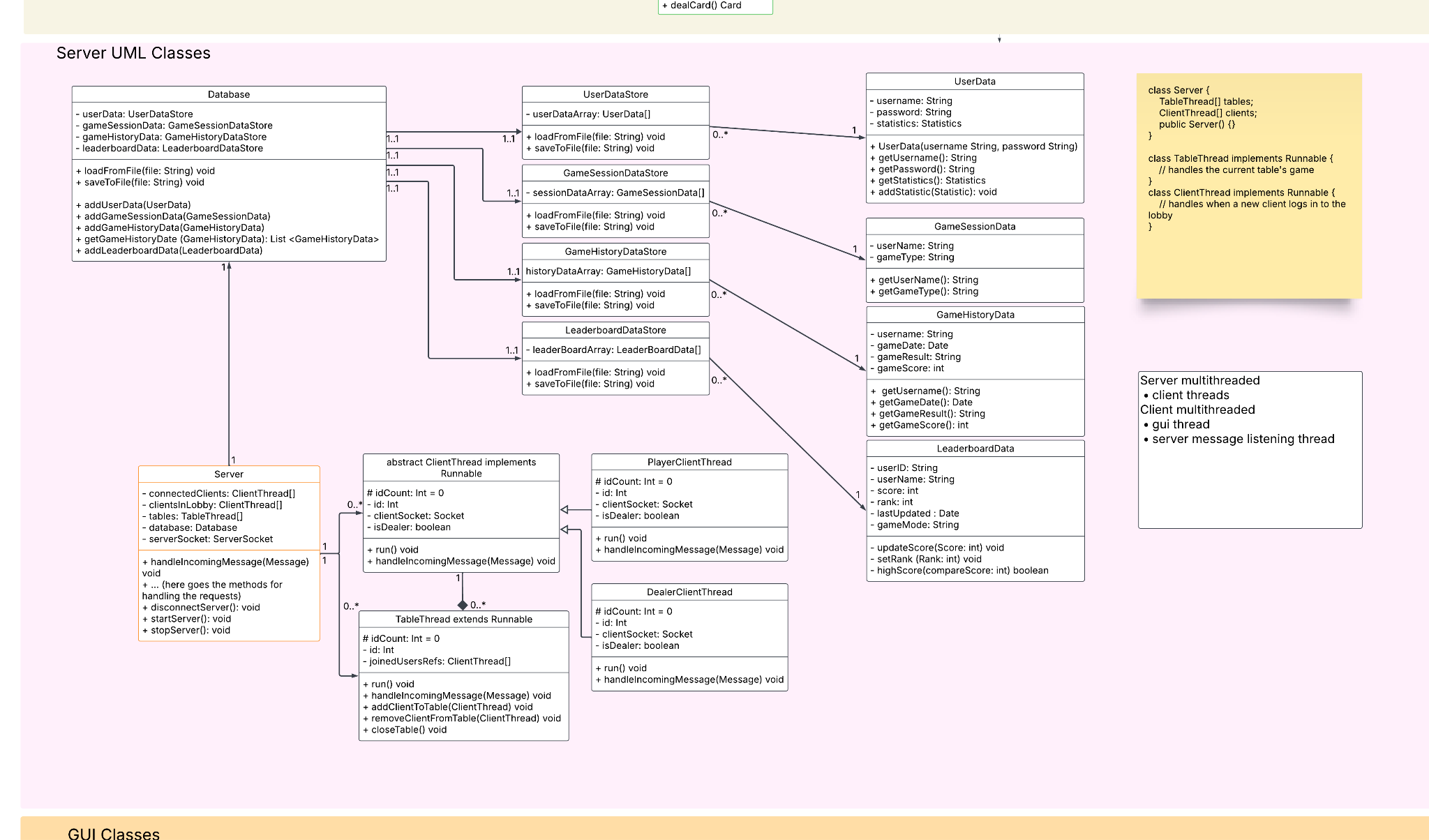
# 

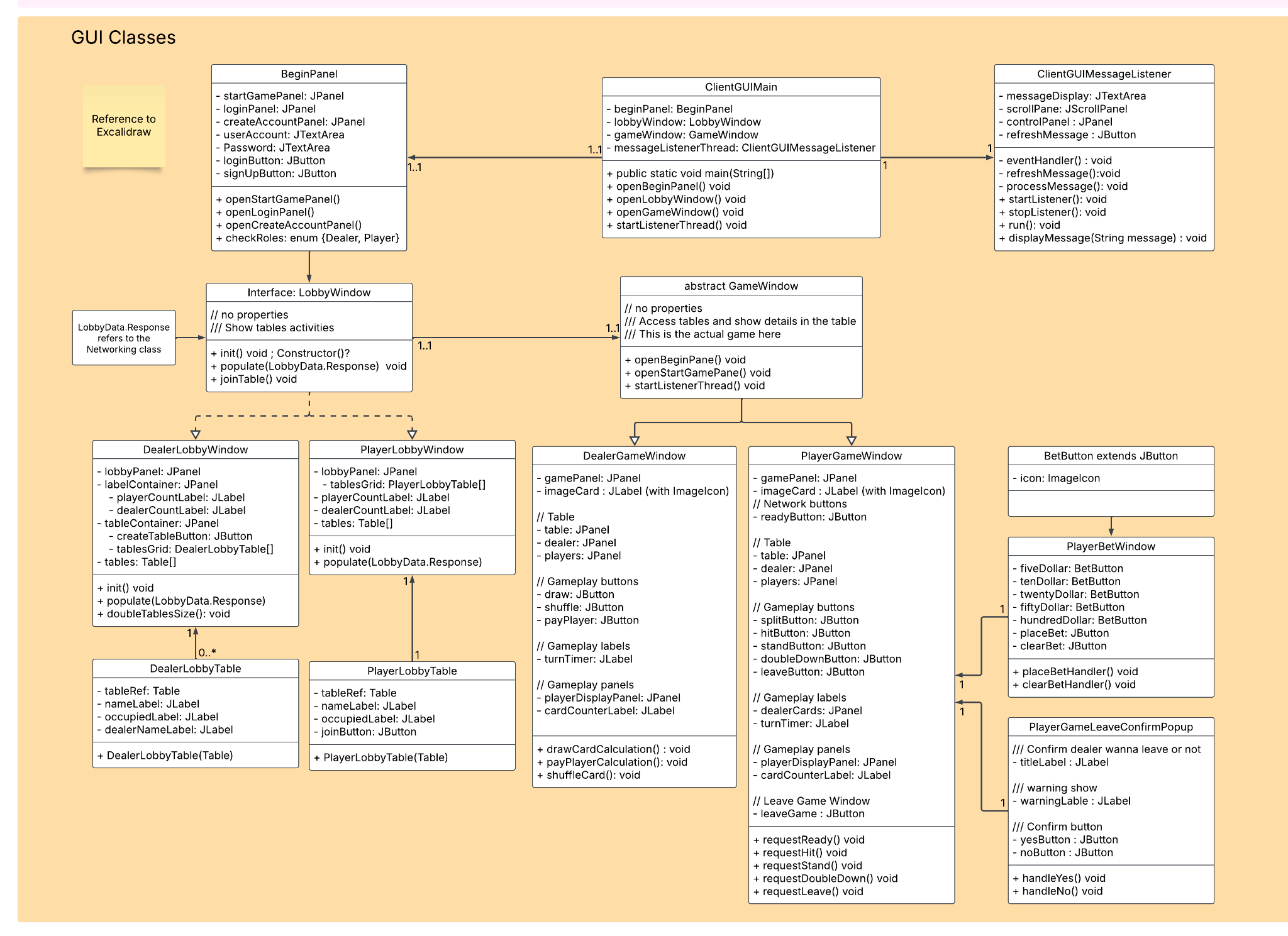
# GUI Prototypes



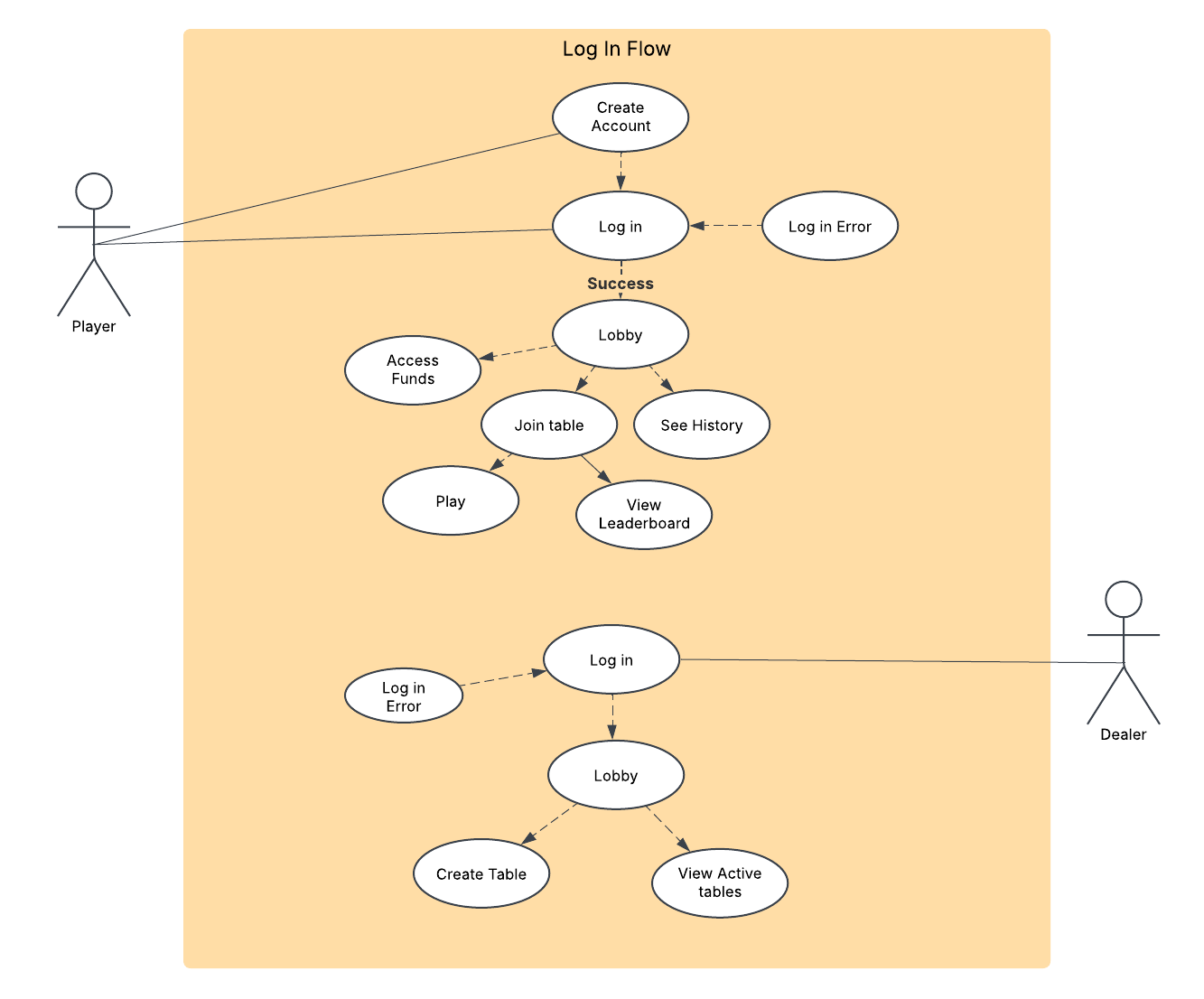
# UML Class Diagrams

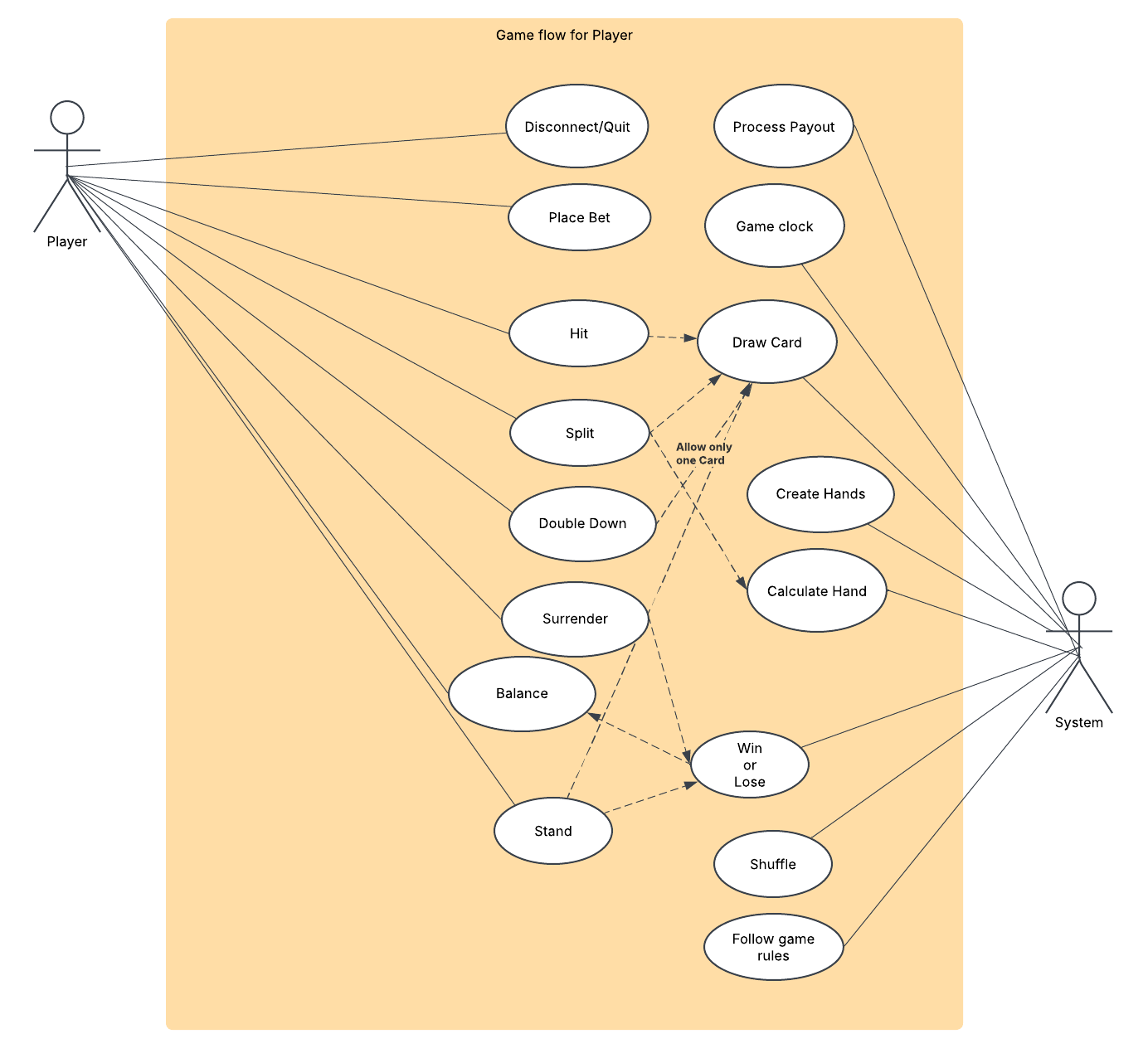


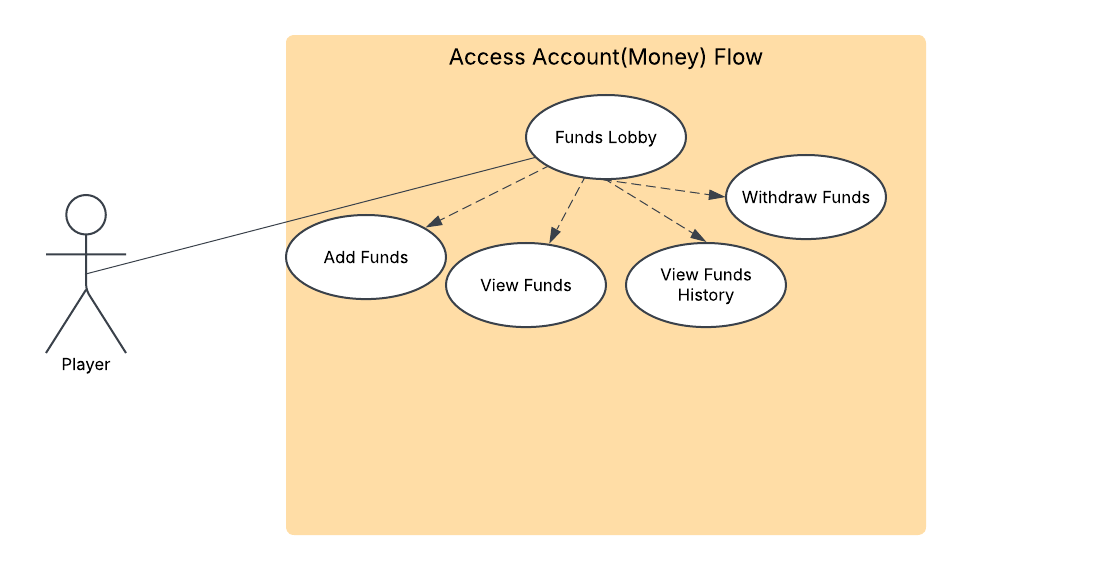
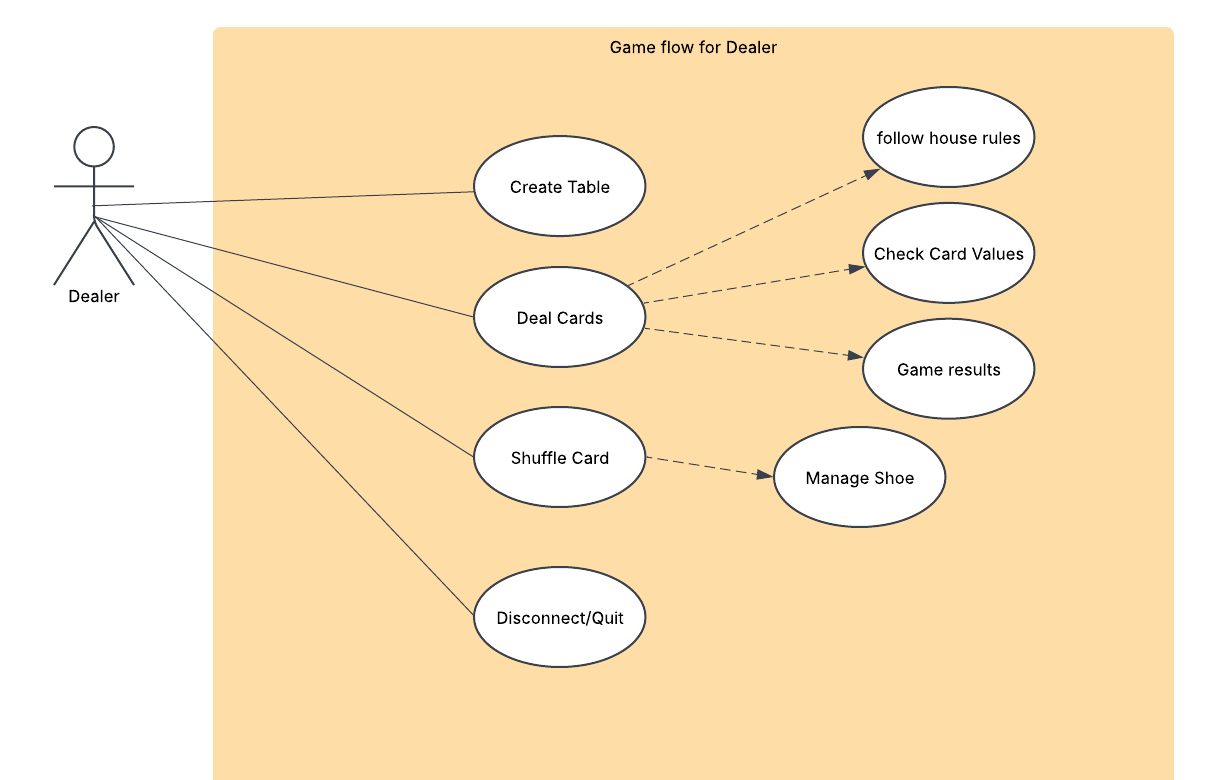




# UML Use Case Diagrams

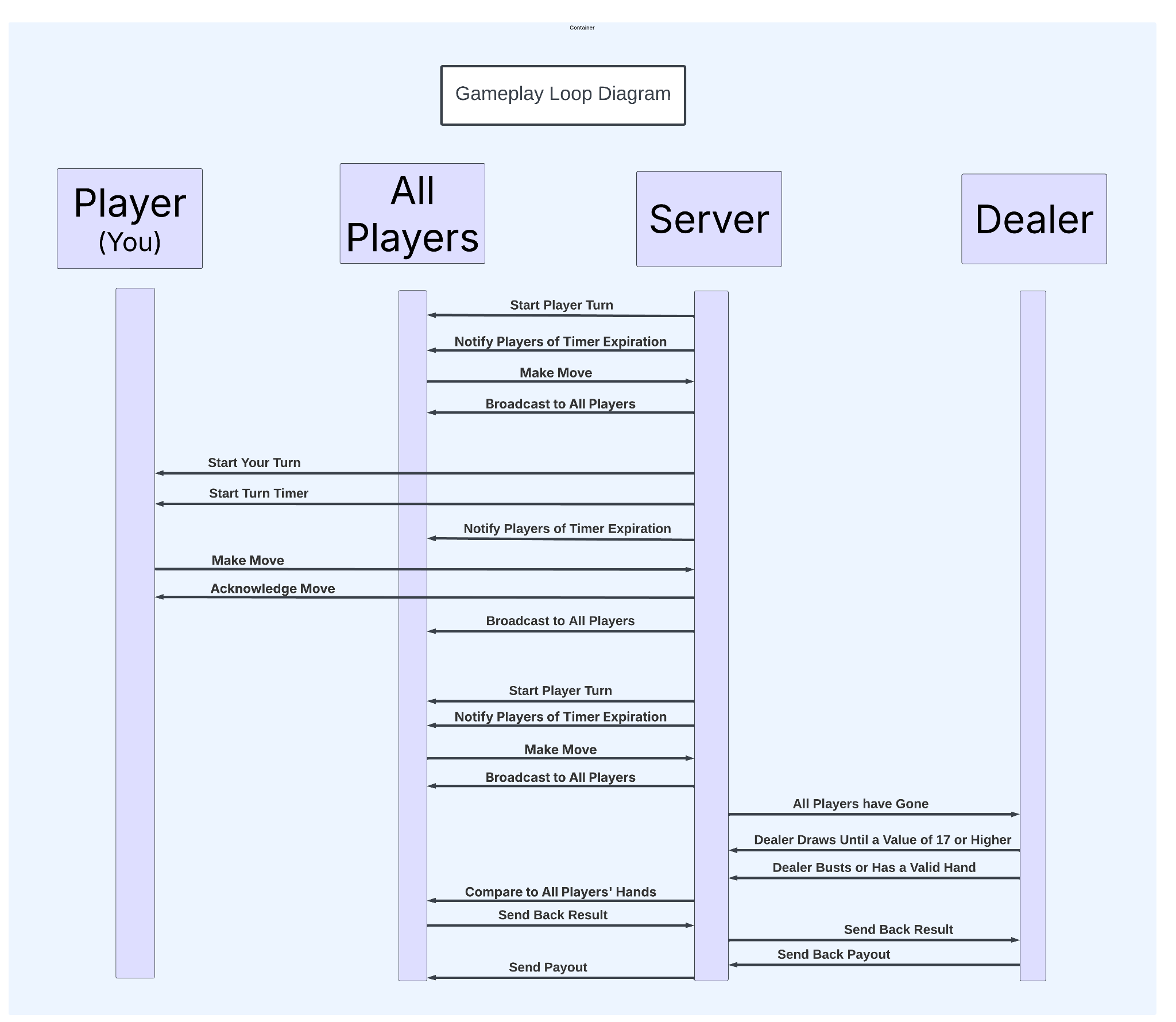


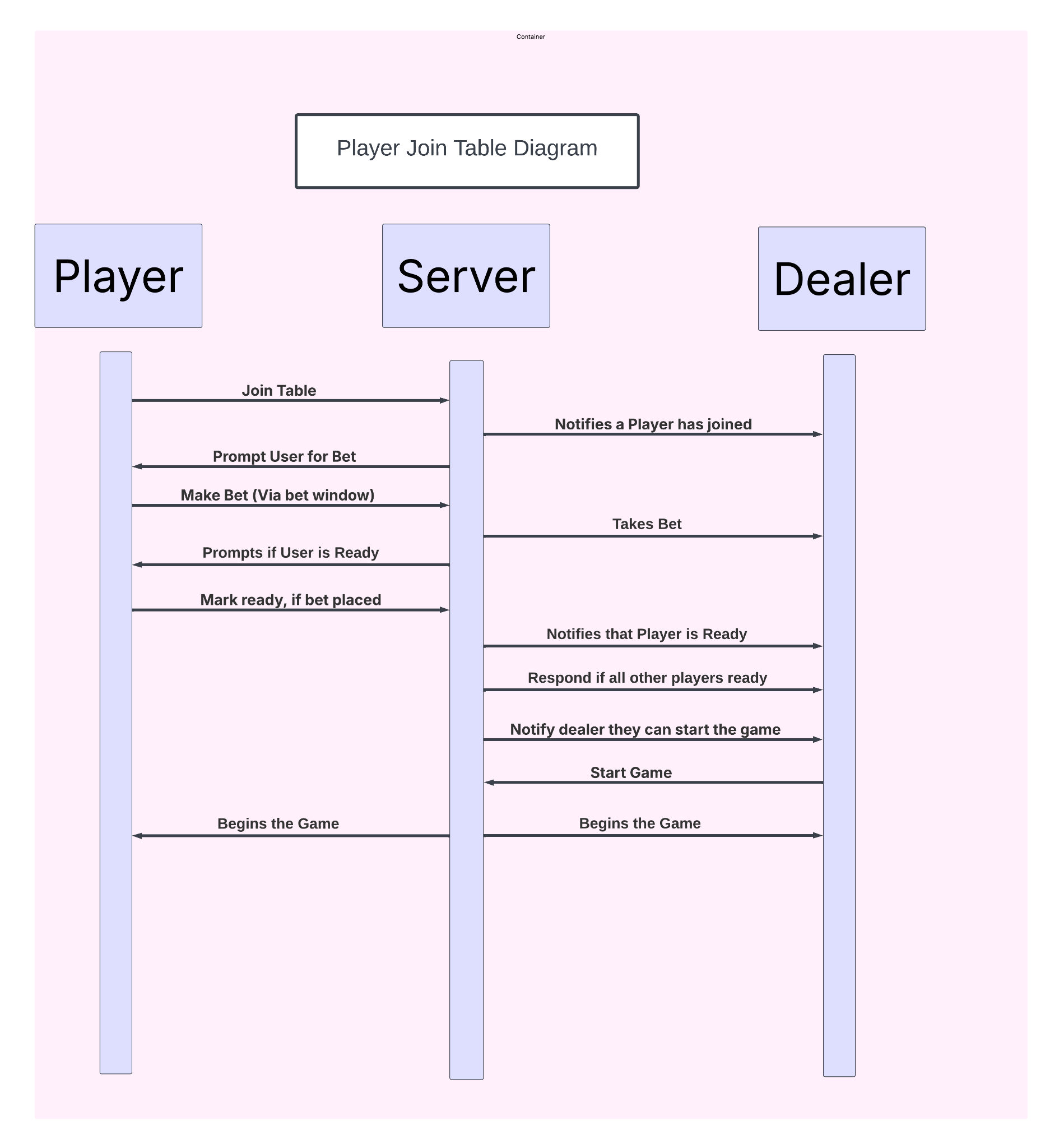


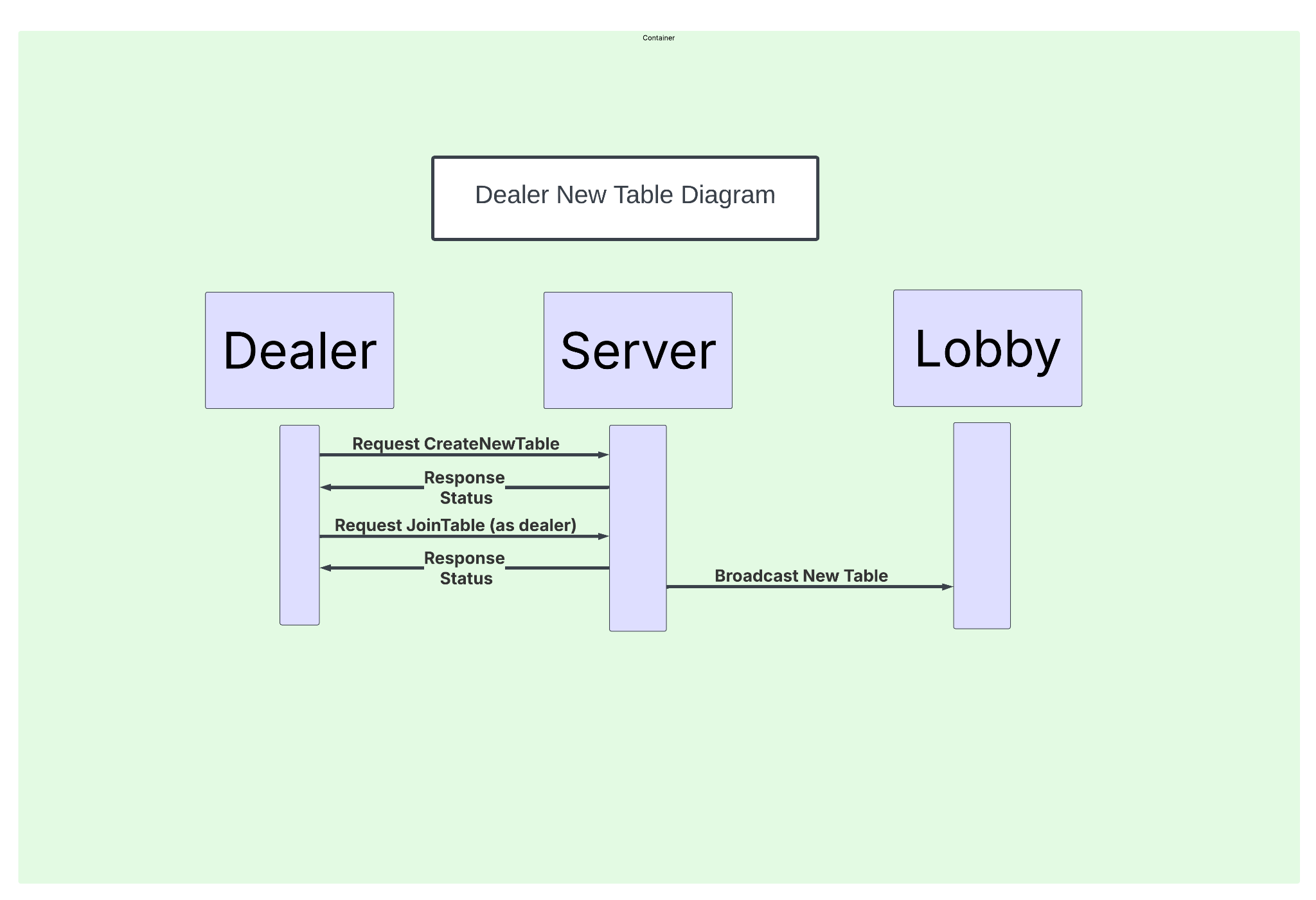


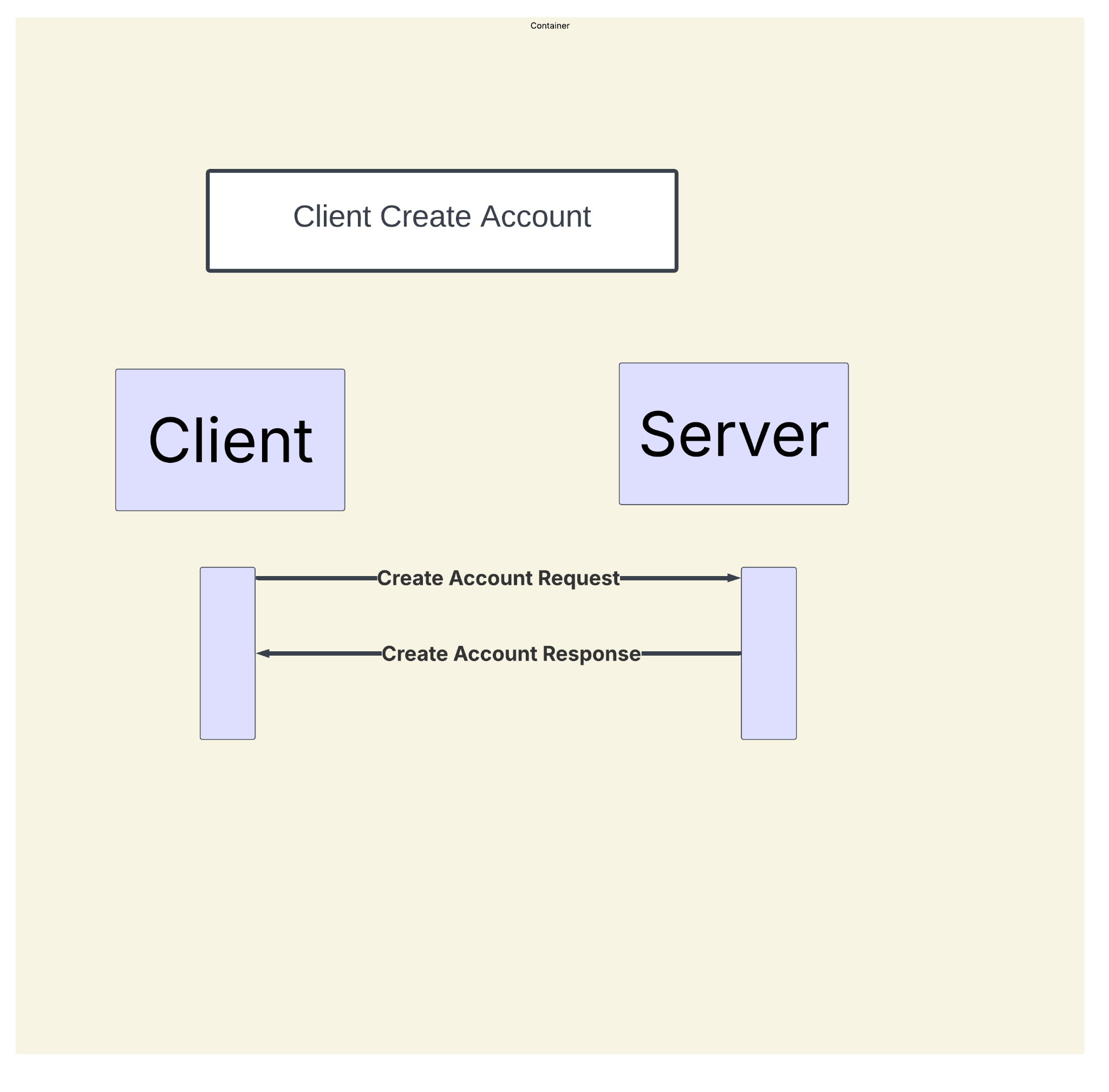
# UML Sequence Diagrams

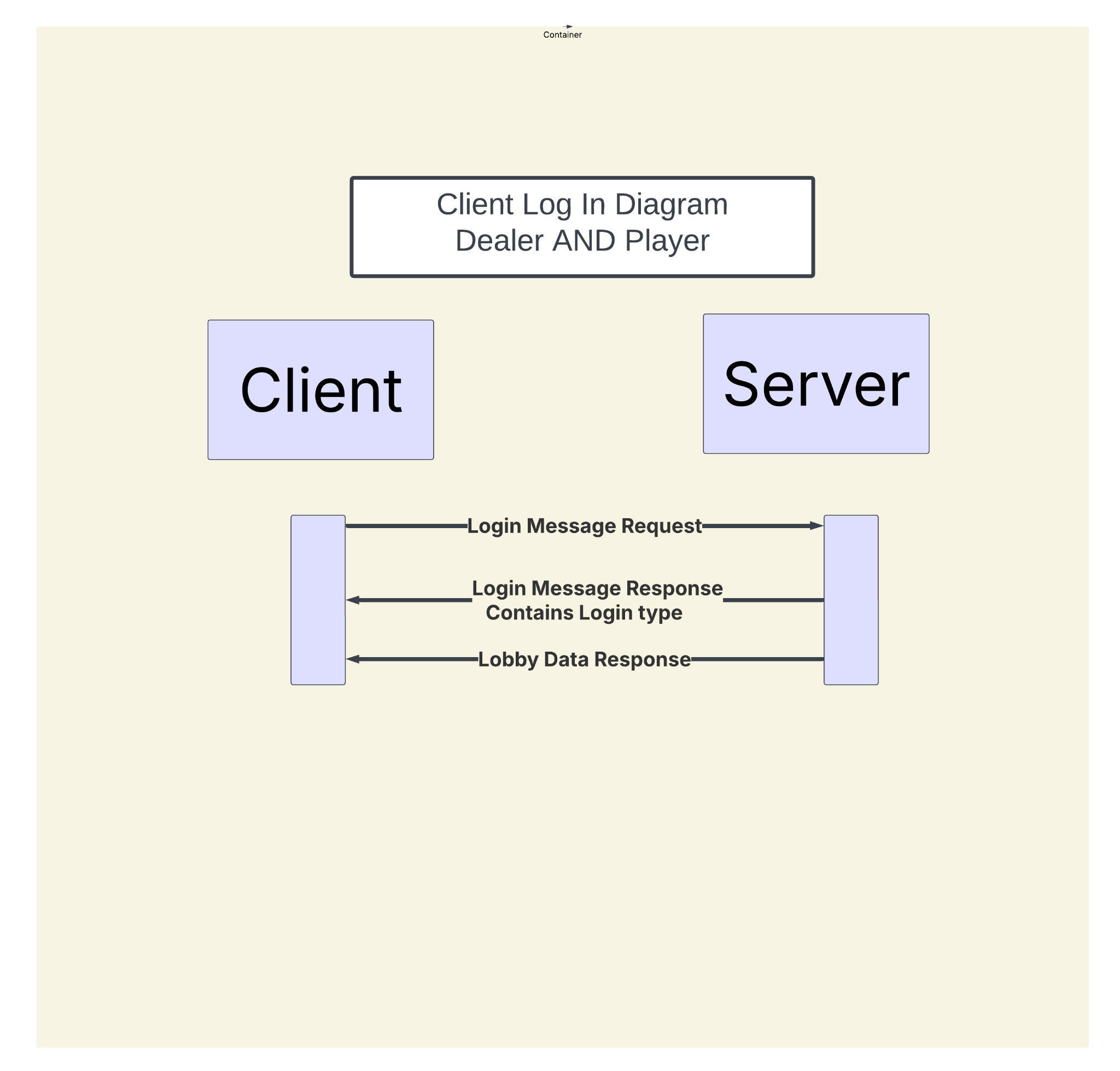
# 



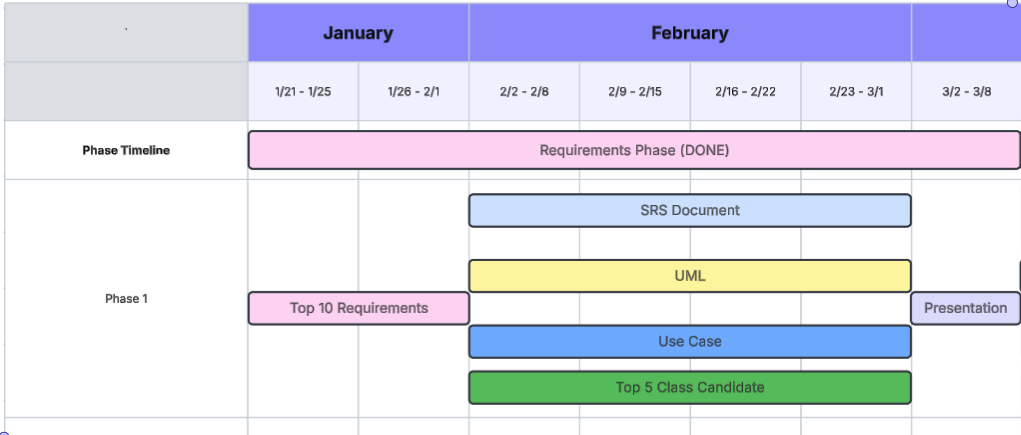


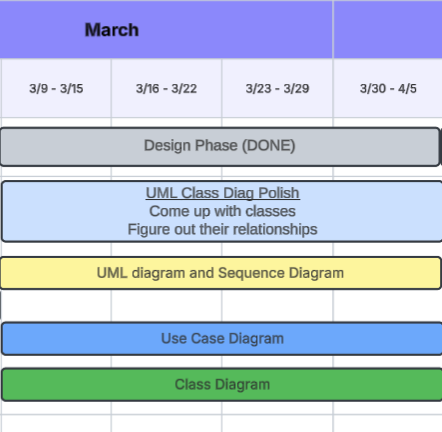
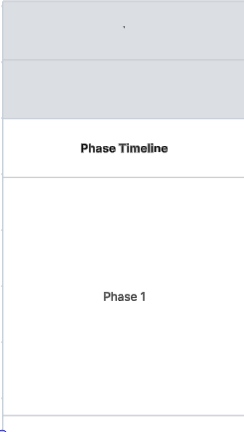


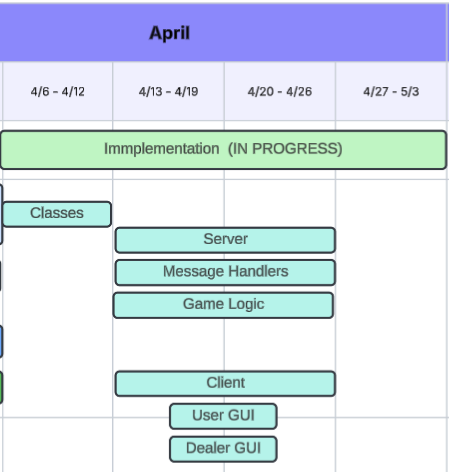
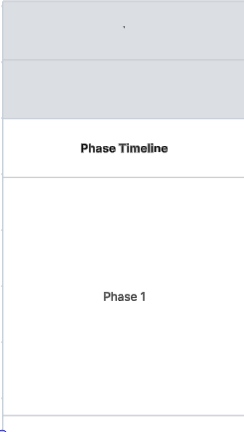




# Timeline







# GitHub Repo Link

<https://github.com/CSUEB-Black-Jack-Group-5/CS401-BlackJack.git>