Software Requirements Specification

Revision History

Date	Revision	Description	Author
09/22/2025	1.0	Purpose	Rahul Suthar
9/23/2025	2.0	Started on Overall Description	Blake Messer
9/23/2025	2.1	Started Specific requirements	Rahul Suthar
9/23/2025	3.0	External requirements	Samuel Millet
9/24/2025	3.1	Internal requirements	Samuel Millet
9/24/2025	4.0	Started Non-Functional Requirements (security)	Blake Messer
9/24/2025	4.1	Non-Functional Requirements (Environmental)	Blake Messer
9.25/2025	4.2	Non-Functional Requirements (Performance)	Rahul Suthar
9/29/2025	5.0	Started Use Case Specifications (1-3)	Rahul Suthar
9/29/2025	5.1	Use Case Specifications (3-4)	Blake Messer
9/29/2025	5.2	Use Case Specifications(5)	Samuel Millet
9/29/2025	5.3	Use Case Specifications(6)	Rahul Suthar
9/29/2025	6.0	UML Case Diagrams (1-2)	Rahul Suthar
9/30/2025	6.1	UML Case Diagrams (3-5)	Blake Messer
9/30/2025	6.2	UML Case Diagrams (6)	Samuel Millet
10/1/2025	7.0	Class Diagram (1-2)	Samuel Millet
10/1/2025	7.1	Class Diagram (3-4)	Rahul Suthar
10/1/2025	7.2	Class Diagram (5-6)	Blake Messer
10/1/2025	8.0	Sequence Diagram (1-3)	Samuel Millet
10/1/2025	8.1	Sequence Diagram (4)	Rahul Suthar
10/1/2025	8.2	Sequence Diagram (5-6)	Blake Messer
10/10/2025	9.0	Worked on Card, Client, GUI, Server Classes	All
10/12/2025	9.1	Worked on Player, Dealer, Deck Classes	All
10/16/2025	9.2	Worked on Menu, Client, Message Classes	All
10/17/2025	9.3	Login Manager, Game Table, Hand, Shoe Classes	All
10/24/2025	9.4	Worked on Account, Bet Classes	All

Table of Contents

Purpose

This document outlines the requirements for a Blackjack Application

Scope

This document will catalog the user, system, and hardware requirements for the Multiplayer Blackjack Gaming System.
It will define what the system must accomplish but will not detail how these requirements will be implemented.

Definitions, Acronyms, Abbreviations

1.2.1 Hand: The cards a player has during a round
1.2.2 Table: A game room consisting of 1-7 players
1.2.3 Dealer: The person who deals cards to active players
1.2.4 BlackJack: A hand consisting of 2 cards that add up to 21
1.2.5 Table Limit: The minimum and maximum number of people at a single table
1.2.6 Hit: When a player requests an additional card from the dealer.
1.2.7 Stand: When a player decides not to take any more cards and ends their turn.
1.2.8 Bust: When a player's hand value exceeds 21, resulting in an automatic loss.
1.2.9 Card: A single playing card used in the game, with a rank and suit.
1.2.10 Suit: One of the four categories of cards (Hearts, Diamonds, Clubs, Spades).
1.2.11 Numbered Card: Of a card, numbered 2-10
1.2.12 Face Card: Of a card, of rank Jack, Queen, King
1.2.13 Ace card: A special card valued at either 1 or 11, depending on the hand.
1.2.14 Deck: A standard set of 52 playing cards, consisting of 13 ranks in 4 suits.
1.2.15 Push: A tie between the player and dealer where the bet is returned
1.2.16 Split: When a player has two cards of the same rank and separates them into two hands.
1.2.17 Double Down: When a player doubles their bet and receives only one additional card.
1.2.18 TCP/IP: Transmission Control Protocol/Internet Protocol, a set of networking rules enabling communication over the internet.
1.2.19 GUI: Graphical User Interface, a visual way for users to interact with the system.
1.2.20 RNG: Random Number Generator

- ## References

1.3.1 Use Case Specification Document: Refer to Use Case Specification.docx
1.3.2 Github Repository: https://github.com/Rahul4301/Blackjack/
1.3.3 UML Use Case Diagrams Document
1.3.4 Class Diagrams Document
1.3.5 Sequence Diagrams Document

- ## Overview

The multiplayer blackjack system is a Java-based application featuring a graphical user interface (GUI).
It allows players to join virtual tables, place bets, and play according to standard blackjack rules.
The system runs over TCP/IP to provide real-time gameplay, secure fund transactions, fair play, and protection against cheating.

Overall Description

Product Perspective

The multiplayer blackjack system is a client-server application. The server is the main authority, managing all game state, player accounts, and logic.

Product Architecture

2.2.1 **Client Communication Module**: Manages network connections and messaging with all client applications

2.2.2 **Game Logic Module:** Enforces the rules of blackjack, manages game state for all active tables, and handles dealer's actions.

2.2.3 **Player and Session Management Module:** Handles user authentication, account balances, and active player sessions.

Product Functionality/Features

2.3.1 User account login and management of current session.
2.3.2 Player account management; like viewing account balances, adding or removing funds.
2.3.3 Concurrent multiplayer tables.
2.3.4 A game lobby that shows current game tables and if they are open/available.
2.3.5 Game rules are enforced.
2.3.6 There are real-time updates to all players at a table.

Constraints

2.4.1 Minimum JDK Version

2.4.2 No databases, frameworks, or external libraries unless explicitly approved

2.4.3 Client must provide a graphical user interface (GUI)

Assumptions and Dependencies

2.5.1 Assume that players understand the rules and how to play blackjack.

2.5.2 Assume that system depends on a stale network connection.

2.5.3 Assume that players have a stable internet connection.

2.5.4 Players contain enough funds to engage in the game.

2.5.5 Assume that each game table works within the proper player limit.

2.5.6 Assume that each player has a unique username and unique password for their account.

2.5.7 Each player should make a decision in a certain amount of time or else they will be kicked or booted from the game.

2.5.8 The game should start once at least one player minimum joins the game table.

2.5.9 Once confirmed any bet made is final and cannot be rescinded.

● Specific Requirements

○ Functional Requirements

●.○.1. Common Requirements:

3.1.1.1 Users should be allowed to create an account with a unique username and password, between 5-20 characters

3.1.1.2 Users should be able to log out of the system at any time.

3.1.1.3 Users should be allowed to register for a new account with a unique username and password.

3.1.1.4 Users should be able to recover or reset their password in case of loss.

3.1.1.5 The system shall enforce account states (New, Active, Blocked, Banned) for user access control.

●.○.2. Payouts and Betting Module Requirements:

3.1.2.1 The system must allow players to wager money from their account funds when they are in an active game.

3.1.2.2 The system must add game winnings from each game to the winning player's funds.

3.1.2.3 The system must deduct funds from player accounts when placing a bet if sufficient funds are available

3.1.2.4 Players must place a minimum bet of $1 per hand.

3.1.2.5 Players must place bets using integer values only; decimal values are not allowed.

3.1.2.6 The system must prevent bets exceeding the player's available balance.

3.1.2.7 The system shall log all bets and payouts for audit and reporting.

3.1.2.8 1:1 Payout odds, 3:2 payout for a blackjack (21)

●.○.3. Game Logic Module:

3.1.3.1 The system will ensure fair play by utilizing true shuffle algorithms for the decks of cards

3.1.3.2 The system will check win and lose conditions after every turn, recognizing if either is met.

3.1.3.3 The system will be able to handle checking game states for multiple tables simultaneously

3.1.3.4 The system must track and update the game's current state during transitions between betting, card dealing, player turns, and round end.

3.1.3.5 A time limit of 30 seconds will be implemented for each player's turn to ensure smooth gameplay.

3.1.3.6 If a player fails to act within the time limit, the system will automatically stand for them.

3.1.3.7 The system must compare the dealer's final hand against each player's hand to determine winners.

3.1.3.8 The system will remove a table from the list of available tables once there is no dealer present.

3.1.3.9 The system must handle game restarts and prepare the table for a new round.

●.○.4. Network Module Requirements:

3.1.1.1 The system shall allow users to connect to the game server securely.

3.1.1.2 The system shall update game data in real time for all players at a table.

3.1.1.3 The system shall allow players to disconnect at any time; dealers must wait until the end of the current game before disconnecting.

3.1.1.4 If a player loses connection, they will be dropped from the game and all their current bets will be forfeited.

3.1.1.5 The system shall handle reconnection attempts and restore sessions if possible.

●.○.5. Player Module Requirements

3.1.5.1 Players are allowed to deposit and withdraw money from their account.

3.1.5.2 Player winnings shall be credited automatically, and withdrawals may be requested at any time.

3.1.5.3 The player account will include session management and current balance.

3.1.5.4 Players can only join a table if it does not exceed the maximum of 6 players and if they have sufficient funds.

3.1.5.5 Players must wait in the lobby until invited to join a table by the dealer.

3.1.5.6 Players must be able to view their game history, including past bets and results.

3.1.5.7 The system must enforce one active session per player account.

●.○.6. Dealer Module Requirements

3.1.6.1 The dealer must play only after all players have completed their actions.

3.1.6.2 The dealer must hand out two cards to each player and themselves initially.

3.1.6.3 Dealers may not leave a table while there is an active game in progress.

3.1.6.4 Dealers must manage player seating and invitations from the lobby.

●.○.7. Table Module Requirements

3.1.7.1 In order for a game to start, there must be at least one player and exactly one dealer, with a maximum of seven players.

3.1.7.2 A table will remain active as long as there is exactly one dealer present.

3.1.7.3 The system shall ensure that no player is seated at more than one table simultaneously.

3.1.7.4 The system must provide unique identifiers for each table to distinguish active sessions.

○ External Interface Requirements

3.2.1. The system must provide a user-friendly graphical interface for both players and dealers.

3.2.2. The GUI must display real-time game updates, including player actions, card draws, bets, payouts, results, and actions like standing, hitting, busting, splitting, and doubling down.

3.2.3. Separate dashboards must be provided for players and dealers.

3.2.4. The GUI must be visually engaging and immersive to enhance the player's experience.

3.2.5. Animations for shuffling, dealing cards, and moving chips must be included to improve immersion.

3.2.6. Players must be able to place bets, hit, stand, double down, split, or leave the game.

3.2.7. Before joining a game, the system must present lobbies showing available tables.

○ Internal Interface Requirements

1.3.1. The system must authenticate users by verifying their username and password during login.

1.3.2 If a user times out, quits, or disconnects, all in-game interactions must end immediately.

1.3.3. The system must track and display all game status changes in real time, including hand results for both players and dealers.

1.3.4. The internal interface must support all valid player actions: hit, stand, double down, split, and leave the game.

1.3.5. A Random Number Generator must be used to ensure fair shuffling of the deck.

1.3.6. If a player lacks sufficient funds, the system must reject their bet before the game begins.

1.3.7. The deck must simulate a "shoe" of three decks combined, with reshuffling triggered halfway through the game.

1.3.8. The GUI must receive and display data from the game engine in real time.

• Non-Functional Requirements

○ Security and Privacy Requirements

4.1.1 **Anti Cheating Measures:** The system will include anti-cheating measures to prevent malicious intent by using two decks of cards for card counters and reshuffle the deck halfway.
4.1.2 **Secure Authentication:** The player has to log into the system using a valid and unique username and password.
4.1.3 **Controls based on Role:** Players of certain roles will have access to limited functions.

○ Environmental Requirements

4.2.1 System must be developed using the Java programming language that must be compatible with all platforms including Windows, Linux, and MacOS.
4.2.2 System must have consistent UI design from all screens from the blackjack lobby, game table, and balance inquiry screens.
4.2.3 System must operate over TCP/IP, to ensure that real-time gameplay is achieved with more than one player on multiple different devices over the same network.
4.2.4 System needs to have a stable network connection to ensure real-time gameplay engagement.

○ Performance Requirements

4.3.1 **Table Players:** System must be able to support up to 6 players at the same time for each game table while also maintaining real-time gameplay to ensure the engagement is continued.
4.3.2 **Lobby Players:** System must be able to support a lobby that consists of multiple tables of 6 coexisting players happening at the same time without any performance complications.
4.3.3 **Disconnecting/Inactive Players:** In an event where a player loses connection to the system, quits the game, or their connection is timed out due to being inactive, then the game will remove them from the table to ensure a consistent experience without any interruption.

● Use Case Specification

Use Case 1: Player Registration and Login

Use Case ID: UC01
Use Case Name: Player Registration and Login
Primary Actor: Player, Server

Preconditions:

- Server is running
- Player has the client application open

Flow:

1. Player opens the client application
2. Client displays login screen via GUI
3. Player selects "Register" option
4. Player enters username, password, and initial deposit amount
5. Client creates a Message object with type "REGISTER" and sends to Server
6. Server validates username is unique and creates new Player account
7. Server sends confirmation Message back to Client
8. Client displays success message and returns to login screen
9. Player enters username and password
10. Client sends Message with type "LOGIN" to Server
11. Server authenticates credentials
12. Server sends Player object back to Client
13. Client stores currentUser and displays lobby screen via GUI

Player Can:

- Register new account (username, password)
- Log into existing account
- View profile and check balance

Postconditions:

- Player account is created and saved on Server
- Player is logged in and viewing the lobby

Extensions / Alternate Flow:

- Username already exists → system requests another.
- Invalid credentials → deny login.

Use Case 2: Joining a Table and Placing a Bet

Use Case ID: UC02
Use Case Name: Joining a Table and Placing a Bet
Primary Actor: Player, Dealer, Server

Preconditions:

- Player is logged in
- At least one GameTable exists with available seats
- Dealer is managing a table

Flow:

1. Player views available tables in lobby (Client requests table list from Server)
2. Server sends list of GameTable objects to Client
3. GUI displays tables with dealer names and available seats
4. Player selects a table and clicks "Join"
5. Client sends Message with type "JOIN_TABLE" and tableID to Server
6. Server adds Player to the GameTable's player list
7. Server updates GameTable.state and broadcasts a game snapshot via GAME_UPDATE enum
8. Client receives GameState and GUI displays the game table
9. Dealer starts a new round (sends Message with type "DEALER_ACTION", action "START_ROUND")
10. Server updates GameTable gameState to "BETTING"
11. GUI prompts Player to place a bet
12. Player enters bet amount and clicks "Place Bet"
13. Client sends Message with type "PLAYER_ACTION", action "PLACE_BET", amount
14. Server validates bet against Player balance and table limits
15. Server updates Player's currentBet and deducts from balance
16. Server broadcasts updated GameState to all players at table
17. Client receives GameState and GUI displays updated balance and bet

Player Can:

- Browse open tables and view players
- Join table with available seats
- Place initial bet within table limits
- Adjust bet amount or clear bet entirely
- Leave joined table at any time

Postconditions:

- Player is seated at the table
- Player's bet is placed and balance is updated
- All players at table can see the new player and their bet

Use Case 3: Playing a Hand (Hit and Stand)

Use Case ID: UC03
Use Case Name: Playing a Hand
Primary Actor: Player, Dealer, Server

Preconditions:

- Player is seated at a table with a bet placed
- Dealer has dealt initial cards
- It is the Player's turn

Flow:

1. Dealer deals initial cards (2 cards to each player, 2 to dealer with one hidden)
2. Server updates GameState with all hands and sets currentTurn to first player
3. Client receives GameState and GUI displays cards and available actions (HIT, STAND, DOUBLE_DOWN, SPLIT)
4. Player clicks "Hit" button
5. Client sends Message with type "PLAYER_ACTION", action "HIT" to Server
6. Server processes hit: deals one card from Deck to Player's Hand
7. Server checks if Player busts (hand value > 21)
8. Server updates GameState with new hand and broadcasts to all clients
9. Client receives GameState and GUI updates to show new card
10. Player's hand value is 19, Player clicks "Stand"
11. Client sends Message with type "PLAYER_ACTION", action "STAND" to Server
12. Server marks Player as finished and moves to next player's turn
13. After all players finish, Dealer plays their turn automatically
14. Server processes dealer turn: reveals hidden card, hits until 17 or higher
15. Server evaluates all hands and determines winners
16. Server updates Player balances based on wins/losses
17. Server creates GameState with results and broadcasts to all clients
18. Client receives GameState and GUI displays results and updated balance

Player Can:

- Hit (request another card)
- Stand (keep current hand)
- Double Down (double bet for one card)
- View cards and hand value during gameplay

Postconditions:

- Round is complete
- Player balance is updated based on win/loss
- Table is ready for next round

Use Case 4: Session Management and Exit

Use Case ID: UC04
Use Case Name: Session Management and Exit
Primary Actor: Player, Server

Preconditions:

- Player is seated at a table
- No active round is in progress (or Player has finished their turn)

Flow:

1. Player clicks "Leave Table" button in GUI
2. Client sends Message with type "LEAVE_TABLE" to Server
3. Server removes Player from GameTable's player list
4. Server saves updated Player balance to their account
5. Server broadcasts updated GameState to remaining players at table
6. Server sends confirmation Message to Client
7. Client receives confirmation and GUI displays lobby screen
8. Player views their updated balance in the lobby
9. Player clicks "Logout" button
10. Client sends Message with type "LOGOUT" to Server
11. Server sets Player's isOnline status to false
12. Server saves all account data
13. Client disconnects from Server
14. GUI displays login screen

Player Can:

- Leave table between rounds
- Cash out chips and return to the main lobby
- Log out of account
- Exit application with balance automatically saved

Postconditions:

- Player is removed from the table
- Player balance is saved on Server
- Remaining players see updated table with Player gone
- Player is logged out and can close the application

Use Case 5: Player Profile Management

Use Case ID: UC05
Use Case Name: Player Profile Management
Primary Actor: Player, Server

Preconditions:

- Player is logged into the system
- Player is in the main lobby

Flow:

1. Player clicks "View Profile" button in lobby
2. Client sends Message with type "REQUEST_PROFILE" to Server
3. Server retrieves Player's account statistics and game history
4. Server sends Account object containing balance, win/loss record, and game history to Client
5. Client receives Account data and GUI displays profile screen
6. Player views their statistics including:
    a. Current chip balance
    b. Total games played
    c. Wins/Losses record
    d. Recent game history
7. Player clicks "Back to Lobby" when finished
8. Client returns to lobby screen

Player Can:

- View current chip balance and account details
- Check win/loss statistics and game history
- Monitor performance trends and achievements
- Return to main lobby at any time

Postconditions:

- Player has viewed their profile information
- Player returns to main lobby screen
- All profile remains unchanged

Use Case 6: Advanced Game Actions

Use Case ID: UC06
Use Case Name: Advanced Game Actions
Primary Actor: Player, Dealer, Server

Preconditions:

- Player is seated at a table with a bet placed
- Player has been dealt initial two cards
- It is the Player's turn

Flow:

1. Player receives two initial cards of the same value (e.g., two 8s)
2. GUI displays "Split" button as available action
3. Player clicks "Split" button
4. Client sends Message with type "PLAYER_ACTION" with action "SPLIT" to Server
5. Server splits the hand into two separate hands
6. Server places matching bet for second hand
7. Server deals additional card to each split hand
8. Player plays first split hand to completion
9. Player plays second split hand to completion
10. If dealer shows Ace, GUI displays "Insurance" option
11. Player can choose to take insurance bet
12. Client sends appropriate "PLAYER_ACTION" messages for all decisions

Player Can:

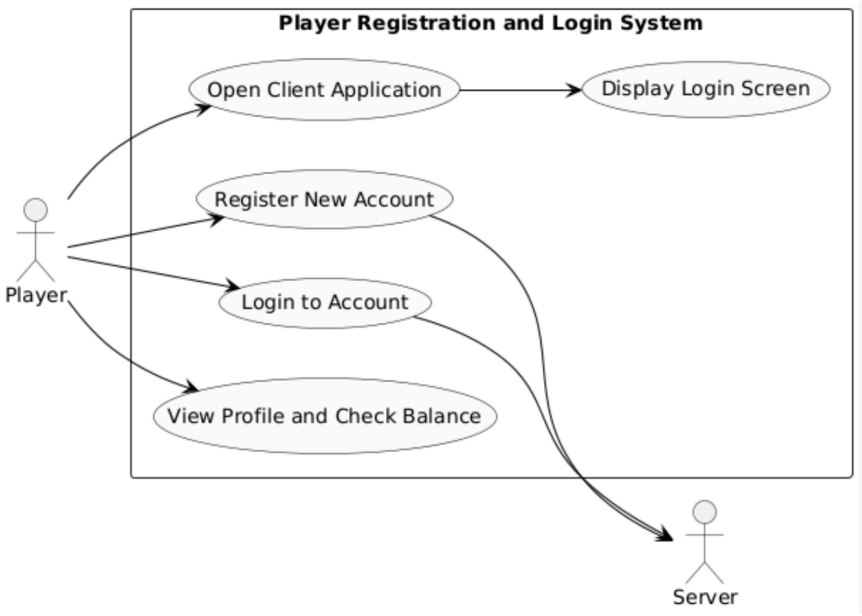
- Split pairs when dealt two cards of same value
- Take insurance when dealer shows Ace
- Play each split hand independently with standard actions (HIT, STAND, DOUBLE_DOWN)
- Manage multiple simultaneous bets during split hands

Postconditions:

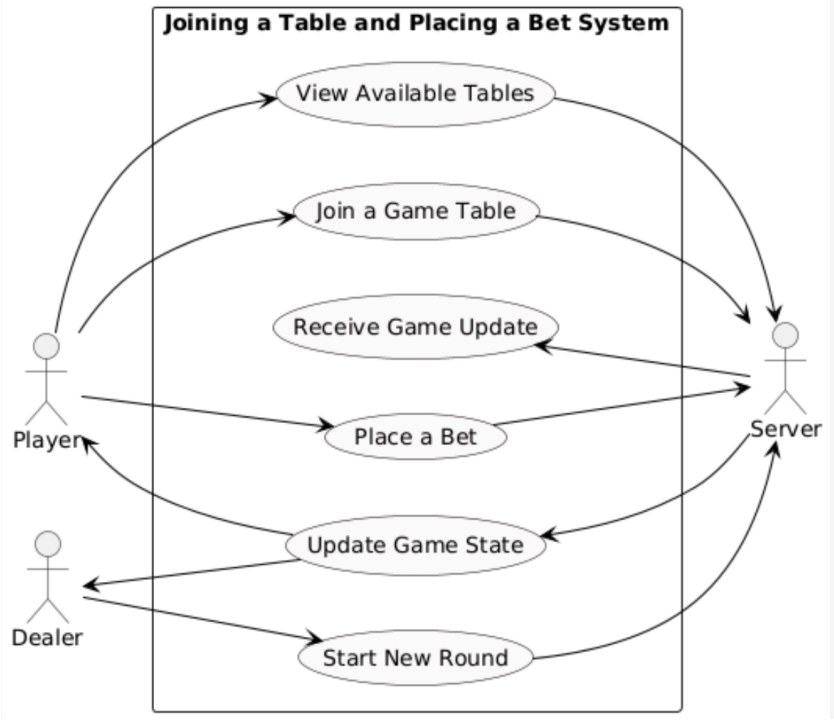
- Split hands are resolved independently
- Player's balance is updated based on outcomes of all hands
- Table proceeds to next player or dealer turn

• UML Use Case Diagram

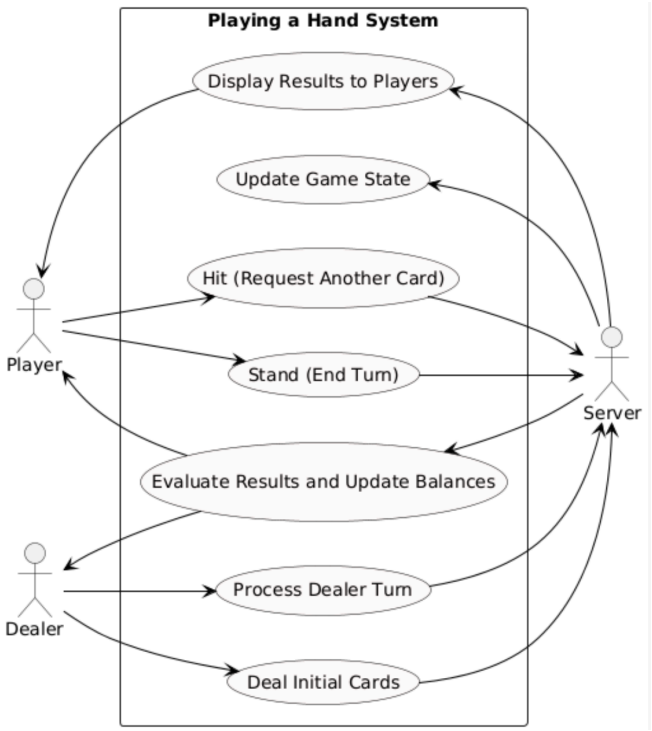
Use Case 1: Login and registration



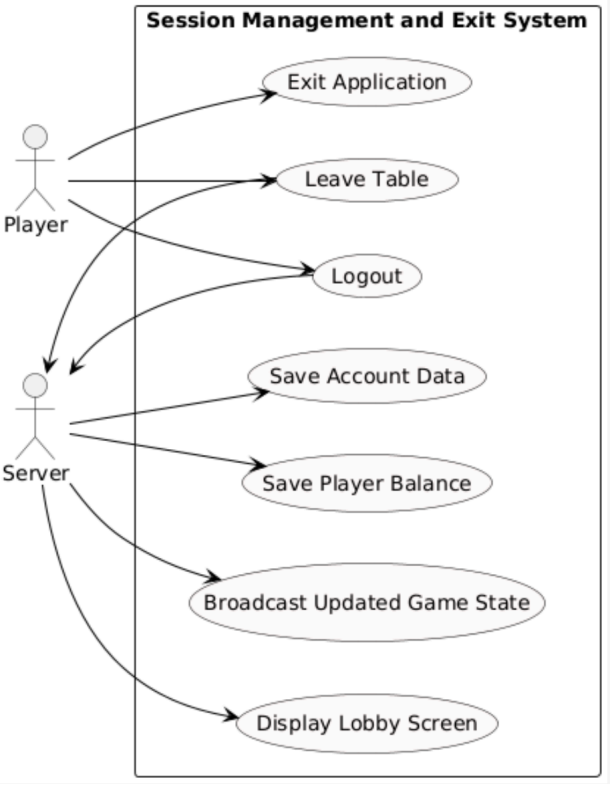
Use Case 2: Joining a table and placing a bet



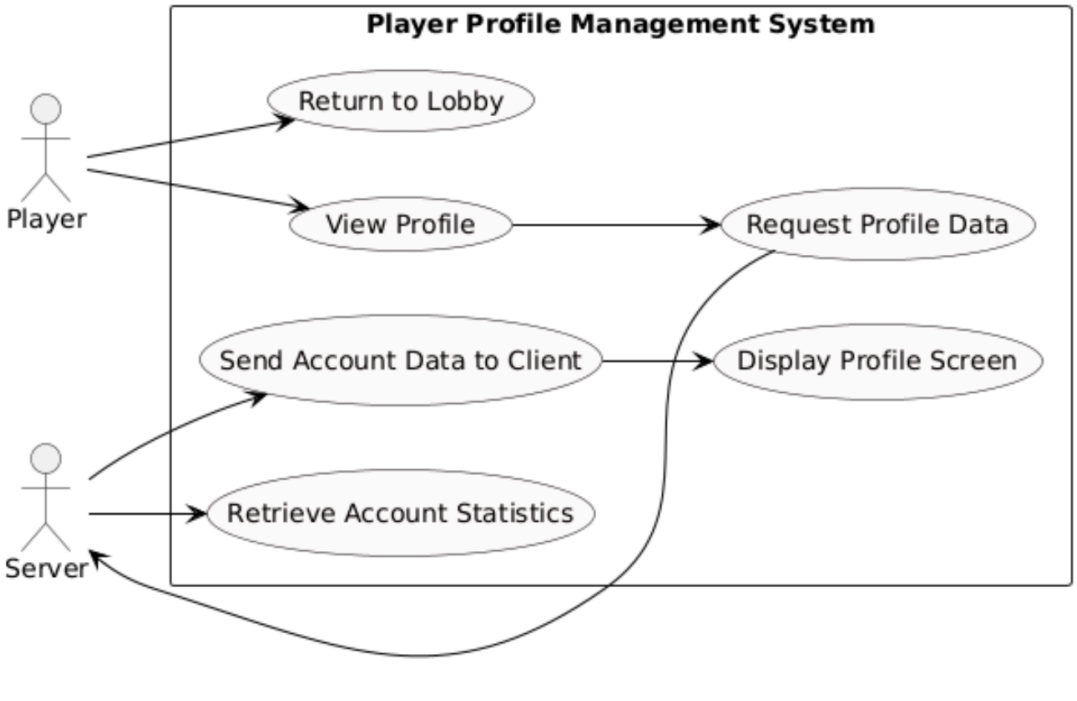
Use Case 3: Playing a hand



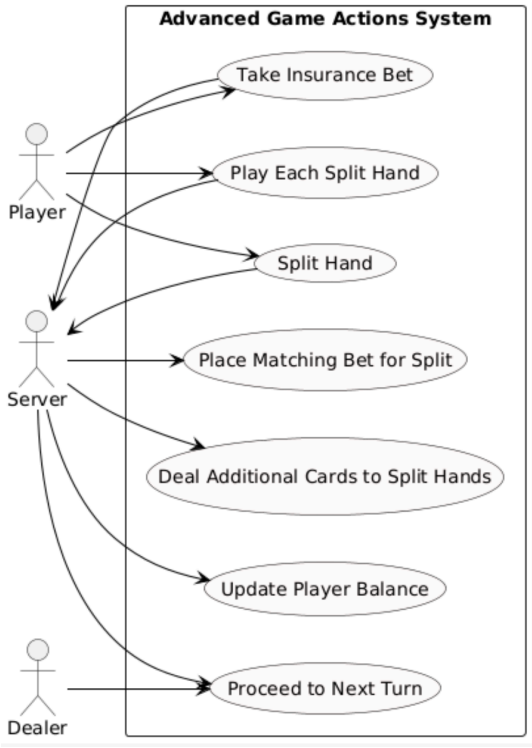
Use Case 4: Session Management



Use Case 5: Profile Management



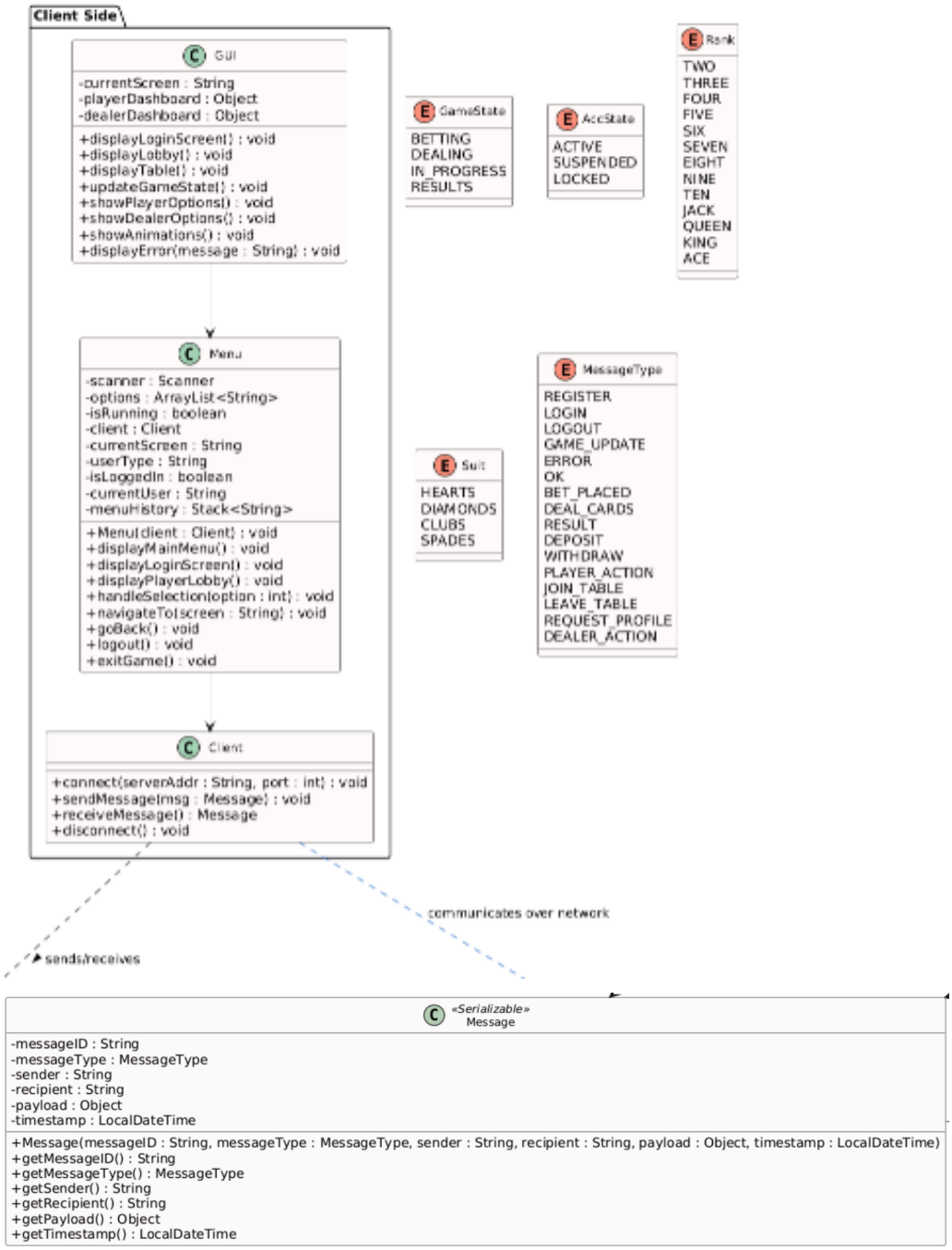
Use Case 6: Advanced Game Actions



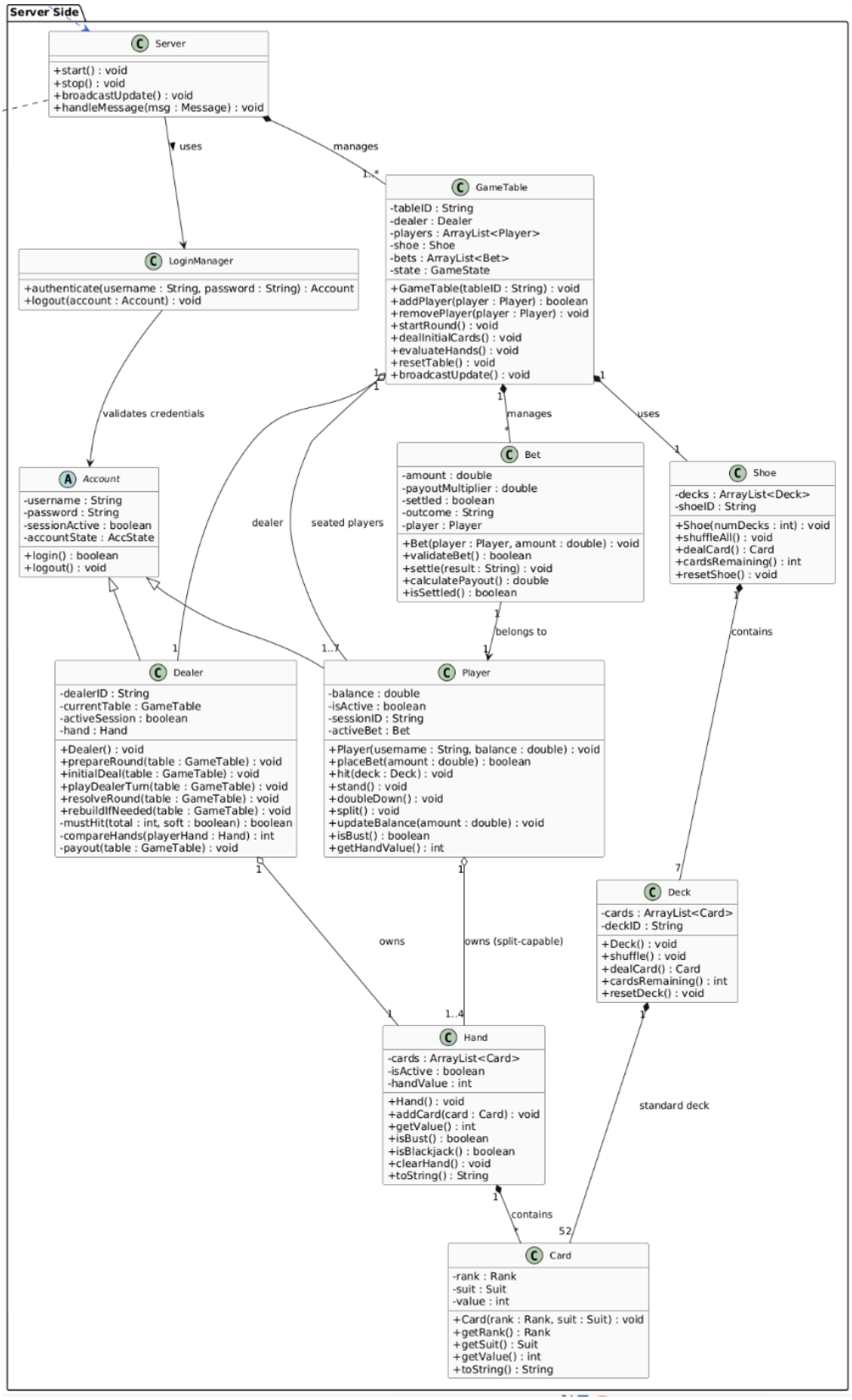
• Class Diagrams

The Client side sends requests to the Server anytime a player wants to do anything. The Client and Server will communicate through message passing.

CLIENT SIDE:

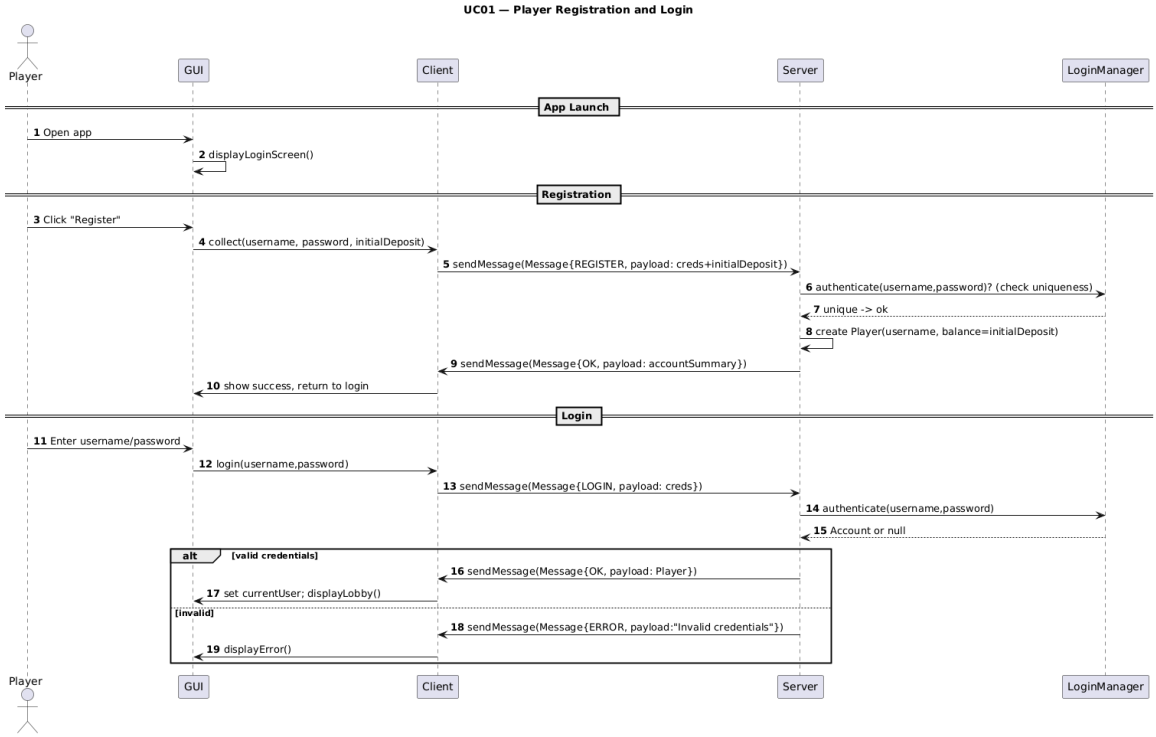


SERVER SIDE

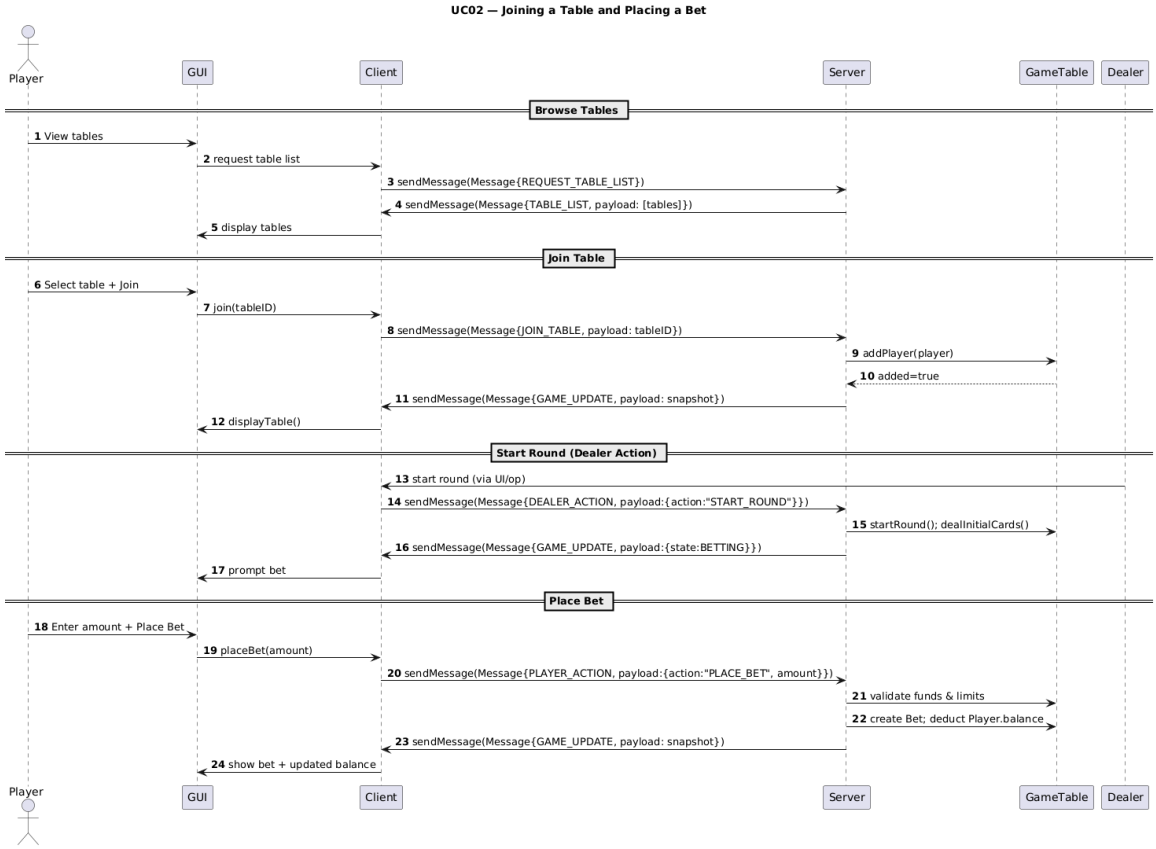

Server Side
Server
+start() : void
+stop() : void
+broadcastUpdate() : void
+handleMessage(msg : Message) : void
uses
manages
1..*
GameTable
-tableID : String
-dealer : Dealer
-players : ArrayList<Player>
-shoe : Shoe
-bets : ArrayList<Bet>
-state : GameState
+GameTable(tableID : String) : void
+addPlayer(player : Player) : boolean
+removePlayer(player : Player) : void
+startRound() : void
+dealInitialCards() : void
+evaluateHands() : void
+resetTable() : void
+broadcastUpdate() : void
LoginManager
+authenticate(username : String, password : String) : Account
+logout(account : Account) : void
validates credentials
manages
uses
Account
-username : String
-password : String
-sessionActive : boolean
-accountState : AccState
+login() : boolean
+logout() : void
dealer
seated players
Bet
-amount : double
-payoutMultiplier : double
-settled : boolean
-outcome : String
-player : Player
+Bet(player : Player, amount : double) : void
+validateBet() : boolean
+settle(result : String) : void
+calculatePayout() : double
+isSettled() : boolean
Shoe
-decks : ArrayList<Deck>
-shoeID : String
+Shoe(numDecks : int) : void
+shuffleAll() : void
+dealCard() : Card
+cardsRemaining() : int
+resetShoe() : void
belongs to
contains
Dealer
-dealerID : String
-currentTable : GameTable
-activeSession : boolean
-hand : Hand
+Dealer() : void
+prepareRound(table : GameTable) : void
+initialDeal(table : GameTable) : void
+playDealerTurn(table : GameTable) : void
+resolveRound(table : GameTable) : void
+rebuildIfNeeded(table : GameTable) : void
-mustHit(total : int, soft : boolean) : boolean
-compareHands(playerHand : Hand) : int
-payout(table : GameTable) : void
Player
-balance : double
-isActive : boolean
-sessionID : String
-activeBet : Bet
+Player(username : String, balance : double) : void
+placeBet(amount : double) : boolean
+hit(deck : Deck) : void
+stand() : void
+doubleDown() : void
+split() : void
+updateBalance(amount : double) : void
+isBust() : boolean
+getHandValue() : int
owns
owns (split-capable)
Deck
-cards : ArrayList<Card>
-deckID : String
+Deck() : void
+shuffle() : void
+dealCard() : Card
+cardsRemaining() : int
+resetDeck() : void
standard deck
Hand
-cards : ArrayList<Card>
-isActive : boolean
-handValue : int
+Hand() : void
+addCard(card : Card) : void
+getValue() : int
+isBust() : boolean
+isBlackjack() : boolean
+clearHand() : void
+toString() : String
contains
52
Card
-rank : Rank
-suit : Suit
-value : int
+Card(rank : Rank, suit : Suit) : void
+getRank() : Rank
+getSuit() : Suit
+getValue() : int
+toString() : String

• Sequence Diagrams

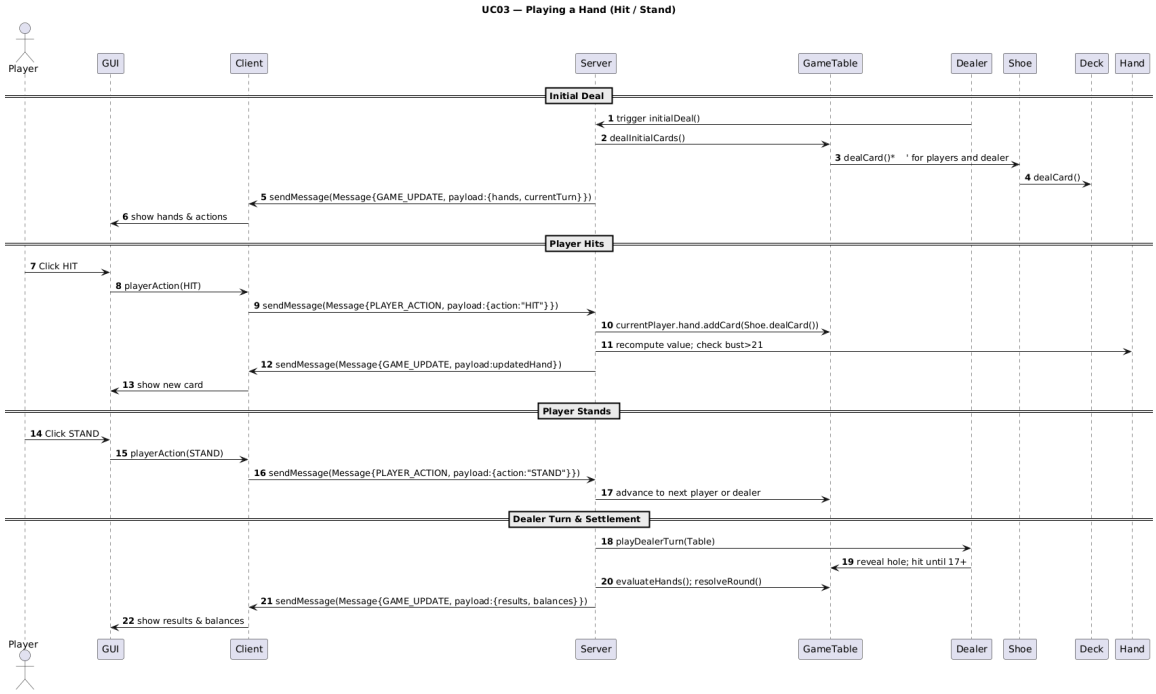
Use Case 1: Player Registration and Login



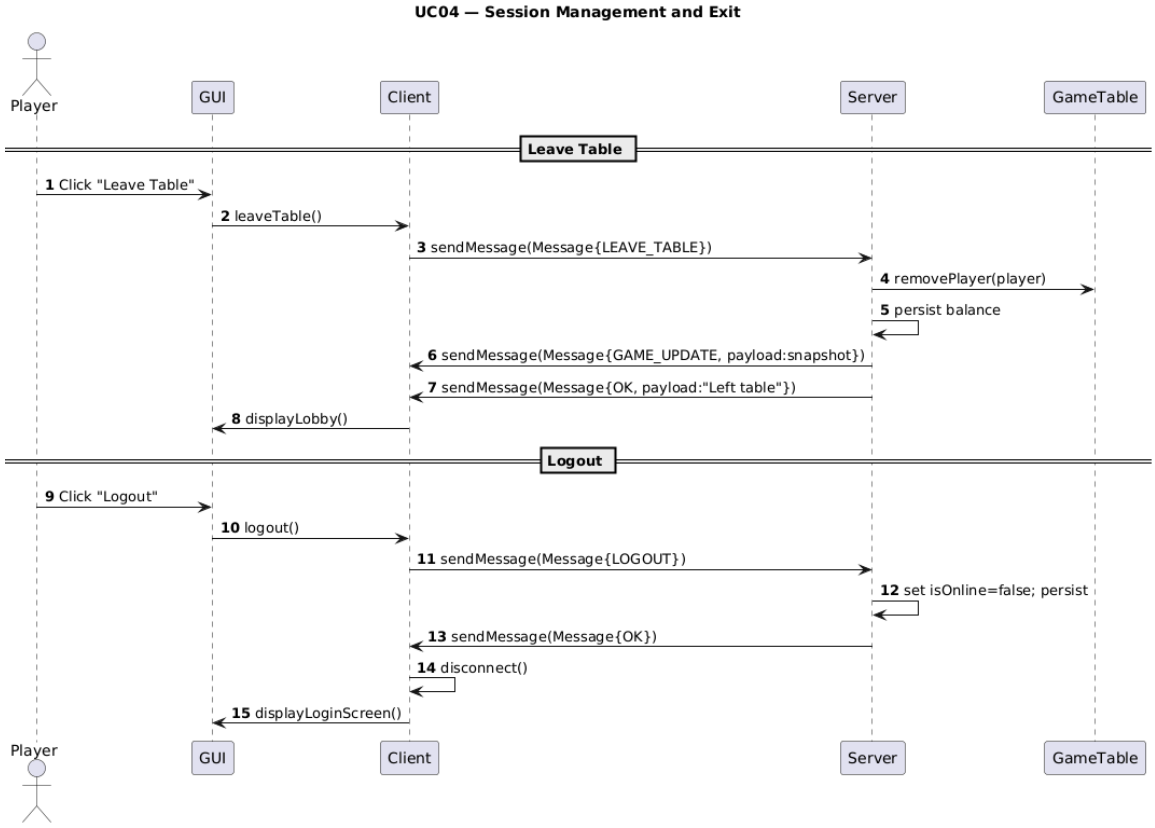
Use Case 2: Joining a Table and Placing a Bet



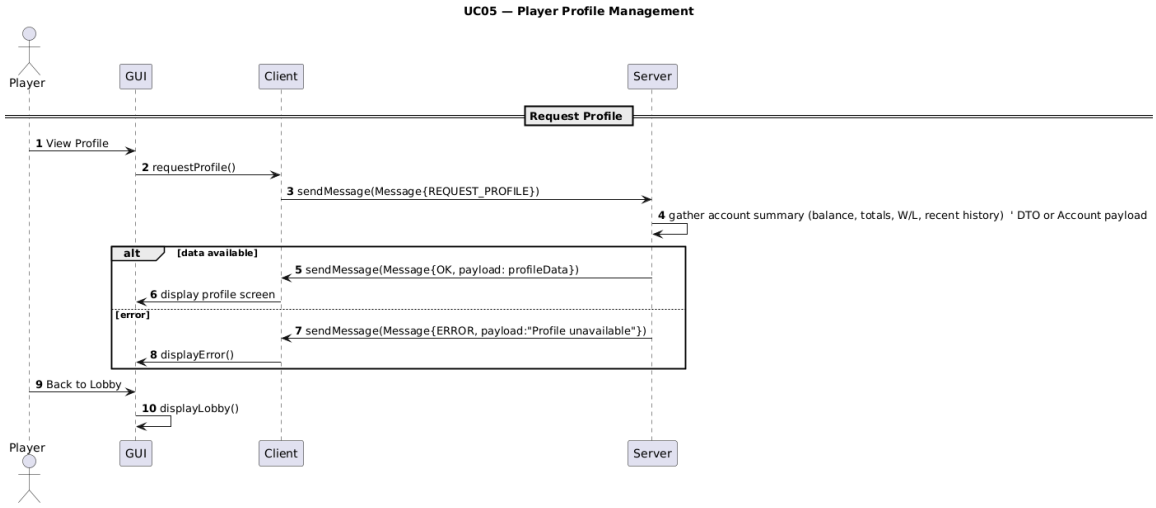
Use Case 3: Playing a hand



Use Case 4: Session Management and Exit



Use Case 5: Player Profile Management



Use Case 6: Advanced Game Actions (Split / Insurance / Double Down)