Multi-Player Blackjack

Blake, Rahul, Samuel

Top Requirements

- 1. User Authentication and Account Management
  The system must allow users to create accounts, log in with unique credentials, reset passwords, and manage account states (New, Active, Blocked, Banned). This ensures secure and controlled access. Passwords stored in a text file.

- 2. Secure Betting and Fund Management
  Players must be able to deposit and withdraw funds, place bets only if sufficient balance exists, and have all transactions logged for audit. Bets must follow game rules (≥ $1, integer values, no decimals).

- 3. Game Logic Enforcement
  The system must enforce Blackjack rules, including correct dealing, turn-taking, win/loss evaluation, busts, blackjack payout rules, and automated handling of timeouts (default to stand).

- 4. Fairness via Randomized Shuffling
  Card shuffling must be implemented using a random card generator. The system should simulate a multi-deck shoe and reshuffle once every round to prevent cheating and card counting.

Top Requirements

- 5. Real-Time Network Communication
  The client-server system must ensure real-time updates for all players at a table, including bets, dealt cards, and game outcomes. Players who disconnect should lose their bets(in case of ragequitting), while reconnection attempts must be supported.

- 6. Dealer Controls
  Dealers must be able to open tables, invite players from the lobby, and deal cards. They cannot leave mid-game, ensuring stability. The dealer always acts after players to follow standard Blackjack rules.

- 7. Table Management
  Each table must have exactly one dealer and 1–6 players. A unique ID must identify each table session. Tables should close automatically when no dealer is present or if no players join within a set time.

- 8. User Interface (GUI)
  The system must provide an interactive GUI for both players and dealers. It should display real-time actions (bets, cards, results), include animations (shuffling, dealing, chips), and provide separate dashboards for players vs. dealers.

Top Requirements

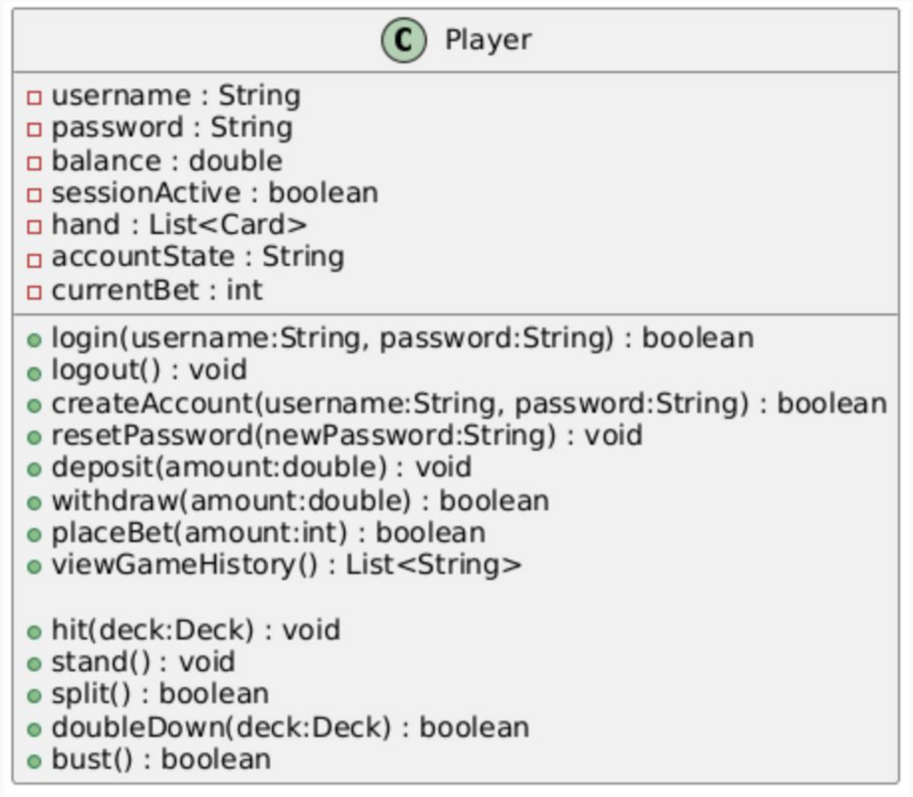
- 9. Security and Anti-Cheating
  The system must implement secure authentication (saving username/passwords in a text file), prevent duplicate sessions per account, and include anti-cheating measures such as multiple decks and reshuffling to counter card counting.

- 10. Performance and Scalability
  The system must support multiple concurrent tables, each with up to 6 players, while maintaining smooth real-time gameplay. It must handle disconnections gracefully to ensure the game flow is not disrupted.

- 11. Session management and timeout handling
  Each player account must allow only one active session. If a player quits, times out, or disconnects, the system must immediately terminate in-game actions, free the seat, and forfeit bets as appropriate

Player

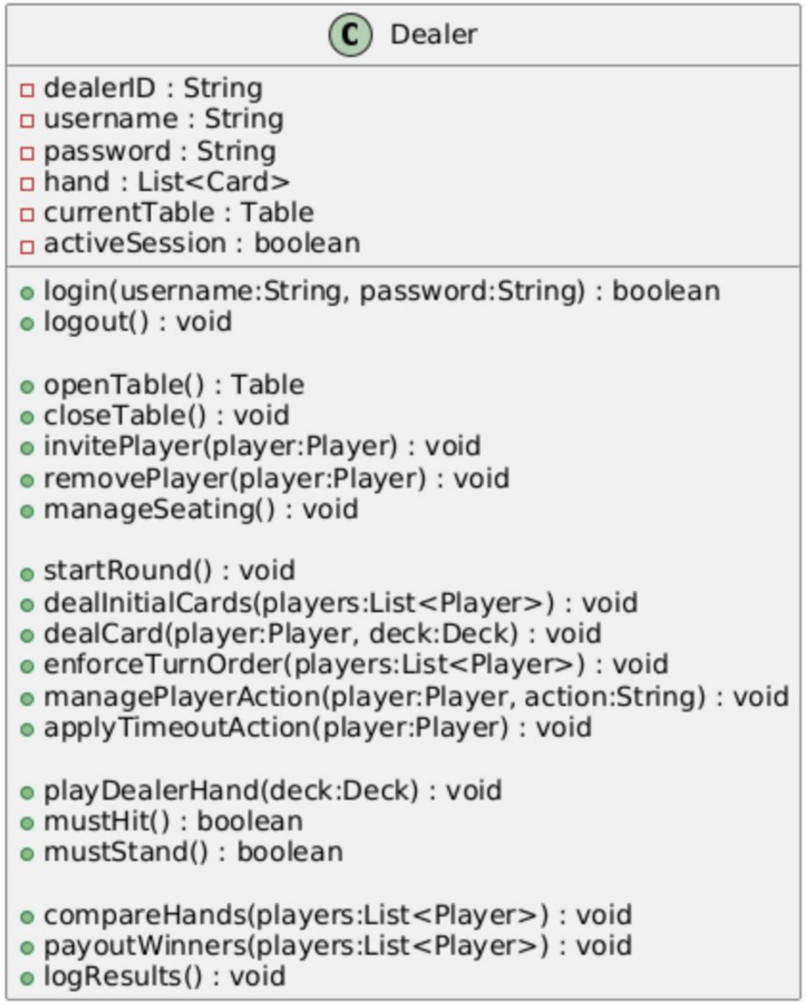
- login() allows the user to log into their account
- logout() allows the user to log out
- createAccount() allows the user to create account and sets to true after created
- resetPassword() allows user to reset password
- deposit() allows user to deposit money into their account
- withdraw() allows money to be withdrawn and if successful set to true
- placeBet() lets user place bet
- viewGameHistory() lets user view the games they've played
- hit() allows user to add another card to their hand
- stand() allows a user to end their turn
- split() shows if a user chooses to split
- doubleDown() shows if a user chooses to double down
- bust() shows if a user has a card total over 21
- Is an Account



```
username : String
password : String
balance : double
sessionActive : boolean
hand : List<Card>
accountState : String
currentBet : int

login(username:String, password:String) : boolean
logout() : void
createAccount(username:String, password:String) : boolean
resetPassword(newPassword:String) : void
deposit(amount:double) : void
withdraw(amount:double) : boolean
placeBet(amount:int) : boolean
viewGameHistory() : List<String>

hit(deck:Deck) : void
stand() : void
split() : boolean
doubleDown(deck:Deck) : boolean
bust() : boolean
```

Dealer

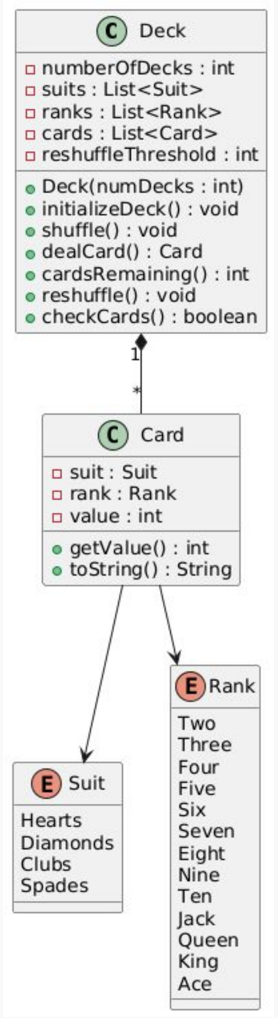
- login() and logout() allow the dealer to log in and out of their account
- openTable() allows the dealer to open a table for players to join
- closeTable() allows the dealer to close a table
- invitePlayer() allows the dealer to let a player play at a table
- removePlayer() lets dealer remove a player from the table
- manageSeating() lets dealer seat players
- dealInitialCards() has dealer deal 2 cards to all players
- dealCard() lets dealer deal a card to a player or itself
- enforceTurnOrder() lets dealer manage turn order
- playDealerHand() lets dealer deal themselves cards
- mustHit() makes dealer hit while their total is below the player's or below 17
- mustStand() makes dealer stand if total above player's or total >=17
- compareHands() lets dealer compare hands with player
- payoutWinners() lets dealer pay players
- logResults() logs results of who wins and loses



```
Dealer

dealerID : String
username : String
password : String
hand : List<Card>
currentTable : Table
activeSession : boolean

login(username:String, password:String) : boolean
logout() : void

openTable() : Table
closeTable() : void
invitePlayer(player:Player) : void
removePlayer(player:Player) : void
manageSeating() : void

startRound() : void
dealInitialCards(players:List<Player>) : void
dealCard(player:Player, deck:Deck) : void
enforceTurnOrder(players:List<Player>) : void
managePlayerAction(player:Player, action:String) : void
applyTimeoutAction(player:Player) : void

playDealerHand(deck:Deck) : void
mustHit() : boolean
mustStand() : boolean

compareHands(players:List<Player>) : void
payoutWinners(players:List<Player>) : void
logResults() : void
```

Deck

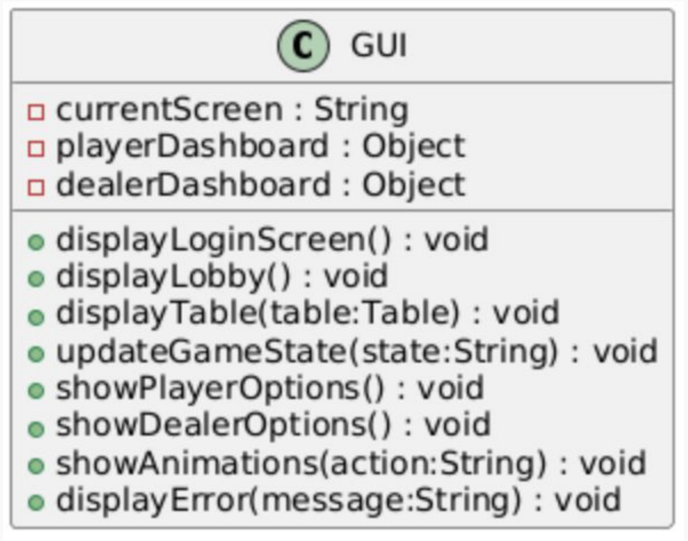
- 1 Deck will have 52 Cards
- shuffle() will be used to randomize the order of the Cards
- dealCard() removes one card from the Deck and places it on the table for either the Dealer or a Player
- cardsRemaining() shows how many cards are remaining in the shoe
- checkCards() ensures there are always 1-52 cards in the Deck
- reshuffle() allows the dealer to reshuffle at any point they feel
- Only Dealer can call Deck methods



GUI

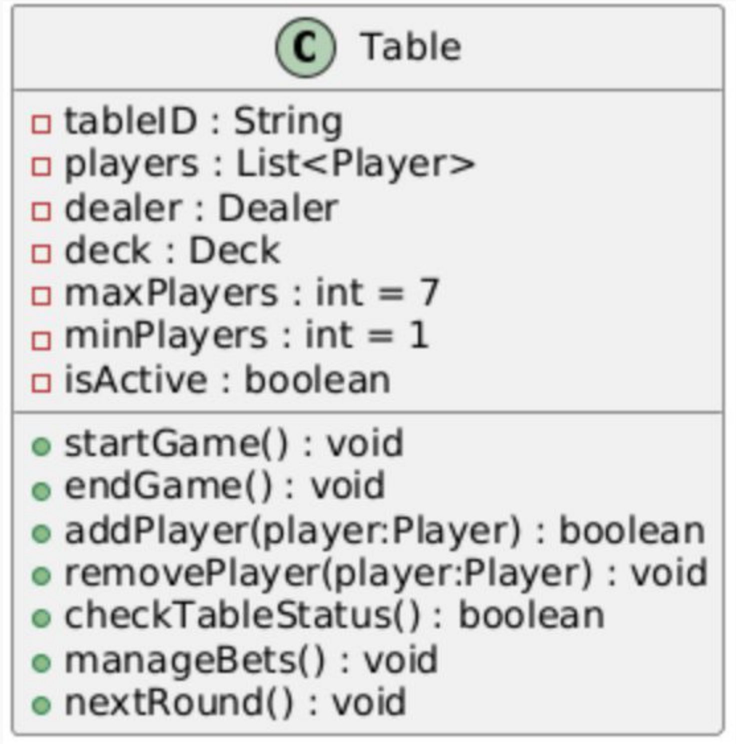
- GUI will manage what is displayed for the player in the login, lobby, and in-game screens.

- displayLoginScreen() shows the login screen
- displayLobby() shows the different lobbies available
- displayTable() shows the table players will play on
- updateGameState() determines if a player won
- showPlayerOptions() shows the options a player can do based on their balance and hand
- showDealerOptions() shows the options a dealer can do
- showAnimations() shows animations of cards being dealt, shuffled
- displayError() shows an error message if something is wrong


```
(C) GUI
□ currentScreen : String
□ playerDashboard : Object
□ dealerDashboard : Object
● displayLoginScreen() : void
● displayLobby() : void
● displayTable(table:Table) : void
● updateGameState(state:String) : void
● showPlayerOptions() : void
● showDealerOptions() : void
● showAnimations(action:String) : void
● displayError(message:String) : void
```

Table

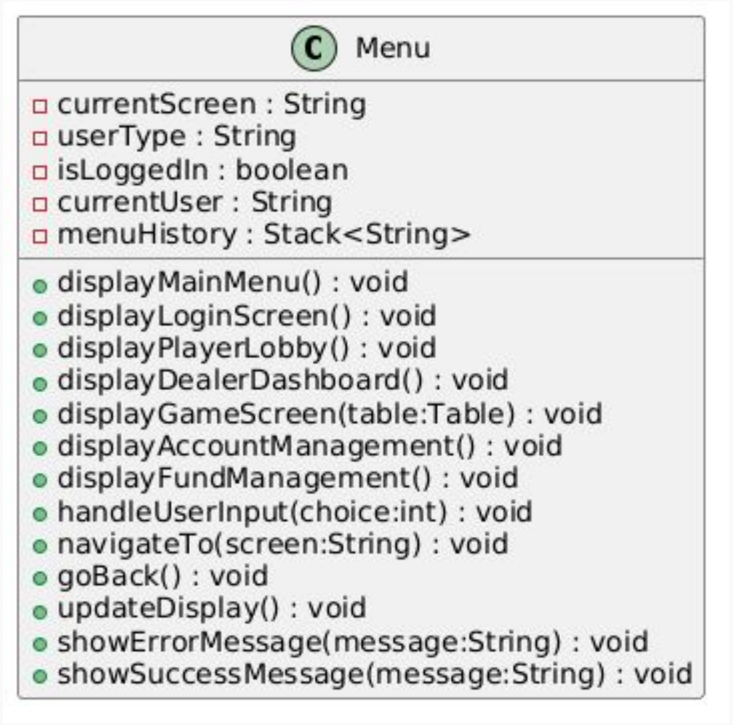
- startGame() allows system to start the game once there are enough players
- endGame() allows system to end the game if there are no players or no dealer
- addPlayer() allows system to add a player to a table
- removePlayer() allows system to kick player
- checkTableStatus() allows system to check if there are enough people at a table
- manageBets() allows system to manage bets for each player
- nextRound() allows system to move onto the next round once winnings/losses have been dealt



```
Table
□ tableID : String
□ players : List<Player>
□ dealer : Dealer
□ deck : Deck
□ maxPlayers : int = 7
□ minPlayers : int = 1
□ isActive : boolean
● startGame() : void
● endGame() : void
● addPlayer(player:Player) : boolean
● removePlayer(player:Player) : void
● checkTableStatus() : boolean
● manageBets() : void
● nextRound() : void
```

Menu

- Menu handles user interface navigation and the menu system
    - Manages flow of navigation between login, lobby, and game screens
- displayMainMenu() displays the initial app entry point with the login and account creation
- displayPlayerLobby() displays player's current view/main hub with available options to join a table as well as managing account
- displayPlayerLobby() and displayDealerDashboard() display different dashboards for player and dealer
- displayGameScreen() displays gave screen
- displayAccountManagement() and displayFundManagement() show setting for user profile and settings to manage funds
- showErrorMessage() and showSuccessMessage() show messages of error and success



```
Menu
□ currentScreen : String
□ userType : String
□ isLoggedIn : boolean
□ currentUser : String
□ menuHistory : Stack<String>
● displayMainMenu() : void
● displayLoginScreen() : void
● displayPlayerLobby() : void
● displayDealerDashboard() : void
● displayGameScreen(table:Table) : void
● displayAccountManagement() : void
● displayFundManagement() : void
● handleUserInput(choice:int) : void
● navigateTo(screen:String) : void
● goBack() : void
● updateDisplay() : void
● showErrorMessage(message:String) : void
● showSuccessMessage(message:String) : void
```

Thanks!