**Blackjack Project**

Group 5  
Software Requirements Specification

Revision History

| **Date** | **Revision** | **Description** | **Author** |
| --- | --- | --- | --- |
| 02/08/2023 | 1.0 | Initial Version | Regina |
| 02/20/2023 | 1.1 | Finished overall product description. | Grayson |
| 02/26/2023 | 1.2 | Finished Specific Requirements | Regina |
| 02/26/2023 | 1.3 | Finished Non Functional (Security, Privacy) | Anas |
| 02/26/2023 | 1.4 | Added Use Cases and UML Diagram | James |
| 02/26/2023 | 1.5 | Edited use case in section 2 | Grayson |
| 02/27/2023 | 1.6 | Finished Non Functional (Environmental) | Anas |
| 02/28/2023 | 1.7 | Updated the Table of Contents | James |
| 02/28/2023 | 1.8 | Finished 4.3 and modified 4.2 | Anas |
| 02/28/2023 | 1.9 | Finished section 1 | Grayson |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Github link:  
<https://github.com/graysonman/Software-Engineering-SRS>

Table of Contents

[**1. Purpose 5**](#_heading=h.1fob9te)

[1.1. Scope 5](#_heading=h.3znysh7)

[1.2. Definitions, Acronyms, Abbreviations 5](#_heading=h.2et92p0)

[1.3. References 5](#_heading=h.tyjcwt)

[1.4. Overview 5](#_heading=h.3dy6vkm)

[**2. Overall Description 6**](#_heading=h.1t3h5sf)

[2.1. Product Perspective 6](#_heading=h.4d34og8)

[2.2. Product Architecture 6](#_heading=h.2s8eyo1)

[2.3. Product Functionality/Features 6](#_heading=h.17dp8vu)

[2.4. Constraints 6](#_heading=h.3rdcrjn)

[2.5. Assumptions and Dependencies 7](#_heading=h.lnxbz9)

[**3. Specific Requirements 8**](#_heading=h.35nkun2)

[3.1. Functional Requirements 8](#_heading=h.1ksv4uv)

[3.1.1. Common Requirements: 8](#_heading=)

[3.1.2. Game Module Requirements: 8](#_heading=)

[3.1.3. Client Module Requirements: 8](#_heading=)

[3.1.4. Server Module Requirements: 8](#_heading=)

[3.2. External Interface Requirements 8](#_heading=h.4i7ojhp)

[3.3. Internal Interface Requirements 8](#_heading=h.1ci93xb)

[**4. Non-Functional Requirements 10**](#_heading=h.qsh70q)

[4.1. Security and Privacy Requirements 10](#_heading=h.3as4poj)

[4.2. Environmental Requirements 10](#_heading=h.49x2ik5)

[4.3. Performance Requirements 10](#_heading=h.23ckvvd)

[**5. Use Cases 11**](#_heading=h.kx7vab3tbr6w)

[**6. UML Structure Diagram 14**](#_heading=h.wjfina57vdtq)

[**7. Class Diagrams 15**](#_heading=h.42slmlkg8bvx)

# Purpose

This document outlines the requirements for the Blackjack game.

## Scope

This document will catalog the user and software requirements for the blackjack multiplayer card game. It will not, however, document how these requirements will be implemented.

## Definitions, Acronyms, Abbreviations

UI: User Interface

IP: Internet Protocol

VPN: Virtual Private Network

## References

Use Case Specification Document – Section 5

UML Use Case Diagrams Document – Section 6

Class Diagrams – Section 7

Sequence Diagrams – Section 6

## Overview

This game will be an implementation of the popular blackjack card game, including online multiplayer features.

# Overall Description

## Product Perspective

The Java based blackjack game will allow for clients to connect to a server. Clients are able to join games from the lobby. The lobby should show the players the game rooms available and allow them to join any not full room (max 5 players). The server will deal with syncing the game and player modules together and check for player side cheats in editing hand values. The client module will have a player class which will maintain the player’s hand and balance of the player as well as a status of the player being in a game or in the lobby. The game module will deal with all the rules of the game and actual running of the game. The dealer will be run under the game module and the deck would be included in the game module. Each round there will be a payout based upon winning and losing status of each player.

## Product Architecture

The design of the system will be as follows:

* Game module: This module is responsible for all game rules and running of the game. This will include the dealer and deck of cards as well as determining who wins or loses.
* Client module: This module will have the players statistics and deal with the inputs from the user.
* Server module: This module is responsible for connecting other players and checking servers values against players values to mitigate cheating. This will directly interact with the client module and relay it to the game module.
* UI module: This module is responsible for displaying the activity of the game as well as the lobby interface. This will display all aspects of the game module and player module. It will be directly interacting with the game, server, and client module.

## Product Functionality/Features

The high-level features of the system are as follows (see section 3 of this document for more detailed requirements that address these features):

* Ability for players to play against other players in real time.
* Play black jack with standard rules including hits and doubling down.
* Support for multiple games to be running at the same time.
* Real time communication between players and server for smooth gaming experience.
* Ability to add money to an account or withdraw after logging into their account

## Constraints

Software must be implemented with Java.

Software must be able to handle a max player count of 10,000 concurrent users

Software must be able to to run on any computer platform that uses Java

Software must have error catching for any problems that occur due to sync issues or problems with user misinputs.

## Assumptions and Dependencies

Software requires an internet connection to run.

Software requires that Java is on the computer to run the program.

Software depends on server hardware to scale user count and store user information.

Software depends on hardware for the speed at which the server can run and maintain all instances of games.

Software may depend on third party libraries or frameworks in order to run properly.

# Specific Requirements

## Functional Requirements

### Common Requirements:

3.1.1.1 The dealer should be run by the program and not a real person and players must all be real people, no bots or automated users are allowed to be players.

3.1.1.2 There should be a minimum and maximum amount of money that can be held in Player’s accounts

### Game Module Requirements:

3.1.2.1 Each room should have a maximum of 5 players.

3.1.2.2 Players must not be allowed to join any full rooms

3.1.2.3 There must be at least one player in a game table lobby to start a game. Empty games should not be running the dealer with no players active.

3.1.2.4 There needs to be a limit on how much one person can bet as well as the minimum bet for initial game and a per round basis.

### Client Module Requirements:

3.1.3.1 Players will be allowed to leave games during matches but must forfeit their bet to the other players. Players will be allowed to leave games between matches with no penalty.

3.1.3 This module will communicate with the server module regarding game related and account related events.

### Server Module Requirements:

3.1.4.1 The balance will be server sided meaning it can’t be modified by a cheater or cheating software. The balances of a player will be checked against the server sided balance on a per round basis. This check will include checking cards against what the server dealt to players so they can’t change the value of their cards.

3.1.4.2 Account creation should not allow duplicate player accounts. Player usernames/names must be unique as balance will be tied to the player’s username.

3.1.4.3 The server will keep track of all players’ balances, updating the balance for the winning player post game.

## External Interface Requirements

3.2.1 This application needs an IP address to run the server and communicate with other computers over the network.

3.2.2 There needs to be a banking and cashing out system that allows the user to see and adjust the account balance inside and outside of the game.

3.2.3 There must be a way to join games and a lobby for where people can select where to go and the status of the game and players in the room.

## Internal Interface Requirements

3.3.1 The system must process cards played during each player's turn.

3.3.2 The system must update the users balances when they deposit and withdraw funds.

3.3.3 The system must process users login information to verify that they have the correct login information

# Non-Functional Requirements

## Security and Privacy Requirements

4.1.1 Any suspicious activity must be monitored and must notify the admin.

4.1.2 Security questions must be set up during account creation to prevent being locked out of the account or unauthorized access to the user's account.

4.1.3 Lock the account after three failed login attempts.

## Environmental Requirements

4.2.1 Able to save the user's current state if they get disconnected or timeout due to their connection. User’s might be able to resume if rejoined within a certain amount of time.

4.2.2 Sufficient storage to allow this program to be installed on the hardware.

4.2.3 System should support cross-play between different Operating Systems.

## Performance Requirements

4.3.1 System must be able to provide reasonable response time even with a high number of players (i.e., players >= 1,000).

4.3.2 Money withdrawing capability for current users is prioritized and should be unaffected in case of a huge number of users beyond which the system can accommodate joins at the same time (i.e., players >= 10,000).

4.3.3 Should be able to provide more bandwidth on demand to keep servers up if the current bandwidth limit is about to be exceeded.

# 5. Use Cases

**Use Case ID: 100**

Use Case Name: User Login

Relevant Requirements: 3.1.1.1

Primary Actor: Server Module

Pre-conditions: None.

Post-conditions: User is logged in and can play blackjack

Basic Flow or Main Scenario:

1. Login request comes from client connection with a username and password
2. Server authenticates the user to verify their identity
3. Server responds to the client with a success message, and allows access to the game lobby and their account details

Extensions or Alternate Flows: User login information is invalid, and the server rejects their authentication.

Exceptions: Login attempt when already logged in.

Related Use Cases: 101 - User Logout

**Use Case ID: 101**

Use Case Name: User Logout

Relevant Requirements: 3.1.1.1

Primary Actor: Server Module

Pre-conditions: User is logged in.

Post-conditions: User is logged out.

Basic Flow or Main Scenario:

1. Login request comes from client connection with a username and password
2. Server authenticates the user to verify their identity
3. Server responds to the client with a success message, and allows access to the game lobby and their account details

Extensions or Alternate Flows: None.

Exceptions: Logout attempt when already logged out.

Related Use Cases: 100 - User Login

**Use Case ID: 102**

Use Case Name: Join Game Table

Relevant Requirements: 3.1.2.1, 3.1.2.2

Primary Actor: Game Module

Pre-conditions: None.

Post-conditions: User has joined a game table.

Basic Flow or Main Scenario:

1. Game-join request comes in from a client.
2. If there is available space at the table, user is added to it.

Extensions or Alternate Flows: None.

Exceptions: Attempted to join a full table.

Related Use Cases: 103 - Join Game Table

**Use Case ID: 103**

Use Case Name: Leave Game Table

Relevant Requirements: 3.1.3.1

Primary Actor: Game Module

Pre-conditions: Player is seated at a game table

Post-conditions: Player is no longer part of that game.

Basic Flow or Main Scenario:

1. Game at a table is complete, and a user requests to leave the table and collect their winnings.
2. Game module calculates winnings, adds them to the user’s balance, and removes them from the table.

Extensions or Alternate Flows: If game at table is still in-progress, player will forfeit their winnings by leaving early.

Exceptions: Attempting to leave a game when not presently in one.

Related Use Cases: 102 - Join Game Table

**Use Case ID: 200**

Use Case Name: Start Game

Relevant Requirements: 3.1.2.3

Primary Actor: Game Module

Pre-conditions: At least 2 players are at a table.

Post-conditions: Game started at a specific table.

Basic Flow or Main Scenario:

1. Players collectively request to start playing at their table
2. If the game determines that there are at least 2 players at the table, the game can begin.

Extensions or Alternate Flows: None.

Exceptions: Attempting to start a game with less than 2 human players at the table.

Related Use Cases: None.

**Use Case ID: 201**

Use Case Name: Deal Cards to Players

Relevant Requirements: None.

Primary Actor: Game Module

Pre-conditions: Game is started and it is the beginning of the round.

Post-conditions: Starting cards are dealt to all players and the dealer.

Basic Flow or Main Scenario:

1. Round begins at game table.
2. Player by player, two cards are dealt to each players’ hand.

Extensions or Alternate Flows: None.

Exceptions: None.

Related Use Cases: None.

**Use Case ID: 300**

Use Case Name: Check Balance

Relevant Requirements: 3.1.4.3

Primary Actor: Server Module

Pre-conditions: Player has an account in the server.

Post-conditions: Balance information transmitted to the requesting client

Basic Flow or Main Scenario:

1. Logged-in user requests balance information.
2. Server sends balance information to the client.

Extensions or Alternate Flows: None.

Exceptions: None.

Related Use Cases: 301 - Deposit Balance, 302 - Withdraw Balance

**Use Case ID: 301**

Use Case Name: Deposit Balance

Relevant Requirements: 3.1.1.2, 3.1.4.3

Primary Actor: Server Module

Pre-conditions: Player has an account in the server.

Post-conditions: Balance information updated

Basic Flow or Main Scenario:

1. Logged-in user requests to add money to their account balance.
2. The user is prompted to enter the amount they would like to add.
3. The requested funds are added to the account.

Extensions or Alternate Flows: None.

Exceptions: The user attempts to deposit an amount that would cause their balance to exceed the maximum allowed value.

Related Use Cases: 300 - Check Balance, 302 - Withdraw Balance

**Use Case ID: 302**

Use Case Name: Withdraw Balance

Relevant Requirements: 3.1.4.3

Primary Actor: Server Module

Pre-conditions: Player has an account in the server.

Post-conditions: Balance information updated

Basic Flow or Main Scenario:

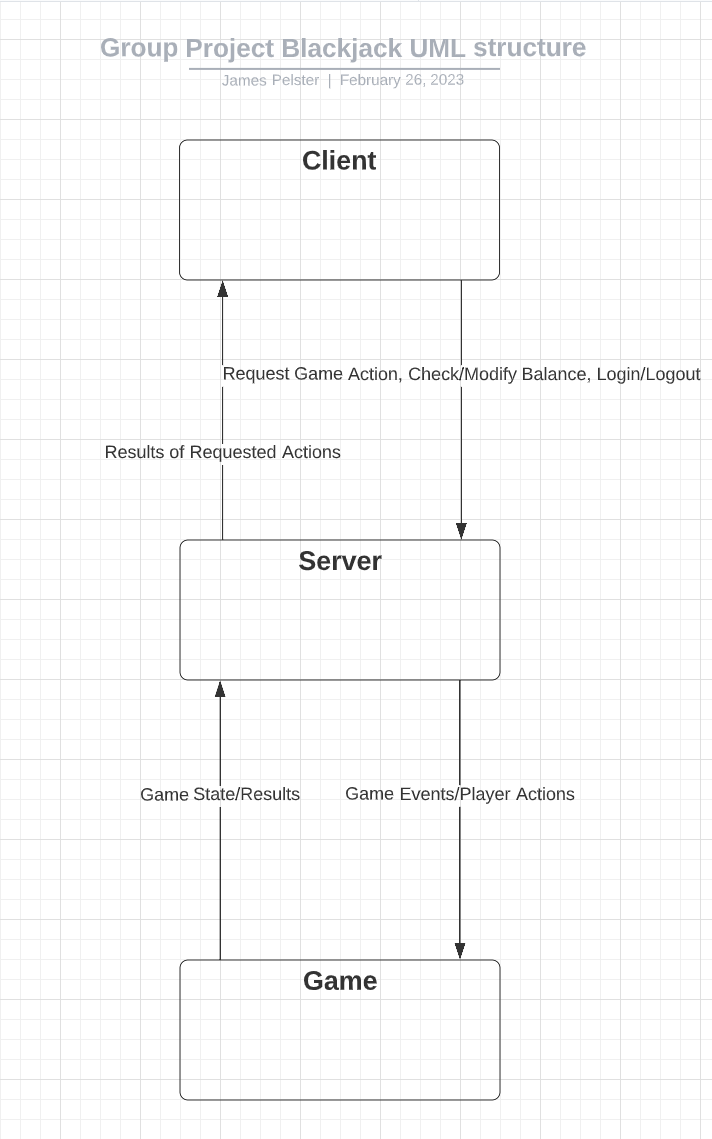
1. Logged-in user requests to cash out a specified amount from their account balance.
2. If their account holds at least that many funds, deduct them from the balance.

Extensions or Alternate Flows: None.

Exceptions: The user attempts to withdraw more than their balance.

Related Use Cases: 300 - Check Balance, 301 - Deposit Balance

# 6. UML Structure Diagram



# 7. Class Diagrams

| **Client** |
| --- |
| -current\_user : Player  -user\_interface : Interface  -server\_ip : java.net.InetAddress |
| getGameDetails(int) : String  joinGame(int) : void  sendMove(int) : void  receive() : Message |

| **Player** |
| --- |
| -id : int  -username : String  -password : String  -balance : float  -hand : array  -current\_bet : float |
| getBalance() : float  addFunds(float) : void  increaseBet(float) : void  handTotal() : int  requestCard() : void  emptyHand(): void |

| **Server** |
| --- |
| -gameList : ArrayList<Game>  -userList : ArrayList<Player> |
| listenForClients() : void  sendMessage(java.net.InetAddress) : void  addGame(): void  addPlayer(Player): void |

| **Game** |
| --- |
| -id : int  -house\_balance: float  -lobby : ArrayList<Player>  -turn : int  -is\_active : Boolean  -betPool : float |
| tableTotal() : int  getPlayerCount(): int  nextTurn(): void  collectPool(): void  awardPool(Player): float |

| **Dealer** |
| --- |
| -deck : ArrayList<Card>  -hand: array |
| reset() : void  shuffle() : void  dealCard() : Card  getRemaining() : int  handTotal(): int  drawCard(): Card  emptyHand(): void |

| **Card** |
| --- |
| -value: int  -name : string  -suit : enum Suit |
| -setAceValue(int) : void |

| enum **Suit** |
| --- |
| HEARTS  DIAMONDS  CLOVERS  SPADES |