Functional Specification: Server Component

# Program Flow

## Program Flow: Registration

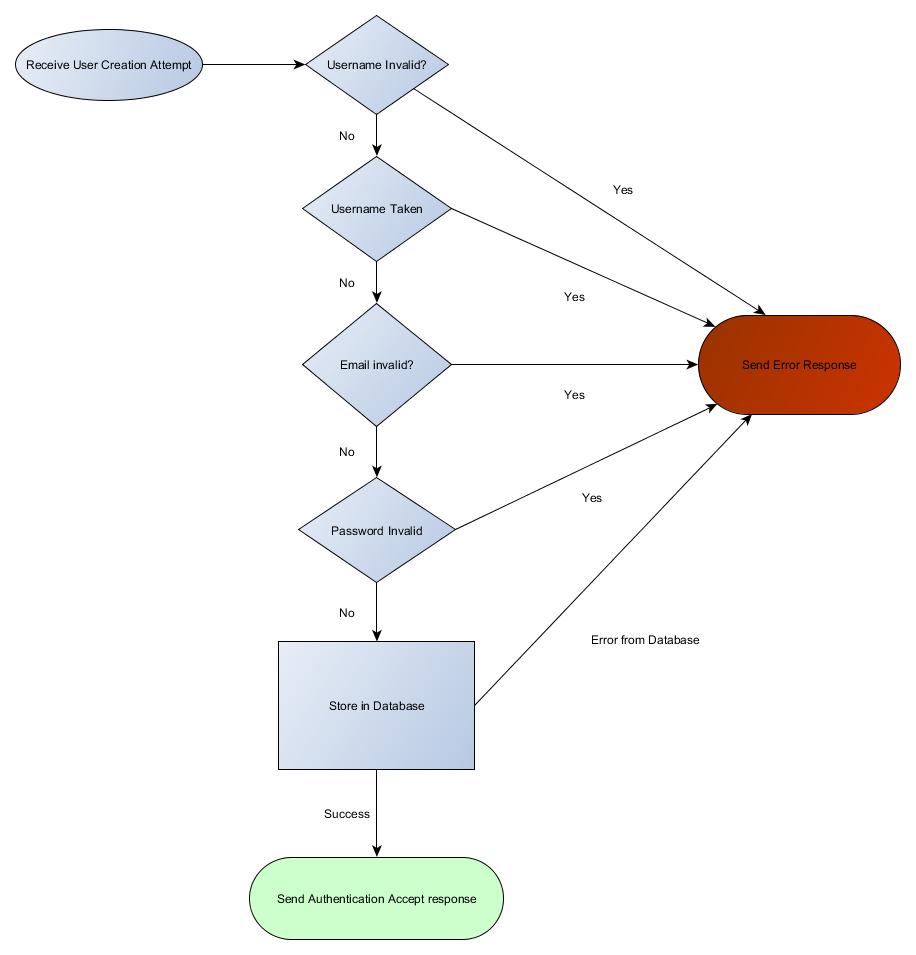


Figure 1: User Creation Attempt

During the registration process, a success message shall only be sent if the username is valid and not already in use, the email is valid and not already in use, and the password is valid. At that point, the new user shall be stored in the database, and a success message returned. If any of the preconditions fail, or if the database cannot be written to for some reason, an error shall be returned.

This shall be handled by the DAO object instantiated when the server was started.

## Program Flow: Login

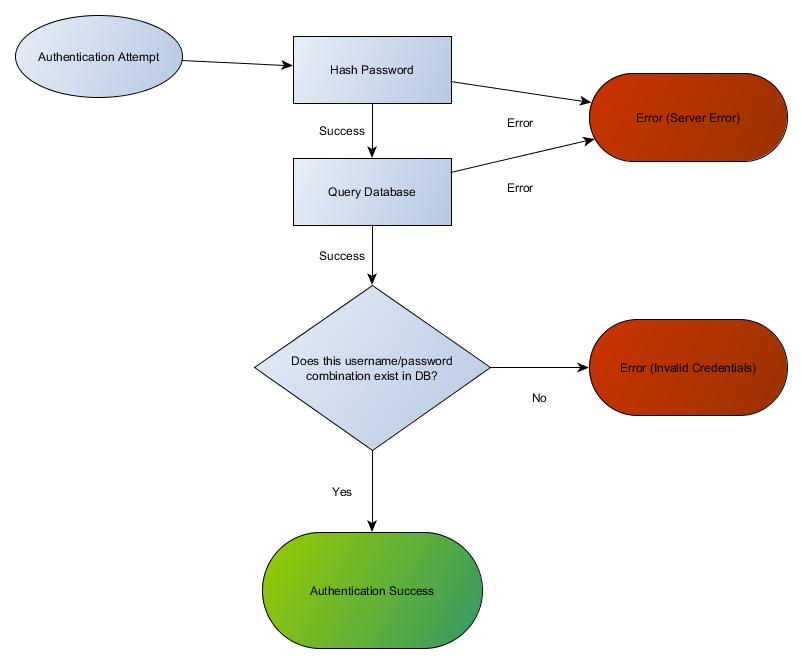


Figure 2: Authentication Attempt

During the login process, the database shall be queried with the combination of username and hashed password. If a record does not exist with that combination in it, an invalid credentials error shall be sent to the client. If there is an issue accessing the database, an error shall be sent to the client. Otherwise, a success message shall be sent back to the client.

## Program Flow: Joining a Game

## Program Flow: Game Actions

# Interfaces

## iClient

This represents a client that can send and receive messages

**Instance Methods**

* receiveMessage
* sendMessage

## iClientObserver

This represents something that observes a client. It defines a common callback to be used.

**Instance Methods**

* onMessageReceived

## iGameObserver

This represents something that observes the state of a game. It defines a common callback to be used.

**Instance Methods**

* onGameStateChanged

## iObservable

This represents something that can be observed.

**Instance Methods**

* addObserver
* removeObserver

# Classes

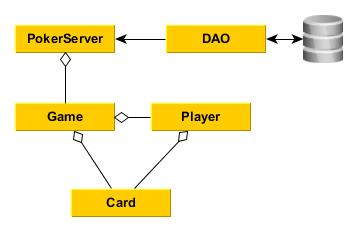


Figure 3: Class Diagram

## PokerServer

Instances of the class “PokerServer” shall represent a single concurrent server running. Each instance of the server shall listen on a specific port. It shall have the responsibility for managing connections and creating game objects as needed.

**Instance Members:**

* ArrayList<Game> activeGames
* ArrayList<Player> connectedPlayers
* ArrayList<Player> lobbyPlayers
* DAO Database

**Instance Methods:**

* main
* createGame
* addClient

## DAO

Instances of the class “DAO” shall represent individual databases in which data may be persisted. In practice, there should only be one instance per server.

**Instance Members**

**Instance Methods**

* updatePlayer
* checkCredentials
* addPlayer

## Game <iObservable>

Instances of the class “Game” shall represent a single game currently in progress. It shall handle the turns of the game, including enforcing the business rules. There is the possibility of observing a game without being a player in it, though no client currently implements this behavior.

**Instance Members**

* ArrayList<Players> players
* ArrayList<iGameObserver> observers
* ArrayList<Card> tableCards
* ArrayList<Card> deck
* Integer chipsInPot

**Instance Methods:**

* attemptTurn
* dealCard
* addPlayer
* removePlayer

## Card

Instances of the class “Card” shall represent individual cards in a deck. A deck consists of 52 Card objects, each different from the last.

**Instance Members**

* Enum suit
* String value

**Instance Methods**

**Class Methods**

* generateDeck

## Player <iGameObserver, iClientObserver>

Instances of the class “Player” shall represent a single player, either in a game or in the lobby.

**Instance Members**

* String username
* URL avatarURL
* Integer chipsRemaining
* ArrayList<Card> currentHand
* Game currentGame
* iClient client

**Instance Methods:**

* addCardToHand
* turnStart
* placeBet
* fold
* getAvatar
* getClientForPlayer
* joinGame
* leaveGame

## ClientSocket <iClient, iObservable>

Instances of the class “ClientSocket” represent external clients connected to the system via TCP sockets. This class will send messages to the client and listen for messages from the client, automatically rejecting anything that is not signed appropriately. This implements the iClient interface.

**Instance Members**

* Socket socket
* String authenticationKey
* ArrayList<iClientObserver> observers