Implementation Planning Document

**Class: "CredentialsDatabase"**

- **Fields:**
  - HashMap <String, Player> playerCredentials
    *Key would be the username and value would just be the player object*

- **Constructor → public CredentialsDatabase()**
  - We will create a new HashMap
  - Then we will loadDatabase (loading database.txt file) → invoke the method *(so basically we will be storing the database in a database.txt and whenever we start a new session of login page (creating a new instance of loginPage in our main function) we will be creating a database instance too which will load the previous data/information about players from a database.txt file into the instance of the database and pass it on to the loginPage instance as a parameter)*
  -
- **Methods:**
  - usernameLookup(username) : boolean
  - addNewPlayer (username, email, password, null, null, null, null) : boolean
  - deleteExistingPlayer(username) : boolean
  - findPlayerByUsername(username) : Player
    *(It will look up the username in the hashmap and just return its value → player obejct*)
  - saveDatabase : void
    *(It will save all the information to the database.txt file before every logout session → store username, and then all the fields separately like password, email, ranking, connect4Stats, ticTacToeStats, checkerStats)*
  - loadDatabase : void
    *(It will load all the information from the database.txt file into the hashmap everytime main is run and main creates a new instance of Database class → take all the fields stored in database.txt file and create a new player object and then store it into the hashmap)*

**Class: "Player"**

- **Fields:**
  - Username / playerID : String
  - Email : String
  - Password : String
  - Ranking : Rank (Enum)
  - connect4Stats : Connect4Stats
  - ticTacToeStats : TicTacToeStats
  - checkerStats : CheckerStats

- **Constructor → public Player (username)**
- **Methods :**
    - getStats(gameType : String) : GeneralStats
    - All the other getter and setter methods for the fields
    - updateUsername(old username, new username)
    - updateEmail(username, email)
    - updatePassword(username, password)

**Class: "LoginPage"**
*//instance of LoginPage will be created by our main function with database as a parameter*

- **ENUM CLASS** : State (Enum) //this is going to have a separate box diagram
    - UsernameTaken
    - EmailFormatWrong
    - VerificationCodeWrong
    - Success

- **Fields :**
    - database : CredentialsDatabase

- **Constructor** → LoginPage( CredentialsDatabase : database)
- **Methods**:
    - login(username, password) : HomePage
    - *(We will basically be doing a lookup in the database object like Player player = database.findPlayerByUsername(username) and then we will check if the player is not null (the username exists in the database and the password entered is the same as player.getPassword and everything is correct, we will create a new Homepage with the player and database as its parameter and return that, so basically the homepage will be associated to that player, or if not then we will return null. So the existence of homePage will determine if the login was successful or not. Something like:*
      *HomePage home = loginPage.login("player1", "password123");*
      *if (home != null) { System.out.println("Login successful!"); )*

    - register (username, password, email) : ENUM
      *(Just check if entered code is not empty for verification purpose)*
    - forgot password (username) : boolean

**Class: "HomePage"**

- **Fields:**
    - player : Player
    - database : CredentialsDatabase
    - accountSettings : Settings

- **Constructor:**
  **public HomePage(Player player, Database database)**
  **this.accountSettings = new Settings(player, database)**
- **Methods:**
  - viewYourOwnRecords:
  - PlayerStats playerStats = new PlayerStats(player)

  - viewOtherPlayerRecords:
  - PlayerStats otherPlayerStats = new PlayerStats(otherPlayer)

  - viewSettings : return accountSettings of type Settings

  - playConnect4() : void
  - playTicTacToe() : void
  - playCheckers() : void

**Class : PlayerStats**

```
public class PlayerStats {
    private Player player;

    public PlayerStats(Player player) {
        this.player = player;
    }

    public int getWinsForConnect4() { return player.getStats("Connect4").getWins(); }
    public int getLossesForConnect4() { return player.getStats("Connect4").getLosses(); }
    public int getTiesForConnect4() { return player.getStats("Connect4").getTies(); }
    public Rank getRankForConnect4() { return player.getStats("Connect4").getRank(); }

    public int getWinsForTicTacToe() { return player.getStats("TicTacToe").getWins(); }
    public int getLossesForTicTacToe() { return player.getStats("TicTacToe").getLosses(); }
    public int getTiesForTicTacToe() { return player.getStats("TicTacToe").getTies(); }
    public Rank getRankForTicTacToe() { return player.getStats("TicTacToe").getRank(); }

    public int getWinsForChecker() { return player.getStats("Checker").getWins(); }
    public int getLossesForChecker() { return player.getStats("Checker").getLosses(); }
    public int getTiesForChecker() { return player.getStats("Checker").getTies(); }
    public Rank getRankForChecker() { return player.getStats("Checker").getRank(); }
}
```

**Class: Settings**
**Fields:**

- Player
- Database

**Constructor:**

**public Settings (Player player, CredentialsDatabase)**

this.player = player, this.data = database

**Methods**:

- deleteAccount(password): boolean
- changeUsername(newUsername) : boolean
- changeEmail(newEmail) : boolean
- *(Just check if the player has entered a non-empty verif. code)*
- changePassword(oldPassword, newPassword): boolean
- logout : boolean

*(Implementation : We will be saving data from the hashmap database object to the database.txt and then logout. More to be discussed on how to reflect the changes in the hashmap database object, is it our team doing it? Whenever a player wins a match or their ranking changes, who will be updating that change to the credentials database hashmap)*