Cairo University  
Faculty of Computers and Information



**CS251**

**Software Engineering I**

GoFo Booking System

Software Design Specifications

And Implementation

Version 2.0

Mahmoud Mohamed Abdelazim : [mahmoud2011989@gmail.com](mailto:mahmoud2011989@gmail.com)

Ahmed Alaa Eldin : [Ahmedshikhtawel@gmail.com](mailto:Ahmedshikhtawel@gmail.com)

Amr Bumadian : [Amrbomadian8@gmail.com](mailto:Amrbomadian8@gmail.com)

Ziad Amr: [ziadamr129@gmail.com](mailto:ziadamr129@gmail.com)

Eslam Fawzy: [eslamfawzy103@gmail.com](mailto:eslamfawzy103@gmail.com)

May 2020

Contents

[1. Team 3](#_Toc40016739)

[2. Document Purpose and Audience 3](#_Toc40016740)

[3. System Models 3](#_Toc40016741)

[I. Class Diagram(s) 3](#_Toc40016742)

[II. Class Descriptions 5](#_Toc40016743)

[III. Sequence diagrams 5](#_Toc40016744)

[Class - Sequence Usage Table 6](#_Toc40016745)

[IV. User Interface Design 7](#_Toc40016746)

[4. Tools 7](#_Toc40016747)

[5. Ownership Report 7](#_Toc40016748)

[6. References 7](#_Toc40016749)

[Appendix A: Code Listing and Screen Snapshots 8](#_Toc40016750)

[Authors 8](#_Toc40016751)

# Team

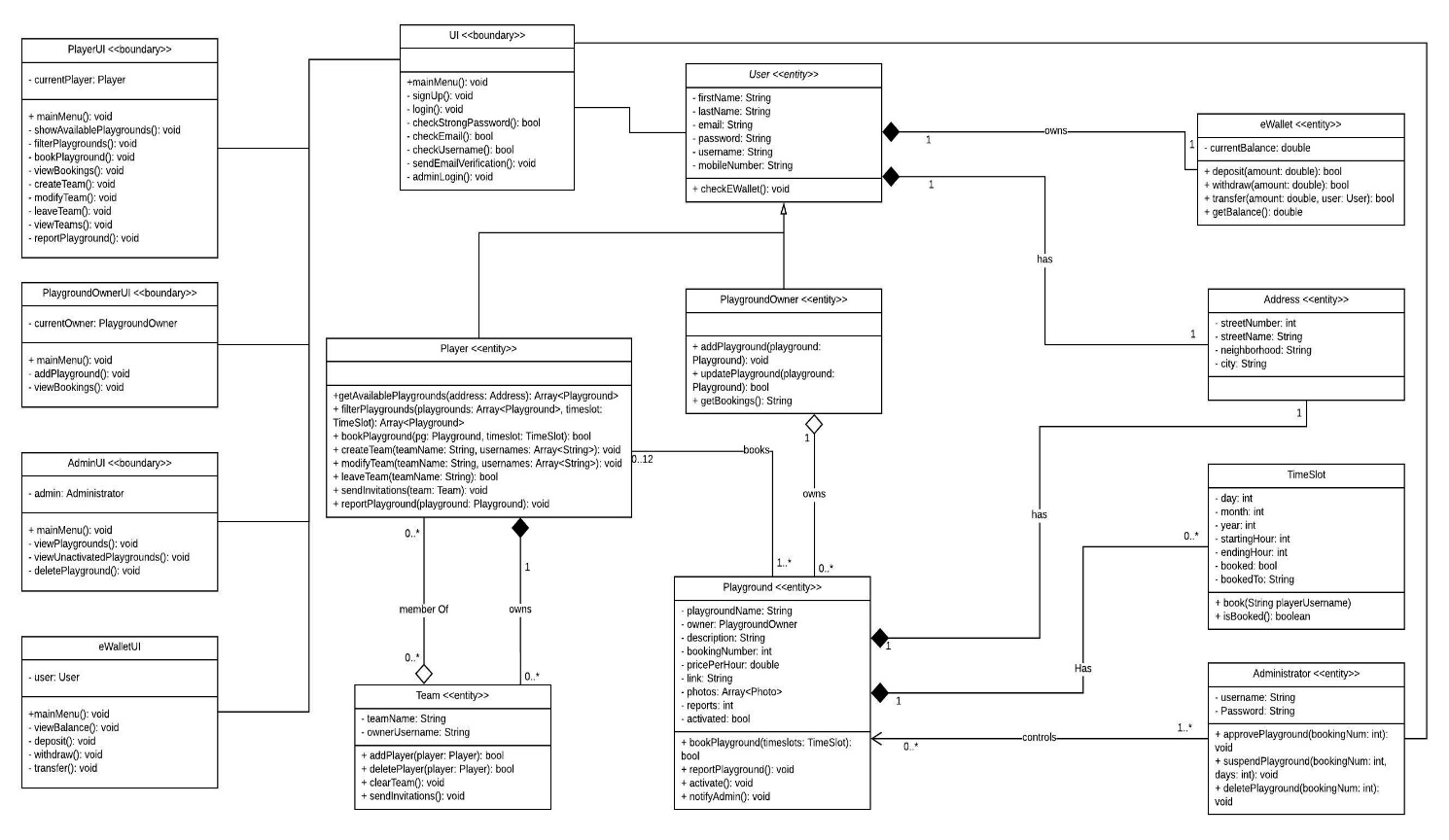
|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Name** | **Email** | **Mobile** |
| 20180263 | Mahmoud Mohamed Abdelazim | [Mahmoud2011989@gmail.com](mailto:Mahmoud2011989@gmail.com) | 01125401805 |
| 20180435 | Ahmed Alaa Eldin | [Ahmedshikhtawel@gmail.com](mailto:Ahmedshikhtawel@gmail.com) | 01142510014 |
| 20180436 | Amr Bumadian | [Amrbomadian8@gmail.com](mailto:Amrbomadian8@gmail.com) | 01032838645 |
| 20180379 | Ziad Amr | [Ziadamr129@gmail.com](mailto:Ziadamr129@gmail.com) | 01116202887 |
| 20180367 | Eslam Fawzy | eslamfawzy103@gmail.com | 01013315724 |

# Document Purpose and Audience

* **This document includes Software Design Specifications for GoFo football playground booking system. It describes the Class diagrams, Sequence diagrams and State diagrams.**
* **It is specially meant for developers that will work on the project in order to be organized.**

# System Models

## I. Class Diagram(s)



## 

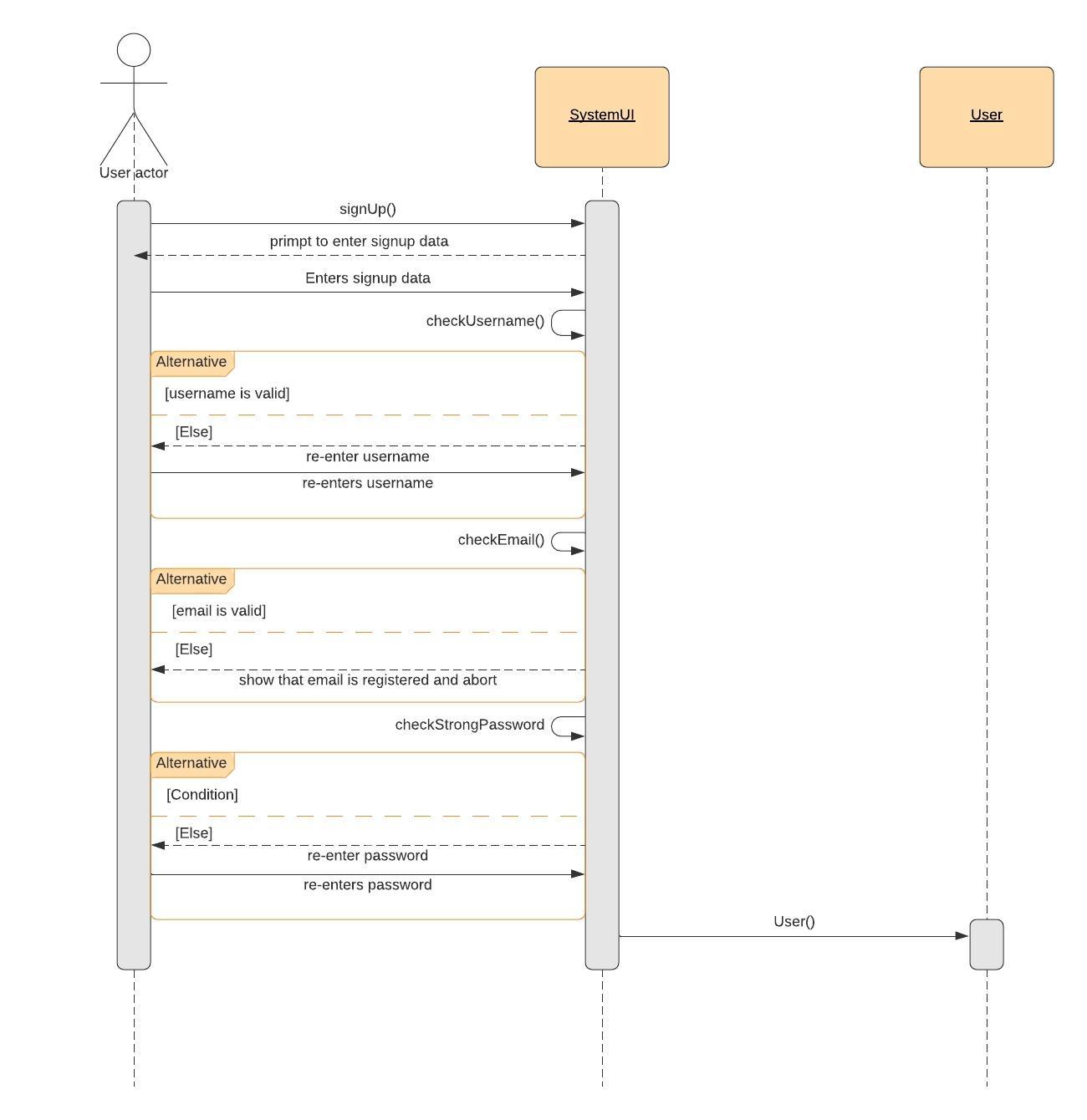
## II. Class Descriptions

| **Class ID** | **Class Name** | **Description & Responsibility** |
| --- | --- | --- |
| 1 | User | An abstract class (generalization) that holds the common properties between a Player and a Playground Owner |
| 2 | Player | A class that inherits User, and allows player to book playgrounds and do player functionalities. |
| 3 | PlaygroundOwner | A class that inherits User, and allows an owner to add (own) playgrounds and control them |
| 4 | Playground | A class that hold the attributes and operations that are done on a Playground, and it is ownder by a PlaygroundOwner |
| 5 | Team | A class that is meant to group players together to form a team, and has a team owner Player and team member Players. |
| 6 | eWallet | A class that has the responsibility of controlling financial operations in the system, and can be connected to an outside method like Fawry, and any User must have one eWallet. |
| 7 | Address | A class that hold an address information divided into pieces in order to make the process of searching for playgrounds in a specific area easier, and it’s owned by Users. |
| 8 | Administrator | A class that can do special operations concerning playgrounds such as delete or suspend a playground |
| 9 | TimeSlot | A class that is meant to store time slot data which is a date and a starting hour and an ending hour. |
| 10 | UI | A boundary class for enabling users to sign up and login to the system, |
| 11 | PlayerUI | A boundary class for enabling players to interact with the system, book playgrounds, etc. |
| 12 | PlaygroundOwnerUI | A boundary class for enabling owners to interact with the system, add playgrounds, review their current bookings, etc. |
| 13 | AdminUI | A boundary class for enabling admins to control playgrounds, like activating, suspending or deleting a playground |
| 14 | eWalletUI | A boundary class for users whether players of owners to check their eWallet balance, deposit using an external service, or withdraw their money. |

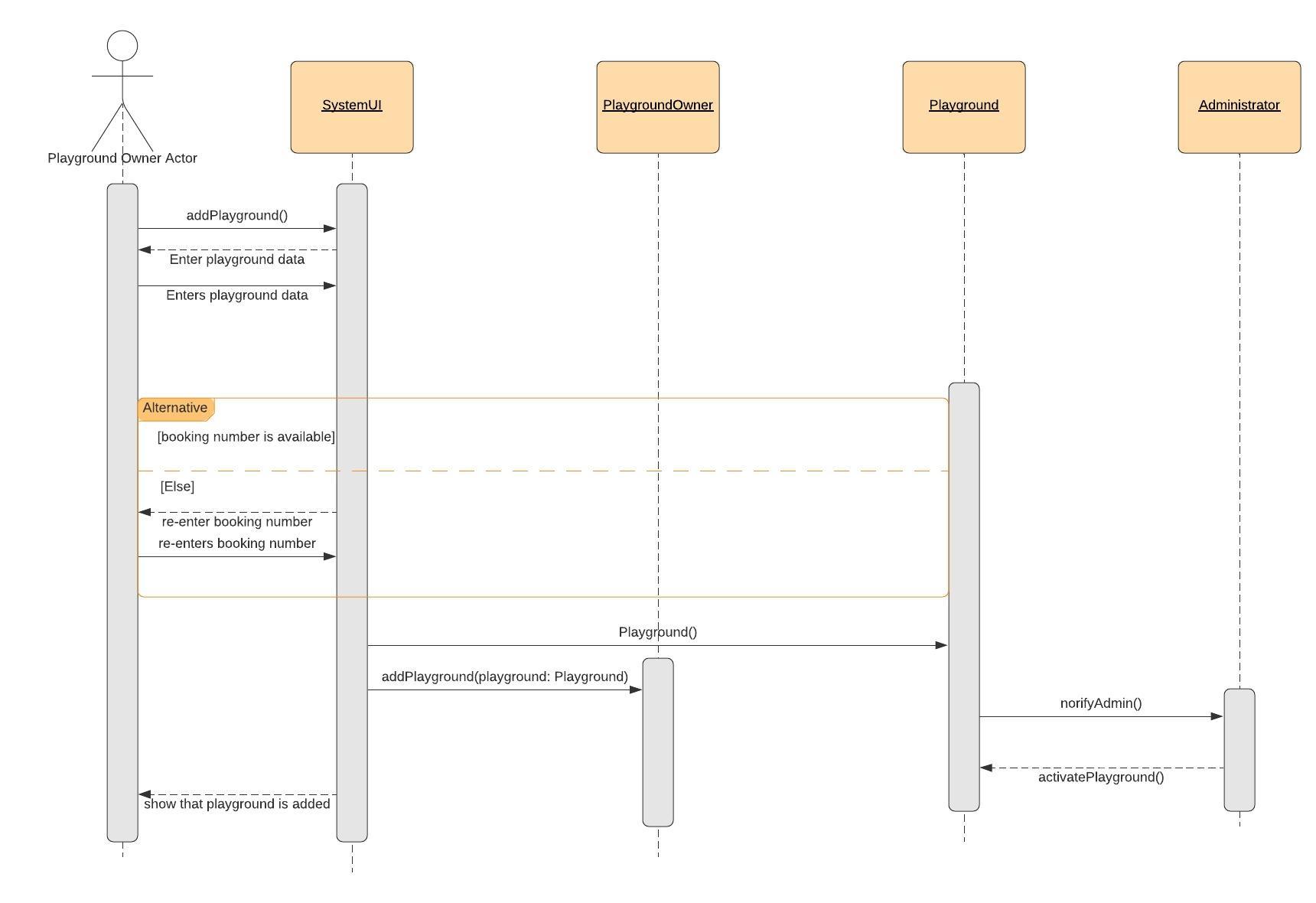
## 

## III. Sequence diagrams

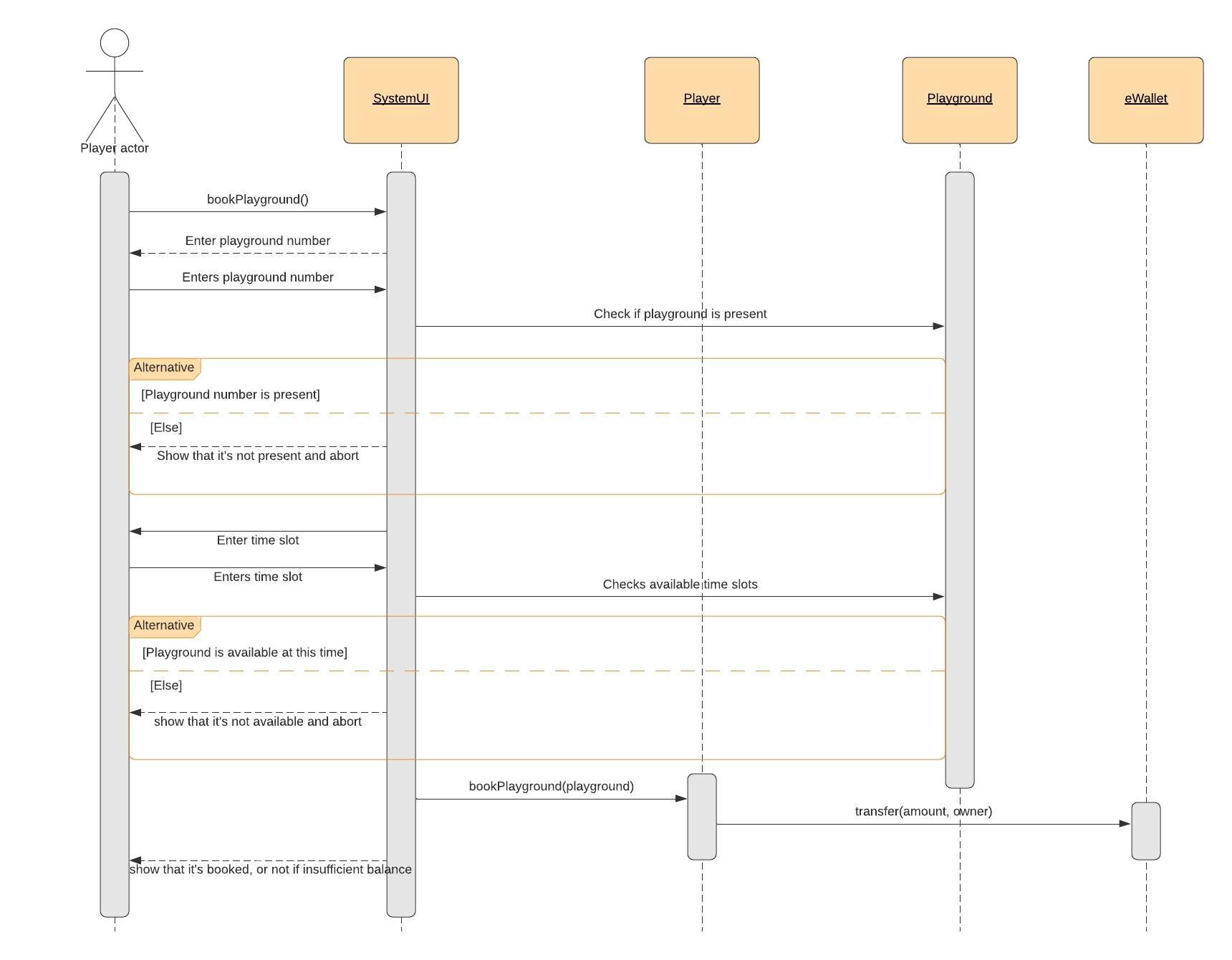
1. **Register a new user**



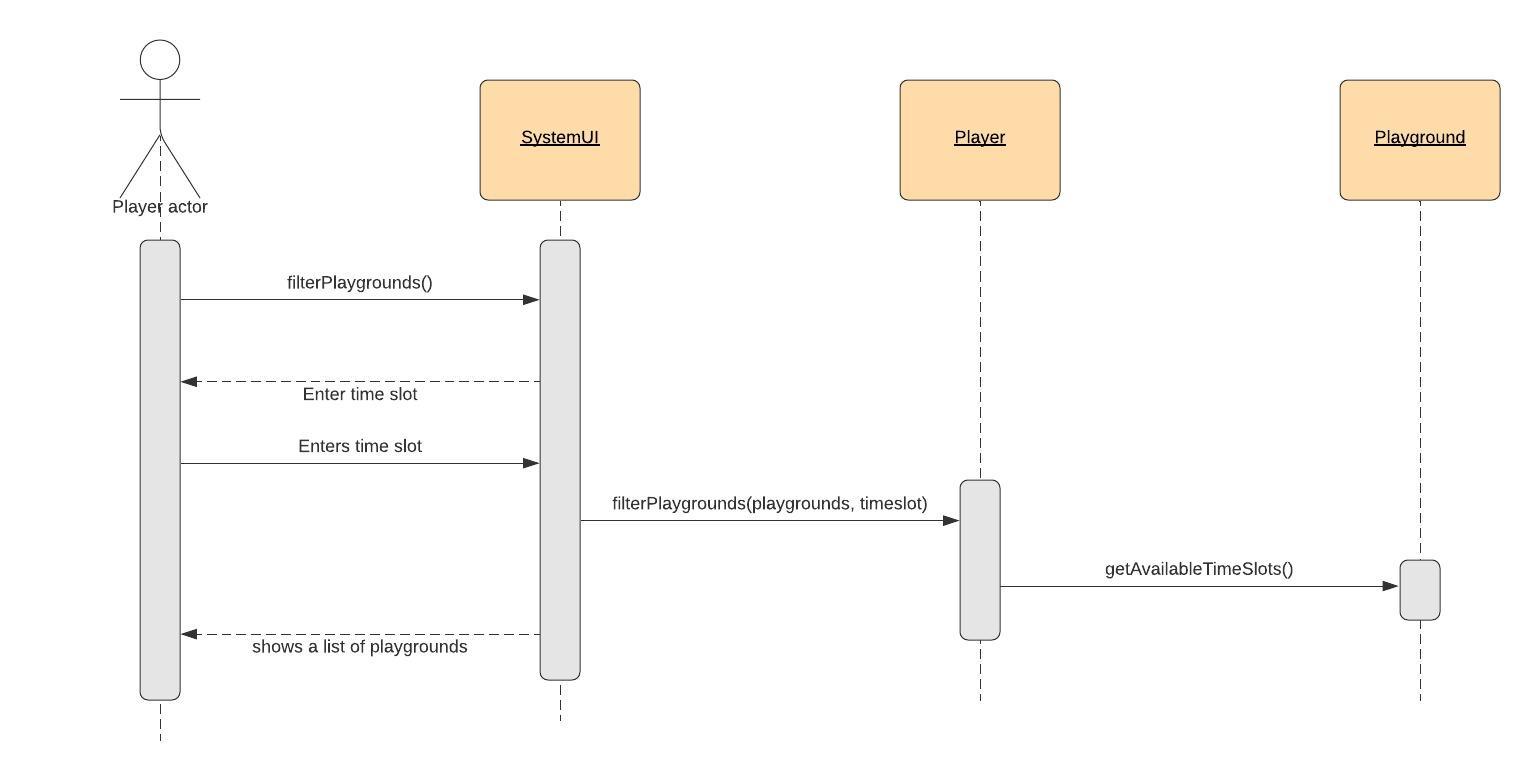
### Add a Playground



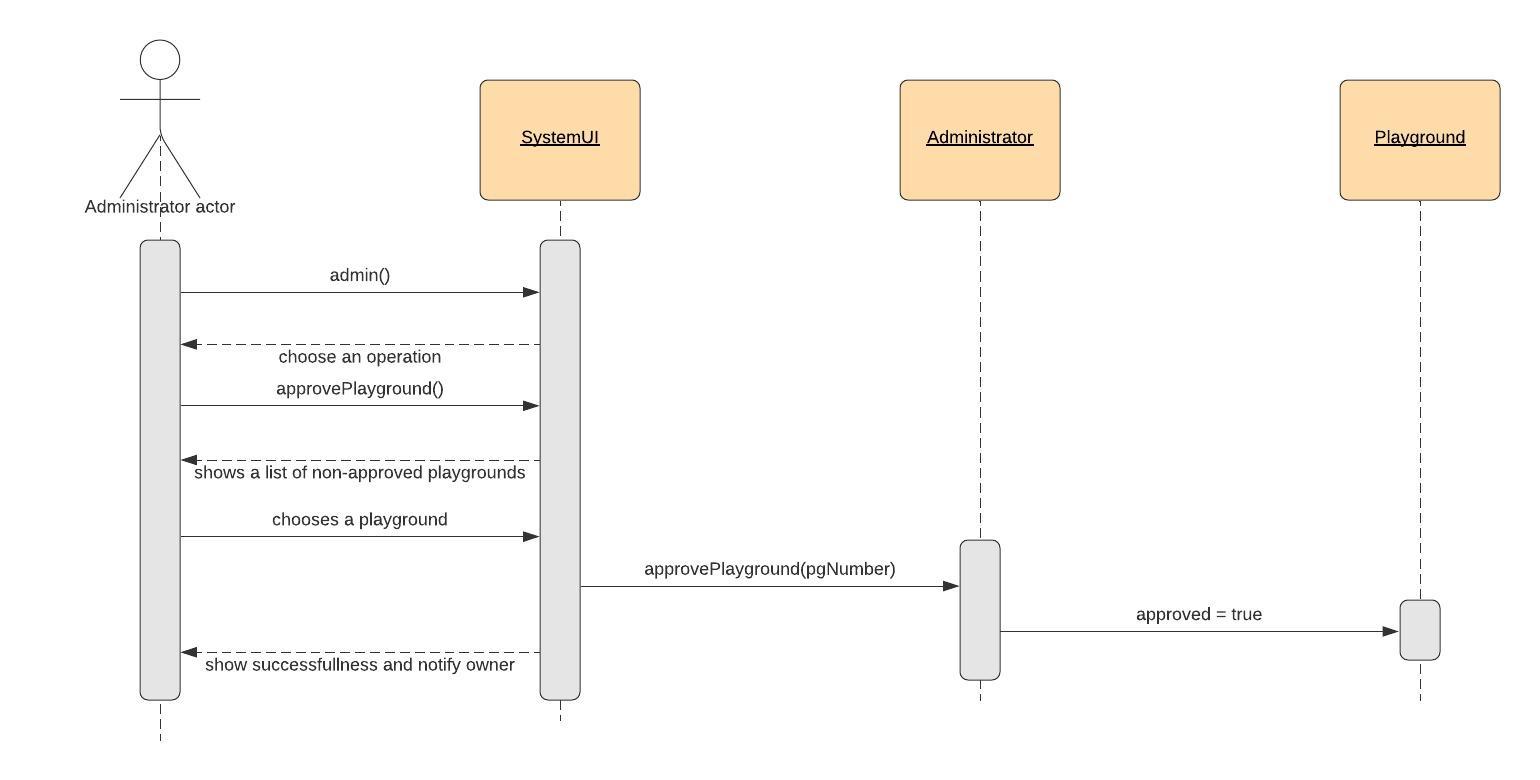
### Book a Playground



### Filter Playgrounds using time slots



### Approve a playground by an Administrator



### Create a TeamCreate a Team

### Class - Sequence Usage Table

| **Sequence Diagram** | **Classes Used** | **All Methods Used** |
| --- | --- | --- |
| Register a new user | Class UI  Class User | signUp()  checkUsername()  checkEmail()  checkStrongPassword()  User() |
| Add a Playground | Class PlaygroundOwnerUI  Class PlaygroundOwner  Class Playground  Class Administrator | addPlayground()  Playground()  addPlayground(playground)  notifyAdmin()  activatePlayground() |
| Book a Playground | Class PlayerUI  Class Player  Class Playground  Class eWallet | bookPlayground()  bookPlayground(playground, timeslot)  transfer(amount, owner) |
| Filter Playgrounds by timeslots | Class PlayerUI  Class Player  Class Playground | filterPlaygrounds()  filterPlaygrounds(playgrounds, timeslots)  getAvailableTimeSlots() |
| Approve a Playground by Administrator | Class AdminUI  Class Administrator  Class Playground | Admin()  approvePlayground()  approvePlayground(pgNumber) |
| Create a Team | Class PlayerUI  Class Player  Class Team | createTeam()  createTeam(usernames)  addPlayer(player) |

## 

## IV. User Interface Design

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Screen ID** | **Screen Name** | **Screen / Wireframe Description** |
| 1 | Application View | The Main menu of the application |
| 2 | Sign Up | If the user chooses to sign up from the main menu, this view appears allowing the user to sign up by entering his email and password |
| 2” | Verification | If the email and password are valid, this view appears informing the user that a verification code has been sent to his email and that he should enter it |
| 3 | Personal Information | If the verification code matches, this view occurs allowing the user to fill in his personal information and chooses if his account will be an owner’s account or a player’s account |
| 4 | Owner View | If the owner chooses to be an owner and signs in, this view occurs allowing the user to choose between several options which are adding a playground, viewing current booked playgrounds, and accessing his eWallet |
| 5 | Add Playground | If the owner chooses to add a playground, this view occurs and allows the owner to fill his playground’s specifications |
| 6 | Add Playground Continued | This is a continuation of the previous view where the owner fills in his playground’s available time slots with dates and time, he could add several available time slots |
| 7 | Player View | If the user chooses his account to be a player’s account and signs in this view occurs allowing the user to choose between 11 different options |
| 8 | Player View Continued | A continuation of the Player’s main menu view. |
| 9 | Filter Playgrounds | If the player chooses the filter playgrounds using time slots, this view occurs allowing the user to choose the available playground that is available in his desired time and date |
| 10 | Booking a playground | If the player chooses to book a playground, this view occurs allowing the player to book by entering the number of the playground and the desired date and time |
| 11 | eWallet | The eWallet view could be accessed by both an owner and a player, this view allows the user to choose between several options including depositing and withdrawing by entering amount of money desired and viewing the current balance |
| 12 | Administration Panel | This view is restricted to Administrators only who have specific username and passwords. If the admin signs in using the correct username and password, this view occurs allowing admin to choose from several options like deleting a playground, viewing all playgrounds, and registering un-registered playgrounds because playgrounds can only be available if approved by the administrator. |
|  |  |  |

# Tools

* **LucidChart : Used in the UML Class Diagram and Sequence Diagrams**
* **Just in mind: Used to create the UI mockups**

# Ownership Report

|  |  |
| --- | --- |
| **Item** | **Owners** |
| Part of Class diagram and Part of GoFo implementation | Mahmoud Mohamed Abdelazim |
| Part of GoFo implementation and Class diagram | Ahmed Alaa Eldin |
| Part of GoFo implementation and part of sequence diagrams | Amr Bumadian |
| UI designs and part of GoFo implementation | Ziad Amr |
| UI designs and part of sequence diagrams | Eslam Fawzy |

# References

* <http://www.mhhe.com/engcs/compsci/pressman/graphics/Pressman5sepa/common/cs1/design.pd>
* Mockups (<https://moqups.com/>).
* How to use Moqups <https://www.youtube.com/watch?v=glijkZFo4AY>
* Example wireframes and designs (you can contact the author for questions) <http://malakumar.com/wp-content/uploads/2018/12/MalaKumar_SampleWireframes-1.pdf>

# Appendix A: Code Listing and Screen Snapshots

**Code Listing:**

**package src.Users;  
import src.Utilities.Address;  
import src.Utilities.eWallet;  
  
*/\*\*  
 \* A {@code User} class is an abstract class used to make generalization for Players and Playground Owners in the system.  
 \*/*public abstract class User {  
 */\*\*  
 \* The first name of the user  
 \*/* protected String firstName;  
 */\*\*  
 \* The last name of the user  
 \*/* protected String lastName;  
 */\*\*  
 \* The email of the user  
 \*/* protected String email;  
 */\*\*  
 \* The password of the user  
 \*/* protected String password;  
 */\*\*  
 \* The username of the user, it's the identifying attribute of the {@code User} object  
 \*/* protected String username;  
 */\*\*  
 \* The mobile phone number of the user  
 \*/* protected String mobileNumber;  
 */\*\*  
 \* The address of the user  
 \*/* protected Address address;  
 */\*\*  
 \* the eWallet of the user, used to make money transactions between different users  
 \*/* protected eWallet ewallet;  
  
 */\*\*  
 \* A constructor for the {@code User} object  
 \* @param fn first name  
 \* @param ln last name  
 \* @param em email  
 \* @param pass password  
 \* @param un username  
 \* @param mn mobile number  
 \* @param ad address  
 \*/* public User(String fn, String ln, String em, String pass, String un, String mn, Address ad) {  
 firstName = fn;  
 lastName = ln;  
 email = em;  
 password = pass;  
 username = un;  
 mobileNumber = mn;  
 address = ad;  
 ewallet = new eWallet();  
 }  
  
 */\*\*  
 \* Prints the current balance of the user's eWallet  
 \*/* public void checkEWallet() {  
 System.*out*.println("Your Balance is: " + ewallet.getBalance());  
 }  
  
 */\*\*  
 \* Gets the first name of the user  
 \* @return the first name  
 \*/* public String getFirstName() {  
 return firstName;  
 }  
  
 */\*\*  
 \* Sets the first name of the user  
 \* @param firstName the new first name  
 \*/* public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 */\*\*  
 \* Gets the last name of the uesr  
 \* @return the last name  
 \*/* public String getLastName() {  
 return lastName;  
 }  
  
 */\*\*  
 \* Sets the last name of the user  
 \* @param lastName the new last name  
 \*/* public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 */\*\*  
 \* Gets the email of the user  
 \* @return the email  
 \*/* public String getEmail() {  
 return email;  
 }  
  
 */\*\*  
 \* Sets the email of the user  
 \* @param email the new email  
 \*/* public void setEmail(String email) {  
 this.email = email;  
 }  
  
 */\*\*  
 \* Gets the password of the user  
 \* @return the password  
 \*/* public String getPassword() {  
 return password;  
 }  
  
 */\*\*  
 \* Sets the password of the user  
 \* @param password the new password  
 \*/* public void setPassword(String password) {  
 this.password = password;  
 }  
  
 */\*\*  
 \* Gets the username of the user  
 \* @return the username  
 \*/* public String getUsername() {  
 return username;  
 }  
  
 */\*\*  
 \* Sets the username of the user  
 \* @param username the new username  
 \*/* public void setUsername(String username) {  
 this.username = username;  
 }  
  
 */\*\*  
 \* Gets the mobile number of the user  
 \* @return the mobile number  
 \*/* public String getMobileNumber() {  
 return mobileNumber;  
 }  
  
 */\*\*  
 \* Sets the mobile number of the user  
 \* @param mobileNumber the new mobile number  
 \*/* public void setMobileNumber(String mobileNumber) {  
 this.mobileNumber = mobileNumber;  
 }  
  
 */\*\*  
 \* Gets the address of the user  
 \* @return the address  
 \*/* public Address getAddress() {  
 return address;  
 }  
  
 */\*\*  
 \* Sets the address of the user  
 \* @param address the new address  
 \*/* public void setAddress(Address address) {  
 this.address = address;  
 }  
  
 */\*\*  
 \* Gets the eWallet of the user  
 \* @return the eWallet  
 \*/* public eWallet getEwallet() {  
 return ewallet;  
 }  
}**

**package src.Users;  
import src.Users.User;  
import src.Utilities.Address;  
import src.Utilities.Playground;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* A {@code PlaygroundOwner} class is an extension of the {@code User} class, adding the properties and methods of the owner  
 \*/*public class PlaygroundOwner extends User {  
 */\*\*  
 \* A list of playgrounds that the owner owns  
 \*/* public ArrayList<Playground> playgrounds;  
  
 */\*\*  
 \* A constructor for the {@code PlaygroundOwner} object  
 \* @param fn first name of the owner  
 \* @param ln last name of the owner  
 \* @param em email of the owner  
 \* @param pass password of the owner  
 \* @param un username of the owner  
 \* @param mn mobile number of the owner  
 \* @param ad address of the owner  
 \*/* public PlaygroundOwner(String fn, String ln, String em, String pass, String un, String mn, Address ad) {  
 super(fn, ln, em, pass, un, mn, ad);  
 playgrounds = new ArrayList<>();  
 }  
  
 */\*\*  
 \* Adds a new playground to the owner's list  
 \* @param temp the playground to be added  
 \*/* public void addPlayground(Playground temp) {  
 playgrounds.add(temp);  
 }  
  
 */\*\*  
 \* Updates a playground's data from the owner's list of playgrounds  
 \* @param prev the previous playground data  
 \* @param neww the new playground data  
 \* @return whether the playground is successfully updated  
 \*/* public boolean updatePlayground(Playground prev, Playground neww) {  
 for (int i = 0; i < playgrounds.size(); i++)  
 if (playgrounds.get(i) == prev) {  
 playgrounds.set(i, neww);  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \* Gets the list of playgrounds the owner owns  
 \* @return the list of playgrounds  
 \*/* public ArrayList<Playground> getPlaygrounds() {  
 return playgrounds;  
 }  
  
 */\*\*  
 \* Gets the data of all the owner's playgrounds as a String  
 \* @return a String containing the details of the playgrounds  
 \*/* public String getBookings() {  
 String res = "";  
 for (int i = 0; i < playgrounds.size(); i++) {  
 res += playgrounds.get(i).toString();  
 }  
 return res;  
 }  
  
 public void showBookings()  
 {  
 if (playgrounds.size()==0)  
 {  
 System.*out*.println("There is no Bookings now");  
 return;  
 }  
 for (int i = 0 ; i<playgrounds.size();i++)  
 System.*out*.println(playgrounds.get(i).toString());  
 }  
  
}**

**package src.Users;  
  
import src.Utilities.Playground;  
import src.Utilities.TimeSlot;  
import src.Utilities.Address;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* A {@code Player} class is an extension of the {@code User} class, adding properties and methods of the player  
 \*/*public class Player extends User {  
 */\*\*  
 \* the team that the player owns, if any  
 \*/* private Team teamOwned;  
 */\*\*  
 \* a list of teams the player is enrolled in  
 \*/* private ArrayList<Team> teamsEnrolled;  
  
 */\*\*  
 \* A constructor for the {@code Player} object  
 \* @param fn first name of the player  
 \* @param ln last name of the player  
 \* @param em email of the player  
 \* @param pass password of the player  
 \* @param un username of the player  
 \* @param mn mobile number of the plyer  
 \* @param ad address of the player  
 \*/* public Player(String fn, String ln, String em, String pass, String un, String mn, Address ad) {  
 super(fn, ln, em, pass, un, mn, ad);  
 teamsEnrolled = new ArrayList<>();  
 teamOwned = null;  
 }  
  
 */\*\*  
 \* Books a playground for the player at a specific time slot  
 \* @param playground the playground to be booked  
 \* @param timeSlot the time slot to book the playground at  
 \* @return whether successful of not  
 \*/* public boolean bookPlayground(Playground playground, TimeSlot timeSlot) {  
 if (playground.isActivated() && playground.bookPlayground(timeSlot, this) ) {  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \* Creates a new team owned by the player  
 \* @param name the name of the team  
 \* @param players a list of players to be in the team  
 \*/* public void createTeam(String name, ArrayList<Player> players) {  
 teamOwned = new Team(name, username);  
 for (Player player : players) {  
 teamOwned.addPlayer(player);  
 sendInvitations(player.getEmail());  
 }  
 }  
  
 */\*\*  
 \* Modifies the team that the player owns  
 \* @param newName the new name of the team  
 \* @param players the new list of players in the team  
 \*/* public void modifyTeam(String newName, ArrayList<Player> players) {  
 if (teamOwned == null) return;  
 teamOwned.setTeamName(newName);  
 teamOwned.getPlayers().clear();  
 for (Player player : players) {  
 teamOwned.addPlayer(player);  
 sendInvitations(player.getEmail());  
 }  
 }  
  
 */\*\*  
 \* Removes the player from the team specified  
 \* @param name the name of the team  
 \* @return whether successful or not  
 \*/* public boolean leaveTeam(String name) {  
 for (int i = 0; i < teamsEnrolled.size(); i++) {  
 if (teamsEnrolled.get(i).getTeamName().equalsIgnoreCase(name)) {  
 teamsEnrolled.get(i).deletePlayer(this);  
 return true;  
 }  
 }  
 if (teamOwned.getTeamName().equalsIgnoreCase(name)) {  
 teamOwned.clearTeam();  
 teamOwned = null;  
 return true;  
 }  
 return false;  
 }  
  
 */\*\*  
 \* Reports a playground  
 \* @param playground the playground to be reported  
 \*/* public void reportPlayground(Playground playground) {  
 if (playground.isActivated()) {  
 playground.reportPlayground();  
 }  
 }  
  
 */\*\*  
 \* Gets the team owned by the player  
 \* @return the team owned by the player  
 \*/* public Team getTeamOwned() {  
 return teamOwned;  
 }  
  
 */\*\*  
 \* Gets the teams the player is enrolled in  
 \* @return the list of teams the player is enrolled in  
 \*/* public ArrayList<Team> getTeamsEnrolled() {  
 return teamsEnrolled;  
 }  
  
 */\*\*  
 \* Sends email invitation to a specific email  
 \* @param email the email to send the invitation to  
 \*/* public void sendInvitations(String email) {  
 System.*out*.println("Invitation sent to " + email);  
 }  
}**

**package src.Users;  
import java.util.ArrayList;  
  
*/\*\*  
 \* A {@code Team} class is used to group players together into a team with an owner and team members  
 \*/*public class Team {  
 */\*\*  
 \* the name of the team  
 \*/* private String teamName;  
 */\*\*  
 \* the username of the {@code Player} owner  
 \*/* private String ownerUsername;  
 */\*\*  
 \* the team member players of the team  
 \*/* private ArrayList<Player> players;  
 private static int *cnt* = 0;  
  
 */\*\*  
 \* A constructor for the {@code Team} object  
 \* @param name name of the team  
 \* @param username username of the owner of the team  
 \*/* public Team(String name, String username) {  
 teamName = name;  
 ownerUsername = username;  
 players=new ArrayList<>();  
 }  
  
 */\*\*  
 \* Adds a player to the team  
 \* @param player the player to be added to the team  
 \*/* public void addPlayer(Player player) {  
 players.add(player);  
 player.getTeamsEnrolled().add(this);  
 }  
  
 */\*\*  
 \* Deletes a player from the team  
 \* @param player the player to be deleted from the team  
 \*/* public void deletePlayer(Player player) {  
 players.remove(player);  
 player.getTeamsEnrolled().remove(this);  
 }  
  
 */\*\*  
 \* Clears the team, making it empty with no players  
 \*/* public void clearTeam() {  
 for (int i = 0; i < players.size(); i++) {  
 deletePlayer(players.get(0));  
 }  
 }  
  
 @Override  
 public String toString() {  
 String res =  
 "Team name: " + teamName  
 + "\nUsername of the Owner Player: " + ownerUsername  
 + "\nOther Players: \n";  
 for (Player player: players) {  
 res += "- ";  
 res += player.getUsername() + "\n";  
 }  
 return res;  
 }  
  
 */\*\*  
 \* Sends invitation emails to the team members  
 \*/* public void sendInvitations() {  
 for (Player player: players) {  
 if (!player.getUsername().equalsIgnoreCase(ownerUsername)) {  
 System.*out*.println("Invitation sent to: " + player.getEmail());  
 }  
 }  
 }  
  
 */\*\*  
 \* Gets the name of the team  
 \* @return the name of the team  
 \*/* public String getTeamName() {  
 return teamName;  
 }  
  
 */\*\*  
 \* Sets the name of the team  
 \* @param teamName the new team name  
 \*/* public void setTeamName(String teamName) {  
 this.teamName = teamName;  
 }  
  
 */\*\*  
 \* Gets the username of the owner of the team  
 \* @return the username of the owner  
 \*/* public String getOwnerUsername() {  
 return ownerUsername;  
 }  
  
 */\*\*  
 \* Gets a list of players that are in the team  
 \* @return the list of players in the team  
 \*/* public ArrayList<Player> getPlayers() {  
 return players;  
 }  
}**

**package src.Users;  
  
import src.Utilities.Playground;  
  
*/\*\*  
 \* An {@code Administrator} class is used to allow administrators to access playgrounds and activate or delete any of them  
 \*/*public class Administrator {  
 */\*\*  
 \* the username of the admin  
 \*/* private String username;  
 */\*\*  
 \* the password of the admin  
 \*/* private String password;  
  
 */\*\*  
 \* A constructor for the {@code Administrator} object  
 \* @param username the username of the admin  
 \* @param password the password of the admin  
 \*/* public Administrator(String username, String password) {  
 this.username = username;  
 this.password = password;  
 }  
  
 */\*\*  
 \* Gets the username of the admin  
 \* @return the username of the admin  
 \*/* public String getUsername() {  
 return username;  
 }  
  
 */\*\*  
 \* Gets the password of the admin  
 \* @return the password of the admin  
 \*/* public String getPassword() {  
 return password;  
 }  
  
 */\*\*  
 \* Activates a specific playground  
 \* @param playground the playground to be activated  
 \*/* public void activatePlayground(Playground playground) {  
 playground.setActivated(true);  
 }  
}**

**package src.Utilities;  
  
import src.Users.PlaygroundOwner;  
import src.Users.Player;  
  
import java.util.ArrayList;  
  
*/\*\*  
 \* A {@code Playground} class is used to store playgrounds' data and the operations to be done on them  
 \*/*public class Playground {  
 */\*\*  
 \* the owner of the playground  
 \*/* private PlaygroundOwner owner;  
 */\*\*  
 \* the name of the playground  
 \*/* private String playgroundName;  
 */\*\*  
 \* the description of the playground  
 \*/* private String description;  
 */\*\*  
 \* the address of the playground  
 \*/* private Address address;  
 */\*\*  
 \* the available time slots of the playground  
 \*/* private ArrayList<TimeSlot> availability;  
 */\*\*  
 \* the booking number (ID) of the playground  
 \*/* private int bookingNumber;  
 */\*\*  
 \* the price per hour of the playground  
 \*/* private double pricePerHour;  
 */\*\*  
 \* the website link of the playground  
 \*/* private String link;  
 */\*\*  
 \* the number of reports made to the playground  
 \*/* private int reports;  
 */\*\*  
 \* stores whether the playground is activated or not  
 \*/* private boolean activated;  
 private static int *cnt* = 0;  
  
 */\*\*  
 \* A default constructor for the {@code Playground} object  
 \* @param owner the playground owner of the playground  
 \*/* public Playground(PlaygroundOwner owner) {  
 this.owner = owner;  
 playgroundName = "playground";  
 description = "";  
 pricePerHour = 0;  
 bookingNumber = ++*cnt*;  
 link = "";  
 availability = new ArrayList<>();  
 activated = false;  
 reports = 0;  
 }  
  
 */\*\*  
 \* Books the playground at a specific time slot  
 \* @param timeSlot the time slot to book the playground in  
 \* @param player the player that wants to book the playground  
 \* @return whether the playground is successfully booked or not  
 \*/* public boolean bookPlayground(TimeSlot timeSlot, Player player) {  
 if (!this.isActivated()) {  
 return false;  
 }  
 double totalPrice;  
 if (timeSlot.getStartingHour() < timeSlot.getEndingHour())  
 totalPrice = pricePerHour \* (timeSlot.getEndingHour() - timeSlot.getStartingHour());  
 else  
 totalPrice = pricePerHour \* ((timeSlot.getEndingHour() + 24) - timeSlot.getStartingHour());  
 if (player.getEwallet().getBalance() <= totalPrice) {  
 return false;  
 }  
 for (int i = 0; i < availability.size(); i++)  
 if (availability.get(i).equals(timeSlot)) {  
 if (availability.get(i).isBooked())  
 return false;  
 else {  
 availability.get(i).book(player.getUsername());  
 player.getEwallet().transfer(totalPrice, owner);  
 return true;  
 }  
 }  
 return false;  
 }  
  
 */\*\*  
 \* Sets the available time slots of the playground  
 \* @param ts an ArrayList of {@code TimeSlot}s  
 \*/* public void setAvailability(ArrayList<TimeSlot> ts) {  
 availability.clear();  
 availability.addAll(ts);  
 }  
  
 */\*\*  
 \* Gets the time slots of the playground  
 \* @return the time slots of the playground  
 \*/* public ArrayList<TimeSlot> getAvailability() {  
 return availability;  
 }  
  
 @Override  
 public String toString() {  
 String res =  
 "=============================================\n" +  
 "playgroundName: " + playgroundName + '\n' +  
 "description: " + description + '\n' +  
 "bookingNumber: " + bookingNumber + '\n' +  
 "pricePerHour: " + pricePerHour + '\n' +  
 "link: " + link + '\n' +  
 "activated: " + activated + '\n' +  
 "Address: " + address.toString() + '\n' +  
 "Availability: " + '\n';  
 for (int i = 0; i < availability.size(); i++) {  
 res += availability.get(i).toString();  
 res += '\n';  
 }  
 res += "\n========================================";  
 return res;  
 }  
  
 */\*\*  
 \* Reports the playground, adding one to the number of reports  
 \*/* public void reportPlayground() {  
 reports++;  
 }  
  
 */\*\*  
 \* Gets the name of the playground  
 \* @return the name of the playground  
 \*/* public String getPlaygroundName() {  
 return playgroundName;  
 }  
  
 */\*\*  
 \* Sets the name of the playground  
 \* @param playgroundName the new name of the playground  
 \*/* public void setPlaygroundName(String playgroundName) {  
 this.playgroundName = playgroundName;  
 }  
  
 */\*\*  
 \* Gets the description of the playground  
 \* @return the description of the playground  
 \*/* public String getDescription() {  
 return description;  
 }  
  
 */\*\*  
 \* Sets the description of the playground  
 \* @param description the new description of the playground  
 \*/* public void setDescription(String description) {  
 this.description = description;  
 }  
  
 */\*\*  
 \* Gets the booking number of the playground  
 \* @return the booking number of the playground  
 \*/* public int getBookingNumber() {  
 return bookingNumber;  
 }  
  
 */\*\*  
 \* Gets the price per hour of the playground  
 \* @return the price per hour of the playground  
 \*/* public double getPricePerHour() {  
 return pricePerHour;  
 }  
  
 */\*\*  
 \* Sets the price per hour of the playground  
 \* @param pricePerHour the new price per hour  
 \*/* public void setPricePerHour(double pricePerHour) {  
 this.pricePerHour = pricePerHour;  
 }  
  
 */\*\*  
 \* Gets the link of the website of the playground  
 \* @return link of the playground  
 \*/* public String getLink() {  
 return link;  
 }  
  
 */\*\*  
 \* Sets the link of the website of the playground  
 \* @param link the new link of the playground  
 \*/* public void setLink(String link) {  
 this.link = link;  
 }  
  
 */\*\*  
 \* Checks whether the playground is activated or not  
 \* @return activated or not  
 \*/* public boolean isActivated() {  
 return activated;  
 }  
  
 */\*\*  
 \* Sets the activated attribute of the playground  
 \* @param activated the new activated value  
 \*/* public void setActivated(boolean activated) {  
 this.activated = activated;  
 }  
  
 */\*\*  
 \* Gets the address of the playground  
 \* @return the address of the playground  
 \*/* public Address getAddress() {  
 return address;  
 }  
  
 */\*\*  
 \* Sets the address of the playground  
 \* @param add the new address of the playground  
 \*/* public void setAddress(Address add) {  
 address = add;  
 }  
  
 */\*\*  
 \* Gets the number of reports made to the playground  
 \* @return the number of reports made to the playground  
 \*/* public int getReports() {  
 return reports;  
 }  
}**

**package src.Utilities;  
  
*/\*\*  
 \* A {@code TimeSlot} class is used to store a date and a starting hour and and ending hour to a specific time slot  
 \*/*public class TimeSlot {  
 */\*\*  
 \* a day of the month  
 \*/* private int day;  
 */\*\*  
 \* a month of the year  
 \*/* private int month;  
 */\*\*  
 \* the year  
 \*/* private int year;  
 */\*\*  
 \* the starting hour of the time slot  
 \*/* private int startingHour;  
 */\*\*  
 \* the ending hour of the time slot  
 \*/* private int endingHour;  
 */\*\*  
 \* a boolean that specifies whether the time slot is booked or not  
 \*/* private boolean booked;  
 */\*\*  
 \* the username of the player that the time slot is booked to, if any  
 \*/* private String bookedTo;  
  
 */\*\*  
 \* A default constructor for the {@code TimeSlot} object  
 \*/* public TimeSlot() {  
 this.day = 1;  
 this.month = 1;  
 this.year = 2020;  
 this.startingHour = 0;  
 this.endingHour = 0;  
 booked = false;  
 }  
  
 */\*\*  
 \* A constructor for the {@code TimeSlot} object  
 \* @param day day of the month  
 \* @param month month of the year  
 \* @param year year of the time slot  
 \* @param startingHour starting hour of the time slot  
 \* @param endingHour ending hour of the time slot  
 \*/* public TimeSlot(int day, int month, int year, int startingHour, int endingHour) {  
 this.day = day;  
 this.month = month;  
 this.year = year;  
 this.startingHour = startingHour;  
 this.endingHour = endingHour;  
 booked = false;  
 }  
  
 */\*\*  
 \* Books the time slot to a specific player and stores his username  
 \* @param username username of the player  
 \*/* public void book(String username) {  
 booked = true;  
 bookedTo = username;  
 }  
  
 */\*\*  
 \* Unbooks the time slot  
 \*/* public void unbook() {  
 booked = false;  
 bookedTo = "";  
 }  
  
 */\*\*  
 \* Checks whether the time slot is booked  
 \* @return booked or not  
 \*/* public boolean isBooked() {  
 return booked;  
 }  
  
 @Override  
 public String toString() {  
 String res = "--------------------------------------\n" +  
 "Date: " + day + "/" + month + "/" + year  
 + "\nStarting Hour: " + startingHour  
 + "\nEnding Hour: " + endingHour;  
 if (isBooked()) {  
 res += "\nBooked: Yes";  
 res += "\nBooked to: " + bookedTo;  
 } else {  
 res += "\nBooked: No";  
 }  
 res += "\n------------------------------------";  
 return res;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 TimeSlot timeSlot = (TimeSlot) o;  
 return day == timeSlot.day &&  
 month == timeSlot.month &&  
 year == timeSlot.year &&  
 startingHour == timeSlot.startingHour &&  
 endingHour == timeSlot.endingHour;  
 }  
  
 */\*\*  
 \* Gets the day of the month  
 \* @return day of the month  
 \*/* public int getDay() {  
 return day;  
 }  
  
 */\*\*  
 \* Sets the day of the month  
 \* @param day the new day  
 \* @return whether day is valid or not  
 \*/* public boolean setDay(int day) {  
 if (day < 1 || day > 31) return false;  
 this.day = day;  
 return true;  
 }  
  
 */\*\*  
 \* Gets the month of the year  
 \* @return month of the year  
 \*/* public int getMonth() {  
 return month;  
 }  
  
 */\*\*  
 \* Sets the month of the year  
 \* @param month the new month  
 \* @return whether the month is valid or not  
 \*/* public boolean setMonth(int month) {  
 if (month < 1 || month > 12) return false;  
 this.month = month;  
 return true;  
 }  
  
 */\*\*  
 \* Gets the year of the time slot  
 \* @return year of the time slot  
 \*/* public int getYear() {  
 return year;  
 }  
  
 */\*\*  
 \* Sets the year of the time slot  
 \* @param year the new year  
 \* @return whether the year is valid or not  
 \*/* public boolean setYear(int year) {  
 if (year < 2020) return false;  
 this.year = year;  
 return true;  
 }  
  
 */\*\*  
 \* Gets the starting hour of the time slot  
 \* @return startingHour of the time slot  
 \*/* public int getStartingHour() {  
 return startingHour;  
 }  
  
 */\*\*  
 \* Sets the starting hour of the time slot  
 \* @param startingHour the new starting hour  
 \* @return whether the starting hour is valid or not  
 \*/* public boolean setStartingHour(int startingHour) {  
 if (startingHour < 0 || startingHour > 23) return false;  
 this.startingHour = startingHour;  
 return true;  
 }  
  
 */\*\*  
 \* Gets the ending hour of the time slot  
 \* @return endingHour of the time slot  
 \*/* public int getEndingHour() {  
 return endingHour;  
 }  
  
 */\*\*  
 \* Sets the ending hour of the time slot  
 \* @param endingHour the new ending hour of the time slot  
 \* @return whether the ending hour is valid or not  
 \*/* public boolean setEndingHour(int endingHour) {  
 if (endingHour < 0 || endingHour > 23) return false;  
 this.endingHour = endingHour;  
 return true;  
 }  
  
 */\*\*  
 \* Gets the username of the player the time slot is booked to  
 \* @return the username of the player the time slot is booked to  
 \*/* public String getBookedTo() {  
 return bookedTo;  
 }  
  
}**

**package src.Utilities;  
  
import src.Users.User;  
  
import java.util.Scanner;  
  
*/\*\*  
 \* A {@code eWallet} class is used to call an external service to transfer money between different users in the system  
 \*/*public class eWallet {  
 */\*\*  
 \* the current balance of the user's account  
 \*/* private double currentBalance;  
  
 */\*\*  
 \* A constructor for the {@code eWallet} object  
 \*/* public eWallet() {  
 currentBalance = 0;  
 }  
  
 */\*\*  
 \* Deposits an amount to the user's account  
 \* @param amount the amount to be deposited  
 \* @return whether successful or not  
 \*/* public boolean deposit(double amount) {  
 if (amount <= 0)  
 return false;  
 currentBalance += amount;  
 return true;  
 }  
  
 */\*\*  
 \* Withdraws an amount from the user's account  
 \* @param amount the amount to be withdrawn  
 \* @return whether successful or not  
 \*/* public boolean withdraw(double amount) {  
 if (amount > currentBalance || amount <= 0)  
 return false;  
 currentBalance -= amount;  
 return true;  
 }  
  
 */\*\*  
 \* Transfers an amount from the user's account to another user's account  
 \* @param amount the amount to be transferred  
 \* @param user the user to transfer to  
 \* @return whether successful or not  
 \*/* public boolean transfer(double amount, User user) {  
 if (amount > currentBalance || amount <= 0)  
 return false;  
 user.getEwallet().deposit(amount);  
 currentBalance -= amount;  
 return true;  
 }  
  
 */\*\*  
 \* Gets the balance of the user's account  
 \* @return the current balance  
 \*/* public double getBalance() {  
 return currentBalance;  
 }  
  
}**

**package src.Utilities;  
  
import java.util.Scanner;  
  
*/\*\*  
 \* An {@code Address} class is used to store the data of an address, dividing it into parts  
 \*/*public class Address {  
 */\*\*  
 \* the street number  
 \*/* private int streetNumber;  
 */\*\*  
 \* the street name  
 \*/* private String streetName;  
 */\*\*  
 \* the neighborhood  
 \*/* private String neighborhood;  
 */\*\*  
 \* the city  
 \*/* private String city;  
  
 */\*\*  
 \* A constructor for the {@code Address} object  
 \* @param streetNumber the street number  
 \* @param streetName the street name  
 \* @param neighborhood the neighborhood  
 \* @param city the city  
 \*/* public Address(int streetNumber, String streetName, String neighborhood, String city) {  
 this.streetNumber = streetNumber;  
 this.streetName = streetName;  
 this.neighborhood = neighborhood;  
 this.city = city;  
 }  
  
 @Override  
 public String toString() {  
 return streetNumber + " " + streetName + ", " + neighborhood + ", " + city;  
 }  
  
 */\*\*  
 \* Gets the street number of the address  
 \* @return the street number  
 \*/* public int getStreetNumber() {  
 return streetNumber;  
 }  
  
 */\*\*  
 \* Sets the street number of the address  
 \* @param streetNumber the new street number  
 \*/* public void setStreetNumber(int streetNumber) {  
 this.streetNumber = streetNumber;  
 }  
  
 */\*\*  
 \* Gets the street name of the address  
 \* @return the street name  
 \*/* public String getStreetName() {  
 return streetName;  
 }  
  
 */\*\*  
 \* Sets the street name of the address  
 \* @param streetName the new street name  
 \*/* public void setStreetName(String streetName) {  
 this.streetName = streetName;  
 }  
  
 */\*\*  
 \* Gets the neighborhood of the address  
 \* @return the neighborhood  
 \*/* public String getNeighborhood() {  
 return neighborhood;  
 }  
  
 */\*\*  
 \* Sets the neighborhood of the address  
 \* @param neighborhood the new neighborhood  
 \*/* public void setNeighborhood(String neighborhood) {  
 this.neighborhood = neighborhood;  
 }  
  
 */\*\*  
 \* Gets the city of the address  
 \* @return the city  
 \*/* public String getCity() {  
 return city;  
 }  
  
 */\*\*  
 \* Sets the city of the address  
 \* @param city the new city  
 \*/* public void setCity(String city) {  
 this.city = city;  
 }  
}**

**package src.System;  
  
import src.Users.\*;  
import src.Utilities.Address;  
import src.Utilities.Playground;  
  
import java.util.\*;  
  
*/\*\*  
 \* {@code GoFoSystem} class is used as the basis of the system, it stores all the users and admins' data and accesses the UI  
 \*/*public class GoFoSystem {  
 */\*\*  
 \* A list of all players in the system  
 \*/* private ArrayList<Player> players;  
 */\*\*  
 \* A list of all playground owners in the system  
 \*/* private ArrayList<PlaygroundOwner> owners;  
 */\*\*  
 \* A list of all playgrounds in the system  
 \*/* private ArrayList<Playground> playgrounds;  
 */\*\*  
 \* A list of admins in the system  
 \*/* private ArrayList<Administrator> admins;  
 */\*\*  
 \* A UI to enable users to interact with the system  
 \*/* private UI ui;  
  
 */\*\*  
 \* A constructor for the system, which initializes the lists and and starts the UI  
 \*/* public GoFoSystem() {  
 players = new ArrayList<>();  
 owners = new ArrayList<>();  
 playgrounds = new ArrayList<>();  
 admins = new ArrayList<>();  
 players.add(new Player("", "", "player@example.com", "player", "player", "", new Address(3, "", "", "")));  
 owners.add(new PlaygroundOwner("", "", "owner@example.com", "owner", "owner", "", new Address(3, "", "", "")));  
 admins.add(new Administrator("admin", "admin"));  
 ui = new UI(players, owners, playgrounds, admins);  
 }  
}**

**package src.System;  
  
import src.Utilities.Address;  
import src.Utilities.Playground;  
import src.Users.PlaygroundOwner;  
import src.Users.Administrator;  
import src.Users.Player;  
import java.util.\*;  
  
*/\*\*  
 \* {@code UI} class is used to enable users and admins to interact with the system using the console, and direct them to other specific UI classes  
 \*/*public class UI {  
  
 ArrayList<Player> players;  
 ArrayList<PlaygroundOwner> owners;  
 ArrayList<Playground> playgrounds;  
 ArrayList<Administrator> admins;  
 private Scanner scanner = new Scanner(System.*in*);  
  
 */\*\*  
 \* A constructor for the {@code UI} object  
 \*/* public UI(ArrayList<Player> players, ArrayList<PlaygroundOwner> owners, ArrayList<Playground> playgrounds, ArrayList<Administrator> admins) {  
 this.players = players;  
 this.owners = owners;  
 this.playgrounds = playgrounds;  
 this.admins = admins;  
 mainMenu();  
 }  
  
 */\*\*  
 \* A main menu that allows users and admins to login or register to the system, and redirects them to different parts of the system  
 \*/* public void mainMenu() {  
 String option = "";  
 while (true) {  
 System.*out*.println("\n1- Signup" +  
 "\n2- Login" +  
 "\n3- Login As an Admin" +  
 "\n4- Exit");  
 option = scanner.nextLine();  
 if (option.equalsIgnoreCase("1")) {  
 signUp();  
 } else if (option.equalsIgnoreCase("2")) {  
 login();  
 } else if (option.equalsIgnoreCase("3")) {  
 adminLogin();  
 } else if (option.equalsIgnoreCase("4")) {  
 break;  
 }  
 else {  
 System.*out*.println("\*\*\*Enter valid option\*\*\*\n");  
 }  
 }  
 System.*out*.println("\nThanks for using GoFo Booking System");  
 }  
  
 */\*\*  
 \* Registers a new user to the system by entering the appropriate data  
 \*/* private void signUp() {  
 String username, password, phone, email, firstName, lastName, streetName, neighborhood, city;  
 int strNumber;  
 Address address;  
 while (true) {  
 System.*out*.print("Enter new username: ");  
 username = scanner.nextLine();  
 boolean validUsername = checkUsername(username);  
 if (!validUsername) {  
 System.*out*.println("\*\*\*Username already taken\*\*\*\n");  
 } else {  
 break;  
 }  
 }  
 while (true) {  
 System.*out*.print("Enter new Email: ");  
 email = scanner.nextLine();  
 boolean validEmail = checkEmail(email);  
 if (!validEmail) {  
 System.*out*.println("\*\*\*Email Not available\*\*\*\n");  
 } else {  
 break;  
 }  
 }  
 while (true) {  
 System.*out*.print("Enter new password (at least 8 letters, include 1 symbol and 1 Uppercase letter): ");  
 password = scanner.nextLine();  
 boolean validPassword = checkStrongPassword(password);  
 if (!validPassword) {  
 System.*out*.println("\*\*\*Enter strong password\*\*\*\n");  
 } else {  
 break;  
 }  
 }  
 sendEmailVerificationCode();  
 System.*out*.print("Enter Phone Number: ");  
 phone = scanner.nextLine();  
 System.*out*.print("Enter First Name: ");  
 firstName = scanner.nextLine();  
 System.*out*.print("Enter Last Name: ");  
 lastName = scanner.nextLine();  
 System.*out*.println("Address Informations;- ");  
 System.*out*.print("Enter Street number: ");  
 strNumber = scanner.nextInt();  
 System.*out*.print("Enter Street name: ");  
 scanner.skip("\n");  
 streetName = scanner.nextLine();  
 System.*out*.print("Enter Neighborhood: ");  
 neighborhood = scanner.nextLine();  
 System.*out*.print("Enter City: ");  
 city = scanner.nextLine();  
 address = new Address(strNumber, streetName, neighborhood, city);  
 String type;  
 while (true) {  
 System.*out*.println("Enter the type of account: " +  
 "\n1- Player" +  
 "\n2- Playground Owner");  
 type = scanner.nextLine();  
 if (type.equalsIgnoreCase("1")) {  
 players.add(new Player(firstName, lastName, email, password, username, phone, address));  
 break;  
 } else if (type.equalsIgnoreCase("2")) {  
 owners.add(new PlaygroundOwner(firstName, lastName, email, password, username, phone, address));  
 break;  
 } else {  
 System.*out*.println("\*\*\*Enter valid option\*\*\*");  
 }  
 }  
 }  
  
 */\*\*  
 \* Logs users into the system and redirects them to their proper UI  
 \*/* private void login() {  
 String username, password;  
 boolean foundUser = false;  
 boolean correctPassword = false;  
 System.*out*.print("Enter Username: ");  
 username = scanner.nextLine();  
 System.*out*.print("Enter Password: ");  
 password = scanner.nextLine();  
 int userIdx = -1;  
 for (Player player: players) {  
 userIdx++;  
 if (player.getUsername().equalsIgnoreCase(username)) {  
 foundUser = true;  
 if (player.getPassword().equals(password)) {  
 correctPassword = true;  
 break;  
 }  
 }  
 }  
 if (foundUser && correctPassword) {  
 var playerUI = new PlayerUI(playgrounds, players, userIdx);  
 return;  
 }  
 userIdx = -1;  
 for (PlaygroundOwner owner: owners) {  
 userIdx++;  
 if (owner.getUsername().equalsIgnoreCase(username)) {  
 foundUser = true;  
 if (owner.getPassword().equals(password)) {  
 correctPassword = true;  
 break;  
 }  
 }  
 }  
 if (foundUser && correctPassword) {  
 var PlaygroundOwnerUI = new PlaygroundOwnerUI(playgrounds, owners, userIdx);  
 return;  
 }  
 if (!foundUser) {  
 System.*out*.println("Username not found");  
 } else if (!correctPassword) {  
 System.*out*.println("Incorrect Password");  
 }  
 }  
  
 */\*\*  
 \* Checks if the password is strong or not  
 \* @param password the password to be checked  
 \* @return whether the password is strong  
 \*/* private boolean checkStrongPassword(String password) {  
 boolean strong = false;  
 if (password.length() < 8) {  
 return strong;  
 }  
 int symbols = 0;  
 int uppercase = 0;  
 for (int i = 0; i < password.length(); ++i) {  
 if (Character.*isLetterOrDigit*(password.charAt(i))) {  
 if (Character.*isUpperCase*(password.charAt(i))) {  
 ++uppercase;  
 }  
 } else {  
 ++symbols;  
 }  
 }  
 return (symbols >= 1 && uppercase >= 1);  
 }  
  
 */\*\*  
 \* Checks the email is valid or not  
 \* @param email the email to be checked  
 \* @return whether the email is valid  
 \*/* private boolean checkEmail(String email) {  
 boolean validEmail;  
  
 validEmail = email.matches("(?:[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*|\"" +  
 "(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])" +  
 "\*\")@(?:(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\\.)+[a-z0-9]" +  
 "(?:[a-z0-9-]\*[a-z0-9])?|\\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.)" +  
 "{3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]\*[a-z0-9]:" +  
 "(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])+)\\])");  
 return validEmail && availableEmail(email);  
 }  
  
 */\*\*  
 \* Checks if the email is available or already registered  
 \* @param email the email to be checked  
 \* @return whether the email is available  
 \*/* private boolean availableEmail(String email) {  
 for (Player player: players) {  
 if (player.getEmail().equalsIgnoreCase(email)) {  
 return false;  
 }  
 }  
 for (PlaygroundOwner owner: owners) {  
 if (owner.getEmail().equalsIgnoreCase(email)) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 */\*\*  
 \* Checks if the username is available or already taken  
 \* @param username the username to be checked  
 \* @return whether the username is available  
 \*/* private boolean checkUsername(String username) {  
 for (Player player: players) {  
 if (player.getUsername().equalsIgnoreCase(username)) {  
 return false;  
 }  
 }  
 for (PlaygroundOwner owner: owners) {  
 if (owner.getUsername().equalsIgnoreCase(username)) {  
 return false;  
 }  
 }  
 return true;  
 }  
  
 */\*\*  
 \* Sends a verification code to the email and verifies it  
 \*/* private void sendEmailVerificationCode() {  
 System.*out*.println("Enter verification code sent to your email: ");  
 String userCode = scanner.nextLine();  
 System.*out*.println("Verified ✅");  
 }  
  
 */\*\*  
 \* Logs an admin into the system and redirects him to the Admin UI  
 \*/* private void adminLogin() {  
 String username, password;  
 System.*out*.print("Enter username: ");  
 username = scanner.nextLine();  
 System.*out*.print("Enter Password: ");  
 password = scanner.nextLine();  
 boolean found = false, correctPassword = false;  
 for (Administrator admin: admins) {  
 if (admin.getUsername().equalsIgnoreCase(username)) {  
 found = true;  
 if (admin.getPassword().equalsIgnoreCase(password)) {  
 correctPassword = true;  
 AdminUI adminUI = new AdminUI(playgrounds, admin);  
 }  
 break;  
 }  
 }  
 if (!found) System.*out*.println("Username is invalid");  
 else if (!correctPassword) System.*out*.println("Incorrect Password");  
 }  
}**

**package src.System;  
  
import src.Utilities.Address;  
import src.Utilities.Playground;  
import src.Users.PlaygroundOwner;  
import src.Utilities.TimeSlot;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
*/\*\*  
 \* {@code PlaygroundOwnerUI} class is used to enable playground owners to interact with the system using the console  
 \*/*public class PlaygroundOwnerUI{  
  
 private Scanner scanner = new Scanner(System.*in*);  
 private ArrayList<Playground> playgrounds;  
 private ArrayList<PlaygroundOwner> playgroundOwners;  
 private int currentOwner;  
  
 */\*\*  
 \* A constructor for the PlaygroundOwnerUI  
 \* @param currentOwner the index of the current playground owner logged into the system  
 \*/* public PlaygroundOwnerUI(ArrayList<Playground> playgrounds, ArrayList<PlaygroundOwner> playgroundOwners, int currentOwner) {  
 this.playgrounds = playgrounds;  
 this.playgroundOwners = playgroundOwners;  
 this.currentOwner = currentOwner;  
 mainMenu();  
 }  
  
 */\*\*  
 \* A main menu that allows owners to choose an operation to be done and redirects them to different parts of the system  
 \*/* public void mainMenu() {  
 String option = "";  
 while (true) {  
 System.*out*.println("\n1- Add a Playground" +  
 "\n2- View current Bookings" +  
 "\n3- Access eWallet" +  
 "\n4- Logout");  
 option = scanner.nextLine();  
 if (option.equalsIgnoreCase("1")) {  
 addPlayground();  
 } else if (option.equalsIgnoreCase("2")) {  
 viewBookings();  
 } else if (option.equalsIgnoreCase("3")) {  
 eWalletUI eWalletUI = new eWalletUI(playgroundOwners.get(currentOwner));  
 } else if (option.equalsIgnoreCase("4")) {  
 break;  
 }  
 else {  
 System.*out*.println("\*\*\*Enter valid option\*\*\*\n");  
 }  
 }  
 System.*out*.println("\nLogging out ...");  
 }  
  
 */\*\*  
 \* Allows an owner to enter a new playground's data and to be added to the system and storing it in their account  
 \*/* private void addPlayground() {  
 Playground playground = new Playground(playgroundOwners.get(currentOwner));  
  
 System.out.print("Enter Playground Name: ");  
 String playgroundName = scanner.nextLine();  
 playground.setPlaygroundName(playgroundName);  
  
 System.out.println("Enter Playground description: ");  
 String description = scanner.nextLine();  
 playground.setDescription(description);  
  
 System.out.println("Enter Address: ");  
 System.out.print("Enter Street number: ");  
 int streetNumber = scanner.nextInt();  
 System.out.print("Enter Street name: ");  
 scanner.skip("\n");  
 String streetName = scanner.nextLine();  
 System.out.print("Enter Neighborhood: ");  
 String neighborhood = scanner.nextLine();  
 System.out.print("Enter City: ");  
 String city = scanner.nextLine();  
  
 Address address = new Address(streetNumber, streetName, neighborhood, city);  
 playground.setAddress(address);  
  
 System.out.println("Enter Link: ");  
 String link = scanner.nextLine();  
 playground.setLink(link);  
  
 System.out.println("Enter pricePerHour: ");  
 double pricePerHour = scanner.nextDouble();  
 playground.setPricePerHour(pricePerHour);  
  
 var timeSlots = new ArrayList<TimeSlot>();  
 TimeSlot timeSlot;  
 while (true) {  
 System.out.println("Enter available timeslot: ");  
 timeSlot = setTimeslot();  
 timeSlots.add(timeSlot);  
 System.out.println("Add another timeslot? Enter 'Y' for yes or any key to exit");  
 String option = scanner.nextLine();  
 if (!option.equalsIgnoreCase("Y")) {  
 break;  
 }  
 }  
 playground.setAvailability(timeSlots);  
 playgroundOwners.get(currentOwner).addPlayground(playground);  
 playgrounds.add(playgroundOwners.get(currentOwner).getPlaygrounds()  
 .get(playgroundOwners.get(currentOwner).getPlaygrounds().size() - 1));  
 System.out.println("\nPlayground added successfully ✅, waiting for approval by an administrator.");  
 }  
  
 */\*\*  
 \* Outputs the list of playgrounds the owner owns and their details  
 \*/* private void viewBookings() {  
 playgroundOwners.get(currentOwner).showBookings();  
 }  
  
 */\*\*  
 \* Allows a user to enter the data of a time slot  
 \* @return the time slot entered by the user  
 \*/* private TimeSlot setTimeslot() {  
 int day, month, year, startingHour, endingHour;  
 System.out.println("Enter timeslot details: ");  
 while (true) {  
 System.out.print("\tEnter day: ");  
 day = scanner.nextInt();  
 if (day >= 1 && day <= 31) {  
 break;  
 } else {  
 System.out.println("\*\*\*Enter valid day\*\*\*");  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter month: ");  
 month = scanner.nextInt();  
 if (month >= 1 && month <= 12) {  
 break;  
 } else {  
 System.out.println("\*\*\*Enter valid month\*\*\*");  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter year: ");  
 LocalDate date = LocalDate.now();  
 year = scanner.nextInt();  
 if (year < date.getYear()) {  
 System.out.println("\*\*\*Enter valid year\*\*\*");  
 } else {  
 break;  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter startingHour: ");  
 startingHour = scanner.nextInt();  
 if (startingHour < 0 || startingHour > 23) {  
 System.out.println("\*\*\*Enter valid hour\*\*\*");  
 } else {  
 break;  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter endingHour: ");  
 endingHour = scanner.nextInt();  
 if (endingHour < 0 || endingHour > 23) {  
 System.out.println("\*\*\*Enter valid hour\*\*\*");  
 } else {  
 break;  
 }  
 }  
 scanner.skip("\n");  
 return new TimeSlot(day, month, year, startingHour, endingHour);  
 }  
}**

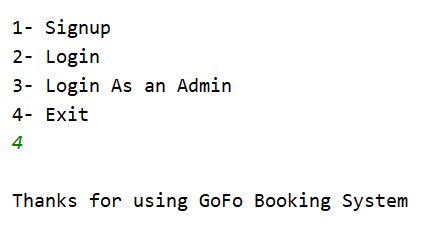
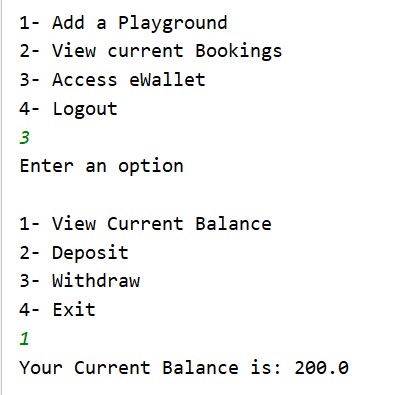
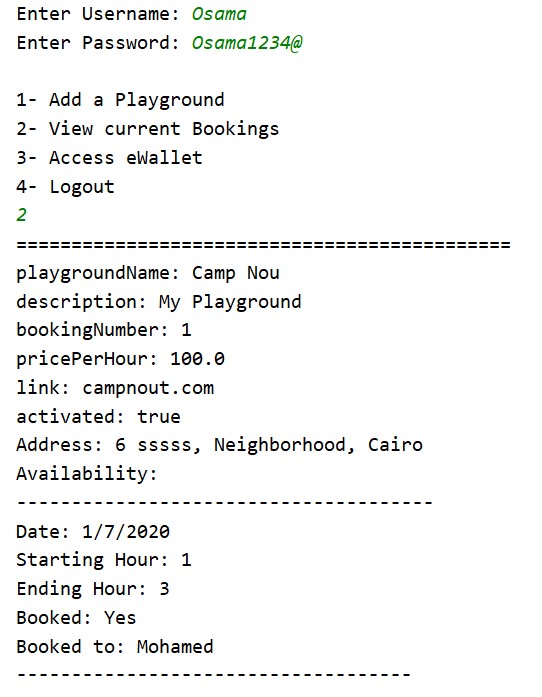
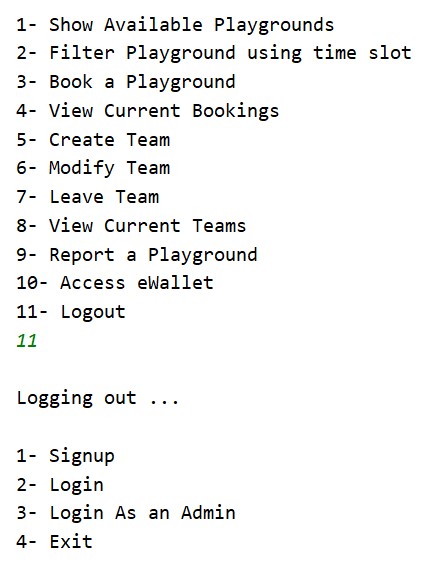
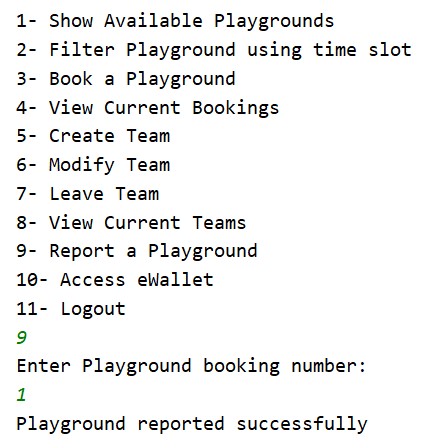
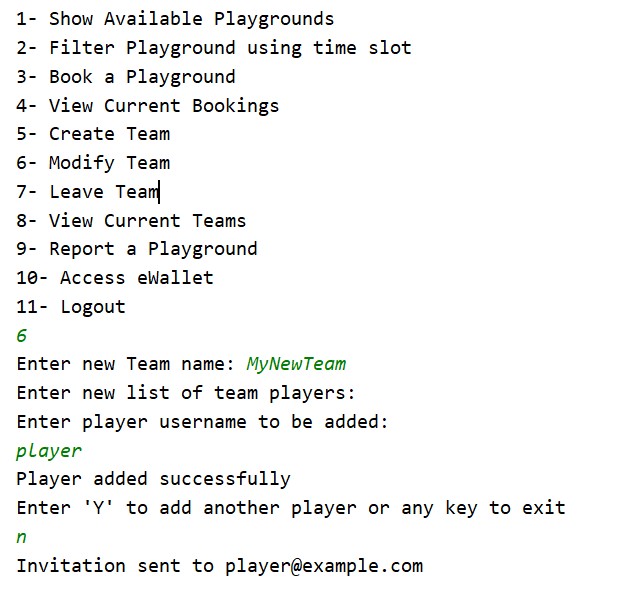
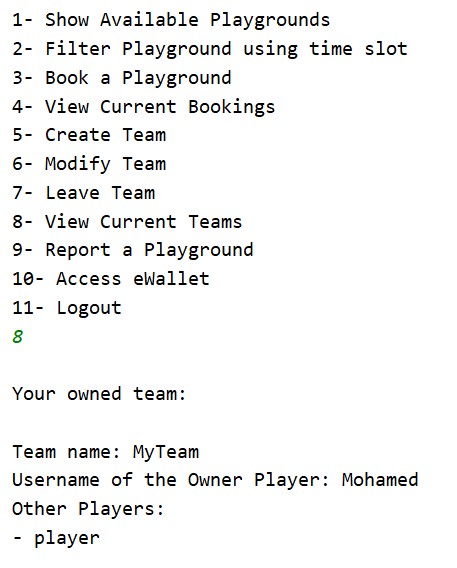
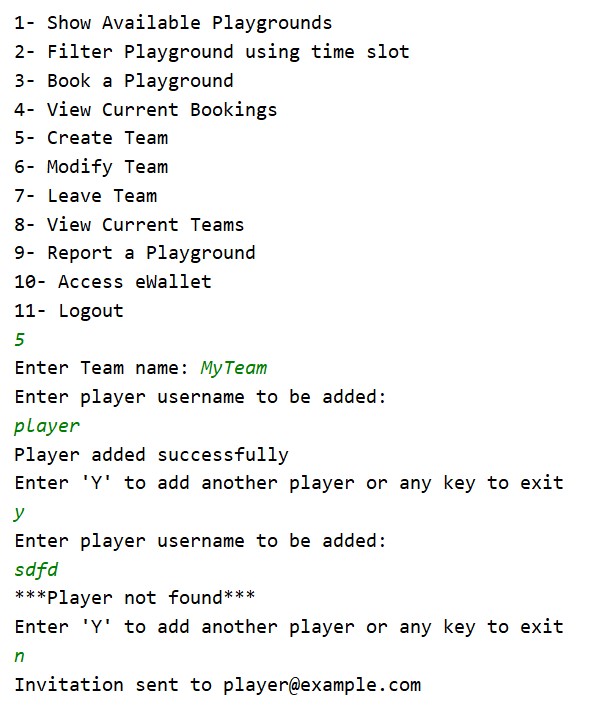
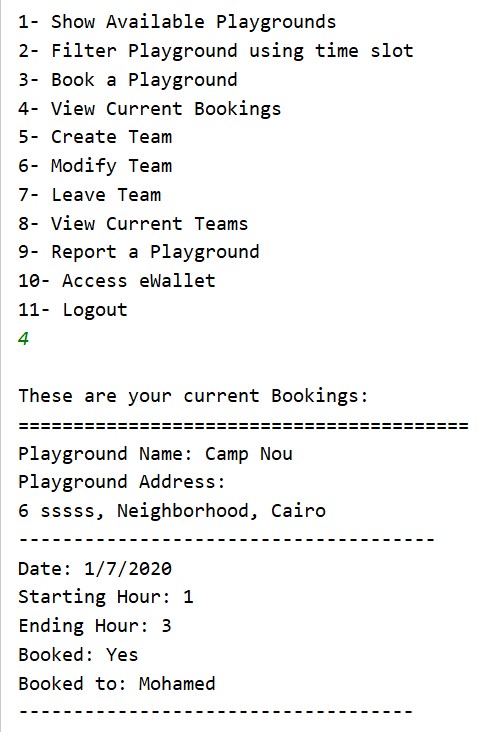
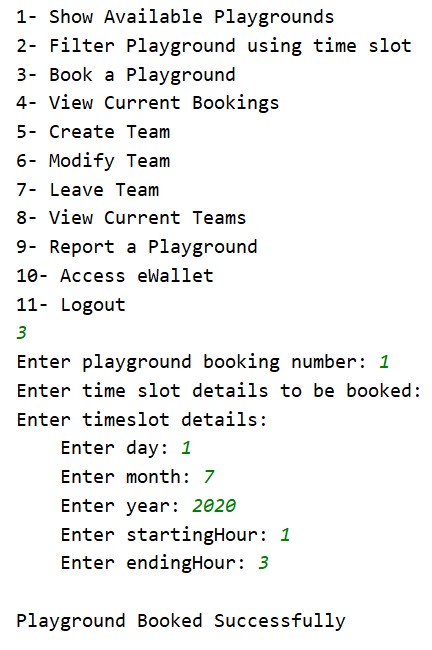
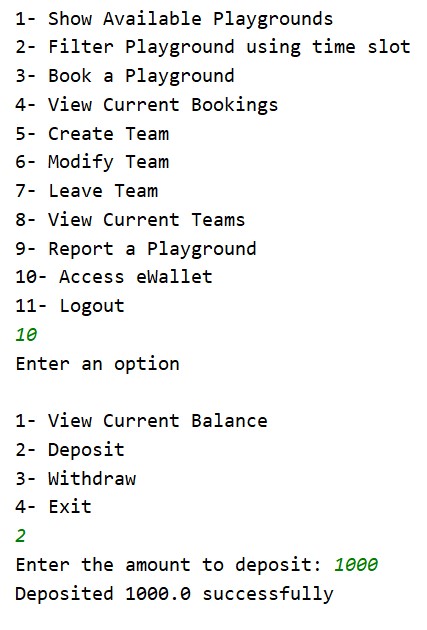
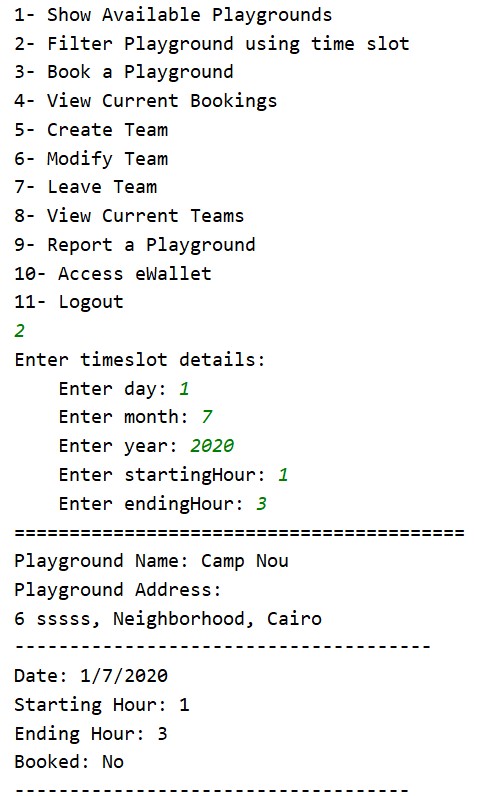
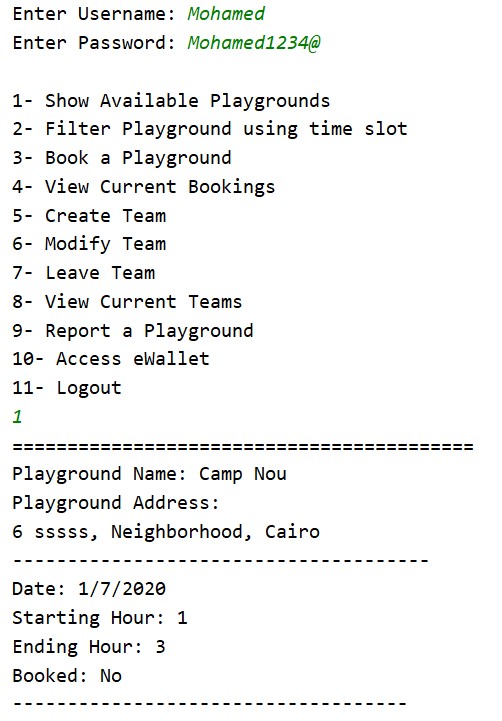
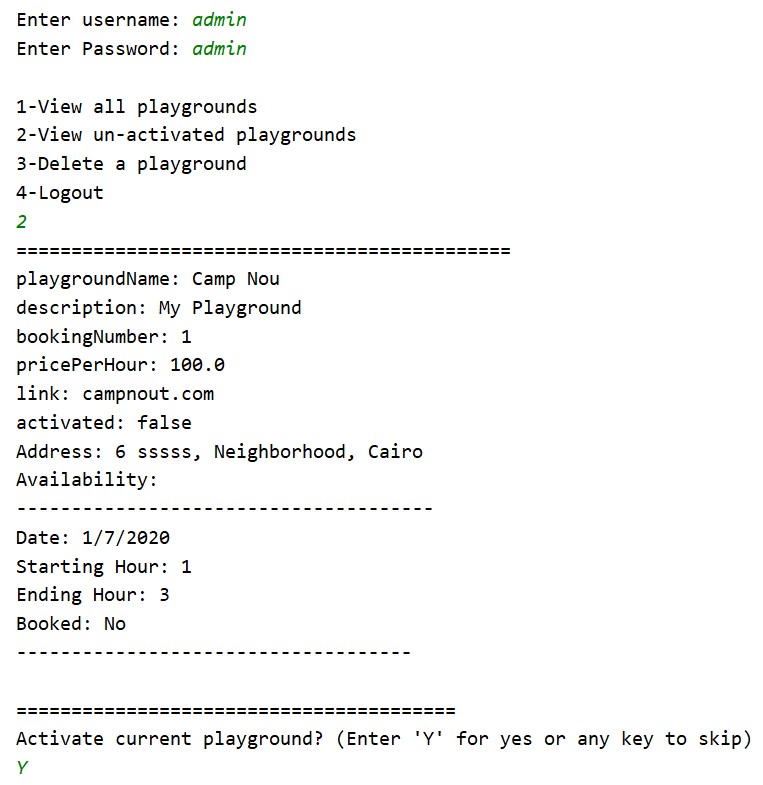
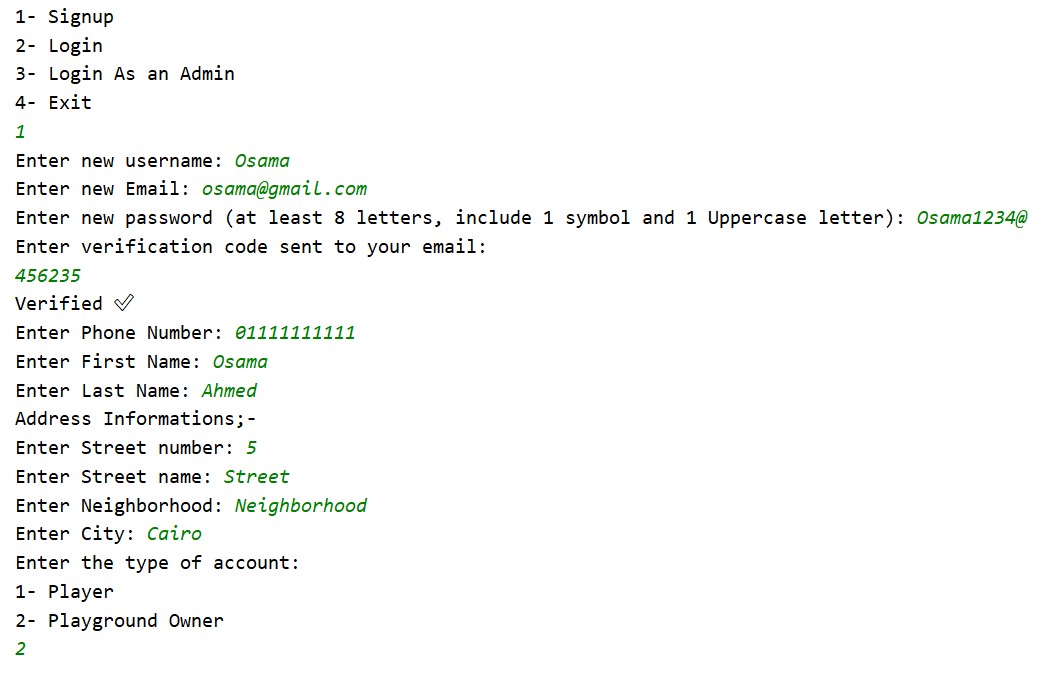
**package src.System;  
  
import src.Utilities.Playground;  
import src.Utilities.TimeSlot;  
import src.Users.Player;  
import src.Users.Team;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
*/\*\*  
 \* {@code PlayerUI} class is used to enable players to interact with the system using the console  
 \*/*public class PlayerUI {  
  
 private Scanner scanner = new Scanner(System.*in*);  
 private ArrayList<Playground> playgrounds;  
 private ArrayList<Player> players;  
 private int currentPlayer;  
  
 */\*\*  
 \* A constructor for the {@code PlayerUI} object  
 \* @param currentPlayer the index of the current player currently logged into the system  
 \*/* public PlayerUI(ArrayList<Playground> playgrounds, ArrayList<Player> players, int currentPlayer) {  
 this.playgrounds = playgrounds;  
 this.players = players;  
 this.currentPlayer = currentPlayer;  
 mainMenu();  
 }  
  
 */\*\*  
 \* A main menu the allows players to choose an operation to be done and redirects them to different parts of the system  
 \*/* public void mainMenu() {  
 String option = "";  
 while (true) {  
 System.*out*.println("\n1- Show Available Playgrounds" +  
 "\n2- Filter Playground using time slot" +  
 "\n3- Book a Playground" +  
 "\n4- View Current Bookings" +  
 "\n5- Create Team" +  
 "\n6- Modify Team" +  
 "\n7- Leave Team" +  
 "\n8- View Current Teams" +  
 "\n9- Report a Playground" +  
 "\n10- Access eWallet" +  
 "\n11- Logout");  
 option = scanner.nextLine();  
 if (option.equalsIgnoreCase("1")) {  
 viewAvailablePlaygrounds();  
 } else if (option.equalsIgnoreCase("2")) {  
 filterPlaygrounds();  
 } else if (option.equalsIgnoreCase("3")) {  
 bookPlayground();  
 } else if (option.equalsIgnoreCase("4")) {  
 viewBookings();  
 } else if (option.equalsIgnoreCase("5")) {  
 createTeam();  
 } else if (option.equalsIgnoreCase("6")) {  
 modifyTeam();  
 } else if (option.equalsIgnoreCase("7")) {  
 leaveTeam();  
 } else if (option.equalsIgnoreCase("8")) {  
 viewTeams();  
 } else if (option.equalsIgnoreCase("9")) {  
 reportPlayground();  
 } else if (option.equalsIgnoreCase("10")) {  
 eWalletUI eWalletUI = new eWalletUI(players.get(currentPlayer));  
 } else if (option.equalsIgnoreCase("11")) {  
 break;  
 } else {  
 System.out.println("\*\*\*Enter valid option\*\*\*\n");  
 }  
 }  
 System.out.println("\nLogging out ...");  
 }  
  
 */\*\*  
 \* Outputs the available playgrounds and their available time slots  
 \*/* private void viewAvailablePlaygrounds() {  
 for (Playground playground : playgrounds) {  
 if (!playground.isActivated()) continue;  
 for (int i = 0; i < playground.getAvailability().size(); ++i) {  
 if (!playground.getAvailability().get(i).isBooked()) {  
 System.out.println("==========================================");  
 System.out.println("Playground Name: " + playground.getPlaygroundName());  
 System.out.println("Playground Address: \n" + playground.getAddress().toString());  
 }  
 System.out.println(playground.getAvailability().get(i).toString());  
 }  
 }  
 }  
  
 */\*\*  
 \* Allows a user to enter the data of a time slot  
 \* @return the time slot the user entered  
 \*/* private TimeSlot setTimeslot() {  
 int day, month, year, startingHour, endingHour;  
 System.out.println("Enter timeslot details: ");  
 while (true) {  
 System.out.print("\tEnter day: ");  
 day = scanner.nextInt();  
 if (day >= 1 && day <= 31) {  
 break;  
 } else {  
 System.out.println("\*\*\*Enter valid day\*\*\*");  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter month: ");  
 month = scanner.nextInt();  
 if (month >= 1 && month <= 12) {  
 break;  
 } else {  
 System.out.println("\*\*\*Enter valid month\*\*\*");  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter year: ");  
 LocalDate date = LocalDate.now();  
 year = scanner.nextInt();  
 if (year < date.getYear()) {  
 System.out.println("\*\*\*Enter valid year\*\*\*");  
 } else {  
 break;  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter startingHour: ");  
 startingHour = scanner.nextInt();  
 if (startingHour < 0 || startingHour > 23) {  
 System.out.println("\*\*\*Enter valid hour\*\*\*");  
 } else {  
 break;  
 }  
 }  
 while (true) {  
 System.out.print("\tEnter endingHour: ");  
 endingHour = scanner.nextInt();  
 if (endingHour < 0 || endingHour > 23) {  
 System.out.println("\*\*\*Enter valid hour\*\*\*");  
 } else {  
 break;  
 }  
 }  
 scanner.skip("\n");  
 return new TimeSlot(day, month, year, startingHour, endingHour);  
 }  
  
 */\*\*  
 \* Allows a player to enter a time slot and view the available playgrounds at this time slot  
 \*/* private void filterPlaygrounds() {  
 TimeSlot timeSlot = setTimeslot();  
 for (Playground playground : playgrounds) {  
 if (!playground.isActivated()) continue;  
 boolean printed = false;  
 for (int i = 0; i < playground.getAvailability().size(); ++i) {  
 if (!playground.getAvailability().get(i).isBooked()  
 && playground.getAvailability().get(i).equals(timeSlot)) {  
 if (!printed) {  
 System.out.println("=========================================");  
 System.out.println("Playground Name: " + playground.getPlaygroundName());  
 System.out.println("Playground Address: \n" + playground.getAddress().toString());  
 printed = true;  
 }  
 System.out.println(playground.getAvailability().get(i).toString());  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* Allows a player to book a playground by entering its booking number (ID) and the desired time slot  
 \*/* private void bookPlayground() {  
 System.out.print("Enter playground booking number: ");  
 int bookingNumber = scanner.nextInt();  
 Playground playground = null;  
 for (Playground pg : playgrounds) {  
 if (!pg.isActivated()) continue;  
 if (pg.getBookingNumber() == bookingNumber) {  
 playground = pg;  
 break;  
 }  
 }  
 if (playground == null) {  
 System.out.println("Playground booking number is invalid!");  
 return;  
 }  
 boolean booked = false;  
 System.out.println("Enter time slot details to be booked: ");  
 TimeSlot timeSlot = setTimeslot();  
 if (players.get(currentPlayer).bookPlayground(playground, timeSlot)) {  
 System.out.println("\nPlayground Booked Successfully");  
 if (players.get(currentPlayer).getTeamOwned() != null) {  
 System.out.println("\nDo you want to send invitations to your owned team ?" +  
 "\n'Y' for yes or any key for no");  
 String option;  
 option = scanner.nextLine();  
 if (option.equalsIgnoreCase("Y")) {  
 players.get(currentPlayer).getTeamOwned().sendInvitations();  
 }  
 }  
 } else {  
 System.out.println("\nCould not book playground" +  
 "\nInvalid time slot or insufficient eWallet balance");  
 }  
 }  
  
 */\*\*  
 \* Creates a team of the player by entering its name and the players to be added to the team  
 \*/* private void createTeam() {  
 System.out.print("Enter Team name: ");  
 String name = scanner.nextLine();  
 String playerName;  
 boolean added = false;  
 var teamPlayers = new ArrayList<Player>();  
 while (true) {  
 added = false;  
 System.out.println("Enter player username to be added: ");  
 playerName = scanner.nextLine();  
 for (Player player : players) {  
 if (player.getUsername().equalsIgnoreCase(playerName)) {  
 teamPlayers.add(player);  
 added = true;  
 break;  
 }  
 }  
 if (!added) {  
 System.out.println("\*\*\*Player not found\*\*\*");  
 } else {  
 System.out.println("Player added successfully");  
 }  
 System.out.println("Enter 'Y' to add another player or any key to exit");  
 String option = scanner.nextLine();  
 if (!option.equalsIgnoreCase("Y")) {  
 break;  
 }  
 }  
 players.get(currentPlayer).createTeam(name, teamPlayers);  
 }  
  
 */\*\*  
 \* Modifies the team the player owns by entering the new team name and team members  
 \*/* private void modifyTeam() {  
 if (players.get(currentPlayer).getTeamOwned() == null) {  
 System.out.println("\*\*\*There is no team to modify, consider creating a team firstly\*\*\*");  
 return;  
 }  
 System.out.print("Enter new Team name: ");  
 String name = scanner.nextLine();  
 String playerName;  
 boolean added = false;  
 var teamPlayers = new ArrayList<Player>();  
 System.out.println("Enter new list of team players: ");  
 while (true) {  
 System.out.println("Enter player username to be added: ");  
 playerName = scanner.nextLine();  
 for (Player player : players) {  
 if (player.getUsername().equalsIgnoreCase(playerName)) {  
 teamPlayers.add(player);  
 added = true;  
 break;  
 }  
 }  
 if (!added) {  
 System.out.println("\*\*\*Player not found\*\*\*");  
 } else {  
 System.out.println("Player added successfully");  
 }  
 System.out.println("Enter 'Y' to add another player or any key to exit");  
 String option = scanner.nextLine();  
 if (!option.equalsIgnoreCase("Y")) {  
 break;  
 }  
 }  
 players.get(currentPlayer).modifyTeam(name, teamPlayers);  
 }  
  
 */\*\*  
 \* Allows a player to leave a team by entering its name  
 \*/* private void leaveTeam() {  
 System.out.println("Enter team name: ");  
 String teamName = scanner.nextLine();  
 if (players.get(currentPlayer).leaveTeam(teamName)) {  
 System.out.println("Left team successfully");  
 } else {  
 System.out.println("You're not enrolled in a team called " + teamName);  
 }  
 }  
  
 */\*\*  
 \* Allows a player to report a playground by entering its name  
 \*/* private void reportPlayground() {  
 System.out.println("Enter Playground booking number: ");  
 int playgroundName = scanner.nextInt();  
 scanner.skip("\n");  
 boolean found = false;  
 for (Playground playground : playgrounds) {  
 if (playground.getBookingNumber() == playgroundName && playground.isActivated()) {  
 players.get(currentPlayer).reportPlayground(playground);  
 found = true;  
 break;  
 }  
 }  
 if (found) System.out.println("Playground reported successfully");  
 else System.out.println("There is no such playground");  
 }  
  
 */\*\*  
 \* Outputs the current bookings of the player  
 \*/* private void viewBookings() {  
 System.out.println("\nThese are your current Bookings: ");  
 for (Playground playground : playgrounds) {  
 boolean printed = false;  
 for (TimeSlot timeSlot : playground.getAvailability()) {  
 if (timeSlot.isBooked() && timeSlot.getBookedTo().equalsIgnoreCase(players.get(currentPlayer).getUsername())) {  
 if (!printed) {  
 System.out.println("=========================================");  
 System.out.println("Playground Name: " + playground.getPlaygroundName());  
 System.out.println("Playground Address: \n" + playground.getAddress().toString());  
 printed = true;  
 }  
 System.out.println(timeSlot.toString());  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* Outputs the current teams the player is enrolled into or owns  
 \*/* private void viewTeams() {  
 boolean hasTeams = false;  
 if (players.get(currentPlayer).getTeamOwned() != null) {  
 hasTeams = true;  
 System.out.println("\nYour owned team: \n");  
 System.out.println(players.get(currentPlayer).getTeamOwned().toString());  
 }  
 if (players.get(currentPlayer).getTeamsEnrolled().size() > 0) {  
 hasTeams = true;  
 System.out.println("\nTeams you're enrolled in: \n");  
 for (Team team : players.get(currentPlayer).getTeamsEnrolled()) {  
 System.out.println(team.toString());  
 }  
 }  
 if (!hasTeams) System.out.println("\nYou don't own a team and you're not enrolled in one");  
 }  
}**

**package src.System;  
  
import src.Utilities.Playground;  
import src.Users.Administrator;  
  
import java.util.ArrayList;  
import java.util.Scanner;  
  
*/\*\*  
 \* {@code AdminUI} class allows the admins to interact with the system using the console  
 \*/*public class AdminUI {  
  
 private Administrator admin;  
 private Scanner scanner = new Scanner(System.*in*);  
 private ArrayList<Playground> playgrounds;  
  
 */\*\*  
 \* A constructor for the {@code AdminUI} object  
 \*  
 \* @param playgrounds a list of all playgrounds currently in the system  
 \* @param admin the current admin logged into the system  
 \*/* public AdminUI(ArrayList<Playground> playgrounds, Administrator admin) {  
 this.playgrounds = playgrounds;  
 this.admin = admin;  
 mainMenu();  
 }  
  
 */\*\*  
 \* A main menu that allows an admin to choose an operation  
 \*/* public void mainMenu() {  
 while (true) {  
 System.*out*.println("\n1-View all playgrounds");  
 System.*out*.println("2-View un-activated playgrounds");  
 System.*out*.println("3-Delete a playground");  
 System.*out*.println("4-Logout");  
  
 String option = scanner.nextLine();  
 if (option.equals("1")) {  
 viewPlaygrounds();  
 } else if (option.equals("2")) {  
 viewUnactivated();  
 } else if (option.equals("3")) {  
 deletePlayground();  
 } else if (option.equals("4")) {  
 break;  
 } else {  
 System.out.println("\*\*\*Enter valid option\*\*\*");  
 }  
 }  
 System.out.println("Logging out ...");  
 }  
  
 */\*\*  
 \* Outputs all the playgrounds currently in the system and their details  
 \*/* private void viewPlaygrounds() {  
 for (Playground playground : playgrounds) {  
 System.out.println(playground.toString());  
 }  
 }  
  
 */\*\*  
 \* Outputs all the playgrounds that are not activated in the system, and allows an admin to activate a playground  
 \*/* private void viewUnactivated() {  
 for (Playground playground : playgrounds) {  
 if (!playground.isActivated()) {  
 System.out.println(playground.toString());  
 System.out.println("Activate current playground? (Enter 'Y' for yes or any key to skip)");  
 String option = scanner.nextLine();  
 if (option.equalsIgnoreCase("Y")) {  
 admin.activatePlayground(playground);  
 System.out.println("\nPlayground activated successfully ✅");  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \* Allows an admin to delete a playground by entering its booking number (ID)  
 \*/* private void deletePlayground() {  
 System.out.println("Enter playground booking number: ");  
 int bookingNumber = scanner.nextInt();  
 scanner.skip("\n");  
 Playground playground = null;  
 for (int i = 0; i < playgrounds.size(); ++i) {  
 if (playgrounds.get(i).getBookingNumber() == (bookingNumber)) {  
 playground = playgrounds.get(i);  
 break;  
 }  
 }  
 if (playground == null) {  
 System.out.println("\*\*\*Playground not found\*\*\*");  
 return;  
 }  
 playgrounds.remove(playground);  
 System.out.println("Playground removed successfully ✅");  
 }  
}**

**package src.System;  
  
import src.Users.\*;  
  
import java.util.Scanner;  
  
*/\*\*  
 \* {@code eWalletUI} class is used to enable users to access their eWallets and do operations on them  
 \*/*public class eWalletUI {  
 private User user;  
 Scanner scanner = new Scanner(System.*in*);  
  
 */\*\*  
 \* A constructor for the {@code eWalletUI} object  
 \* @param user the current user logged into the system  
 \*/* public eWalletUI(User user) {  
 this.user = user;  
 mainMenu();  
 }  
  
 */\*\*  
 \* A main menu that allows a user to do choose an operation to be done on their eWallets  
 \*/* public void mainMenu() {  
 String option = "";  
 System.*out*.println("Enter an option");  
 while (true) {  
 System.*out*.println("\n1- View Current Balance" +  
 "\n2- Deposit" +  
 "\n3- Withdraw" +  
 "\n4- Exit");  
 option = scanner.nextLine();  
 if (option.equalsIgnoreCase("1")) {  
 viewBalance();  
 } else if (option.equalsIgnoreCase("2")) {  
 deposit();  
 } else if (option.equalsIgnoreCase("3")) {  
 withdraw();  
 } else if (option.equalsIgnoreCase("4")) {  
 break;  
 } else {  
 System.*out*.println("\*\*\*Enter valid option\*\*\*\n");  
 }  
 }  
 System.*out*.println("\nExiting ...");  
 }  
  
 */\*\*  
 \* Outputs the current balance of the user's eWallet  
 \*/* public void viewBalance() {  
 System.*out*.println("Your Current Balance is: " + user.getEwallet().getBalance());  
 }  
  
 */\*\*  
 \* Allows users to deposit to their eWallet accounts by entering the amount  
 \*/* public void deposit() {  
 System.*out*.print("Enter the amount to deposit: ");  
 double amount;  
 amount = scanner.nextDouble();  
 scanner.skip("\n");  
 if (user.getEwallet().deposit(amount)) {  
 System.*out*.println("Deposited " + amount + " successfully");  
 } else {  
 System.*out*.println("Invalid amount to deposit");  
 }  
 }  
  
 */\*\*  
 \* Allows users to withdraw from their eWallet accounts by entering the amount  
 \*/* public void withdraw() {  
 System.*out*.print("Enter the amount to withdraw: ");  
 double amount;  
 amount = scanner.nextDouble();  
 scanner.skip("\n");  
 if (user.getEwallet().withdraw(amount)) {  
 System.out.println("Withdrew " + amount + " successfully");  
 } else {  
 System.out.println("Invalid amount to withdraw");  
 }  
 }  
}**

**package src.System;  
*/\*\*  
 \* @author Mahmoud Mohamed Abdelazim 20180263  
 \* @author Ahmed Alaa Eldin 20180435  
 \* @author Amr Bumadian 20180436  
 \* @author Ziad Amr 20180379  
 \* @author Eslam Fawzy 20180367  
 \*/*import java.util.Scanner;  
  
public class Main {  
  
 public static void main(String[] args) {  
 GoFoSystem GoFo = new GoFoSystem();  
  
 */\* Usernames and passwords for Testing:  
 owner owner  
 player player  
 admin admin  
 \*/* }  
}**

**Screenshots:**



**Github Link (added github.com/mramly as a collaborator):**

[**https://github.com/MahmoudAbdelazim/Software-Eng-Project**](https://github.com/MahmoudAbdelazim/Software-Eng-Project)

**Google Drive Link (shared with** [**m.elramly@fci-cu.edu.eg**](mailto:m.elramly@fci-cu.edu.eg) **):**

**https://drive.google.com/file/d/1r-qObbA7uEyr41A\_pEpSCaqWfN1pa6vN/view?usp=sharing**

# Authors

* Mostafa Saad and Mohammad El-Ramly (Edited by Mohamed Samir) (V1.0)
* Updated by Mohammad El-Ramly (V2.0)