Cairo University
Faculty of Computers and Information

CS251

Software Engineering I

GOFO

Software Design Specifications

CS251: Avengers
Project: GOFO

Software Design Specification

Contents

CS251: Avengers
Project: GOFO

Software Design Specification

Team

ID	Name	Email	Mobile
20180250	Mahmoud Ashraf	ma5027300@gmail.com	01102488789
20180304	Nader Fikry	naderfikry245@gmail.com	01097632795
20180193	Fady Emad	fadyemad14705@gmail.com	01203732443

Document Purpose and Audience

- This program helps players to find and book football fields without wasting much time and effort.
- This document describes the shape, structure and design of the program through models.

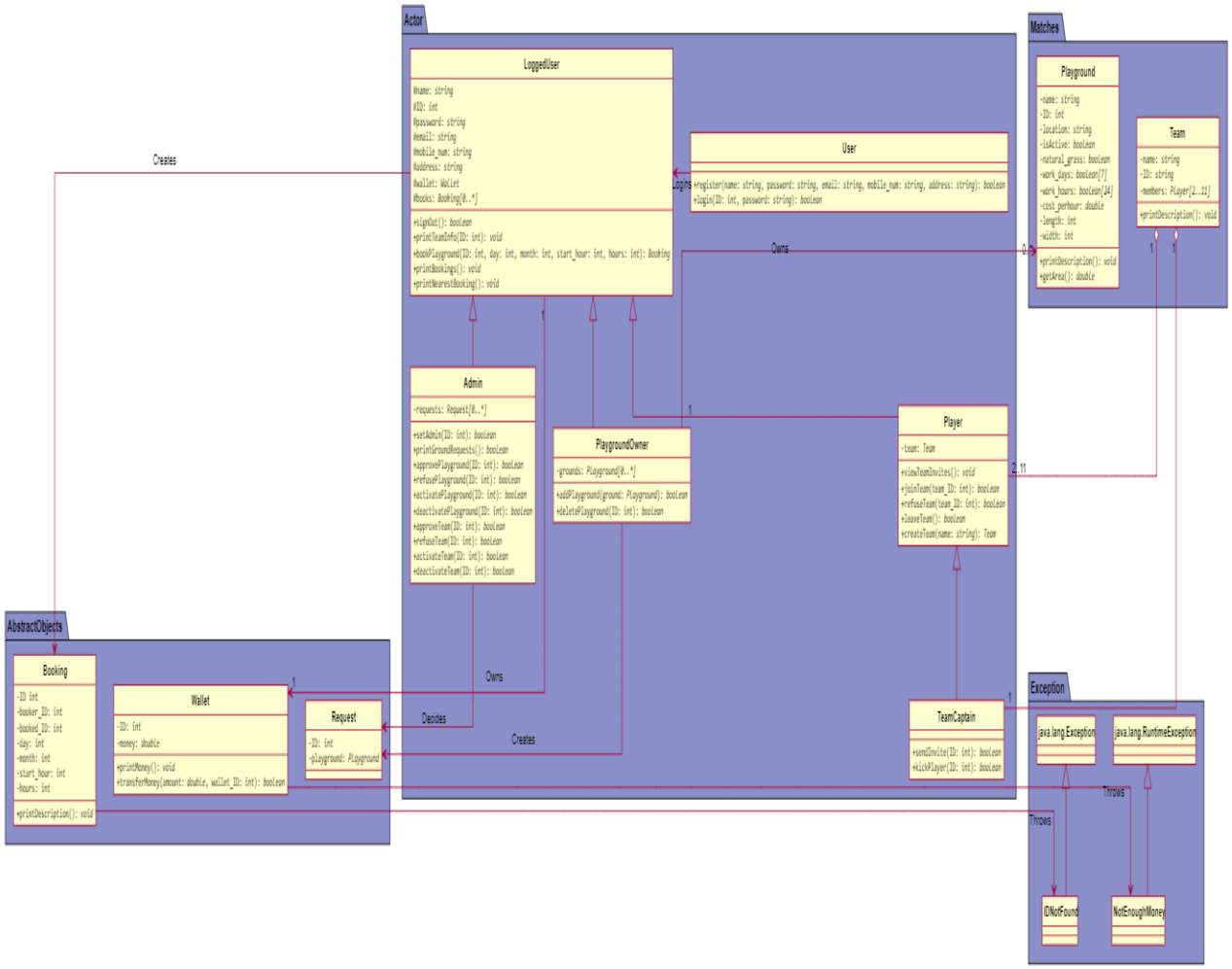
CS251: Avengers
Project: GOFO

Software Design Specification

- There are two main audiences in the program first, the player who wants to book football fields second, is the playground owner who wants to provide playgrounds fields.

System Models

I. Class Diagram(s)



Software Design Specification

II. Class Descriptions

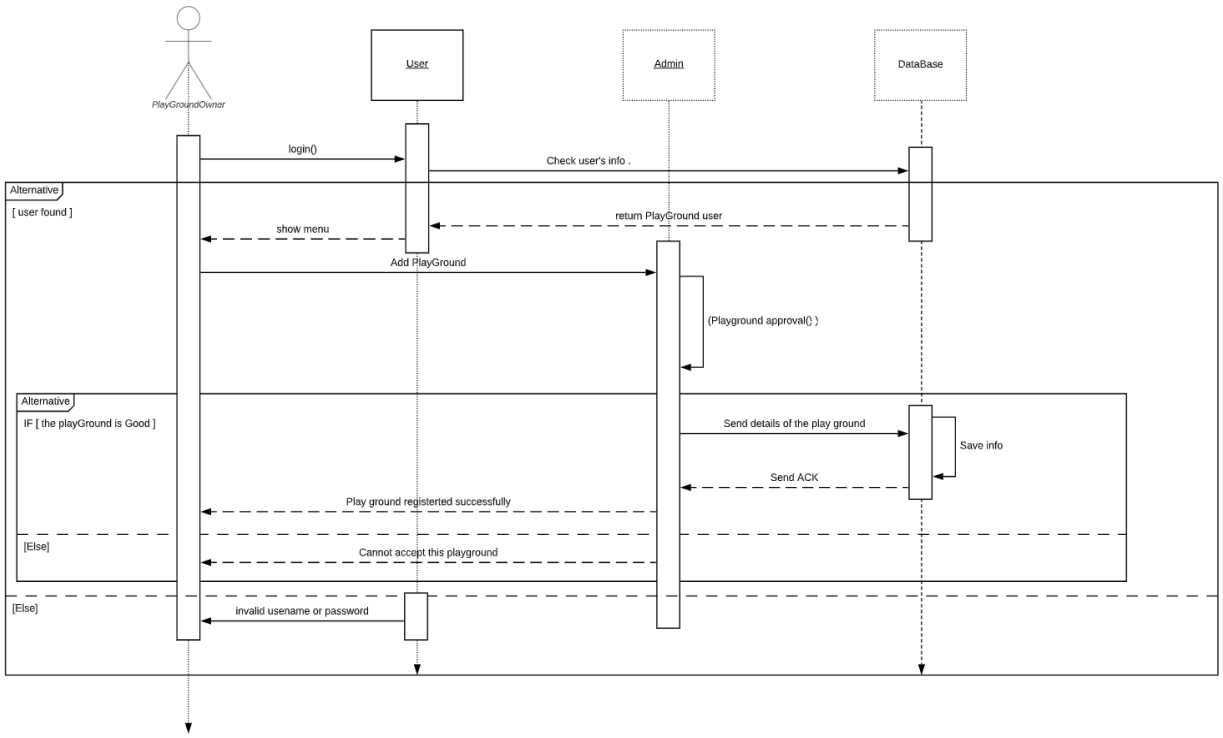
Class ID	Class Name	Description & Responsibility
1.	User	A class that represents a user that is neither registered nor logged in. A user that has just opened a program and only has two responsibilities: to login or register.
2.	LoggedUser	A class that represents a user that is registered and is logged in. Classes: Player, PlaygroundOwner and Admin inherit from this class. Responsibilities: Sign out, Book a playground, view his bookings and check his wallet.
3.	Player	A class that represents a player. Responsibilities: All LoggedUser responsibilites + create a team, view Teams he is invited to and accept or refuse invitations.(See TeamCaptain for more info)
4.	PlaygroundOwner	A class that represents a playground owner, who owns 1 or more playgrounds. Responsibilities: Add or Remove owned Playgrounds. An admin must accept or refuse such requests.
5.	Admin	A class that represents a user with special privileges. An admin is responsible for managing the program. He is responsible for Managing Playground and Team requests. And suspending Playgrounds with bad history.
6.	TeamCaptain	A class that inherits from a player. It represents a normal player but who has created his own team. A team must have one and only one TeamCaptain He is responsible for sending invites to other players and kick players from his team.
7.	Playground	A class that represents a playground. It has many important attributes to determine its name, its location, if it is active, if it has natural grass, its area, its work days, its work hours and its cost.
8.	Team	A class that is created when a player creates his own team. It has only one captain and up to 10 normal players. It also has a name.
9.	Booking	A class to represent a booking that is made by a player to a playground. It saves the ID of the booker, the ID of the booked playground, total cost and the date and duration of the booking.

Software Design Specification

Class ID	Class Name	Description & Responsibility
10.	Wallet	A class that represents a wallet that saves a LoggedUser's money. Depositing money to the wallet happens out of the system's boundary.
11.	Request	A class that represents a request made to the admins. It could be a Playground request or a Team creation request. It is deleted when responded by one of the admins.
12.	IDNotFound	An exception that is thrown whenever the user specifies an ID of a player, playground.. etc that doesn't exist.
13.	NotEnoughMoney	An exception that is thrown when trying to book a playground but the wallet doesn't have enough money.

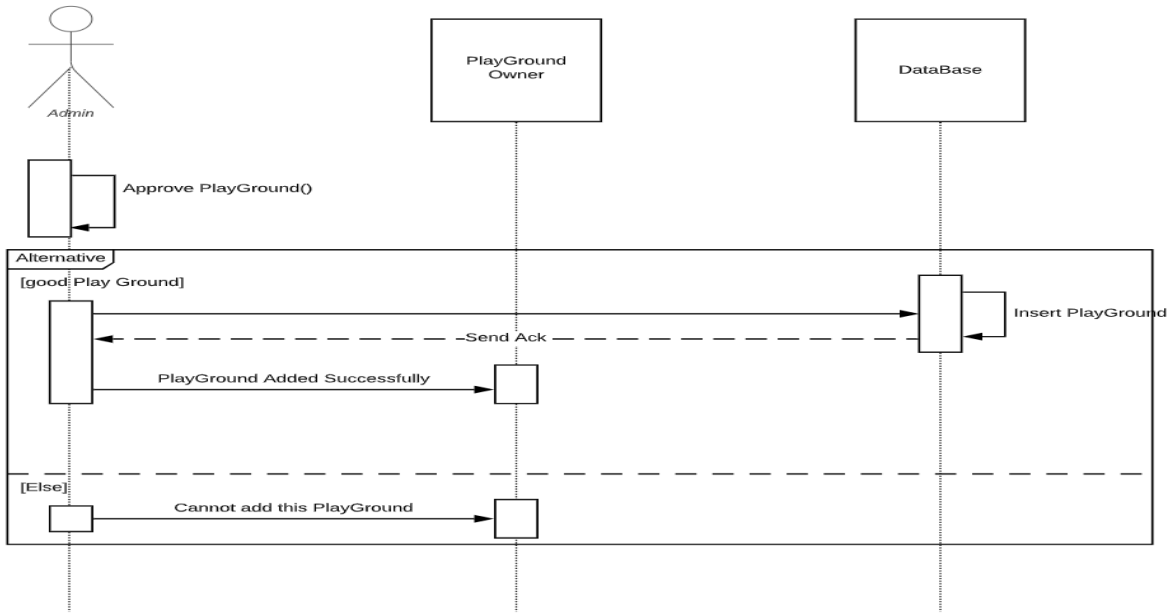
III. Sequence diagrams

Add Playground:

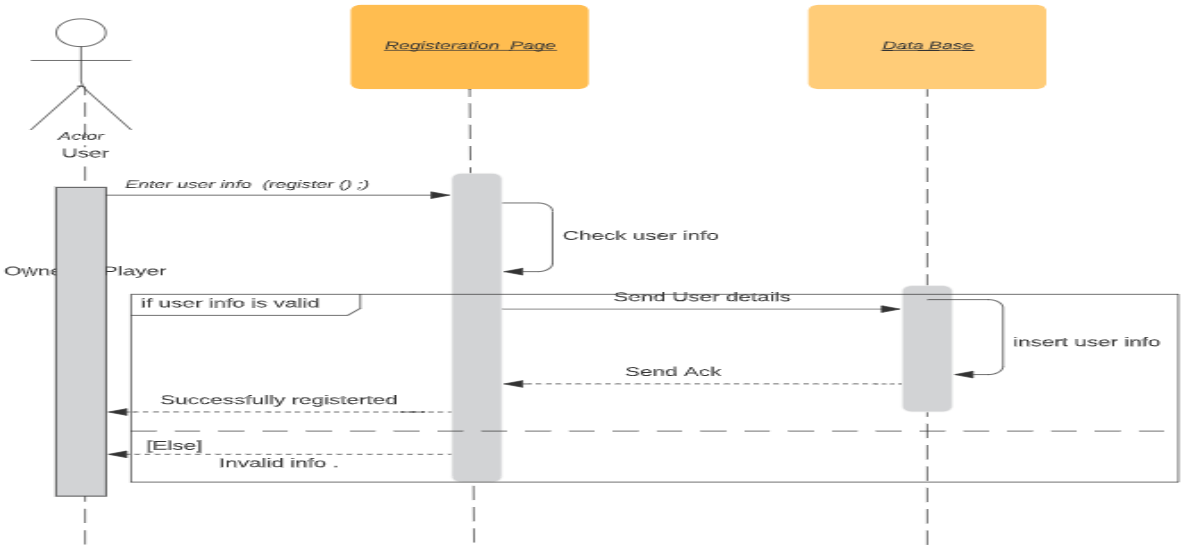


Software Design Specification

Approve Playground:

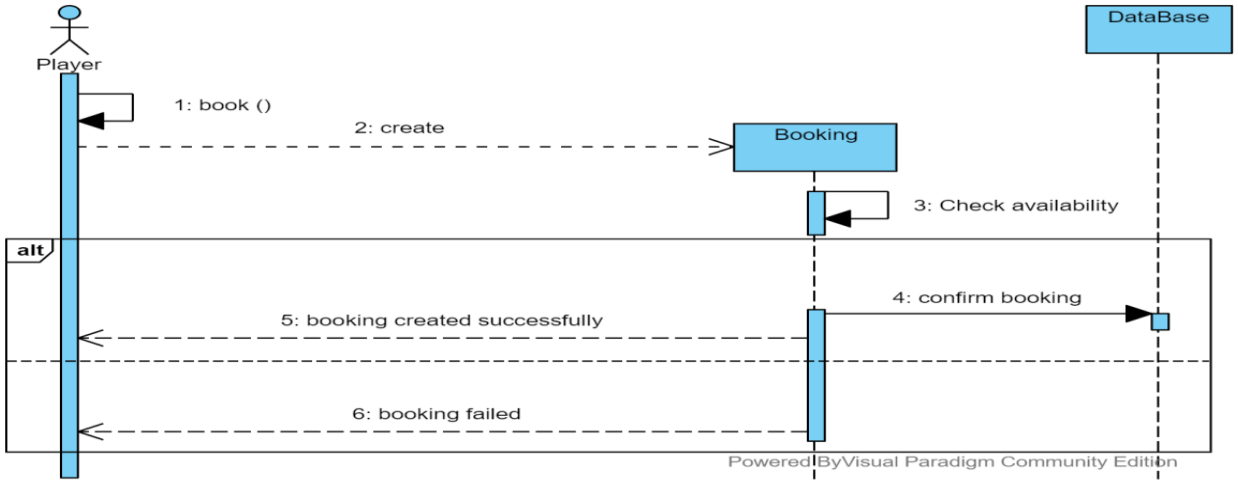


Register user:

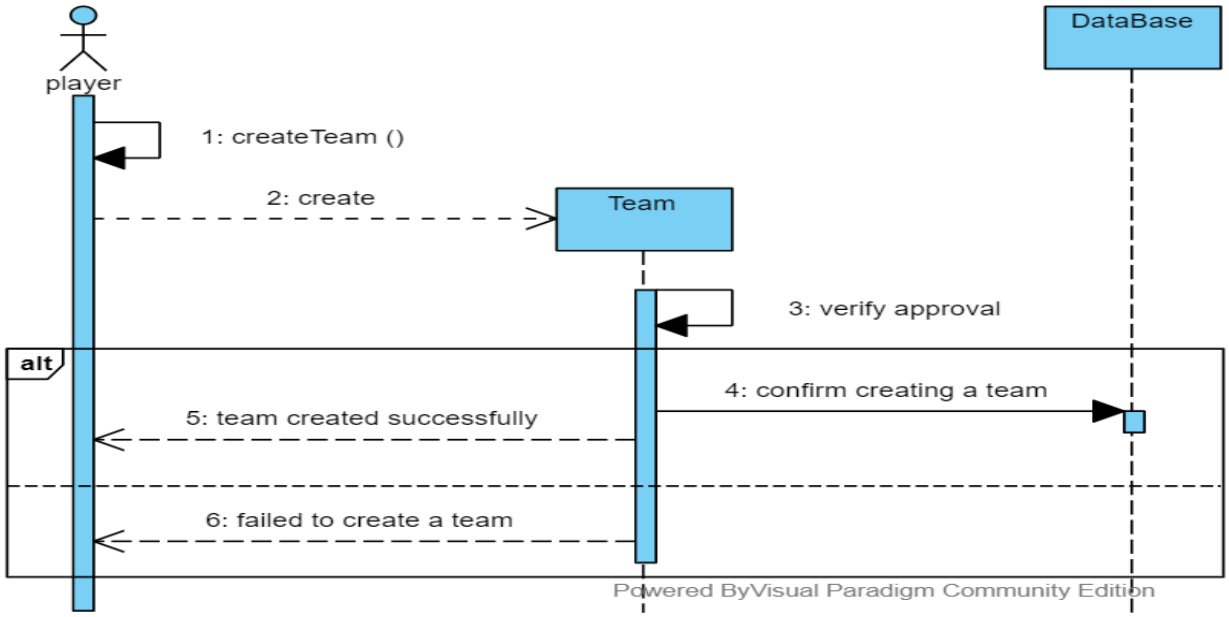


Software Design Specification

Book a Playground:

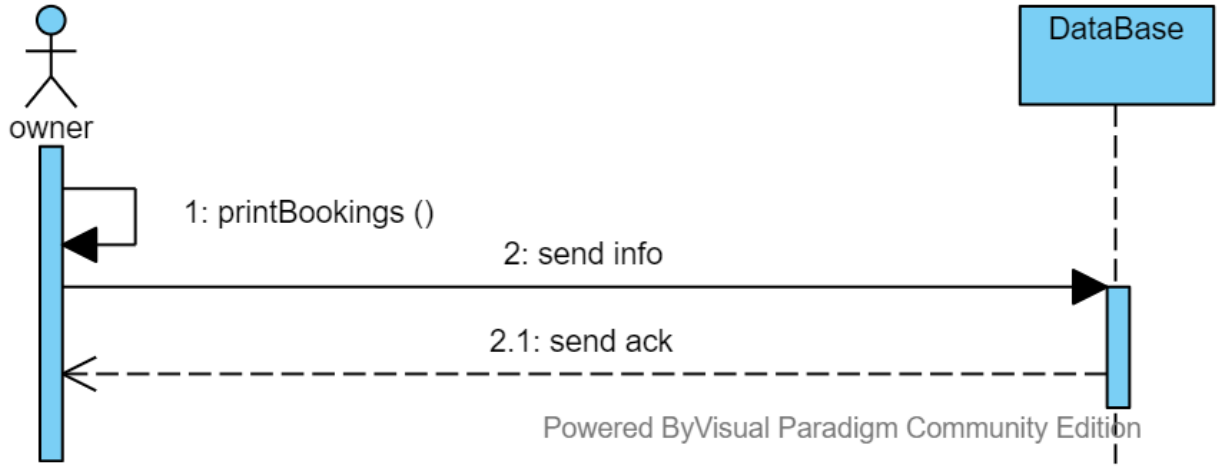


Create team:



Software Design Specification

View playing hours:



Class - Sequence Usage Table

Sequence Diagram	Classes Used	All Methods Used
1. Add Playground	Admin User	login()
2. Approve Playground	PlaygroundOwner	approvePlayground()
3. Book a Playground	Booking	book()
4. Create team	Team	createTeam()
5. Register user	User	Register()
6. View playing hours	PlaygroundOwner	printBookings()

Software Design Specification

IV. User Interface Design

CS251: Avengers
Project: GOFO

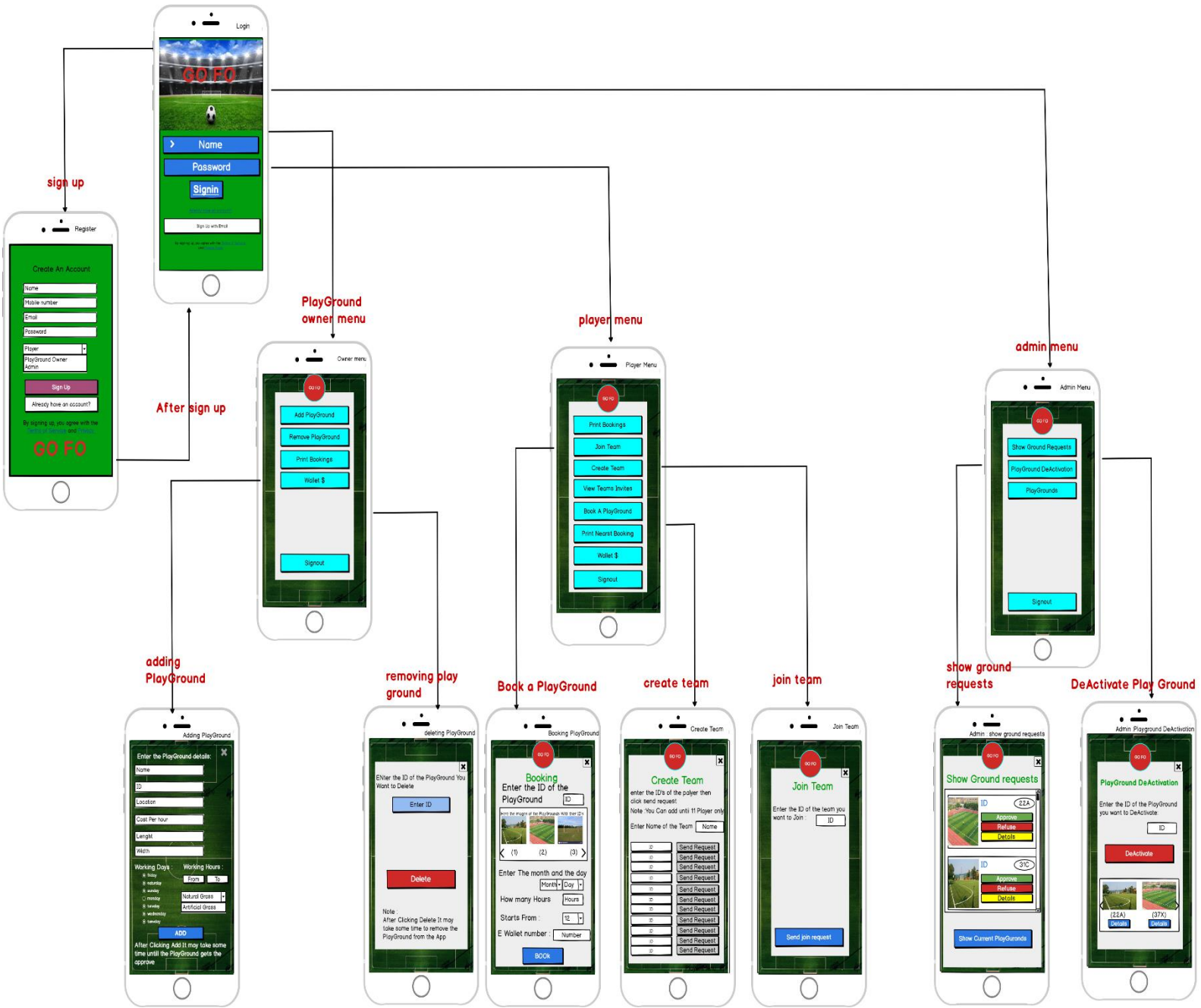
Software Design Specification

Screen ID	Screen Name	Screen / Wireframe Description
#1	Login	Used to login if you have an account
#2	Register	Used to make an account for admin or player or playground owner
#3	Owner Menu	This menu appears when Play Ground owner sign it shows the functionalities of the Owner that he can do in the program.
#4	Adding Play Ground	A form that takes the playground details from the owner to add the playground in the application.
#5	Deleting Play Ground	A form that takes an ID of one of the play grounds of the owner to delete it from the application
#6	Player menu	This menu appears when a player sign it shows the functionalities of the player that he can do in the program.
#7	Booking Play Ground	This screen shows the player ground with their id's to allow the player to book a playground at a specific time.
#8	Create Team	A form that takes the id's of the other players to invite them to join to the team of the user.
#9	Join Team	Form that takes the id of the team that the user want to join to send a join request to the captain of the team
#10	Admin Menu	This menu appears when admin sign it shows the functionalities of the admin that he can do in the program.
#11	Show Grounds Requests	A form that shows the playground that need to get an approve from the admin to be added to the application
#12	Play Ground De Activation	A form that takes a playground id to deactivate it.

CS251: Avengers
Project: GOFO

Software Design Specification

Navigation Map for the Screens:



CS251: Avengers
Project: GOFO

Software Design Specification

Tools

- **Visual-Paradigm**
- **Mocqus**
- **Lucidchart**
- **PlantUML**

Ownership Report

Owners	Item
Mahmoud Ashraf	Three sequence diagrams (first three),User Interface design , Class Diagram (Admin, PlayGround).
Nader Fikry	Three sequence diagrams(last three), SDS document, Implementation of Abstract Objects package and a video that describes the components of the system.
Fady Emad	Class Diagram and Console Implementation

Software Design Specification

Screenshots and Video

```
Welcome to GoFo.
You can interact with the UI by entering the number matching your choice.
You're currently unregistered. If you do not have an account then register.
If you have an account then log in.
1) Register
2) Login
3) Exit the Program
Enter your choice: 1
```

```
Do you want to create an account as a:
1) Player
2) Playground Owner
Enter your choice: 1
```

```
Enter your Name: ahmed
Enter your Password: passw0rd
Enter your Email: ahmed@gofomail.com
Enter your Address: cairo, egypt
Enter your Mobile Number: 6660666
The account was successfully registered. Please log in.
1) Register
2) Login
3) Exit the Program
Enter your choice: 2
```

Software Design Specification

```
Do you want to log in to a:
1) Player account
2) Playground Owner account
3) Admin account
Enter your choice: 1
Enter your Email: ahmed@gofomail.com
Enter your Password: passw0rd
Logged in successfully.
```

```
1) Create team
2) View team invites
3) Accept a team invitation
4) Book a playground
5) View all playgrounds
6) Search playgrounds by location
7) Check wallet
8) Sign out
Enter your choice: 1
Enter the name of the Team: myteam
Your team was successfully created
Please log in again to update your account.
1) Register
2) Login
3) Exit the Program
Enter your choice: 2
```

Software Design Specification

```
Do you want to log in to a:
1) Player account
2) Playground Owner account
3) Admin account
Enter your choice: 1
Enter your Email: ahmed@gofomail.com
Enter your Password: passw0rd
Logged in successfully.
1) Invite player to team
2) Book a playground
3) View all playgrounds
4) Search playgrounds by location
5) Check wallet
6) Sign out
Enter your choice: 6
```

Software Design Specification

```
You've signed out.
1) Register
2) Login
3) Exit the Program
Enter your choice: 1
Do you want to create an account as a:
1) Player
2) Playground Owner
Enter your choice: 2
Enter your Name: mohamed
Enter your Password: passw0rd
Enter your Email: mohamed@gofomail.com
Enter your Address: cairo, egypt
Enter your Mobile Number: 9990999
The account was successfully registered. Please log in.
```

```
1) Register
2) Login
3) Exit the Program
Enter your choice: 2
Do you want to log in to a:
1) Player account
2) Playground Owner account
3) Admin account
Enter your choice: 2
Enter your Email: mohamed@gofomail.com
Enter your Password: passw0rd
Logged in successfully.
1) Add playground
2) View owned playgrounds
3) Check wallet
4) Sign out
Enter your choice: 1
```

Software Design Specification

```
Enter the Playground name: the playground
Enter the Playground location: cairo, egypt
Enter the Playground booking cost per hour: 41.5
Does the playground have natural grass? (y/n): y
Enter the opening day number for the playground(between [0, 6] if day 0 is Saturday): 0
Enter the closing day number for the playground(between [0, 6] if day 0 is Saturday): 6
Enter the opening hour for the playground(between [0, 23]if 0 is 12 AM): 7
Enter the closing hour for the playground(between [0, 23] if 0 is 12 AM): 18
Enter the Playground length in metres: 100
Enter the Playground width in metres: 50
Your request for the playground has been successfully created.
Wait for an admin to accept your request for the playground to be activated.
```

```
1) Add playground
2) View owned playgrounds
3) Check wallet
4) Sign out
Enter your choice: 4
You've signed out.
1) Register
2) Login
3) Exit the Program
Enter your choice: 2
Do you want to log in to a:
1) Player account
2) Playground Owner account
3) Admin account
Enter your choice: 3
```

Software Design Specification

```
Enter your Email: a
Enter your Password: a
Logged in successfully.
1) Print Requests
2) Approve a Playground
3) Sign out
Enter your choice: 1
```

```
Request ID: 1
Playground Details

Name: the playground
ID: 1
Location: cairo, egypt
Activated: false
Natural Grass: true
Opening Days(if first element day is Saturday): [true, true, true, true, true, true, false]
Opening hours(if first hour is 12 AM): [false, false, false, false, false, false, false, true, true, true, true, true, true, true, true, true, tr
Hourly Cost: 41.5
Dimensions(in metres): 100x50
```

```
1) Print Requests
2) Approve a Playground
3) Sign out
Enter your choice: 2
Enter the ID of the request you want to accept: 1
Playground was approved.
```

Software Design Specification

```
1) Print Requests
2) Approve a Playground
3) Sign out
Enter your choice: 3
You've signed out.
1) Register
2) Login
3) Exit the Program
Enter your choice: 2
Do you want to log in to a:
1) Player account
2) Playground Owner account
3) Admin account
Enter your choice: 1
Enter your Email: ahmed@gofomail.com
Enter your Password: passw0rd
Logged in successfully.
```

```
1) Invite player to team
2) Book a playground
3) View all playgrounds
4) Search playgrounds by location
5) Check wallet
6) Sign out
Enter your choice: 3
```

Software Design Specification

```
Name: the playground
ID: 1
Location: cairo, egypt
Activated: true
Natural Grass: true
Opening Days(if first element day is Saturday): [true, true, true, true, true, true, false]
Opening hours(if first hour is 12 AM): [false, false, false, false, false, false, false, true, true, true, true, true, true, true, true, true, tr
Hourly Cost: 41.5
Dimensions(in metres): 100x50
1) Invite player to team
2) Book a playground
3) View all playgrounds
4) Search playgrounds by location
5) Check wallet
6) Sign out
Enter your choice: 2
```

```
Enter the ID of the playground you want to book: 1
Enter the month you want to book in(1 - 12): 7
Enter the day you want to book in(1 - 31): 22
Enter the start hour you want to book in(0 - 23): 12
Enter the amount of hours you want to book(0 - 23): 2
The total cost of the booking is: 83.0
```

Source Code

The filename is written first then the code inside the file.

Booking.java:

```
package AbstractObjects;
/**
 * Class representing a booking made by a player to a playground.
 * contains required attributes for details regarding the booking
 */
public class Booking {
    private int ID;
    private int booker_ID;
    private int booked_ID;
    private int day;
    private int month;
```

Software Design Specification

```
    private int start_hour;
    private int hours;
    public Booking(int booker_ID, int booked_ID, int day, int month,
int start_hour, int hours) {
        this.ID = IDGenerator.Booking();
        this.booker_ID = booker_ID;
        this.booked_ID = booked_ID;
        this.day = day;
        this.month = month;
        this.start_hour = start_hour;
        this.hours = hours;
    }
    public int getID() {
        return ID;
    }
    /**
     * getBookerID method to return the value of booker_ID attribute.
     * @return
     */
    public int getBookerID() {
        return booker_ID;
    }
    /**
     * getBookedID method to return the value of booked_ID attribute.
     * @return
     */
    public int getBookedID() {
        return booked_ID;
    }
    /**
     * getDay method to return the value of day attribute.
     * @return
     */
    public int getDay() {
        return day;
    }
    /**
     * getMonth method to return the value of the month attribute.
     * @return
     */
    public int getMonth() {
        return month;
    }
    /**
```

Software Design Specification

```
     * getStartHour method to return the value of start_hour
attribute.
     * @return
     */
    public int getStartHour() {
        return start_hour;
    }
    /**
     * getHours method to return the value of hours attribute.
     * @return
     */
    public int getHours() {
        return hours;
    }
    /**
     * printDescription method to print the information of the class.
     */
    public void printDescription() {
        System.out.print("The booking ID is: " + getID() + "/nThe
booker ID is: " + getBookerID() + "/nThe booked playground ID is: "
                + getBookedID() + "/nThe booking day is: " + getDay()
+ "/nThe booking month is: " + getMonth()
                + "/nThe booking start hour is: " +getStartHour() +
"/nThe booking hours is: " + getHours());
    }
}
```

Database.java:

```
package AbstractObjects;
import java.util.ArrayList;
import Actor.Admin;
import Actor.Player;
import Actor.PlaygroundOwner;
import Matches.Playground;
import Exception.IDNotFound;

/**
 *  A class to store the necessary data registered in the program.
 *  has an Arraylist attribute for each category(Ex: players,
playground owners).
 */
public class Database {
    public static ArrayList<Player> players_accounts = new
ArrayList<Player>();
```

Software Design Specification

```
    public static ArrayList<PlaygroundOwner> playgroundOwners_accounts
= new ArrayList<PlaygroundOwner>();
    public static ArrayList<Admin> admins_accounts = new
ArrayList<Admin>();
    public static ArrayList<Playground> playgrounds = new
ArrayList<Playground>(); // this array contains active playgrounds
only
    public static ArrayList<Booking> bookings = new
ArrayList<Booking>();
    /**
     * Verify a player account as stored in the database
     * @param email given email
     * @param password given password
     * @return null if account doesn't exist. otherwise returns the
account
     */
    public static Player verifyPlayer(String email, String password){
        for(int i = 0; i < players_accounts.size(); ++i){

if(players_accounts.get(i).getEmail().equalsIgnoreCase(email) &&
players_accounts.get(i).getPassword().equals(password)){
                return players_accounts.get(i);
            }
        }
        return null;
    }
    /**
     * Verify a playground owner account as stored in the database
     * @param email given email
     * @param password given password
     * @return null if account doesn't exist. otherwise returns the
account
     */
    public static PlaygroundOwner verifyPlaygroundOwner(String email,
String password){
        for(int i = 0; i < playgroundOwners_accounts.size(); ++i){

if((playgroundOwners_accounts.get(i).getEmail().equalsIgnoreCase(email
)) &&
(playgroundOwners_accounts.get(i).getPassword().equals(password))){
                return playgroundOwners_accounts.get(i);
            }
        }
        return null;
    }
```

Software Design Specification

```
    /**
     * Verify an admin account as stored in the database
     * @param email given email
     * @param password given password
     * @return null if account doesn't exist. otherwise returns the
account
     */
    public static Admin verifyAdmin(String email, String password){
        for(int i = 0; i < admins_accounts.size(); ++i){

if(admins_accounts.get(i).getEmail().equalsIgnoreCase(email) &&
admins_accounts.get(i).getPassword().equals(password)){
                return admins_accounts.get(i);
            }
        }
        return null;
    }
    /**
     * searches the playgrounds for the given playground ID
     */

    public static Playground searchPlayground(int ID) {
        for(int i = 0; i < playgrounds.size(); ++i){
            if(playgrounds.get(i).getID() == ID){
                return playgrounds.get(i);
            }
        }
        return null;
    }
    /**
     * Checks if the given email is already registered.
     * used for verification while registering
     * second parameter decides to search in players or playground
owners array
     * true for players
     * false for playground owners
     */
    public static boolean isDuplicate(String email, boolean isPlayer){
        if(isPlayer){
            for(int i = 0; i < players_accounts.size(); ++i){

if(players_accounts.get(i).getEmail().equalsIgnoreCase(email)){
                    return true;
                }
            }
```

Software Design Specification

```
        }
        else {
            for(int i = 0; i < playgroundOwners_accounts.size(); ++i){

if(playgroundOwners_accounts.get(i).getEmail().equalsIgnoreCase(email)
){
                    return true;
                }
            }
        }
        return false;
    }
    public static Player searchPlayer(int ID) {
        for(int i = 0; i < players_accounts.size(); ++i){
            if(players_accounts.get(i).getID() == ID){
                return players_accounts.get(i);
            }
        }
        return null;
    }
}
```

IDGenerator.java:

```
package AbstractObjects;
/**
 * A class to generate unique IDs for different categories in the
program
 * has methods resembling the different categories. each method
returns a unique ID for that category.
 */
public class IDGenerator {
    private static int booking_ID = 0;
    private static int request_ID = 0;
    private static int wallet_ID = 0;
    private static int user_ID = 0;
    private static int team_ID = 0;
    private static int playground_ID = 0;

    public static int Booking(){
        ++booking_ID;
        return booking_ID;
    }
    public static int Request(){
        ++request_ID;
```

Software Design Specification

```
        return request_ID;
    }
    public static int Wallet(){
        ++wallet_ID;
        return wallet_ID;
    }
    public static int User(){
        ++user_ID;
        return user_ID;
    }
    public static int Team(){
        ++team_ID;
        return team_ID;
    }
    public static int Playground(){
        ++playground_ID;
        return playground_ID;
    }
}
```

Request.java:

```
package AbstractObjects;
import Matches.Playground;
/**
 * A class that represents a request made by a playground owner to
admins to add a playground
 */
public class Request {
    private int ID;
    private Playground playground;
    /**
     * Parameterized constructor to give the attributes a certain
value.
     * @param playground Initialize a request with the requested
playground
     */
    public Request(Playground playground) {

        this.playground = playground;
        this.ID = IDGenerator.Request();
    }
    /**
     * setID method to set the value of ID attribute.
     * @param ID
```

Software Design Specification

```
     */
    public void setID(int ID) {
        this.ID = ID;
    }
    /**
     * getID method to return the value of ID attribute.
     * @return
     */
    public int getID() {
        return ID;
    }
    /**
     * get method to return the requested playground
     * @return
     */
    public Playground getPlayground() {
        return playground;
    }
}
```

Wallet.java:

```
package AbstractObjects;
/**
 * A class represents the wallet that saves a user's money
 */
public class Wallet {
    private int ID;
    private double money;
    /**
     * Default constructor to initialize the attributes.
     */
    public Wallet() {
        this.ID = IDGenerator.Wallet();
        this.money = 1000;
    }
    /**
     * setMoney method to set the value of the money.
     * @param money
     */
    public void setMoney(double money) {
        this.money = money;
    }
    /**
     * getMoney method to return the value of the money.
```

Software Design Specification

```
     * @return
     */
    public double getMoney() {
        return money;
    }
    public int getID() {
        return ID;
    }
    /**
     * printMoney method to print the amount of the money.
     */
    public void printMoney() {
        System.out.println("Your current balance is: " + getMoney());
    }
}
```

Admin.java:

```
package Actor;
import AbstractObjects.Booking;
import AbstractObjects.Database;
import AbstractObjects.Request;
import Matches.Playground;
import Exception.IDNotFound;
import java.util.ArrayList;
import java.util.Scanner;
/**
 * A user that has special control over the program, such as accepting
playgrounds.
 */
public class Admin extends LoggedUser {
    public static ArrayList<Request> requests = new
ArrayList<Request>();
    public Admin(String name, String password, String email, String
address, String mobile_num) {
        super(name, password, email, address, mobile_num);
    }
    /**
     * a static function called for creating a new request for a new
playground
     * @param playground
     */
    public static void request(Playground playground){
        Request req = new Request(playground);
        requests.add(req);
```

Software Design Specification

```
    }
    /**
     * Prints all the requested playgrounds
     */
    public void printRequests(){
        for(int i = 0; i < requests.size(); ++i){
            System.out.println("\n\nRequest ID: " +
requests.get(i).getID());
            System.out.println(requests.get(i).getPlayground());
        }
    }
    /**
     * accept a playground in the requests array
     */
    public void accept() throws IDNotFound {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter the ID of the request you want to
accept: ");
        int ID = Integer.parseInt(input.nextLine());

        for(int i = 0; i < requests.size(); ++i){
            if(requests.get(i).getID() == ID){
                requests.get(i).getPlayground().setActive(true);

Database.playgrounds.add(requests.get(i).getPlayground());
                requests.remove(i);
                System.out.println("Playground was approved.");
                return;
            }
        }
        throw new IDNotFound();
    }
    /**
     * main menu screen for admin account
     */
    public void menu(){
        System.out.println("1) Print Requests\n2) Approve a
Playground\n3) Search for an account by name\n4) Sign out");
    }
}
```

LoggedUser.java:

```
package Actor;
```

Software Design Specification

```
import AbstractObjects.Booking;
import AbstractObjects.Database;
import AbstractObjects.IDGenerator;
import AbstractObjects.Wallet;
import java.util.ArrayList;
import java.util.Scanner;
/**
 *  Class to resemble a user that has an account in the database in
general.
 *  Player, PlaygroundOwner and Admin inherit from this class.
 */
public class LoggedUser {
    protected String name;
    protected String password;
    protected int ID;
    protected String email;
    protected String address;
    protected String mobile_num;
    protected Wallet wallet;
    public LoggedUser(String name, String password, String email,
String address, String mobile_num) {
        this.name = name;
        this.password = password;
        this.email = email;
        this.address = address;
        this.mobile_num = mobile_num;
        this.wallet = new Wallet();
        this.ID = IDGenerator.User();
    }
    public LoggedUser() { // empty constructor added to initialize
team captain from a player without generating a new ID
    }
    public String getEmail() {
        return email;
    }
    public String getName() {
        return name;
    }
    public String getPassword() {
        return password;
    }
    public Wallet getWallet() {
        return wallet;
    }
    public int getID() {
```

Software Design Specification

```
        return ID;
    }
    // Allows LoggedUsers to search for an another user's Info using
name.
    // If no matches were found in the three databases nothing will be
printed.
    public void searchUserInfo(){
        System.out.print("Enter the Username of the User you want to
check ID: ");
        Scanner input = new Scanner(System.in);
        String name_input = input.nextLine();
        for (int i = 0; i < Database.admins_accounts.size(); ++i){

if(Database.admins_accounts.get(i).getName().equalsIgnoreCase(name_inp
ut)){
                System.out.println("An Admin with this name was found.
ID: " + Database.admins_accounts.get(i).getID() + ", Email: " +
Database.admins_accounts.get(i).getEmail());
            }
        }
        for (int i = 0; i < Database.playgroundOwners_accounts.size();
++i){

if(Database.playgroundOwners_accounts.get(i).getName().equalsIgnoreCas
e(name_input)){
                System.out.println("A Playground Owner with this name
was found. ID: " + Database.playgroundOwners_accounts.get(i).getID() +
", Email: " + Database.playgroundOwners_accounts.get(i).getEmail());
            }
        }
        for (int i = 0; i < Database.players_accounts.size(); ++i){

if(Database.players_accounts.get(i).getName().equalsIgnoreCase(name_in
put)){
                System.out.println("A Player with this name was found.
ID: " + Database.players_accounts.get(i).getID() + ", Email: " +
Database.players_accounts.get(i).getEmail());
            }
        }
    }
    @Override
    public String toString() {
        return "Name: " + name + "\nEmail: " + email + "\nID: " + ID;
    }
}
```

Software Design Specification

Player.java:

```
package Actor;
import AbstractObjects.Booking;
import AbstractObjects.Database;
import Matches.Playground;
import Matches.Team;
import java.util.ArrayList;
import java.util.Scanner;
import Exception.*;
/**
 * Class to represent a registered Player.
 */
public class Player extends LoggedUser {
    protected Team team;
    protected ArrayList<Team> team_invitations;
    protected ArrayList<Booking> bookings;
    public Player(String name, String password, String email, String
address, String mobile_num) {
        super(name, password, email, address, mobile_num);
        bookings = new ArrayList<Booking>();
        team_invitations = new ArrayList<Team>();
        team = null;
    }
    public Player() { // empty constructor added to initialize team
captain from a player without generating a new ID
    }
    public Team getTeam() {
        return team;
    }
    public ArrayList<Team> getTeam_invitations() {
        return team_invitations;
    }
    /**
     * Book a playground
     * The user enters the id of the playground then the function
searches for the playground in each
     * playground owner account
     */
    public void bookPlayground() throws IDNotFound {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the ID of the playground you want to
book: ");
        int ID = Integer.parseInt(input.nextLine());
```

Software Design Specification

```
        // search if the playground exists and is active
        Playground playground = Database.searchPlayground(ID);
        if(playground == null){
            throw new IDNotFound();
        }
        System.out.print("Enter the month you want to book in(1 - 12):
");
        int month = Integer.parseInt(input.nextLine());
        System.out.print("Enter the day you want to book in(1 - 31):
");
        int day = Integer.parseInt(input.nextLine());
        System.out.print("Enter the start hour you want to book in(0 -
23): ");
        int start_hour = Integer.parseInt(input.nextLine());
        System.out.print("Enter the amount of hours you want to book(0
- 23): ");
        int hours = Integer.parseInt(input.nextLine());
        while(start_hour + hours > 24){
            System.out.println("Invalid booking time. Please reenter
start hour and amount of hours.");
            System.out.print("Enter the start hour you want to book
in(0 - 23): ");
            start_hour = Integer.parseInt(input.nextLine());

            System.out.print("Enter the amount of hours you want to
book(0 - 23): ");
            hours = Integer.parseInt(input.nextLine());
        }
        double cost = playground.getCost_per_hour() * hours;
        System.out.println("The total cost of the booking is: " +
cost);
        if(cost > wallet.getMoney()){
            throw new NotEnoughMoney();
        }
        wallet.setMoney(wallet.getMoney() - cost);

playground.getOwner().wallet.setMoney(playground.getOwner().wallet.get
Money() + cost);
        Booking booking = new Booking(this.ID, ID, day, month,
start_hour, hours);
        Database.bookings.add(booking);
    }
    /**
     * Prints activated playgrounds in the database
     */
```

Software Design Specification

```
    public void printPlaygrounds(){
        for(int i = 0; i < Database.playgrounds.size(); ++i){
            System.out.println(Database.playgrounds.get(i));
        }
    }
    /**
     * Prints playground with location filter entered by the player.
     */
    public void searchPlaygroundsLocation(){
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the location you want to see
playgrounds in: ");
        String location = input.nextLine();
        for(int i = 0; i < Database.playgrounds.size(); ++i){

if(Database.playgrounds.get(i).getLocation().equalsIgnoreCase(location
)) {
                System.out.println(Database.playgrounds.get(i));
            }
        }
    }
    /**
     * Print all team invitations
     */
    public void printInvitations(){
        for (int i = 0; i < team_invitations.size(); ++i){
            System.out.println(team_invitations.get(i));
        }
    }
    /**
     * Creating a new team.
     * The player must not be in a team to create his own team
     * the team creator is automatically assigned as the team captain
     */
    public void createTeam(){
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the name of the Team: ");
        String name = input.nextLine();
        this.team = new Team(name, this);
        System.out.println("Your team was successfully created");
    }
    /**
     * Accepting a team that invited the player
     */
    public void accept(){
```

Software Design Specification

```
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the ID of the team you want to accept:
");
        int ID = Integer.parseInt(input.nextLine());
        for(int i = 0; i < team_invitations.size(); ++i){
            if(team_invitations.get(i).getID() == ID){
                team = team_invitations.get(i);
                team_invitations.get(i).getMembers().add(this);
                team_invitations.clear();
                System.out.println("You joined the team
successfully.");
                return;
            }
        }
        System.out.println("The team with this ID did not invite you
or it doesn't exist.");
    }
    /**
     * Main menu screen for the player
     */
    public void menu(){
        System.out.println("1) Create team\n2) View team invites\n3)
Accept a team invitation\n4) Book a playground\n5) View all
playgrounds\n6) Search playgrounds by location\n7) Check wallet\n8)
Search for an account by name\n9) Sign out");
    }
}
```

PlaygroundOwner.java:

```
package Actor;
import AbstractObjects.Database;
import Matches.Playground;
/**
 * Class to represent a registered Playground Owner.
 * Has all attributes of a LoggedUser + an array of grounds resembling
playgrounds owned.
 */
import java.util.ArrayList;
import java.util.Scanner;
public class PlaygroundOwner extends LoggedUser {
    private ArrayList<Playground> grounds;
    public PlaygroundOwner(String name, String password, String email,
String address, String mobile_num) {
        super(name, password, email, address, mobile_num);
```

Software Design Specification

```
        grounds = new ArrayList<Playground>();
    }
    /**
     * Add a new playground and wait for it to be accepted by an admin
     */
    public void addPlayground() {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the Playground name: ");
        String name = input.nextLine();
        System.out.print("Enter the Playground location: ");
        String location = input.nextLine();
        System.out.print("Enter the Playground booking cost per hour:
");
        double cost_per_hour = Double.parseDouble(input.nextLine());
        System.out.print("Does the playground have natural grass?
(y/n): ");
        String boolean_input = input.nextLine();
        boolean natural_grass;
        if(boolean_input.equalsIgnoreCase("y")){
            natural_grass = true;
        }
        else{
            natural_grass = false;
        }
        System.out.print("Enter the opening day number for the
playground(between [0, 6] if day 0 is Saturday): ");
        int lower = Integer.parseInt(input.nextLine());
        System.out.print("Enter the closing day number for the
playground(between [0, 6] if day 0 is Saturday): ");
        int upper = Integer.parseInt(input.nextLine());
        // loop to iterate and assign true to the days between lower
and upper limit
        boolean[] work_days = new boolean[7];
        for(int i = 0; i < 7; ++i){
            if(i >= lower && i < upper){
                work_days[i] = true;
            }
            else{
                work_days[i] = false;
            }
        }
        System.out.print("Enter the opening hour for the
playground(between [0, 23]if 0 is 12 AM): ");
        lower = Integer.parseInt(input.nextLine());
```

Software Design Specification

```
        System.out.print("Enter the closing hour for the
playground(between [0, 23] if 0 is 12 AM): ");
        upper = Integer.parseInt(input.nextLine());
        // loop to iterate and assign true to the days between lower
and upper limit
        boolean[] work_hours = new boolean[24];
        for(int i = 0; i < 24; ++i){
            if(i >= lower && i < upper){
                work_hours[i] = true;
            }
            else{
                work_hours[i] = false;
            }
        }
        System.out.print("Enter the Playground length in metres: ");
        int length = Integer.parseInt(input.nextLine());
        System.out.print("Enter the Playground width in metres: ");
        int width = Integer.parseInt(input.nextLine());
        Playground playground = new Playground(name, location,
natural_grass, work_days, work_hours, cost_per_hour, length, width,
this);
        grounds.add(playground);
        Admin.request(playground);
        System.out.println("Your request for the playground has been
successfully created.");
        System.out.println("Wait for an admin to accept your request
for the playground to be activated.");
    }
    /**
     * Print all playgrounds owned. whether activated or not.
     */
    public void printPlaygrounds(){
        for(int i = 0; i < grounds.size(); ++i){
            System.out.println(grounds.get(i));
        }
    }
    /**
     * Main menu for the playground owner account
     */
    public void menu(){
        System.out.println("1) Add playground\n2) View owned
playgrounds\n3) Check wallet\n4) Search for an account by name\n5)
Sign out");
    }
}
```

Software Design Specification

TeamCaptain.java:

```
package Actor;
import AbstractObjects.Database;
import Exception.IDNotFound;
import Matches.Team;
import java.util.ArrayList;
import java.util.Scanner;
/**
 * A class to represent a player that has created a team.
 * a team must have one Team Captain only
 */
public class TeamCaptain extends Player {
    public TeamCaptain(Player player) {
        this.name = player.name;
        this.ID = player.ID;
        this.wallet = player.wallet;
        this.password = player.password;
        this.email = player.email;
        this.address = player.address;
        this.mobile_num = player.mobile_num;
        this.bookings = player.bookings;
        this.team = player.team;
        this.team_invitations = new ArrayList<Team>();
    }
    /**
     * Invite a player to the captain's team
     */
    public void invite() throws IDNotFound {
        if(getTeam().getMembers().size() > 10){
            System.out.print("The team is full. you can't invite more
players.");
            return;
        }
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the ID of the player you want to
invite: ");
        int ID = Integer.parseInt(input.nextLine());
        Player player = Database.searchPlayer(ID);
        if(player == null){
            throw new IDNotFound();
        }
        else{
            if(player.getTeam() == null){
```

Software Design Specification

```
                player.getTeam_invitations().add(this.getTeam());
                System.out.println("The invitation was successfully
sent.");
            }
            else{
                System.out.println("The player is already in a team.
Invitation failed.");
            }
        }
    }
    /**
     * Main menu screen for team captain
     */
    @Override
    public void menu() {
        System.out.println("1) Invite player to team\n2) Book a
playground\n3) View all playgrounds\n4) Search playgrounds by
location\n5) Check wallet\n6) Search for an account by name\n7) Sign
out");
    }
}
```

User.java:

```
package Actor;
import AbstractObjects.Database;
import ConsoleUI.Main;
import java.util.Scanner;
/**
 *  A class that is used for any user that has just opened the
program. An unregistered user.
 */
public class User {
    /**
     * Allows the user to register to the program database.
     * An email can only be associated with one account. Thus it is
used to identify duplicate accounts.
     */
    public void register() {
        Scanner input = new Scanner(System.in);
        System.out.println("Do you want to create an account as a:\n1)
Player\n2) Playground Owner");
        int choice = Main.makeChoice(1, 2);
        System.out.print("Enter your Name: ");
        String name = input.nextLine();
```

Software Design Specification

```
        System.out.print("Enter your Password: ");
        String password = input.nextLine();
        System.out.print("Enter your Email: ");
        String email = input.nextLine();
        System.out.print("Enter your Address: ");
        String address = input.nextLine();
        System.out.print("Enter your Mobile Number: ");
        String mobile_num= input.nextLine();
        if(choice == 1){
            if(Database.isDuplicate(email, true)) {
                System.out.println("This email is already registered.
Registration failed.");
            }
            else{
                Player account = new Player(name, password, email,
address, mobile_num);
                Database.players_accounts.add(account);
                System.out.println("The account was successfully
registered. Please log in.");
            }
        }
        else{
            if(Database.isDuplicate(email, false)) {
                System.out.println("This email is already registered.
Registration failed.");
            }
            else{
                PlaygroundOwner account = new PlaygroundOwner(name,
password, email, address, mobile_num);
                Database.playgroundOwners_accounts.add(account);
                System.out.println("The account was successfully
registered. Please log in.");
            }
        }
    }
    /**
     * Login to a player account using email and password
     */
    public Player loginPlayer() {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter your Email: ");
        String email = input.nextLine();
        System.out.print("Enter your Password: ");
        String password = input.nextLine();
        return Database.verifyPlayer(email, password);
```

Software Design Specification

```
    }
    /**
     * Login to an owner account using email and password
     * The only difference between logins is calling a different
database ArrayList
     */
    public PlaygroundOwner loginPlaygroundOwner() {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter your Email: ");
        String email = input.nextLine();
        System.out.print("Enter your Password: ");
        String password = input.nextLine();
        return Database.verifyPlaygroundOwner(email, password);
    }
    /**
     * Login to an admin account
     */
    public Admin loginAdmin() {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter your Email: ");
        String email = input.nextLine();
        System.out.print("Enter your Password: ");
        String password = input.nextLine();

        return Database.verifyAdmin(email, password);
    }
    /**
     * Main menu screen for the User class
     */
    public void menu(){
        System.out.println("1) Register\n2) Login\n3) Exit the
Program");
    }
}
```

Main.java:

```
// This is a console implementation of project GoFo
// The wallet is initialized by default to 1000 pounds to test some
functionalities
// An admin, player and playground owners accounts were added to
Database class for testing
package ConsoleUI;
import AbstractObjects.Database;
```

Software Design Specification

```
import Actor.*;
import Exception.*;
import Matches.Playground;
import java.util.Scanner;
public class Main {
    /**
     *  Function the takes an integer choice from the user and returns
the choice.
     *  Used for main interaction with Console UI.
     *  Has a lower and upper value to force user input to be in the
range
     * @return the choice entered by the player
     */
    public static int makeChoice(int lower, int upper){
        Scanner input = new Scanner(System.in);
        System.out.print("Enter your choice: ");
        int choice = Integer.parseInt(input.nextLine());
        while(choice < lower || choice > upper){
            System.out.println("Please enter a valid number matching
your choice. ");
            System.out.print("Enter your choice: ");
            choice = Integer.parseInt(input.nextLine());
        }
        return choice;
    }
    public static void main(String[] args) throws IDNotFound,
IDNotFound {
        // added some accounts just to test functionalities
        Database.admins_accounts.add(new Admin("a", "a", "a", "a",
"a"));
        Database.players_accounts.add(new Player("a", "a", "a", "a",
"a"));
        Database.playgroundOwners_accounts.add(new
PlaygroundOwner("a", "a", "a", "a", "a"));
        // Welcome messages.
        System.out.println("Welcome to GoFo.");
        System.out.println("You can interact with the UI by entering
the number matching your choice.");
        System.out.println("You're currently unregistered. If you do
not have an account then register.");
        System.out.println("If you have an account then log in.");
        // Main loop of the program.
        User user = new User();
        while(true){
            // menu screen
```

Software Design Specification

```
            user.menu();
            int choice = makeChoice(1, 3);
            // register choice
            if(choice == 1) {
                user.register();
            }
            // login choice
            else if(choice == 2){
                System.out.println("Do you want to log in to a:\n1)
Player account\n2) Playground Owner account\n3) Admin account");
                choice = makeChoice(1, 3);
                // player choice
                if(choice == 1){
                    Player account = user.loginPlayer();
                    if(account == null) {
                        System.out.println("Failed to log in. invalid
credentials.");
                    }
                    // logged in as a player and is the captain of his
team
                    else if((account.getTeam() != null) &&
(account.getTeam().getCaptain().getEmail().equals(account.getEmail()))
){
                        System.out.println("Logged in successfully.");
                        TeamCaptain captain_account = new
TeamCaptain(account);
                        while(true) {
                            captain_account.menu();
                            choice = makeChoice(1, 7);
                            if(choice == 1){
                                captain_account.invite();
                            }
                            else if(choice == 2){
                                captain_account.bookPlayground();
                            }
                            else if(choice == 3){
                                captain_account.printPlaygrounds();
                            }
                            else if(choice == 4){
                                account.searchPlaygroundsLocation();
                            }
                            else if(choice == 5){
                                System.out.println("Your current
balance is: " + account.getWallet().getMoney());
                            }
```

Software Design Specification

```
                                else if(choice == 6){
                                    account.searchUserInfo();
                                }
                                else{
                                    System.out.println("You've signed
out.");
                                    break;
                                }
                            }
                        }
                        // logged in as a player and is not captain of his
team or is not in a team
                        else{
                            System.out.println("Logged in successfully.");
                            while(true){
                                account.menu();
                                choice = makeChoice(1, 9);
                                if(choice == 1){
                                    account.createTeam();
                                    System.out.println("Please log in
again to update your account.");
                                    break; // had to log out a player who
has just created a team to enter the loop of TeamCaptain
                                }
                                else if(choice == 2){
                                    account.printInvitations();
                                }
                                else if(choice == 3){
                                    account.accept();
                                }
                                else if(choice == 4){
                                    account.bookPlayground();
                                }
                                else if(choice == 5){
                                    account.printPlaygrounds();
                                }
                                else if(choice == 6){
                                    account.searchPlaygroundsLocation();
                                }
                                else if(choice == 7){
                                    System.out.println("Your current
balance is: " + account.getWallet().getMoney());
                                }
                                else if(choice == 8){
                                    account.searchUserInfo();
```

Software Design Specification

```
                            }
                            else{
                                System.out.println("You've signed
out.");
                                break;
                            }
                        }
                    }
                }
                // playground owner choice
                else if(choice == 2){
                    PlaygroundOwner account =
user.loginPlaygroundOwner();
                    if(account == null) {
                        System.out.println("Failed to log in. invalid
credentials.");
                    }
                    // logged in as a playground owner
                    else{
                        System.out.println("Logged in successfully.");
                        while(true) {
                            account.menu();
                            choice = makeChoice(1, 5);
                            if(choice == 1){
                                account.addPlayground();
                            }
                            else if(choice == 2){
                                account.printPlaygrounds();
                            }
                            else if(choice == 3) {
                                System.out.println("Your current
balance is: " + account.getWallet().getMoney());
                            }
                            else if(choice == 4){
                                account.searchUserInfo();
                            }
                            else{
                                System.out.println("You've signed
out.");
                                break;
                            }
                        }
                    }
                }
                else{
```

Software Design Specification

```
                    Admin account = user.loginAdmin();
                    if(account == null) {
                        System.out.println("Failed to log in. invalid
credentials.");
                    }
                    // logged in as an admin
                    else{
                        System.out.println("Logged in successfully.");
                        while(true) {
                            account.menu();
                            choice = makeChoice(1, 4);
                            if(choice == 1){
                                account.printRequests();
                            }
                            else if(choice == 2){
                                account.accept();
                            }
                            else if(choice == 3){
                                account.searchUserInfo();
                            }
                            else{
                                System.out.println("You've signed
out.");
                                break;
                            }
                        }
                    }
                }
            }
            // quit the program choice
            else {
                break;
            }
        }
    }
}
```

IDNotFound.java:

```
package Exception;
/**
 * An exception that is thrown whenever an ID is searched but is not
found in the database
 */
public class IDNotFound extends java.lang.Exception {
```

Software Design Specification

```
}
```

NotEnoughMoney.java:

```
package Exception;
/**
 * An exception that is thrown whenever money transaction is made
without enough money in wallet
 */
public class NotEnoughMoney extends java.lang.RuntimeException {
}
```

Playground.java:

```
package Matches;
import AbstractObjects.IDGenerator;
import Actor.PlaygroundOwner;
import java.util.Arrays;
/**
 * A class for a playground. Which players book and playground owners
own.
 */
public class Playground {
    private String name;
    private int ID;
    private String location;
    private boolean isActive;
    private boolean natural_grass;
    private boolean[] work_days;
    private boolean[] work_hours;
    private double cost_per_hour;
    private int length;
    private int width;
    private PlaygroundOwner owner;
    public Playground(String name, String location, boolean
natural_grass, boolean[] work_days, boolean[] work_hours, double
cost_per_hour, int length, int width, PlaygroundOwner owner) {
        this.name = name;
        this.location = location;
        this.natural_grass = natural_grass;
        this.work_days = work_days;
        this.work_hours = work_hours;
        this.cost_per_hour = cost_per_hour;
        this.length = length;
        this.width = width;
```

Software Design Specification

```
        this.isActive = false;
        this.owner = owner;
        this.ID = IDGenerator.Playground();
    }
    public int getID() {
        return ID;
    }
    public double getCost_per_hour() {
        return cost_per_hour;
    }
    public PlaygroundOwner getOwner() {
        return owner;
    }
    public void setActive(boolean active) {
        isActive = active;
    }
    public String getLocation() {
        return location;
    }
    @Override
    public String toString() {
        return "Playground Details\n" +
                "\nName: " + name +
                "\nID: " + ID +
                "\nLocation: " + location +
                "\nActivated: " + isActive +
                "\nNatural Grass: " + natural_grass +
                "\nOpening Days(if first element day is Saturday): " +
Arrays.toString(work_days) +
                "\nOpening hours(if first hour is 12 AM): " +
Arrays.toString(work_hours) +
                "\nHourly Cost: " + cost_per_hour +
                "\nDimensions(in metres): " + length + "x" + width;
    }
}
```

Team.java:

```
package Matches;
import AbstractObjects.IDGenerator;
import Actor.Player;
import Actor.TeamCaptain;
import java.util.ArrayList;
/**
```

Software Design Specification

```
 * A class to represent a team which has 1 captain and up to 10
players(members)
 */
public class Team {
    private String name;
    private int ID;
    private ArrayList<Player> members; // array of players in the team
not including the team captain. maximum number of members is 10
    private TeamCaptain captain; // each team must have a captain
    public Team(String name, Player captain) {
        this.name = name;
        this.members = new ArrayList<Player>();
        this.captain = new TeamCaptain(captain);
        this.ID = IDGenerator.Team();
    }
    public TeamCaptain getCaptain() {
        return captain;
    }
    public int getID() {
        return ID;
    }
    public ArrayList<Player> getMembers() {
        return members;
    }
    @Override
    public String toString() {
        return "Team name: " + name +
                "\nTeam ID: " + ID +
                "\nTeam Captain Email: " + captain.getEmail();
    }
}
```

- Link for Github repository: https://github.com/dudo48/FCAI-CU-Software-Engineering-1
- Link for the video that describes the components of the system: https://www.powtoon.com/c/dtnZ89xeHdW/1/m
- Link for the source code in google drive: https://drive.google.com/file/d/180X52MSz8_6PP4C4JLAmszqcxqKw1Omw/view?usp=sharing