

Sportigo Turf Booking System

Mini Project Report

Submitted by

HILAL HABEEB

Reg. No.: AJC22MCA-2048

In Partial Fulfillment for the Award of the Degree of

**MASTER OF COMPUTER APPLICATIONS
(MCA TWO YEAR)
[Accredited by NBA]**

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY



**AMAL JYOTHI COLLEGE OF ENGINEERING
KANJIRAPPALLY**

[Affiliated to APJ Abdul Kalam Technological University, Kerala. Approved by AICTE, Accredited by NAAC. Koovappally, Kanjirappally, Kottayam, Kerala – 686518]

2023-2024

DEPARTMENT OF COMPUTER APPLICATIONS
AMAL JYOTHI COLLEGE OF ENGINEERING
KANJIRAPPALLY



CERTIFICATE

This is to certify that the Project report, “**Sportigo turf boking system**” is the bonafide work of **Hilal Habeeb (Regno: AJC22MCA-2048)** in partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications under APJ Abdul Kalam Technological University during the year 2023-24.

Ms. Nimmy Francis

Internal Guide

Ms. Meera Rose Mathew

Coordinator

Rev. Fr. Dr. Rubin Thottupurathu Jose

Head of the Department

DECLARATION

I hereby declare that the project report “**Sportigo turf boking system**” is a bonafide work done at Amal Jyothi College of Engineering, towards the partial fulfilment of the requirements for the award of the Degree of Master of Computer Applications from APJ Abdul Kalam Technological University, during the academic year 2023-2024.

Date:

Hilal Habeeb

KANJIRAPPALLY

Reg: AJC22MCA-2048

ACKNOWLEDGEMENT

First and foremost, I thank God almighty for his eternal love and protection throughout the project. I take this opportunity to express my gratitude to all who helped me in completing this project successfully. It has been said that gratitude is the memory of the heart. I wish to express my sincere gratitude to our Manager **Rev. Fr. Dr. Mathew Paikatt** and Principal **Dr. Lillykutty Jacob** for providing good faculty for guidance.

I owe a great depth of gratitude and appreciation towards our Head of the Department **Rev.Fr.Dr. Rubin Thottupurathu Jose** for helping us. I extend my whole hearted thanks to the project coordinator **Ms.Meera Rose Mathew** for her valuable suggestions and for overwhelming concern and guidance from the beginning to the end of the project. I would also express sincere gratitude to my guide **Ms.Nimmy Francis** for her inspiration and helping hand.

I thank our beloved teachers for their cooperation and suggestions that helped me throughout the project. I express my thanks to all my friends and classmates for their interest, dedication, and encouragement shown towards the project. I convey my hearty thanks to my family for the moral support, suggestions, and encouragement to make this venture a success.

HILAL HABEEB

ABSTRACT

The Sportigo Turf Booking System stands as a comprehensive, user-centric software application designed to revolutionize the reservation and management of sports facilities. Engineered with an intuitive interface and seamless functionalities, this system empowers users to effortlessly navigate available time slots, secure reservations, and efficiently administer bookings. By automating administrative tasks, the system optimizes resource allocation and elevates the overall user experience.

Tailored to meet the diverse needs of sports facility owners, managers, and users, this robust solution offers an array of features. Facility owners can easily oversee availability, set pricing structures, and customize booking policies. Managers gain oversight to track reservations, generate insightful reports, and analyze usage trends. Users—comprising coaches, players, and enthusiasts—enjoy a streamlined process to search, book, and receive secure confirmations with online payment options.

The Sportigo Turf Booking System extends beyond mere booking capabilities by facilitating effective coordination and communication among all stakeholders. Facility owners communicate crucial updates, promotions, and announcements directly through the system. Users benefit from real-time booking updates.

With scalable architecture, the system caters seamlessly to sports facilities of varying sizes, from local community fields to expansive multi-sport complexes. Its robust backup mechanisms and encryption protocols ensure data security and high availability. Accessible across web browsers on desktops, laptops, and mobile devices, the system prioritizes flexibility and convenience for users.

Implementing the Sportigo Turf Booking System empowers sports facilities to streamline operations, boost revenue through optimized resource utilization, and deliver a seamless booking experience. Whether it's a local sports club, university campus, or commercial sports complex, this system offers a standardized and efficient approach to sports facility management. Embrace the future of sports facility booking with the Sportigo Turf Booking System.

CONTENT

SL. NO	TOPIC	PAGE NO
1	INTRODUCTION	1
1.1	PROJECT OVERVIEW	2
1.2	PROJECT SPECIFICATION	2
2	SYSTEM STUDY	3
2.1	INTRODUCTION	4
2.2	EXISTING SYSTEM	4
2.3	DRAWBACKS OF EXISTING SYSTEM	5
2.4	PROPOSED SYSTEM	5
2.5	ADVANTAGES OF PROPOSED SYSTEM	6
3	REQUIREMENT ANALYSIS	7
3.1	FEASIBILITY STUDY	8
3.1.1	ECONOMICAL FEASIBILITY	8
3.1.2	TECHNICAL FEASIBILITY	8
3.1.3	BEHAVIORAL FEASIBILITY	9
3.1.4	FEASIBILITY STUDY QUESTIONNAIRE	9
3.2	SYSTEM SPECIFICATION	12
3.2.1	HARDWARE SPECIFICATION	12
3.2.2	SOFTWARE SPECIFICATION	12
3.3	SOFTWARE DESCRIPTION	13
3.3.1	DJANGO	13
3.3.2	SQLITE	13
4	SYSTEM DESIGN	14
4.1	INTRODUCTION	15
4.2	UML DIAGRAM	15
4.2.1	USE CASE DIAGRAM	16
4.2.2	SEQUENCE DIAGRAM	18
4.2.3	STATE CHART DIAGRAM	20
4.2.4	ACTIVITY DIAGRAM	20
4.2.5	CLASS DIAGRAM	23
4.2.6	OBJECT DIAGRAM	24

4.2.7	COMPONENT DIAGRAM	25
4.2.8	DEPLOYMENT DIAGRAM	26
4.3	USER INTERFACE DESIGN USING FIGMA	28
4.4	DATABASE DESIGN	31
5	SYSTEM TESTING	37
5.1	INTRODUCTION	38
5.2	TEST PLAN	38
5.2.1	UNIT TESTING	38
5.2.2	INTEGRATION TESTING	39
5.2.3	VALIDATION TESTING	39
5.2.4	USER ACCEPTANCE TESTING	40
5.2.5	AUTOMATION TESTING	40
5.2.6	SELENIUM TESTING	40
6	IMPLEMENTATION	57
6.1	INTRODUCTION	58
6.2	IMPLEMENTATION PROCEDURE	58
6.2.1	USER TRAINING	58
6.2.2	TRAINING ON APPLICATION SOFTWARE	59
6.2.3	SYSTEM MAINTENANCE	59
7	CONCLUSION & FUTURE SCOPE	60
7.1	CONCLUSION	61
7.2	FUTURE SCOPE	61
8	BIBLIOGRAPHY	62
9	APPENDIX	64
9.1	SAMPLE CODE	65
9.2	SCREEN SHOTS	105

List of Abbreviation

HTML	Hyper Text Markup Language
COBOL	Common Business Oriented Language
UML	Unified Modelling Language
CSS	Cascading Style Sheet
AJAX	Asynchronous JavaScript and XML Environment
GB	GigaBytes
SSD	Solid-State Drive
JS	JavaScript

CHAPTER 1

INTRODUCTION

1.1 PROJECT OVERVIEW

The Turf Booking System is a software application designed to facilitate the reservation and management of sports facilities, such as football fields and cricket courts. This system allows users to conveniently browse available timeslots, make reservations, manage bookings, and handle administrative tasks related to sports facility usage. The system aims to enhance user experience, optimize resource utilization, and facilitate efficient coordination among all stakeholders involved in the sports facility booking process.

1.2 PROJECT SPECIFICATION

The Sportigo Turf Booking System is a web-based software application that facilitates the reservation and management of sports turf facilities. It aims to provide a seamless and efficient booking experience for both Turf Providers and Normal Users. The system will be owned and managed by an Admin who has full control over the platform's functionalities.

1. User Registration and Authentication:

- Normal Users and Turf Providers must be able to register and create user accounts.
- The system should support secure authentication mechanisms, such as username/password or social media login options.

2. User Roles and Permissions:

- The Admin should have exclusive access to administrative features.
- Turf Providers should be able to manage their own turf listings and associated information.
- Normal Users should have the ability to search, view, and make reservations for turfs.

3. Turf Provider Management:

- Turf Providers should be able to create and manage their turf profiles, including facility details, pricing, and availability.
- Turf Providers should have the ability to update, delete, and deactivate turf listings.

4. Turf Booking:

- Normal Users should be able to search for available turfs based on location, date, time, and other filters.

- Users should be able to view detailed information about each turf, such as facilities, amenities, and reviews.
 - Normal Users should be able to select a turf, choose a date and time slot, and make a reservation request.
 - Users should receive confirmation and booking details after a successful reservation.
5. Payment Integration:
- The system should integrate with a reliable and secure payment gateway to facilitate online payments for turf bookings.
 - Users should have the option to securely store payment information for future use.
6. Notification and Communication:
- The system should send email notifications to users for booking confirmations, updates, and reminders.
7. Reporting and Analytics:
- The Admin should have access to a comprehensive dashboard with reports and analytics on turf bookings, revenue, and user activity.
 - Turf Providers should have access to basic reports on their turf bookings and revenue.

CHAPTER 2

SYSTEM STUDY

2.1 INTRODUCTION

In an era witnessing the digitization of numerous industries, the management of sports facilities calls for innovative solutions. The Sportigo Turf Booking System emerges as a comprehensive response to this need, envisioning a software platform that seamlessly connects administrators, turf providers, and users in the realm of sports facility booking and management. This system aims to transcend the traditional manual booking methods, offering a suite of functionalities to streamline operations for stakeholders and enhance the booking experience for sports enthusiasts.

2.2 EXISTING SYSTEM

The current landscape of sports facility management relies predominantly on manual and decentralized booking systems. These outdated methods result in inefficiencies and limitations in effectively managing sports amenities. The lack of a centralized platform hampers real-time updates, impedes efficient resource utilization, and presents challenges in facilitating seamless communication between stakeholders. Moreover, the absence of a centralized platform exacerbates the challenges faced by facility managers, users, and organizers. Facility owners struggle with the cumbersome task of managing bookings across various channels, leading to inconsistencies in availability updates and pricing discrepancies. Users experience inconveniences due to limited visibility into real-time turf availability and the absence of transparent booking procedures.

2.2.1 NATURAL SYSTEM STUDIED

The Sportigo Turf Booking System functions as a dynamic natural system, blending technological infrastructure with user behaviors and market forces. Technological components, including databases and user interfaces, form the backbone of the system, enabling streamlined booking management and fostering user interactions. User behaviors and preferences significantly influence the system's responsiveness to evolving market trends and fluctuations in sports facility usage patterns.

2.2.2 DESIGNED SYSTEM STUDIED

The meticulously designed Sportigo Turf Booking System prioritizes user-centric features, catering to the distinct needs of administrators, turf providers, and users. Administrators gain access to robust tools for efficient facility management, while turf providers can optimize resource allocation and tailor booking policies. Users are provided with an intuitive interface facilitating the effortless browsing of available time slots, secure payment processing, and simplified booking procedures. This system endeavors to revolutionize sports facility management by amalgamating enhanced user experiences with streamlined operational efficiency.

2.3 DRAWBACKS OF EXISTING SYSTEM

- **Limited Availability:** Comprehensive sports facility booking platforms are scarce, hindering efficient management.
- **Manual Systems Inefficiency:** Manual systems result in delays, lack of real-time updates, and potential errors in bookings.
- **Underutilization of Resources:** Inadequate optimization of sports amenities leads to potential wastage of available resources.
- **Communication Challenges:** Lack of convenient communication channels among stakeholders causes mismanagement and delays in operations.
- **Scalability and Access:** Current systems might not be scalable enough to accommodate increasing user demands and may lack mobile responsiveness.

2.4 PROPOSED SYSTEM

The proposed Sportigo Turf Booking System aims to bridge these gaps by providing a centralized platform for efficient sports facility management. This comprehensive solution will offer real-time updates, streamlined booking processes, and enhanced communication channels among administrators, turf providers, and users. The focus will be on optimizing resource utilization and improving user experiences across various sports amenities.

2.5 ADVANTAGES OF PROPOSED SYSTEM

- **Centralized Platform:** Provides a centralized and user-friendly interface for convenient sports facility bookings.
- **Real-time Updates:** Ensures real-time updates and notifications for users and administrators.
- **Optimized Resource Utilization:** Enhances efficient allocation and utilization of sports amenities.
- **Improved Communication:** Facilitates seamless communication channels among stakeholders for effective coordination.
- **Enhanced User Experiences:** Streamlines processes and enhances user experiences for booking sports facilities.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY

Feasibility study for the Sportigo Turf Booking System is a comprehensive assessment aimed at evaluating the project's viability and potential success. This study encompasses various aspects, including technical, economic, legal, operational, and scheduling feasibility. It aims to provide valuable insights into potential challenges and opportunities, ensuring informed decision-making and successful execution of the Sportigo Turf Booking System.

3.1.1 Economic Feasibility

The Sportigo Turf Booking System presents a promising business opportunity in the sports facility management sector. The project, while entailing significant development costs, holds substantial revenue potential through user subscriptions, service fees from turf providers, and potential advertising avenues. A large market for sports facility bookings and efficient resource utilization can lead to sustainable revenue streams. Success hinges on attracting a substantial user base and establishing partnerships with sports facilities.

Key Points:

- Lucrative market for sports facility management
- Revenue generation through user subscriptions, service fees, and potential advertising
- Potential for sustained profitability with a significant user base and optimized resource utilization
- Success dependent on effective marketing strategies and user acquisition.

3.1.2 Technical Feasibility

The technical feasibility of the Sportigo Turf Booking System is promising, employing established technologies such as HTML, CSS, JavaScript, and backend frameworks for web development. The development team possesses expertise in web development, database management, and security protocols, ensuring a robust and reliable platform. The system is designed to be scalable, adaptable across devices, and capable of integration with various platforms. Rigorous testing protocols guarantee the identification and resolution of technical issues, ensuring a stable and resilient platform. With the requisite infrastructure and support, the Sportigo Turf Booking System can be effectively deployed, meeting the technical requisites for seamless sports facility management.

3.1.3 Operational Feasibility

Operational feasibility for the Sportigo Turf Booking System is assured by the high demand for streamlined sports facility booking processes among users, turf providers, and administrators. The project has allocated adequate resources and adopted cost-effective measures to meet technical requirements. User-friendly interfaces, training resources, and responsive support ensure straightforward adoption. The system integrates seamlessly into existing processes, scales efficiently, and effectively addresses potential risks. The Sportigo Turf Booking System is poised to benefit the sports community by simplifying sports facility bookings, optimizing resource utilization, and fostering efficient sports facility management practices.

3.1.4 Feasibility Study Questionnaire

Project Overview

Response: The Turf Booking System project aims to modernize and simplify the process of booking sports turfs. It provides a user-friendly web and mobile platform for sports enthusiasts, clubs, and turf providers to book and manage turf reservations. This system aims to enhance the accessibility and convenience of booking sports facilities, benefiting both users and providers.

To what extent the system is proposed for?

Response: The system is proposed for full-scale deployment to serve a wide range of sports enthusiasts, clubs, and turf providers in our target region. It is designed for long-term use and scalability as the user base grows.

Specify the Viewers/Public which is to be involved in the System?

Response: The system is intended for use by the following groups:

- Regular Users: Individuals looking to book sports turfs for personal use.
- Club Users: Clubs or organizations requiring turf bookings for extended periods.
- Admins: System administrators responsible for user management and system oversight.
- Turf Providers: Entities offering sports turfs for booking.

List the Modules included in your System?

Response: The Turf Booking System comprises the following key modules:

1. User Management
2. Turf Listing and Management

3. Booking and Reservation

4. Admin Dashboard

5. Club Booking (for extended reservations) Identify the users in your project?

Response: The primary user roles in our project are as follows:

- Regular Users
- Club Users
- Admins
- Turf Providers

Who owns the system?

Response: The Turf Booking System is owned and managed by our organization(sportigo), responsible for its development, operation, and maintenance.

System is related to which firm/industry/organization?

Response: The system is related to the sports and recreation industry, specifically targeting sports enthusiasts, sports clubs, and turf providers in our region.

Details of the person you have contacted for data collection?

Response: We have contacted Firoz Muhammed, a representative from a local sports facility management organization. This contact has provided valuable insights and data regarding turf booking requirements, user expectations, and industry standards.

Questionnaire to Collect Details About the Project :

What is the primary objective of the Turf Booking System project?

Response: The primary objective is to create a user-friendly platform for booking sports turfs, improving accessibility and convenience for sports enthusiasts and clubs.

Who are the primary users of the system, and what are their specific needs and roles?

Response:

- Regular Users: Need a simple booking process and visibility into turf availability.
- Club Users: Require extended booking options and reservation management.

- Admins: Need tools for user management and system oversight.
- Turf Providers: Want to list and manage their turf offerings.

What modules or components have been identified for inclusion in the system, and what are their key functionalities?

Response:

- User Management: User registration, authentication, and profile management.
- Turf Listing and Management: Turf providers can list, edit, and manage their turf listings.
- Booking and Reservation: Users can view availability, book turfs, and receive confirmations.
- Admin Dashboard: Admins can manage user accounts, verify turf listings, and handle issues.

How is data collected and managed within the system, and what types of data are stored?

Response: Data is collected through user input and stored securely in a database. User data includes usernames, emails, and booking history. Turf data includes listings, availability, and pricing.

Are there any specific industry standards or regulations that the system needs to comply with?

Response: Yes, the system will comply with data privacy laws and regulations regarding online transactions and user data protection.

What technologies or frameworks are being used to develop the Turf Booking System?

Response: We are using web technologies such as HTML, CSS, JavaScript for the frontend, and a Python-based Django framework for the backend.

Is there a specific timeline or deadline for the project's completion?

Response: Yes, the project is planned for completion within 6months.

What are the key challenges or risks associated with the project, and how do you plan to mitigate them?

Response: Challenges include competition, data security, and user adoption. We plan to address these through targeted marketing, robust security measures, and user-friendly design.

Have you identified any potential areas for future enhancements or expansion of the system?

Response: Yes, potential areas for expansion include integrating additional payment methods and adding support for different types of sports facilities.

3.2 SYSTEM SPECIFICATION

3.2.1 Hardware Specification

- Processor - Intel Core i5
- RAM - 8 GB or higher
- Hard disk - 256 GB SSD or higher

3.2.2 SOFTWARE SPECIFICATION

- Front End - HTML5, CSS3, Bootstrap, JavaScript, jQuery
- Back End - Django (Python)
- Database - SQLite
- Client on PC - Windows 7 and above.
- Technologies used - JS, HTML5, AJAX, JQuery, Python, CSS, Django

3.3 SOFTWARE DESCRIPTION

3.3.1 Django

Django serves as a high-level web framework in Python, streamlining the creation of secure and easily maintainable websites. Developed by skilled professionals, Django simplifies many complex aspects of web development, allowing you to focus on building your application without the need to start from scratch. It's an open-source framework, meaning you can use it without any financial obligations. With a vibrant community, thorough documentation, and free or paid support options, Django was created to expedite the development of sophisticated web applications that heavily depend on data.

3.3.2 SQLite

SQLite, a popular choice for web development, is a self-contained, serverless, and open-source relational database management system. Its unique design allows it to operate without the need for a separate server process, making it an excellent option for local or client-side storage in web applications. With its widespread use and reputation for simplicity and reliability, SQLite is renowned for its seamless integration capabilities. It efficiently handles data in various formats, making it a versatile and user-friendly solution for managing your website's information. Its robust features ensure optimal performance, data integrity, and scalability, providing a solid foundation for your web development needs.

CHAPTER 4

SYSTEM DESIGN

4.1 INTRODUCTION

In the realm of product and system development, the design phase marks the initial and critical stage. This creative process is instrumental in ensuring the seamless and effective functioning of any system. Design, in this context, refers to the meticulous application of diverse techniques and principles to meticulously outline a process or system, facilitating its tangible realization. It involves the comprehensive description of the intricacies of a machine or system to enable its accurate implementation. In software development, the design phase holds exceptional significance, serving as the cornerstone of creating efficient and precise software solutions. System design, specifically, involves meticulous planning for the construction of a well-optimized and reliable software framework. It signifies a shift from user-focused documentation to a more programmer-oriented approach, outlining the logical and physical design aspects of the system. This meticulous software design process ensures the seamless integration of functionality and precision, laying the foundation for a robust and user-friendly final product.

4.2 UML DIAGRAM

UML, the Unified Modeling Language, is an industry-standard visual language utilized for specifying, constructing, and documenting software system artifacts. It was established by the Object Management Group (OMG), with the initial UML 1.0 draft presented in January 1997. Unlike traditional programming languages such as C++, Java, or COBOL, UML serves as a graphical language, facilitating the creation of software blueprints. Its primary function is to provide a versatile and comprehensive modeling approach, enabling the visualization, specification, construction, and documentation of software systems. While UML finds extensive use in modeling software systems, its applications are not confined to this domain exclusively, extending to areas like process flows in manufacturing units and beyond. While UML itself is not a programming language, it supports code generation in various programming languages through UML diagrams.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Activity diagram
- State chart diagram

- Deployment diagram
- Component diagram

4.2.1 USE CASE DIAGRAM

Explanation

A use case is instrumental in identifying and organizing the essential requirements of a system, facilitating a structured approach to manage and prioritize these elements. Applied within the Unified Modeling Language (UML), use case diagrams serve as visual representations of real-world systems. They enable the comprehensive modeling of diverse systems, outlining their key functionalities and interactions. A use case diagram generally comprises four fundamental components, contributing to a clear and concise depiction of system requirements and behavior.

Use case diagrams aid in the documentation of a system's functional specifications. They demarcate the system's boundaries, delineating its distinctions from surrounding elements. Actors represent the individuals involved in distinct roles within the system. These diagrams facilitate a comprehensive understanding of the interactions between various entities or components within a specific context or problem. Constructing a proficient use case diagram necessitates adherence to certain rules. Ensuring appropriate nomenclature for use cases and actors is vital. The diagram should clearly illustrate relationships and dependencies while only displaying the essential connections for optimal diagram functionality. Supplementary notes may be utilized when necessary to elucidate critical points.

Diagram

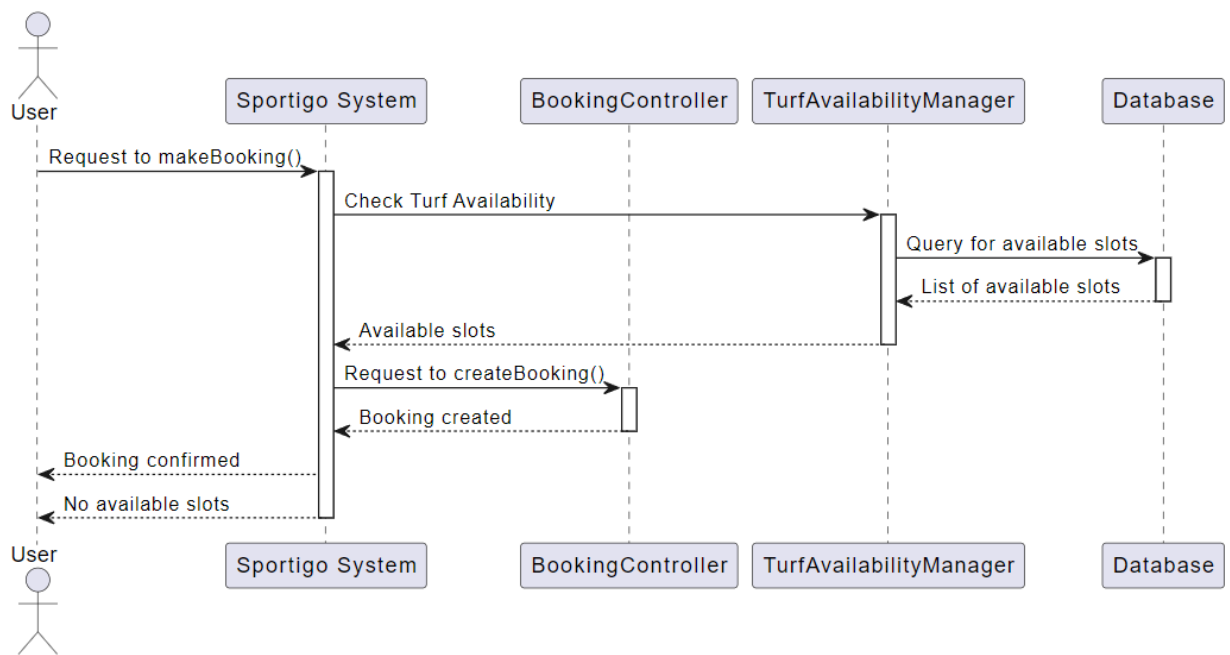
4.2.1 SEQUENCE DIAGRAM

Explanation

A sequence diagram serves to illustrate the systematic interaction between objects in a predetermined order. It elucidates the chronological flow of activities within a system. Referred to as event diagrams or event scenarios, sequence diagrams provide a comprehensive overview of the interplay among various system components, showcasing the specific sequence of their actions. These diagrams are valuable tools for both business professionals and software developers, enabling them to comprehend and depict the requirements of both new and existing systems effectively.

Notations in Sequence Diagrams:

- i. **Actors:** Represented as non-system individuals who utilize the system, actors play distinct roles within the diagram, depicted as simple stick figures.
- ii. **Lifelines:** Each component within the system is denoted as a lifeline at the top of the sequence diagram.
- iii. **Messages:** Objects communicate through the exchange of messages, depicted as arrows following the order of the lifeline. These messages form the core components of the sequence diagram.
- iv. **Guards:** Utilized to define various conditions, guards restrict message flow based on specific conditions. They provide guidelines for software developers, outlining the rules and processes within the system.

Diagram

4.2.2 State Chart Diagram

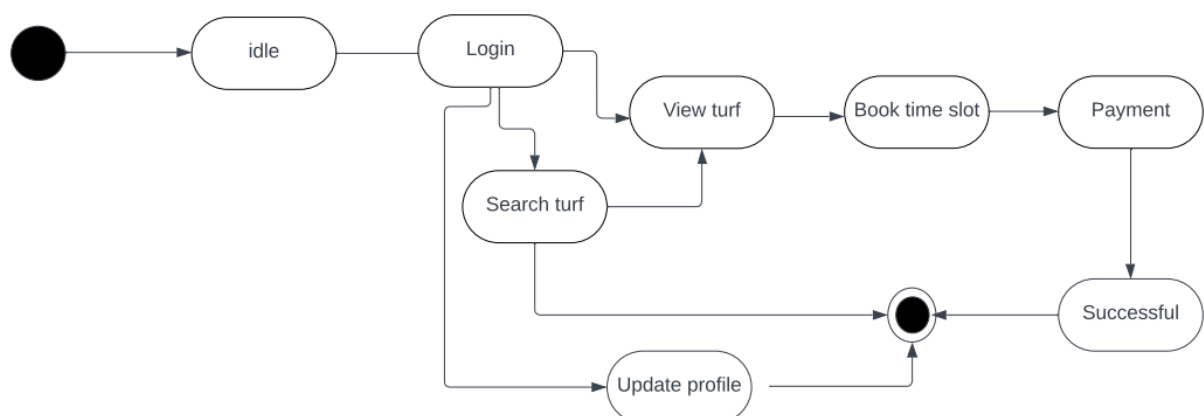
Explanation

The state machine diagram, also known as a state chart, effectively represents the various states of an object within a system and their sequential transitions. It provides a comprehensive overview of the system's behavioral patterns, delineating the object's progression through different states. It is instrumental in illustrating the interworking dynamics of various elements within a system, be it a group of individuals, a team, a larger collective, or an entire organization. The state machine diagram serves as a strategic blueprint, elucidating how objects evolve during different events and encapsulating the specific states that each element assumes within the system.

Notations in a State Machine Diagram:

- **Initial state:** Signifying the system's initiation, this notation is depicted as a black circle.
- **Final state:** Denoting the system's culmination, this notation is depicted as a filled circle within another circle.
- **Decision box:** Shaped like a diamond, it aids in making decisions based on the evaluation of a guard.
- **Transition:** Represents the shift of power or control from one state to another, illustrated as an arrow with a label indicating the cause of the transition.
- **State box:** Reflects the current status of an element within a group, represented as a rectangular shape with rounded corners.

Diagram



4.2.2 Activity Diagram

Explanation

The activity diagram visually portrays the sequential or simultaneous occurrence of events and activities, effectively illustrating how tasks unfold in a predetermined order. It serves as a comprehensive representation of the flow of activities, demonstrating the interconnectedness of different processes. The activity diagram showcases a variety of flows, encompassing actions occurring sequentially, in parallel, or along distinct pathways. To facilitate these dynamic flows, activity diagrams are equipped with tools such as fork and join, enabling the depiction of multiple simultaneous activities. This type of diagram is often referred to as an object-focused illustration, emphasizing the systematic portrayal of a process's workflow and operational intricacies.

Components of an Activity Diagram:

- a) Activities:** These refer to the grouping of behaviors into a sequence of actions, visually represented as interconnected nodes. The actions progress from the initial point to the end point, outlining the various tasks, control mechanisms, and resources utilized throughout the process.
- b) Activity partition / swim lane:** This organizational tool groups similar tasks into distinct categories, enhancing the modularity and comprehensibility of the activity diagram. It aids in the categorization and visualization of different activity sets, offering a clear representation of their interdependencies and interactions.
- c) Forks:** Fork nodes enable the concurrent execution of various components of a task, allowing the simultaneous progression of multiple activities. They facilitate the divergent flow of information, ensuring that data is dispersed efficiently across different pathways.
- d) Join Nodes:** Distinguished from fork nodes, join nodes enable the convergence of data from multiple sources, consolidating the incoming data streams into a unified output. These nodes serve to synchronize and coordinate the flow of information within the activity diagram.

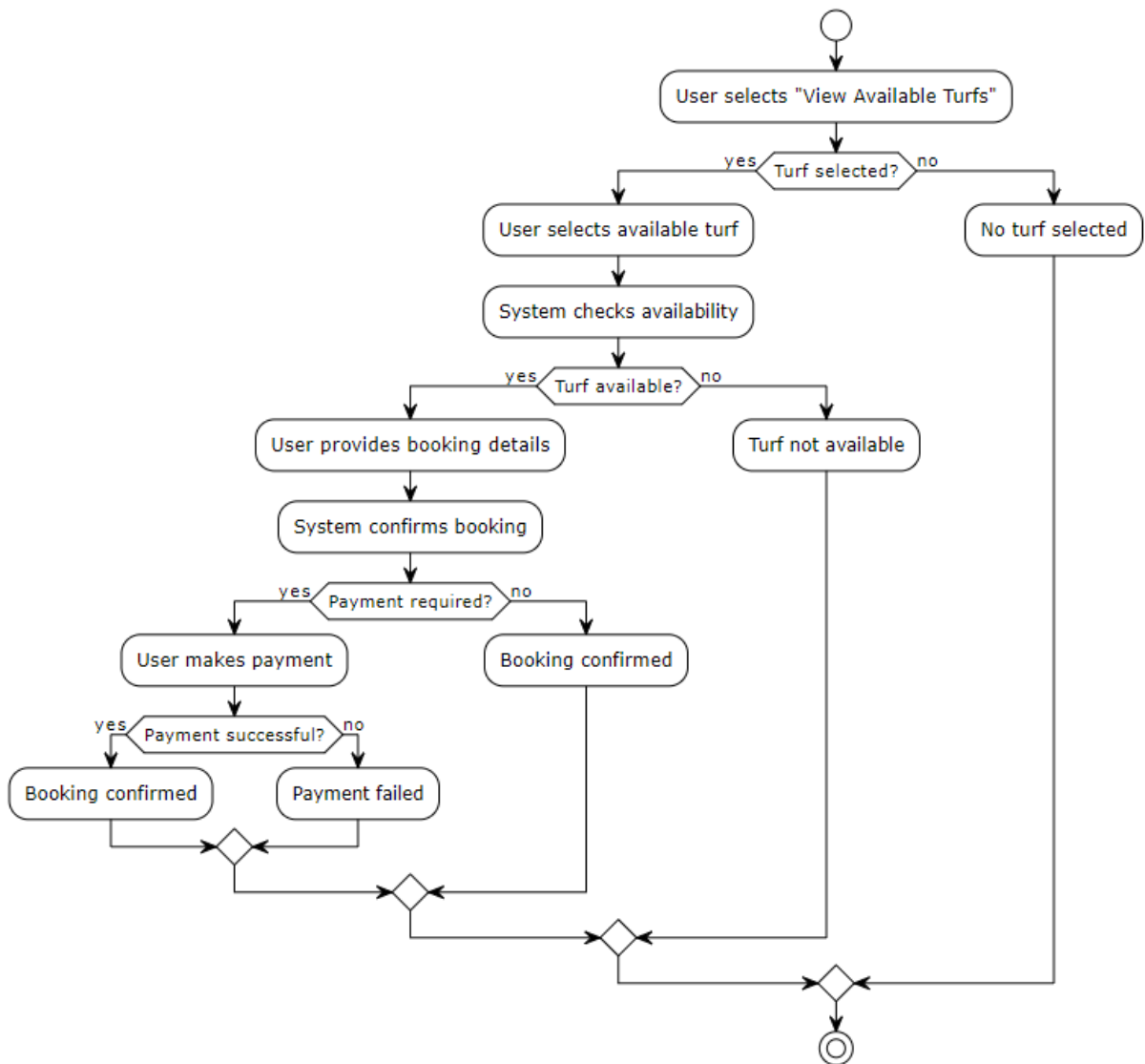
Notations in an Activity Diagram:

Initial State: Signifies the commencement or initial step of a process.

Final State: Represents the conclusion or endpoint of a process, indicating the completion of all associated activities.

Decision Box: Ensures that the progression of activities adheres to a predetermined course.

Action Box: Denotes the specific tasks or actions that need to be executed as part of the overall process

Diagram

4.2.3 Class Diagram

Explanation

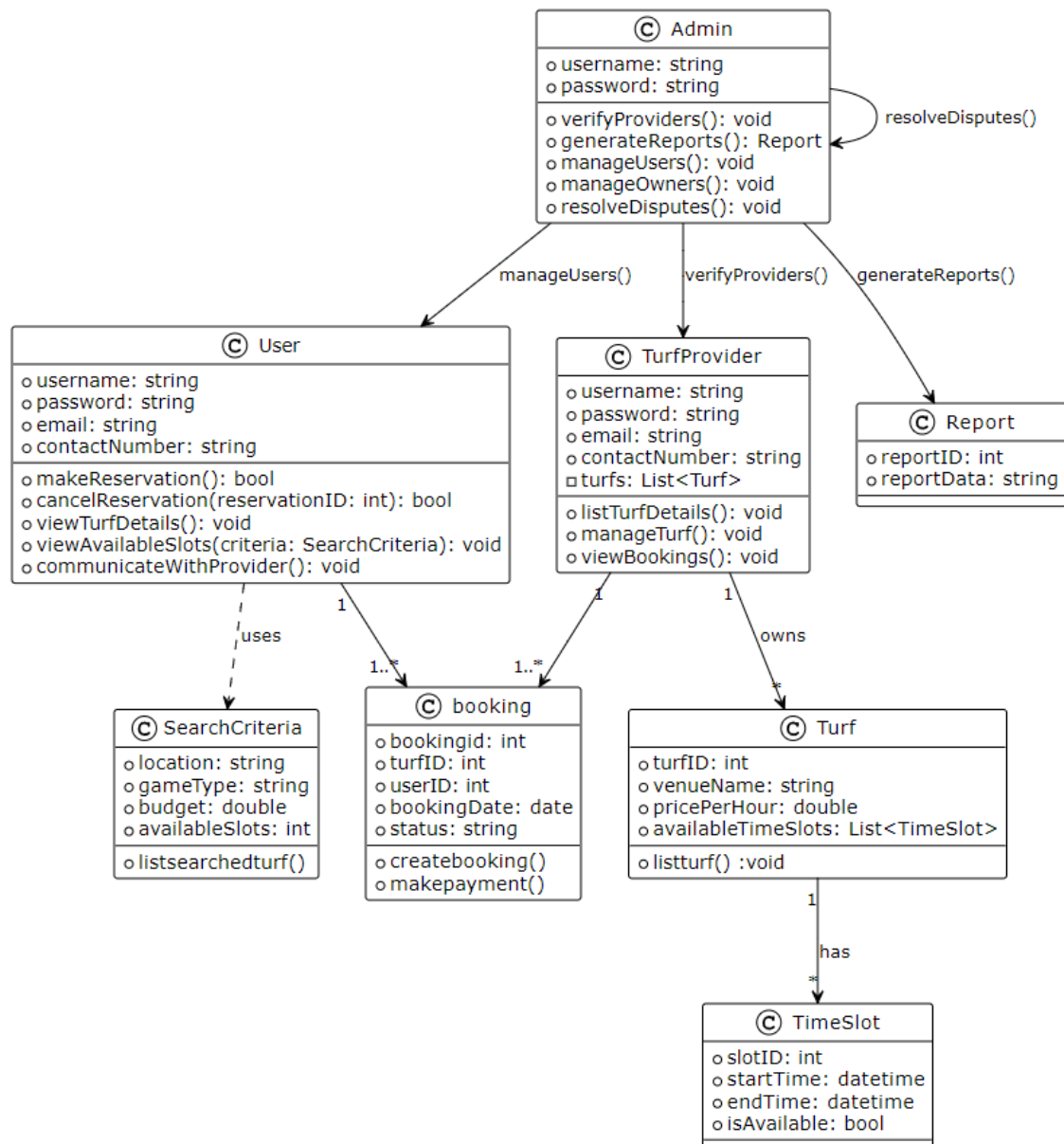
The class diagram is a static representation that illustrates the constituent elements and their associations within the system, providing a visual overview of its structure and components. It serves as a blueprint for organizing various attributes, relationships, and actions, streamlining the software development process by encapsulating critical information about the system's design and functionality. Essentially, a class diagram presents a holistic depiction of the system, offering insights into its composition, interdependencies, and behavior through a structured representation.

Components of a Class Diagram:

- **Upper Section:** This segment entails the class name, serving as a distinct identifier for a group of similar entities within the system. The class name is highlighted in bold letters, positioned centrally at the top. For an abstract class, the title is presented in slanted writing style, emphasizing its abstract nature.
- **Middle Section:** This section delineates the class attributes, demonstrating its properties and visibility factors. Attributes are depicted alongside their corresponding visibility indicators, such as public (+), private (-), protected (#), and package (~), reflecting the accessibility levels of each attribute.
- **Lower Section:** The lower segment encompasses the class methods or operations, outlined in a comprehensive list format. Each method is depicted as a discrete line item, emphasizing the interactions between the class and its associated data.
- Within the context of UML, the relationships depicted in the class diagram adhere to specific conventions:
 - **Dependency:** Signifies the impact of one element's changes on another.
 - **Generalization:** Illustrates the hierarchical relationship between classes, where one class acts as a parent while another assumes the role of the child.
 - **Association:** Indicates the connections between different elements within the system.
 - **Multiplicity:** Sets constraints on the permissible quantities associated with specific attributes or relationships.

- **Aggregation:** Represents a collection or group forming part of an association.
- **Composition:** Describes a specialized form of aggregation, emphasizing the interdependence between parent and child elements.

Diagram

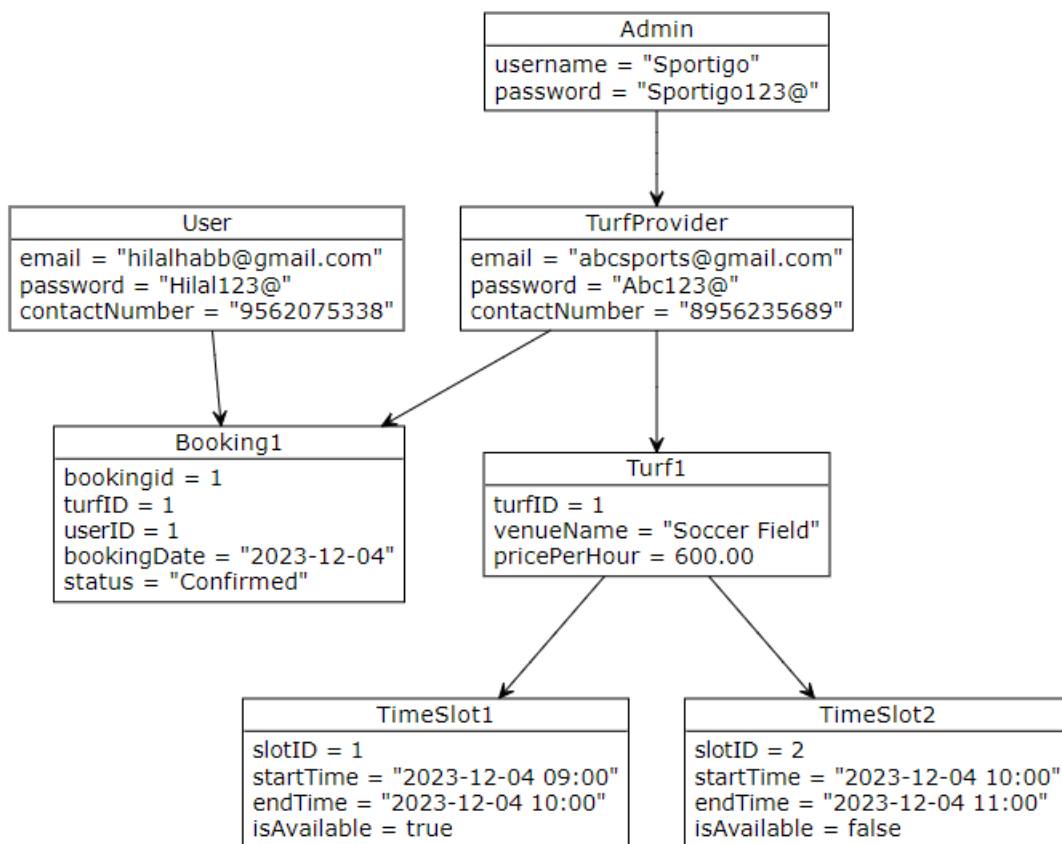


4.2.4 Object Diagram

Explanation

Object diagrams are closely related to class diagrams and are derived from them. They offer a snapshot of a specific moment within an object-based system, illustrating a group of elements represented by a class. While they share similarities with class diagrams, object diagrams provide a more concrete depiction by showcasing specific instances of objects at a particular instant, enhancing the comprehension of the system's functionality and structure.

Diagram



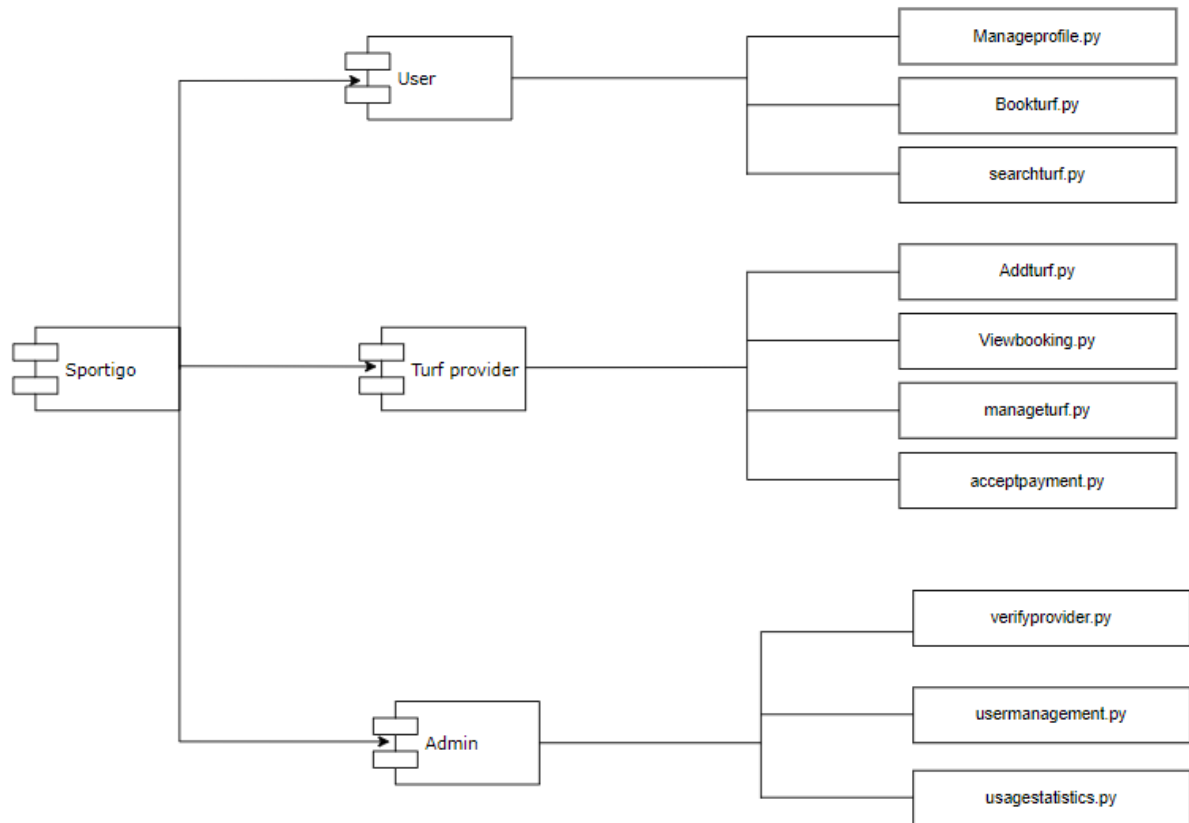
4.2.5 Component Diagram

Explanation

A component diagram is instrumental in partitioning a complex system composed of objects into more manageable segments. It provides an overview of the internal components such as programs, documents, and tools within the node. By illustrating the connections and organization

of elements in the system, it facilitates the creation of a usable system. Components are discreet sections of a system capable of autonomous function and conceal their internal operations, akin to a confidential box that operates only when used correctly.

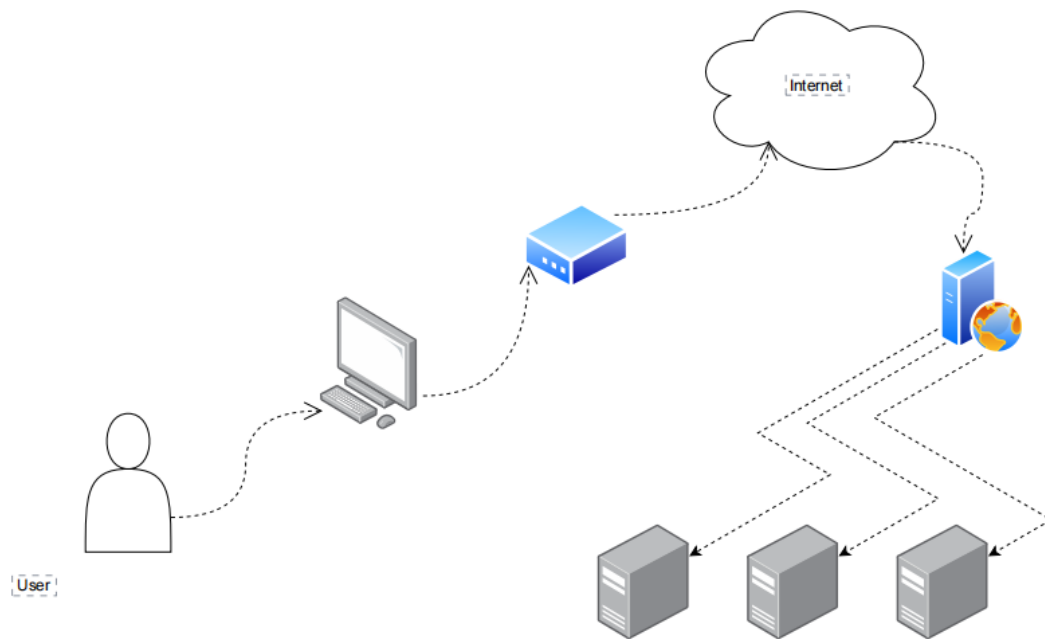
Diagram



4.2.8 Deployment Diagram

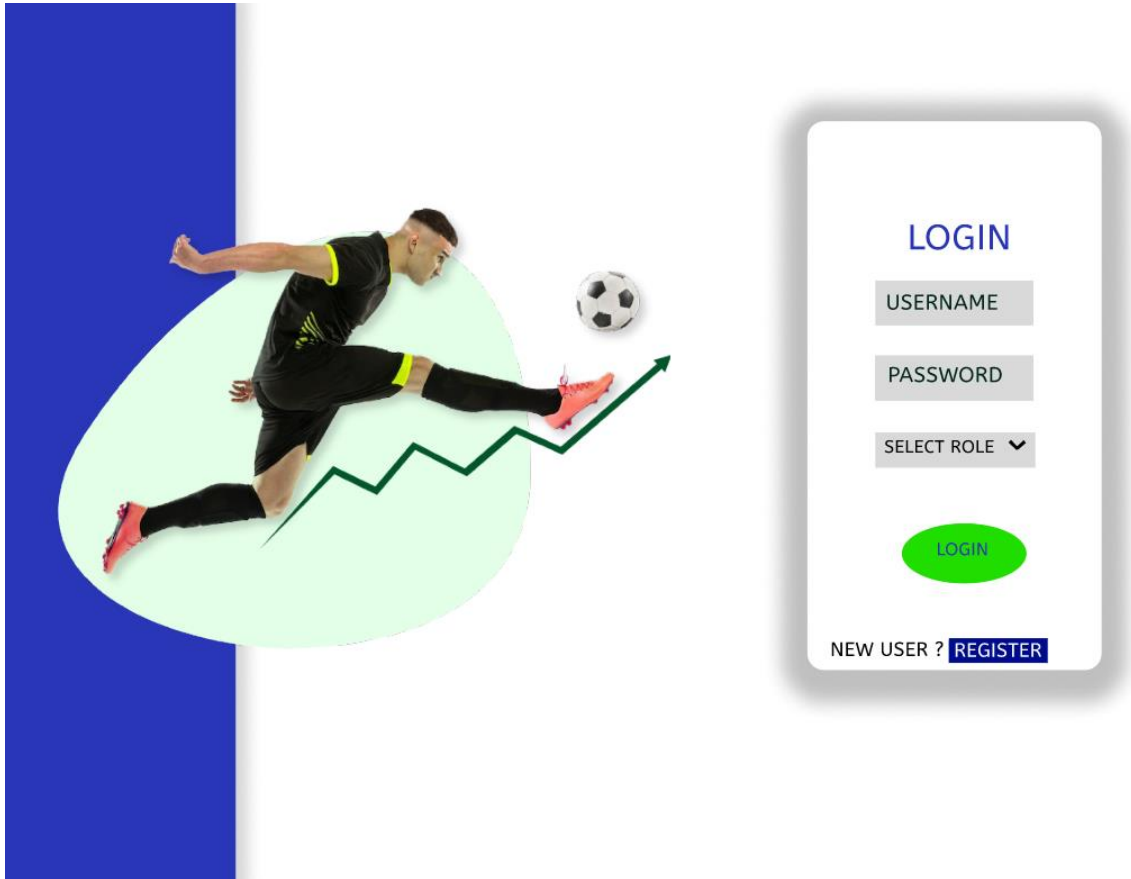
Explanation

The deployment diagram visually represents the placement of software components within the physical computer or server. It provides insights into the static arrangement of the system's elements, including nodes and their interconnections. The diagram outlines the implementation of software on the computer system, detailing the architecture's composition and organization.


Diagram

4.3 USER INTERFACE DESIGN USING FIGMA

Form Name: Login



Form Name: Home



USER PROFILE
ID : 14253658613

Dashboard


My Account

MY BOOKINGS

COMPARE

Your Account Details

Avialabe Turf




Old traffod

★ ★ ★

26THMILE, KANJIRAPPALLY

BOOK NOW




Camp nou

★

PETTA JN, KANJIRAPPALLY

BOOK NOW



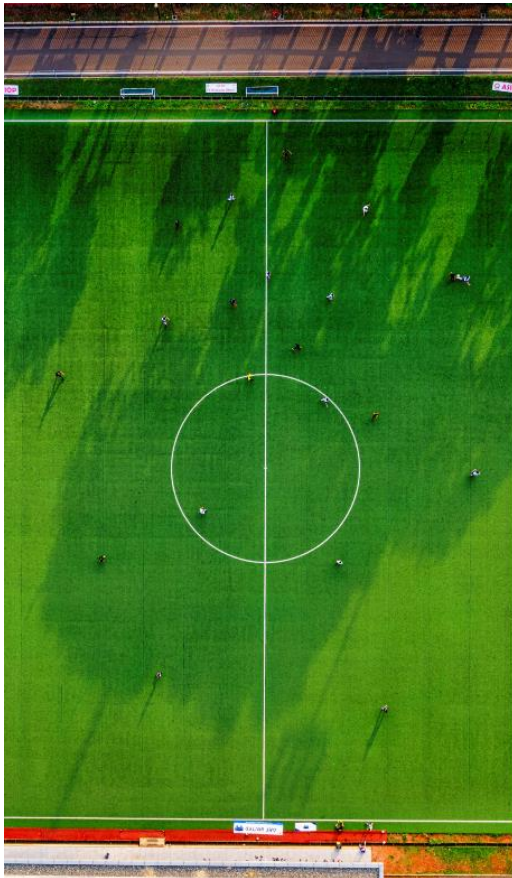
Alliance

★ ★ ★

NADAKKAL, ERATTUPETTA

BOOK NOW

© Dashboard, All rights reserved

Form Name: Registration

REGISTER

FIRST NAME

LAST NAME

EMAIL ID

PASSWORD

REGISTER

ALREADY HAVE AN ACCOUNT? [LOGIN](#)

4.4 DATABASE DESIGN

4.4.1 Relational Database Management System (RDBMS)

- A relational database management system (RDBMS) is a structured system designed to efficiently store and retrieve information, ensuring its accessibility and security for users.
- Developing a database involves understanding user requirements and tailoring the database system to meet their specific needs.
- The process begins with creating an organizational plan for the data, independent of any particular computer program.
- Subsequently, this plan is translated into a specific design for the chosen computer program, focusing on the effective storage of information within the database system.
- Key aspects in this process include ensuring data integrity, maintaining the accuracy and consistency of data, and achieving data independence, allowing data to be accessed without being affected by changes in the database schema or structure.

4.4.2 Normalization

Normalization is the process of organizing a database into tables and ensuring that the data is efficiently stored and free from redundancy. It involves structuring the tables to minimize data duplication and dependency. Each table consists of rows and columns, where a row, also known as a tuple, represents a specific data set, and a column header, termed an attribute, identifies the type of data being stored. In the context of the relational model, a group of interrelated tables constitutes a relational database, facilitating effective management and utilization of data.

4.4.3 Sanitization

Django incorporates several pre-existing tools for data cleansing. It provides a variety of field types that come with built-in validation and cleaning capabilities. For instance, the CharField automatically verifies and purifies text input, while the EmailField ensures the correct formatting of email addresses. These field types serve to prevent the storage of harmful or invalid data within the database. Moreover, Django emphasizes the use of form validation to sanitize user input. Its forms are equipped with predefined validation methods and validators that can be applied to different form fields. These validators conduct various checks and data cleansing procedures, such as confirming that numeric values fall within specified ranges and verifying the formats of uploaded files.

4.4.4 Indexing

Indexing involves organizing specific data or data groups in a database in a structured order based on their values. This ordered arrangement of index entries facilitates the quick and efficient retrieval of exact matches or data falling within a particular range. By using indexes, users can easily access information in a database without the need to search through every record during database operations. An index acts as a navigational tool for locating information within a database, enabling swift data lookup and facilitating the identification of records based on specific ordering. Additionally, an index can be established based on one or multiple columns within the table.

4.5 TABLE DESIGN

Usertable

Primary key: email

No	Field Name	Datatype	Key Constraints	Description of the field
1	username	CharField	Optional	Username for the user
2	role	CharField	Default: "normal_user"	Role of the user
3	<u>email (PK)</u>	EmailField	Primary key, Unique	Unique identifier for the user
4	dob	DateField	Default: '2000-01-01'	Date of birth for the user
5	phone_number	CharField	NULL	User's phone number
6	Password	CharField	NOT NULL	Login password

TurfProvider

Primary key: email

No	Field Name	Datatype	Key Constraints	Description of the field
1	venue_name	CharField	NOT NULL	Name of the venue provided by TurfProvider
2	email (PK)	EmailField	Unique	Primary key and unique identifier for TurfProvider
3	contact_number	CharField	NOT NULL	Contact number for the TurfProvider
4	document	FileField	NOT NULL	Uploaded identity document of TurfProvider
5	address	TextField	NULL	Address of the TurfProvider
6	location	CharField	NULL	Location of the TurfProvider
7	is_active	BooleanField	Default: False	Indicates if the TurfProvider is active or not
8	random_password	CharField	NULL	Randomly generated password for TurfProvider (optional)
9	password_updated	BooleanField	Default: False	Indicates if the password for TurfProvider has been updated

TurfListing

Primary key: turfId

No	Field Name	Datatype	Key Constraints	Description of the field
1	turfId (PK)	Primary Key	UNIQUE	Primary id of each turf
2	turf_provider	ForeignKey	(TurfProvider)	Relationship field linking TurfListing to TurfProvider
3	turf_name	CharField	NULL	Name of the turf provided
4	description	TextField	NULL	Description about the TurfListing
5	price_per_hour	DecimalField	NULL	Price per hour for using the turf
6	location	CharField	NULL	Location of the turf
7	is_available	BooleanField	Default: True	Indicates if the turf is available for booking
8	available_from	TimeField	NOT NULL	Starting time for the availability of the turf
9	available_to	TimeField	NOT NULL	Ending time for the availability of the turf
10	created_at	DateTimeField	Auto-generated	Timestamp for when the TurfListing was created

TurfImage

Primary key: imageId

No	Field Name	Datatype	Key Constraints	Description of the field
1	imageId (PK)	Primary Key	UNIQUE	Unique id for each turf image
2	turf_listing	ForeignKey	(TurfListing)	Relationship field linking TurfImage to TurfListing
3	image	ImageField	NULL	Uploaded image related to the TurfListing

booking :

Primary key: bid

No	Field Name	Datatype	Key Constraints	Description of the field
1	bid (PK)	Primary Key	UNIQUE	Booking id
2	user	ForeignKey	(Usertable)	Relationship field linking Booking to Usertable
3	turf_listing	ForeignKey	(TurfListing)	Relationship field linking Booking to TurfListing
4	turf_provider	ForeignKey	(TurfProvider)	Relationship field linking Booking to TurfProvider
5	booking_date	DateField	NULL	Date of the booking
6	start_time	TimeField	NULL	Start time of the booking
7	end_time	TimeField	NULL	End time of the booking
8	total_cost	DecimalField	NULL	Total cost of the booking
9	created_at	DateTimeField	Auto-generated	Timestamp for when the booking was created

Payment :

Primary key: Pid

No	Field Name	Datatype	Key Constraints	Description of the field
1	Pid (PK)	Primary Key	UNIQUE	Payment ID
2	Bid (FK)	ForeignKey	(Booking)	Relationship field linking Payment to Booking
3	Amount	DecimalField	NULL	Amount of the payment
4	Payment Date	DateField	NULL	Date of the payment
5	Is Successful	BooleanField	Default: False	Indicates if the payment was successful

CHAPTER 5

SYSTEM TESTING

5.1 INTRODUCTION

Software testing serves as a critical means of ensuring that a computer program operates in accordance with its intended functionality. It is employed to verify that the software performs as expected, meeting the specified requirements and standards. Validation, in this context, involves the thorough examination and testing of software to confirm that it aligns with the user's desired outcomes. Software testing is a comprehensive process that, alongside methods such as inspection and program walkthroughs, aims to identify potential errors and discrepancies within the software. The objectives of testing can be guided by several principles, including the execution of a program with the aim of uncovering errors, the creation of effective test cases, and the successful identification of previously undiscovered errors. When conducted successfully, testing can pinpoint flaws in the software, ensuring that the computer program operates efficiently and as intended. Additionally, three fundamental methods are employed for assessing a computer program: correctness, implementation efficiency, and computational complexity.

5.2 TEST PLAN

A test plan serves as a comprehensive set of guidelines for conducting various types of tests, functioning as a navigational tool that outlines specific steps to follow. Software developers create instructions for both program usage and the organization of essential information to ensure the program's seamless operation. They meticulously verify each component of the program to confirm its intended functionality. The responsibility of thoroughly testing the built software lies with the ITG (Information Technology Group), ensuring comprehensive and rigorous testing beyond the initial development phase. The objectives of testing should be well-defined and quantifiable, with the test plan encompassing details about the frequency of malfunctions, associated costs for rectification, occurrence rates, and the time required for comprehensive reevaluation.

- Unit testing
- Integration Testing
- Data validation Testing
- Output Testing

5.2.1 Unit Testing

Software components or modules, which are the smallest units of a software design, are tested through unit testing. Using the design guidance, test key control pathways in a module to identify issues. This implies the level of difficulty of the tests for each minor component of a program and the components that haven't been put to the test. One kind of testing called unit testing examines the internal workings of the code and can include completed concurrently for several program components.

We must first determine whether data is correctly flowing between the various components of the computer software before beginning any more tests. All other checks are useless if the data isn't entering and exiting the system appropriately. When creating anything, it's critical to consider potential difficulties and develop a plan to address them. This could entail rerouting the procedure or ending it altogether.

To test the Sell-Soft System, individual components were examined and subjected to various testing. A few errors in the modules' design were found and corrected. Following the drafting of the instructions for various components, each component is examined and tested independently. We removed unnecessary code and ensured that everything functions as it should.

5.2.2 Integration Testing

Integration testing is a technique for developing a program while simultaneously searching for errors in the interoperability of its many components. Using tested components to develop a software according to plan is the goal. To ensure proper operation, the entire software is tested. Errors are corrected, but then mistakes occur again, and so on. Following the inspection of every component of the system, the components were assembled to ensure optimal performance. Furthermore, they created uniform programs rather than varying them.

5.2.3 Validation Testing or System Testing

This concludes the testing phase. During this test, the entire system was examined to ensure that all of the various building blocks and instructions interacted as intended. This type of testing is known as system testing or black box testing. One technique to make sure the software performs as intended is to use black box testing. Through the use of several input formats, black box testing assists software engineers in identifying every issue present in a program. Black box testing examines code for errors in startup and termination, performance, data access, functions, and interfaces.

5.2.4 Output Testing or User Acceptance Testing

The system is being evaluated to see if users like it and if it satisfies the business' requirements. While it is being developed or updated, the computer program must remain connected to the user.

The following factors are taken into consideration:

- Input screen designs
- Output screen designs

The aforementioned is tested using various types of data. For system testing, having the test data ready is crucial. After gathering test data, they use that information to check the system they are researching. We look for errors in the system and use the procedures we are already familiar with to find and correct them. In order to use these fixes in the future, we keep a record of them.

5.2.5 Automation Testing

Before a piece of software is officially used, automated testing determines whether it functions properly and complies with standards. This testing technique makes use of tools that execute written instructions. When a computer system employs a specialized tool to test things automatically, this is known as UI automation testing. We write scripts that perform this task automatically for each different test instead of relying on users to click around the application to ensure everything is in working order. When you verify information and then add it, there are a few things you should do. When you need to run the same test simultaneously on numerous computers, automatic testing is required.

5.2.6 Selenium Testing

Selenium is a free and practical tool that automatically tests websites. It's critical that web developers are aware of it. Selenium automation testing is the practice of testing things using the Selenium tool. Selenium is a collection of tools that each perform unique tasks for automation testing, not just one single tool. Although manual testing is a crucial step in the development of an application, it has flaws. Its potential to be monotonous and repetitive is a major issue. Jason Huggins, a ThoughtWorks employee, developed a method to automatically test things rather than doing it by hand to make things simpler. He created a program called the JavaScriptTestRunner to aid in automatically testing websites. The program's name was changed to Selenium in 2004.

Example:**Test Case 1****Code :**

```
package testdefinitions;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class login {

    WebDriver driver = CommonDriver.getDriver();

    @Given("browser is open")
    public void browser_is_open() {
        driver.manage().window().maximize();
    }

    @And("user is on login page")
    public void user_is_on_login_page() {
        driver.get("http://127.0.0.1:8000/user_login/");
    }

    @When("user enters email and password")
    public void user_enters_username_and_password() {
        WebElement username_input = driver.findElement(By.id("email"));
        WebElement passwordInput = driver.findElement(By.id("password"));

        username_input.sendKeys("hilalhabb@gmail.com");
        passwordInput.sendKeys("Hilal123@");
    }

    @And("User clicks on login")
    public void user_clicks_on_login() {
        WebElement loginButton = driver.findElement(By.id("login-btn"));
        loginButton.click();
    }

    @Then("user is navigated to the home page")
    public void user_is_navigated_to_the_home_page() {
        driver.quit();
    }
}
```

Screenshot

```

Scenario: Check login is succesfull with valid credentials # src/test/resources/Features/login.feature:3
  Given browser is open # testdefinitions.login.browser_is_open()
  And user is on login page # testdefinitions.login.user_is_on_login_page
  When user enters email and password # testdefinitions.login.user_enters_username
  And User clicks on login # testdefinitions.login.user_clicks_on_login
  Then user is navigated to the home page # testdefinitions.login.user_is_navigated_to

1 Scenarios (1 passed)
5 Steps (5 passed)
0m13.727s

```

Test Report

Test Case 1					
Project Name: Sportigo					
Login Test Case					
Test Case ID: 1			Test Designed By: Hilal Habeeb		
Test Priority(Low/Medium/High):High			Test Designed Date: 4-12-2023		
Module Name: Login Screen			Test Executed By : Ms Nimmy Francis		
Test Title : Login			Test Execution Date: 05-12-2023		
Description: Verify login with valid email and password					
Pre-Condition :User has valid email and password					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/ Fai l)
1	Navigation to Login Page		Home should be displayed	Login page displayed	Pass
2	Provide Valid email	Email: hilalhabb@gmail.com	User should be able to Login	User Logged in and navigated to Player Home	Pass
3	Provide Valid Password	Password: Hilal123@			
4	Click on Login Button				
Post-Condition: The user's credentials are validated against the database, granting successful login access to the user's account. The account session details are accurately recorded and logged in the database for future reference and tracking purposes.					

Test Case 2:**Code:**

```
package testdefinitions;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class search {

    WebDriver driver;

    @Given("the user is on the login page for turf Search")
    public void the_user_is_on_the_login_page_for_turf_search() {
        driver = new FirefoxDriver();
        driver.get("http://127.0.0.1:8000/user_login/");
    }

    @When("the user enters valid credentials and logs in for turf Search")
    public void the_user_enters_valid_credentials_and_logs_in_for_turf_search() {
        WebElement emailInput = driver.findElement(By.id("email"));
        WebElement passwordInput = driver.findElement(By.id("password"));

        emailInput.sendKeys("hilalhabb@gmail.com");
        passwordInput.sendKeys("Hilal123@");

        WebElement loginButton = driver.findElement(By.id("login-btn"));
        loginButton.click();
    }

    @And("the user navigates to the home page")
    public void the_user_navigates_to_the_home_page() {
        // Assuming the home page is navigated after successful login
        // Modify the URL or add logic if navigation occurs differently
    }

    @And("the user searches for the turf {string}")
    public void the_user_searches_for_the_turf(String turfName) {
```

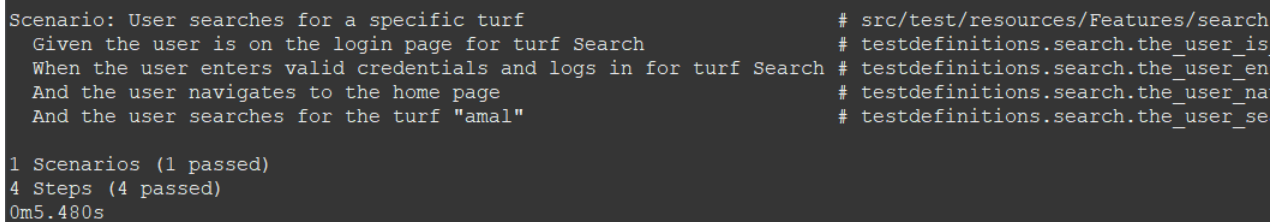
```
        WebElement searchInput = driver.findElement(By.id("search-input"));
        searchInput.sendKeys(turfName);

        WebElement searchButton =
driver.findElement(By.cssSelector("form#search-form button"));
        searchButton.click();
    }

    @Then("the user should see in the list of turfs")
    public void the_user_should_see_in_the_list_of_turfs(String turfName) {
        WebElement searchResults = driver.findElement(By.id("search-results"));
        String searchResultsText = searchResults.getText();

        if (searchResultsText.contains(turfName)) {
            System.out.println(turfName + " is displayed in the list of turfs.");
        } else {
            System.out.println(turfName + " is not found in the list of turfs.");
        }
        throw new io.cucumber.java.PendingException();
    }
}
```

Screenshot



```
Scenario: User searches for a specific turf                                     # src/test/resources/Features/search
  Given the user is on the login page for turf Search                         # testdefinitions.search.the_user_is
  When the user enters valid credentials and logs in for turf Search          # testdefinitions.search.the_user_en
  And the user navigates to the home page                                    # testdefinitions.search.the_user_na
  And the user searches for the turf "amal"                                  # testdefinitions.search.the_user_se

1 Scenarios (1 passed)
4 Steps (4 passed)
0m5.480s
```

Test report

Test Case 2					
Project Name:Sportigo					
Search Turf					
Test Case ID: 2			Test Designed By: Hilal Habeeb		
Test Priority(Low/Medium/High):High			Test Designed Date: 04-12-2023		
Module Name: Home Screen			Test Executed By : Ms Nimmy Francis		
Test Title : Search turf			Test Execution Date: 05-12-2023		
Description: User creates a feed					
Pre-Condition :User has valid email and password					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/ Fail)
1	Navigation to Login Page		Home should be displayed	Login page displayed	Pass
2	Provide Valid email	Email: hilalhabb@gmail.com	User should be able to Login	User Logged in and navigated to Player Home	Pass
3	Provide Valid Password	Password: Romo!0481			
5	Search for a turf	Turf: amal	User should be able to see the turf	The searched turf displayed on the homepage	Pass
6	Click on search				
Post-Condition: The user successfully logs in, search a turf, and view the searched turf.					

Test Case 3:**Code:**

```

package testdefinitions;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class update {

    WebDriver driver = CommonDriver.getDriver();
    @Given("user is on the settings page")
    public void user_is_on_the_settings_page() {
        driver.get("http://127.0.0.1:8000/userprofile");
    }

    @When("user enters a new full name {string}")
    public void user_enters_a_new_full_name(String string) {
        WebElement fullname_input =
driver.findElement(By.id("new_first_name"));
        fullname_input.clear();
        fullname_input.sendKeys("Abcd");
    }
    @And("user submits the update form")
    public void user_submits_the_update_form() {
        WebElement submitButton = driver.findElement(By.id("submitt"));
        submitButton.click();
    }
    @Then("the full name should be updated to {string}")
    public void the_full_name_should_be_updated_to(String string) {
    }
}

```

Screenshot

```

Scenario: User updates their full name                                # src/test/resources/Features/updates.feature
  Given browser is open                                              # testdefinitions.login.browser_is_open
  And user is on login page                                          # testdefinitions.login.user_is_on_login_page
  When user enters email and password                                # testdefinitions.login.user_enters_email_and_password
  And User clicks on login                                           # testdefinitions.login.user_clicks_login
  Given user is on the settings page                                  # testdefinitions.update.user_is_on_settings_page
  When user enters a new full name "New Full Name"                  # testdefinitions.update.user_enters_new_full_name
  And user submits the update form                                    # testdefinitions.update.user_submits_update_form

1 Scenarios (1 passed)
7 Steps (7 passed)
0m5.773s

```

Test report

Test Case 3					
Project Name:UrbanScout					
Update Profile					
Test Case ID: 3			Test Designed By: Hilal Habeeb		
Test Priority(Low/Medium/High):High			Test Designed Date: 04-12-2023		
Module Name: userprofile			Test Executed By : Ms Nimmy Francis		
Test Title : Update			Test Execution Date: 05-12-2023		
Description: User updates their first name					
Pre-Condition :User has valid email and password					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/ Fai l)
1	Navigation to Login Page		Home should be displayed	Login page displayed	Pass
2	Provide Valid email	Email: hilalhabb@gmail.com	User should be able to Login	User Logged in and navigated to Player Home	Pass
3	Provide Valid Password	Password: Hilal123@			
5	Click on profile	Button click	User should be able to update the firstname	User firstname changed	Pass
6	View profile and edit firstname	Newfirstname: "ABCD"			
Post-Condition: After logging in, the user accesses the home page, then in the profile section user updates their firstname.					

Test Case 4:**Code:**

```
package testdefinitions;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

public class function3 {

    WebDriver driver = CommonDriver.getDriver();

    @Given("user is on the login page")
    public void user_is_on_the_login_page() {
        driver.get("http://127.0.0.1:8000/user_login");
    }

    @When("admin enters email and password")
    public void admin_enters_email_and_password() {
        WebElement emailInput = driver.findElement(By.id("email"));
        WebElement passwordInput = driver.findElement(By.id("password"));

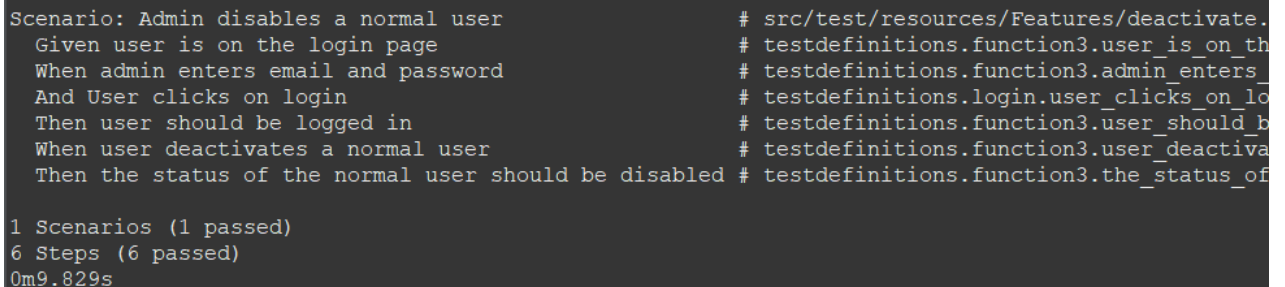
        emailInput.clear();
        emailInput.sendKeys("sportigoplayspot@gmail.com");
        passwordInput.clear();
        passwordInput.sendKeys("sportigo");
    }

    @Then("user should be logged in")
    public void user_should_be_logged_in() {
        WebElement usersDiv = driver.findElement(By.id("userss"));
    }
}
```



```
        usersDiv.click();
        WebElement normalDiv = driver.findElement(By.id("normalDiv"));
        normalDiv.click();
    }
    @When("user deactivates a normal user")
    public void user_deactivates_a_normal_user() {
        WebElement suspendCheck = driver.findElement(By.id("suspendCheck"));
        suspendCheck.click();
        WebElement statusUpd = driver.findElement(By.id("statusUpd"));
        statusUpd.click();
    }
    @Then("the status of the normal user should be disabled")
    public void the_status_of_the_normal_user_should_be_disabled() {
    }
}
```

Screenshot



```
Scenario: Admin disables a normal user # src/test/resources/Features/deactivate.
  Given user is on the login page # testdefinitions.function3.user_is_on_th
  When admin enters email and password # testdefinitions.function3.admin_enters_
  And User clicks on login # testdefinitions.login.user_clicks_on_lo
  Then user should be logged in # testdefinitions.function3.user_should_b
  When user deactivates a normal user # testdefinitions.function3.user_deactiva
  Then the status of the normal user should be disabled # testdefinitions.function3.the_status_of

1 Scenarios (1 passed)
6 Steps (6 passed)
0m9.829s
```

Test report

Test Case 4					
Project Name: Sportigo					
Admin deactivating a user					
Test Case ID: 4			Test Designed By: Hilal Habeeb		
Test Priority(Low/Medium/High):High			Test Designed Date: 04-12-2023		
Module Name: admin home			Test Executed By : Ms Nimmy Francis		
Test Title : Deactivate a User			Test Execution Date: 05-12-2023		
Description: Admin deactivates a user					
Pre-Condition :Admin has valid email and password					
Step	Test Step	Test Data	Expected Result	Actual Result	Status(Pass/Fail)
1	Navigation to Login Page		Home should be displayed	Login page displayed	Pass
2	Provide Valid email	Email: sportigoplayspot@gmail.com	Admin should be able to Login	Admin Logged in and navigated to Admin Home	Pass
3	Provide Valid Password	Password: sportigo			
5	Click on manage users		Admin should be able to see every users and can deactivate user	Admin	Pass
6	Click on normal _user	User email: hilalhabb@gmail.com			
7	Deactivate a user		User should be deactivated	The user is deactivated	Pass
Post-Condition: Admin login to their home page , see the list of users from that deactivating a user from normal user category					

CHAPTER 6

IMPLEMENTATION

6.1 INTRODUCTION

Implementation is the crucial phase during which the planned system materializes into a tangible, operational entity. Building user trust and confidence in the newly developed system is vital for its successful integration. The focus lies on user training and the creation of supportive resources. Conversion typically occurs during or after user training. In essence, implementation signifies the transition from a conceptual system design to its practical, functional existence. It involves the process of replacing or modifying existing systems to ensure a seamless shift towards the new system. Thorough planning, comprehensive system analysis, and the formulation of effective transition strategies are integral to the implementation process.

6.2 IMPLEMENTATION PROCEDURES

During the software implementation process, the software is installed in its intended environment and functions as expected. In certain organizations, an individual not directly involved in software usage oversees and greenlights the project's development. Initial hesitations may arise, and it is crucial to address any uncertainties before they escalate into formidable resistance. It is essential to demonstrate to users why the new system is superior to its predecessor and to instill a sense of trust in the software. Users should receive comprehensive training to ensure their comfort and confidence with the application. To evaluate the outcome, one must ensure that the server program is operational on the server before assessing the system's functionality.

6.2.1 User Training

User training is essential for imparting the necessary skills to individuals in utilizing and adapting to the system. It is imperative that users feel at ease and confident in their ability to navigate and operate the computer system effectively. Particularly when faced with complex functionalities, the significance of comprehensive user training becomes even more pronounced. This training equips users with the ability to input information, manage errors, query the database, and utilize various tools for generating reports and accomplishing other critical tasks.

6.2.2 Training on the Application Software

Training on the application software involves an in-depth understanding of how to operate a new program after familiarizing oneself with the fundamentals of computer usage. This training aims to provide comprehensive guidance on navigating the new system,

encompassing the utilization of various screens, accessing support resources, troubleshooting mistakes, and rectifying errors. Its primary objective is to equip users with the requisite knowledge and skills to effectively engage with the system, tailored to the specific needs and roles of different user groups.

6.2.3 System Maintenance

System maintenance is a complex challenge within the sphere of system development. It encompasses vital tasks and ensures the effective functioning of software during the maintenance phase of the software life cycle. Once a system is operational, it requires ongoing attention to ensure its continued smooth operation. Prioritizing software maintenance throughout the development process is essential to enable the system to adapt to environmental changes. The practice of software maintenance extends beyond mere error detection within the code.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

The Sportigo Turf Booking System stands as a revolutionary solution catering to the dynamic needs of sports enthusiasts and facility managers alike. Its intuitive interface and robust backend infrastructure have significantly streamlined the process of reserving sports facilities, optimizing user experiences, and enhancing operational efficiency. Transitioning from conventional booking methods to the Sportigo platform has marked a substantial advancement in the accessibility and management of sports facilities. The system's seamless user registration process and comprehensive booking functionalities have simplified the way users secure their desired sports venues, enabling hassle-free scheduling and reducing administrative overheads. Rigorous testing and user feedback have validated the system's adherence to predefined project objectives, ensuring reliability and user satisfaction.

7.2 FUTURE SCOPE

Looking forward, the Sportigo Turf Booking System aims to integrate cutting-edge technologies to further elevate its performance and user engagement. Implementation of artificial intelligence (AI) algorithms is envisioned to analyze booking patterns and user preferences, enabling the system to suggest tailored recommendations for available time slots, preferred sports facilities, and ancillary services. Additionally, the incorporation of real-time availability updates and dynamic pricing models will enhance transparency and flexibility for users, allowing for instant bookings and optimized cost-efficiency. The introduction of mobile applications with augmented reality (AR) features is on the horizon, providing users with immersive experiences by offering virtual tours of sports facilities and interactive maps for navigation within the venues. Furthermore, fostering a community-driven platform for user reviews, sharing game statistics, and organizing events is envisioned, cultivating an engaged sports community and fostering a collaborative environment for sports enthusiasts.

CHAPTER 8

BIBLIOGRAPHY

- Roger S. Pressman, Bruce R. Maxim, "Software Engineering: A Practitioner's Approach", McGraw-Hill Education, 8th Edition (2014) - Influential in guiding the systematic approach and software development methodologies employed in the creation of the Sportigo Turf Booking System.
- Martin Fowler, "Patterns of Enterprise Application Architecture", Addison-Wesley Professional, 1st Edition (2002) - Contributing to the architectural design principles utilized in developing the scalable and maintainable structure of the Sportigo platform.
- Scott Ambler, Larry Constantine, "The Unified Process Construction Phase: Best Practices in Implementing the Unified Process", Prentice Hall, 1st Edition (2000) - Contributing to the adoption of structured methodologies during the construction phase of the Sportigo Turf Booking System's development lifecycle.
- James Shore, Shane Warden, "The Art of Agile Development", O'Reilly Media, 1st Edition (2007) - Influential in adopting agile development practices, fostering flexibility, and iterative enhancements in the creation of Sportigo's booking system.

WEBSITES:

- @ playspots.in
- @ sporloc.com
- @footballtalent.net

CHAPTER 9

APPENDIX

9.1 Sample Code

Views.py

```
from decimal import Decimal
import random
import string
from django.contrib.auth import login, logout, authenticate, get_user_model

from django.http import HttpResponse, JsonResponse
from django.shortcuts import redirect, render
from .models import *
from django.contrib import messages
from django.views.decorators.cache import never_cache
from django.contrib.auth.decorators import login_required
from django.core.mail import send_mail
from django.contrib.auth.hashers import check_password
from django.shortcuts import get_object_or_404
from django.views.decorators.http import require_POST
from django.utils.dateparse import parse_date
from .models import TurfProvider
from django.contrib.auth import login
import matplotlib.pyplot as plt
from io import BytesIO
import base64
from .forms import TurfImageForm, TurfListingForm
from django.views.decorators.csrf import csrf_exempt
from io import BytesIO
import base64
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt

def generate_pie_chart(normal_users_count, club_users_count, turf_providers_count):

    labels = ['Normal Users', 'Club Users', 'Turf Providers']
    sizes = [normal_users_count, club_users_count, turf_providers_count]
    colors = ['#66b3ff', '#99ff99', '#c2c2f0']
    explode = (0.1, 0, 0) # explode 1st slice (Admin)

    # Create a pie chart
    plt.figure(figsize=(6, 4))
    patches, texts, autotexts = plt.pie(
        sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', startangle=90,
        textprops={'color': 'white'} # Set label text color to black
    )
    plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
    for text in texts:
        text.set_fontsize(15)
    for autotext in autotexts:
        autotext.set_fontsize(12)

    # Save the chart to a BytesIO object
    chart_image = BytesIO()
```

```
plt.savefig(chart_image, format='png', transparent=True)
plt.close() # Close the Matplotlib figure after saving it
chart_image.seek(0)
# Convert the BytesIO object to a base64-encoded string
chart_image_base64 = base64.b64encode(chart_image.getvalue()).decode('utf-8')

return chart_image_base64

def index(request):
    return render(request, "index.html")

def signup(request):
    if request.method == 'POST':
        role = request.POST['role']
        firstname = request.POST['fname']
        email = request.POST['email']
        phone_number = request.POST['phone_number']
        lastname = request.POST['lname']
        dob = request.POST['dob']
        password = request.POST['password']
        user = Usertable(first_name=firstname, last_name=lastname, role=role, dob=dob, email=email,
phone_number=phone_number)
        user.set_password(password)
        user.save()

        # Add a success message
        messages.success(request, 'Registration successful. You can now log in.')

        return redirect('user_login')
    return render(request, "registration.html")

def user_login(request):
    if request.method == 'POST':
        email = request.POST.get('email')
        password = request.POST.get('password')

        try:
            # Try to retrieve the user by email from TurfProvider
            provider = TurfProvider.objects.get(email=email)

            if provider.is_active:
                if provider.password_updated:
                    if password == provider.random_password:
                        request.session['email'] = email
                        return redirect('providerhome') # Redirect to providerhome.html
                    else:
                        error_message = "Invalid credentials."
                        messages.error(request, error_message)
                else:
                    # Redirect to the password update page
                    request.session['email'] = email
                    return redirect('provider_update')
            else:
                error_message = "Inactive user."
```

```
        messages.error(request, error_message)
except TurfProvider.DoesNotExist:
    # User does not exist in TurfProvider, check Usertable
    try:
        user = Usertable.objects.get(email=email)

        if user.is_active and check_password(password, user.password):
            request.session['email'] = email
            login(request, user)

            if user.role == 'admin':
                return redirect('adminreg')
            elif user.role == 'normal_user' or user.role == 'club_user':
                return redirect('index2')

        else:
            error_message = "Invalid credentials or inactive user."
            messages.error(request, error_message)
    except Usertable.DoesNotExist:
        error_message = "User does not exist."
        messages.error(request, error_message)

response = render(request, 'login.html')
response['Cache-Control'] = 'no-store, must-revalidate'
return response


def provider_update(request):
    user_email = request.session.get('email')
    provider = TurfProvider.objects.get(email=user_email)

    if request.method == 'POST':
        # Handle the form submission
        new_password = request.POST.get('new_password')

        # Update the random_password field in the TurfProvider model
        provider.random_password = new_password
        provider.password_updated = True # Set the flag to True
        provider.save()

        # Optionally, you can add a success message
        messages.success(request, 'Password updated successfully.')

        # Redirect the user to the providerhome page
        return redirect('providerhome')

    context = {
        'provider': provider,
    }

    return render(request, 'providerupdate.html', context)
from .models import Booking


def providerhome(request):
    if 'email' in request.session:
```

```
# Retrieve the TurfProvider object for the authenticated provider
provider = TurfProvider.objects.get(email=request.session['email'])

# Retrieve the list of turfs associated with the provider
your_turfs = TurfListing.objects.filter(turf_provider=provider)

# Get the count of the provider's turfs
turf_count = your_turfs.count()

# Retrieve booking details for the provider's turfs
bookings = Booking.objects.filter(turf_listing__in=your_turfs)

context = {
    'provider_name': provider.venue_name,
    'your_turfs': your_turfs, # Pass the list of turfs to the template
    'turf_count': turf_count, # Pass the count of turfs to the template
    'bookings': bookings # Pass the list of booking details to the template
}

response = render(request, 'providerhome.html', context)
response['Cache-Control'] = 'no-store, must-revalidate'
return response

else:
    return redirect('index')

def userLogout(request):
    logout(request)
    return redirect('index')

def index2(request):
    if 'email' in request.session:
        # Fetch available turfs from the database
        available_turfs = TurfListing.objects.filter(is_available=True)

        response = render(request, 'index2.html', {'available_turfs': available_turfs})
        response['Cache-Control'] = 'no-store, must-revalidate'
        return response
    else:
        return redirect('index')

def about(request):
    return render(request, "about.html")

def contact(request):
    return render(request, "contact.html")

def check_user_exists(request):
    email = request.GET.get('email') # You can also use 'email' if you're checking by email
    data = {
        'exists': Usertable.objects.filter(email=email).exists()
    }
    return JsonResponse(data)

def forgotPassword(request):
```

```

if request.method == 'POST':
    if 'newpassword' in request.POST:
        user = Usertable.objects.get(email=request.session["email"])
        user.set_password(request.POST['newpassword'])
        user.save()
        return redirect('user_login')
    elif 'otp' in request.POST:
        if request.session["otp"] == request.POST["otp"]:
            otp = "valid"
            # user = Usertable.objects.get(email=request.session["email"])
            return render(request, "forgotPassword.html", {"otp":otp})
        else:
            return HttpResponse("Invalid OTP")
    elif 'email' in request.POST:
        userEmail = request.POST['email']
        if Usertable.objects.filter(email=userEmail).exists():
            request.session["otp"]=generate_otp()
            user = Usertable.objects.get(email=userEmail)
            request.session["email"] = user.email
            valid = "true"
            subject = 'Hello, User'
            message = 'This is a email to Reset your Password\n The OTP to Change Password is : '+request.session["otp"]
            from_email = "spotigoturfcorporation@gmail.com"
            recipient_list = [userEmail]
            send_mail(subject, message, from_email, recipient_list)
            return render(request, "forgotPassword.html", {"valid":valid})
        else:
            return HttpResponse("User Doesn't Exists")
    return True
return render(request, "forgotPassword.html")

def generate_otp(length=6):
    characters = string.digits # Use digits (0-9) for OTP generation
    otp = ''.join(random.choice(characters) for _ in range(length))
    return otp

def adminreg(request):
    if not request.session.get('email'):
        return redirect('index') # Redirect to the index page if the user is not logged in

    if request.method == 'POST':
        # Handle status change for users
        for user in Usertable.objects.all():
            if user.email in request.POST:
                new_status = request.POST.get(user.email) == 'on'
                if user.is_active != new_status:
                    user.is_active = new_status
                    user.save()

                # Send email notification for both active and inactive status changes
                send_status_change_notification(user, new_status)

        # Handle status change for TurfProvider users
        for provider in TurfProvider.objects.all():

```

```

    if provider.email in request.POST:
        new_status = request.POST.get(provider.email) == 'on'
        if provider.is_active != new_status:
            provider.is_active = new_status
            provider.save()

            # Generate a random password
            random_password = generate_random_password()
            provider.random_password = random_password
            provider.save()

            # Send email notification for activation with a random password
            send_turf_provider_activation_notification(provider, random_password, new_status)

# Calculate the total count of users (both Usertable and TurfProvider)
total_users = Usertable.objects.count() + TurfProvider.objects.count()
# Retrieve pending turf providers
pending_providers = TurfProvider.objects.filter(is_active=False)

# Calculate user counts here
admin_count = Usertable.objects.filter(role='admin').count()
normal_users_count = Usertable.objects.filter(role='normal_user').count()
club_users_count = Usertable.objects.filter(role='club_user').count()
turf_providers_count = TurfProvider.objects.count()

# Generate the pie chart
pie_chart_image = generate_pie_chart(normal_users_count, club_users_count, turf_providers_count)

# Retrieve users based on their roles
admin_users = Usertable.objects.filter(role='admin')
normal_users = Usertable.objects.filter(role='normal_user')
club_users = Usertable.objects.filter(role='club_user')
turf_providers = TurfProvider.objects.all()

context = {
    'admin_users': admin_users,
    'normal_users': normal_users,
    'club_users': club_users,
    'turf_providers': turf_providers,
    'total_users': total_users,
    'pending_providers': pending_providers,
    'pie_chart_image': pie_chart_image,
    # Add more data as needed
}

return render(request, 'adminreg.html', context)

def send_status_change_notification(user, new_status):
    subject = "Your Status Change Notification"
    if new_status:
        message = "Your status on Sportigo has been updated. Your new status is: Active"
    else:

```



```

    message = "Your status on Sportigo has been updated. Your new status is: Inactive"
    from_email = "sportigoplayspot@gmail.com" # Your email address
    recipient_list = [user.email]

    send_mail(subject, message, from_email, recipient_list)

def generate_random_password():
    password_length = 10
    characters = string.ascii_letters + string.digits
    return "".join(random.choice(characters) for _ in range(password_length))

def send_turf_provider_activation_notification(provider, random_password, new_status):
    # Implement your email notification logic for TurfProvider activation
    # Example for activation notification:
    subject = 'Turf Provider Account Activation'
    if new_status:
        message = f'Congratulations! Your Turf Provider account is now verified by the admin. Your initial password is: {random_password}'
    else:
        message = "Sorry, your request for joining sportigo is rejected"

    from_email = 'sportigoplayspot@gmail.com'
    recipient_list = [provider.email]
    send_mail(subject, message, from_email, recipient_list)

from django.contrib import messages as django_messages
@login_required
def user_profile(request):
    user = request.user
    formatted_dob = user.dob.strftime('%Y-%m-d') if user.dob else ""

    # Retrieve messages from the message framework using the new variable name
    messages = django_messages.get_messages(request)

    context = {'user': user, 'formatted_dob': formatted_dob, 'messages': messages}
    return render(request, 'userprofile.html', context)

@login_required
def update_user_details(request):

    if request.method == 'POST':

        new_first_name = request.POST.get('new_first_name')
        new_last_name = request.POST.get('new_last_name')
        new_phone_number = request.POST.get('new_phone_number') # Add this line

        # Update the user's details in the database
        user = request.user

        user.first_name = new_first_name
        user.last_name = new_last_name
        user.phone_number = new_phone_number # Add this line
        user.save()

        messages.success(request, 'User details updated successfully.')

```

```

        return redirect('user_profile') # Redirect to the user profile page

    return render(request, 'userprofile.html')

from django.contrib.auth import update_session_auth_hash
from django.contrib.auth.forms import PasswordChangeForm
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required

def add_turf(request):
    if request.method == 'POST':
        turf_form = TurfListingForm(request.POST)
        image_form = TurfImageForm(request.POST, request.FILES)

        if turf_form.is_valid() and image_form.is_valid():
            turf_listing = turf_form.save(commit=False)
            turf_listing.turf_provider = TurfProvider.objects.get(email=request.session['email'])
            turf_listing.save()

            for f in request.FILES.getlist('images'):
                TurfImage.objects.create(turf_listing=turf_listing, image=f)

            return redirect('providerhome')

        else:
            turf_form = TurfListingForm()
            image_form = TurfImageForm()

    return render(request, 'addturf.html', {'turf_form': turf_form, 'image_form': image_form})

@login_required
def turf_detail(request, turf_id):
    if request.method == 'POST':
        selected_time_slot = request.POST.get('selected_time_slot')
        selected_date = request.POST.get('booking_date')

        if not selected_time_slot or not selected_date:
            return HttpResponse("Please select a date and time slot")

        start_time_str, end_time_str = selected_time_slot.split(' - ')
        start_time = datetime.strptime(start_time_str, '%H:%M').time()
        end_time = datetime.strptime(end_time_str, '%H:%M').time()
        turf = get_object_or_404(TurfListing, id=turf_id)
        total_cost = ((end_time.hour - start_time.hour) + 1) * turf.price_per_hour
        amount = total_cost
        amount *= Decimal('100')
        user = request.user

        existing_bookings = Booking.objects.filter(
            Q(booking_date=selected_date),
            Q(start_time__lte=start_time, end_time__gte=start_time) |
            Q(start_time__lte=end_time, end_time__gte=end_time),
            turf_listing=turf
        )

```

```

if existing_bookings.exists():
    messages.error(request, 'This time slot is already booked. Please choose another time.')
    return redirect('turf_detail', turf_id=turf_id)

# Storing booking information in session
request.session['selected_date'] = selected_date
request.session['start_time'] = start_time.strftime('%H:%M')
request.session['end_time'] = end_time.strftime('%H:%M')
request.session['turf_id'] = turf_id
request.session['total_cost'] = float(total_cost)

# Redirect to the confirmation page if the time slot is available
context = {
    'turf': turf,
    'selected_date': selected_date,
    'start_time': start_time,
    'end_time': end_time,
    'total_cost': total_cost,
    'amount': amount
}

return render(request, 'confirmation.html', context)

turf = get_object_or_404(TurfListing, id=turf_id)
available_from = turf.available_from
available_to = turf.available_to
time_slots = []

while available_from < available_to:
    time_slots.append(
        available_from.strftime('%H:%M') + ' - ' + (datetime.combine(datetime.today(), available_from) +
        timedelta(minutes=59)).strftime('%H:%M')
    )
    available_from = (datetime.combine(datetime.today(), available_from) + timedelta(hours=1)).time()

context = {
    'turf': turf,
    'time_slots': time_slots
}

return render(request, 'turf_detail.html', context)

@csrf_exempt
def confirmation(request):
    if request.method == 'POST':
        amount = 50000

        client = razorpay.Client(auth=("rzp_test_gfHLcbNXLAqvpT", "2RsNQRDiZYjnJuYIuMVA4cDr"))
        payment = client.order.create({'amount': amount, 'currency': 'INR', 'payment_capture': '1'})
        razorpay_payment_id = request.POST.get('razorpay_payment_id')

        if razorpay_payment_id:
            # Retrieve booking details from session or request.POST

```

```

        selected_date = request.session.get('selected_date')
        start_time = request.session.get('start_time')
        end_time = request.session.get('end_time')
        turf_id = request.session.get('turf_id')
        total_cost = Decimal(request.session.get('total_cost'))
        user = request.user

        # Get TurfListing instance
        turf = get_object_or_404(TurfListing, id=turf_id)

        # Create and save Booking instance
        booking = Booking.objects.create(
            user=user,
            turf_listing=turf,
            turf_provider=turf.turf_provider, # Replace 'turf.provider' with appropriate attribute or remove
            if not needed
            booking_date=selected_date,
            start_time=start_time,
            end_time=end_time,
            total_cost=total_cost
        )

        # Construct email content without using a template
        subject = 'Booking Confirmation'
        from_email = 'sportigoplayspot@gmail.com' # Replace with your email
        to_email = user.email

        email_content = (
            f"Dear {user.email},\n\n"
            f"Your booking with ID: {booking.id} has been confirmed.\n"
            f"Booking Date: {selected_date}\n"
            f"Start Time: {start_time}\n"
            f"End Time: {end_time}\n"
            f"Total Cost: {total_cost}\n\n"
            "Thank you for booking with us!\n\n"
            "Regards,\nThe Sportigo Team"
        )

        # Send the email
        email = EmailMultiAlternatives(subject, email_content, from_email, [to_email])
        email.send()

        messages.success(request, 'Payment successful! Booking confirmed. Email sent.')
        return redirect('booking_history')
    else:
        messages.error(request, 'Payment unsuccessful. Please try again.')
        return redirect('turf_detail', turf_id=turf_id)

    return render(request, 'confirmation.html', {'payment': payment})

@login_required
def booking_history(request):
    # Retrieve the user's booking history
    user_bookings = Booking.objects.filter(user=request.user).order_by('-created_at')

```

```
messages_list = messages.get_messages(request)
context = {
    'user_bookings': user_bookings,
    'messages': messages_list
}

return render(request, 'booking_history.html', context)

from django.shortcuts import get_object_or_404

from django.shortcuts import get_object_or_404

from django.shortcuts import get_object_or_404

def manage_turf(request, turf_id):
    # Get the existing turf instance
    turf = get_object_or_404(TurfListing, id=turf_id)

    if request.method == 'POST':
        # Create the form with the submitted data and instance
        form = TurfListingForm(request.POST, instance=turf)
        if form.is_valid():
            updated_turf = form.save(commit=False)

            # Check if new images are uploaded
            new_images = request.FILES.getlist('images')
            if new_images:
                # Delete existing images not included in the updated list
                existing_images = turf.images.all()
                for existing_image in existing_images:
                    if existing_image.image not in new_images:
                        existing_image.delete()

                # Add or update new images
                for f in new_images:
                    TurfImage.objects.create(turf_listing=updated_turf, image=f)

            updated_turf.save() # Save the updated turf details
            return redirect('providerhome')

    else:
        # Create the form with the existing turf data
        form = TurfListingForm(instance=turf)

    return render(request, 'manage_turf.html', {'form': form, 'turf': turf})

def search(request):
    query = request.GET.get('query')
    search_results = []

    if query:
        # Query the database for matching TurfListings by turf name or location
        turfs = TurfListing.objects.filter(
            Q(turf_name__icontains=query) | Q(location__icontains=query)
        )
```

```

# Create a list of results with the required attributes and single image for each turf
for turf in turfs:
    # Retrieve the first image for the turf, if available
    image = TurfImage.objects.filter(turf_listing=turf).first()
    image_url = image.image.url if image else None

    search_results.append({
        'turf_name': turf.turf_name,
        'location': turf.location,
        'image_url': image_url,
        'id': turf.id,
        'price_per_hour': turf.price_per_hour,
    })

return JsonResponse(search_results, safe=False)
</script>

```

turf_detail.html

//turf booking template

```

{% load static %}
<!DOCTYPE html>
<html lang="en">
<head></head>
<body>
    <main>
        <div class="wrapper">
            <header class="header-main">
                <div class="header-upper">
                    <div class="container">
                        <div class="row">
                            <ul>
                                <li><a href="{% url 'logout' %}">LOGOUT</a></li>
                            </ul>
                        </div>
                    </div>
                </div>
                <div class="header-lower clearfix">
                    <div class="container">
                        <div class="row">
                            <h1 class="logo"><a href="index.html"></a></h1>
                            <div class="menubar">
                                <nav class="navbar">
                                    <div class="nav-wrapper">
                                        <div class="navbar-header">
                                            <button type="button" class="navbar-toggle"><span
class="sr-only">Toggle navigation</span>
                                            <span class="icon-bar"></span></button>

```

```

</div>
<div class="nav-menu">
  <ul class="nav navbar-nav menu-bar">
    <li><a href="{ % url 'index2' % }" >Home

    <span></span> <span></span></a></li>
    <li><a href="{ % url 'booking_history' % }">History

    <span></span> <span></span></a></li>
    <li><a href="{ % url 'user_profile'
% }">profile<span></span> <span></span> <span></span> <span></span></a>
    </li>

    <li><a href="{ % url 'contact' % }">contact

    <span></span> <span></span> <span></span>

    <span></span></a></li>
  </ul>
</div>
</div>
</nav>
</div>
<div class="social"><a
href="https://www.facebook.com/hilal.habeeb.1?mibextid=ZbWKwL"
class="facebook"><i class="fa fa-facebook"></i></a> <a
href="https://twitter.com/itobuztech" class="twitter"><i class="fa fa-
twitter"></i></a>

<a href="https://www.behance.net/" class="behance"><i class="fa fa-
behance"></i></a></div>
</div>
</div>
</div>
</header>

</div>

<br><br><br><br>

<div class="content">
  <div class="turf-details">
    <div class="turf-image-container" style="position: relative;">
      <div class="turf-slider">
        { % for image in turf.images.all % }
          
        { % endfor % }
      </div>
      <div class="turf-logo">
        
      </div>
    </div>
  </div>
</div>
</div>

<div class="turf-details" style="float: left; width: 50%; margin-bottom: 0;">

```

```

        <div style="border: 0px solid #621e1e; border-radius: 5px; padding: 80px; margin: 10px
auto; max-width: 1000px;">
            <div style="display: flex; justify-content: space-between; align-items: center;
margin-bottom: 20px;">
                <p><i class="fas fa-map-marker-alt"></i> {{ turf.location }}</p>
                <p><i class="fas fa-location-arrow"></i> {{ turf.turf_name }}</p>
                <p><i class="fas fa-rupee-sign"></i> {{ turf.price_per_hour }} /hr</p>
            </div>
            <p><strong>Sports Type:</strong> {{ turf.sports_type }}</p>
            <p><strong>Description:</strong> {{ turf.description }}</p>
            <p><strong>Available From:</strong> {{ turf.available_from }}</p>
            <p><strong>Available To:</strong> {{ turf.available_to }}</p>
        </div>
    </div>

    <div class="booking-section" style="float: right; width: 50%; margin-top: -70px; ">
        <section class="turf-booking">
            <div class="booking-container" style="border: 1px solid #ffffff; border-radius: 5px;
padding: 50px; margin: 10px auto; max-width: 1000px;">
                <h2>Book this Turf</h2>
                <form id="booking-form" method="post">
                    {% csrf_token %}
                    <label for="booking-date">Select Date:</label>
                    <input type="date" id="booking-date" name="booking_date" required>
                    <div class="time-slots">
                        <label for="booking-time">Select Time Slot:</label>
                        <select name="selected_time_slot" required>
                            {% for time_slot in time_slots %}
                                <option value="{{ time_slot }}">{{ time_slot }}</option>
                            {% endfor %}
                        </select>
                    </div>
                    <button id="book-now-button" class="book-button" type="submit">Book
Now</button>
                </form>
                {% if messages %}
                <ul class="messages">
                    {% for message in messages %}
                        <h4><li{% if message.tags %} class="{{ message.tags }}"{% endif
%}>{{ message }}</li></h4>
                    {% endfor %}
                </ul>
                {% endif %}
            </div>
        </section>
    </div>
</div>
</main>

<footer>
    <p>© 2023 Turf Provider, Inc.</p>
</footer>
<script src="{% static 'js/vendor/vendor.js' %}"></script>
<script src="{% static 'js/main.js' %}"></script>
<script>

```



```
// Calculate the minimum and maximum dates (today and 5 days from today)
var today = new Date();
var maxDate = new Date(today);
maxDate.setDate(today.getDate() + 5);

// Format the minimum and maximum dates as YYYY-MM-DD
var todayFormatted = today.toISOString().split('T')[0];
var maxDateFormatted = maxDate.toISOString().split('T')[0];

// Set the minimum and maximum dates in the input field
document.getElementById("booking-date").setAttribute("min", todayFormatted);
document.getElementById("booking-date").setAttribute("max", maxDateFormatted);
</script>
<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script>
    $(document).ready(function () {
        var slideIndex = 0;
        var slides = $(".turf-image");
        var totalSlides = slides.length;

        // Show the first image initially
        showSlide(slideIndex);

        // Function to display the selected slide
        function showSlide(index) {
            if (index >= totalSlides) {
                slideIndex = 0;
            }
            if (index < 0) {
                slideIndex = totalSlides - 1;
            }

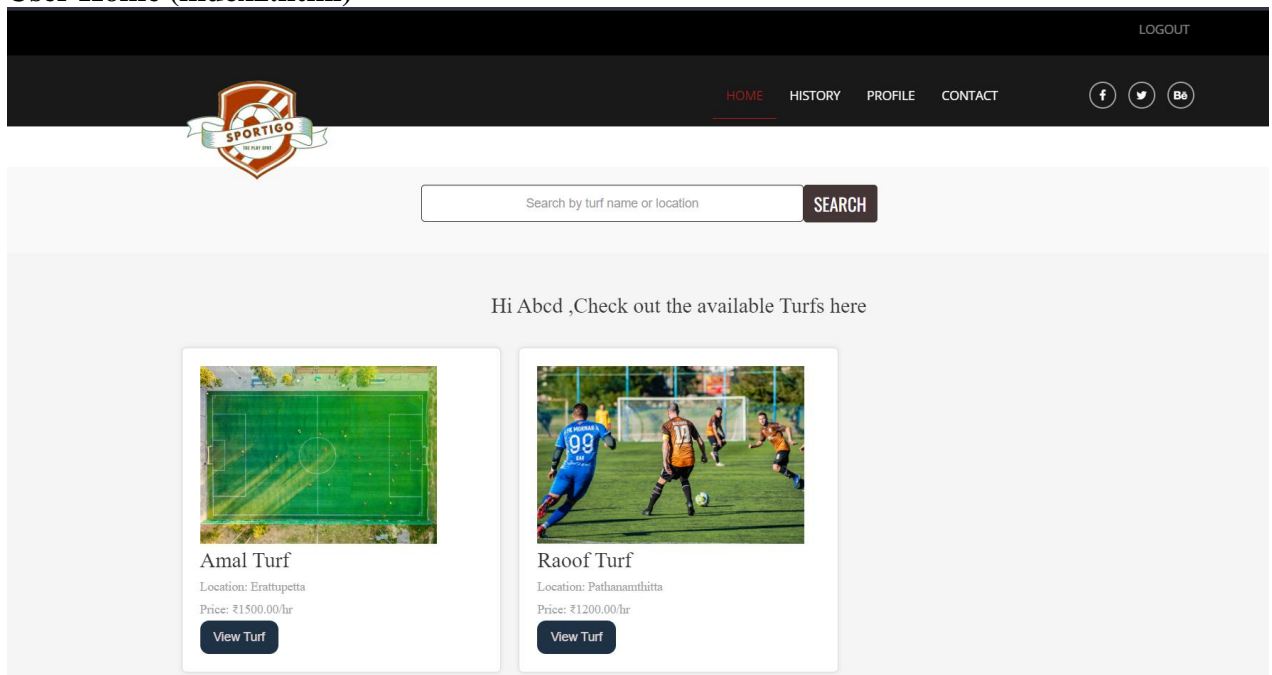
            slides.removeClass("active");
            slides.eq(slideIndex).addClass("active");
        }

        // Functionality for the next slide button
        $(".next-slide").click(function () {
            slideIndex++;
            showSlide(slideIndex);
        });

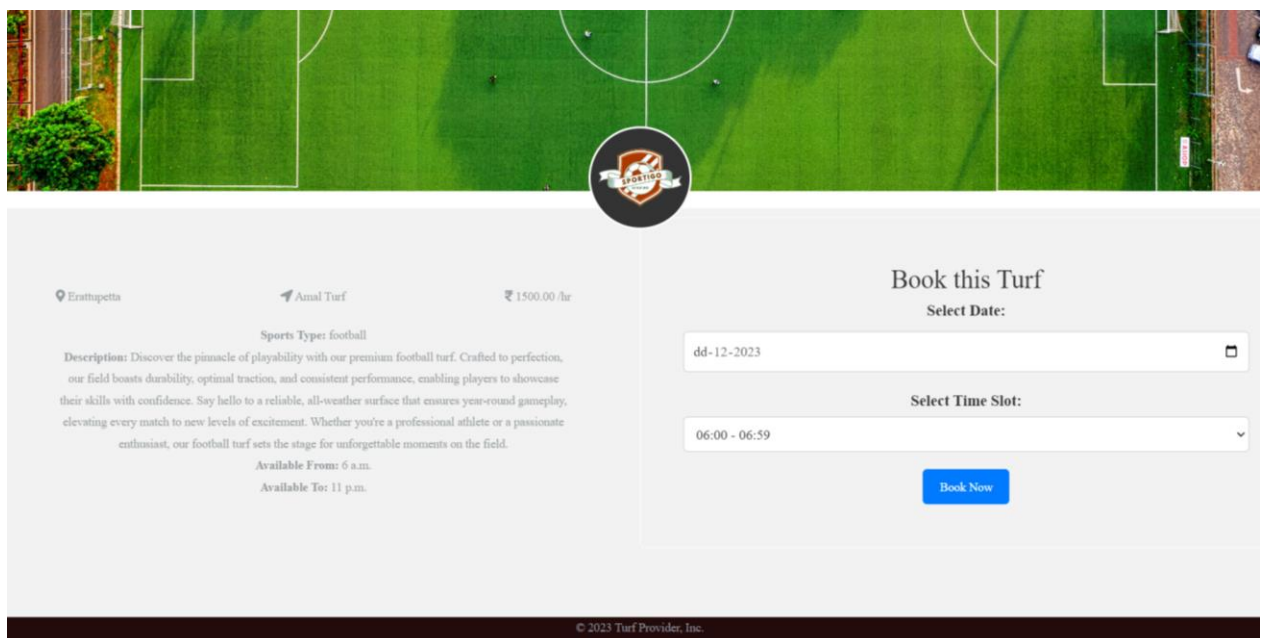
        // Functionality for the previous slide button
        $(".prev-slide").click(function () {
            slideIndex--;
            showSlide(slideIndex);
        });
    });
</script>
</body>
</html>
```

9.2 Screenshots

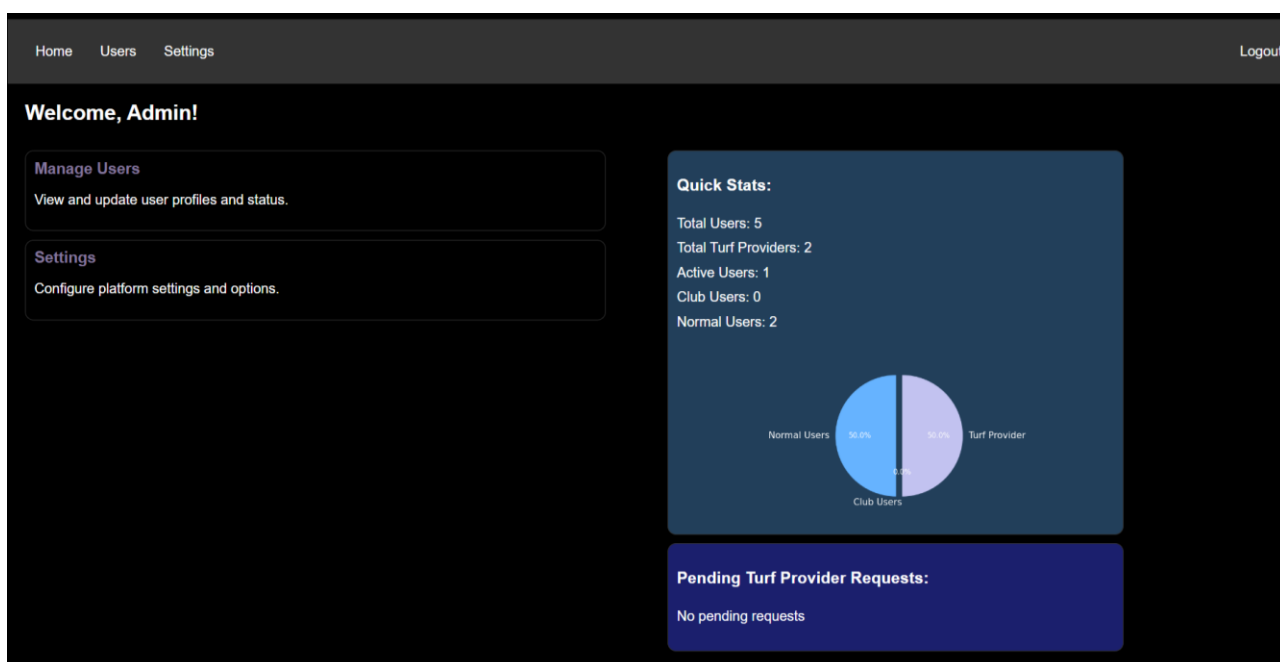
User Home (index2.html)



Booking page (turf_detail.html)



Admin home (adminreg.html)



Admin user management (adminreg.html)

The Admin user management dashboard (adminreg.html) features a dark theme with a top navigation bar containing 'Home', 'Users', 'Settings', and a 'Logout' link. The main content area is divided into three sections:

- Normal User**: The backbone of sportigo
- Club User**: Clubbies...
- Turf Provider**: Partners from different places...

Below these sections is a **Turf Providers** table:

Venue Name	Email	Contact Number	Sports Type	Address	Location	Document	Status
Amal Turf	amalj5634@gmail.com	8974562314	Football	Thyparambil road, erattupetta	9.693831150000001, 76.78879528912865	View Document	<input checked="" type="checkbox"/>
Raoo Turf	raooftenad123@gmail.com	7895684123	Football	Venad road, pathamthitta	9.2655337, 76.7871514	View Document	<input checked="" type="checkbox"/>

Below the table is an **Update Status** button.

Turf provider partnership form (providerreg.html)

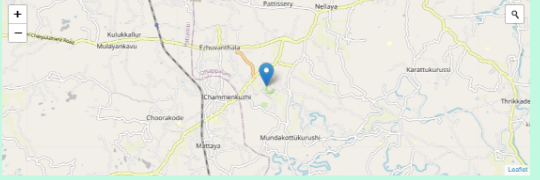
[Home](#) [Contact](#)

PARTNER WITH US
3000+ Users 800+ Venues 150+ Cities


Grow Your Business With Sportigo
IMPROVE THE MANAGEMENT
With our manager app, venues can easily streamline bookings.
INCREASE VISIBILITY
As the largest sports facility network in India, we increase the exposure of the venue.

Get Started in Easy Steps

1. Fill the partnership form
2. Get the approval email
3. Login with the credentials

START YOUR JOURNEY WITH PLAYSPOTS
Partnership form
Venue Name: Email:
Contact Number: Document (PDF): No file chosen
Sports Type: Address:

Location:

Provider home(providerhome.html)

Sportigo Turf Provider  [Logout](#)

[Dashboard](#) [Add Turf](#) [Update Password](#)

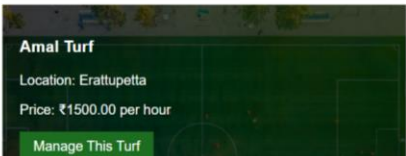
Welcome, Amal Turf

Total Turfs
1

Bookings Today

Revenue Today

Your Turfs



Amal Turf
Location: Erattupetta
Price: ₹1500.00 per hour
[Manage This Turf](#)