A PROJECT REPORT

ON

"Gym Management System"

SUBMITTED
TO

SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE

FOR THE PARTIAL FULFILMENT OF

MASTER OF COMPUTER

APPLICATION(MCA-I, SEM.-II)

BY

Jadhav Rupesh Manoj

UNDER THE GUIDANCE OF

PROF. YOGESH SHARMA

THROUGH

THE DIRECTOR
SINHGAD INSTITUTE OF MANAGEMENT AND COMPUTER APPLICATION
(SIMCA),NARHE, PUNE (AY. 2022-2023)

SINHGAD TECHNICAL EDUCATION SOCIETY'S

SINHGAD INSTITUTE OF MANAGEMENT & COMPUTER APPLICATIONS

(Affiliated to Savitribai Phule Pune University & Approved by AICTE)

'NAAC' Accredited with 'A' Grade

S. No. 49/1, Off Westernly Bypass, Pune – Mumbai Expressway, Narhe, Pune – 411041, Tel : (020) 66831896 / 66831908 / 66831907

E-mail : director_mca_simca@sinhgad.edu Website : www.sinhgad.edu

Prof. M. N. Navale M.E. (Elec), MIE, MBA FOUNDER PRESIDENT	Dr. (Mrs.) Sunanda M. Navale B.A., P.P.M., Ph.D. FOUNDER SECRETARY	Dr. Vijaya Puranik *M.Sc., MMM,MPM, Ph.D.* *DIRECTOR, SIMCA*

CERTIFICATE

This is to certify that, the project entitled "**Gym Management System**", being submitted forthe partial fulfilment of the degree of Master of Computer Application by him to **Sinhgad Institute of Management and Computer Applications** affiliated to **Savitribai Phule Pune University, Pune** is the result of the original work completed by ***Jadhav Rupesh Manoj*** under the guidance of ***Prof. Yogesh Sharma***.

To the best of our knowledge and belief, this work has not been previously submitted by the award of any degree or diploma of Savitribai Phule Pune University or any other University.

PLACE:

DATE:

Prof. Yogesh Sharma	**Prof. Navanath Choudhari**	**Dr. Vijaya Puranik**
Internal Guide	**Project Co-Ordinator**	**Director SIMCA**

External Examiner

DECLARATION

I, the undersigned hereby declare that the project titled “**Gym Management System**”, being submitted for the award of degree of **Master of Computer Application** by me to **Sinhgad Institute of Management and Computer Applications (SIMCA)** affiliated to **Savitribai Phule Pune University** is the result of an independent work carried out under the guidance of **Prof. Yogesh Sharma**, is my original work.

Further I declare that this project has not been submitted to this or any Institution for the award of any degree.

PLACE: PUNE Jadhav Rupesh Manoj

DATE: (Student)

ACKNOWLEDGEMENT

We take this opportunity to express our gratitude towards all those who have helped us throughout the successful completion of our project Firstly, we would like to thank our Director **Dr. Vijaya Puranik**, and project co-ordinator **Prof. Navnath Choudhari** of Masters in Computer Application, for permitting us to complete our project and for showing faith in us and allowing us to develop **Gym Management System**. We must convey our gratitude to our guide **Prof. Yogesh Sharma** for giving us the constant guidance of inspiration and help in preparing the project, personally, correcting our work and providing encouragement throughout the project

Jadhav Rupesh Manoj

INDEX

1. Introduction

The Gym Management System is a comprehensive web-based application designed to revolutionize the management of gyms and fitness centers. It provides a centralized platform for gym owners, staff members, and members to automate administrative tasks, manage memberships, schedule classes, and facilitate course bookings. By leveraging the power of Django, SQLite, HTML, CSS, and JavaScript, the system offers a robust set of features and a user-friendly interface.

With the Gym Management System, gym owners can easily manage member profiles, track membership status, and handle renewals and cancellations efficiently. They can create and manage class schedules, assign instructors, set maximum capacities, and update them in real-time. Members benefit from the convenience of browsing available courses, viewing class details, and booking classes based on their preferences and availability. The system ensures accurate attendance tracking, enabling staff members to monitor member participation in booked courses or classes effectively.

Additionally, the Gym Management System streamlines billing processes by generating invoices for membership fees, course bookings, and additional services. It integrates with secure payment gateways to facilitate online payments, ensuring a seamless and transparent payment experience for members.

Furthermore, the system provides comprehensive reporting and analytics capabilities, offering valuable insights into member attendance, revenue analysis, class popularity, and overall performance. These insights aid in data-driven decision-making, enabling gym owners and staff members to optimize operations, identify trends, and strategize for business growth.

In conclusion, the Gym Management System empowers gyms and fitness centers to streamline their operations, enhance member satisfaction, and optimize resource allocation. It is a comprehensive solution that simplifies processes, improves efficiency, and provides a seamless experience for all stakeholders involved in gym management.

1.1 Abstract

The Gym Management System is a user-friendly web application designed to simplify the management of a gym or fitness center. It offers features such as membership management, class scheduling, course bookings, attendance tracking, billing, and reporting. The system aims to streamline administrative tasks, improve member experience, and optimize resource allocation. Built using Django, SQLite, HTML, CSS, and JavaScript, the Gym Management System provides gym owners, staff members, and members with an efficient and convenient platform to manage gym operations effectively.

1.2 Existing System and Need for System

In the existing system for gym management, many gyms and fitness centers rely on manual processes and disparate tools to handle various administrative tasks. These processes often involve manual record-keeping, paper-based systems, and inefficient communication methods. This approach can lead to challenges such as data inconsistencies, difficulties in managing memberships and schedules, and limited access to real-time information.

However, there is a growing need for a more streamlined and automated system to address these challenges and improve overall gym management. The Gym Management System aims to fulfill this need by providing a centralized and integrated platform for managing all aspects of a gym.

The need for the Gym Management System arises from the following factors :

1. **Efficiency and Time Savings**: The existing manual processes and fragmented tools can be time-consuming and prone to errors. A dedicated system can automate routine tasks, simplify processes, and save time for gym administrators and staff members.

2. **Improved Member Experience**: Members expect convenient and hassle-free experiences when managing their memberships, booking classes, and tracking their progress. The Gym Management System provides a user-friendly interface that enhances member satisfaction and engagement.

3. **Streamlined Administrative Tasks**: The system eliminates the need for manual record-keeping and simplifies tasks such as membership management, class scheduling, attendance tracking, and billing. This streamlining improves efficiency and reduces the risk of errors.

4. **Data Accuracy and Centralized Information**: By centralizing data in a reliable database, the Gym Management System ensures data accuracy and consistency. It provides real-time access to information, enabling gym administrators and staff to make informed decisions based on up-to-date data.

5. **Business Growth and Scalability**: With a scalable and adaptable system, gym owners can accommodate growth and expand their services without compromising on efficiency. The Gym Management System supports the scalability needs of the gym as it continues to evolve.

2. Proposed System

The proposed Gym Management System is a comprehensive web-based application that aims to transform gym management by providing an integrated and efficient platform for gym owners, staff members, and members. The system offers a wide range of features to streamline administrative tasks, enhance member experience, and optimize resource allocation.

One of the key features of the proposed system is membership management. It allows gym administrators to easily handle member registrations, track membership statuses, and manage member profiles. This includes storing personal information, managing membership plans, and handling membership renewals and cancellations. The system maintains a centralized database to ensure accurate and up-to-date member data.

Another crucial aspect of the proposed system is class scheduling and management. Gym owners and staff members can effortlessly create and manage class schedules, assign instructors, and define maximum capacities. The system provides a user-friendly interface for class management, allowing staff members to easily update class information and members to view and book classes based on their preferences.

The proposed system also includes a course booking feature, enabling members to browse available courses, view class details, and make bookings. This feature simplifies the booking process and ensures efficient class capacity management. Additionally, the system incorporates an attendance tracking module that allows staff members to record and update member attendance for booked courses or classes.

Billing and payment processing are streamlined through the proposed system. It automates the generation of invoices for membership fees, course bookings, and additional services. The system integrates with secure payment gateways to facilitate online payments, ensuring a convenient and secure payment experience for members.

Furthermore, the proposed system provides robust reporting and analytics capabilities. Gym owners and staff members can access comprehensive reports on member attendance, revenue analysis, class popularity, and other key metrics. These insights enable data-driven decision-making, allowing gym owners to assess performance, identify trends, and make informed business decisions.

Overall, the proposed Gym Management System aims to optimize gym operations, enhance member satisfaction, and improve resource allocation. By automating administrative tasks, centralizing data, and providing a user-friendly interface, the system empowers gym owners and staff members to deliver exceptional fitness services while achieving greater efficiency and member experience.

2.1 Objectives of Proposed System

1. **Streamline Administrative Tasks**: The system aims to automate and streamline various administrative tasks such as membership management, class scheduling, course bookings, attendance tracking, and billing. This objective ensures efficient and accurate management of gym operations, reducing manual effort and minimizing the risk of errors.

2. **Enhance Member Experience**: The system focuses on improving the overall experience for gym members. By providing a user-friendly interface, easy course browsing and booking, and accurate attendance tracking, the system aims to enhance member satisfaction, engagement, and retention.

3. **Optimize Resource Allocation**: The proposed system seeks to optimize the allocation of resources within the gym. This includes managing class schedules, assigning instructors based

and expertise, and efficiently managing class capacities. The objective is to maximize resource utilization and provide a seamless experience for both members and staff.

4. Improve Data Accuracy and Accessibility: The system aims to centralize and maintain accurate member data, class schedules, attendance records, and financial information. This objective ensures data integrity, eliminates duplication, and enables easy access to real-time information for effectivedecision-making and reporting.

5. Facilitate Efficient Billing and Payment Processing: The system aims to automate billing processes by generating invoices for membership fees, course bookings, and additional services. By integrating with secure payment gateways, the objective is to facilitate online payments, streamline financial transactions, and ensure transparent and hassle-free billing for members.

6. Provide Reporting and Analytics: The proposed system focuses on providing comprehensive reporting and analytics capabilities. This includes generating reports on member attendance, revenue analysis, class popularity, and overall gym performance. The objective is to enable data- driven decision-making, track key metrics, identify trends, and optimize business strategies.

7. Support Scalability and Growth: The system is designed to support the scalability and growth of the gym or fitness center. It should accommodate an increasing number of members, classes, and services while maintaining system performance and efficiency. The objective is to provide a flexible and adaptable solution that can grow alongside the gym's needs.

By achieving these objectives, the proposed Gym Management System aims to improve operational efficiency, enhance member satisfaction, optimize resource utilization, and support the growth and success of the gym or fitness center.

2.2 Users Summery

The proposed Gym Management System caters to multiple user roles, each with specific responsibilities and functionalities. The user roles can be summarized as follows:

1. Gym Owners/Administrators: These users have overall control and management of the gym. They have access to administrative features, including membership management, class scheduling, and financial management. Gym owners/administrators can add and manage staff members, define membership plans and pricing, view reports and analytics, and make strategic decisions for the gym's growth and success.

2. Staff Members/Instructors: Staff members, such as instructors and receptionists, have specific responsibilities within the gym. They can access features related to class management, attendance tracking, and member engagement. Staff members can update class schedules, record member attendance, manage bookings, and interact with members to provide assistance and support.

3. Gym Members: These users are the primary customers of the gym. They can access the system to browse and book classes, manage their memberships, view their attendance records, and make payments. Gym members can explore available courses, check class availability, book sessions, and receive notifications and reminders regarding their bookings and upcoming classes.

2.3 Scope of the system Requirements

Functional Requirements of Gym Management System :
The functional requirements of the Gym Management System include the following features and functionalities:

1. User Registration and Login: Users should be able to create accounts, provide necessary information, and securely log in to the system using credentials.

2. Membership Management: The system should allow gym administrators to manage memberships, including creating membership plans, setting pricing, handling renewals and cancellations, and tracking membership statuses.

3. Class Scheduling: Gym owners and staff members should be able to create, update, and manage class schedules, including defining class names, assigning instructors, setting timings, and specifying maximum capacities.

4. Course Booking: Members should be able to browse available courses, view class details, and book classes based on their preferences and availability. The system should handle class availability checks, allow members to select sessions, and provide confirmation of their bookings.

5. Attendance Tracking: Staff members should have the ability to record and update member attendance for the booked courses or classes. The system should provide an interface to mark attendance, track member participation, and generate attendance reports.

6. Billing and Invoicing: The system should generate invoices for membership fees, course bookings, and additional services. It should handle recurring billing, calculate amounts, and provide payment due dates. Integration with secure payment gateways should enable online payments.

7. User Management: The system should have different user roles with specific permissions and access levels. User management features should include adding, editing, and deleting user accounts, as well as managing user profiles.

8. Data Management: The system should provide functionality to manage and store member information, class schedules, attendance records, billing details, and other relevant data securely.

9. User Interface and Experience: The system should have a user-friendly interface that is intuitive and easy to navigate for all users, providing a smooth and seamless experience.

Non-Functional Requirements of Gym Management System :

Non-functional requirements define the quality attributes and constraints of the Gym Management System.

1. Usability: The system should have an intuitive and user-friendly interface to ensure ease of use for gym owners, staff members, and members. It should be visually appealing and provide clear navigation, minimizing the learning curve for users.

2. Performance: The system should be able to handle a large volume of concurrent users and process requests quickly. It should have fast response times and minimal latency to ensure a smooth user experience.

3. Scalability: The system should be designed to handle future growth and scalability. It should accommodate an increasing number of users, classes, and memberships without compromisingperformance.

4. Security: The system should ensure the security and privacy of user data. It should incorporate measures such as data encryption, secure user authentication, and access control to protect sensitive information.

5. Reliability: The system should be highly reliable, with minimal downtime and system failures. Itshould have backup and recovery mechanisms in place to ensure data integrity and availability.

6. Compatibility: The system should be compatible with different web browsers and devices toallow users to access it from various platforms.

7. Maintainability: The system should be designed and developed in a modular and maintainable manner. It should have clear and well-documented code, allowing for future enhancements and updates.

8. Integration: The system should be capable of integrating with other external systems or APIs, such as payment gateways, to facilitate seamless data exchange and enhance functionality.

9. Accessibility: The system should adhere to accessibility standards, ensuring that users with disabilities can access and use the system with assistive technologies.

10. Performance Monitoring: The system should have mechanisms in place to monitor and analyze system performance, including logging, error tracking, and performance metrics, to identify and resolve issues promptly.

2.4 System requirements

2.4.1 Software Requirements :

- Technology: Python, Django.
- Web Browser
- Visual Studio Code
- SQL lite Database

2.4.2 Hardware Requirements:

- Processor: Intel/AMD dual-core or above
- RAM: 4 GB RAM
- Disk: 512 GB of SSD/HDD
- Operating system: Linux, Windows

3. Requirement determination and Analysis

3.1 Fact Finding methods

Fact-finding methods are techniques used to gather information and collect relevant data during the requirements gathering phase. These methods help in understanding the existing system, identifying user needs, and uncovering the necessary information for the development of the Gym Management System. Here are some common fact-finding methods:

1. **Interviews**: Conduct interviews with stakeholders, including gym owners, staff members, and gym members, to gather information about their requirements, preferences, and pain points. Prepare a set of structured questions and engage in open-ended discussions to delve deeper into their needs and expectations.

2. **Questionnaires and Surveys**: Create questionnaires or surveys to collect information from a largeraudience. This method allows for the gathering of data from a broader range of stakeholders and provides quantitative insights. Use online survey tools or distribute paper-based questionnaires to gather responses.

3. **Observation**: Observe the current gym management processes and workflows in action. This method involves directly observing how tasks are performed, interactions between staff members and gym members, and any existing challenges or bottlenecks. Take notes and document the observed activities.

4. **Document Analysis**: Analyze existing documents, such as user manuals, reports, invoices, and system documentation, to gain insights into the current system functionalities, data flows, and business rules. This method helps in understanding the existing processes and serves as a reference for system improvements.

5. **Prototyping and Mockups**: Create prototypes or mockups of the system's user interface and functionalities. Use these prototypes to gather feedback from stakeholders and validate their requirements. This method helps in visualizing the system and uncovering usability and designconsiderations.

6. **Workshops and Focus Groups**: Organize workshops or focus group discussions with relevant stakeholders to facilitate collaborative idea sharing and requirements gathering. Use brainstorming sessions, group discussions, and interactive activities to elicit valuable insights and consensus among participants.

7. **Site Visits**: Visit the gym or fitness center premises to gain a better understanding of the physical environment, available facilities, equipment, and layout. This method helps in identifying any constraints or considerations related to the physical aspects of the system.

8. **Data Analysis**: Analyze existing data and reports related to gym operations, membership records, attendance logs, and financial transactions. This method helps in identifying trends, patterns, and areas that require improvement.

9. **Benchmarking**: Conduct research and analysis of similar gym management systems or industry best practices to gain insights into commonly used features, functionalities, and successful

implementation strategies. This method helps in benchmarking and identifying opportunities for innovation.

10. Expert Consultation: Seek advice and consultation from industry experts, system analysts, or consultants who have expertise in gym management systems. Their knowledge and experience can provide valuable insights and recommendations for system requirements.

3.2 Feasibility study

1. Technical Feasibility: This examines whether the proposed system can be developed and implemented from a technical perspective. Considerations include the availability of necessary technology, required hardware and software resources, compatibility with existing systems, and technical expertise.

2. Economic Feasibility: This assesses the financial viability of the project. It involves analyzing thecosts and benefits associated with developing and maintaining the system. Consider factors such as development costs, operational costs, potential savings or revenue generation, return on investment (ROI), and cost-benefit analysis.

3. Operational Feasibility: This evaluates whether the proposed system aligns with the existing operational processes and workflows. It assesses the impact of implementing the system on day-to-day operations, staff responsibilities, training requirements, and the ability to adapt to changes.

4. Schedule Feasibility: This examines the feasibility of completing the project within the desiredtimeframe. Consider factors such as project milestones, development timelines, resource availability, and any external dependencies or constraints that may affect the project schedule.

5. Legal and Ethical Feasibility: Consider legal and ethical considerations that may impact the project. Assess compliance with data protection regulations, privacy laws, intellectual propertyrights, and any other legal or ethical obligations.

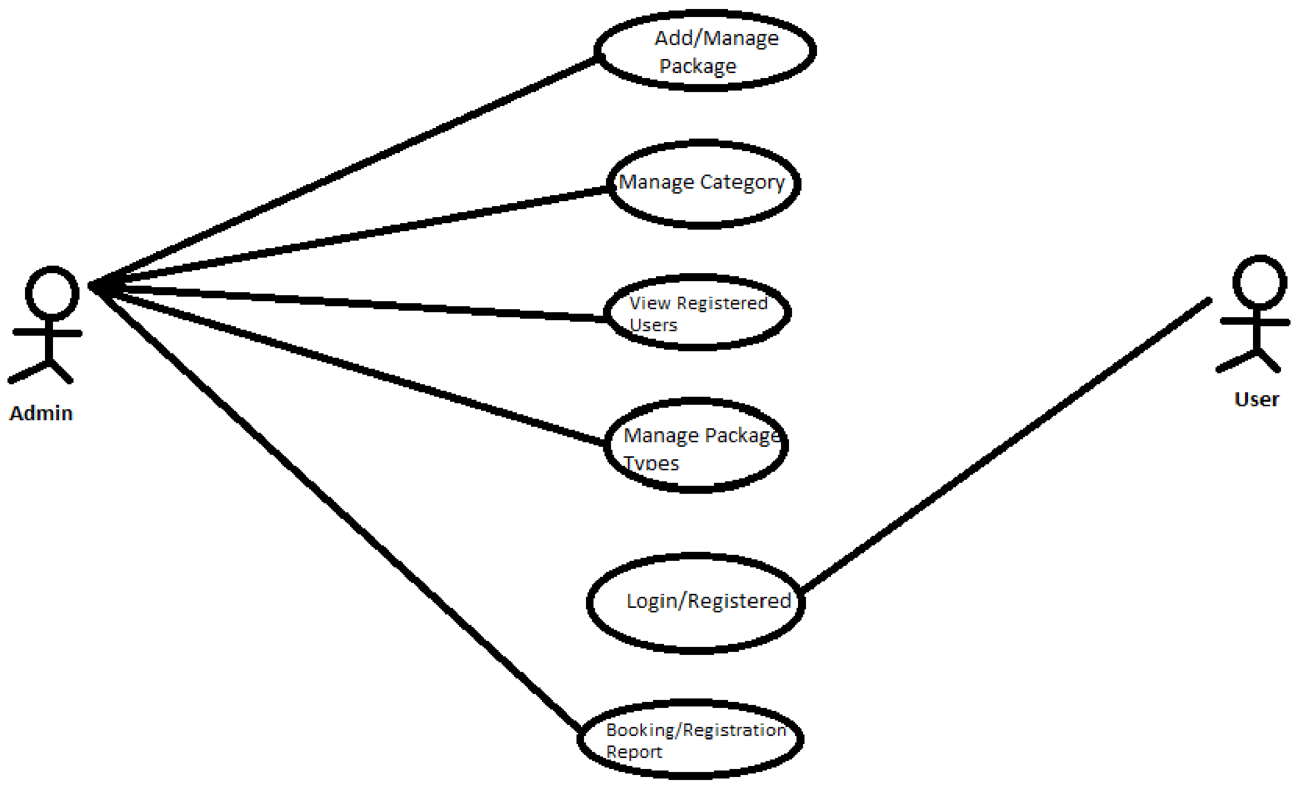
6. Organizational Feasibility: Evaluate the readiness and willingness of the organization to undertake the project. Consider factors such as management support, organizational culture,resource allocation, and the impact on stakeholders.

7. Risk Analysis: Identify potential risks and challenges associated with the project and assess their potential impact. Develop strategies to mitigate risks and ensure contingency plans are in place.

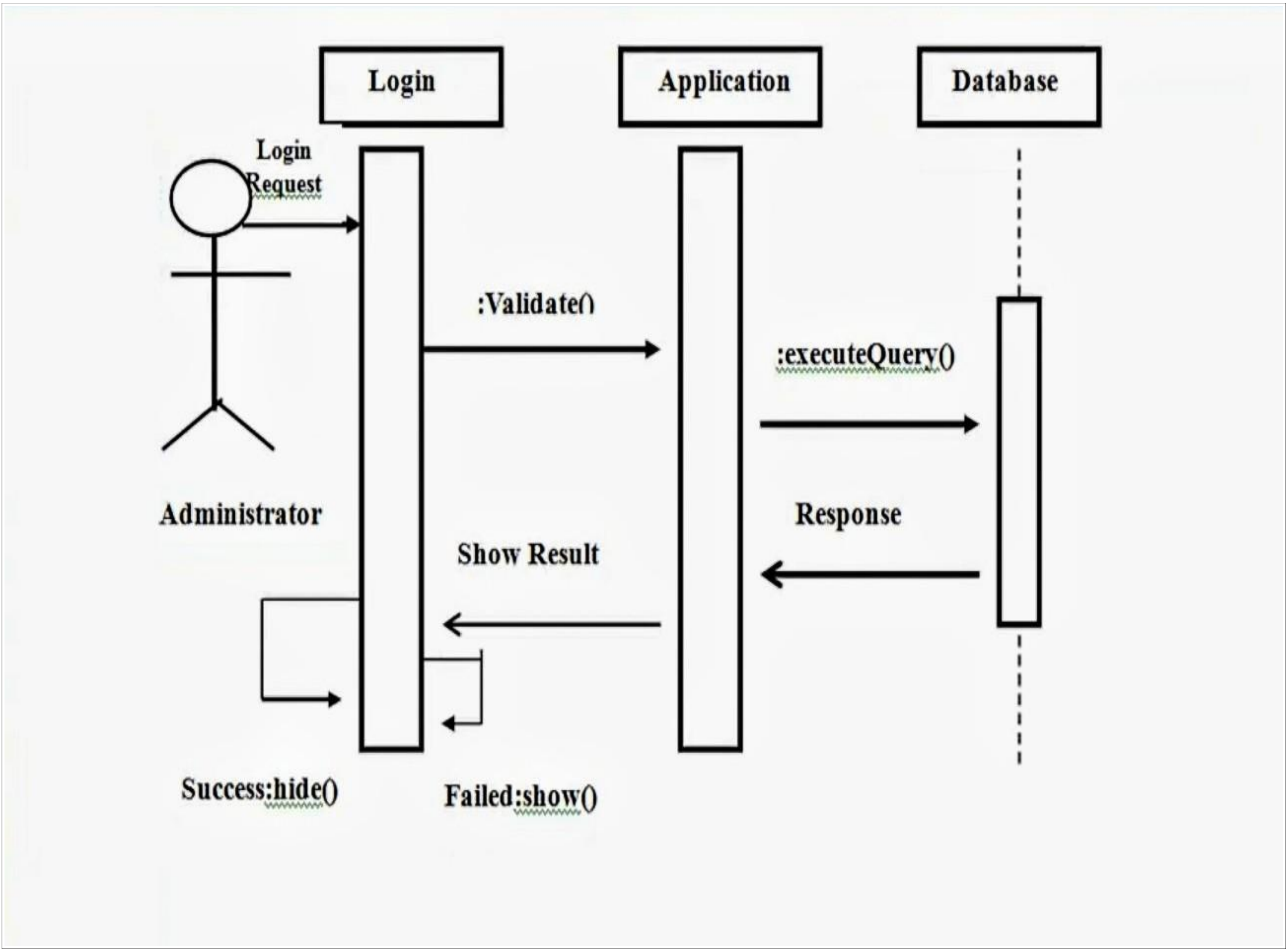
8. Recommendation: Based on the analysis of the feasibility factors, provide a recommendation on whether to proceed with the project. This recommendation should consider the findings of the study, including the feasibility aspects, potential benefits, and risks.

4. System Analysis and Design

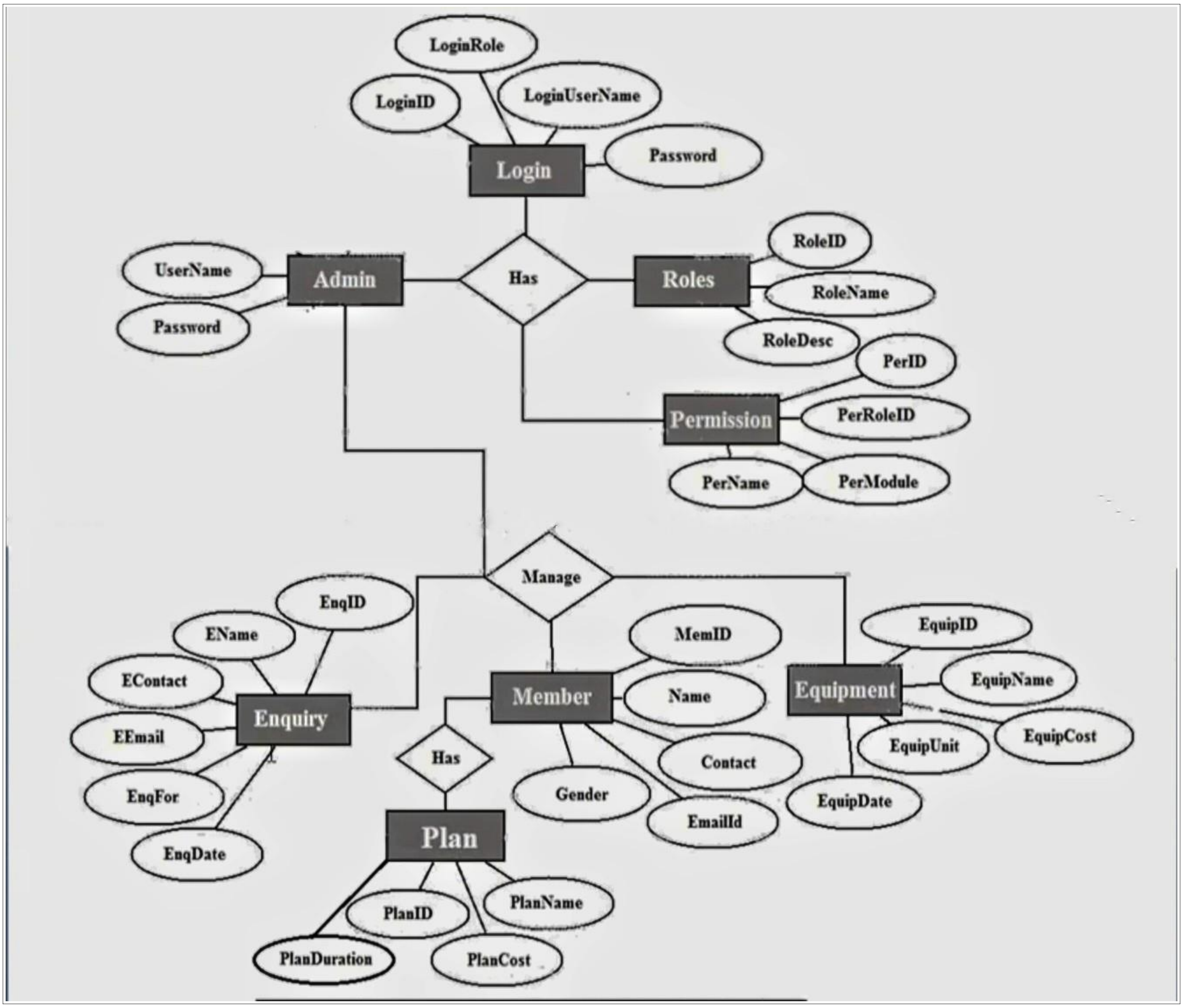
4.1 Use Case Diagram :



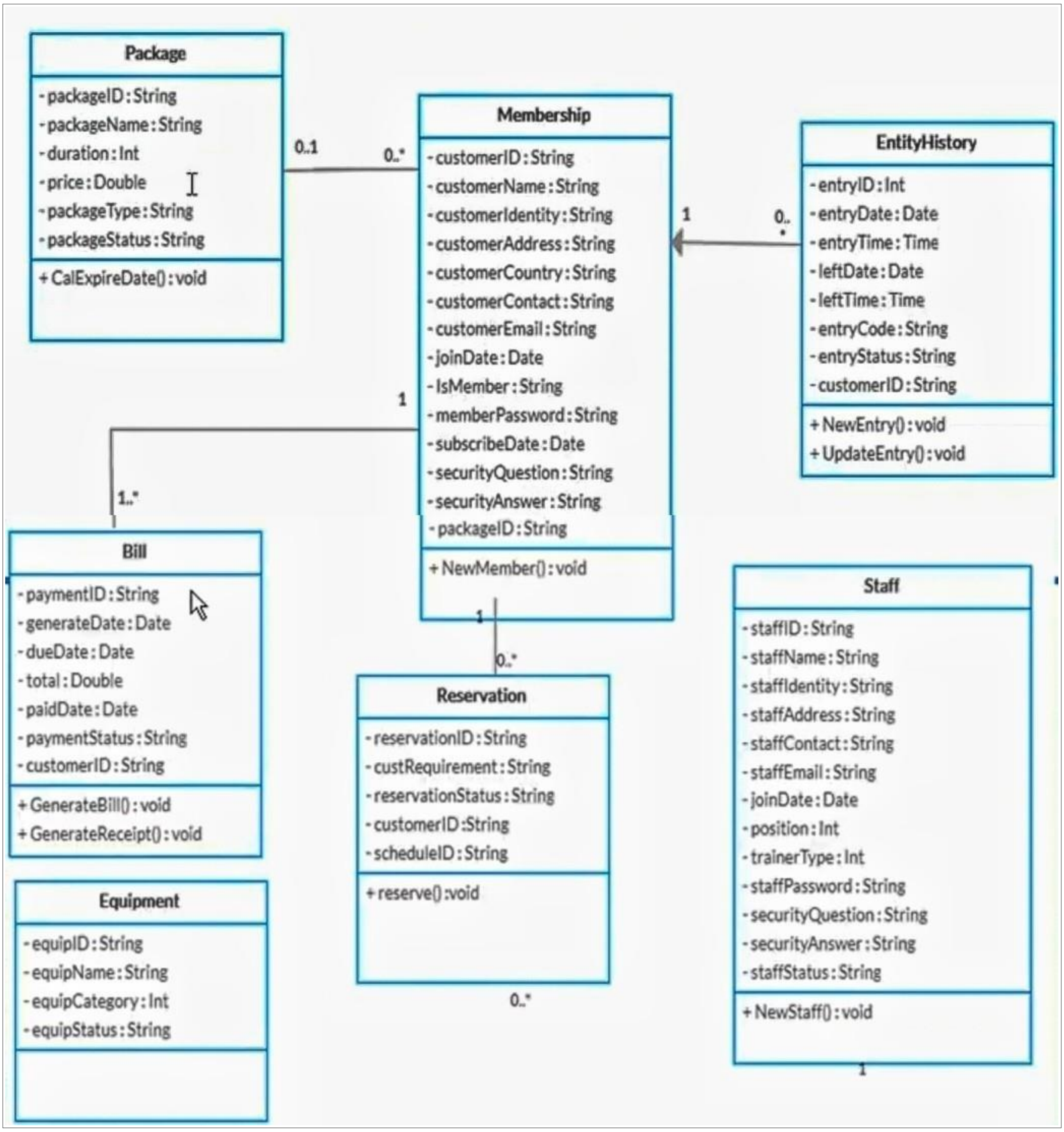
4.2 Sequence Diagram :



4.3 ER-Diagram :

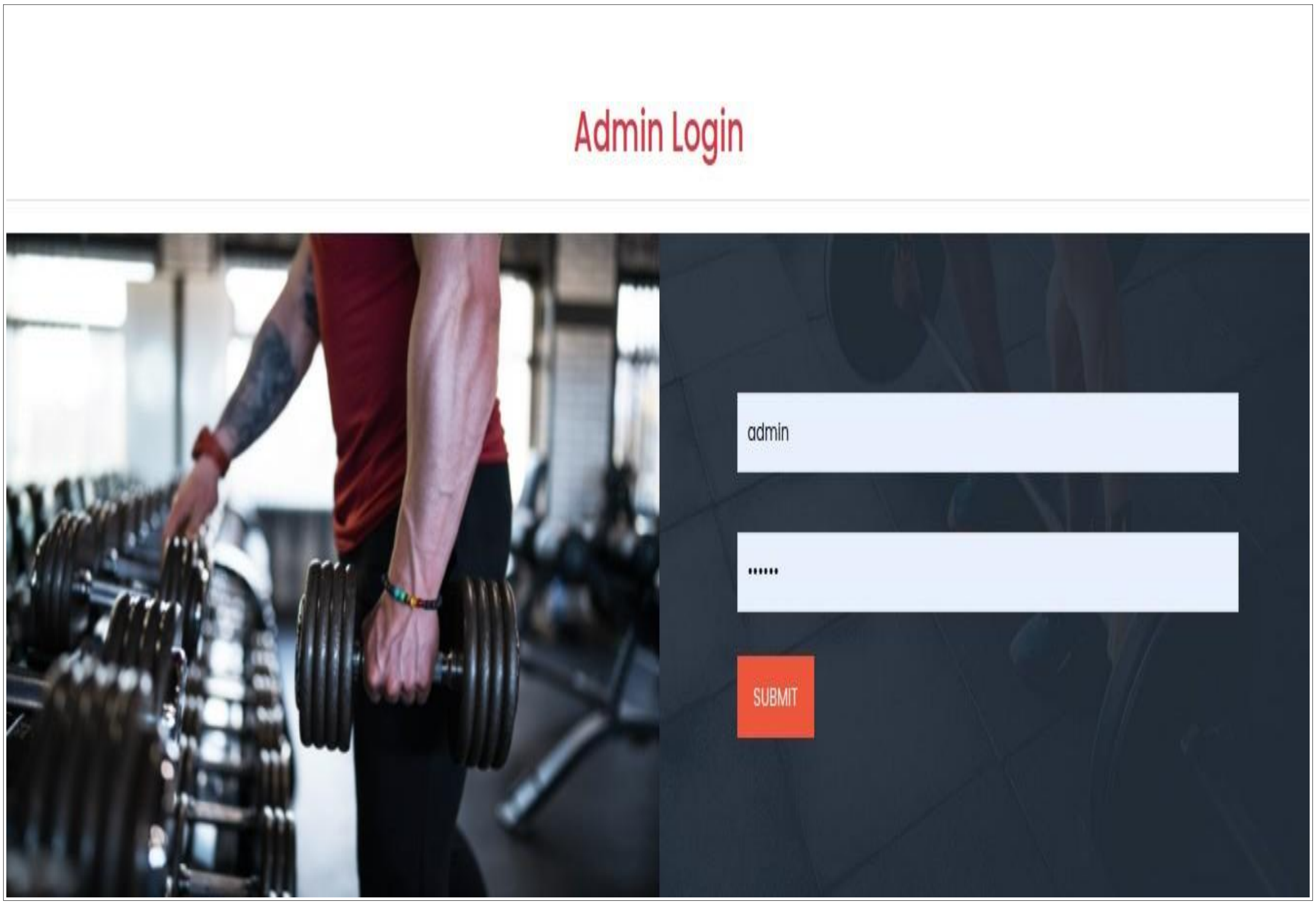


4.4 Class Diagram :

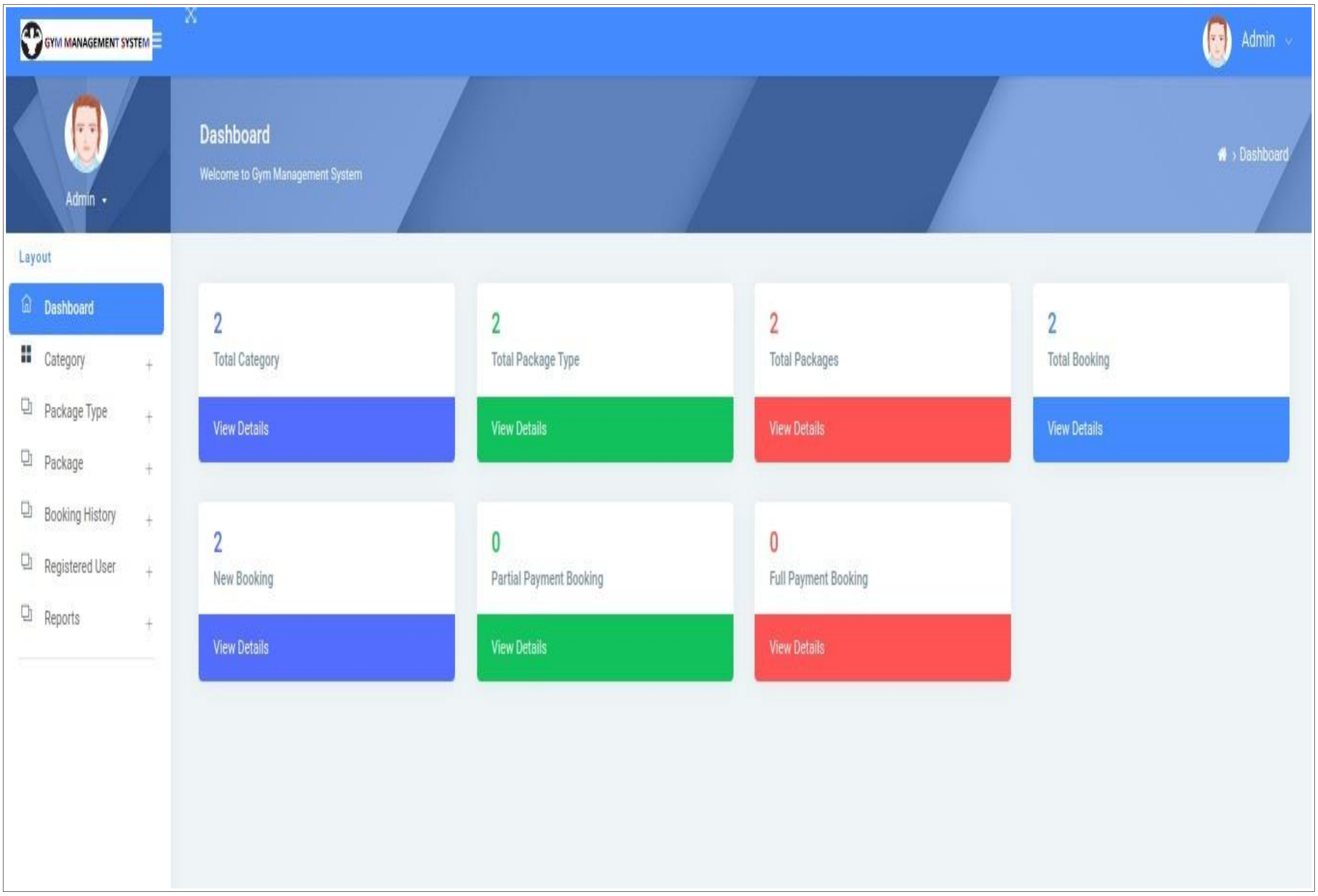


4.6 User Interface Design (Screens) :

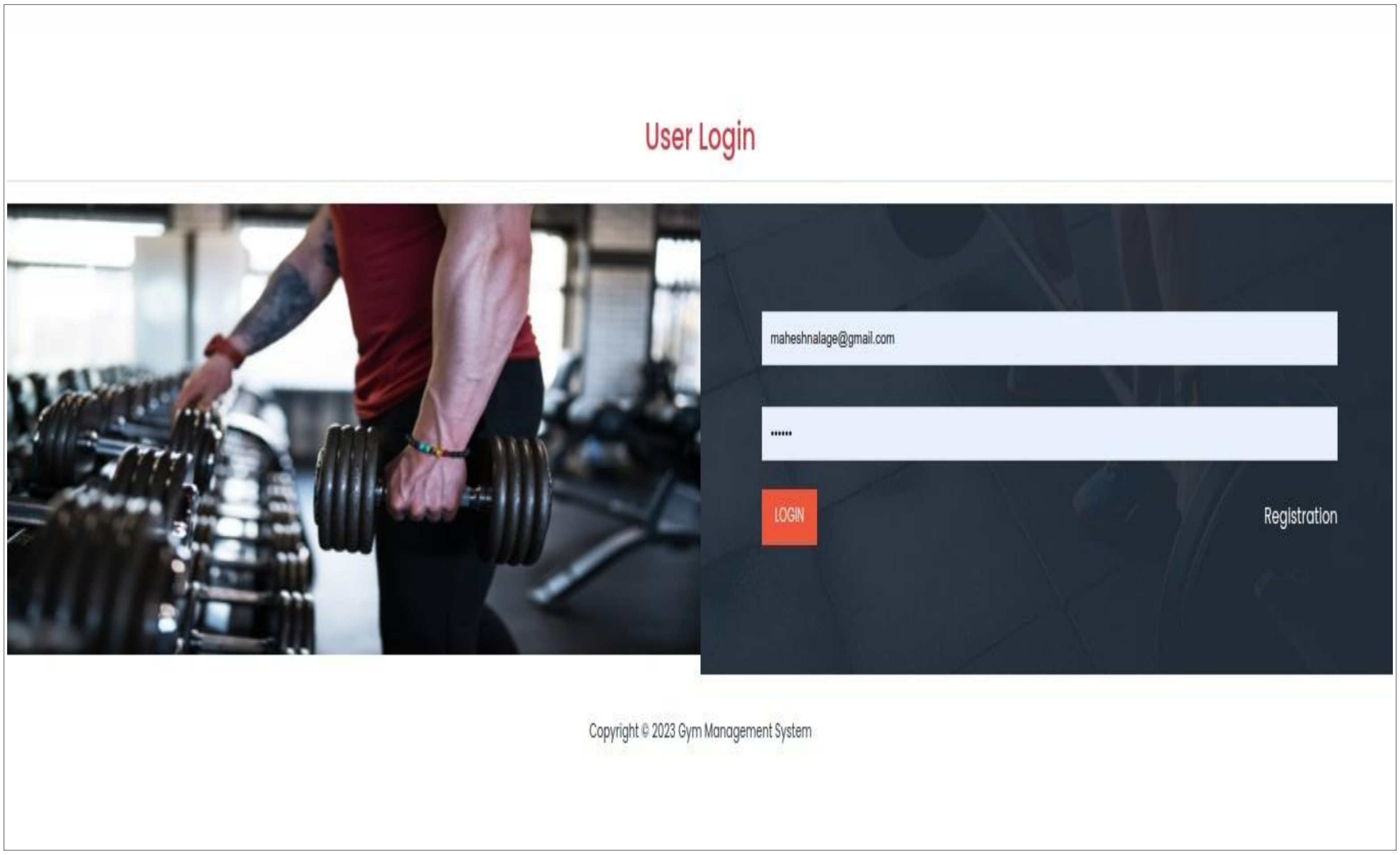
Admin Login Page -



Admin Dashboard Page -



User Login Page -



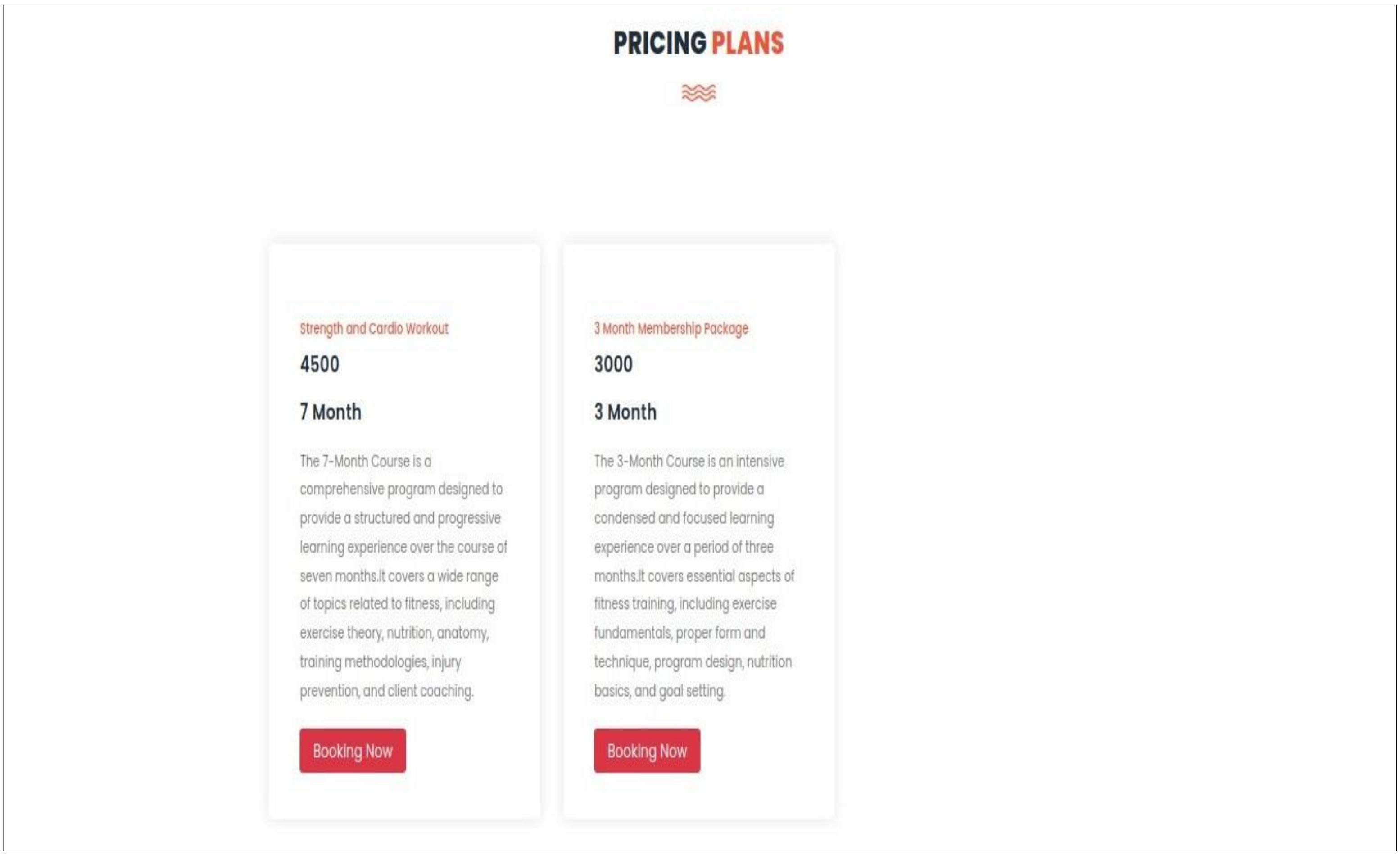
User Login

maheshnalage@gmail.com

••••••

Copyright © 2023 Gym Management System

Pricing Plans Page -



PRICING PLANS

Strength and Cardio Workout

4500

7 Month

The 7-Month Course is a comprehensive program designed to provide a structured and progressive learning experience over the course of seven months.It covers a wide range of topics related to fitness, including exercise theory, nutrition, anatomy, training methodologies, injury prevention, and client coaching.

Booking Now

3 Month Membership Package

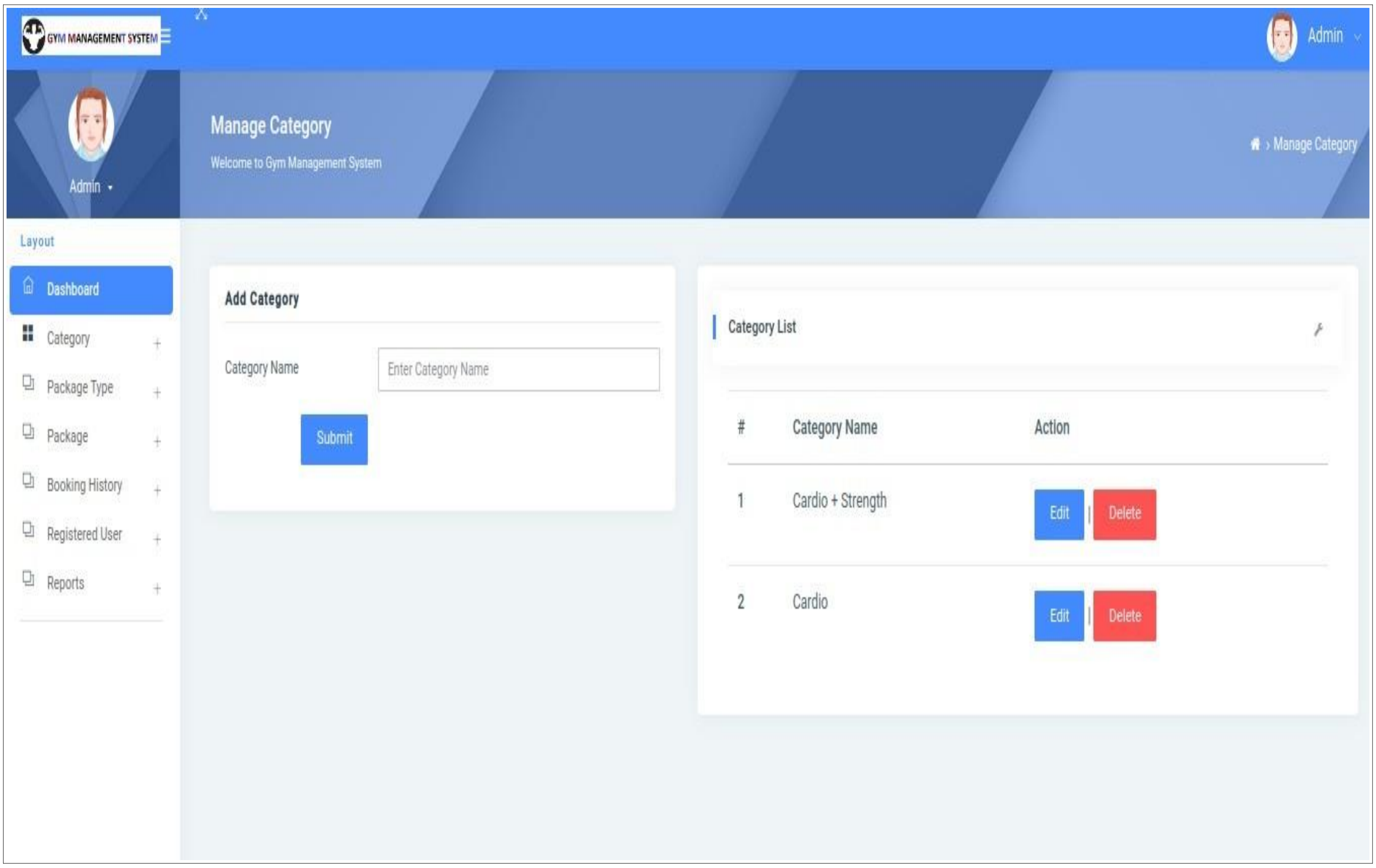
3000

3 Month

The 3-Month Course is an intensive program designed to provide a condensed and focused learning experience over a period of three months.It covers essential aspects of fitness training, including exercise fundamentals, proper form and technique, program design, nutrition basics, and goal setting.

Booking Now

Manage Category Page -



#	Category Name	Action
1	Cardio + Strength	Edit | Delete
2	Cardio	Edit | Delete

5. Coding

The coding phase of a project, such as the Job-Link online job portal, involves the actual implementation of the system based on the design specifications and requirements. It is the stage where developers write the code that brings the system to life.

1) Front-End Development:

- HTML: Used for creating the structure and content of the web pages.
- CSS: Utilized for styling and formatting the appearance of the web pages.
- JavaScript: Employed for client-side interactivity and dynamic functionalities on the web pages.

2) Back-End Development:

- Python: Serves as the primary programming language for the back-end development.
- Django: A Python web framework used for rapid development and to handle server-side logic, routing, and data processing.
- Django Templates: Used to create dynamic HTML pages by embedding Python code within HTML templates.
- Django Views: Handles requests from the front-end, performs necessary computations or data retrieval, and returns responses.
- Django Models: Defines the data models and database schemas using Python classes, allowing interaction with the database.

3) Database:

- SQLite: A lightweight and file-based relational database management system used for development and testing purposes. SQLite is seamlessly integrated with Django and supports SQL queries.

Source Code:

Admin.py

```python
from django.contrib import admin
from .models import *
# Register your models here.

admin.site.register(Packagetype)
admin.site.register(Booking)
admin.site.register(Category)
admin.site.register(Package)
admin.site.register(Signup)
admin.site.register(Paymenthistory)
```

Apps.py

```python
from django.contrib import admin
from .models import *
# Register your models here.

admin.site.register(Packagetype)
admin.site.register(Booking)
admin.site.register(Category)
admin.site.register(Package)
admin.site.register(Signup)
admin.site.register(Paymenthistory)
```

forms.py

```python
from django import forms
from .models import *
from django.contrib.auth.forms import UserCreationForm


class BookingForm(forms.ModelForm):
    class Meta:
        model = Booking
        fields = ('bookingnumber', 'status',)
```

models.py

```python
from django.db import models
from django.contrib.auth.models import User

# Create your models here.

class Category(models.Model):
    categoryname = models.CharField(max_length=200, null=True)
    status = models.CharField(max_length=300, null=True)
    creationdate = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.categoryname

class Packagetype(models.Model):
    category = models.ForeignKey(Category, on_delete=models.CASCADE, null=True)
    packagename = models.CharField(max_length=200, null=True)
    creationdate = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.packagename

class Signup(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True)
    mobile = models.CharField(max_length=15, null=True)
    state = models.CharField(max_length=150, null=True)
    city = models.CharField(max_length=150, null=True)
    address = models.CharField(max_length=200, null=True)
    creationdate = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.user.first_name

class Package(models.Model):
    category = models.ForeignKey(Category, on_delete=models.CASCADE, null=True)
    packagename = models.ForeignKey(Packagetype, on_delete=models.CASCADE, null=True)
    titlename = models.CharField(max_length=200, null=True)
    packageduration = models.CharField(max_length=50, null=True)
    price = models.CharField(max_length=200, null=True)
    description = models.CharField(max_length=200, null=True)
    creationdate = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.titlename

STATUS = ((1, "Not Updated Yet"), (2, "Partial Payment"), (3, 'Full Payment'))
class Booking(models.Model):
    package = models.ForeignKey(Package, on_delete=models.CASCADE, null=True, blank=True)
    register = models.ForeignKey(Signup, on_delete=models.CASCADE, null=True, blank=True)
    bookingnumber = models.CharField(max_length=100, null=True, blank=True)
    status = models.IntegerField(choices=STATUS, default=1)
    creationdate = models.DateTimeField(auto_now_add=True)

class Paymenthistory(models.Model):
    user = models.ForeignKey(Signup, on_delete=models.CASCADE, null=True, blank=True)
    booking = models.ForeignKey(Booking, on_delete=models.CASCADE, null=True, blank=True)
    price = models.CharField(max_length=100, null=True, blank=True)
    status = models.IntegerField(choices=STATUS, default=1)
    creationdate = models.DateTimeField(auto_now_add=True)
```

views.py

```python
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.shortcuts import render,redirect
from django.contrib.auth import authenticate, logout, login
from .models import *
from .forms import *
from random import randint

# Create your views here.
def index(request):
    if request.method == "POST":
        u = request.POST['uname']
        p = request.POST['pwd']
        user = authenticate(username=u, password=p)
        if user:
            if user.is_staff:
                login(request, user)
                messages.success(request, "Logged In Successfully")
                return redirect('admin_home')
            else:
                messages.success(request, "Invalid Credentials, Please try again")
                return redirect('index')
    package = Package.objects.filter().order_by('id')[:5]
    return render(request, 'index.html', locals())

def registration(request):
    if request.method == "POST":
        fname = request.POST['firstname']
        lname = request.POST['secondname']
        email = request.POST['email']
        pwd = request.POST['password']
        mobile = request.POST['mobile']
        address = request.POST['address']

        user = User.objects.create_user(first_name=fname, last_name=lname, email=email,
password=pwd, username=email)
        Signup.objects.create(user=user, mobile=mobile,address=address)
        messages.success(request, "Register Successful")
        return redirect('user_login')
    return render(request, 'registration.html', locals())

def user_login(request):
    if request.method == "POST":
        email = request.POST['email']
        pwd = request.POST['password']
        user = authenticate(username=email, password=pwd)
        if user:
            if user.is_staff:
                messages.success(request, "Invalid User")
                return redirect('user_login')
            else:
                login(request, user)
                messages.success(request, "User Login Successful")
                return redirect('index')
        else:
            messages.success(request, "Invalid User")
            return redirect('user_login')
    return render(request, 'user_login.html', locals())

def admin_home(request):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    totalcategory = Category.objects.all().count()
    totalpackagetype = Packagetype.objects.all().count()
    totalpackage = Package.objects.all().count()
    totalbooking = Booking.objects.all().count()
    New = Booking.objects.filter(status="1")
    Partial = Booking.objects.filter(status="2")
    Full = Booking.objects.filter(status="3")
    return render(request, 'admin/admin_home.html', locals())

def Logout(request):
    logout(request)
    messages.success(request, "Logout Successfully")
    return redirect('index')

def user_logout(request):
    logout(request)
    messages.success(request, "Logout Successfully")
    return redirect('index')

def user_profile(request):
    if request.method == "POST":
        fname = request.POST['firstname']
        lname = request.POST['secondname']
        email = request.POST['email']
        mobile = request.POST['mobile']
        address = request.POST['address']

        user = User.objects.filter(id=request.user.id).update(first_name=fname,
last_name=lname, email=email)
        Signup.objects.filter(user=request.user).update(mobile=mobile, address=address)
        messages.success(request, "Updation Successful")
        return redirect('user_profile')
    data = Signup.objects.get(user=request.user)
    return render(request, "user_profile.html", locals())

def user_change_password(request):
    if request.method=="POST":
        n = request.POST['pwd1']
        c = request.POST['pwd2']
        o = request.POST['pwd3']
        if c == n:
            u = User.objects.get(username__exact=request.user.username)
            u.set_password(n)
            u.save()
            messages.success(request, "Password changed successfully")
            return redirect('/')
        else:
            messages.success(request, "New password and confirm password are not same.")
            return redirect('user_change_password')
    return render(request,'user_change_password.html')

def manageCategory(request):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    category = Category.objects.all()
    try:
        if request.method == "POST":
            categoryname = request.POST['categoryname']

            try:
                Category.objects.create(categoryname=categoryname)
                error = "no"
            except:
                error = "yes"
    except:
        pass
    return render(request, 'admin/manageCategory.html', locals())

def editCategory(request, pid):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    error = ""
    category = Category.objects.get(id=pid)
    if request.method == "POST":
        categoryname = request.POST['categoryname']

        category.categoryname = categoryname

        try:
            category.save()
            error = "no"
        except:
            error = "yes"
    return render(request, 'admin/editCategory.html', locals())

def deleteCategory(request, pid):
    if not request.user.is_authenticated:
```

```python
        return redirect('admin_login')
    category = Category.objects.get(id=pid)
    category.delete()
    return redirect('manageCategory')

@login_required(login_url='/admin_login/')
def reg_user(request):
    data = Signup.objects.all()
    d = {'data': data}
    return render(request, "admin/reg_user.html", locals())

@login_required(login_url='/admin_login/')
def delete_user(request, pid):
    data = Signup.objects.get(id=pid)
    data.delete()
    messages.success(request, "Delete Successful")
    return redirect('reg_user')

def managePackageType(request):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    package = Packagetype.objects.all()
    category = Category.objects.all()
    try:
        if request.method == "POST":
            cid = request.POST['category']
            categoryid = Category.objects.get(id=cid)

            packagename = request.POST['packagename']

            try:
                Packagetype.objects.create(category=categoryid, packagename=packagename)
                error = "no"
            except:
                error = "yes"
    except:
        pass
    return render(request, 'admin/managePackageType.html', locals())

def editPackageType(request, pid):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    category = Category.objects.all()
    package = Packagetype.objects.get(id=pid)
    if request.method == "POST":
        cid = request.POST['category']
        categoryid = Category.objects.get(id=cid)
        packagename = request.POST['packagename']
        package.category = categoryid
        package.packagename = packagename

        try:
            package.save()
            error = "no"
        except:
            error = "yes"
    return render(request, 'admin/editPackageType.html', locals())


def deletePackageType(request, pid):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    package = Packagetype.objects.get(id=pid)
    package.delete()
    return redirect('managePackageType')

def addPackage(request):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    category = Category.objects.all()
    packageid = request.GET.get('packagename', None)
    mypackage = None
    if packageid:
        mypackage = Packagetype.objects.filter(packagename=packageid)
    if request.method == "POST":
        cid = request.POST['category']
        categoryid = Category.objects.get(id=cid)
        packagename = request.POST['packagename']
        packageobj = Packagetype.objects.get(id=packagename)
        titlename = request.POST['titlename']
        duration = request.POST['duration']
        price = request.POST['price']
        description = request.POST['description']

        try:
            Package.objects.create(category=categoryid,packagename=packageobj,
                                   titlename=titlename,
packageduration=duration,price=price,description=description)
            error = "no"
        except:
            error = "yes"
    mypackage = Packagetype.objects.all()
    return render(request, 'admin/addPackage.html',locals())

def managePackage(request):
    package = Package.objects.all()
    return render(request, 'admin/managePackage.html',locals())

@login_required(login_url='/user_login/')
def booking_history(request):
    data = Signup.objects.get(user=request.user)
    data = Booking.objects.filter(register=data)
    return render(request, "booking_history.html", locals())

@login_required(login_url='/admin_login/')
def new_booking(request):
    action = request.GET.get('action')
    data = Booking.objects.filter()
    if action == "New":
        data = data.filter(status="1")
    elif action == "Partial":
        data = data.filter(status="2")
    elif action == "Full":
        data = data.filter(status="3")
    elif action == "Total":
        data = data.filter()
    if request.user.is_staff:
        return render(request, "admin/new_booking.html", locals())
    else:
        return render(request, "booking_history.html", locals())


def booking_detail(request, pid):
    data = Booking.objects.get(id=pid)
    if request.method == "POST":
        price = request.POST['price']
        status = request.POST['status']
        data.status = status
        data.save()
        Paymenthistory.objects.create(booking=data, price=price, status=status)
        messages.success(request, "Action Updated")
        return redirect('booking_detail', pid)
    payment = Paymenthistory.objects.filter(booking=data)
    if request.user.is_staff:
        return render(request, "admin/admin_booking_detail.html", locals())
    else:
        return render(request, "user_booking_detail.html", locals())

def editPackage(request, pid):
    category = Category.objects.all()
    if request.method == "POST":
        cid = request.POST['category']
        categoryid = Category.objects.get(id=cid)
        packagename = request.POST['packagename']
        packageobj = Packagetype.objects.get(id=packagename)
        titlename = request.POST['titlename']
        duration = request.POST['duration']
        price = request.POST['price']
        description = request.POST['description']

        Package.objects.filter(id=pid).update(category=categoryid,packagename=packageobj,
                                           titlename=titlename,
packageduration=duration,price=price,description=description)
        messages.success(request, "Updated Successful")
        return redirect('managePackage')
    data = Package.objects.get(id=pid)
    mypackage = Packagetype.objects.all()
    return render(request, "admin/editPackage.html", locals())

def load_subcategory(request):
    categoryid = request.GET.get('category')
    subcategory = Package.objects.filter(category=categoryid).order_by('PackageName')
    return render(request,'subcategory_dropdown_list_options.html',locals())

def deletePackage(request, pid):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    package = Package.objects.get(id=pid)
    package.delete()
    return redirect('managePackage')

def deleteBooking(request, pid):
    booking = Booking.objects.get(id=pid)
    booking.delete()
    messages.success(request, "Delete Successful")
    return redirect('new_booking')

def bookingReport(request):
    data = None
    data2 = None
    if request.method == "POST":
        fromdate = request.POST['fromdate']
        todate = request.POST['todate']

        data = Booking.objects.filter(creationdate__gte=fromdate,
creationdate__lte=todate)
        data2 = True
    return render(request, "admin/bookingReport.html", locals())

def regReport(request):
    data = None
    data2 = None
    if request.method == "POST":
        fromdate = request.POST['fromdate']
        todate = request.POST['todate']

        data = Signup.objects.filter(creationdate__gte=fromdate, creationdate__lte=todate)
        data2 = True
    return render(request, "admin/regReport.html", locals())


def changePassword(request):
    if not request.user.is_authenticated:
        return redirect('admin_login')
    error = ""
    user = request.user
    if request.method == "POST":
        o = request.POST['oldpassword']
        n = request.POST['newpassword']
        try:
            u = User.objects.get(id=request.user.id)
            if user.check_password(o):
                u.set_password(n)
                u.save()
                error = "no"
            else:
                error = 'not'
        except:
            error = "yes"
    return render(request, 'admin/changePassword.html',locals())

def random_with_N_digits(n):
    range_start = 10**(n-1)
    range_end = (10**n)-1
    return randint(range_start, range_end)

# @login_required(login_url='/user_login/')
# def booking(request):
#     booking = None
#     bookinged = Booking.objects.filter(register__user=request.user)
#     bookinged_list = [i.policy.id for i in bookinged]
#     data = Package.objects.filter().exclude(id__in=bookinged_list)
#     if request.method == "POST":
#         booking = Package.objects.filter()
#         booking = BookingForm(request.POST, request.FILES, instance=booking)
#         if booking.is_valid():
#             booking = booking.save()
#             booking.bookingnumber = random_with_N_digits(10)
#             data.booking = booking
#             data.save()
#         Booking.objects.create(package=booking)
#         messages.success(request, "Action Updated")
#         return redirect('booking')
#     return render(request, "/", locals())

@login_required(login_url='/user_login/')
def apply_booking(request, pid):
    data = Package.objects.get(id=pid)
    register = Signup.objects.get(user=request.user)
    booking = Booking.objects.create(package=data, register=register,
bookingnumber=random_with_N_digits(10))
    messages.success(request, 'Booking Applied')
    return redirect('/')
```

asgi.py

```
"""
ASGI config for GymManagementSystem project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'GymManagementSystem.settings')

application = get_asgi_application()
```

settings.py

```
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'ib7jq+u@3&w$flar34(yx%u*lucwj_m--n*6a$^fo(_fh^e&vi'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

```
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'gym',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'GymManagementSystem.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'GymManagementSystem.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases
```

```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
MEDIA_URL='/media/'
MEDIA_ROOT=os.path.join(BASE_DIR,'media')
```

```
DEFAULT_AUTO_FIELD='django.db.models.AutoField'
```

urls.py

```
from django.contrib import admin
from django.urls import path
from gym.views import *
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', index, name='index'),
    path('admin_home/', admin_home, name='admin_home'),
    path('logout/',Logout, name="logout"),
    path('user_logout/',user_logout, name="user_logout"),
    path('user_profile/', user_profile, name="user_profile"),
    path('user_change_password/', user_change_password, name="user_change_password"),
    path('booking_history/', booking_history, name='booking_history'),
    path('manageCategory/', manageCategory, name='manageCategory'),
    path('editCategory/<int:pid>', editCategory, name='editCategory'),
    path('deleteCategory/<int:pid>', deleteCategory, name='deleteCategory'),
    path('managePackageType/', managePackageType, name='managePackageType'),
    path('editPackageType/<int:pid>', editPackageType, name='editPackageType'),
    path('deletePackageType/<int:pid>', deletePackageType, name='deletePackageType'),
    path('reg_user/', reg_user, name="reg_user"),
    path('delete_user/<int:pid>', delete_user, name="delete_user"),
    path('deleteBooking/<int:pid>', deleteBooking, name='deleteBooking'),
    path('addPackage', addPackage, name='addPackage'),
    path('managePackage/', managePackage, name='managePackage'),
    path('deletePackage/<int:pid>', deletePackage, name='deletePackage'),
    path('new_booking/', new_booking, name='new_booking'),
    path('bookingReport/', bookingReport, name='bookingReport'),
    path('regReport/', regReport, name='regReport'),
    path('changePassword', changePassword, name='changePassword'),
    path('editPackage/<int:pid>', editPackage, name='editPackage'),
    path('registration',registration, name="registration"),
    path('user_login/',user_login, name="user_login"),
    path('apply-booking/<int:pid>/', apply_booking, name="apply_booking"),
    path('booking_detail/<int:pid>/', booking_detail, name="booking_detail"),

]+static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Wsgi.py

```python
"""
WSGI config for GymManagementSystem project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.1/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'GymManagementSystem.settings')

application = get_wsgi_application()
```

manage.py

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'GymManagementSystem.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

6. Testing

Testing is vital to the success of any system. Testing is done at different stages within the development phase. System testing makes a logical assumption that all parts of the system is correct and the goals will be achieved successfully. Inadequate testing or no testing leads to errors that may come up after a long time when correction would be extremely difficult. Another objective of testing is its utility as a user-oriented vehicle before implementation. The testing of the system was done on both test and user data. The following tests are performed. Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive. A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

6.1 Strategic Approach to Testing

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decrease the level of abstraction on each turn. A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.

6.2 Unit Testing

In unit testing the focuses is on the verification of the smallest unit of the project that is a module or a function. In unit testing we work according to white box testing that is providing the input set and checking the output is in accordance with the expected output or not.

5.2.1 White Box Testing

This type of testing ensures that

- All the independent modules and function should be executed at least once in the testing phase.
- All the inputs of must include the boundary values & middle values.
- All the logical decisions must be have output as true or false.

To follow the concept of white box testing we have tested each of the above mentioned forms. Tests are done to ensure correct flow of data in the system. All conditions of the system are exercised to check their accuracy.

5.2.2 Conditional Testing

In Conditional testing, each condition is tested to both true and false aspects. And all the resulting paths of true and false output are tested. So that each path that may be generate on particular condition is traced to uncover any possible errors in the system. All the conditions like selecting the option button for pdf and word document, not selecting the option button, providing the file path, not providing the file path etc.

5.2.3 Data Flow Testing

Data Flow Testing selects the path of the program according to the location of definition and use of variables. This kind of testing is used only when some local variable were declared and their scope in the program is to be tested. The definition-use chain method is used in this type of testing. It is used in situations like selecting the word document option button and then checking file is converted to pdf format.

- **Test Cases**

Sr.No	**Data Input**	**Expected Output**	**Actual Output**	**Pass/Fail**
1	Error *All fields are empty	Error *All fields are compulsory	Error *All fields are compulsory	Pass
2	Username	Error *Username is compulsory	Error Message: *Invalid Username	Pass
3	Email	Error *Email is compulsory	Error Message: *Invalid Email	Pass
4	Password	Success	Error Message: Invalid Password	Pass
5	Register	Error *User already exist use different Email	Register Successful	Pass
6	Login	Error: *Email or Password is wrong	Login Successful	Pass

7. Limitations and Enhancements

Limitations:

1. Scalability The current system may have limitations in scaling up to accommodate a large number of users, classes, and bookings. As the user base grows, the system may experience performance issues or become less responsive.

2. Customization : The system may lack flexibility in terms of allowing users to customize their preferences or tailor their workout plans according to their specific needs. It may have limited options for personalization and adaptability.

3. Reporting and Analytics: The system may lack comprehensive reporting and analytics features. It may not provide in-depth insights into key metrics, such as class attendance, user engagement, or revenue analysis, which are crucial for gym management decision-making.

Enhancements:

1. Integration with Payment Gateways: Enhancing the system to integrate with popular payment gateways would enable users to make online payments for memberships, classes, and other services. This would improve convenience and streamline the payment process.

2. Mobile Application: Developing a mobile application alongside the website would provide users with on-the-go access to the system. They can easily book classes, manage their profile, receive notifications, and track their fitness progress through a mobile app.

3. Automated Waitlist Management: Implementing a waitlist management feature would allow users to join a waitlist for fully booked classes. When a spot becomes available, users on the waitlist can be automatically notified and given the option to confirm their booking.

4. Advanced Reporting and Analytics: Enhancing the reporting capabilities of the system would provide detailed insights into key performance indicators, such as class attendance, user retention, and revenue trends. Customizable reports and visualizations would aid in decision-making and business planning.

5. Integration with Wearable Devices: Integrating the system with popular fitness trackers or wearable devices would enable users to sync their workout data, track their progress, and receive personalized recommendations based on their fitness goals.

6. Social Media Integration: Adding social media integration would allow users to share their workout achievements, class bookings, or fitness goals on platforms like Facebook, Instagram, or Twitter, fostering a sense of community and promoting the gym's services.

8. Conclusion

In conclusion, the Gym Management System is a comprehensive solution designed to streamline and automate various processes within a gym or fitness center. Throughout this project, we have identified the existing system's limitations and recognized the need for an efficient and user-friendly system. The proposed system aims to address these limitations and provide a range of functionalities that benefit both gym owners and members.

By implementing the Gym Management System, users will have the ability to register and log in to their accounts, view class schedules, book classes, manage their bookings, view membership details, renew memberships, and update their profiles. These features enhance user convenience, improve accessibility to gym services, and promote a personalized and engaging experience.

The objectives of the proposed system include improving operational efficiency, enhancing user experience, and facilitating effective management of gym resources. The system's functionality aligns with the requirements gathered through various fact-finding methods, ensuring that it meets the needs and expectations of stakeholders.

Furthermore, the feasibility study conducted highlights the technical, economic, operational, scheduling, legal, and ethical aspects of the project. This study assures that the proposed system is both feasible and practical to implement, considering the available resources, potential benefits, and associated risks.

However, it is important to acknowledge that every project has limitations and areas for further enhancement. The proposed system may require scalability improvements to handle a larger user base and customization options to cater to individual needs. Additionally, incorporating features such as integration with payment gateways, a mobile application, automated waitlist management, advanced reporting and analytics, integration with wearable devices, and social media integration would further enhance the system's capabilities and user experience.

Overall, the Gym Management System offers a solid foundation for managing and optimizing gym operations, enhancing member satisfaction, and driving business growth. By addressing the identified limitations and considering the suggested enhancements, the system has the potential to revolutionize gym management, fostering a seamless and enjoyable fitness experience for both gym owners and members.

9. Bibliography

- Google

  https://www.google.com
- YouTube
  https://www.youtube.com
- Django project

  https://www.djangoproject.com/
- SQLlite

  https://www.sqlite.org/index.html
- Stack overflow

  https://stackoverflow.com
- w3schools

  https://www.w3schools.com

10. Annexure

1. User Interface Designs:
   - Home Page Design
   - User Registration Page Design
   - User Login Page Design
   - Class Schedule Page Design
   - Booking Page Design
   - User Profile Page Design

2. Database Schema:
   - Entity-Relationship Diagram (ERD) illustrating the database structure
   - Table definitions and attributes for entities such as Users, Classes, Bookings, Memberships, etc.

3. Test Cases:
   - User Registration Test Cases
   - User Login Test Cases
   - Class Booking Test Cases
   - Membership Renewal Test Cases
   - Error Handling Test Cases

4. User Manuals:
   - User Manual for Gym Members
   - User Manual for Gym Administrators
   - Step-by-step guides on using various features of the system

5. Technical Documentation:
   - System Architecture Diagram
   - Component Diagram
   - API Documentation for integrating with external systems
   - Deployment Guide for setting up the system in a production environment

6. Glossary:
   - Definitions of key terms and acronyms used in the project

7. References:
   - List of books, articles, websites, and other resources used for research and development of the system