AIRPORT MANAGEMENT SYSTEM

Database Management Systems
Course Code: CS-222

Report by:

Aaliyan Mansoor CS-21146
Syed Shahzain Hassan CS-21128
Faizan Soomro CS-21090
Sameed Fayiz CS-21127

Batch: 2021
Submission Date: 19/07/2023

Department of Computer and Information Systems Engineering
NED University of Engineering And Technology

Table of Contents

Abstract

The Airport Management System is a web-based application designed to enhance the airport experience for travelers and provide efficient management for administrators. By accessing the system through the main page, users can easily view and toggle between departure and arrival records. The navigation bar ensures quick access to relevant information, while the service section showcases airport amenities. The tickets section allows users to view and book flights, and a dedicated "my ticket" section enables users to manage their bookings. The project aims to streamline operations and improve user experiences within the airport environment.

1. Introduction

1.1 Overview

The database system developed for this project aims to solve various organizational problems related to airport and airline management. Some potential problems that could be addressed include:

1. **Efficient management of airport spaces:** The AIRPORT_PLACE and TERMINAL tables allow for tracking and efficient utilization of different places and terminal capacities within the airport. This helps optimize passenger flow and space allocation.

2. **Restaurant information and management:** The RESTAURANTS table enables the organization to keep track of the restaurants within the airport, their brand names, and potentially other relevant details for better management and customer service.

3. **Organization structure and personnel management:** The DEPARTMENT, SECURITY, ENGINEER, and STAFF tables provide a means to organize and manage different departments, security personnel, engineers, and staff members within the airport or airline organization. This facilitates efficient personnel management and ensures smooth operations.

4. **Airline and ticket management:** The AIRLINE_OPERATES, TICKETS, TICKETS_LUGGAGE, and LUGGAGE_DETAILS tables allow for effective

management of airlines, their operations, ticketing information, and luggage details associated with each ticket. This helps streamline ticketing processes and luggage handling.

Overall, the database system developed for this project aims to improve the efficiency, organization, and management of various aspects within an airport or airline organization, ultimately enhancing the overall customer experience and operational effectiveness.

1.2 Requirement

The database design for Airport Management System requires developing a model for the database which is efficient enough to store data about Airlines, departmental information, and Airport places like terminals and others. The system will enable the airport to manage its scheduling, bookings, revenue, and other transactions for tickets. The Database will manage information related to departments and it will also take care of airline slots and places.
The tickets will help the customer or end-user view the details of tickets associated with unique airlines. The user can buy the tickets. The next important feature is to construct a system that will provide the airport with the tools to monitor the airline terminals and manage departments for engineers, staff and security personnel. The system will be designed to be user-friendly and accessible to both the airport and the user. The project aims to provide an all-in-one solution for the airport to manage their operations, improve customer experience, and address its challenges.

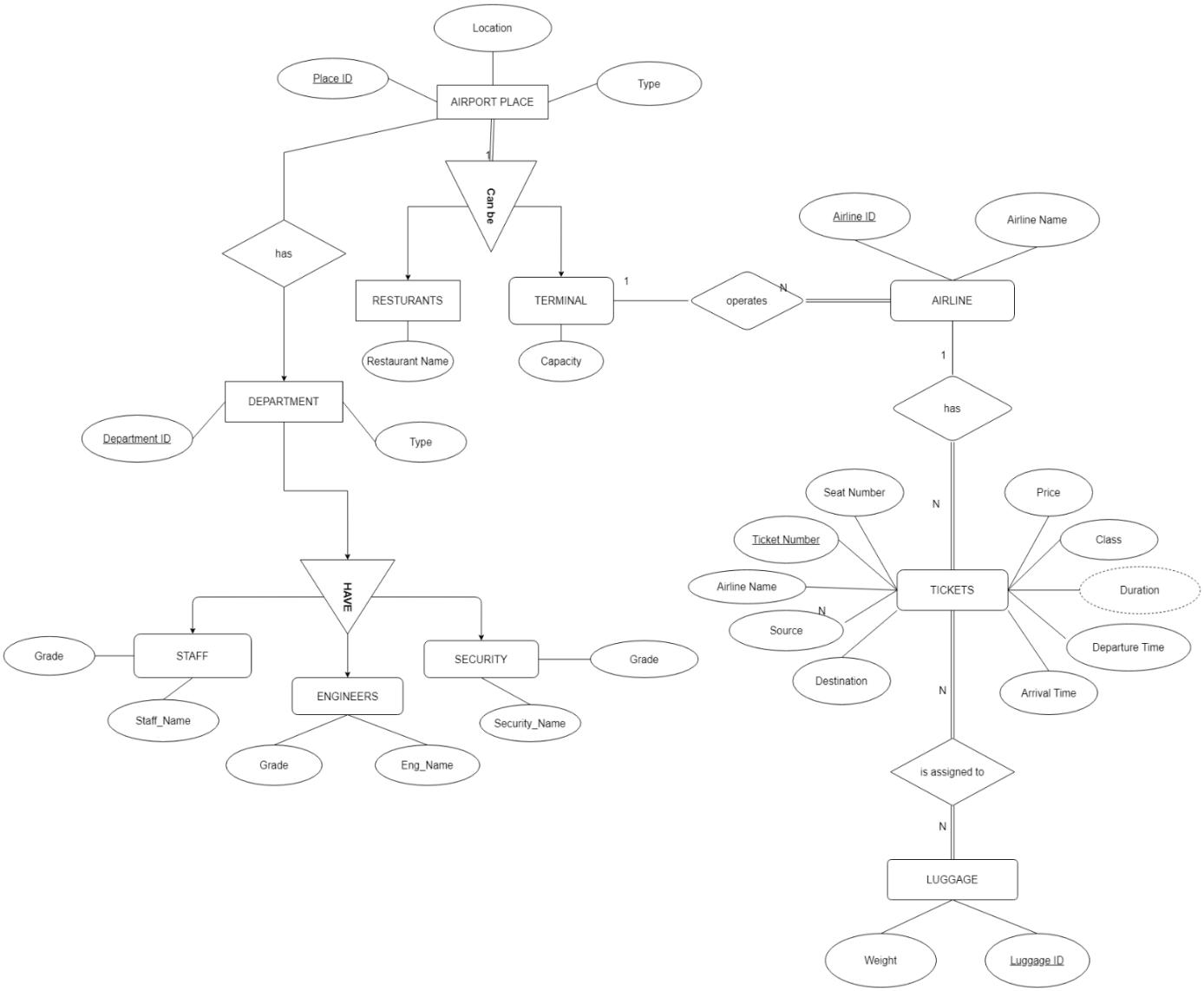
1.3 Formulation and Analysis

The thought process of designing the database for an Airport Management System is a crucial step in ensuring efficient data management and seamless operation of the system. The formulation process involves systematically analyzing the requirements, identifying the entities, relationships and attributes relevant to the airport domain, and representing them using an Entity-Relationship (ER) model. This process serves as the foundation for the subsequent steps of designing and implementing the database schema.

By carefully formulating the ER model, we can capture the essential components of the system, establish meaningful relationships between entities, and define the necessary attributes to store and retrieve information accurately. Through this process, we aim to create a comprehensive database that supports the various functionalities of the Airport Management System and facilitates smooth operations for airlines, departments, and other entity sets involved.

2. Conceptual Design

2.1 Entity-Relationship Model



2.2 Relations

AIRPORT_PLACE (placeID, location, type)

RESAURANTS (placeID, brand)

TERMINAL (placeID, capacity)

DEPARTMENT (IDNo, type)

SECURITY (IDNo, secName, grade)

ENGINEER (IDNo, engName, grade)

STAFF (IDNo, staffName, grade)

AIRLINE_OPERATES (airlineID, airlineName, capacity)

TICKETS (seatNo, airlineName, price, class, source, destination, departureTime, arrivalTime)

TICKETS_IS_ASSIGNED_LUGGAGE (seatNo, airlineName, luggageID, weight, price, class, source, destination, departureTime, arrivalTime)

3. Logical Design

In this chapter, we will delve into the logical design of the database system. We have normalized the tables up to Boyce-Codd Normal Form (BCNF). By applying the principles of normalization, we have eliminated redundant data and ensured data integrity. The following tables are already normalized up to BCNF Form, we need to analyse the functional dependencies and ensure that every determinant is a candidate key.

Let's analyze each table and normalize them:

Table: AIRPORT_PLACE Functional dependencies:

- placeid -> location, type

The table is already in BCNF since the determinant (placeid) is the primary key, and there is only one functional dependency.

Table: RESTAURANTS Functional dependencies:

- placeid -> brand

The table is already in BCNF since the determinant (placeid) is the primary key, and there is only one functional dependency.

Table: TERMINAL Functional dependencies:

- placeid -> capacity

The table is already in BCNF since the determinant (placeid) is the primary key, and there is only one functional dependency.

Table: DEPARTMENT Functional dependencies:

- IDNo -> type

The table is already in BCNF since the determinant (IDNo) is the primary key, and there is only one functional dependency.

Table: SECURITY Functional dependencies:

- IDNo -> secName, grade

The table is already in BCNF since the determinant (IDNo) is the primary key, and there is only one functional dependency.

Table: ENGINEER Functional dependencies:

- IDNo -> engName, grade

The table is already in BCNF since the determinant (IDNo) is the primary key, and there is only one functional dependency.

Table: STAFF Functional dependencies:

- IDNo -> staffName, grade

The table is already in BCNF since the determinant (IDNo) is the primary key, and there is only one functional dependency.

Table: AIRLINE_OPERATES Functional dependencies:

- airlineid -> airlinename, capacity

The table is already in BCNF since the determinant (airlineid) is the primary key, and there is only one functional dependency.

Table: TICKETS Functional dependencies:

- seatNo -> airlinename, price, class, takeoff, destination, departureTime, arrivalTime

The table is already in BCNF since the determinant (seatNo) is the primary key, and there is only one functional dependency.

Table: TICKETS_LUGGAGE Functional dependencies:

- seatNo -> airlinename, luggageID, price, class, takeoff, destination, departureTime, arrivalTime

The table is already in BCNF since the determinant (seatNo) is the primary key, and there is only one functional dependency.

Table: LUGGAGE_DETAILS Functional dependencies:

- airlinename, luggageID -> weight

The table is already in BCNF since the determinant (airlinename, luggageID) is the primary key, and there is only one functional dependency.

All the given tables are already in BCNF, so no further normalization is required. Each table has a single determinant (primary key) and only one functional dependency, ensuring that they meet the requirements of BCNF.

3.1 Normalized Tables

AIRPORT_PLACE (placeID, location, type)

RESAURANTS (placeID, brand)

TERMINAL (placeID, capacity)

DEPARTMENT (IDNo, type)

SECURITY (IDNo, secName, grade)

ENGINEER (IDNo, engName, grade)

STAFF (IDNo, staffName, grade)

AIRLINE_OPERATES (airlineID, airlineName, capacity)

TICKETS (seatNo, airlineName, price, class, source, destination, departureTime, arrivalTime)

TICKETS_LUGGAGE (seatNo, airlineName, luggageID, price, class, source, destination, departureTime, arrivalTime)

LUGGAGE_DETAILS (airlineName, luggageID, weight)

4. Implementation Design

4.1 Create Tables

We have used Django to create Tables for our project. Following are the Statements used to create tables for our project:

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

class AirportPlace(models.Model):
    placeID = models.AutoField(primary_key=True)
    type = models.CharField(max_length=50)

    class Meta:
        indexes = [models.Index(fields=['placeID'])]

class Restaurants(models.Model):
    placeID = models.ForeignKey("AirportPlace", on_delete=models.CASCADE)
```

```
    brand = models.CharField(max_length=50)
    location = models.CharField(max_length=50)


class Terminal(models.Model):
    placeID = models.ForeignKey("AirportPlace", on_delete=models.CASCADE)
    capacity = models.CharField(max_length=50)
    location = models.CharField(max_length=50)
class Department(models.Model):
    IDno = models.AutoField(primary_key=True)
    type = models.CharField(max_length=50)
    placeID = models.ForeignKey("AirportPlace", on_delete=models.CASCADE)

    class Meta:
        indexes = [models.Index(fields=['IDno'])]

class Security(models.Model):
    IDno = models.ForeignKey("Department", on_delete=models.CASCADE)
    secName = models.CharField(max_length=50)
    grade = models.CharField(max_length=50)


class Staff(models.Model):
    IDno = models.ForeignKey("Department", on_delete=models.CASCADE)
    staffName = models.CharField(max_length=50)
    grade = models.CharField(max_length=50)

class Engineer(models.Model):
    IDno = models.ForeignKey("Department", on_delete=models.CASCADE)
    engName = models.CharField(max_length=50)
    grade = models.CharField(max_length=50)

class AirlineOperates(models.Model):
    airlineID = models.AutoField(primary_key=True)
```

```
    airlineName = models.CharField(max_length=50)
    placeID = models.ForeignKey("Terminal", on_delete=models.CASCADE)

    class Meta:
        indexes = [models.Index(fields=['airlineID'])]

class Tickets(models.Model):
    ticketNO = models.IntegerField(primary_key=True)
    seatNO = models.CharField(max_length=50)
    airlineID = models.ForeignKey("AirlineOperates", on_delete=models.CASCADE)
    price = models.PositiveBigIntegerField()
    airlineClass = models.CharField(max_length=50)
    source = models.CharField(max_length=100)
    destination = models.CharField(max_length=100)
    departureTime = models.TimeField()
    arrivalTime = models.TimeField()

class Tickets_Luggage(models.Model):
    ticketNO = models.ForeignKey("Tickets", on_delete=models.CASCADE,
related_name='ticketnumber')
    seatNO = models.ForeignKey("Tickets", on_delete=models.CASCADE)
    luggageID = models.IntegerField(primary_key=True)
    weight = models.IntegerField()

class myTickets(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True, blank=True)
    id = models.IntegerField(primary_key=True),
    ticketid = models.ForeignKey("Tickets", on_delete=models.CASCADE)
    username = models.CharField(max_length=50, default="")
```

4.2 Create Views

We have also used Django to create Views for our project. Following are the Statements used to create views for our project:

```
from django.shortcuts import render, redirect
from django.http import HttpResponse,HttpResponseRedirect
from .models import Tickets
from .models import myTickets
from django.contrib import messages
from django.contrib.auth.models import User
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required

from django.db import connection

def homePage(request):
    cursor=connection.cursor()
    cursor.execute('SELECT arrivalTime, airlineName, airlineClass, source, destination,departureTime,arrivalTime FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE source <> "Karachi"')
    results_arrival=cursor.fetchall()
    cursor.execute('SELECT arrivalTime, airlineName, airlineClass, source, destination,departureTime,arrivalTime FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE destination <> "Karachi"')
    results_departure=cursor.fetchall()
    return render(request,"index.html",{'arrival_results':results_arrival,'departure_results':results_departure})
```

```
def full_schedule(request):
    cursor=connection.cursor()
    cursor.execute('SELECT arrivalTime, airlineName, airlineClass, source, destination FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE source <> "Karachi"')
    results_arrival=cursor.fetchall()
    cursor.execute('SELECT arrivalTime, airlineName, airlineClass, source, destination FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE destination <> "Karachi"')
    results_departure=cursor.fetchall()
    return render(request, 'flightSchedule.html',{'arrival_results':results_arrival,'departure_results':results_departure})

def view_tickets(request):
    cursor=connection.cursor()
    cursor.execute('SELECT ticketNO,airlineName,destination,departureTime,airlineClass,price FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE destination <> "Karachi"')
    results_tickets=cursor.fetchall()
    cursor.execute('SELECT DISTINCT destination FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE destination <> "Karachi"')
    distict_tickets=cursor.fetchall()
    return render(request,"tickets.html",{'ticket_results':results_tickets,'tickets_distinct':distict_tickets})

@login_required
def add_my_ticket(request):
    active_user1 = request.session['username']
    tick_id = request.POST.get('tick_id')
    ticket_name = Tickets.objects.get(ticketNO = tick_id)
    myTickets(tick_id, ticketid = ticket_name, username=active_user1).save()
    messages.info(request, 'Ticket Successfully booked!!')
```

```
    cursor=connection.cursor()
    slt = "SELECT ticketNO,airlineName,destination,departureTime,airlineClass,price FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE ticketNO <> %(ticketno)s AND destination <> 'karachi'"
    cursor.execute(slt,{'ticketno':tick_id})
    results_tickets=cursor.fetchall()
    cursor.execute('SELECT DISTINCT destination FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE destination <> "Karachi"')
    distict_tickets=cursor.fetchall()
    return render(request,"tickets.html",{'ticket_results':results_tickets,'tickets_distinct':distict_tickets})

@login_required
def show_tickets(request):
    active_user1 = request.session['username']
    cursor=connection.cursor()
    select_stmt = "SELECT airlineName,destination,departureTime,airlineClass,price,username FROM airport_airlineoperates INNER JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id INNER JOIN airport_mytickets ON airport_tickets.ticketNO = airport_mytickets.id WHERE username = %(active_user)s"
    cursor.execute(select_stmt,{'active_user':active_user1})
    ticket=cursor.fetchall()
    return render(request, "mytickets.html", {'all':ticket})


def login_user(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')

        if not User.objects.filter(username = username).exists():
            messages.info(request, 'User does not exist')
```

```
            return redirect('login')

        user = authenticate(username=username, password=password)

        if user is None:
            messages.info(request, 'Invalid email or password')
            return redirect('login')
        else:
            login(request, user)
            request.session['username'] = username
            return redirect('homepage')
    return render(request, 'auth.html')


def logout_user(request):
    logout(request)
    return redirect('homepage')


def register_user(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        email = request.POST.get('email')
        password = request.POST.get('password')
        user = User.objects.filter(username = username)

        if user.exists():
            messages.info(request, 'Username already taken')
            return redirect('register')

        user = User.objects.filter(email = email)
```

```
        if user.exists():
            messages.info(request, 'Email already taken')
            return redirect('register')

        user = User.objects.create(
            username = username,
            first_name = first_name,
            last_name = last_name,
            email = email)
        user.set_password(password)
        user.save()
        messages.info(request, 'Account created successfully')
        return redirect('login')
    return render(request, 'register.html')

@login_required
def editprofile(request):
    active_user = request.session['username']
    if request.method == 'POST':
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        email = request.POST.get('email')
        password = request.POST.get('password')

        user = User.objects.filter(email = email)

        if user.exists():
            messages.info(request, 'Email already taken')
            return redirect('editprofile')

        updated_user = User.objects.get(username = active_user)
        updated_user.first_name = first_name
        updated_user.last_name = last_name
        updated_user.email = email
```

```
        updated_user.set_password(password)
        updated_user.save()
        messages.success(request, 'Credentials Successfully Updated!')

    return render(request,'editprofile.html')

def search(request):
    cursor=connection.cursor()
    answer = request.POST.get('dropdown')
    if answer == 'all':
        return view_tickets(request)
    query = "SELECT ticketNO,airlineName,destination,departureTime,airlineClass,price,username FROM airport_airlineoperates INNER JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id INNER JOIN airport_mytickets ON airport_tickets.ticketNO = airport_mytickets.id WHERE destination = %(desti)s"

    cursor.execute(query,{'desti':answer})
    results_tickets=cursor.fetchall()

    cursor.execute('SELECT DISTINCT destination FROM airport_airlineoperates JOIN airport_tickets ON airport_airlineoperates.airlineID = airport_tickets.airlineID_id WHERE destination <> "Karachi"')
    distict_tickets=cursor.fetchall()

    if request.method == 'GET':
        return redirect('tickets.html')
    else:
        return render(request,"tickets.html",{'ticket_results':results_tickets,'tickets_distinct':distict_tickets})

def restaurants(request):
    cursor=connection.cursor()
    cursor.execute('SELECT * FROM airport_restaurants')
```

```
    results_restaurants=cursor.fetchall()
    return render(request,'restuarants.html',{"restaurants_results":results_restaurants})

def aboutUS(request):
    return render(request,'aboutUs.html')

def stores(request):
    return render(request, 'stores.html')
```

5. Chapter 5 and onwards

In the implementation of our project, we have utilized various tools to develop both the front-end and back-end components of the application. This chapter focuses on describing the tools and technologies employed in each area, highlighting their roles and contributions to the overall system.

5.1 Front-end Implementation

For the front-end development, we adopted a combination of HTML, CSS, and Bootstrap. HTML (Hypertext Markup Language) served as the foundation for structuring the content and layout of our web pages. We utilized its markup tags to define the structure and semantic elements of the user interface.

CSS (Cascading Style Sheets) was employed to handle the presentation and styling aspects of our web pages. It allowed us to customize the visual appearance, colors, fonts, and overall aesthetics of the application. By separating the design from the structure, CSS enhanced the maintainability and flexibility of our front-end codebase.

To expedite the development process and ensure responsiveness across different devices, we leveraged the capabilities of Bootstrap, a popular front-end framework. Bootstrap provided a range of pre-designed components and responsive grid systems that facilitated the creation of a visually appealing and user-friendly interface.

Furthermore, we incorporated JavaScript to introduce interactivity and dynamic functionalities to specific pages or components. JavaScript allowed us to implement client-side validation, handle user events, and perform asynchronous operations, enhancing the overall user experience.

5.2 Back-end Implementation

For the back-end development, we chose to work with the Python Django module, a robust and scalable web framework. Django offered a high-level, model-view-controller (MVC) architecture, which streamlined the development process and facilitated the management of data and application logic.

Using Django's ORM (Object-Relational Mapping), we were able to define and interact with our database models using Python classes. This abstraction layer eliminated the need for writing complex SQL queries manually, simplifying the integration and manipulation of data within our application.

To enhance the administrative interface and provide an intuitive user experience for managing the database, we utilized the Jazzmin built-in model in Django admin. Jazzmin offered a sleek and customizable UI, empowering our administrators to perform CRUD (Create, Read, Update, Delete) operations efficiently and securely.

5.3 Database Hosting

To host our project's database, we utilized db4free, a free database hosting service. This service provided us with shared access among all our group members, allowing us to collaborate seamlessly on the development and testing of our application's data layer.

5.4 Description of forms and reports

Upon launching the virtual environment and running the manage.py server command, our website directs users to the main page at http://127.0.0.1:8000/. This page displays partial records of departure and arrival, which can be viewed interchangeably. The navigation bar provides access to all departure and arrival records.
Additionally, we have a service section that showcases information about the restaurants and stores located at the airport. Users can explore these details to plan their dining and shopping experiences.

The tickets section displays existing tickets available through the airport management system. Users who are logged in can book tickets by clicking the "book" button next to each ticket record. Once a user signs up through the signup page form, they gain access to ticket booking functionality.
For each individual user, there is a "my ticket" section where they can view their booked tickets. This personalized section allows users to keep track of their travel arrangements conveniently.

Furthermore, our website includes an "about us" section that provides relevant information about the developers and their contributions to the project.

Administrators, upon signing up, have access to perform CRUD (Create, Read, Update, Delete) operations, giving them control over managing the system.

5.5 Reports Generated

Below are some of the snapshots of The Airport Management System:

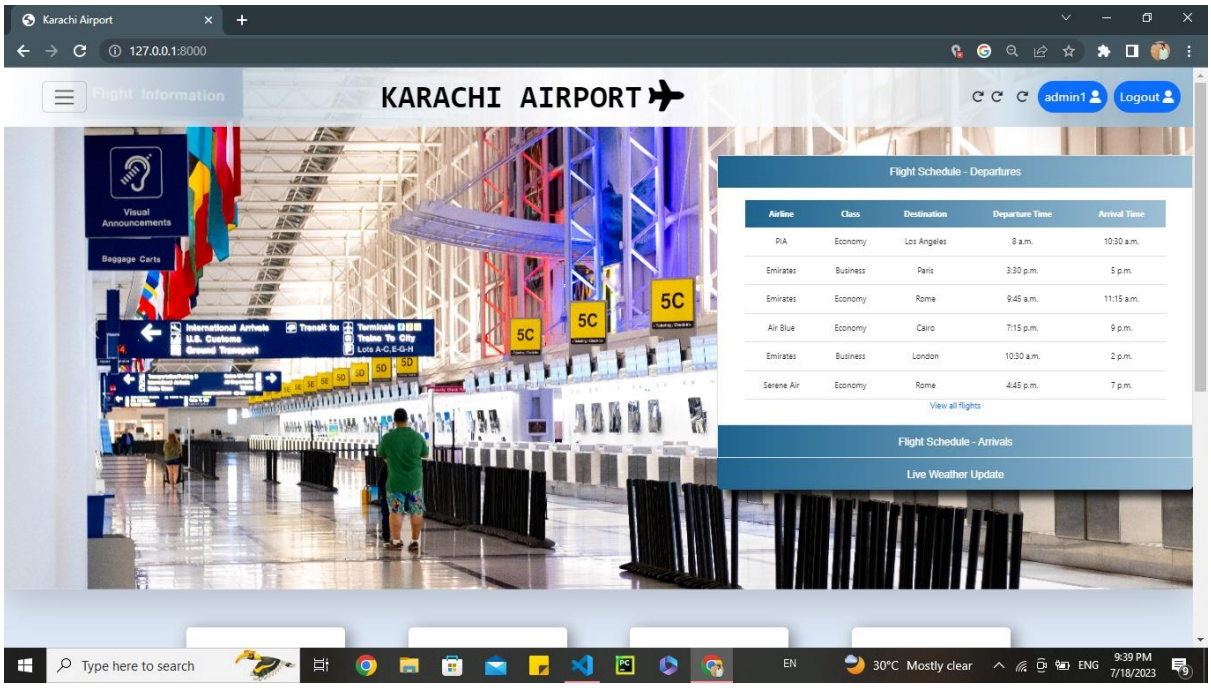


Figure 1 shows home page of our website

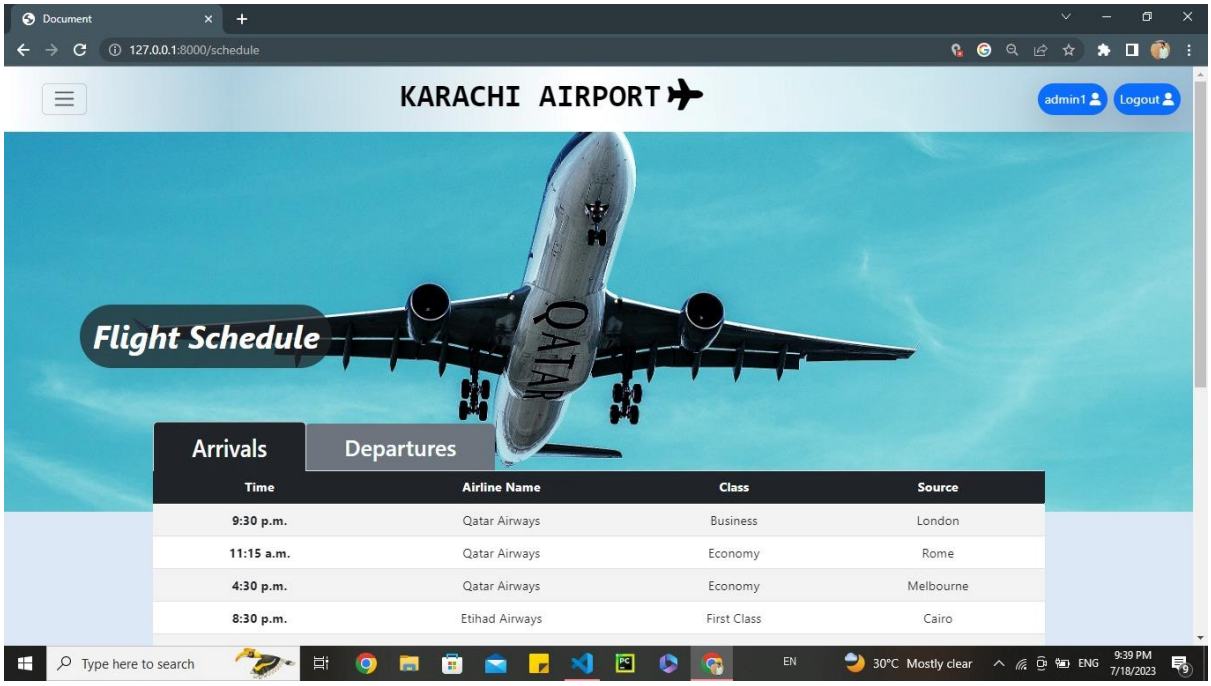


Figure 2 shows the flight schedule page

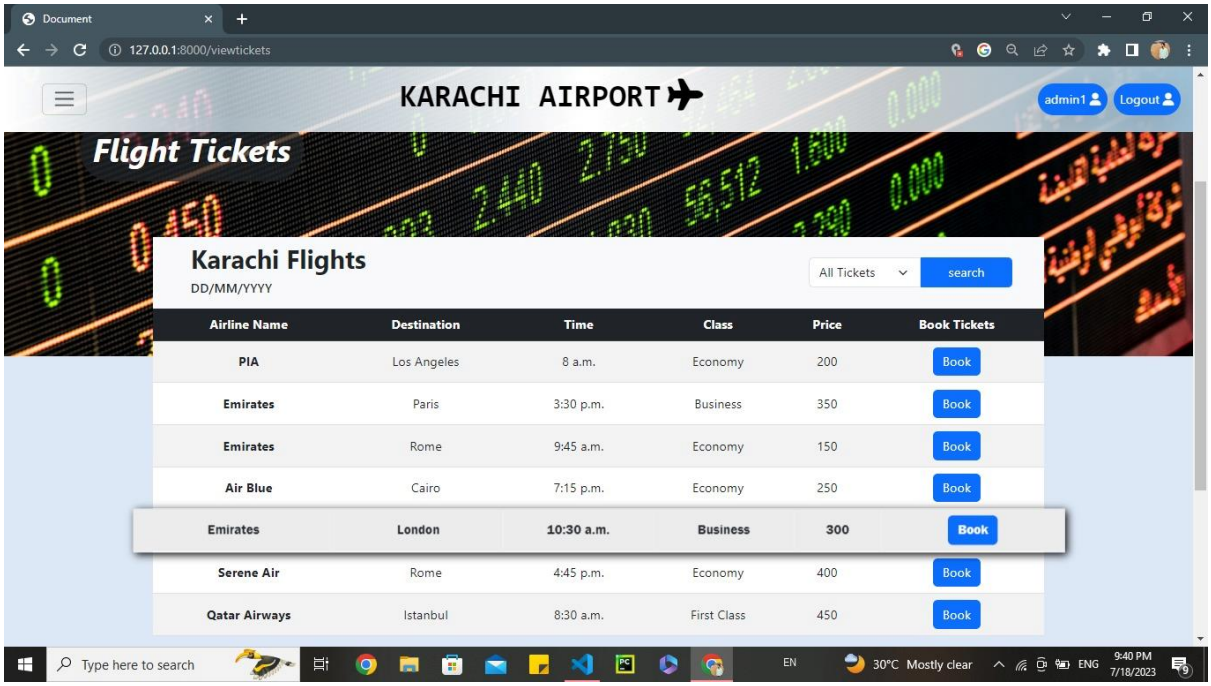


Figure 3 shows the ticket booking page

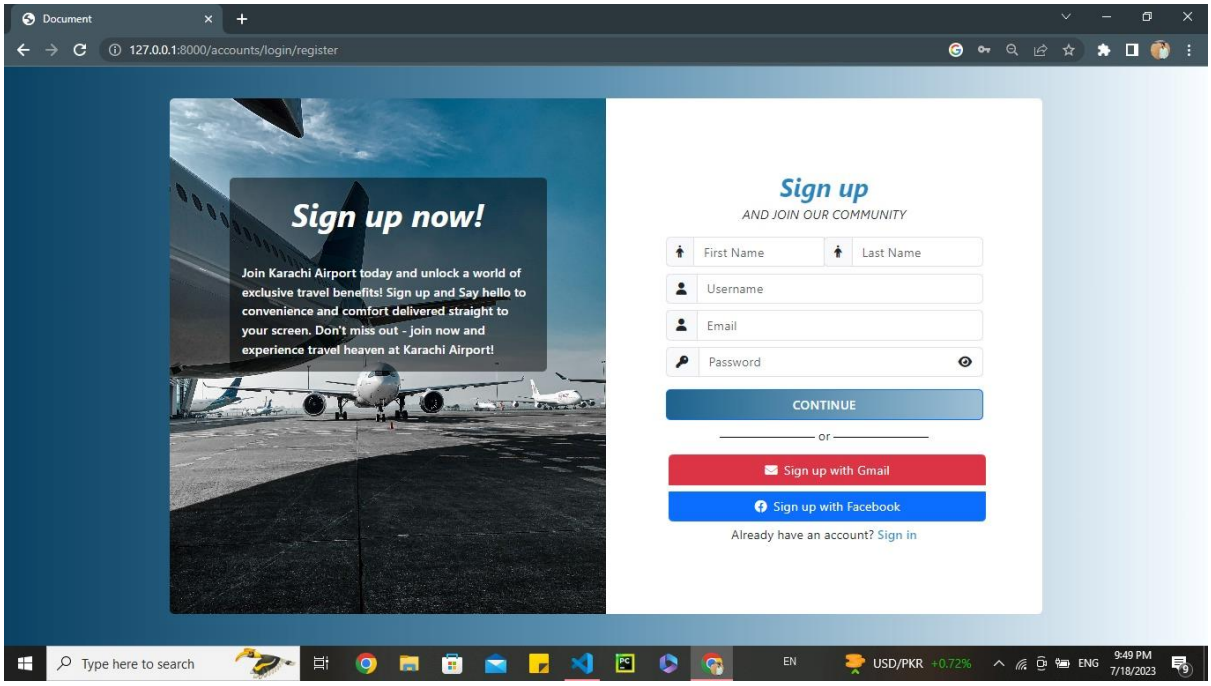


Figure 4 shows the Sign-up page

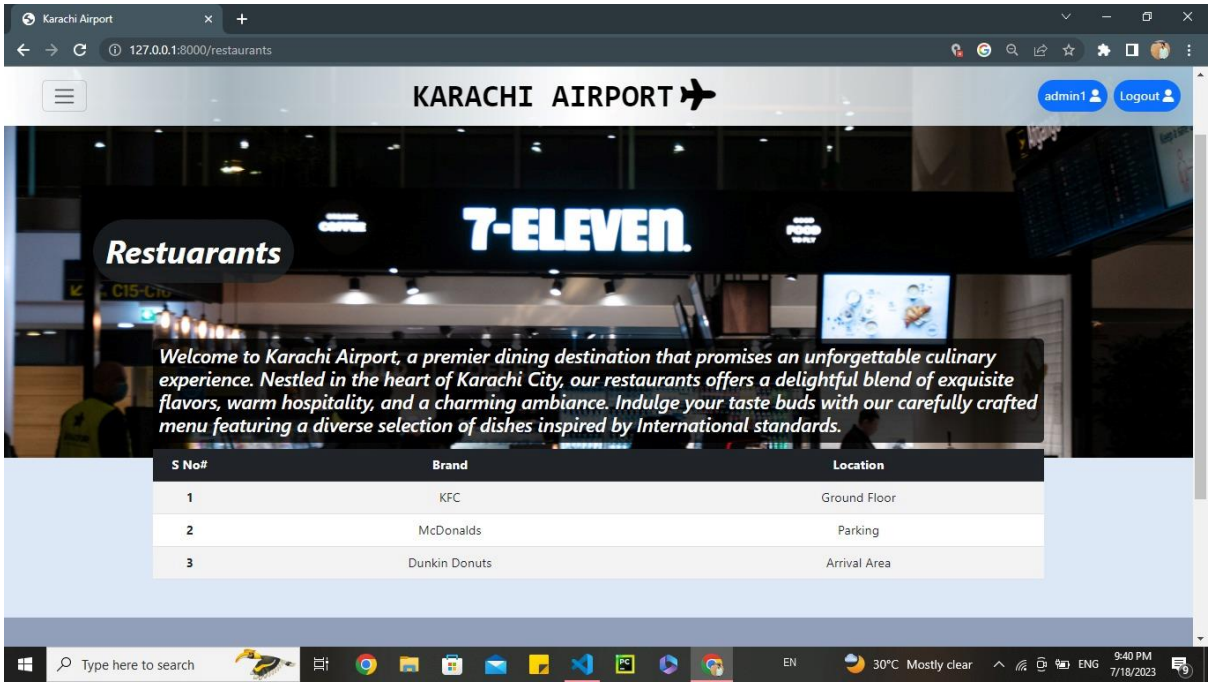


Figure 5 shows the restaurants page

6. Last Chapter

6.1 Implementation of Security

We have added passwords to maintain the security of the database. The admin and the user login pages are different from each other. Moreover, a user can only see their own details and access page as each user is assigned a unique password to protect their privacy.

6.2 Suggested Improvements

The following things can be improved for the future expansions of the project:

1. **Integration of Airline Booking Systems:** Streamlining the booking process by integrating with various airline systems to offer a wider range of flight options and enhance the user experience.

2. **Addition of Billing Options:** Implementing a secure and seamless billing system that supports various payment methods, ensuring a convenient and efficient transaction process for users.

3. **Integration of Location Services:** Incorporating location-based services to provide real-time updates on flight status, gate changes, baggage claim information, and other relevant details to enhance the overall travel experience.

4. **Inclusion of Customer Service Features:** Developing a comprehensive customer service module that allows users to seek assistance, make inquiries, and receive prompt support through various channels, such as live chat, email, or phone.

7. Bibliography

- GeeksforGeeks. (n.d.). GeeksforGeeks - A computer science portal for geeks. Retrieved from https://www.geeksforgeeks.org/
- Stack Overflow. (n.d.). Stack Overflow - Where Developers Learn, Share, and Build Careers. Retrieved from https://stackoverflow.com/
- Django. (n.d.). Django documentation. Retrieved from https://docs.djangoproject.com/en/4.2/
- Bootstrap. (n.d.). Bootstrap - The world's most popular front-end open source toolkit. Retrieved from https://getbootstrap.com/
- OpenAI. (n.d.). ChatGPT: Large-scale language models. Retrieved from https://openai.com/blog/chatgpt
- Bing. (n.d.). Microsoft Bing. Retrieved from https://www.bing.com/?/ai

Contribution of Group members

Syed Shahzain Hassan ~ Back-end coding using Python module Django.

Faizan Soomro ~ Back-end coding using Python module Django.

Sameed Fayiz ~ Front-end coding using HTML, CSS and JavaScript, UI/UX designing.

Aaliyan Mansoor ~ ER Model Diagram, Normalization of Tables, Populating and handling Database.

C E R T I F I C A T E

Certified that following students have successfully completed the DBMS project titled <<***PROJECTTITLE***>> assigned as a mandatory requirement for qualifying the practical examination of the course.

Aaliyan Mansoor	CS-21146	_____<signature>
Syed Shahzain Hassan	CS-21128	_____<signature>
Sameed Fayiz	CS-21127	_____<signature>
Faizan Soomro	CS-21090	_____<signature>

_____

Name & Signature
Project Examiner

Course Code: CS 222, Course Title: Database Management Systems
Evaluation Rubric
CLO: CLO-4, Taxonomy Level: C3

Group Members:

Student No.	Name	Roll No.
S1		
S2		
S3		
S4		

CRITERIA AND SCALES				Marks Obtained			
Criterion 1: Project objective & background (CPA-1)							
0.5	1	1.5	2	S1	S2	S3	S4
Many major objectives, constraints and criteria are not identified	Some of the objectives, constraints and criteria are identified	All major objectives are identified.	The objectives are well defined.				
Criterion 2: Methodology followed (CPA-1, CPA-3)							
0.5	1	1.5	2				
Information is collected but without any analysis	Most constraints are identified; some are not properly analyzed	Sufficient information is obtained. Design is reasonable and mostly supported by the information	All relevant information is obtained and accurately analyzed				
Criterion 3: Clarity and presentation of the report							
0.5	1	2	3				
Report lacks an overall organization. Many spelling & grammatical blunders are present.	Report is organized via topic/flow, but in some areas, it is difficult to follow the flow of ideas due to grammatical blunders.	Report is organized and clearly written for the most part. Few spelling & grammatical errors are present.	Report is well organized and clearly written without spelling & grammatical mistakes.				
Criterion 4: Completion of the task							
0.5	1	1.5	2				
Progress is not satisfactory w.r.t plan. No discussions on achieved milestones	Progress is mostly satisfactory w.r.t plan. Some discussions on achieved milestones	Progress is highly satisfactory w.r.t plan. Detailed discussions on achieved milestones	Progress is beyond expectations w.r.t plan. Highly detailed discussions on achieved milestones				
Criterion 5: Use of Database administration and development skills (CPA-2, CPA-3)							
0.5	1	1.5	2				
Does not make use of analytical tools and/or software engineering techniques relevant to project	Employed some analytical tools and/or software engineering techniques acquired	Employ appropriate analytical tools and/or software engineering techniques	Employ excellent analytical tools and/or software engineering techniques				
Criterion 6: Project demonstration							
0.5	1	1.5	2				
The product is incomplete or does not work. The demonstration failed to capture what was communicated.	Present a working product but some desired functions are malfunctioned. The demonstration only conveys main ideas.	Present a working project with support to all desired functions. The demonstration techniques are reasonable.	Present a fully functioning working project. The demonstration techniques are very effective.				

Criterion 7: Answer to questions							
0.5	1	1.5	2				
Student has no or very less knowledge of both problem and solution	Student is uncomfortable with information. Seems novice and can answer basic questions only	Student has competent knowledge and is at ease with information	Student has presented full knowledge of both problem and solution.				