Big Data Storage
Assignment-1
Topic- Ticket Booking System

Submitted by:

Dhuruv Kumar

SAP ID: 500107769

Roll. No: R2142221199

Semester III

B. Tech CSE Big Data B2 (Non-Hons)

Submitted To:

Mr Deepak Kumar Sharma

DATA SCIENCE CLUSTER

UNIVERSTIY OF PETROLEUM & ENERGY STUDIES

DEHRADUN

Domain: Ticket booking System

Objective:

The objective of the code is to create a ticket booking system with MongoDB integration. It allows users to register, login, and book tickets for buses, flights, and trains. The system supports functionalities such as viewing routes, updating timing and pricing, cancelling bookings, and accessing booking statistics. MongoDB is utilized for efficient storage and retrieval of user data, route information, and booking details, enhancing the system's scalability and performance.

Introduction:

The ticket booking system is designed to facilitate users in booking tickets for various modes of transportation such as buses, flights, and trains. MongoDB is utilized as the database management system to store and manage data related to routes, bookings, and user information.

Flow of the Entire Code:

1. **Main Menu:**
    - The program starts by displaying the main menu options: Register, Login, About Us, and Exit.
    - Users can register with the system, log in using their credentials, view information about the company, or exit the program.
2. **Register:**
    - Users can register by providing their details such as username, password, name, and contact information.
    - The registration data is stored in the MongoDB database for future authentication.
3. **Login:**
    - Registered users can log in with their username and password.
    - Upon successful login, users are directed to the ticket menu.
4. **Ticket Menu:**
    - The ticket menu offers options to book tickets, view booked tickets, cancel bookings, or logout.
    - Users can perform various ticket-related actions based on their requirements.
5. **Book Ticket:**
    - Users can book tickets by specifying the mode of transportation, source, destination, date, time, and passenger details.
    - Booked ticket information is stored in the MongoDB database.
6. **View Tickets:**
    - Users can view their booked tickets along with details such as source, destination, date, time, price, and passenger information.

- Ticket information is retrieved from the MongoDB database and displayed to the user.

7. **Cancel Booking:**
    - Users can cancel their booked tickets by selecting the mode of transportation and specifying the booking to cancel.
    - The cancelled booking is removed from the MongoDB database.
8. **About Us:**
    - Users can view information about the company, including its name, address, and contact details.
9. **Exit:**
    - Users can choose to exit the ticket booking system, terminating the program.
10. **Admin Menu:**
    - Administrators have access to additional functionalities via the admin menu.
    - Admin options include adding new routes, updating route timing and pricing, deleting existing routes, viewing all bookings, viewing route statistics, and viewing details for specific vehicles.
11. **MongoDB Functionality:**
    - **Storing User Data:** User registration data, including usernames, passwords, and contact information, is stored in MongoDB.
    - **Storing Route Information:** Details of routes for buses, flights, and trains are stored in MongoDB collections, including source, destination, time, price, and vehicle information.
    - **Storing Booked Tickets:** Information about booked tickets, including mode of transportation, source, destination, date, time, price, and passenger details, is stored in MongoDB.
    - **Querying and Updating Data:** MongoDB queries are used to retrieve route information, user details, and booked tickets. Data is also updated, such as when cancelling bookings or updating route timing and pricing.
    - **Aggregation:** Aggregation operations are utilized to compute statistics such as the total number of buses, flights, and trains, as well as the number of vehicles available per route.

Complete code:

```
import pymongo
import sys
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["ticket_booking_system"]
users_collection = db["users"]
bus_collection = db["Bus"]
flight_collection = db["Flight"]
train_collection = db["Train"]
booked_tickets_collection = db["booked_tickets"]

# Function to register a new user
def register():
    name = input("Enter name: ")  # Input user's name
    age = input("Enter age: ")  # Input user's age
    email = input("Enter email: ")  # Input user's email
    username = input("Create username: ")  # Create username
    while True:
        password = input("Enter password: ")  # Input password
        password1 = input("Re-enter password: ")  # Re-enter password for confirmation
        if password != password1:
            print("Password does not match")  # Print message if passwords don't match
        else:
            break  # Break the loop if passwords match
    user = {"username": username, "password": password, "name": name, "age": age}  #
Create user object
    users_collection.insert_one(user)  # Insert user into the collection
    print("Registration successful!")  # Print success message


# Function to login a user
def login():
    username = input("Enter username: ")  # Input username
    password = input("Enter password: ")  # Input password
    user = users_collection.find_one({"username": username, "password": password})  #
Find user in the collection

    if user:
        if username == "Dhruv" and password == "Dhruv@30":  # Check if admin login
            print("Admin login successful!")  # Print admin login success message
            admin_menu()  # Call admin menu function
            return "admin"  # Return "admin" to identify admin login
        else:
            print("Login successful!")  # Print login success message
            print("hii!", username, "welcome to UPES ticket booking system")  # Print
welcome message
            return username  # Return username
    else:
        print("Invalid username or password.")  # Print message for invalid credentials
```

```
        return None  # Return None if login fails


# Function to book a ticket
def book_ticket(username):
    while True:
        if username:
            print("\nTransportation Modes:")
            print("1. Bus")
            print("2. Train")
            print("3. Flight")
            print("4. Back")
            mode = input("Choose mode of transportation (1/2/3): ")  # Choose mode of
transportation
            if mode == "1":
                book_bus_ticket(username)  # Call function to book a bus ticket
            elif mode == "2":
                book_train_ticket(username)  # Call function to book a train ticket
            elif mode == "3":
                book_flight_ticket(username)  # Call function to book a flight ticket
            elif mode == "4":
                ticket_menu(username)  # Go back to ticket menu
            else:
                print("Invalid choice.")  # Print message for invalid choice

def book_bus_ticket(username):
    mode = "bus"  # Set the mode of transportation to "bus"
    while True:
        source = input("Enter source station: ")  # Input source station
        destination = input("Enter destination station: ")  # Input destination station

        while True:
            date = input("Enter date (YYYY-MM-DD): ")
            date_parts = date.split("-")
            if len(date_parts) == 3:
                year = date_parts[0]
                month = date_parts[1]
                day = date_parts[2]

                if int(year) >= 2024:
                    break
                else:
                    print("Invalid year. Please enter a year greater than or equal to
2024.")

                if 1 <= int(month) <= 12:
                    break
                else:
                    print("Invalid month. Please enter a month between 1 to 12.")
```

```
                if 1 <= int(day) <= 30:
                    break
                else:
                    print("Invalid date. Please enter a date between 1 to 30.")
            else:
                print("Invalid date format. Please enter date in YYYY-MM-DD format.")  #
Print the date

        # Find buses available for the specified source and destination
        buses = bus_collection.find({"source": source, "destination": destination})
        lst = list(buses)

        if lst:
            k = 1
            for bus in lst:
                # Print available buses with their details
                print(f"slot {k} --> Source: {bus['source']}, Destination:
{bus['destination']}, Time: {bus['time']}, Price per passenger: {bus['price']}")
                k += 1

            ch = int(input("Enter Choice: "))  # Get user choice
            ch -=1
            while True:
                if 0 <= ch < len(lst):
                    num_passengers = int(input("Enter the number of passengers: "))  #
Get number of passengers

                    # Calculate total price based on the number of passengers
                    total_price = num_passengers * lst[ch]['price']

                    passengers = allocate_seats(num_passengers, mode, source,
destination, date)  # Allocate seats
                    # Book the selected bus
                    print("Booked:", lst[ch]['source'], lst[ch]['destination'],
lst[ch]['time'], "Rs.",total_price, "Date:", date)
                    print("Passengers:")
                    for passenger in passengers:
                        # Print passenger details
                        print(f"Name: {passenger['name']}, Age: {passenger['age']}, Seat:
{passenger['seat']}")
                    confirm = input("Do you want to book this bus? (1 for yes, 2 for no):
")  # Confirm booking
                    if confirm == "1":
                        booked_bus = lst[ch]
                        book_ticket = {
                            "username": username,
                            "mode": "Bus",
                            "source": source,
                            "destination": destination,
                            "date": date,
```

```
                            "bus_number": booked_bus["bus_number"],
                            "time": booked_bus["time"],
                            "price": total_price,  # Update price with the total price
                            "passengers": passengers
                        }
                        booked_tickets_collection.insert_one(book_ticket)  # Insert
booking into collection
                        print("Bus ticket booked successfully!")  # Print success message
                    elif confirm == "2":
                        print("Booking canceled.")  # Print cancellation message
                    else:
                        print("Invalid choice.")  # Print message for invalid choice
                    break  # Exit the loop if the booking is successful

                else:
                    print("Invalid choice. Please enter a valid slot number.")  # Print
message for invalid choice
            break
        else:
            print("No bus available for this route. Please enter valid source and
destination.")  # Print message if no buses available


# Function to book a flight ticket
def book_flight_ticket(username):
    mode = "flight"  # Set the mode of transportation to "flight"
    while True:
        source = input("Enter source station: ")  # Input source station
        destination = input("Enter destination station: ")  # Input destination station

        while True:
            date = input("Enter date (YYYY-MM-DD): ")
            date_parts = date.split("-")
            if len(date_parts) == 3:
                year = date_parts[0]
                month = date_parts[1]
                day = date_parts[2]

                if int(year) >= 2024:
                    break
                else:
                    print("Invalid year. Please enter a year greater than or equal to
2024.")

                if 1 <= int(month) <= 12:
                    break
                else:
                    print("Invalid month. Please enter a month between 1 to 12.")

                if 1 <= int(day) <= 30:
```

```
                    break
                else:
                    print("Invalid date. Please enter a date between 1 to 30.")
            else:
                print("Invalid date format. Please enter date in YYYY-MM-DD format.")

        # Find flights available for the specified source and destination
        flights = flight_collection.find({"source": source, "destination": destination})
        lst = list(flights)

        if lst:
            k = 1
            for flight in lst:
                # Print available flights with their details
                print(f"slot {k} --> Source: {flight['source']}, Destination:
{flight['destination']}, Time: {flight['time']}, Price: {flight['price']}")
                k += 1

            ch = int(input("Enter Choice: "))  # Get user choice
            ch -=1
            while True:
                if 0 <= ch < len(lst):
                    num_passengers = int(input("Enter the number of passengers: "))  #
Get number of passengers
                    # Calculate total price based on the number of passengers
                    total_price = num_passengers * lst[ch]['price']

                    passengers = allocate_seats(num_passengers, mode, source,
destination, date)  # Allocate seats
                    # Book the selected flight
                    print("Booked:", lst[ch]['source'], lst[ch]['destination'],
lst[ch]['time'], "Rs.",total_price,"Date:",date)
                    print("Passengers:")
                    for passenger in passengers:
                        # Print passenger details
                        print(f"Name: {passenger['name']}, Age: {passenger['age']}, Seat:
{passenger['seat']}")
                    confirm = input("Do you want to book this flight? (1 for yes, 2 for
no): ")  # Confirm booking
                    if confirm == "1":
                        booked_flight = lst[ch]
                        book_ticket = {
                            "username": username,
                            "mode": "Flight",
                            "source": source,
                            "destination": destination,
                            "date": date,
                            "flight_number": booked_flight["flight_number"],
                            "time": booked_flight["time"],
                            "price": total_price,
```

```
                        "passengers": passengers
                    }
                    booked_tickets_collection.insert_one(book_ticket)  # Insert
booking into collection
                    print("Flight ticket booked successfully!")  # Print success
message
                elif confirm == "2":
                    print("Booking canceled.")  # Print cancellation message
                else:
                    print("Invalid choice.")  # Print message for invalid choice
                break  # Exit the loop if the booking is successful
            else:
                print("Invalid choice. Please enter a valid slot number.")  # Print
message for invalid choice
                break
        break
    else:
        print("No flight available for this route. Please enter valid source and
destination.")  # Print message if no flights available


# Function to book a train ticket
def book_train_ticket(username):
    mode = "train"  # Set the mode of transportation to "train"
    while True:
        source = input("Enter source station: ")  # Input source station
        destination = input("Enter destination station: ")  # Input destination station

        while True:
            date = input("Enter date (YYYY-MM-DD): ")
            date_parts = date.split("-")
            if len(date_parts) == 3:
                year = date_parts[0]
                month = date_parts[1]
                day = date_parts[2]

                if int(year) >= 2024:
                    break
                else:
                    print("Invalid year. Please enter a year greater than or equal to
2024.")

                if 1 <= int(month) <= 12:
                    break
                else:
                    print("Invalid month. Please enter a month between 1 to 12.")

                if 1 <= int(day) <= 30:
                    break
                else:
```

```
                    print("Invalid date. Please enter a date between 1 to 30.")
            else:
                print("Invalid date format. Please enter date in YYYY-MM-DD format.")  #
Print the date

        # Find trains available for the specified source and destination
        trains = train_collection.find({"source": source, "destination": destination})
        lst = list(trains)

        if lst:
            k = 1
            for train in lst:
                # Print available trains with their details
                print(f"slot {k} --> Source: {train['source']}, Destination:
{train['destination']}, Time: {train['time']}, Price: {train['price']}")
                k += 1

            ch = int(input("Enter Choice: "))  # Get user choice
            ch -=1
            while True:
                if 0 <= ch < len(lst):

                    num_passengers = int(input("Enter the number of passengers: "))  #
Get number of passengers
                    # Calculate total price based on the number of passengers
                    total_price = num_passengers * lst[ch]['price']

                    passengers = allocate_seats(num_passengers, mode, source,
destination, date)  # Allocate seats
                    # Book the selected train
                    print("Booked:", lst[ch]['source'], lst[ch]['destination'],
lst[ch]['time'], "Rs.",total_price,"Date:",date)
                    print("Passengers:")
                    for passenger in passengers:
                        # Print passenger details
                        print(f"Name: {passenger['name']}, Age: {passenger['age']}, Seat:
{passenger['seat']}")
                    confirm = input("Do you want to book this train? (1 for yes, 2 for
no): ")  # Confirm booking
                    if confirm == "1":
                        booked_train = lst[ch]
                        book_ticket = {
                            "username": username,
                            "mode": "Train",
                            "source": source,
                            "destination": destination,
                            "date": date,
                            "train_number": booked_train["train_number"],
                            "time": booked_train["time"],
                            "price": total_price,
```

```
                            "passengers": passengers
                        }
                        booked_tickets_collection.insert_one(book_ticket)  # Insert
booking into collection
                        print("Train ticket booked successfully!")  # Print success
message
                    elif confirm == "2":
                        print("Booking canceled.")  # Print cancellation message
                    else:
                        print("Invalid choice.")  # Print message for invalid choice
                    break  # Exit the loop if the booking is successful
                else:
                    print("Invalid choice. Please enter a valid slot number.")  # Print
message for invalid choice
                    break
            break
        else:
            print("No train available for this route. Please enter valid source and
destination.")  # Print message if no trains available

# Function to allocate seats for passengers based on mode, source, destination, and date
def allocate_seats(num_passengers, mode, source, destination, date):
    # Define total available seats based on the mode of transportation
    if mode == "bus":
        total_seats = 50  # Total seats for buses
    elif mode == "train":
        total_seats = 100  # Total seats for trains
    elif mode == "flight":
        total_seats = 150  # Total seats for flights
    else:
        print("Invalid mode of transportation.")
        return None  # Return None if mode is invalid

    # Get already booked seats for the same mode, source, destination, and date
    booked_seats = set()
    booked_tickets = booked_tickets_collection.find({"mode": mode, "source": source,
"destination": destination, "date": date})
    for ticket in booked_tickets:
        for passenger in ticket['passengers']:
            booked_seats.add(passenger['seat'])

    available_seats = set(range(1, total_seats + 1)) - booked_seats  # Calculate
available seats

    # Check if there are enough available seats for the requested number of passengers
    if len(available_seats) < num_passengers:
        print("Sorry, there are not enough available seats for the requested number of
passengers.")
        return None
```

```
    passengers = []
    for i in range(num_passengers):
        passenger_name = input(f"Enter name of passenger {i+1}: ")  # Input passenger's
name
        passenger_age = int(input(f"Enter age of passenger {i+1}: "))  # Input
passenger's age

        # Find the next available seat
        if available_seats:
            seat_number = available_seats.pop()
            passengers.append({"name": passenger_name, "age": passenger_age, "seat":
seat_number})  # Add passenger details
        else:
            print("Sorry, all available seats are booked.")
            return None

    return passengers  # Return list of passengers with allocated seats

# Function for the main menu
def main_menu():
    while True:
        print("\nMain Menu")
        print("1. Register")
        print("2. Login")
        print("3. About Us")
        print("4. Exit")
        choice = input("Enter your choice: ")  # Input choice

        if choice == "1":
            register()  # Call register function
        elif choice == "2":
            username = login()  # Call login function
            if username:
                ticket_menu(username)  # If login successful, go to ticket menu
        elif choice == "3":
            display_about_us()  # Call function to display about us
        elif choice == "4":
            print("Thank you for using the ticket booking system.")
            sys.exit()  # Exit the program
        else:
            print("Invalid choice. Please try again.")  # Print message for invalid
choice

# Function to display information about the company
def display_about_us():
    print("\nAbout Us")
    print("UPES TRAVEL AGENCY")
    print("Address: 216B I,\n Second Floor,Splendor Forum,\n Plot Bearing No. 3,Jasola
District Centre, Jasola,\n New Delhi-110025")
    print("Contact: 18001028737")
```

```python
# Function for the ticket menu
def ticket_menu(username):
    while True:
        print("\nTicket Menu")
        print("1. Book Ticket")
        print("2. View Tickets")
        print("3. Cancel Booking")
        print("4. Logout")
        choice = input("Enter your choice: ")  # Input choice
        print()

        if choice == "1":
            book_ticket(username)  # Call function to book a ticket
        elif choice == "2":
            view_tickets(username)  # Call function to view tickets
        elif choice == "3":
            cancel_booking(username)  # Call function to cancel booking
        elif choice == "4":
            print("Logged out successfully.")
            main_menu()  # Go back to main menu
        else:
            print("Invalid choice. Please try again.")  # Print message for invalid
choice


# Function to cancel a booking for a user
def cancel_booking(username):
    # Input for selecting the mode of transportation
    mode_choice = input("Enter mode of transportation (1 for bus, 2 for flight, 3 for
train): ")

    # Determine the mode based on user input
    if mode_choice == '1':
        mode = 'Bus'
    elif mode_choice == '2':
        mode = 'Flight'
    elif mode_choice == '3':
        mode = 'Train'
    else:
        print("Invalid mode selection.")
        return

    # Retrieve bookings for the user and mode
    bookings = list(booked_tickets_collection.find({"username": username, "mode": mode}))

    # If no bookings found, print a message and return
    if not bookings:
        print(f"No {mode} bookings found for this user.")
        return
```

```
    # Display existing bookings for the user
    print("Existing bookings:")
    for index, booking in enumerate(bookings, 1):
        print(f"{index}. Source: {booking['source']}, Destination:
{booking['destination']}, Date: {booking['date']}, Time: {booking['time']}")

    # Input for selecting the booking to cancel
    cancel_choice = input("Enter the number of booking to cancel: ")
    if cancel_choice.isdigit():
        cancel_index = int(cancel_choice) - 1
        if 0 <= cancel_index < len(bookings):
            booking_to_cancel = bookings[cancel_index]

            # Delete the booking
            booked_tickets_collection.delete_one({"_id": booking_to_cancel["_id"]})
            print("Booking canceled successfully.")
        else:
            print("Invalid booking number.")
    else:
        print("Invalid input. Please enter a number.")

# Function to view tickets booked by a user
def view_tickets(username):
    # Retrieve tickets booked by the user
    user_tickets = booked_tickets_collection.find({"username": username})

    # Display details of each ticket
    for ticket in user_tickets:
        print("Ticket Number:", ticket['_id'])
        print("Source:", ticket['source'])
        print("Destination:", ticket['destination'])
        print("Time:", ticket['time'])
        print("Date:", ticket['date'])
        print("Price:", ticket['price'])
        mode = ticket['mode']
        print(f"{mode.capitalize()} number:", ticket[f"{mode.lower()}_number"])
        print("Passengers:")
        for passenger in ticket['passengers']:
            print(f"Name: {passenger['name']}, Age: {passenger['age']}, Seat number:
{passenger['seat']}")
        print()


# Function to delete a route for a mode of transportation
def delete_route():
    mode = input("Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train):
")
    if mode not in ['1', '2', '3']:
        print("Invalid mode of transportation.")
```

```
        return

    source = input("Enter source: ")
    destination = input("Enter destination: ")

    # Determine the collection and mode string based on the mode of transportation
    if mode == '1':
        collection = bus_collection
        mode_str = 'Bus'
    elif mode == '2':
        collection = flight_collection
        mode_str = 'Flight'
    elif mode == '3':
        collection = train_collection
        mode_str = 'Train'

    # Retrieve routes based on source and destination
    routes = collection.find({"source": source, "destination": destination})
    routes_list = list(routes)

    # If no routes found, print a message and return
    if not routes_list:
        print(f"No {mode_str} routes found for the given source and destination.")
        return

    # Display available routes
    print(f"Available {mode_str} Routes:")
    for i, route in enumerate(routes_list, 1):
        if mode_str == 'Bus':
            print(f"{i}. Source: {source}, Destination: {destination}, Price:
{route['price']}, Bus Number: {route['bus_number']}")
        elif mode_str == 'Flight':
            print(f"{i}. Source: {source}, Destination: {destination}, Price:
{route['price']}, Flight Number: {route['flight_number']}")
        elif mode_str == 'Train':
            print(f"{i}. Source: {source}, Destination: {destination}, Price:
{route['price']}, Train Number: {route['train_number']}")

    # Input for selecting the route to delete
    choice = input("Enter the number of the route to delete: ")

    if choice.isdigit():
        route_index = int(choice) - 1
        if 0 <= route_index < len(routes_list):
            route_to_delete = routes_list[route_index]

            # Delete the route
            collection.delete_one({"_id": route_to_delete["_id"]})
            print("Route deleted successfully.")
        else:
```

```
            print("Invalid route number.")
    else:
        print("Invalid input. Please enter a number.")

#Function to handle admin menu options
def admin_menu():
    while True:
        print("\nAdmin Menu")
        print("1. Add New Route")
        print("2. Update Route Timing")
        print("3. View All Routes")
        print("4. Update Route Pricing")
        print("5. Delete Existing Route")
        print("6. View All Bookings")
        print("7. View Route Statistics")
        print("8. View details for specific vehicle")
        print("9. Logout")
        choice = input("Enter your choice: ")
        print()

        if choice == "1":
            add_new_route()
        elif choice == "2":
            update_route_timing()
        elif choice == "3":
            view_all_routes()
        elif choice == "4":
            update_route_pricing()
        elif choice == "5":
            delete_route()  # Option to delete a route
        elif choice == "6":
            view_all_bookings()
        elif choice == "7":
            view_route_statistics()  # Option to view route statistics
        elif choice == "8":
            show_tickets_and_passengers()
        elif choice == "9":
            print("Logged out successfully.")
            main_menu()
        else:
            print("Invalid choice. Please try again.")


# Function to view statistics for different routes
def view_route_statistics():
    # Count the total number of buses, flights, and trains
    total_buses = bus_collection.count_documents({})
    total_flights = flight_collection.count_documents({})
    total_trains = train_collection.count_documents({})
```

```
    print("Vehicle Statistics:")
    print(f"Total Buses: {total_buses}")
    print(f"Total Flights: {total_flights}")
    print(f"Total Trains: {total_trains}")
    while True:

        # Prompt the user to select a mode (bus, flight, or train)
        mode = input("Select mode (1 for Bus / 2 for Flight / 3 for Train, 0 to exit): ")
        if mode == '0':
            print("Exiting...")
            return
        elif mode not in ['1', '2', '3']:
            print("Invalid mode selection.")
            continue

        # Determine the collection and mode string based on the selected mode
        if mode == '1':
            collection = bus_collection
            mode_str = 'Bus'
        elif mode == '2':
            collection = flight_collection
            mode_str = 'Flight'
        else:
            collection = train_collection
            mode_str = 'Train'

        # Aggregate to count the number of vehicles per route
        routes = collection.aggregate([
            {"$group": {
                "_id": {"source": "$source", "destination": "$destination"},
                "count": {"$sum": 1}
            }},
            {"$sort": {"_id.source": 1, "_id.destination": 1}}
        ])

        # Display the results
        total_routes = 0
        print(f"\nNumber of {mode_str}s Available per Route:")
        for route in routes:
            total_routes += 1
            print(f"\nSource: {route['_id']['source']}, Destination:
{route['_id']['destination']}, Number of {mode_str}s: {route['count']}")

        print(f"\nTotal Routes Available: {total_routes}")
        print()


# Function to display all bookings and a summary of total tickets and passengers for each
mode of transportation
def view_all_bookings():
```

```
    # Pipeline to aggregate total tickets and passengers for each mode
    pipeline = [
        {
            "$group": {
                "_id": "$mode",
                "total_tickets": {"$sum": 1},
                "total_passengers": {"$sum": {"$size": "$passengers"}}  # Count total
passengers
            }
        }
    ]

    # Aggregate the bookings summary
    bookings_summary = booked_tickets_collection.aggregate(pipeline)

    print("Booking Summary:")
    for summary in bookings_summary:
        mode = summary["_id"]
        total_tickets = summary["total_tickets"]
        total_passengers = summary["total_passengers"]
        print(f"Mode: {mode}, Total Tickets booked: {total_tickets}, Total Passengers:
{total_passengers}")

    # Retrieve all bookings
    all_bookings = booked_tickets_collection.find({})
    all_bookings_list = list(all_bookings)  # Convert cursor to list to count
    if len(all_bookings_list) == 0:
        print("No bookings found.")
        return

    print("\nAll Bookings:")
    index = 1
    for booking in all_bookings_list:
        print(f"{index}. Username: {booking['username']}, Mode: {booking['mode']},
Source: {booking['source']}, Destination: {booking['destination']}, Date:
{booking['date']}, Time: Date: {booking['time']}")
        mode = booking['mode']
        if mode == 'Bus':
            print(f"   Bus Number: {booking['bus_number']}")
        elif mode == 'Train':
            print(f"   Train Number: {booking['train_number']}")
        elif mode == 'Flight':
            print(f"   Train Number: {booking['flight_number']}")

        passengers = booking['passengers']
        passenger_count = len(passengers)
        ticket_count = 1 if passenger_count > 0 else 0
        print(f"   Passenger count: {passenger_count}")

        if passengers:
```

```
            print("   Passengers:")
            for passenger in passengers:
                print(f"      Name: {passenger['name']}, Age: {passenger['age']}, Seat:
{passenger['seat']}")
        else:
            print("   No passengers.")
        print()
        index += 1

# Function to display tickets and passengers details for a specific mode of
transportation
from bson import ObjectId

def show_tickets_and_passengers():
    modes = {"1": "bus", "2": "flight", "3": "train"}

    while True:
        # Ask user for mode of transportation (bus/flight/train)
        mode_input = input("Enter mode of transportation (1: bus, 2: flight, 3: train, 0:
exit): ").strip()
        if mode_input == "0":
            print("Exiting.")
            break
        elif mode_input not in modes:
            print("Invalid mode. Please enter 1, 2, 3, or 0.")
            continue

        mode = modes[mode_input]

        # Aggregate to get distinct numbers and total tickets for the selected mode
        pipeline = [
            {"$match": {f"{mode}_number": {"$exists": True}}},
            {"$group": {"_id": f"${mode}_number", "total_tickets": {"$sum": 1}}}
        ]

        distinct_numbers = booked_tickets_collection.aggregate(pipeline)

        # Display all mode-specific numbers with their respective number of booked
tickets
        print(f"\n{mode.capitalize()} Numbers with Booked Tickets:")

        for index, number_info in enumerate(distinct_numbers, 1):
            number = number_info["_id"]
            total_tickets = number_info["total_tickets"]
            print(f"Slot => {index}. {mode.capitalize()} Number: {number}, Total Tickets
Booked: {total_tickets}")

        # Ask user to select a specific number or exit
        while True:
```

```
            choice = input(f"\nEnter a Slot number to show more details (enter '0' to
exit): ")
            if choice == "0":
                print("Exiting.")
                return
            elif choice.isdigit() and 1 <= int(choice) <= index:
                specific_number = distinct_numbers[int(choice) - 1]["_id"]
                break
            else:
                print("Invalid input. Please enter a valid number or '0' to exit.")

        # Find bookings for the specific mode and number
        bookings = booked_tickets_collection.find({f"{mode}_number": specific_number})

        # Display passengers details for the specific number
        print(f"\nPassengers Details for {mode.capitalize()} Number: {specific_number}:")
        for booking in bookings:
            passengers = booking.get("passengers", [])
            print(f"Username: {booking['username']}, Mode: {booking['mode']}, Source:
{booking['source']}, Destination: {booking['destination']}, Date: {booking['date']},
Time: {booking.get('time', '')}")

            if passengers:
                print("Passengers:")
                for passenger in passengers:
                    print(f"   Name: {passenger['name']}, Age: {passenger['age']}, Seat:
{passenger['seat']}")
            else:
                print("No passengers for this booking.")

            print()


# Function to add a new route based on user input
def add_new_route():
    mode = input("Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train):
")
    if mode not in ['1', '2', '3']:
        print("Invalid mode of transportation.")
        return

    # Mapping mode input to mode string
    mode_map = {'1': 'Bus', '2': 'Flight', '3': 'Train'}
    mode = mode_map[mode]

    source = input("Enter source station: ")
    destination = input("Enter destination station: ")
    time = input("Enter departure time: ")
    price = float(input("Enter ticket price: "))
```

```
    new_route = {
        "source": source,
        "destination": destination,
        "time": time,
        "price": price
    }

    # Depending on mode, prompt for specific vehicle number and insert the new route into
the corresponding collection
    if mode == 'Bus':
        bus_number = input("Enter bus number: ")
        new_route["bus_number"] = bus_number
        bus_collection.insert_one(new_route)
        print("New bus route added successfully!")
    elif mode == 'Train':
        train_number = input("Enter train number: ")
        new_route["train_number"] = train_number
        train_collection.insert_one(new_route)
        print("New train route added successfully!")
    elif mode == 'Flight':
        flight_number = input("Enter flight number: ")
        new_route["flight_number"] = flight_number
        flight_collection.insert_one(new_route)
        print("New flight route added successfully!")

# Function to update the timing of a route
def update_route_timing():
    mode = input("Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train):
")
    if mode not in ['1', '2', '3']:
        print("Invalid mode of transportation.")
        return

    source = input("Enter source: ")
    destination = input("Enter destination: ")

    # Determine the collection and mode string based on the mode of transportation
    if mode == '1':
        collection = bus_collection
        mode_str = 'Bus'
    elif mode == '2':
        collection = flight_collection
        mode_str = 'Flight'
    elif mode == '3':
        collection = train_collection
        mode_str = 'Train'

    # Retrieve routes based on source and destination
    routes = collection.find({"source": source, "destination": destination})
    routes_list = list(routes)
```

```
    # If no routes found, print a message and return
    if not routes_list:
        print(f"No {mode_str} routes found for the given source and destination.")
        return

    # Display available routes for the selected mode
    print(f"Available {mode_str} Routes:")
    for i, route in enumerate(routes_list, 1):
        if mode_str == 'Bus':
            print(f"{i}. Source: {source}, Destination: {destination}, Time:
{route['time']}, Bus Number: {route['bus_number']}")
        elif mode_str == 'Flight':
            print(f"{i}. Source: {source}, Destination: {destination}, Time:
{route['time']}, Flight Number: {route['flight_number']}")
        elif mode_str == 'Train':
            print(f"{i}. Source: {source}, Destination: {destination}, Time:
{route['time']}, Train Number: {route['train_number']}")

    # Input for selecting the route to update timing
    choice = input("Enter the number of the route to update timing: ")

    if choice.isdigit():
        route_index = int(choice) - 1
        if 0 <= route_index < len(routes_list):
            route_to_update = routes_list[route_index]
            new_time = input("Enter new departure time: ")
            collection.update_one({"_id": route_to_update["_id"]}, {"$set": {"time":
new_time}})
            print("Route timing updated successfully.")
        else:
            print("Invalid route number.")
    else:
        print("Invalid input. Please enter a number.")

# Function to update the pricing of a route
def update_route_pricing():
    mode = input("Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train):
")
    if mode not in ['1', '2', '3']:
        print("Invalid mode of transportation.")
        return

    source = input("Enter source: ")
    destination = input("Enter destination: ")

    # Determine the collection and mode string based on the mode of transportation
    if mode == '1':
        collection = bus_collection
        mode_str = 'Bus'
```

```
    elif mode == '2':
        collection = flight_collection
        mode_str = 'Flight'
    elif mode == '3':
        collection = train_collection
        mode_str = 'Train'

    # Retrieve routes based on source and destination
    routes = collection.find({"source": source, "destination": destination})
    routes_list = list(routes)

    # If no routes found, print a message and return
    if not routes_list:
        print(f"No {mode_str} routes found for the given source and destination.")
        return

    # Display available routes for the selected mode
    print(f"Available {mode_str} Routes:")
    for i, route in enumerate(routes_list, 1):
        if mode_str == 'Bus':
            print(f"{i}. Source: {source}, Destination: {destination}, Price:
{route['price']}, Bus Number: {route['bus_number']}")
        elif mode_str == 'Flight':
            print(f"{i}. Source: {source}, Destination: {destination}, Price:
{route['price']}, Flight Number: {route['flight_number']}")
        elif mode_str == 'Train':
            print(f"{i}. Source: {source}, Destination: {destination}, Price:
{route['price']}, Train Number: {route['train_number']}")

    # Input for selecting the route to update pricing
    choice = input("Enter the number of the route to update pricing: ")

    if choice.isdigit():
        route_index = int(choice) - 1
        if 0 <= route_index < len(routes_list):
            route_to_update = routes_list[route_index]
            new_price = float(input("Enter new ticket price: "))
            collection.update_one({"_id": route_to_update["_id"]}, {"$set": {"price":
new_price}})
            print("Route pricing updated successfully.")
        else:
            print("Invalid route number.")
    else:
        print("Invalid input. Please enter a number.")

# Function to view all routes for a selected mode of transportation
def view_all_routes():
    while True:
        mode = input("Choose mode of transportation (1 for Bus, 2 for Train, 3 for
Flight, 0 to exit): ")
```

```
        if mode == '0':
            print("Exiting...")
            return
        elif mode not in ['1', '2', '3']:
            print("Invalid mode of transportation.")
        else:
            # Determine the mode string and corresponding collection based on user input
            if mode == '1':
                mode_str = 'Bus'
                routes = bus_collection.find({})
            elif mode == '2':
                mode_str = 'Train'
                routes = train_collection.find({})
            elif mode == '3':
                mode_str = 'Flight'
                routes = flight_collection.find({})

            # Display all routes for the selected mode
            print(f"All {mode_str} routes:")
            for route in routes:
                print(f"Source: {route['source']}, Destination: {route['destination']},
Time: {route['time']}, Price: {route['price']}")


# Entry point of the program
if __name__ == "__main__":
    main_menu()
```

Output:

1. **New User Registration:**

```
Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice: 1
Enter name: Dhruv Kumar
Enter age: 21
Enter email: dhruvkumar@gmail.com
Create username: Dhuruv
Enter password: Dk123
Re-enter password: Dk123
Registration successful!
```

2. **User login:**

```
Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice: 2
Enter username: Dhuruv
Enter password: Dk123
Login successful!
hii! Dhuruv welcome to UPES ticket booking system

Ticket Menu
1. Book Ticket
2. View Tickets
3. Cancel Booking
4. Logout
Enter your choice: 
```

2.1 Booking Ticket:

2.1.1 Booking Bus Ticket

```
Ticket Menu
1. Book Ticket
2. View Tickets
3. Cancel Booking
4. Logout
Enter your choice: 1


Transportation Modes:
1. Bus
2. Train
3. Flight
4. Back
Choose mode of transportation (1/2/3): 1
Enter source station: Mumbai
Enter destination station: Pune
Enter date (YYYY-MM-DD): 2024-08-27
slot 1 --> Source: Mumbai, Destination: Pune, Time: 8:00 AM, Price per passenger: 2500.0
slot 2 --> Source: Mumbai, Destination: Pune, Time: 9:00 AM, Price per passenger: 1100
slot 3 --> Source: Mumbai, Destination: Pune, Time: 10:00 AM, Price per passenger: 1150
slot 4 --> Source: Mumbai, Destination: Pune, Time: 11:00 AM, Price per passenger: 1200
slot 5 --> Source: Mumbai, Destination: Pune, Time: 12:00 AM, Price per passenger: 2100.0
Enter Choice: 3
Enter the number of passengers: 2
Enter name of passenger 1: Dhruv
Enter age of passenger 1: 21
Enter name of passenger 2: Ram
Enter age of passenger 2: 25
Booked: Mumbai Pune 10:00 AM Rs. 2300 Date: 2024-08-27
Passengers:
Name: Dhruv, Age: 21, Seat: 1
Name: Ram, Age: 25, Seat: 2
Do you want to book this bus? (1 for yes, 2 for no): 1
Bus ticket booked successfully!
```

2.1.2 Booking Train Ticket

```
Transportation Modes:
1. Bus
2. Train
3. Flight
4. Back
Choose mode of transportation (1/2/3): 2
Enter source station: Kolkata
Enter destination station: Patna
Enter date (YYYY-MM-DD): 2024-09-14
slot 1 --> Source: Kolkata, Destination: Patna, Time: 8:00 AM, Price: 3050
slot 2 --> Source: Kolkata, Destination: Patna, Time: 10:00 AM, Price: 3150
slot 3 --> Source: Kolkata, Destination: Patna, Time: 05:00 PM, Price: 3200
slot 4 --> Source: Kolkata, Destination: Patna, Time: 12:00 AM, Price: 3250
Enter Choice: 3
Enter the number of passengers: 2
Enter name of passenger 1: Sohan
Enter age of passenger 1: 25
Enter name of passenger 2: Ruhi
Enter age of passenger 2: 24
Booked: Kolkata Patna 05:00 PM Rs. 6400 Date: 2024-09-14
Passengers:
Name: Sohan, Age: 25, Seat: 1
Name: Ruhi, Age: 24, Seat: 2
Do you want to book this train? (1 for yes, 2 for no): 1
Train ticket booked successfully!
```

2.1.3 Booking Flight Ticket

```
Transportation Modes:
1. Bus
2. Train
3. Flight
4. Back
Choose mode of transportation (1/2/3): 3
Enter source station: Delhi
Enter destination station: Dehradun
Enter date (YYYY-MM-DD): 2025-01-01
slot 1 --> Source: Delhi, Destination: Dehradun, Time: 02:00 PM, Price: 5050
slot 2 --> Source: Delhi, Destination: Dehradun, Time: 05:00 PM, Price: 5100
slot 3 --> Source: Delhi, Destination: Dehradun, Time: 05:00 PM, Price: 5150
slot 4 --> Source: Delhi, Destination: Dehradun, Time: 05:00 PM, Price: 5200
slot 5 --> Source: Delhi, Destination: Dehradun, Time: 05:00 PM, Price: 5250
Enter Choice: 5
Enter the number of passengers: 4
Enter name of passenger 1: Sham
Enter age of passenger 1: 52
Enter name of passenger 2: Adi
Enter age of passenger 2: 30
Enter name of passenger 3: Deep
Enter age of passenger 3: 26
Enter name of passenger 4: Ishan
Enter age of passenger 4: 27
Booked: Delhi Dehradun 05:00 PM Rs. 21000 Date: 2025-01-01
Passengers:
Name: Sham, Age: 52, Seat: 1
Name: Adi, Age: 30, Seat: 2
Name: Deep, Age: 26, Seat: 3
Name: Ishan, Age: 27, Seat: 4
Do you want to book this flight? (1 for yes, 2 for no): 1
Flight ticket booked successfully!
```

2.1.4 Back

```
Transportation Modes:
1. Bus
2. Train
3. Flight
4. Back
Choose mode of transportation (1/2/3): 4

Ticket Menu
1. Book Ticket
2. View Tickets
3. Cancel Booking
4. Logout
Enter your choice: 
```

2.2 View Ticket:

```
Ticket Menu
1. Book Ticket
2. View Tickets
3. Cancel Booking
4. Logout
Enter your choice: 2

Ticket Number: 660fdeb7f024e29bf47fac2c
Source: Mumbai
Destination: Pune
Time: 10:00 AM
Date: 2024-08-27
Price: 2300
Bus number: 1003
Passengers:
Name: Dhruv, Age: 21, Seat number: 1
Name: Ram, Age: 25, Seat number: 2

Ticket Number: 660fe256ddfd93cfcccf9b05
Source: Mumbai
Destination: Pune
Time: 8:00 AM
Date: 2024-09-14
Price: 9150
Train number: 10001
Passengers:
Name: Mohan, Age: 35, Seat number: 1
Name: Vishal, Age: 19, Seat number: 2
Name: Himanshi, Age: 20, Seat number: 3

Ticket Number: 660fe32addfd93cfcccf9b06
Source: Delhi
Destination: Dehradun
Time: 05:00 PM
Date: 2025-01-01
Price: 21000
Flight number: 5005
Passengers:
Name: Sham, Age: 52, Seat number: 1
Name: Adi, Age: 30, Seat number: 2
Name: Deep, Age: 26, Seat number: 3
Name: Ishan, Age: 27, Seat number: 4
```

2.3 Cancel Ticket:

```
Ticket Menu
1. Book Ticket
2. View Tickets
3. Cancel Booking
4. Logout
Enter your choice: 3

Enter mode of transportation (1 for bus, 2 for flight, 3 for train): 3
Existing bookings:
1. Source: Mumbai, Destination: Pune, Date: 2024-09-14, Time: 8:00 AM
Enter the number of booking to cancel: 1
Booking canceled successfully.
```

2.4 Logout:

```
Ticket Menu
1. Book Ticket
2. View Tickets
3. Cancel Booking
4. Logout
Enter your choice: 4

Logged out successfully.

Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice:
```

3. **Admin login:**

```
Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice: Dhruv
Invalid choice. Please try again.

Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice: 2
Enter username: Dhruv
Enter password: Dhruv@30
Admin login successful!

Admin Menu
1. Add New Route
2. Update Route Timing
3. View All Routes
4. Update Route Pricing
5. Delete Existing Route
6. View All Bookings
7. View Route Statistics
8. View details for specific vehicle
9. Logout
Enter your choice: 
```

3.1 Adding new route:

```
Enter your choice: 1

Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train): 3
Enter source station: Delhi
Enter destination station: Mumbai
Enter departure time: 06:45 PM
Enter ticket price: 6149
Enter train number: 01256
New train route added successfully!
```

3.2 Update route timing:

```
Enter your choice: 2

Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train): 2
Enter source: Chennai
Enter destination: Bangalore
Available Flight Routes:
1. Source: Chennai, Destination: Bangalore, Time: 8:00 AM, Flight Number: 301
2. Source: Chennai, Destination: Bangalore, Time: 9:00 AM, Flight Number: 302
3. Source: Chennai, Destination: Bangalore, Time: 10:00 AM, Flight Number: 303
4. Source: Chennai, Destination: Bangalore, Time: 11:00 AM, Flight Number: 304
5. Source: Chennai, Destination: Bangalore, Time: 12:00 AM, Flight Number: 305
Enter the number of the route to update timing: 4
Enter new departure time: 09:35 PM
Route timing updated successfully.
```

3.3 View all the available route:

```
Enter your choice: 3

Choose mode of transportation (1 for Bus, 2 for Train, 3 for Flight, 0 to exit): 1
All Bus routes:
Source: Mumbai, Destination: Pune, Time: 8:00 AM, Price: 2500.0
Source: Mumbai, Destination: Pune, Time: 9:00 AM, Price: 1100
Source: Mumbai, Destination: Pune, Time: 10:00 AM, Price: 1150
Source: Mumbai, Destination: Pune, Time: 11:00 AM, Price: 1200
Source: Mumbai, Destination: Pune, Time: 12:00 AM, Price: 2100.0
Source: Delhi, Destination: Jaipur, Time: 8:00 AM, Price: 1050
Source: Delhi, Destination: Jaipur, Time: 9:00 AM, Price: 1100
Source: Delhi, Destination: Jaipur, Time: 10:00 AM, Price: 1150
Source: Delhi, Destination: Jaipur, Time: 11:00 AM, Price: 1200
Source: Delhi, Destination: Jaipur, Time: 12:00 AM, Price: 1250
Source: Chennai, Destination: Bangalore, Time: 8:00 AM, Price: 1050
Source: Chennai, Destination: Bangalore, Time: 9:00 AM, Price: 1100
Source: Chennai, Destination: Bangalore, Time: 10:00 AM, Price: 1150
Source: Chennai, Destination: Bangalore, Time: 11:00 AM, Price: 1200
Source: Chennai, Destination: Bangalore, Time: 12:00 AM, Price: 1250
Source: Kolkata, Destination: Patna, Time: 8:00 AM, Price: 1050
Source: Kolkata, Destination: Patna, Time: 9:00 AM, Price: 1100
Source: Kolkata, Destination: Patna, Time: 10:00 AM, Price: 1150
Source: Kolkata, Destination: Patna, Time: 11:00 AM, Price: 1200
Source: Kolkata, Destination: Patna, Time: 12:00 AM, Price: 1250
Source: Hyderabad, Destination: Vizag, Time: 8:00 AM, Price: 1050
Source: Hyderabad, Destination: Vizag, Time: 9:00 AM, Price: 1100
Source: Hyderabad, Destination: Vizag, Time: 10:00 AM, Price: 1150
Source: Hyderabad, Destination: Vizag, Time: 11:00 AM, Price: 1200
Source: Hyderabad, Destination: Vizag, Time: 12:00 AM, Price: 1250
Source: Ahmedabad, Destination: Surat, Time: 8:00 AM, Price: 1050
Source: Ahmedabad, Destination: Surat, Time: 9:00 AM, Price: 1100
Source: Ahmedabad, Destination: Surat, Time: 10:00 AM, Price: 1150
```

3.4 Update route pricing:

```
Admin Menu
1. Add New Route
2. Update Route Timing
3. View All Routes
4. Update Route Pricing
5. Delete Existing Route
6. View All Bookings
7. View Route Statistics
8. View details for specific vehicle
9. Logout
Enter your choice: 4

Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train): 2
Enter source: Kolkata
Enter destination: Patna
Available Flight Routes:
1. Source: Kolkata, Destination: Patna, Price: 8765.0, Flight Number: 401
2. Source: Kolkata, Destination: Patna, Price: 5100, Flight Number: 402
3. Source: Kolkata, Destination: Patna, Price: 5150, Flight Number: 403
4. Source: Kolkata, Destination: Patna, Price: 12365.0, Flight Number: 404
5. Source: Kolkata, Destination: Patna, Price: 5250, Flight Number: 405
Enter the number of the route to update pricing: 3
Enter new ticket price: 6549
Route pricing updated successfully.
```

3.5 Delete Existing route:

```
Enter your choice: 5

Enter mode of transportation (1 for Bus / 2 for Flight / 3 for Train): 3
Enter source: Kolkata
Enter destination: Patna
Available Train Routes:
1. Source: Kolkata, Destination: Patna, Price: 3050, Train Number: 40001
2. Source: Kolkata, Destination: Patna, Price: 3100, Train Number: 40002
3. Source: Kolkata, Destination: Patna, Price: 3150, Train Number: 40003
4. Source: Kolkata, Destination: Patna, Price: 3200, Train Number: 40004
5. Source: Kolkata, Destination: Patna, Price: 3250, Train Number: 40005
Enter the number of the route to delete: 2
Route deleted successfully.
```

3.6 View all existing bookings:

```
Enter your choice: 6

Booking Summary:
Mode: Bus, Total Tickets booked: 1, Total Passengers: 2
Mode: Flight, Total Tickets booked: 1, Total Passengers: 4
Mode: Train, Total Tickets booked: 1, Total Passengers: 2

All Bookings:
1. Username: Dhuruv, Mode: Bus, Source: Mumbai, Destination: Pune, Date: 2024-08-27, Time: Date: 10:00 AM
   Bus Number: 1003
   Passenger count: 2
   Passengers:
      Name: Dhruv, Age: 21, Seat: 1
      Name: Ram, Age: 25, Seat: 2

2. Username: Dhuruv, Mode: Flight, Source: Delhi, Destination: Dehradun, Date: 2025-01-01, Time: Date: 05:00 PM
   Train Number: 5005
   Passenger count: 4
   Passengers:
      Name: Sham, Age: 52, Seat: 1
      Name: Adi, Age: 30, Seat: 2
      Name: Deep, Age: 26, Seat: 3
      Name: Ishan, Age: 27, Seat: 4

3. Username: Dhuruv, Mode: Train, Source: Kolkata, Destination: Patna, Date: 2024-09-14, Time: Date: 05:00 PM
   Train Number: 40004
   Passenger count: 2
   Passengers:
      Name: Sohan, Age: 25, Seat: 1
      Name: Ruhi, Age: 24, Seat: 2
```

3.7 View route Statistics:

```
Vehicle Statistics:
Total Buses: 339
Total Flights: 341
Total Trains: 339
Select mode (1 for Bus / 2 for Flight / 3 for Train, 0 to exit): 2

Number of Flights Available per Route:

Source: Ahmedabad, Destination: Surat, Number of Flights: 5

Source: Allahabad, Destination: Varanasi, Number of Flights: 5

Source: Amritsar, Destination: Chandigarh, Number of Flights: 5

Source: Auli, Destination: Dehradun, Number of Flights: 5

Source: Aurangabad, Destination: Nagpur, Number of Flights: 5

Source: Badrinath, Destination: Dehradun, Number of Flights: 5

Source: Bangalore, Destination: Chennai, Number of Flights: 5

Source: Bangalore, Destination: Mysore, Number of Flights: 5

Source: Bhopal, Destination: Indore, Number of Flights: 5
```

```
Source: Vadodara, Destination: Nashik, Number of Flights: 5

Source: Varanasi, Destination: Allahabad, Number of Flights: 5

Source: Vijayawada, Destination: Visakhapatnam, Number of Flights: 5

Source: Visakhapatnam, Destination: Vijayawada, Number of Flights: 5

Source: Vizag, Destination: Hyderabad, Number of Flights: 5

Total Routes Available: 69
```

3.8 View details for specific vehicle:

```
Enter your choice: 8

Enter mode of transportation (1: bus, 2: flight, 3: train, 0: exit): 2

Flight Numbers with Booked Tickets:
Slot => 1. flight Number: 5005, Total Tickets Booked: 1

Enter a Slot number to show more details (enter '0' to exit): 1

Passengers Details for flight Number: 5005:
Username: Dhuruv, Mode: Flight, Source: Delhi, Destination: Dehradun, Date: 2025-01-01, Time: 05:00 PM
Passengers:
   Name: Sham, Age: 52, Seat: 1
   Name: Adi, Age: 30, Seat: 2
   Name: Deep, Age: 26, Seat: 3
   Name: Ishan, Age: 27, Seat: 4

Enter mode of transportation (1: bus, 2: flight, 3: train, 0: exit): 1

Bus Numbers with Booked Tickets:
Slot => 1. bus Number: 1003, Total Tickets Booked: 1

Enter a Slot number to show more details (enter '0' to exit): 1

Passengers Details for bus Number: 1003:
Username: Dhuruv, Mode: Bus, Source: Mumbai, Destination: Pune, Date: 2024-08-27, Time: 10:00 AM
Passengers:
   Name: Dhruv, Age: 21, Seat: 1
   Name: Ram, Age: 25, Seat: 2
```

3.9 Logout:

```
Admin Menu
1. Add New Route
2. Update Route Timing
3. View All Routes
4. Update Route Pricing
5. Delete Existing Route
6. View All Bookings
7. View Route Statistics
8. View details for specific vehicle
9. Logout
Enter your choice: 9

Logged out successfully.

Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice: 
```

4. About us:

```
Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice: 3

About Us
UPES TRAVEL AGENCY
Address: 216B I,
 Second Floor,Splendor Forum,
 Plot Bearing No. 3,Jasola District Centre, Jasola,
 New Delhi-110025
Contact: 18001028737
```

5. **Exit:**

```
Main Menu
1. Register
2. Login
3. About Us
4. Exit
Enter your choice: 4
Thank you for using the ticket booking system.
```

Mongodb concepts and usage in the given code:

MongoDB is a leading NoSQL database management system renowned for its document-oriented architecture, scalability, and performance. Storing data in JSON-like documents called BSON, MongoDB offers flexibility with dynamic schemas, making it ideal for managing semi-structured and unstructured data. Its distributed architecture supports horizontal scaling through sharding and provides high availability with replica sets. MongoDB's aggregation framework facilitates advanced data analysis and manipulation, while its ad hoc query language enables real-time data retrieval without rigid schema constraints. With a rich ecosystem including official drivers for various languages, MongoDB Atlas for cloud-based deployments, and MongoDB Compass for intuitive GUI administration, MongoDB serves diverse use cases ranging from content management systems to real-time analytics and IoT applications, positioning itself as a versatile solution for modern data management challenges.

1. **MongoClient Connection:**

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
```

- The code initializes a connection to the MongoDB server using **pymongo.MongoClient()**.
- The connection string **"mongodb://localhost:27017/"** specifies the server's address and port.
- This connection enables communication between the Python application and the MongoDB server.

2. Database Selection:

```
db = client["ticket_booking_system"]
```

- Following the MongoClient connection, the code selects the "ticket_booking_system" database using **client["ticket_booking_system"]**.
- The **client** object represents the connection to the MongoDB server, and **"ticket_booking_system"** is the name of the database.
- If the specified database doesn't exist, MongoDB will create it when data is first stored.

2. **Collection Selection:**

```
users_collection = db["users"]
bus_collection = db["Bus"]
flight_collection = db["Flight"]
train_collection = db["Train"]
booked_tickets_collection = db["booked_tickets"]
```

- After selecting the database, the code selects various collections within the "ticket_booking_system" database.
- Collections are analogous to tables in relational databases and store BSON documents.

a. **Users Collection:**

- The **users_collection** object represents the "users" collection within the database.

b. **Bus Collection:**

- The **bus_collection** object represents the "Bus" collection within the database.

c. **Flight Collection:**

- The **flight_collection** object represents the "Flight" collection within the database.

d. **Train Collection:**

- The **train_collection** object represents the "Train" collection within the database.

e. **Booked Tickets Collection:**

- The **booked_tickets_collection** object represents the "booked_tickets" collection within the database

<u>Mondodb Commands:</u>

1. **Inserting User Document:**

```
    user = {"username": username, "password": password, "name": name, "age": age}  #
Create user object
    users_collection.insert_one(user)
```

- **Explanation:** Inserts a single document (**user**) into the **users** collection.

2. **Finding User Document for Login:**

```
    username = input("Enter username: ")  # Input username
    password = input("Enter password: ")  # Input password
   user = users_collection.find_one({"username": username, "password": password})
```

- **Explanation:** Retrieves a single document from the **users** collection where the username and password match the provided credentials.

3. **Inserting Route Document:**

```
book_ticket = {"username": username,
                           "source": source,
                           "destination": destination,
                           "date": date,
                           "flight_number": booked_flight["flight_number"],
                           "time": booked_flight["time"],
                           "price": total_price,
                           "passengers": passengers}
```

- **Explanation:** Retrieves a single document from the **users** collection where the username and password match the provided credentials.

```
  booked_tickets_collection.insert_one(book_ticket)
```

- **Explanation:** Inserts a single document (**new_route**) into the **Bus** collection, representing a new bus route.

4. **Deleting Route Document:**

```
if mode_str == 'Bus':
                print(f"{i}. Source: {source}, Destination: {destination}, Price: {route['price']}, Bus Number: {route['bus_number']}")
        elif mode_str == 'Flight':
                print(f"{i}. Source: {source}, Destination: {destination}, Price: {route['price']}, Flight Number: {route['flight_number']}")
        elif mode_str == 'Train':
                print(f"{i}. Source: {source}, Destination: {destination}, Price: {route['price']}, Train Number: {route['train_number']}")

collection.delete_one({"_id": route_to_delete["_id"]})
```

- **Explanation:** Deletes a single document from the collection (**collection**) where the **_id** field matches the **_id** of the route to be deleted.

5. **Updating Route Timing:**

```
collection.update_one({"_id": route_to_update["_id"]}, {"$set": {"time": new_time}})
```

- **Explanation:** Updates a single document in the collection (**collection**) where the **_id** field matches the **_id** of the route to be updated. It sets the value of the **time** field to **new_time**.

6. **Updating Route Pricing:**

```
new_price = float(input("Enter new ticket price: "))
                collection.update_one({"_id": route_to_update["_id"]}, {"$set": {"price": new_price}})
```

- **Explanation:** Updates a single document in the collection (**collection**) where the **_id** field matches the **_id** of the route to be updated. It sets the value of the **price** field to **new_price**.

7. **Aggregating Route Statistics:**

```
routes = collection.aggregate([
            {"$group": {
                "_id": {"source": "$source", "destination": "$destination"},
                "count": {"$sum": 1}
            }},
            {"$sort": {"_id.source": 1, "_id.destination": 1}}

  ])
```

$group Stage:

- This stage groups documents by the combination of the source and destination fields. It treats documents with the same source and destination as belonging to the same group.
- "_id": Specifies the fields to group by. Here, it's set to an object containing the source and destination fields, which forms a unique identifier for each route.
- "count": Uses the $sum operator to calculate the total number of documents (vehicles) within each group. The value 1 passed to $sum indicates that it should count each document once.

$sort Stage:

- This stage sorts the grouped documents based on the **source** and **destination** fields in ascending order (**1** indicates ascending order).
- Sorting ensures that the routes are displayed in a consistent and organized manner.

After defining the aggregation pipeline stages, the **aggregate()** method is called on the appropriate collection (**bus_collection**, **flight_collection**, or **train_collection**) based on the selected mode. This pipeline is then executed to perform the aggregation operation.

8. **Finding Bookings for Specific Mode:**

```
pipeline = [
            {"$match": {f"{mode}_number": {"$exists": True}}},
            {"$group": {"_id": f"${mode}_number", "total_tickets": {"$sum": 1}}}
        ]
        distinct_numbers = booked_tickets_collection.aggregate(pipeline)
```

$match Stage:

- This stage filters documents based on the existence of a field related to the specific mode of transportation ({f"{mode}_number": {"$exists": True}}). It ensures that only documents with the specified mode number field are considered for further processing.
- In this case, f"{mode}_number" dynamically selects the field based on the chosen **mode.**

$group Stage:

- This stage groups the filtered documents by the mode-specific number.
- "_id": Specifies the field to group by, which is set to f"${mode}_number". This dynamically selects the mode-specific number field based on the chosen mode.
- "total_tickets": Uses the $sum operator to calculate the total number of documents (tickets) within each group. Each document represents a booked ticket.

9. **Counting Documents in Collection:**

```
pipeline = [{"$group": {
                "_id": "$mode",
                "total_tickets": {"$sum": 1},
               "total_passengers": {"$sum": {"$size": "$passengers"}}  # Count total
passengers
            }}]
    # Aggregate the bookings summary
    bookings_summary = booked_tickets_collection.aggregate(pipeline)
```

- **$group Stage:** This stage groups documents by a specified expression and applies accumulator expressions to the grouped documents. In this case:
  - "_id": "$mode": Groups documents by the mode field.
  - "total_tickets": {"$sum": 1}: Calculates the total number of documents (tickets) for each group (mode).
  - "total_passengers": {"$sum": {"$size": "$passengers"}: Calculates the total number of passengers for each group (mode). It uses the $size operator to find the size of the passengers array within each document and then sums these sizes.

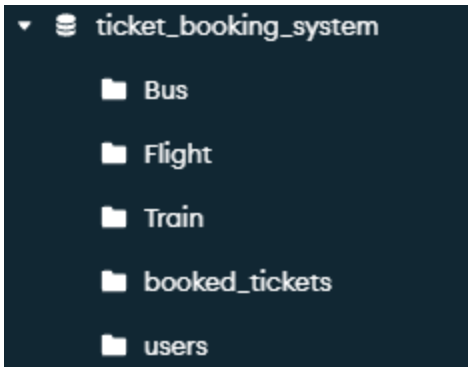
This pipeline aggregates the data based on the mode field, calculating the total number of tickets and passengers for each mode.

10. **Checking for available seats:**

```
booked_seats = set()
      booked_tickets  =  booked_tickets_collection.find({"mode":  mode,  "source":
source, "destination": destination, "date": date})
    for ticket in booked_tickets:
        for passenger in ticket['passengers']:
            booked_seats.add(passenger['seat'])
```

- **find**: This command retrieves documents from the collection that match the specified query criteria. In this function, it is used to find trains available for the specified source and destination.
- **insert_one**: This command inserts a single document into the collection. In this function, it is used to insert the booked train ticket into the **booked_tickets_collection**.
- **find (again)**: This command is used to find already booked tickets for the same mode, source, destination, and date. It is used to check for already booked seats in order to allocate new seats for the current booking.

MongoDb database and collection used :



booked_tickets

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	2	312.00 B	1	36.86 kB

Bus

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
24.58 kB	339	113.00 B	1	36.86 kB

Flight

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
24.58 kB	341	116.00 B	1	36.86 kB

Train

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
24.58 kB	339	115.00 B	1	36.86 kB

users

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
20.48 kB	11	77.00 B	1	36.86 kB

My Queries | Bus | Flight | Train | booked_tickets | users

ticket_booking_system > Bus

Documents 339 | Aggregations | Schema | Indexes 1 | Validation

Type a query: { field: 'value' } or Generate query

Explain | Reset | Find | Options

ADD DATA | EXPORT DATA | UPDATE | DELETE

1 - 20 of 339

```
_id: ObjectId('6609d67bd820915bb040432d')
bus_number : 1001
source : "Mumbai"
destination : "Pune"
price : 2500
time : "8:00 AM"

_id: ObjectId('6609d67bd820915bb040432e')
bus_number : 1002
source : "Mumbai"
destination : "Pune"
price : 1100
time : "9:00 AM"

_id: ObjectId('6609d67bd820915bb040432f')
bus_number : 1003
source : "Mumbai"
destination : "Pune"
price : 1150
time : "10:00 AM"

_id: ObjectId('6609d67bd820915bb0404330')
bus_number : 1004
source : "Mumbai"
destination : "Pune"
price : 1200
time : "11:00 AM"

_id: ObjectId('6609d67bd820915bb0404331')
bus_number : 1005
source : "Mumbai"
destination : "Pune"
price : 2100
time : "12:00 AM"
```

My Queries | Bus | ticket_booking_system... Flight | Train | booked_tickets | users

ticket_booking_system > Flight

Documents 341 | Aggregations | Schema | Indexes 1 | Validation

Type a query: { field: 'value' } or Generate query

Explain | Reset | Find | Options

ADD DATA | EXPORT DATA | UPDATE | DELETE

1 - 20 of 341

```
_id: ObjectId('6609d67bd820915bb0404085')
flight_number : 101
source : "Mumbai"
destination : "Pune"
price : 7085
time : "8:00 AM"

_id: ObjectId('6609d67bd820915bb0404086')
flight_number : 102
source : "Mumbai"
destination : "Pune"
price : 9200
time : "9:00 AM"

_id: ObjectId('6609d67bd820915bb0404087')
flight_number : 103
source : "Mumbai"
destination : "Pune"
price : 5150
time : "10:00 AM"

_id: ObjectId('6609d67bd820915bb0404088')
flight_number : 104
source : "Mumbai"
destination : "Pune"
price : 11244
time : "11:00 AM"

_id: ObjectId('6609d67bd820915bb0404089')
flight_number : 105
source : "Mumbai"
destination : "Pune"
price : 5250
time : "12:00 AM"
```

My Queries | Bus | Flight | Train | booked_tickets | users

ticket_booking_system > Train

Documents 339 | Aggregations | Schema | Indexes 1 | Validation

Type a query: { field: 'value' } or Generate query

Explain | Reset | Find | Options

ADD DATA | EXPORT DATA | UPDATE | DELETE

1 - 20 of 339

```
_id: ObjectId('6609d67bd820915bb04041d9')
train_number : 10001
source : "Mumbai"
destination : "Pune"
price : 3050
time : "8:00 AM"

_id: ObjectId('6609d67bd820915bb04041da')
train_number : 10002
source : "Mumbai"
destination : "Pune"
price : 3100
time : "9:00 AM"

_id: ObjectId('6609d67bd820915bb04041db')
train_number : 10003
source : "Mumbai"
destination : "Pune"
price : 3150
time : "10:00 AM"

_id: ObjectId('6609d67bd820915bb04041dc')
train_number : 10004
source : "Mumbai"
destination : "Pune"
price : 3200
time : "11:00 AM"

_id: ObjectId('6609d67bd820915bb04041dd')
train_number : 10005
source : "Mumbai"
destination : "Pune"
price : 3250
time : "12:00 AM"
```

My Queries | Bus | Flight | Train | booked_tickets | users

ticket_booking_system > booked_tickets

Documents 2 | Aggregations | Schema | Indexes 1 | Validation

Type a query: { field: 'value' } or Generate query

Explain | Reset | Find | Options

ADD DATA | EXPORT DATA | UPDATE | DELETE

1 - 3 of 3

```
_id: ObjectId('660fdeb7f024e29bf47fac2c')
username : "Dhuruv"
mode : "Bus"
source : "Mumbai"
destination : "Pune"
date : "2024-08-27"
bus_number : 1003
time : "10:00 AM"
price : 2300
passengers : Array (2)

_id: ObjectId('660fe32addfd93cfcccf9b06')
username : "Dhuruv"
mode : "Flight"
source : "Delhi"
destination : "Dehradun"
date : "2025-01-01"
flight_number : 5005
time : "05:00 PM"
price : 21000
passengers : Array (4)

_id: ObjectId('660fed51ddfd93cfcccf9b08')
username : "Dhuruv"
mode : "Train"
source : "Kolkata"
destination : "Patna"
date : "2024-09-14"
train_number : 40004
time : "05:00 PM"
price : 6400
passengers : Array (2)
```

My Queries | Bus | Flight | Train | booked_tickets | users

ticket_booking_system > users

Documents 11 | Aggregations | Schema | Indexes 1 | Validation

Type a query: { field: 'value' } or Generate query

Explain | Reset | Find | Options

ADD DATA | EXPORT DATA | UPDATE | DELETE

1 – 11 of 11

```
_id: ObjectId('6603d7ec21fff6b49fa2278a')
username : "Dhruv"
password : "Dhruv@30"

_id: ObjectId('6603d82021fff6b49fa2278b')
username : "sdv"
password : "dsv"

_id: ObjectId('66050c321defd3d5f5b0bf3d')
username : "hii"
password : "hello"

_id: ObjectId('66052b0c4a8f20411dd4b55c')
username : "hii"
password : "hello"

_id: ObjectId('6609c89af61a346ba647b39c')
username : "A"
password : "B"

_id: ObjectId('660b11caebeaffb26a5fcd96')
username : "Krish"
password : "Krish@123"
name : "KRISH"
age : "21"

_id: ObjectId('660b1c30ffe30caa8e7c42b8')
username : "Dk123"
password : "Dk@123"
name : "Dhruv"
age : "21"
```