Mini project report on

Personalized Budget Friendly Travel Planner

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE22CS351A – DBMS Project

Submitted by:

Maitri Maheshkumar Shekhda	**PES2UG22CS294**
Manya Singh	**PES2UG22CS306**

Under the guidance of
Dr. Suja C M
Assistant Professor
PES University

AUG - DEC 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)

Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

Personalized Budget Friendly Travel Planner

is a bonafide work carried out by

Maitri Maheshkumar Shekhda	**PES2UG22CS294**
Manya Singh	**PES2UG22CS306**

In partial fulfilment for the completion of fifth semester DBMS Project (UE22CS351A) in the Program of Study -Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2024 – DEC. 2024. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

Signature
Dr. Suja C M
Assistant Professor

DECLARATION

We hereby declare that the DBMS Project entitled **Personalized Budget Friendly Travel Planner** has been carried out by us under the guidance of **Dr. Suja C M , Assistant Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2024.

Maitri Maheshkumar Shekhda **PES2UG22CS294**

Manya Singh **PES2UG22CS306**

ABSTRACT

The Personalized Budget-Friendly Travel Planner is a web-based application, developed using MySQL and Python-Flask, designed to simplify and personalize travel planning by creating customized itineraries based on user preferences and budgets. Built as an academic project, it provides a structured approach to managing admin authorization, including destination selection, itinerary customization, and booking. The system implements user authentication, role-specific functionalities, and a notification feature to alert users of new travel deals and destination availability. By utilizing a modular code structure, this project offers students practical insights into database management, secure data handling.

TABLE OF CONTENTS

1. INTRODUCTION

The Personalized Budget-Friendly Travel Planner is a user-centered application designed to simplify travel planning by offering customized itineraries tailored to individual preferences and budgets. The system enables users to sign up, log in, and select desired destinations from a list of available options. By inputting a budget range, preferred travel dates, and additional preferences, users can receive optimized itineraries with recommendations for accommodations, activities, and dining options within their financial constraints.

The platform integrates features such as a notification system to alert users about new destinations of interest and the option to securely book and pay for trips online. Additionally, the planner allows users to receive real-time notifications for destination availability and new travel deals. Admin functionality includes the ability to manage destinations, itineraries, and pricing updates to ensure up-to-date information is available to users.

This tool not only provides a streamlined planning experience but also ensures that users can make informed travel decisions that align with their budgetary limitations. The goal is to offer a comprehensive and easy-to-navigate travel planning solution that encourages more people to explore the world affordably.

Key Features:

1. **User Account Management:** Secure sign-up, login, and personalized profile for every user.
2. **Destination Selection and Itinerary Creation:** Allows users to select from multiple destinations and create a detailed itinerary based on preferences.
3. **Budget Integration:** Customized recommendations and itineraries within specified budget ranges.
4. **Notification System:** Alerts users of new travel deals, destination availability, and important updates.
5. **Admin Control Panel:** Enables administrators to add, modify, or delete destinations and itineraries.

Objective: To provide a cost-effective, efficient, and user-friendly travel planning solution that makes budget-conscious travel accessible to a wider audience.

2a . PROBLEM DEFINITION

Planning a personalized and budget-friendly trip can be a complex and time-consuming process, especially for individuals trying to balance their preferences with financial limitations. Travelers often struggle with finding affordable destinations, suitable accommodations, and activities within their budget while keeping up-to-date on travel deals and destination availability. This lack of a streamlined, cost-effective planning tool limits the ability of budget-conscious travelers to make informed and efficient travel decisions.

The Personalized Budget-Friendly Travel Planner addresses this problem by providing a user-centered platform that generates customized itineraries tailored to individual preferences and budgetary constraints. This project aims to simplify the travel planning process, ensuring that users have access to real-time destination updates, optimized recommendations, and secure booking options, all within a single platform. The goal is to enhance accessibility and affordability in travel planning, empowering users to explore new destinations confidently within their financial means.

2b . USER REQUIREMENT SPECIFICATION

1. Functional Requirements

- User Management: Enable secure user registration, login, and profile management.
- Itinerary Customization: Generate tailored itineraries based on user preferences, budgets, and travel dates.
- Destination and Budget Filtering: Allow users to filter options by budget and preferences.
- Notifications: Provide real-time alerts on new destinations, availability, and deals.
- Booking and Payment: Support secure booking and payment for selected itineraries.
- Admin Controls: Allow admins to manage users, destinations, and pricing.

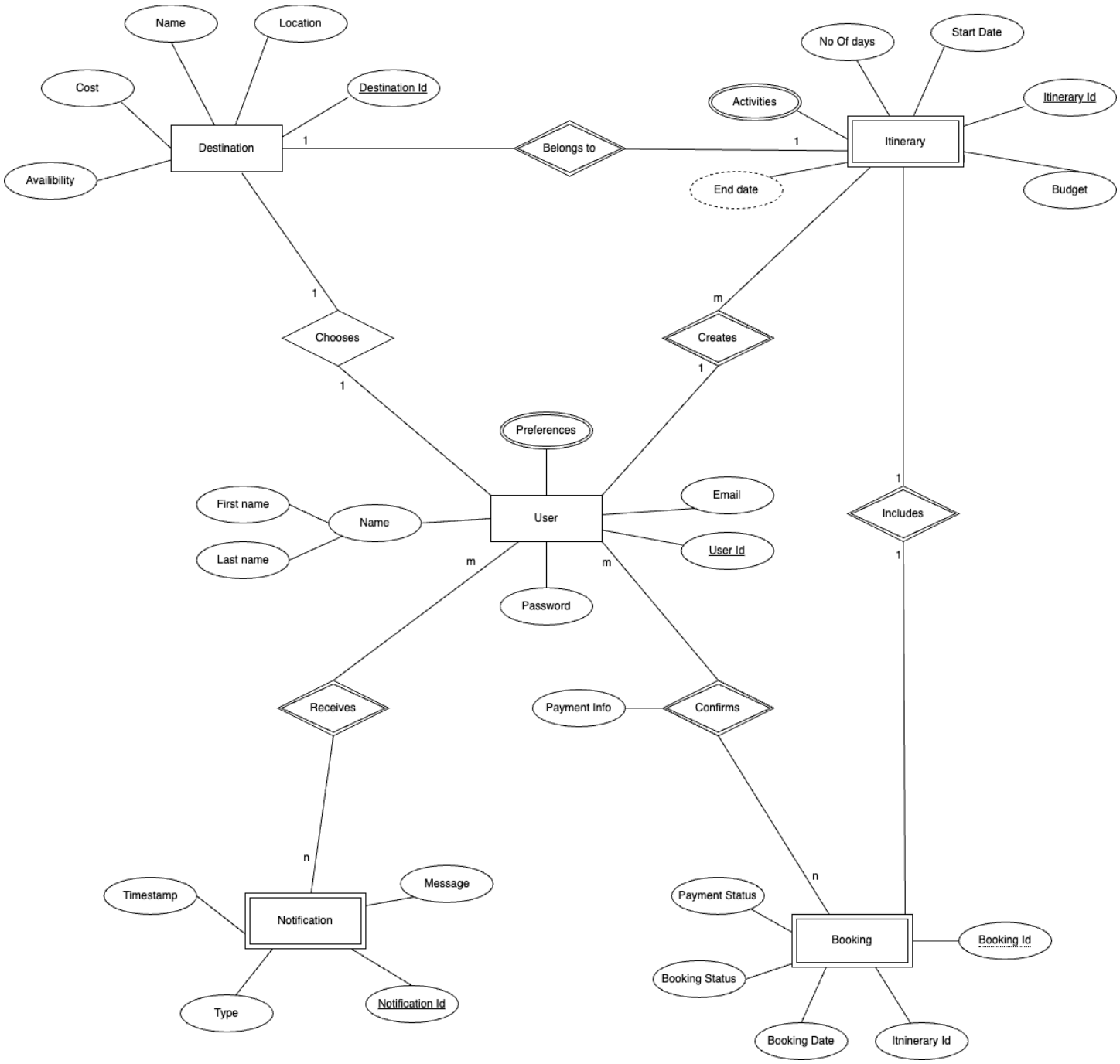
2. Non-Functional Requirements

- Usability: Ensure a user-friendly, intuitive interface.
- Performance: Provide quick response times for itinerary creation and booking.
- Scalability: Design for growing numbers of users and destinations.
- Security: Protect user data with encryption and secure authentication.
- Reliability: Maintain high availability with minimal downtime.
- Compatibility: Support all major devices and web browsers.
- Compliance: Adhere to relevant data protection regulations (e.g., GDPR).

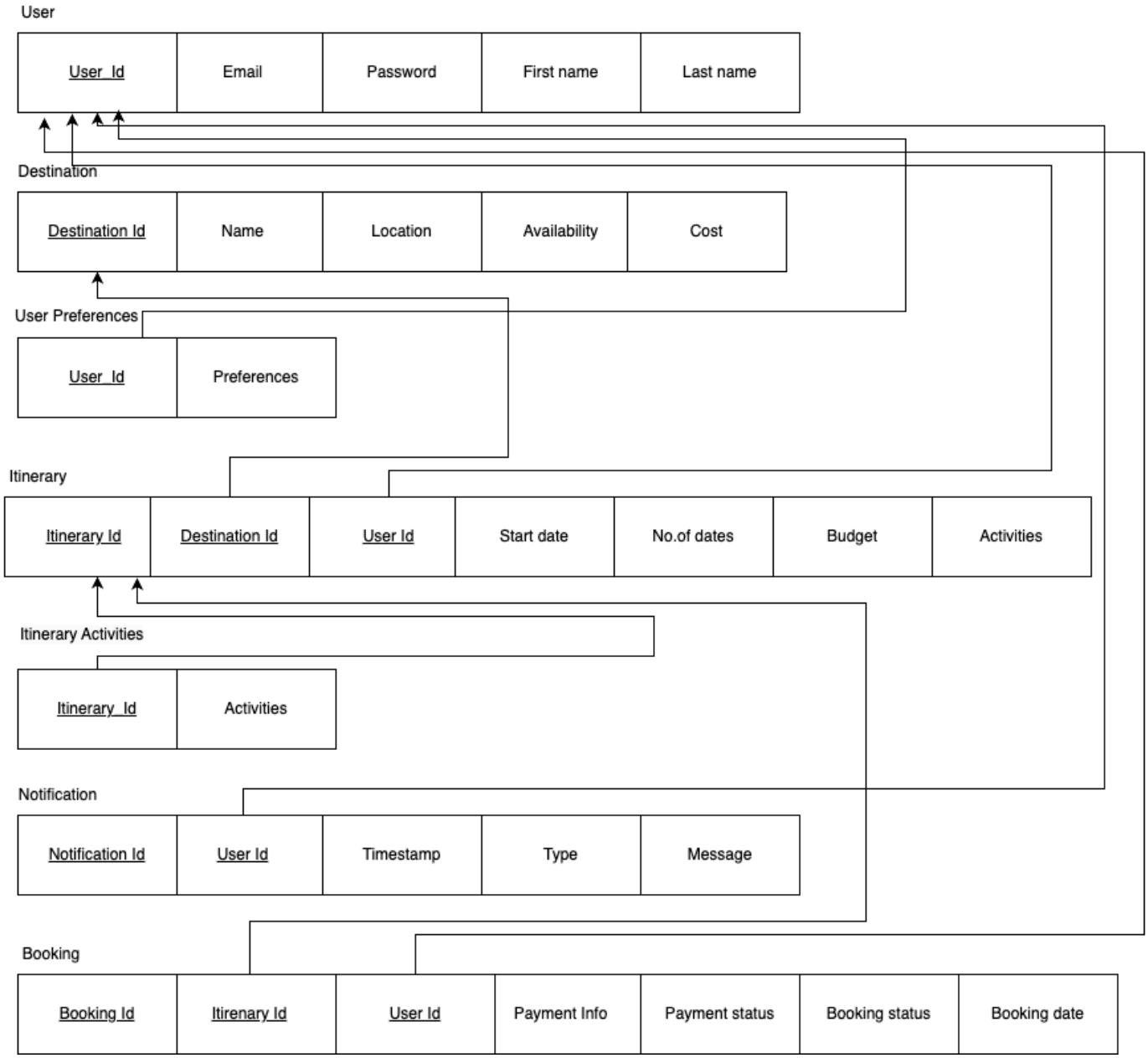
3. LIST OF SOFTWARES/TOOLS/PROGRAMMING LANGUAGES

- Python 3.x: Core language for application logic
- MySQL: Backend database for data storage and retrieval
- Python MySQL Connector: To facilitate database interactions from Python
- Python Flask:For frontend
- Visual Studio Code: Code Editor
- Git : Version Control

4. ER Model



5. ER MODEL TO RELATIONAL MAPPING



6. DDI STATEMENTS

CREATE STATEMENTS:

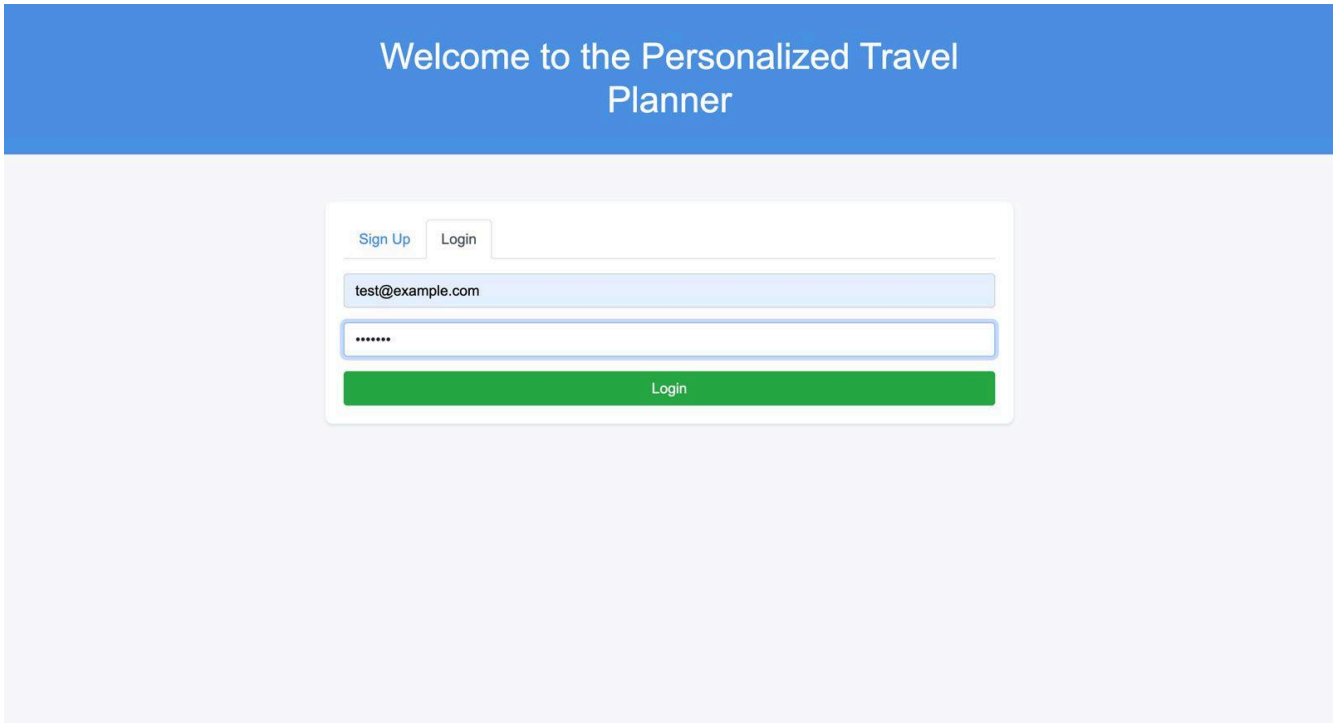
```sql
CREATE TABLE User (
   User_Id INT AUTO_INCREMENT PRIMARY KEY,
   Email VARCHAR(255) NOT NULL UNIQUE,
   Password VARCHAR(255) NOT NULL,
   First_Name VARCHAR(255),
   Last_Name VARCHAR(255)
);

CREATE TABLE Destination (
   Destination_Id INT AUTO_INCREMENT PRIMARY KEY,
   Name VARCHAR(255),
   Location VARCHAR(255),
   Availability BOOLEAN,
   Cost DECIMAL(10, 2)
);

CREATE TABLE UserPreferences (
   User_Id INT,
   Preferences TEXT,
   FOREIGN KEY (User_Id) REFERENCES User(User_Id)
);

CREATE TABLE Itinerary (
   Itinerary_Id INT AUTO_INCREMENT PRIMARY KEY,
   Destination_Id INT,
   User_Id INT,
   Start_Date DATE,
   No_of_Dates INT,
   Budget DECIMAL(10, 2),
   Activities TEXT,
   FOREIGN KEY (Destination_Id) REFERENCES Destination(Destination_Id),
   FOREIGN KEY (User_Id) REFERENCES User(User_Id)
);

CREATE TABLE ItineraryActivities (
   Itinerary_Id INT,
   Activities TEXT,
   FOREIGN KEY (Itinerary_Id) REFERENCES Itinerary(Itinerary_Id)
```

```sql
);

CREATE TABLE Notification (
   Notification_Id INT AUTO_INCREMENT PRIMARY KEY,
   User_Id INT,
   Timestamp DATETIME,
   Type VARCHAR(255),
   Message TEXT,
   FOREIGN KEY (User_Id) REFERENCES User(User_Id)
);

CREATE TABLE Booking (
   Booking_Id INT AUTO_INCREMENT PRIMARY KEY,
   Itinerary_Id INT,
   User_Id INT,
   Payment_Info TEXT,
   Payment_Status VARCHAR(50),
   Booking_Status VARCHAR(50),
   Booking_Date DATE,
   FOREIGN KEY (Itinerary_Id) REFERENCES Itinerary(Itinerary_Id),
   FOREIGN KEY (User_Id) REFERENCES User(User_Id)
);
```

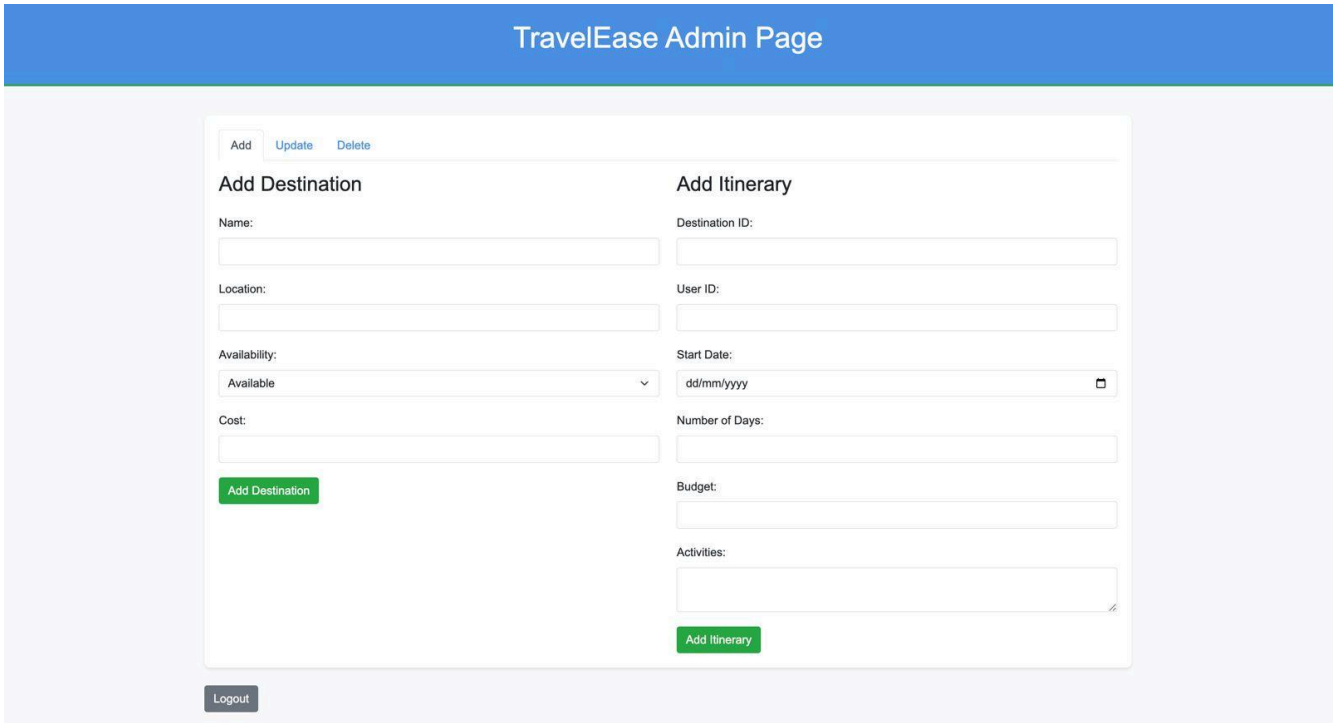
When The DB is initialized for the first time , these commands are run and a user with role admin is created.
 username : test@example.com
 Password : password123

7. DML STATEMENTS(CRUD)

Admin user created on initial initialization of database with credentials Username : admin
Password : admin123



Adding Destination/Itinerary from admin:




```
@app.route('/add_destination', methods=['POST'])
```

```python
def add_destination():

    if request.method == 'POST':

        name = request.form['name']

        location = request.form['location']

        availability = int(request.form['availability'])  # Expect 1 or 0

        cost = float(request.form['cost'])



        conn = create_connection()

        cursor = conn.cursor()



        # Insert the new destination into the Destination table

        insert_query = """

            INSERT INTO Destination (Name, Location, Availability, Cost)

            VALUES (%s, %s, %s, %s)

        """

        cursor.execute(insert_query, (name, location, availability, cost))

        conn.commit()



        # Redirect back to the admin page or a success page

        flash('Destination added successfully')

        return redirect(url_for('admin'))



    return render_template('admin.html')
```

```python
@app.route('/add_itinerary', methods=['POST'])
def add_itinerary():
    if request.method == 'POST':
        # Retrieve form data
        destination_id = int(request.form['destination_id'])
        user_id = int(request.form['user_id'])
        start_date = request.form['start_date']
        no_of_dates = int(request.form['no_of_dates'])
        budget = float(request.form['budget'])
        activities = request.form['activities']

        # Connect to the database
        conn = create_connection()
        cursor = conn.cursor()

        # Insert the new itinerary into the Itinerary table
        insert_query = """
            INSERT INTO Itinerary (Destination_Id, User_Id, Start_Date,
No_of_Dates, Budget, Activities)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
```

```
        cursor.execute(insert_query, (destination_id, user_id, start_date,
no_of_dates, budget, activities))

        conn.commit()



        # Redirect back to the admin page or a success page

        return redirect(url_for('admin'))



    return render_template('admin.html')
```

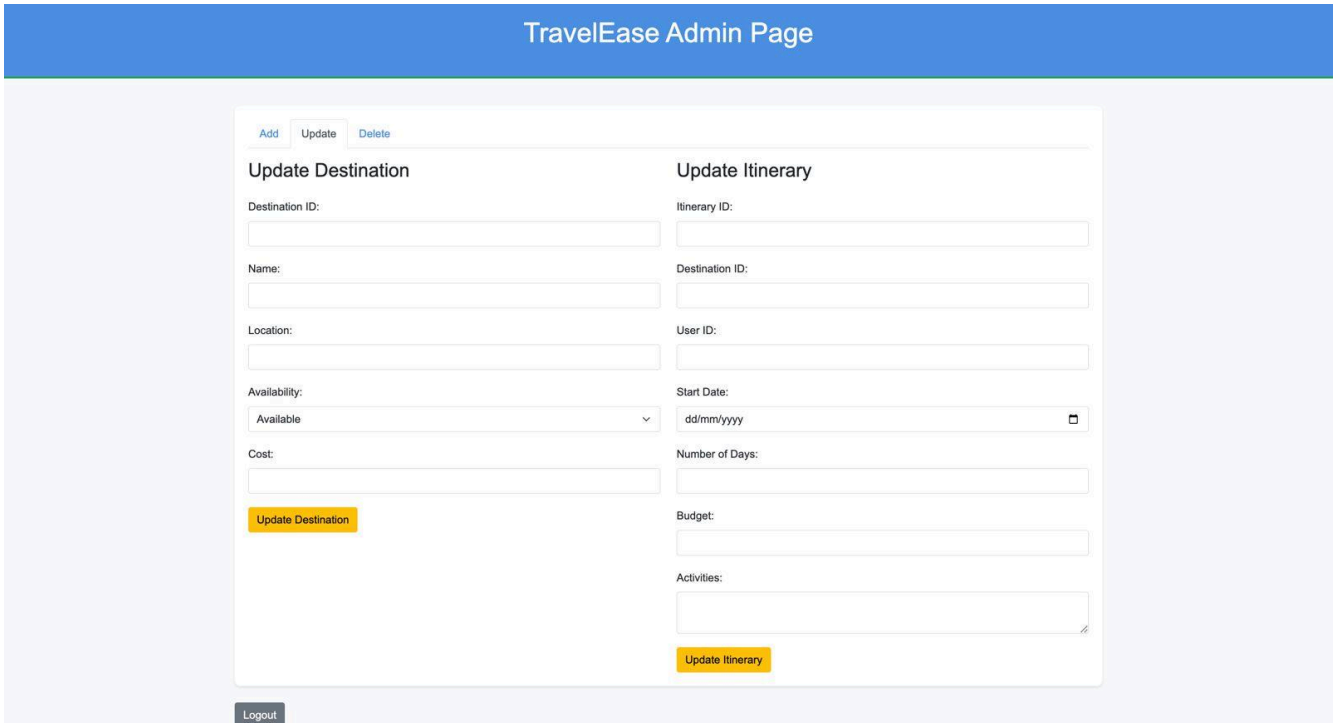
Select * from destination;

Destination_Id	Name	Location	Availability	Cost
1	Japan	Tokyo	1	1500.00
2	Thailand	Bangkok	1	800.00
3	France	Paris	1	2000.00
4	India	Delhi	1	600.00
5	Indonesia	Bali	1	900.00
6	pakistan	istanbul	1	800.00
8	afghanistan	istanbul	1	800.00
9	bangla	dhaka	1	700.00
NULL	NULL	NULL	NULL	NULL

Select * from Itinerary;

Itinerary_Id	Destination_Id	User_Id	Start_Date	No_of_Dates	Budget	Activities
1	1	1	2024-12-01	5	1500.00	Sightseeing in Tokyo, visiting Mt. Fuji, traditional…
2	2	1	2024-11-15	7	800.00	Tour in Bangkok, floating market, visit to temples
3	3	1	2024-12-20	6	2000.00	Eiffel Tower visit, Louvre Museum, Seine River c…
4	4	1	2024-10-25	4	600.00	Delhi city tour, Red Fort, India Gate
5	5	1	2024-11-05	5	900.00	Beach activities in Bali, Ubud Monkey Forest, lo…
6	1	1	2024-12-10	4	1200.00	Explore Osaka, visit Universal Studios Japan, O…
7	1	1	2004-11-17	5	1500.00	Sightseeing in Tokyo, visiting Mt. Fuji, traditional…
8	1	1	2004-11-22	4	1200.00	Explore Osaka, visit Universal Studios Japan, O…

Updating Destination/ Itinerary from Admin:




```python
@app.route('/update_destination', methods=['POST'])

def update_destination():

    destination_id = int(request.form['destination_id'])

    name = request.form['name']

    location = request.form['location']

    availability = int(request.form['availability'])

    cost = float(request.form['cost'])



    conn = create_connection()

    cursor = conn.cursor()



    # Update the destination in the Destination table
```

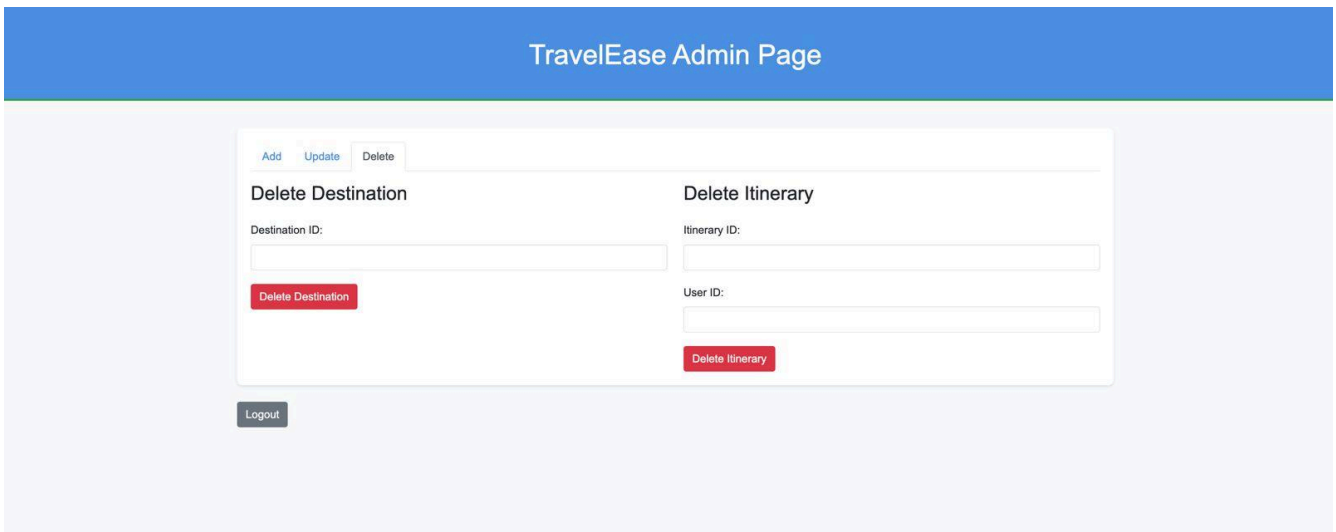
```
    update_query = """

        UPDATE Destination

        SET Name = %s, Location = %s, Availability = %s, Cost = %s

        WHERE Destination_Id = %s

    """

    cursor.execute(update_query, (name, location, availability, cost,
destination_id))

    conn.commit()



    flash('Destination updated successfully')

    notification.send_notification_emails()

    #cursor.execute("TRUNCATE notification")

    conn.commit()

    print("notifs sent")

    return redirect(url_for('admin'))


@app.route('/update_itinerary', methods=['POST'])

def update_itinerary():

    itinerary_id = int(request.form['itinerary_id'])

    destination_id = int(request.form['destination_id'])

    user_id = int(request.form['user_id'])

    start_date = request.form['start_date']

    no_of_dates = int(request.form['no_of_dates'])
```

```
    budget = float(request.form['budget'])

    activities = request.form['activities']



    conn = create_connection()

    cursor = conn.cursor()



    # Update the itinerary in the Itinerary table

    update_query = """

        UPDATE Itinerary

        SET Destination_Id = %s, User_Id = %s, Start_Date = %s, No_of_Dates =
%s, Budget = %s, Activities = %s

        WHERE Itinerary_Id = %s

    """

    cursor.execute(update_query, (destination_id, user_id, start_date,
no_of_dates, budget, activities, itinerary_id))

    conn.commit()



    flash('Itinerary updated successfully')

    return redirect(url_for('admin'))
```

Destination_Id	Name	Location	Availability	Cost
1	Japan	Tokyo	1	1500.00
2	Thailand	Bangkok	1	800.00
3	France	Paris	1	2000.00
4	India	Delhi	1	600.00
5	Indonesia	Bali	1	900.00
6	pakistan	istanbul	1	800.00
8	afghanistan	istanbul	1	800.00
9	bangla	istanbul	0	0.00
NULL	NULL	NULL	NULL	NULL

Itinerary_Id	Destination_Id	User_Id	Start_Date	No_of_Dates	Budget	Activities
1	1	1	2024-12-01	5	1500.00	Sightseeing in Tokyo, visiting Mt. Fuji, traditional…
2	2	1	2024-11-15	7	800.00	Tour in Bangkok, floating market, visit to temples
3	3	1	2024-12-20	6	2000.00	Eiffel Tower visit, Louvre Museum, Seine River c…
4	4	1	2024-10-25	4	600.00	Delhi city tour, Red Fort, India Gate
5	5	1	2024-11-05	5	900.00	Beach activities in Bali, Ubud Monkey Forest, lo…
6	1	1	2024-12-10	4	1200.00	Explore Osaka, visit Universal Studios Japan, O…
7	1	1	2004-11-17	5	1500.00	Sightseeing in Tokyo, visiting Mt. Fuji, traditional…
8	1	1	2024-11-22	5	1000.00	Dotonbori street food tour

Delete Destination/Itinerary from Admin:




```
@app.route('/delete_destination', methods=['POST'])

def delete_destination():

    destination_id = int(request.form['destination_id'])
```

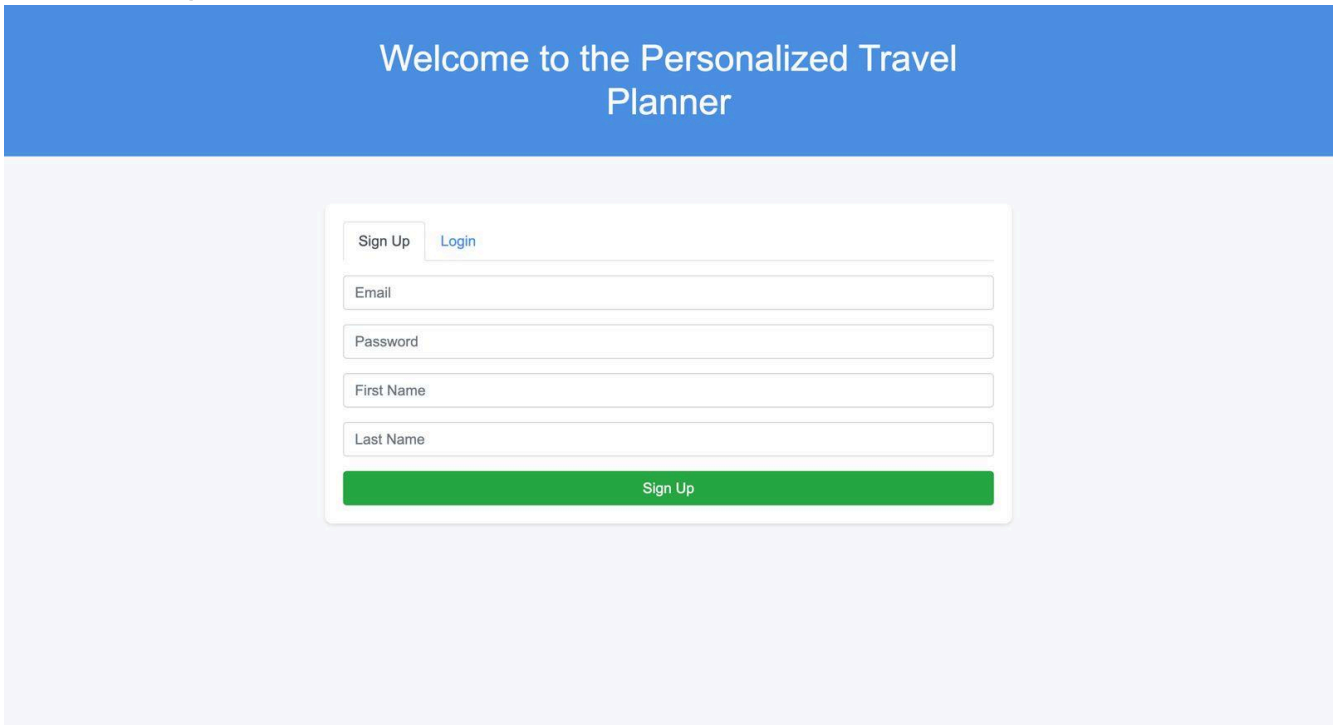
```
    conn = create_connection()

    cursor = conn.cursor()



    # Delete the destination from the Destination table

    delete_query = "DELETE FROM Destination WHERE Destination_Id = %s"

    cursor.execute(delete_query, (destination_id,))

    conn.commit()



    flash('Destination deleted successfully')

    return redirect(url_for('admin'))

@app.route('/delete_itinerary', methods=['POST'])

def delete_itinerary():

    itinerary_id = int(request.form['itinerary_id'])

    user_id = int(request.form['user_id'])



    conn = create_connection()

    cursor = conn.cursor()



    # Delete the itinerary from the Itinerary table

    delete_query = "DELETE FROM Itinerary WHERE Itinerary_Id = %s AND User_Id =
%s"

    cursor.execute(delete_query, (itinerary_id, user_id))

    conn.commit()
```

```
flash('Itinerary deleted successfully')

return redirect(url_for('admin'))
```

Itinerary_Id	Destination_Id	User_Id	Start_Date	No_of_Dates	Budget	Activities
1	1	1	2024-12-01	5	1500.00	Sightseeing in Tokyo, visiting Mt. Fuji, traditional…
2	2	1	2024-11-15	7	800.00	Tour in Bangkok, floating market, visit to temples
3	3	1	2024-12-20	6	2000.00	Eiffel Tower visit, Louvre Museum, Seine River c…
4	4	1	2024-10-25	4	600.00	Delhi city tour, Red Fort, India Gate
5	5	1	2024-11-05	5	900.00	Beach activities in Bali, Ubud Monkey Forest, lo…
6	1	1	2024-12-10	4	1200.00	Explore Osaka, visit Universal Studios Japan, O…
7	1	1	2004-11-17	5	1500.00	Sightseeing in Tokyo, visiting Mt. Fuji, traditional…

Destination_Id	Name	Location	Availability	Cost
1	Japan	Tokyo	1	1500.00
2	Thailand	Bangkok	1	800.00
3	France	Paris	1	2000.00
4	India	Delhi	1	600.00
5	Indonesia	Bali	1	900.00
6	pakistan	istanbul	1	800.00
8	afghanistan	istanbul	1	800.00
NULL	NULL	NULL	NULL	NULL

New user sign up:



If existing user tries to sign up again:

There was an error creating the user.

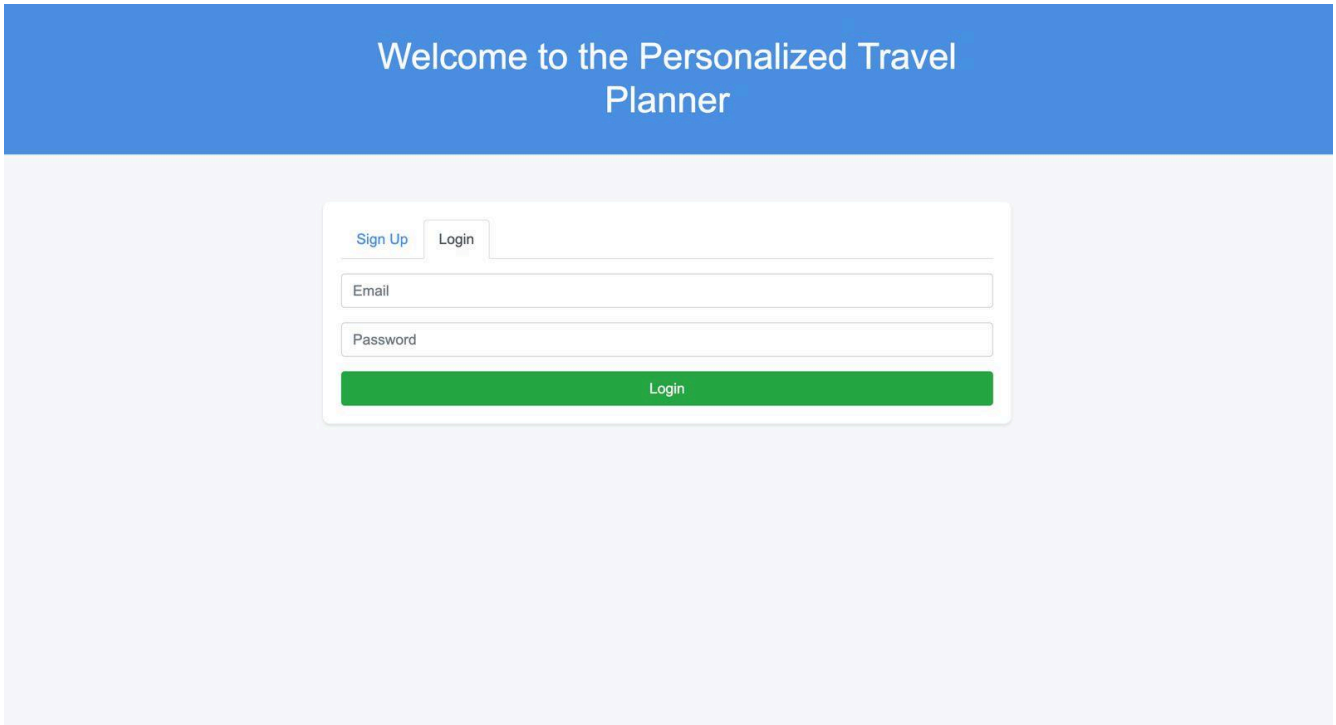
```
def add_user(connection, email, password, first_name, last_name):

    cursor = connection.cursor()

    try:

        query = "INSERT INTO User (Email, Password, First_Name, Last_Name)
VALUES (%s, %s, %s, %s)"

        cursor.execute(query, (email, password, first_name, last_name))

        connection.commit()

        return cursor.lastrowid
```

```python
    except Error as e:
        print(f"An error occurred: {e}")
        return None
@app.route('/submit_user', methods=['POST'])
def submit_user():
    conn = create_connection()
    if conn is None:
        return "Error connecting to the database"

    # Retrieve form data
    email = request.form['email']
    password = request.form['password']
    first_name = request.form['first_name']
    last_name = request.form['last_name']

    # Add user to the database
    user_id = add_user(conn, email, password, first_name, last_name)

    if user_id:
        return f"User created with ID: {user_id}"
    else:
        return "There was an error creating the user."
```

SELECT * from User;

User_Id	Email	Password	First_Name	Last_Name
1	test@example.com	password123	John	Doe
2	manojrakesh@gmail.com	hiiiii	Manoj	Rakesh
3	athreyamr2003@outlook.com	hwll	ath	jiii
5	athreya@outlook.com	hello	Manoj	Rakesh
6	madhura@gmail.com	manya	madhura	H B
7	iot@gmail.com	iot	iot	iot
8	manyasingh1711@gmail.com	hi	manya	singh
NULL	NULL	NULL	NULL	NULL

Existing user login:



If wrong password/email:

Invalid credentials. Please try again.

```python
@app.route('/login', methods=['POST'])

def login():

    conn = create_connection()

    if conn is None:

        return "Error connecting to to the database"



    # Retrieve login data

    email = request.form['email']

    password = request.form['password']









    # Authenticate user

    user = authenticate_user(conn, email, password)

    if email == 'test@example.com' and password == 'password123':

            # Redirect to the admin page

        return redirect(url_for('admin'))

    elif user:

        session['user_id'] = user[0]  # Store user ID in the session

        return redirect(url_for('home'))

    else:

        return "Invalid credentials. Please try again."
```

User itinerary selection:

```
@app.route('/details/<section>', methods=['GET', 'POST'])

def details(section):

    conn = create_connection()

    cursor = conn.cursor()



    # Initialize variables

    itineraries = []

    section_title = ''

    section_content = ''

    show_form = False  # Initially show the destination and budget form

    show_itineraries_form = False  # Show the itineraries selection form after
first submission

    selected_itineraries = []



    # Fetch all destinations from the Destination table

    cursor.execute("SELECT Destination_Id, Name FROM Destination")

    destinations = cursor.fetchall()



    if section == 'first':

        show_form=True

        section_title = 'Choose Your Destination'

        section_content = 'Select a destination and input your budget.'
```

```
        user_id = session.get('user_id')

        selected_destination = request.form.get('destination_id')

    if request.method == 'POST':

        if 'submit_destination' in request.form:

            # First form submission with destination and budget

            selected_destination = request.form.get('destination')

            budget = float(request.form.get('budget'))

            start_date = request.form.get('start_date')



            # Fetch itineraries within the specified budget for the chosen
destination

            query = """

                SELECT Itinerary_Id, Activities, Budget, No_of_Dates

                FROM Itinerary

                WHERE Destination_Id = %s AND Budget <= %s

            """

            cursor.execute(query, (selected_destination, budget))

            itineraries = cursor.fetchall()



            # Show itineraries selection form

            show_form = False  # Hide the initial form

            show_itineraries_form = True  # Show itineraries selection form

            return render_template('details.html',
```

```
                                    section=section,
                                    section_title=section_title,
                                    section_content=section_content,
                                    destinations=destinations,
                                    show_form=show_form,
                                    show_itineraries_form=show_itineraries_form,
                                    itineraries=itineraries,
                                    start_date=start_date)


        elif 'submit_itineraries' in request.form:
            # Second form submission with selected itineraries
            start_date = request.form.get('start_date')
            selected_itinerary_ids = request.form.getlist('selected_itinerary')
            selected_destination = request.form.get('destination_id')
            total_budget = 0
            total_days=0
            combined_activities = []
            itinerary_start_date = datetime.strptime(start_date, '%Y-%m-%d')


            for itinerary_id in selected_itinerary_ids:
                cursor.execute("SELECT Activities, Budget, No_of_Dates FROM
Itinerary WHERE Itinerary_Id = %s", (itinerary_id,))
                itinerary = cursor.fetchone()
```

```
                activities, budget, no_of_days = itinerary

                total_budget += budget

                total_days += no_of_days

                combined_activities.append(activities)

                end_date = itinerary_start_date + timedelta(days=no_of_days)

                selected_itineraries.append((activities,
itinerary_start_date.date(), end_date.date()))

                itinerary_start_date = end_date  # Update start date for next
itinerary



            # Concatenate activities into a single string

            combined_activities_str = '; '.join(combined_activities)



            # Insert combined itinerary into the Itinerary table

            insert_query = """

                INSERT INTO Itinerary (Destination_Id, User_Id, Start_Date,
No_of_Dates, Budget, Activities)

                VALUES (%s, %s, %s, %s, %s, %s)

            """

            cursor.execute(insert_query, (selected_destination, user_id,
start_date, total_days, total_budget, combined_activities_str))

            conn.commit()



            # Display chosen itineraries
```

```
            return render_template('chosen_itineraries.html',
itineraries=selected_itineraries)



    return render_template('details.html',
                           section=section,
                           section_title=section_title,
                           section_content=section_content,
                           destinations=destinations,
                           show_form=show_form,
                           show_itineraries_form=show_itineraries_form,
                           itineraries=itineraries)
```

Choose Your Destination

Select a destination and input your budget.

Select Itineraries within Your Budget

- [ ] Eiffel Tower visit, Louvre Museum, Seine River cruise - Budget: $2000.00 - Days: 6
- [ ] Disneyland Paris, Palace of Versailles - Budget: $2500.00 - Days: 6

Select Itineraries

Back to Home

© 2024 My Website

Selected Itineraries

Your Selected Itineraries with Dates

- Eiffel Tower visit, Louvre Museum, Seine River cruise - Start Date: 2005-11-17 - End Date: 2005-11-23
- Disneyland Paris, Palace of Versailles - Start Date: 2005-11-23 - End Date: 2005-11-29

Back to Home

© 2024 My Website

28	NULL	5	2004-11-17	12	4500.00	Eiffel Tower visit, Louvre Museum, Seine River c…	
29	NULL	5	2005-11-17	12	4500.00	Eiffel Tower visit, Louvre Museum, Seine River c…	
30	NULL	8	2024-11-17	12	4500.00	Eiffel Tower visit, Louvre Museum, Seine River c…	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	

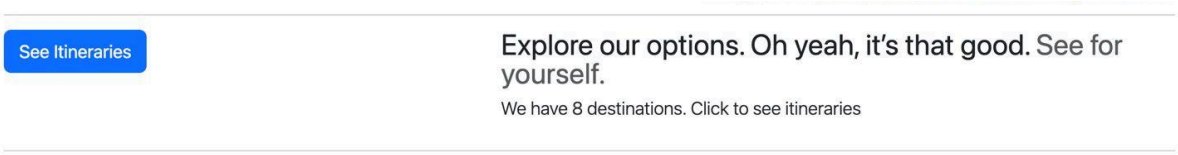
8. QUERIES (JOIN QUERY, AGGREGATE FUNCTION QUERIES AND NESTED QUERY)

JOIN:

```
def get_itineraries_with_destinations():

  """Run a join query to get itineraries with destination names."""

   connection = create_connection()

   cursor = connection.cursor(dictionary=True)

   query = """

       SELECT i.Itinerary_Id, i.Start_Date, i.Budget, d.Name AS
Destination_Name

       FROM Itinerary i

       JOIN Destination d ON i.Destination_Id = d.Destination_Id

   """

   cursor.execute(query)

   results = cursor.fetchall()

   connection.close()

   return results
```

AGGREGATE QUERY:

```
conn = create_connection()

    cursor = conn.cursor()

    cursor.execute('select count(*) from destination')

    data={"number":cursor.fetchone()[0]}
```



NESTED QUERY:

```
def get_users_with_high_budget_itineraries():
    """Run a nested query to get users with itineraries above the average
budget."""
    connection = create_connection()
    cursor = connection.cursor(dictionary=True)
    query = """
        SELECT DISTINCT u.User_Id, u.First_Name, u.Last_Name
        FROM User u
        JOIN Booking b ON u.User_Id = b.User_Id
        WHERE b.Itinerary_Id IN (
            SELECT Itinerary_Id
            FROM Itinerary
            WHERE Budget > (SELECT AVG(Budget) FROM Itinerary)
        )
    """
    cursor.execute(query)
    results = cursor.fetchall()
    connection.close()
    return results
```

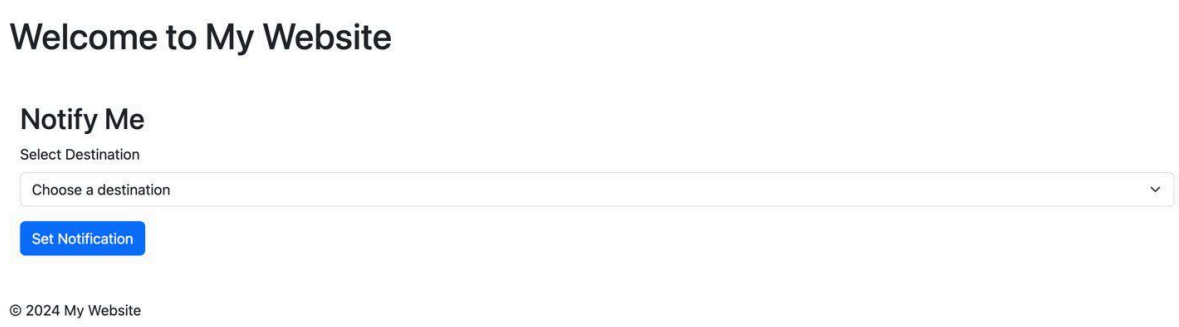
9. STORED PROCEDURE, FUNCTIONS AND TRIGGERS

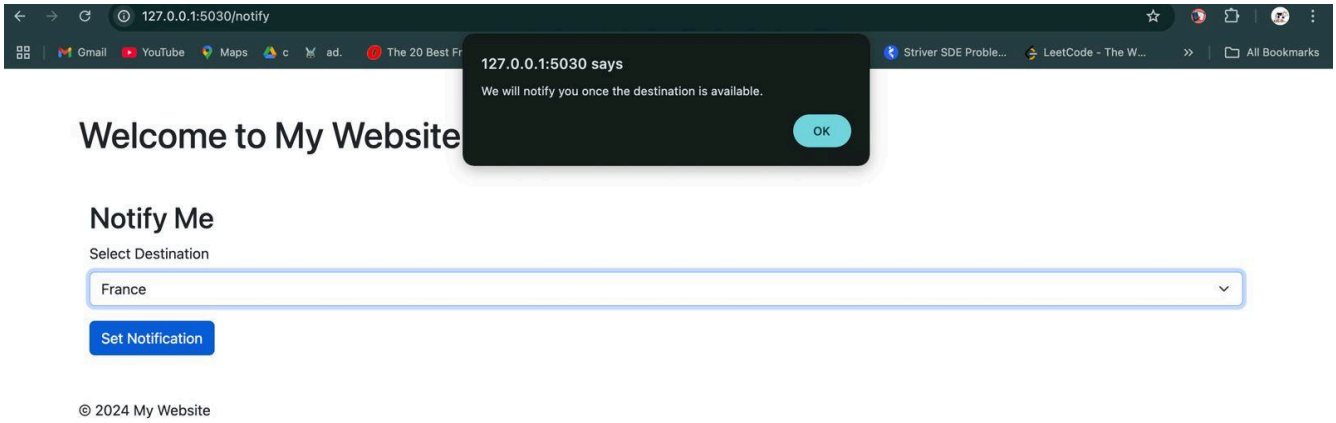
TRIGGER: Trigger is activated when availability of a destination is changed from unavailable(0) to available(1) by the admin, a notification is sent to user

```
DELIMITER $$

CREATE TRIGGER check_destination_availability
AFTER UPDATE ON Destination
FOR EACH ROW
BEGIN
   IF NEW.Availability = 1 AND OLD.Availability = 0 THEN
       INSERT INTO Notification (User_Id, Timestamp, Type, Message)
       SELECT
           up.User_Id,
           NOW(),
           'Destination Available',
           CONCAT('Your preferred destination ', NEW.Name, ' is now
available!')
       FROM
           UserPreferences up
       WHERE
           up.Preferences = NEW.Destination_Id;
   END IF;
END $$

DELIMITER ;
```

```
@app.route('/notify', methods=['GET', 'POST'])
def notify():
   if 'user_id' not in session:
       return redirect(url_for('login'))

   conn = create_connection()
   cursor = conn.cursor(dictionary=True)

   if request.method == 'POST':
```

```
        destination_id = request.form.get('destination')
        user_id = session['user_id']

        # Update UserPreferences with user's chosen destination
        cursor.execute("""
            INSERT INTO UserPreferences (User_Id, Preferences)
            VALUES (%s, %s)
            ON DUPLICATE KEY UPDATE Preferences = %s
        """, (user_id, destination_id, destination_id))
        conn.commit()

        flash('Your preference has been saved. You will be notified when this
destination becomes available.', 'success')
        return redirect(url_for('home'))

    # Fetch available destinations to display in the form
    cursor.execute("SELECT Destination_Id, Name FROM Destination")
    destinations = cursor.fetchall()

    conn.close()
    return render_template('notify.html', destinations=destinations)
```





User preference updated:

User_Id	Preferences
5	1
8	1
8	1
8	1
8	1
8	2
8	1
8	4
8	3
8	3
8	3
5	6
5	4
5	4
8	4
8	4
8	4
8	1
8	1
8	9
8	9
8	9
8	9
8	9
8	9
8	9
8	9

Notification updated:

Notification...	User_Id	Timestamp	Type	Message
1	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
2	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
3	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
4	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
5	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
6	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
7	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
8	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
9	8	2024-11-13 00:42:52	Destination Available	Your preferred destination bangla is now availab...
16	8	2024-11-13 00:49:45	Destination Available	Your preferred destination bangla is now availab...
17	8	2024-11-13 00:49:45	Destination Available	Your preferred destination bangla is now availab...
18	8	2024-11-13 00:49:45	Destination Available	Your preferred destination bangla is now availab...
19	8	2024-11-13 00:49:45	Destination Available	Your preferred destination bangla is now availab...
20	8	2024-11-13 00:49:45	Destination Available	Your preferred destination bangla is now availab...
21	8	2024-11-13 00:49:45	Destination Available	Your preferred destination bangla is now availab...
22	8	2024-11-13 00:49:45	Destination Available	Your preferred destination bangla is now availab...

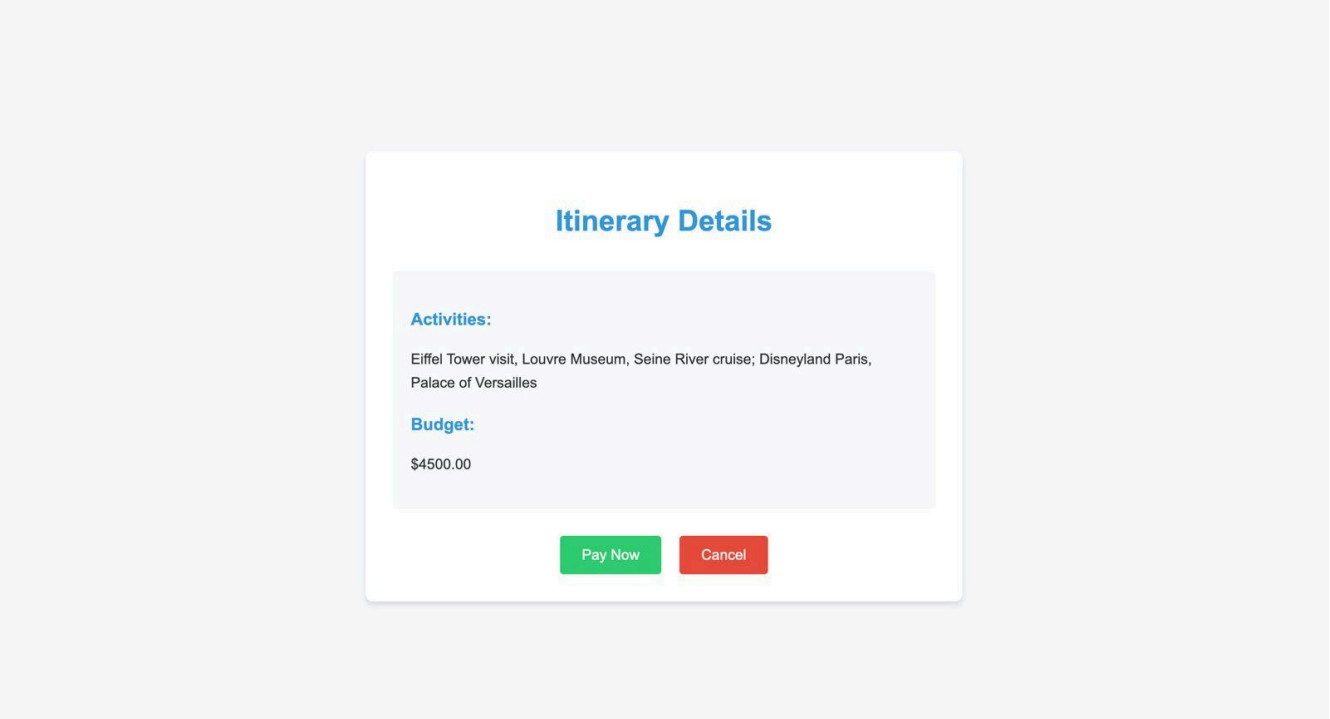
PROCEDURE: Stores booking confirmation/payment in booking table

```
DELIMITER //

CREATE PROCEDURE AddBooking(IN itinerary_id INT,
   IN user_id INT,
   IN payment_status VARCHAR(50),
   IN booking_status VARCHAR(50),
   IN booking_date DATE
)
BEGIN
   DECLARE payment_info TEXT DEFAULT 'Paid through portal';

   -- Insert booking data into the Booking table
   INSERT INTO Booking (Itinerary_Id, User_Id, Payment_Info, Payment_Status,
Booking_Status, Booking_Date)
   VALUES (itinerary_id, user_id, payment_info, payment_status, booking_status,
booking_date);
END ;

DELIMITER //
```

```
@app.route('/view_details', methods=['GET', 'POST'])
def view_details():
   user_id = session.get('user_id')  # Assume user_id is stored in session
after login
```

```
    print(f"User ID: {user_id}")  # Debug: Check if user ID is retrieved from
session

    if user_id:
        conn = create_connection()
        cursor = conn.cursor()

        # Query to fetch the last added itinerary details for the logged-in user
        query = """
            SELECT Itinerary_Id, Activities, Budget
            FROM Itinerary
            WHERE User_Id = %s
            ORDER BY Itinerary_Id DESC
            LIMIT 1
        """
        print("Executing itinerary fetch query...")  # Debug
        cursor.execute(query, (user_id,))
        result = cursor.fetchone()
        print(f"Query Result: {result}")  # Debug: Check if the itinerary
details are fetched

        if result:
            itinerary_id, activities, budget = result
            print(f"Itinerary ID: {itinerary_id}, Activities: {activities},
Budget: {budget}")  # Debug

            # Handle "Pay Now" or "Cancel" action
            if request.method == 'POST':
                payment_confirmation = request.form.get('payment_confirmation')
                print(f"Payment Confirmation: {payment_confirmation}")  # Debug

                # Determine booking status and payment status
                if payment_confirmation == 'confirmed':
                    payment_status = "Paid"
                    booking_status = "Confirmed"
                    flash("Payment confirmed. Your booking is successful.")
                else:
                    payment_status = "Not Paid"
```

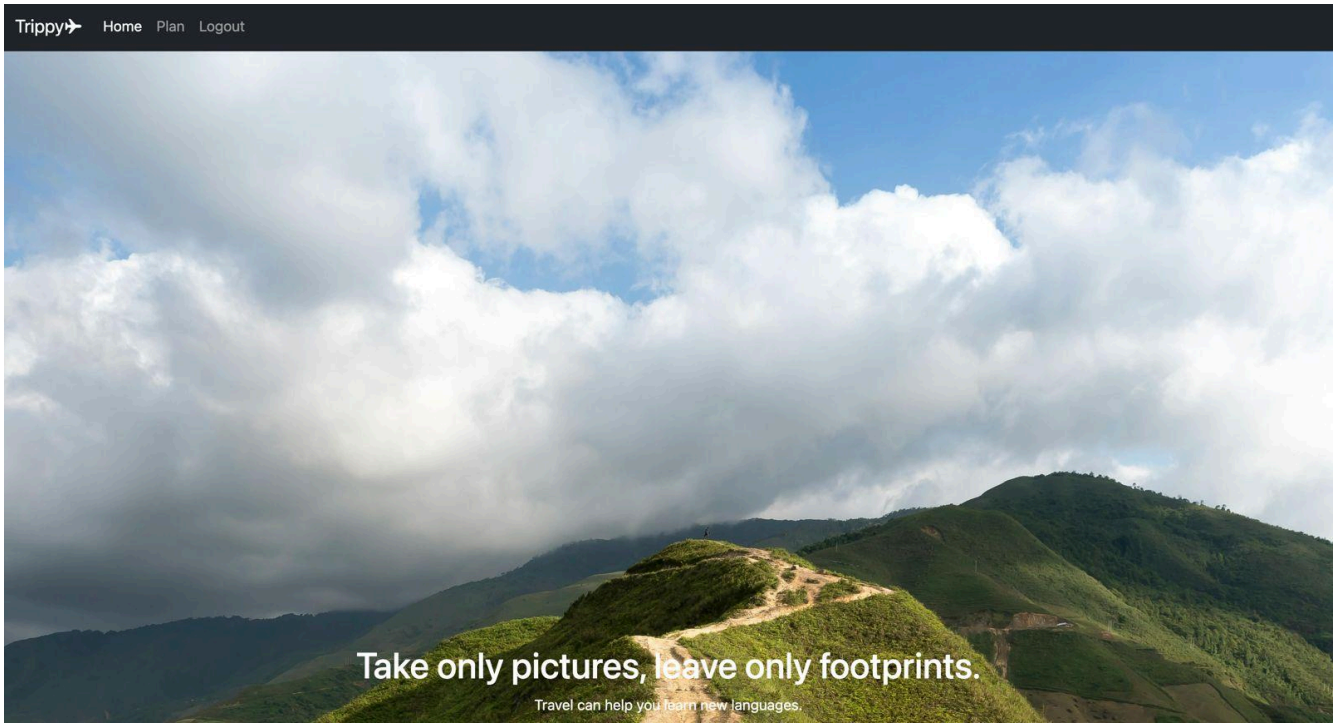
```
                    booking_status = "Unsuccessful"
                    flash("Payment was canceled. Your booking was not
completed.")

                booking_date = datetime.now().date()
                print(f"Calling stored procedure AddBooking with parameters:
itinerary_id={itinerary_id}, user_id={user_id},
payment_status={payment_status}, booking_status={booking_status},
booking_date={booking_date}")  # Debug

                # Call the stored procedure to insert booking data
                try:
                    cursor.callproc('AddBooking', [itinerary_id, user_id,
payment_status, booking_status, booking_date])
                    conn.commit()
                    print("Stored procedure executed successfully")  # Debug
                except Exception as e:
                    print(f"Error calling stored procedure: {e}")  # Debug:
Print any errors
                    flash("There was an error processing your booking. Please
try again.")

                return redirect(url_for('view_details'))

            conn.close()
            return render_template('view_details.html', activities=activities,
budget=budget)
        else:
            conn.close()
            print("No itinerary found for user")  # Debug
            return render_template('view_details.html', error="No itinerary
found.")
    else:
        print("User not logged in, redirecting to login page")  # Debug
        return redirect(url_for('login'))
```

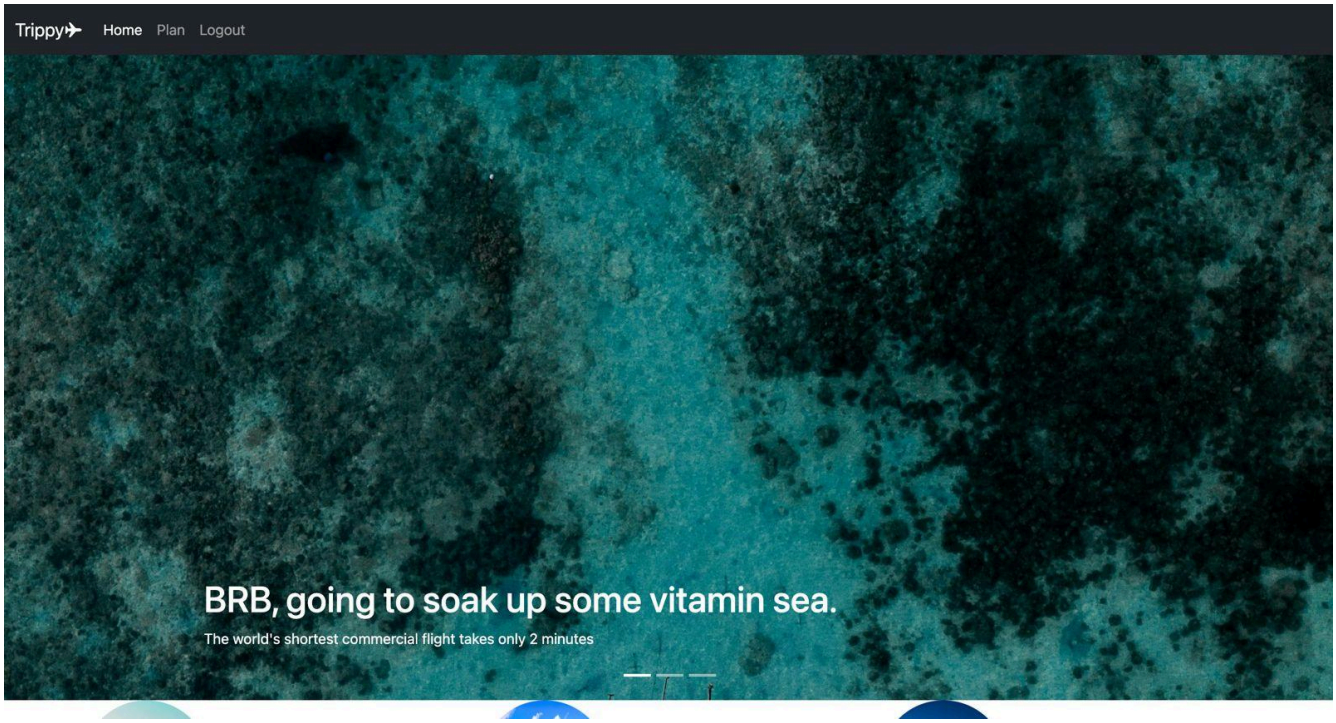


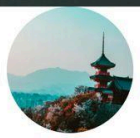
SELECT * FROM BOOKING;

Booking_Id	Itinerary_Id	User_Id	Payment_Info	Payment_Status	Booking_Status	Booking_Date
43	22	5	Paid through portal	Paid	Confirmed	2024-11-13
44	26	7	Paid through portal	Paid	Confirmed	2024-11-13
45	28	5	Paid through portal	Paid	Confirmed	2024-11-13
46	28	5	Paid through portal	Not Paid	Unsuccessful	2024-11-13
47	30	8	Paid through portal	Paid	Confirmed	2024-11-13
48	29	5	Paid through portal	Paid	Confirmed	2024-11-14
NULL	NULL	NULL	NULL	NULL	NULL	NULL

10. FRONT END DEVELOPMENT (FUNCTIONALITIES/FEATURES OF THE APPLICATION)







Plan Trip

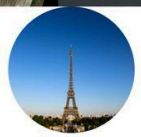
Some representative placeholder content for the three columns of text below the carousel. This is the first column.

View details »

Transport

Another exciting bit of representative placeholder content. This time, we've moved on to the second column.

View details »



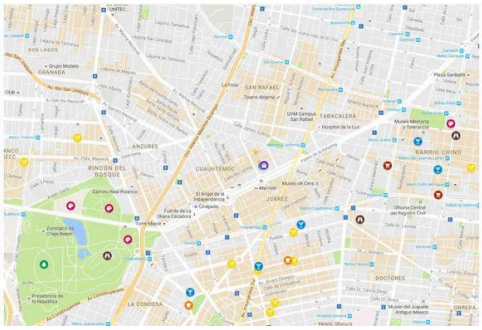
Notify me

Get notified when your preferred destinations become available.

Set Notifications

Maps. It'll blow your mind.

Explore the most amazing places on the globe



See Itineraries

Explore our options. Oh yeah, it's that good. See for yourself.

We have 8 destinations. Click to see itineraries

Explore Local. Checkmate.

And yes, this is the last block of representative placeholder content. Again, not really intended to be actually read, simply here to give you a better view of what this would look like with some actual content. Your content.

Back to top

REFERENCES/BIBLIOGRAPHY

- https://dev.mysql.com/doc/
- https://docs.python.org/
- https://dev.mysql.com/doc/connector-python/en/
- https://flask.palletsprojects.com/en/stable/
- https://github.com/Maitri-Shekhda/Travel-planner-dbms

APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Definitions:

● User Authentication: The process of verifying the identity of a user to ensure secure access to the system.
● Stored Procedure: Stored procedures are a set of SQL statements that are stored in a database and can be executed as a single unit.
● Trigger: A database mechanism that automatically executes a predefined action when a specific database event occurs.

Acronyms:

● SQL: Structured Query Language
● DBMS: Database Management System
● CRUD: Create, Read, Update, Delete Operations