Tourism Management System Database Design Document

V 3.0

Submitted by:  
Muhammad Yasir

### Department of Computer Sciences

### Namal University Mianwali, Pakistan

**REVISION HISTORY**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Approved by** |
| 16/06/25 | V.3.0 | GUI implementation, allows users to perform operations, and determining future works and conclusion. | Miss Asiya Batool |
| 02/06/25 | V.2.0 | Finalized logical database design, functional dependencies, and normalization analysis. | Miss Asiya Batool |
| 18/04/2025 | V 1.0 | Initial submission with base entities, ERD, and data dictionary. | Miss Asiya Batool |

**TABLE OF CONTENTS**

[CHAPTER 1: PROJECT OVERVIEW 6](#_bookmark0)

* 1. [INTRODUCTION 6](#_bookmark1)
  2. [PROBLEM STATEMENT 6](#_bookmark2)
  3. [PROJECT OBJECTIVES: 7](#_bookmark3)
  4. [DOCUMENT OBJECTIVES: 7](#_bookmark4)

[CHAPTER 2: DETAILED DATABASE DESIGN 8](#_bookmark5)

* 1. [ENTITIES 8](#_bookmark6)
  2. [DATA DICTIONARY 8](#_bookmark7)
     1. [.1 User 8](#_bookmark8)

1. [.2 Admin 8](#_bookmark9)
2. [.3 Destination 9](#_bookmark10)
3. [.4 Booking 9](#_bookmark11)
4. [.5 Visit 9](#_bookmark12)
5. [.6 Favourite: 9](#_bookmark13)
6. [.7 9](#_bookmark14)
7. [.8 Review: 10](#_bookmark15)
8. [.9 Restaurant 10](#_bookmark16)
9. [.10 Hotel 10](#_bookmark17)
10. [.11 Transportation: 10](#_bookmark18)
11. [.12 Reservation 10](#_bookmark19)
12. [.13 Travel 11](#_bookmark20)
    1. [RELATIONSHIPS 11](#_bookmark21)
    2. [ENTITY RELATIONSHIP DIAGRAM: 13](#_bookmark22)
13. [: Logical DATABASE DESIGN 14](#_bookmark23)
    1. [RELATIONAL SCHEMA 14](#_bookmark24)
    2. [FUNCTIONAL DEPENDENCIES 16](#_bookmark25)
14. [.1 1. Relation: User 16](#_bookmark26)
15. [.2 2. Relation: Admin 16](#_bookmark27)
16. [.3 3. Relation: Destination 16](#_bookmark28)
17. [.4 4. Relation: Hotel 16](#_bookmark29)
18. [.5 5. Relation: Booking 16](#_bookmark30)
19. [.6 6. Relation: Restaurant 17](#_bookmark31)
20. [.7 7. Relation: Reservation 17](#_bookmark32)
21. [.8 8. Relation: Transportation 17](#_bookmark33)
22. [.9 9. Relation: Travel 17](#_bookmark34)
23. [.10 10. Relation: Visit 18](#_bookmark35)
24. [.11 11. Relation: Favourite 18](#_bookmark36)
25. [.12 12. Relation: Review 18](#_bookmark37)
    1. [NORMALIZATION 18](#_bookmark38)
       1. [.13 A. Partial Functional Dependencies 19](#_bookmark39)
26. [.14 B. Transitive Functional Dependencies 19](#_bookmark40)
    * 1. [Normalization Process 20](#_bookmark41)
         1. [.1 Step 1: First Normal Form (1NF) 20](#_bookmark42)
27. [.2 Step 2: Second Normal Form (2NF) 20](#_bookmark43)
28. [.3 Step 3: Third Normal Form (3NF) 20](#_bookmark44)
    * 1. [Final 3NF Relational Schema 21](#_bookmark45)
         1. [.1 1. User 21](#_bookmark46)
29. [.2 2. Admin 21](#_bookmark47)
30. [.3 3. Destination 21](#_bookmark48)
31. [.4 4. Hotel 21](#_bookmark49)
32. [.5 5. Booking 21](#_bookmark50)
33. [.6 6. Restaurant 21](#_bookmark51)
34. [.7 7. Reservation 21](#_bookmark52)
35. [.8 8. Transportation 21](#_bookmark53)
36. [.9 9. Travel 22](#_bookmark54)
37. [.10 10. Visit 22](#_bookmark55)
38. [.11 11. Favourite 22](#_bookmark56)
39. [.12 12. Review 22](#_bookmark57)
40. [: Physical DATABASE DESIGN 23](#_bookmark58)
    1. [STRUCTURE OF THE TABLES: 23](#_bookmark59)
    2. [DATA SAMPLES INSIDE TABLES: 26](#_bookmark60)

[QUERIES RESULTS: 30](#_bookmark61)

[chapter 5: Interface design 36](#_bookmark62)

* 1. [LANGUAGE/FRAMEWORK 36](#_bookmark63)
     1. [Technology Stack 36](#_bookmark64)
     2. [Advantages of the Chosen Stack 36](#_bookmark65)
  2. [DATABASE CONNECTIVITY 36](#_bookmark66)
     1. [Configuration 36](#_bookmark67)
     2. [Error Handling 36](#_bookmark68)
  3. [STORED PROCEDURES AND FUNCTIONS: 37](#_bookmark69)
     1. [Procedure 1: GetFullUserActivity 37](#_bookmark70)
     2. [Procedure 2: GetTopRatedDestinationsWithHotelOptions 38](#_bookmark71)
     3. [Function 1: GetTotalUserCos 38](#_bookmark72)
     4. [Function 2: GetHotelRevenue 38](#_bookmark73)
  4. [INTERFACES 39](#_bookmark74)
     1. [Authentication & User Management 39](#_bookmark75)
     2. [Main Navigation & Homepage 41](#_bookmark76)
     3. [Destination Features 42](#_bookmark77)
     4. [Hotel Management 43](#_bookmark78)
     5. [Restaurant Features 45](#_bookmark79)
     6. [Admin Interface 46](#_bookmark80)
     7. [Mobile Responsiveness 47](#_bookmark81)

[chapter 6: CONCLUSION 49](#_bookmark82)

* 1. [LESSONS LEARNED 49](#_bookmark83)
     1. [Technical Insights 49](#_bookmark84)
  2. [CHALLENGES AND SOLUTIONS: 49](#_bookmark85)
     1. [Database Design and Relationship Management 49](#_bookmark86)
     2. [User Authentication and Authorization 50](#_bookmark87)
     3. [Responsive and User-Friendly Frontend 50](#_bookmark88)
     4. [Error Handling and User Feedback 50](#_bookmark89)
     5. [Testing and Quality Assurance 51](#_bookmark90)
  3. [FUTURE WORK AND IMPROVEMENTS 51](#_bookmark91)
     1. [Mobile application development 51](#_bookmark92)
     2. [Payment gateway integration 51](#_bookmark93)
     3. [AI-powered recommendation system 51](#_bookmark94)
     4. [Technical Enhancements 51](#_bookmark95)
     5. [Performance optimization 51](#_bookmark96)
     6. [Enhanced security measures 51](#_bookmark97)
  4. [FINAL THOUGHTS 52](#_bookmark98)
     1. [Robust user management 52](#_bookmark99)
     2. [Secure authentication 52](#_bookmark100)
     3. [Efficient database operations 52](#_bookmark101)
     4. [Responsive user interface 52](#_bookmark102)

[7 REFERENCES 52](#_bookmark103)

* 1. [AI PROMPTS 53](#_bookmark104)

# CHAPTER 1: PROJECT OVERVIEW

## INTRODUCTION:

Tourism is a key driver of economic growth in Pakistan, playing an essential role in creating jobs, generating revenue, and preserving cultural heritage. Data from the Pakistan Tourism Development Corporation (PTDC) and the World Travel and Tourism Council (WTTC) shows that the sector has contributed around 5.9% to the country's total employment in recent years. Beyond its economic impact, tourism also promotes cultural exchange by encouraging visitors to discover Pakistan’s rich traditions, vibrant culture, and breathtaking landscapes.

However, many travelers face challenges due to the absence of a unified digital platform that offers reliable, real-time information on travel destinations, routes, accommodations, and local services. This lack of centralized information often results in confusion, poor planning, and a less enjoyable experience especially for international tourists unfamiliar with the country’s geography and infrastructure.

To tackle these issues, a Tourism Management System (TMS) is being proposed. This web- based platform would act as an all-in-one information hub, providing details about tourist spots, hotels, restaurants, transport services, emergency contacts, and user reviews. Built on sound database management principles, the TMS aims to streamline access to essential information, improve service delivery, and ultimately elevate the overall travel experience across Pakistan.

## PROBLEM STATEMENT:

Despite its breathtaking landscapes, deep-rooted history, and rich cultural heritage, Pakistan's tourism industry remains largely untapped. One of the major barriers to growth is the absence of a centralized digital platform that provides tourists with accurate, current, and trustworthy information. As a result, many potential visitors struggle to find reliable details about destinations, routes, accommodations, dining options, and transportation.

Without a structured and dependable source of information, trip planning often becomes fragmented and confusing. Travelers are left to make decisions based on incomplete or outdated data, which can erode trust in the tourism experience. In critical situations, the lack of verified emergency contacts and support only adds to the frustration particularly for international tourists unfamiliar with the region.

This shortfall clearly underscores the need for a robust and unified tourism information system one that consolidates all key travel services into a single, accessible platform. Such a system must allow for:

* + - Easy retrieval of travel information,
    - User-generated content such as reviews and ratings,
    - Booking assistance, and
    - Emergency contact accessibility.

The proposed Tourism Management System (TMS) aims to fill this gap. Designed to be scalable, centralized, and user-friendly, TMS will streamline the entire travel experience in

Pakistan making it safer, more efficient, and far more enjoyable for both local and international tourists.

## PROJECT OBJECTIVES:

The Tourism Management System (TMS) database is designed with several key objectives in mind:

* + - To offer a centralized and organized structure for storing information about tourist spots, accommodations, transportation options, and user profiles.
    - To ensure the smooth and consistent retrieval, updating, and management of all tourism- related data within the system.
    - To accommodate user-generated content, including reviews and ratings for destinations, hotels, and dining establishments.
    - To maintain a reliable directory of emergency contacts such as hospitals, local authorities, and tourist assistance services to help ensure traveler safety.

## DOCUMENT OBJECTIVES:

The objectives of this document are to:

* + - Provide a clear and structured outline of the database design for the Tourism Management System project.
    - Serve as a comprehensive reference for developers, database administrators, and stakeholders.
    - Detail the essential components and architecture of the database system.
    - Ensure consistency, accuracy, and maintainability throughout the database lifecycle.
    - Define the purpose, scope, and requirements of the database system.
    - Describe the logical design: detailing entities, relationships, and data flow within the system.
    - Describe the physical design: specifying how the database will be implemented, including tables, indexes, storage, and access methods.
    - Align the database design with the functional requirements of the web-based system ensuring the database supports all necessary features and operations of the website.
    - Provide a solid foundation for future development, testing, and maintenance.
    - Facilitate effective communication among all stakeholders, including both technical and non-technical users.
    - Outline the structure and organization of this documentation to guide readers clearly through the project details.

# CHAPTER 2: DETAILED DATABASE DESIGN

## ENTITIES:

|  |  |  |
| --- | --- | --- |
| **Sr. No** | **Entity Name** | **Description** |
| 01 | User | A user is an individual who have registered or created accounts within the system. |
| 02 | Admin | An admin is a user with elevated privileges who manages the system, users, and content. |
| 03 | Destination | Represents various tourist destinations available in the system, providing  details and services. |
| 04 | Booking | Represents a user's booking for a destination, including details like travel date and status. |
| 05 | Visit | Tracks the visits made by users to different destinations. |
| 06 | Favorite | Stores the user's favorite destinations for easy access |
| 07 | Review | Represents user feedback on destinations, including ratings and comments. |
| 08 | Restaurant | Represents dining options available at destinations. |
| 09 | Hotel | Represents accommodation options available for users. |
| 10 | Transportation | Represents various transportation options available to users. |
| 11 | Reservation | Represents reservations made by users for dining, hotels, and transportation. |
| 12 | Travel | Represents travel itineraries that include details on destinations, transportation, and activities. |

## DATA DICTIONARY:

* + 1. **User:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| UserID | INT | PK, NOT NULL | Unique identifier for each user |
| Name | VARCHAR(50) | NOT NULL | Name of the user |
| Email | VARCHAR(100) | UNIQUE, NOT NULL | Email address of the user |
| Password | VARCHAR(60) | NOT NULL | Password for user authentication |
| ContactNo | VARCHAR(15) |  | User's contact number |

* + 1. **Admin:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| AdminID | INT | PK, NOT NULL | Unique identifier for each admin |
| Name | VARCHAR(50) | NOT NULL | Name of the admin |
| Email | VARCHAR(100) | UNIQUE, NOT NULL | Email address of the admin |
| Password | VARCHAR(60) | NOT NULL | Admin login password |

* + 1. **Destination:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| DestinationID | INT | PK, NOT NULL | Unique identifier for each destination |
| Name | VARCHAR(100) | NOT NULL | Name part of the destination address |
| Description | TEXT |  | Brief description of the destination |
| Street | VARCHAR(50) | NOT NULL | Street part of the address |
| City | VARCHAR(50) | NOT NULL | City part of the address |
| category | VARCHAR(100) |  | Category/type of the destination (e.g.,  historical, natural). |
| Image\_url | VARCHAR(500) | NULL | Image url of that  destination will be here. |

* + 1. **Booking:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| BookingID | INT | PK, NOT NULL | Unique booking identifier |
| CheckInDate | DATE | NOT NULL | Date of check-in |
| CheckOutDate | DATE | NOT NULL | Date of check-out |
| RoomType | VARCHAR(30) |  | Type of room booked such as (e.g."SINGLE", "DOUBLE",  "SUITE") |
| Cost | DECIMAL(8,2) |  | Total cost of the booking |
| BookingDate | DATETIME | Current\_timestamp | Date when the  booking was made |

* + 1. **Visit:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| DestinationID | INT | PK, NOT NULL | Visited destination ID |
| UserID | INT | PK, NOT NULL | User who visited |
| VisitDate | DATE |  | Date of the visit |

* + 1. **Favourite:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| DestinationID | INT | PK, NOT NULL | Favorited destination |
| UserID | INT | PK, NOT NULL | User who favorited |
| FavouriteDate | DATE |  | Date when the  destination was marked favorite |

* + 1. **Review:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| ReviewID | INT | PK | Unique review ID |
| Rating | TINYINT | NOT NULL | Rating out of 5 |
| Comment | TEXT |  | User's review comment |
| ReviewType | VARCHAR(50) |  | Type of the review  (e.g., hotel, transport) |

* + 1. **Restaurant:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| RestaurantID | INT | PK | Unique ID of the restaurant |
| Name | VARCHAR(50) | NOT NULL | Restaurant name part of address |
| Street | VARCHAR(100) | NOT NULL | Street address |
| City | VARCHAR(30) | NOT NULL | City (part of address) |
| Cuisine | VARCHAR(30) |  | Type of cuisine such as Pakistani , Italian  etc |
| Image\_url | VARCHAR(500) | NULL | url of image of restaurant. |

* + 1. **Hotel:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| HotelID | INT | PK | Hotel unique identifier |
| Name | VARCHAR(50) | NOT NULL | Hotel name as part of address |
| Street | VARCHAR(100) | NOT NULL | Street address |
| City | VARCHAR(30) | NOT NULL | City as part of address |
| Facilities | TEXT |  | Facilities available at the hotel |
| Image\_url | VARCHAR(500) | NULL | url of image of hotel. |

* + 1. **Transportation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| TransportationID | INT | PK | Unique identifier for transport |
| Type | VARCHAR(30) |  | Type of transport (e.g., taxi, bus) |
| Provider | VARCHAR(50) |  | Transport service provider |

* + 1. **Reservation:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| ReservationID | INT | PK | Unique reservation ID |
| ReservationDate | DATE | NOT NULL | Date of reservation |
| ReservationTime | TIME | NOT NULL | Time of reservation |
| NumberOfGuests | INT |  | Number of guests in reservation |

|  |  |  |  |
| --- | --- | --- | --- |
| TableNumber | VARCHAR(20) |  | Assigned table number |

* + 1. **Travel:**

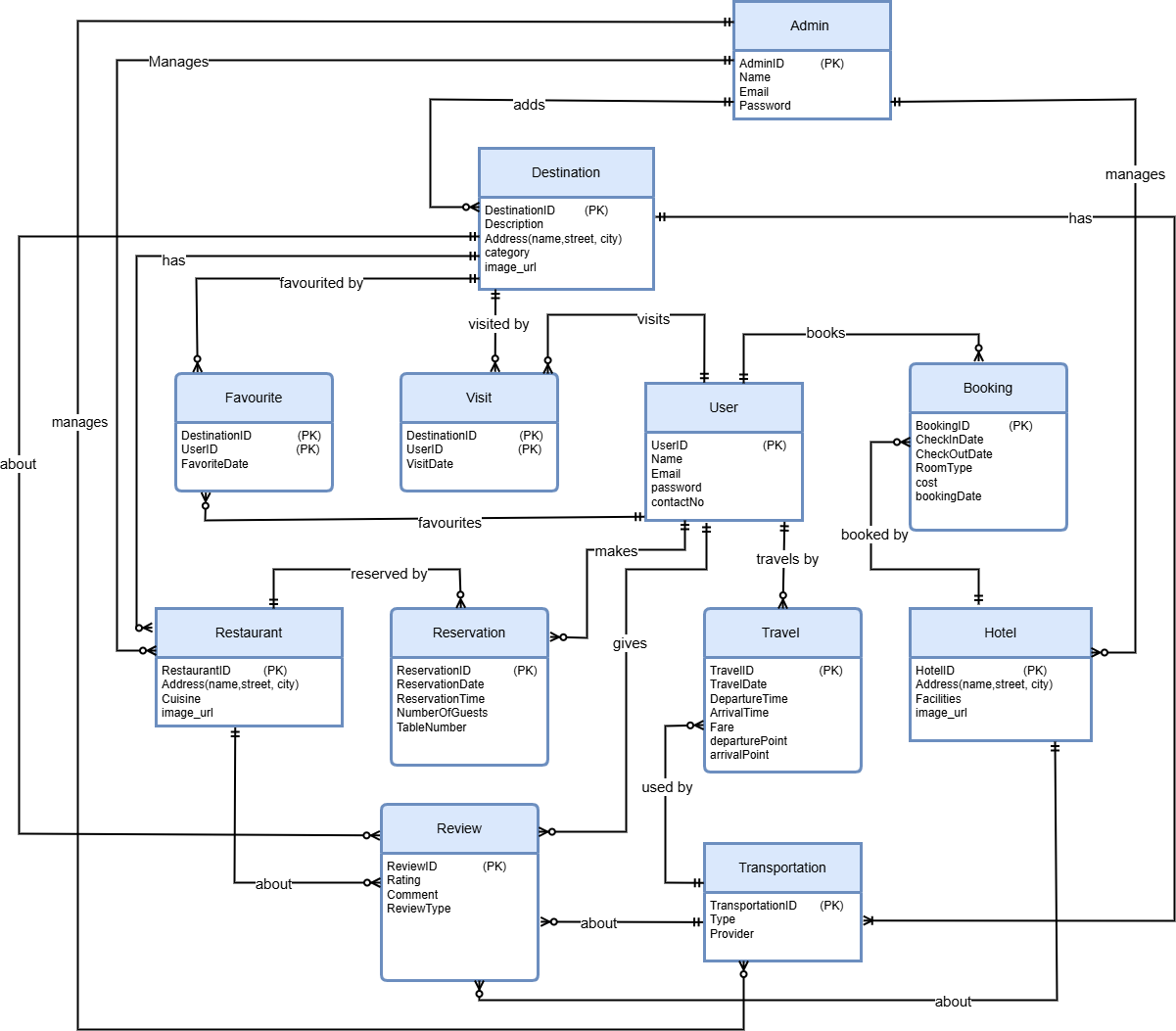
|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute Name** | **Data Type** | **Constraints** | **Description** |
| TravelID | INT | PK | Travel record ID |
| TravelDate | DATE | NOT NULL | Date of travel |
| DepartureTime | TIME | NOT NULL | Time of departure |
| Fare | DECIMAL(8,2) | NOT NULL, Fare => 0 | Fare of travel |
| DeparturePoint | VARCHAR(50) | NOT NULL | Point of departure |
| ArrivalPoint | VARCHAR(50) | NOT NULL | Arrival location |
| ArrivalTIme | TIME | NOT NULL | Time of Arrival |

## RELATIONSHIPS:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Participating Entities** | **Relation** | **Business Rule** |
| 01 | Admin, Destination | Admin manages Destination | An admin can manage multiple destinations; each destination is managed by one admin. |
| 02 | Admin, Hotel | Admin manages Hotel | An admin can manage multiple hotels; each hotel is managed by one admin. |
| 03 | Admin, Restaurant | Admin manages Restaurant | An admin can manage multiple restaurants; each restaurant is managed by one admin. |
| 04 | Admin, Transportation | Admin manages Transportation | An admin can manage multiple transportation options; each transport is managed by one admin. |
| 05 | Destination, Favourite | Destination has Favourite | A destination can be favourited by multiple  users; each favourite links to one destination. |
| 06 | Destination, Visit | Destination has Visit | A destination can be visited by many users; each visit links to one destination. |
| 07 | Destination, Review | Destination is reviewed in Review | A destination may be reviewed in many reviews; each review refers to one destination. |

|  |  |  |  |
| --- | --- | --- | --- |
| 08 | Destination, Restaurant | Destination has Restaurant | A destination can have multiple restaurants; each restaurant belongs to one destination. |
| 09 | Destination, Transportation | Destination has Transportation | A destination can have multiple transportation options; each transport is linked to one destination. |
| 10 | Favourite, User | Favourite is by User | Each favourite is created by one user; a user may  have many favourites. |
| 11 | Visit, User | Visit is by User | Each visit is made by one user; a user can have many visits. |
| 12 | User, Booking | User books Booking | A user can make multiple bookings; each booking is linked to one user. |
| 13 | User, Travel | User travels via Travel | A user can travel multiple times; each travel entry belongs to one user. |
| 14 | User, Review | User gives Review | A user can write multiple reviews; each review is  written by one user. |
| 15 | User, Reservation | User makes Reservation | A user can make multiple reservations; each reservation is linked to one user. |
| 16 | Booking, Hotel | Booking links to Hotel | A booking refers to one hotel; a hotel may have multiple bookings. |
| 17 | Restaurant, Reservation | Restaurant has Reservation | A restaurant can have multiple reservations; each reservation is for one  restaurant. |
| 18 | Review, Restaurant | Review about Restaurant | A review can be about a restaurant; a restaurant can have many reviews. |
| 19 | Review, Transportation | Review about Transportation | A review can be about a transport option; a transport option can be reviewed many times. |
| 20 | Review, Destination | Review about Destination | A review can be about a destination; a destination can have multiple reviews. |
| 21 | Travel, Transportation | Travel uses Transportation | Each travel uses one transportation option; a  transport option can be used in many travels. |

## ENTITY RELATIONSHIP DIAGRAM:

****

*Figure 1: ER Diagram of Tourism Management System*

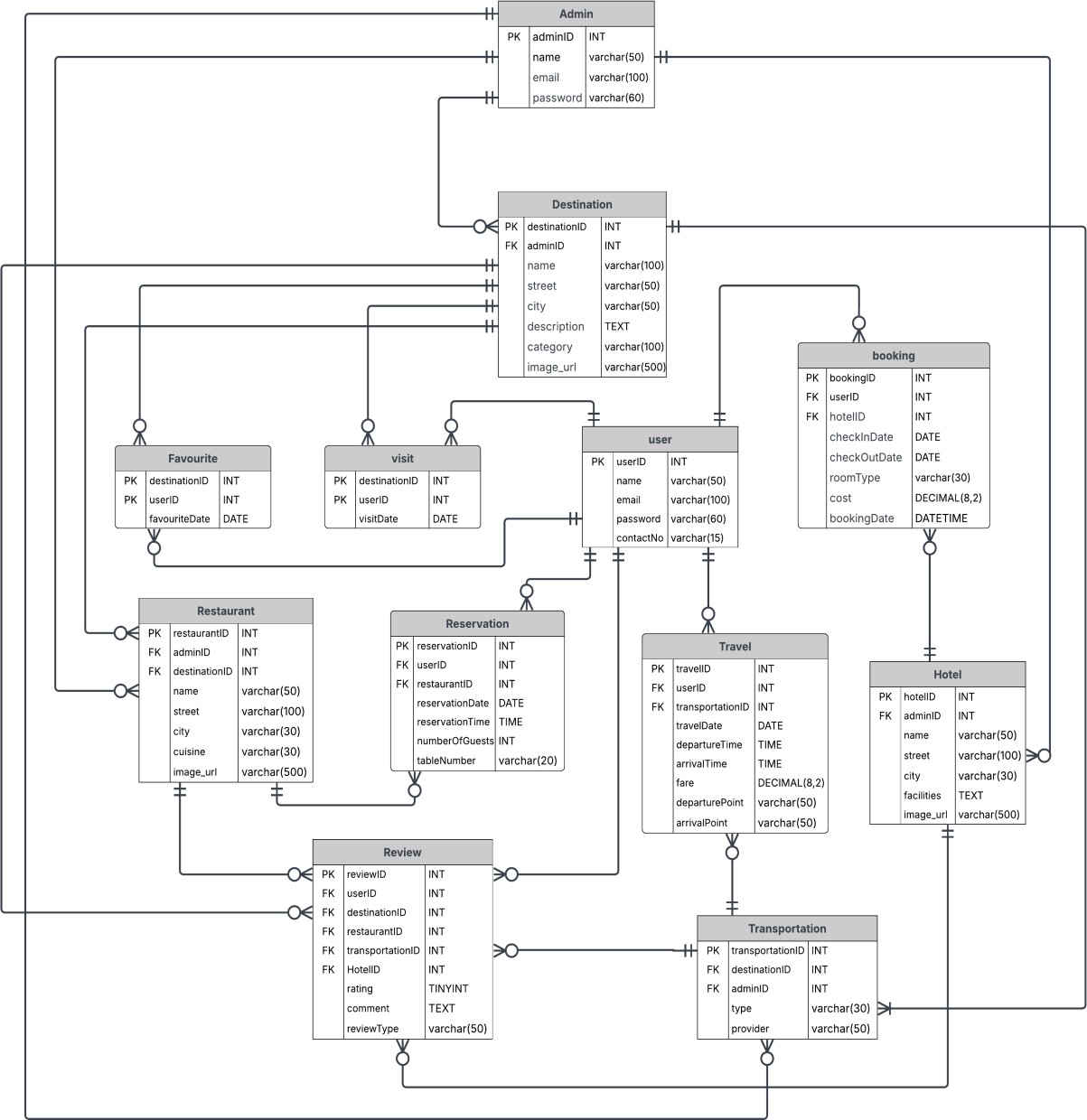
# CHAPTER 3: LOGICAL DATABASE DESIGN

## RELATIONAL SCHEMA:

The transformation from the conceptual design (ERD) to the logical design (relational schema) was carried out by identifying all entities, attributes, primary keys, and the relationships among them. Foreign keys were defined to represent these relationships. Composite keys were used where necessary, particularly in bridge tables such as Favourite and Visit, which handle many-to-many relationships.

##### Principles Applied:

1. **Entities to Tables:** Each entity in the ERD is mapped to a relation (table) in the relational schema.
2. **Attributes to Columns:** Each attribute of an entity becomes a column in the corresponding table. The data type for each column is chosen to best represent the attribute's nature (e.g., INT for identifiers, VARCHAR for names, DATE for dates).
3. **Primary Keys (PK):** A unique identifier for each entity (often an ID attribute) is designated as the primary key for the corresponding table.
4. **One-to-Many (1:M) Relationships:** The primary key of the table on the "one" side of the relationship is included as a foreign key (FK) in the table on the "many" side.
5. **Many-to-Many (M:N) Relationships**: These are resolved by creating an associative (or bridge/junction) table. This new table contains foreign keys referencing the primary keys of the two entities involved in the M:N relationship. The combination of these foreign keys often forms the primary key of the associative table, or a separate surrogate key is introduced.
6. **One-to-One (1:1) Relationships:** Can be implemented by placing the primary key of one entity as a foreign key in the other. The choice often depends on participation constraints (mandatory vs. optional) or can be merged if conceptually very close. A unique constraint is often placed on the foreign key in the referencing table.



*Figure 2: Relational Schema of Tourism ManagementSystem*

## FUNCTIONAL DEPENDENCIES:

#### Relation: User

* + **Primary Key (PK)**: userID
  + **Attributes**: userID, name, email, password, contactNo

##### Functional Dependencies:

* + - userID → name
    - userID → email
    - userID → password
    - userID → contactNo
    - Combined: userID → name, email, password, contactNo

##### Example:

If userID = 101, then name = "Ali Khan", email = ["ali@gmail.com"](mailto:ali@gmail.com), and contactNo = "03001234567" — all values are determined solely by userID.

#### Relation: Admin

* + **Primary Key (PK)**: adminID
  + **Attributes**: adminID, name, email, password

##### Functional Dependencies:

* + - adminID → name, email, password

##### Example*:*

If adminID = 201, then name = "Admin1", email = ["admin1@tms.com"](mailto:admin1@tms.com).

#### Relation: Destination

* + **Primary Key (PK)**: destinationID
  + **Attributes**: destinationID, adminID, name, description, street, city, category, image\_url

##### Functional Dependencies:

* + - destinationID → adminID, name, description, street, city, category, image\_url

##### Example:

If destinationID = 301, then it maps to "Hunza Valley", city = "Gilgit", adminID = 201.

#### Relation: Hotel

* + **Primary Key (PK)**: hotelID
  + **Attributes**: hotelID, adminID, name, street, city, facilities, image\_url

##### Functional Dependencies:

* + - hotelID → adminID, name, street, city, facilities, image\_url

##### Example*:*

If hotelID = 401, then it refers to "Hilltop Resort", adminID = 201.

#### Relation: Booking

* + **Primary Key (PK)**: bookingID
  + **Attributes**: bookingID, userID, hotelID, checkInDate, checkOutDate, roomType, cost, bookingDate

##### Functional Dependencies:

* + - bookingID → userID, hotelID, checkInDate, checkOutDate, roomType, cost,

bookingDate

##### Example:

bookingID = 501 refers to a booking by userID = 101 at hotelID = 401 from '2024- 06-01' to '2024-06-03'.

#### Relation: Restaurant

* + **Primary Key (PK)**: restaurantID
  + **Attributes**: restaurantID, adminID, destinationID, name, street, city, cuisine, image\_url

##### Functional Dependencies:

* + - restaurantID → adminID, destinationID, name, street, city, cuisine, image\_url

##### Example*:*

If restaurantID = 601, then name = "Desi Delight", destinationID = 301, and adminID

= 201.

#### Relation: Reservation

* + **Primary Key (PK)**: reservationID
  + **Attributes**: reservationID, userID, restaurantID, reservationDate, reservationTime, numberOfGuests, tableNumber

##### Functional Dependencies:

* + - reservationID → userID, restaurantID, reservationDate, reservationTime,

numberOfGuests, tableNumber

##### Example*:*

A reservation with reservationID = 701 is for userID = 101, restaurantID = 601 on

'2024-06-15' at '19:00'.

#### Relation: Transportation

* + **Primary Key (PK)**: transportationID
  + **Attributes**: transportationID, destinationID, adminID, type, provider

##### Functional Dependencies:

* + - transportationID → destinationID, adminID, type, provider
  + **Example*:***

If transportationID = 801, then it's a "Bus" provided by "Greenline" for destinationID

= 301.

#### Relation: Travel

* + **Primary Key (PK)**: travelID
  + **Attributes**: travelID, userID, transportationID, travelDate, departureTime, fare, departurePoint, arrivalPoint, arrivalTime

##### Functional Dependencies:

* + - travelID → userID, transportationID, travelDate, departureTime, fare,

departurePoint, arrivalPoint, arrivalTime

* + **Example*:***

If travelID = 901, then userID = 101, departure = "Lahore", arrival = "Islamabad", fare

= 1500.

#### Relation: Visit

* + **Primary Key (PK)**: Composite of userID, destinationID, visitDate
  + **Attributes**: userID, destinationID, visitDate

##### Functional Dependencies:

* + - (userID, destinationID, visitDate) → [composite PK only; no non-key attributes]

##### Example*:*

A row with userID = 101, destinationID = 301, and visitDate = '2024-06-01' uniquely identifies the visit.

#### Relation: Favourite

* + **Primary Key (PK)**: Composite of userID, destinationID
  + **Attributes**: userID, destinationID, favouriteDate

##### Functional Dependencies:

* + - (userID, destinationID) → favouriteDate

##### Example*:*

If userID = 101 favorited destinationID = 301, then favouriteDate = '2024-05-20'.

#### Relation: Review

* + **Primary Key (PK)**: reviewID
  + **Attributes**: reviewID, userID, destinationID, restaurantID, transportID, hotelID, rating, comment, reviewType

##### Functional Dependencies:

* + - reviewID → userID, destinationID, restaurantID, transportID, hotelID, rating, comment, reviewType

##### Example*:*

If reviewID = 1001, then it's a review by userID = 101 for hotelID = 401, with rating

= 4, reviewType = 'hotel'.

## NORMALIZATION:

Functional Dependencies (FDs) and identify any partial or transitive ones based on the original schema (before potential 3NF corrections).

#### Partial Functional Dependencies

Partial dependencies occur when a non-key attribute is functionally dependent on part of a composite primary key.

* + In tables with simple **(**single**-**attribute**)** primary keys such as userID, adminID, destinationID, hotelID, restaurantID, reservationID, travelID, transportationID, bookingID, reviewID, there cannot be partial dependencies by definition.

#### Considering Candidate Keys for visit:

* + Composite Primary Key: {userID, destinationID, visitDate}
  + userID, destinationID, visitDate → [record of visit]
  + No partial dependency: there are no non-key attributes other than the PK components.

#### Considering Candidate Keys for favourite:

* + Composite Primary Key: {userID, destinationID}
  + userID, destinationID → favouriteDate
  + favouriteDate is fully dependent on the entire composite key, not just userID or destinationID alone.

#### Transitive Functional Dependencies

Transitive dependencies occur when a non-key attribute depends on another non-key attribute, which in turn depends on the primary key (i.e., A → B → C, where A is PK, B and C are non-key).

#### Table: Review

* + **PK**: reviewID
  + **Attributes**: reviewID, userID, destinationID, restaurantID, transportID, hotelID, rating, comment, reviewType

##### Analysis:

* + - No transitive dependencies exist.
    - For example, reviewID → hotelID, but hotelID → hotelName is not in the review

table.

* + - No attribute in review is dependent on another non-key attribute inside this same table.

#### Table: Reservation

* + **PK**: reservationID

##### FDs:

* + - reservationID → restaurantID

restaurantID → name (but name is not included in reservation)

* + No transitive dependency exists within this table.

#### Table*:* Booking

* + **PK**: bookingID

##### FDs:

* + - bookingID → hotelID, hotelID → hotel.city
    - city is not stored in booking, so this is not a transitive dependency.
  + No transitive dependencies.

### Normalization Process

#### Step 1: First Normal Form (1NF)

* **Rule**: Each attribute must be atomic (indivisible), and each row must be unique (ensured via primary keys).

##### Assessment:

* + All attributes are atomic (e.g., email, city, cuisine, cost, etc.).
  + Each table has a primary key (either simple or composite).
* **Transformations**: None needed.
* **Status**: All tables are in 1NF**.**

#### Step 2: Second Normal Form (2NF)

* **Rule**: Must be in 1NF, and all non-key attributes must be fully dependent on the **entire**

primary key.

##### Assessment:

* + All single-key tables (e.g., user, admin, destination, hotel, restaurant,etc.)automatically satisfy 2NF.
  + Tables with composite keys (visit, favourite) have all attributes dependent on the entire composite key.
* **Transformations**: None needed.
* **Status**: All tables are in 2NF**.**

#### Step 3: Third Normal Form (3NF)

* **Rule**: Must be in 2NF, and there should be no transitive dependencies**.**

##### Assessment:

* + No non-key attributes depend on other non-key attributes within the same table.
  + All lookup fields (like hotel name, destination name) are accessed via FKs not duplicated.
* **Transformations**: None needed.
* **Status**: All tables are in 3NF**.**

### Final 3NF Relational Schema

Here are the relations confirmed to be in Third Normal Form**:**

#### User

* + - * **PK**: userID
      * **Attributes**: name, email, password, contactNo

#### Admin

* + - * **PK**: adminID
      * **Attributes**: name, email, password

#### Destination

* + - * **PK**: destinationID
      * **Attributes**: adminID (FK), name, description, street, city, category, image\_url

#### Hotel

* + - * **PK**: hotelID
      * **Attributes**: adminID (FK), name, street, city, facilities, image\_url

#### Booking

* + - * **PK**: bookingID
      * **Attributes**: userID (FK), hotelID (FK), checkInDate, checkOutDate, roomType, cost, bookingDate

#### Restaurant

* + - * **PK**: restaurantID
      * **Attributes**: adminID (FK), destinationID (FK), name, street, city, cuisine, image\_url

#### Reservation

* + - * **PK**: reservationID
      * **Attributes**: userID (FK), restaurantID (FK), reservationDate, reservationTime, numberOfGuests, tableNumber

#### Transportation

* + - * **PK**: transportationID
      * **Attributes**: destinationID (FK), adminID (FK), type, provider

#### Travel

* + - * **PK**: travelID
      * **Attributes**: userID (FK), transportationID (FK), travelDate, departureTime, fare, departurePoint, arrivalPoint, arrivalTime

#### Visit

* + - * **PK**: Composite (userID, destinationID, visitDate
      * **Attributes**: none beyond composite key

#### Favourite

* + - * **PK**: Composite (userID, destinationID)
      * **Attributes**: favouriteDate

#### Review

* + - * **PK**: reviewID
      * **Attributes**: userID (FK), destinationID (nullable FK), restaurantID (nullable FK), transportID (nullable FK), hotelID (nullable FK), rating, comment, reviewType

# CHAPTER 4: PHYSICAL DATABASE DESIGN

## STRUCTURE OF THE TABLES:

Here are the SQL queries to show the structure of tables.

|  |  |
| --- | --- |
| **Query** | DESCRIBE admin; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE booking; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE destination; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE favourite; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE hotel; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE reservation; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE restaurant; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE review; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE transportation; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE travel; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE user; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | DESCRIBE visit; |
| **Output** |  |

## DATA SAMPLES INSIDE TABLES:

Here are the SQL queries to retrieve data from tables.

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM admin; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM booking; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM destination; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM favourite; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM hotel; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM reservation; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM restaurant; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM review; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM transportation; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM travel; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM user; |
| **Output** |  |

|  |  |
| --- | --- |
| **Query** | SELECT \* FROM visit; |
| **Output** |  |

## QUERIES RESULTS:

|  |  |
| --- | --- |
| **Task 1** | Average rating per destination (all review types) |
| **Query** | select d.name as destination\_name, avg(r.rating) as average\_rating from destination d  join review r on d.destinationid = r.destinationid group by d.name  order by average\_rating desc; |

|  |  |
| --- | --- |
| **Output** |  |
| **Task 2** | Total number of hotel bookings by each user. |
| **Query** | select u.userid, u.name, count(b.bookingid) as total\_bookings from user u  join booking b on u.userid = b.userid group by u.userid, u.name  order by total\_bookings desc; |
| **Output** |  |
| **Task 3** | Users who made more than 2 restaurant reservations. |
| **Query** | select u.userid, u.name, count(r.reservationid) as total\_reservations from user u  join reservation r on u.userid = r.userid group by u.userid, u.name  having count(r.reservationid) > 2; |

|  |  |
| --- | --- |
| **Output** |  |
| **Task 4** | Find all users who have booked hotels in cities where the admin with adminID=1 manages destinations, showing user name and city. |
| **Query** | select distinct u.name as username, h.city from user u join booking b on u.userid  = b.userid join hotel h on b.hotelid = h.hotelid where h.city in (select city from destination where adminid = 1); |
| **Output** |  |
| **Task 5** | Average cost of hotel bookings per room type. |
| **Query** | select roomtype, avg(cost) as average\_cost from booking  group by roomtype  order by average\_cost desc; |

|  |  |
| --- | --- |
| **Output** |  |
| **Task 6** | Count of each review type submitted by users. |
| **Query** | select reviewtype, count(\*) as total\_reviews from review group by reviewtype order by total\_reviews desc; |
| **Output** |  |
| **Task 7** | List all restaurants that have reservations made by users who have favorited the destination where the restaurant is located (by city), showing restaurant name, city, and number of reservations. |
| **Query** | select r.name as restaurantname, r.city, count(res.reservationid) as totalreservations from restaurant r join reservation res on r.restaurantid = res.restaurantid join user u on res.userid = u.userid join favourite f on u.userid = |

|  |  |
| --- | --- |
|  | f.userid join destination d on f.destinationid = (select destinationid from destination where city = r.city limit 1) group by r.restaurantid, r.name, r.city order by totalreservations desc; |
| **Output** |  |
| **Task 8** | Find the restaurants that have never been reserved by users who have booked hotels in the same city. |
| **Query** | select r.name, r.city from restaurant r where not exists (select 1 from reservation res join booking b on res.userid = b.userid join hotel h on b.hotelid = h.hotelid  where res.restaurantid = r.restaurantid and h.city = r.city) order by r.city, r.name; |
| **Output** |  |
| **Task 9** | Find destinations that have both hotels and restaurants, listing destination name, number of hotels, and number of restaurants. |
| **Query** | select d.name as destinationname, count(distinct h.hotelid) as numhotels, count(distinct r.restaurantid) as numrestaurants from destination d left join hotel h on d.city = h.city left join restaurant r on d.city = r.city group by d.destinationid, d.name having numhotels > 0 and numrestaurants > 0 order by numhotels desc, numrestaurants desc; |

|  |  |
| --- | --- |
| **Output** |  |
| **Task 10** | Get Users Who Have Booked Hotels in the Same City as Their Favorite Destination. |
| **Query** | select name, email from user where userid in (select distinct b.userid from booking b join hotel h on b.hotelid = h.hotelid where h.city in (select d.city from favourite f join destination d on f.destinationid = d.destinationid where f.userid = b.userid)); |
| **Output** |  |

# CHAPTER 5: INTERFACE DESIGN

## LANGUAGE/FRAMEWORK:

### Technology Stack

##### Backend Framework

* + - * + **Flask** (Version 2.0+)

##### Key Technologies Used

* + - * + **Flask** – Core web framework for routing and application logic
        + **Flask-SQLAlchemy** – ORM for seamless MySQL integration
        + **Flask-Login** – User authentication and session management
        + **Flask-WTF** – Form handling and CSRF protection
        + **Jinja2** – Template engine for dynamic HTML rendering

### Advantages of the Chosen Stack

##### Modularity:

Blueprint-based architecture allows for an organized and maintainable code structure.

##### Security:

Built-in CSRF protection and password hashing ensure secure user interactions.

##### Database Integration:

SQLAlchemy provides robust and seamless integration with MySQL, supporting complex queries and relationships.

##### Template Engine:

Jinja2 enables dynamic and reusable HTML templates, improving development efficiency and maintainability.

## DATABASE CONNECTIVITY:

### Configuration

The GUI client connects to the MySQL database using the Flask-SQLAlchemy extension, which serves as an Object Relational Mapper (ORM). This extension allows the application to interact with the database using Python objects rather than direct SQL queries. The database connection details, such as the database type, username, password, host, and database name, are specified in the application's configuration settings. Once configured, Flask-SQLAlchemy manages the connection pool and handles all communication between the application and the MySQL database.

### Error Handling

To ensure reliability, the application includes error handling mechanisms for database operations. If a database error occurs (such as a connection failure or a query error), the application rolls back any incomplete transactions to maintain data integrity. Additionally,

user-friendly error messages are displayed, and detailed error logs are recorded for developers to review and troubleshoot issues. This approach helps prevent application crashes and ensures a smooth user experience even when unexpected database problems arise.

## STORED PROCEDURES AND FUNCTIONS:

#### Procedure 1: GetFullUserActivity

**Purpose**: Shows a complete report of a user's tourism activity: visits, bookings, reservations, reviews, and travel.

**Script**:

DELIMITER //

CREATE PROCEDURE GetFullUserActivity(IN uid INT) BEGIN

-- Visits

SELECT 'Visited Destination' AS Activity, D.name AS Detail, V.visitDate AS Date FROM Visit V

JOIN Destination D ON V.destinationID = D.destinationID WHERE V.userID = uid

UNION

-- Hotel Bookings

SELECT 'Hotel Booking', H.name, B.checkInDate FROM Booking B

JOIN Hotel H ON B.hotelID = H.hotelID WHERE B.userID = uid

UNION

-- Restaurant Reservations

SELECT 'Restaurant Reservation', R.name, RS.reservationDate FROM Reservation RS

JOIN Restaurant R ON RS.restaurantID = R.restaurantID WHERE RS.userID = uid

UNION

-- Travel

SELECT 'Travel Trip', T.arrivalPoint, T.travelDate FROM Travel T

WHERE T.userID = uid UNION

-- Reviews

SELECT 'Review Posted', reviewType, NULL FROM Review

WHERE userID = uid; END;

// DELIMITER ;

#### Procedure 2: GetTopRatedDestinationsWithHotelOptions

**Purpose**: Lists top 5 destinations with average rating ≥ 4 and shows nearby hotel options.

**Script**:

DELIMITER //

CREATE PROCEDURE GetTopRatedDestinationsWithHotelOptions() BEGIN

SELECT

D.name AS Destination, D.city,

ROUND(AVG(R.rating), 2) AS AvgRating, H.name AS HotelNearby

FROM Destination D

JOIN Review R ON R.destinationID = D.destinationID LEFT JOIN Hotel H ON H.city = D.city

GROUP BY D.destinationID, D.name, D.city, H.name HAVING AvgRating >= 4

ORDER BY AvgRating DESC LIMIT 5;

END;

// DELIMITER ;

#### Function 1: GetTotalUserCos

**Purpose**: Calculates the total money a user has spent on bookings + travels.

##### Script:

DELIMITER //

CREATE FUNCTION GetTotalUserCost(uid INT) RETURNS DECIMAL(10,2)

DETERMINISTIC BEGIN

DECLARE total DECIMAL(10,2); SELECT

IFNULL(SUM(B.cost), 0) + IFNULL((SELECT SUM(T.fare) FROM Travel T WHERE

T.userID = uid), 0) INTO total

FROM Booking B WHERE B.userID = uid;

RETURN total;

END;

// DELIMITER ;

* + 1. **Function 2:** GetHotelRevenue

**Purpose**: total revenue generated by a hotel based on the cost of all bookings made for it.

**Script**:

DELIMITER //

CREATE FUNCTION GetHotelRevenue(hid INT) RETURNS DECIMAL(10,2)

DETERMINISTIC BEGIN

DECLARE totalRevenue DECIMAL(10,2);

SELECT IFNULL(SUM(cost), 0.00) INTO totalRevenue FROM Booking

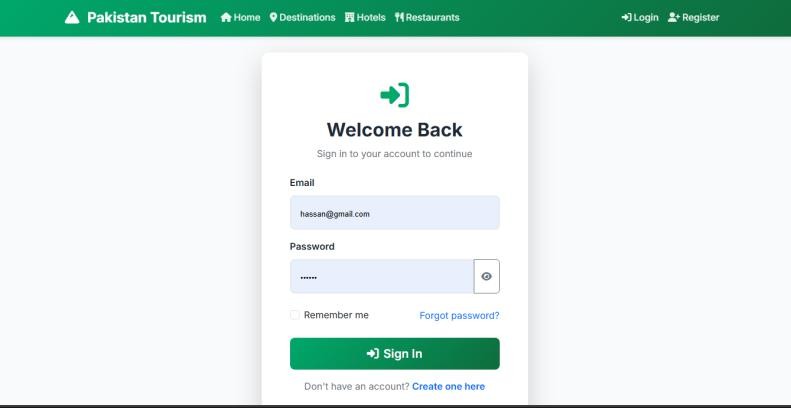
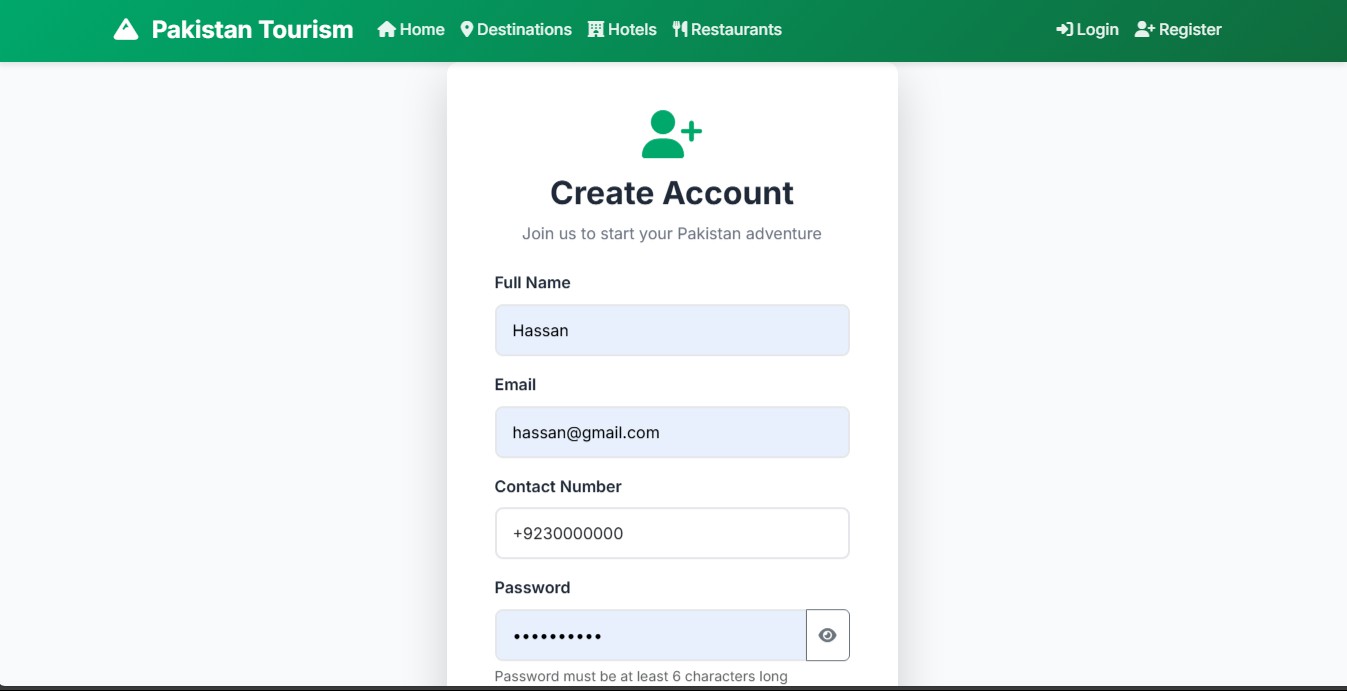
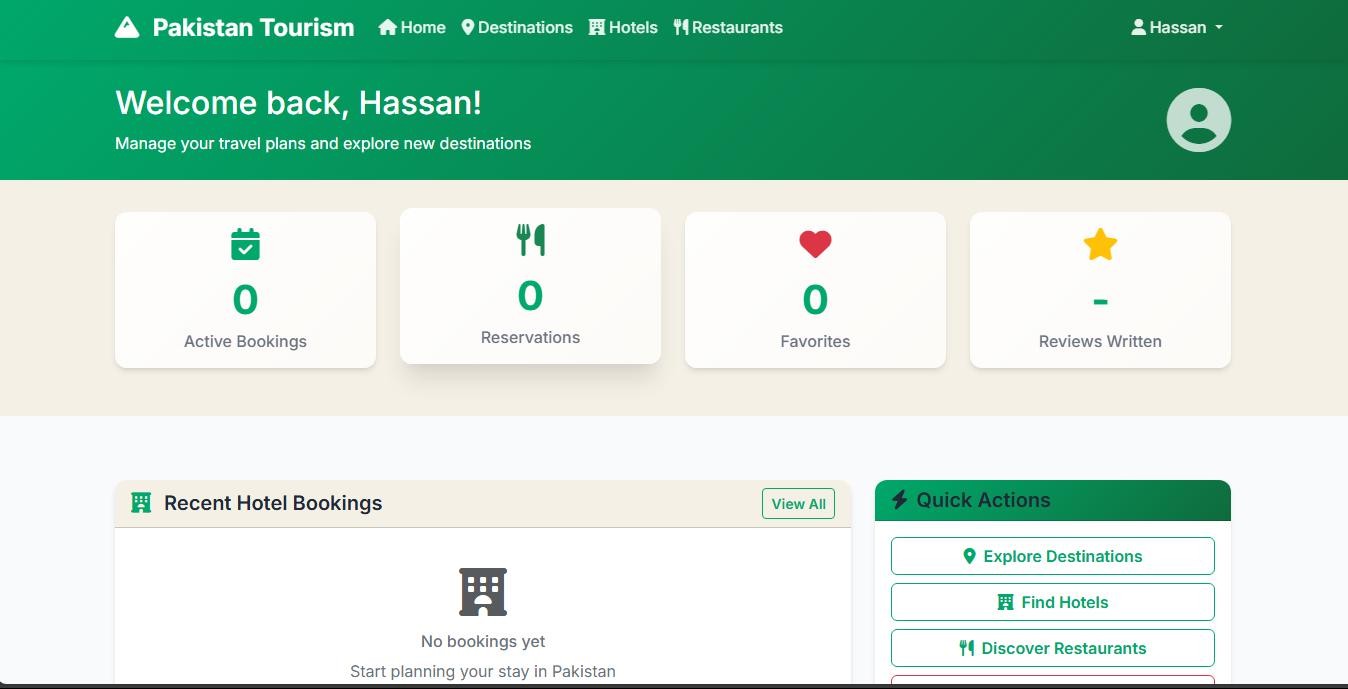
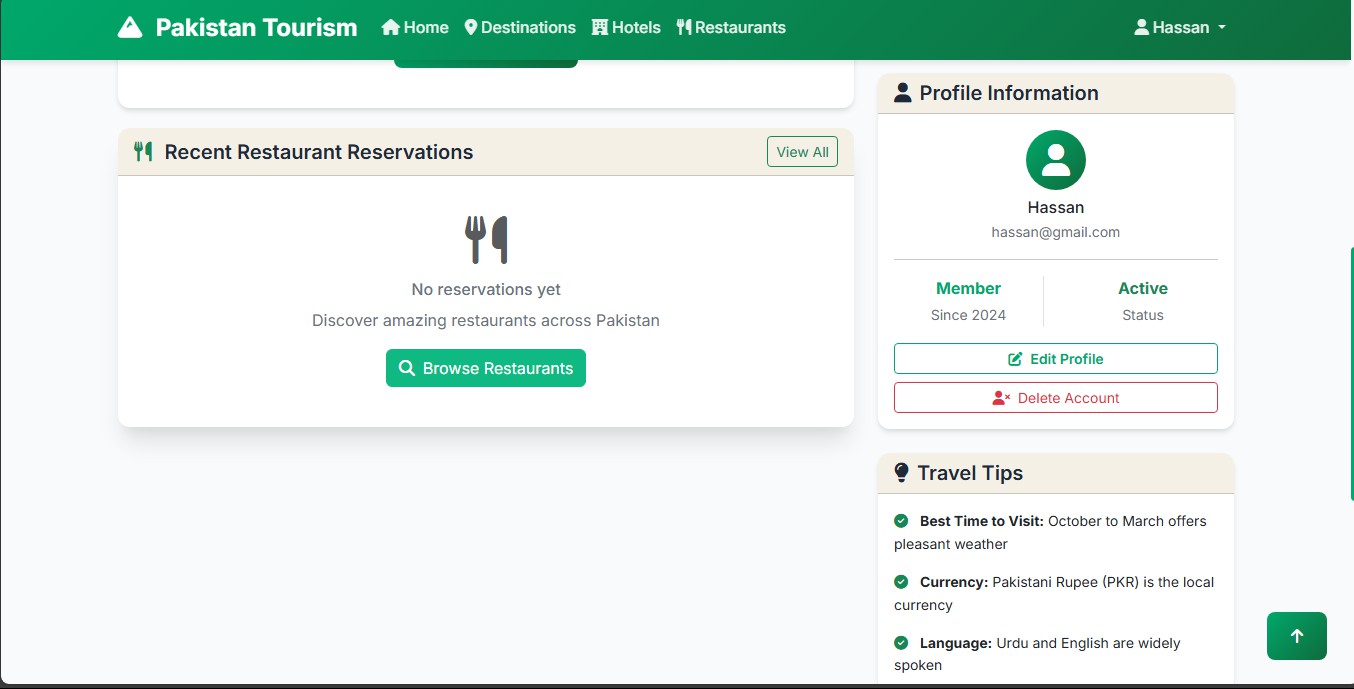
WHERE hotelID = hid; RETURN totalRevenue; END;

// DELIMITER ;

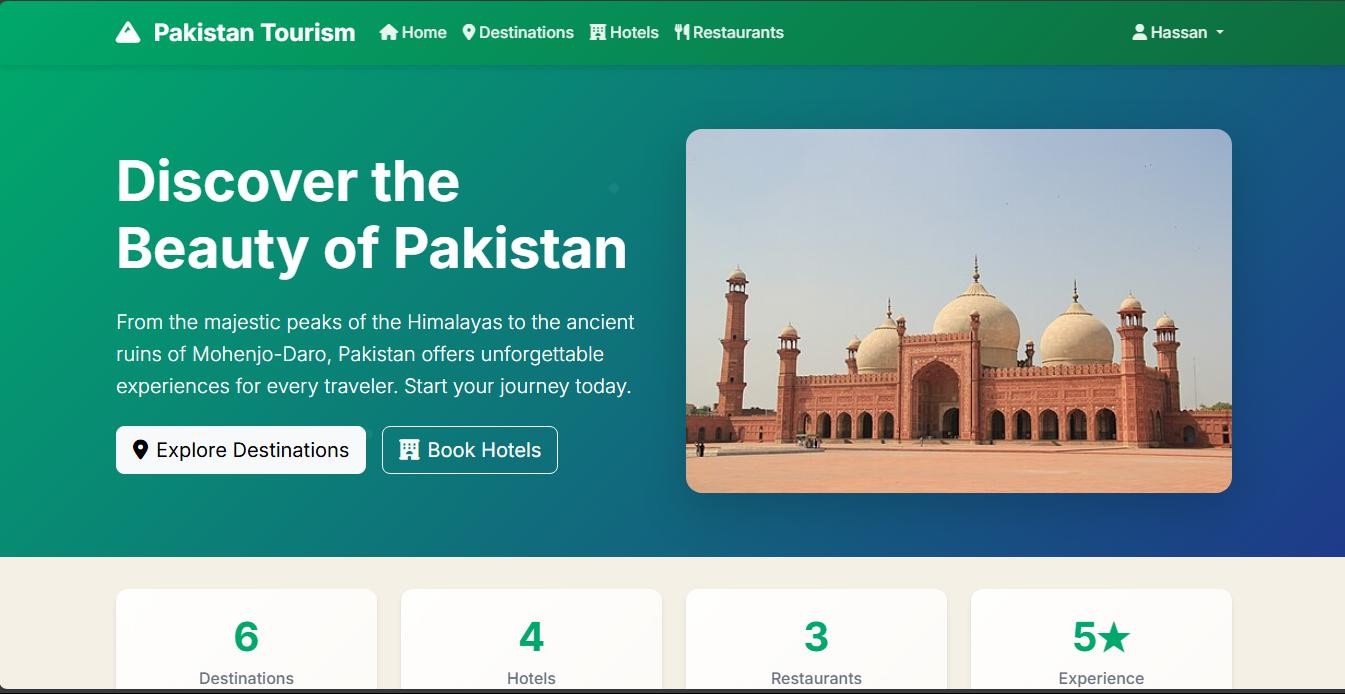
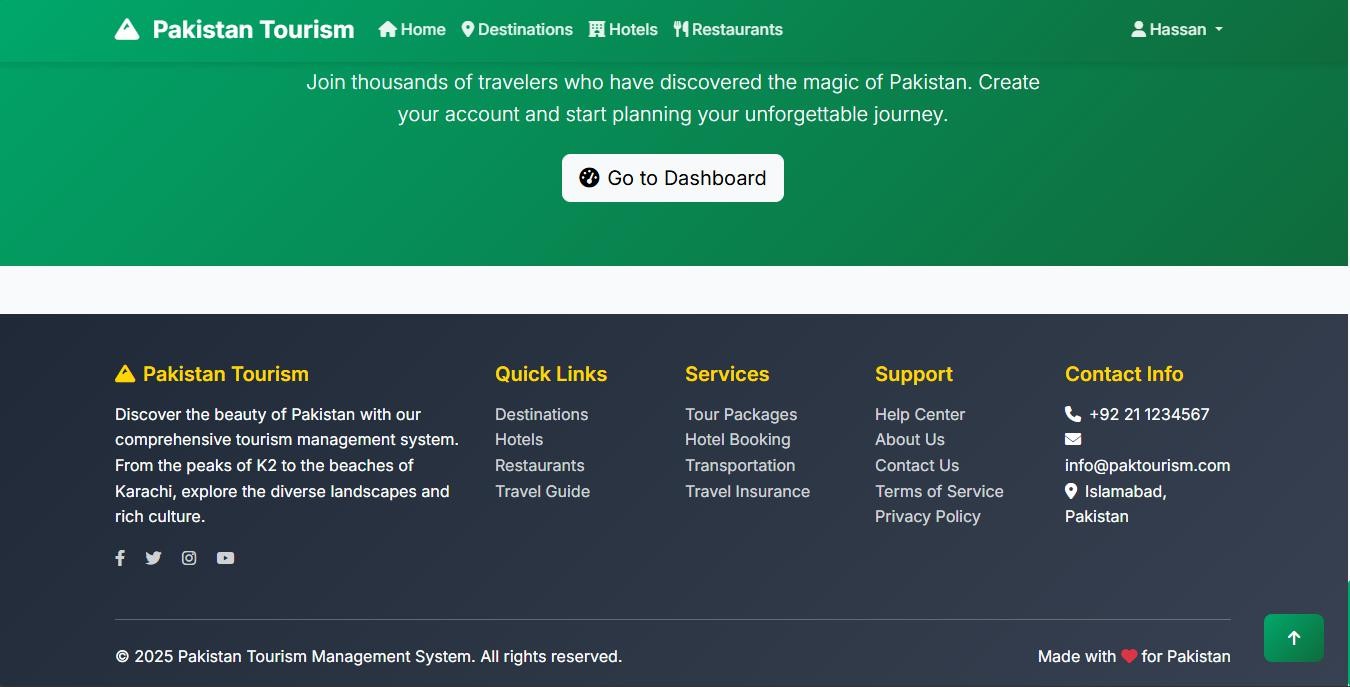
In the current version of the Tourism Management System, we have developed several advanced stored procedures and functions such as GetFullUserActivity, GetTopRatedDestinationsWithHotelOptions, GetHotelRevenue, and GetTotalUserCost. These routines are designed to enhance system intelligence by providing detailed user activity tracking, performance analytics, and financial insights. Although these procedures and functions are not yet integrated into the graphical user interface (GUI), they are a vital part of the system’s backend architecture. In future updates, these components can be effectively connected to dashboards, admin panels, and reporting modules to provide real-time analysis and support decision-making. Their integration will elevate the system from a functional platform to a fully data-driven, enterprise-level tourism management solution.

## INTERFACES:

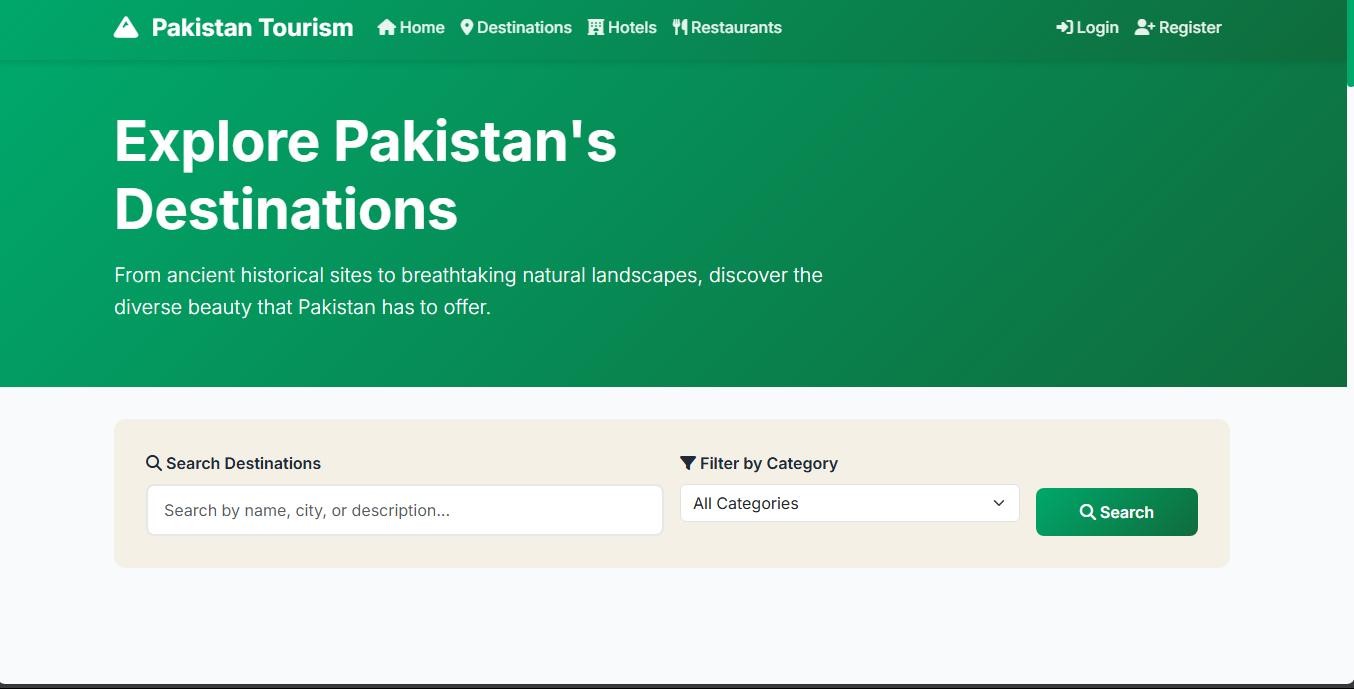
#### Authentication & User Management

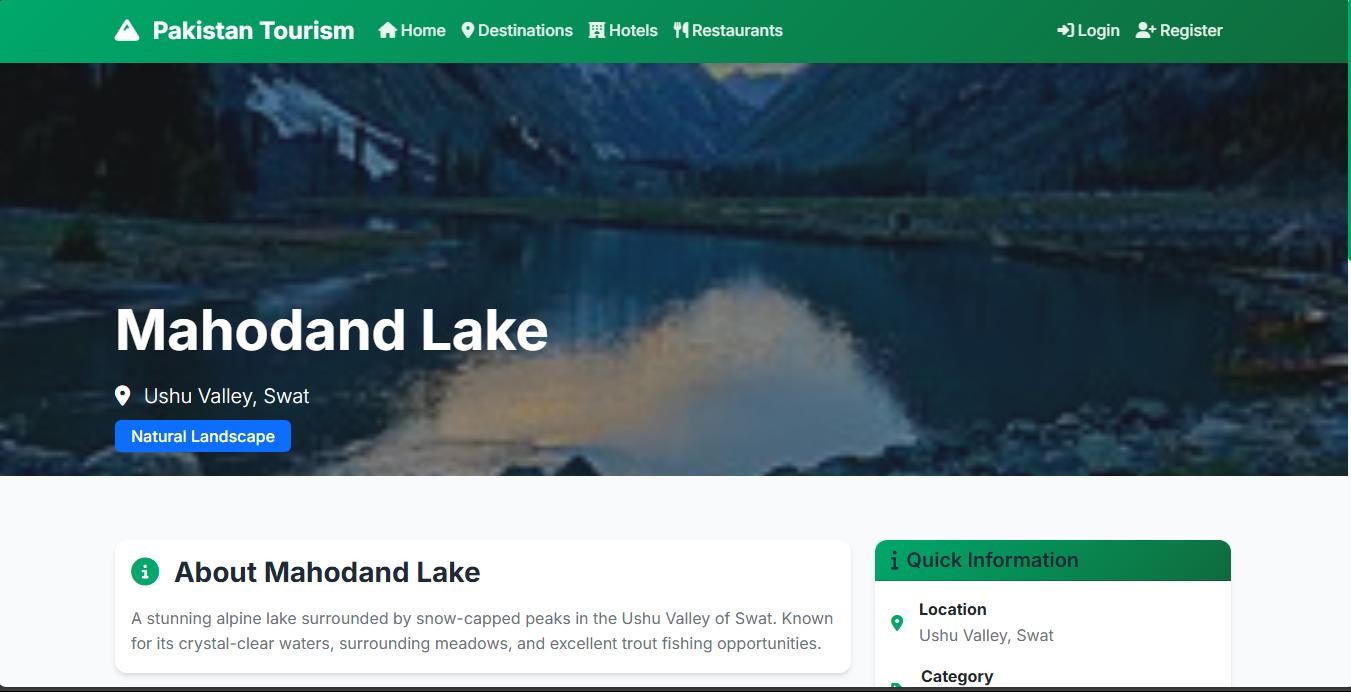
* + - * **Login page**
      * **Registration form**
      * **User dashboard/profile**
      * **Account settings/management**

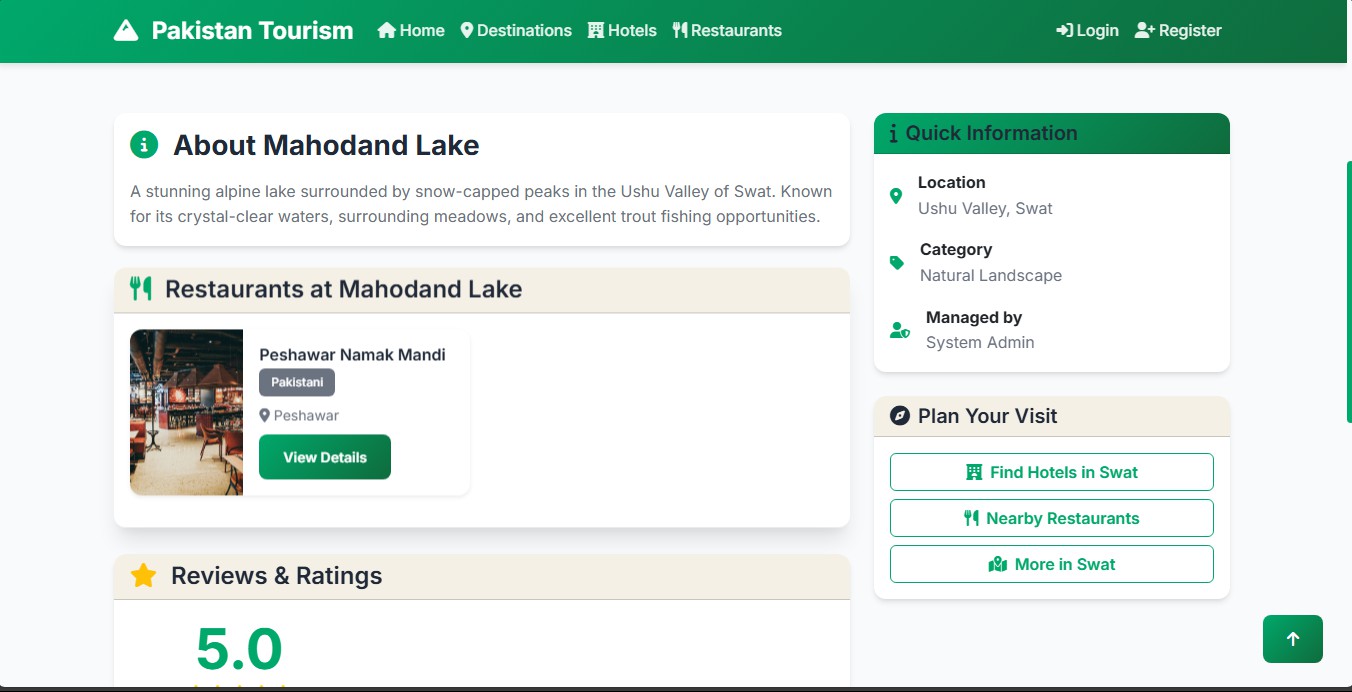
#### Main Navigation & Homepage

* + - * Homepage with hero image
      * Footer with contact information

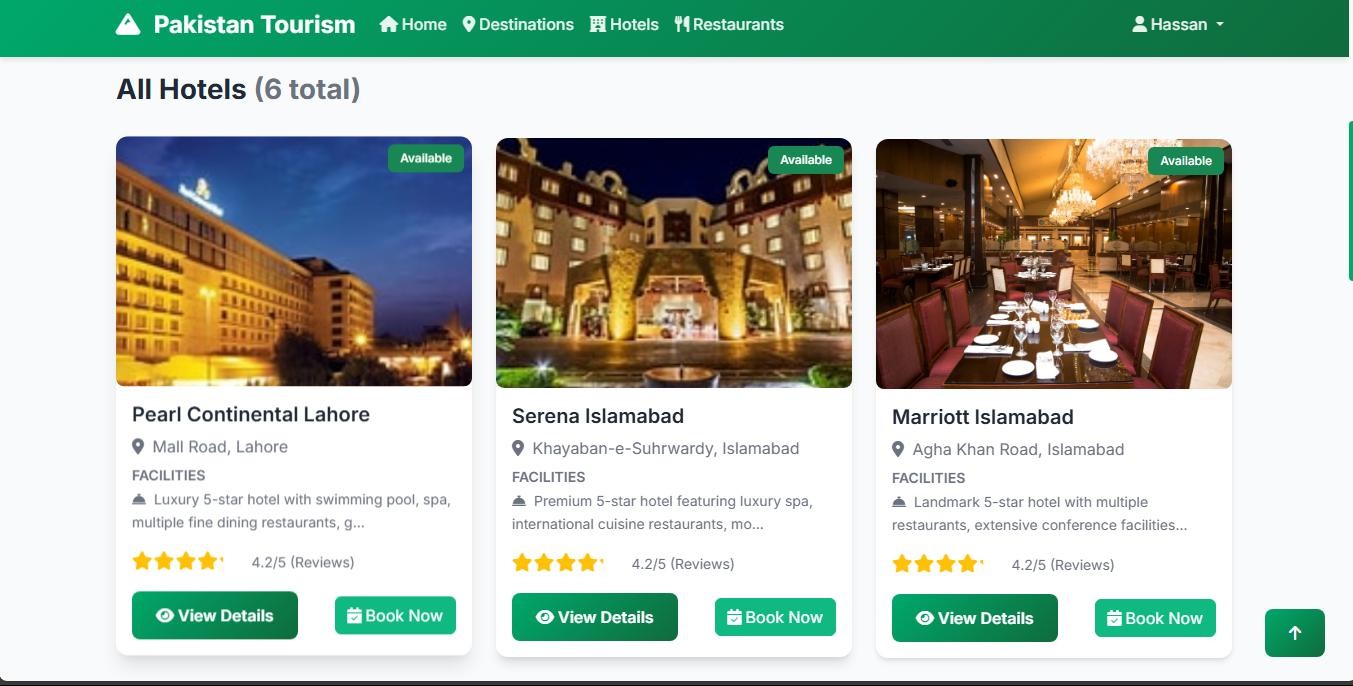
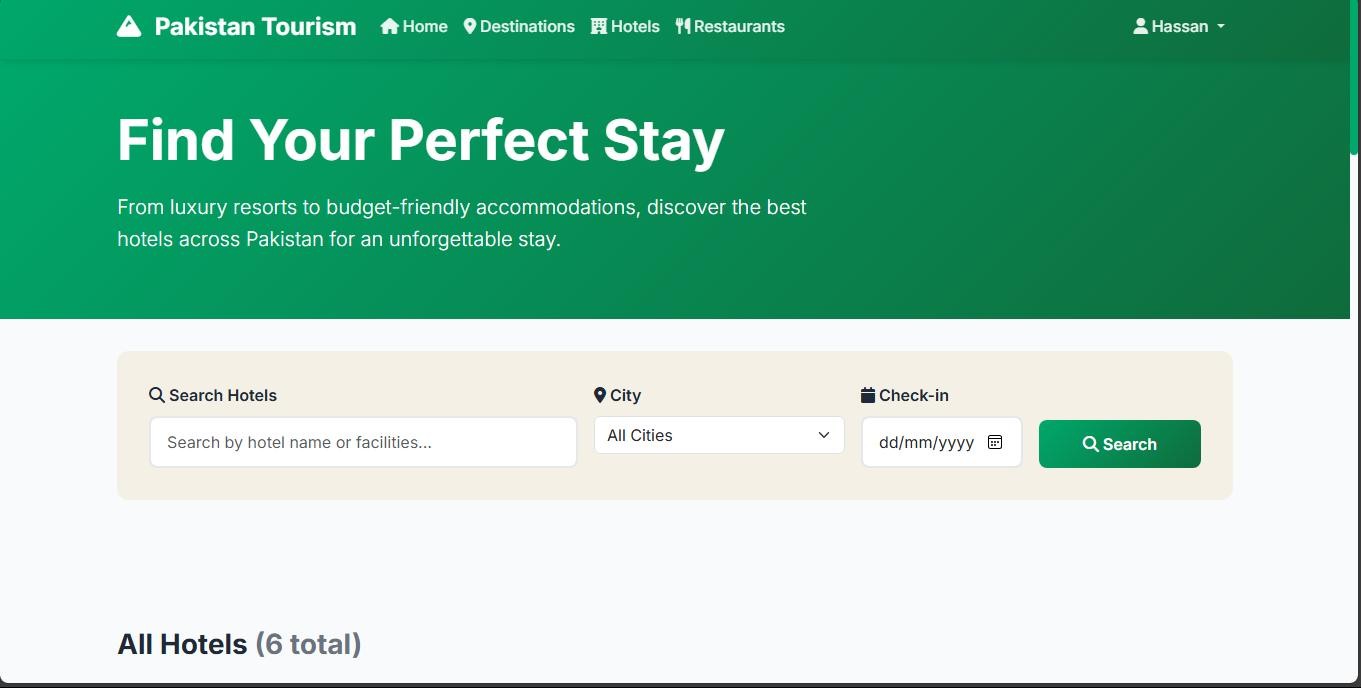
#### Destination Features

* + - * Destinations listing page
      * Individual destination details page

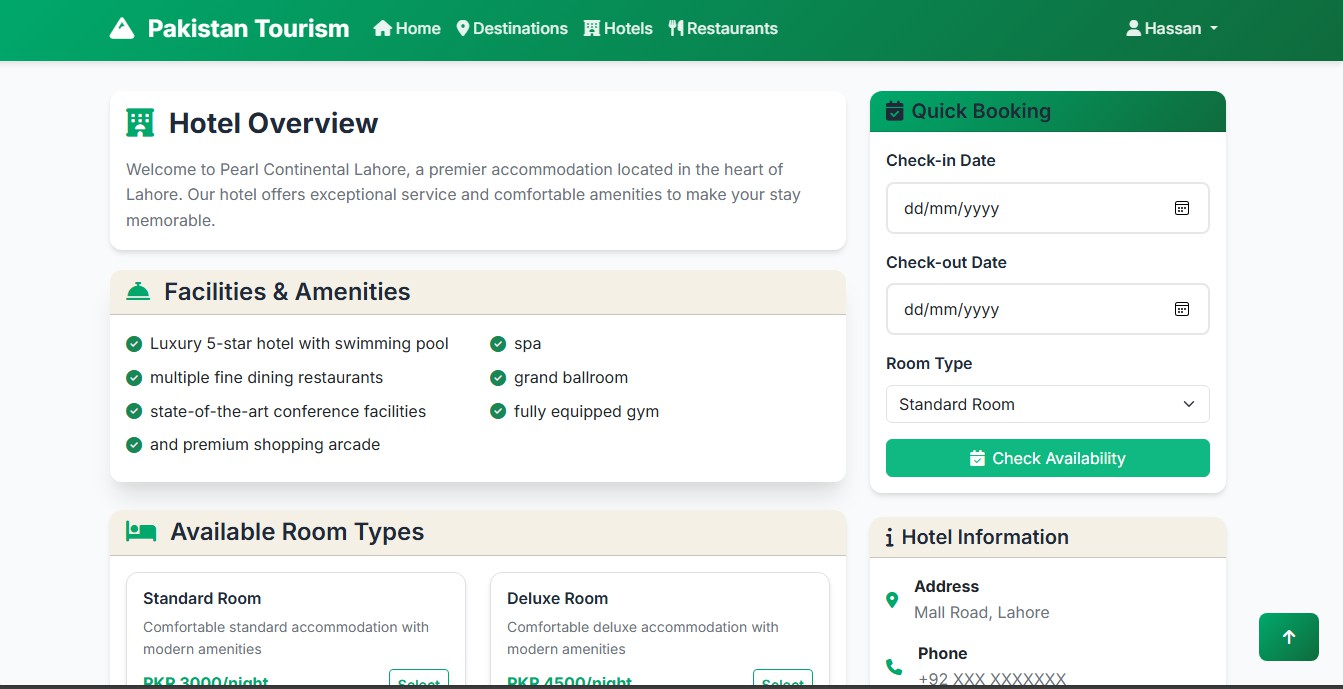


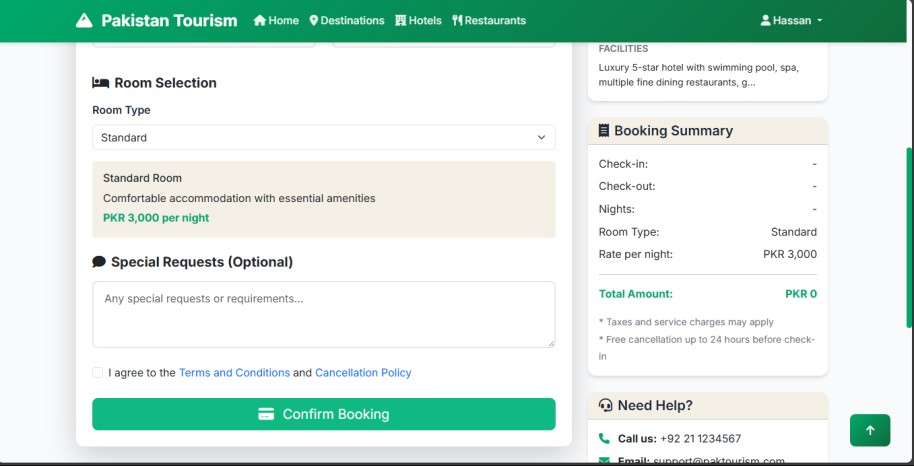
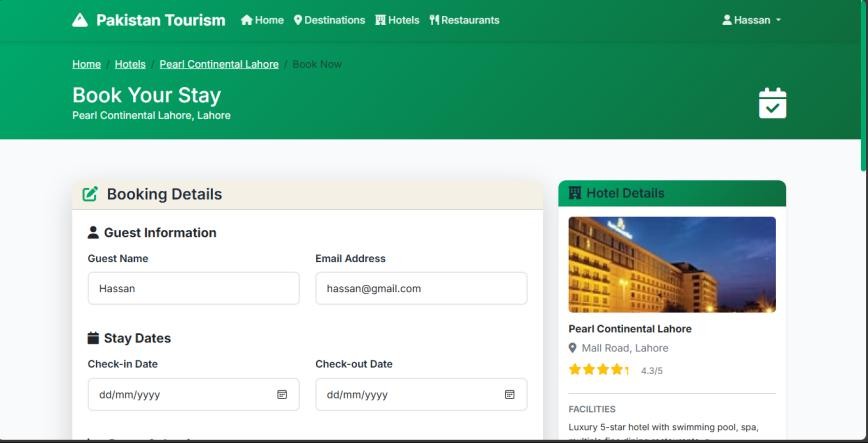


#### Hotel Management

* + - * Hotels listing page
      * Individual hotel details page

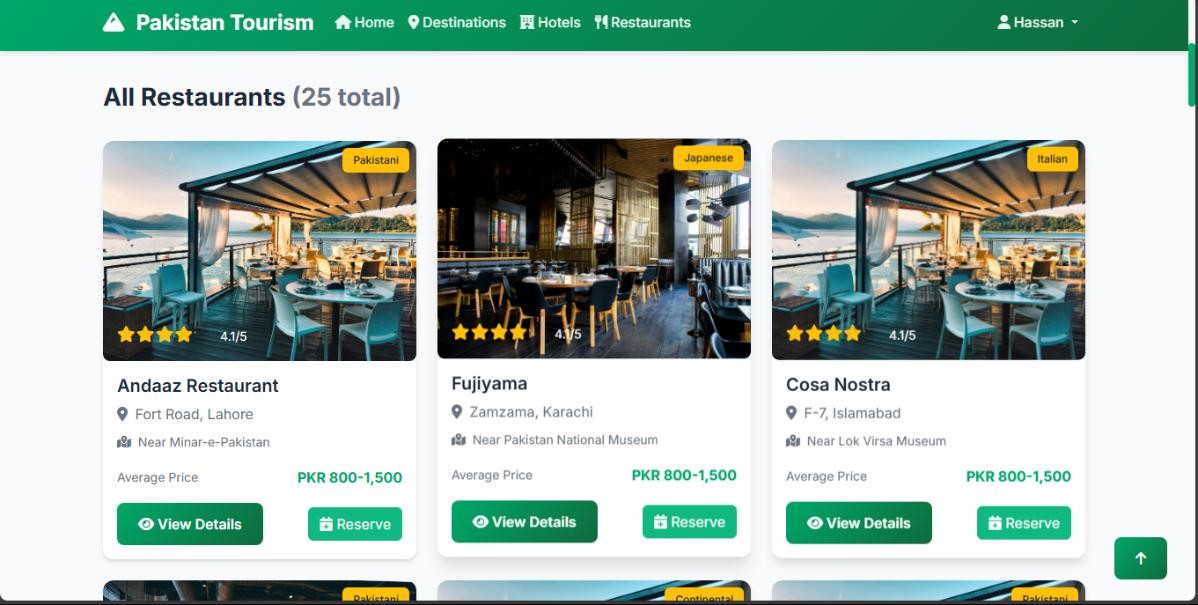




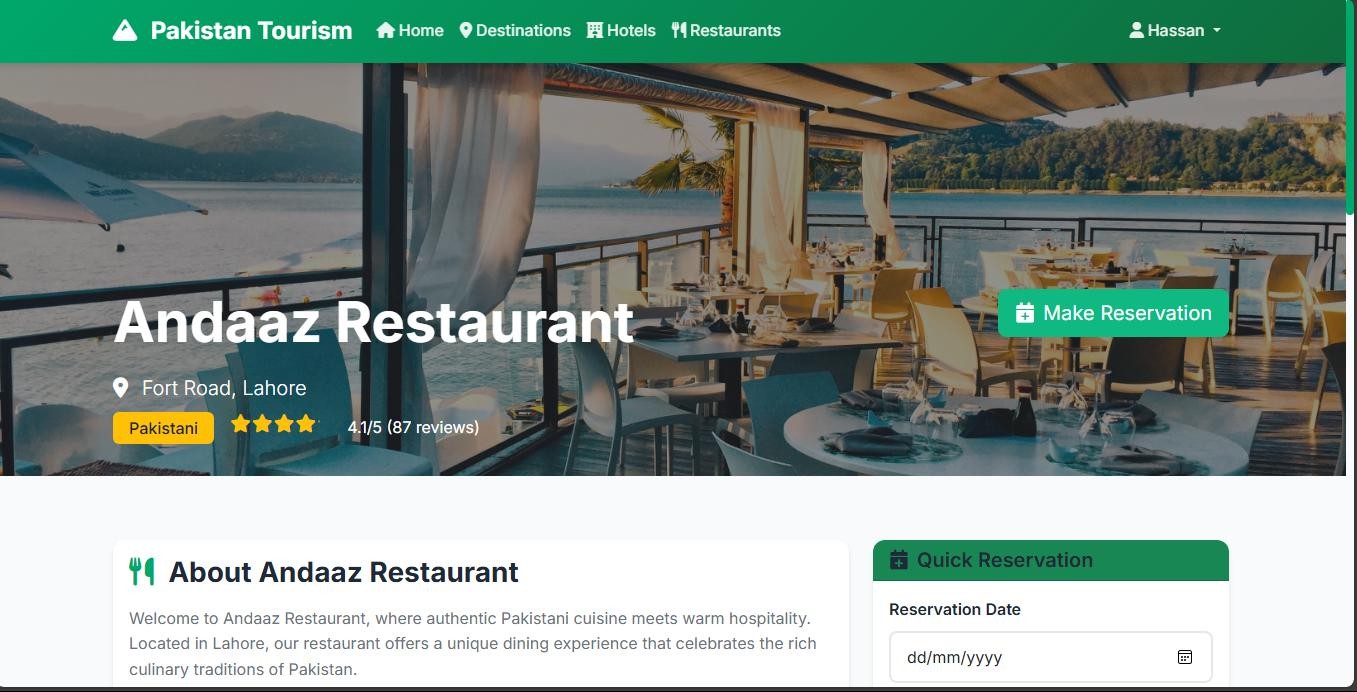
* + - * Hotel booking form

#### Restaurant Features

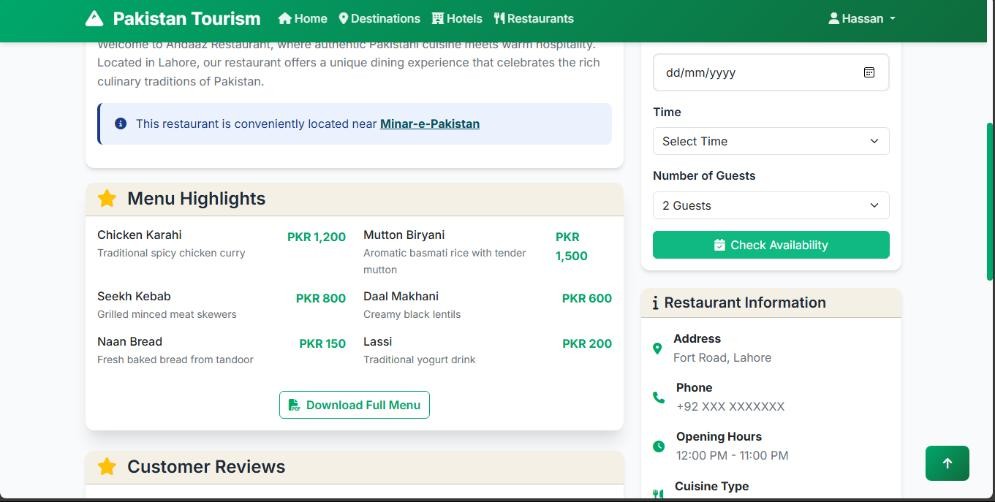
* + - * Restaurants listing page



* + - * Restaurant details page



* + - * Menu display

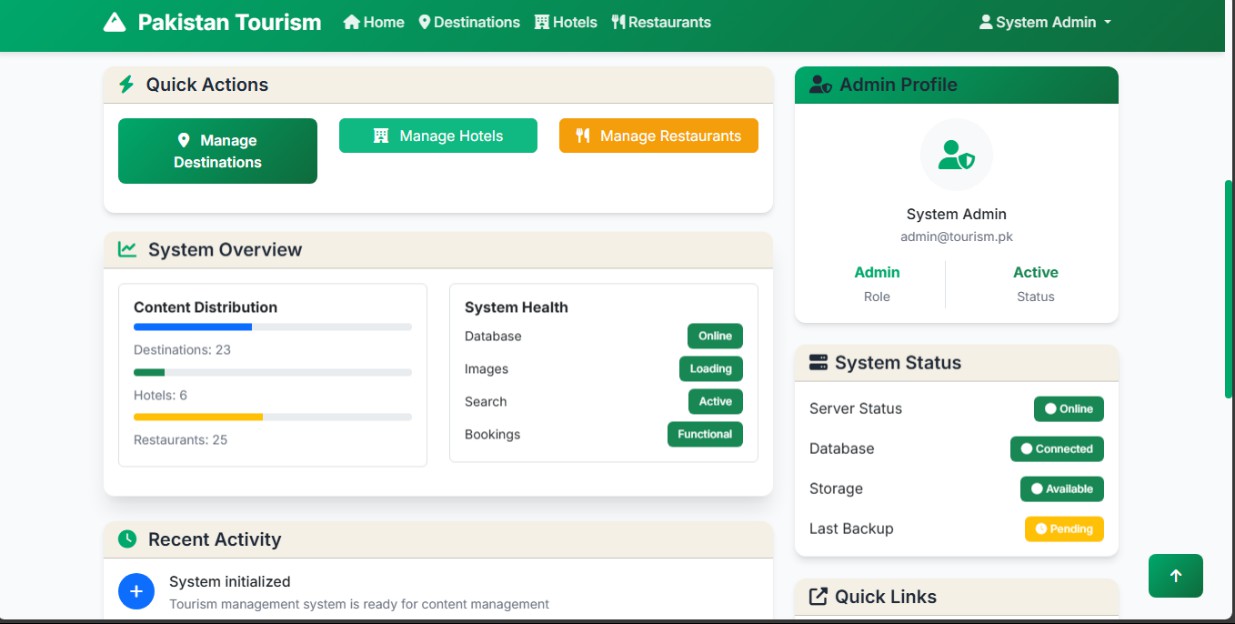


#### Admin Interface

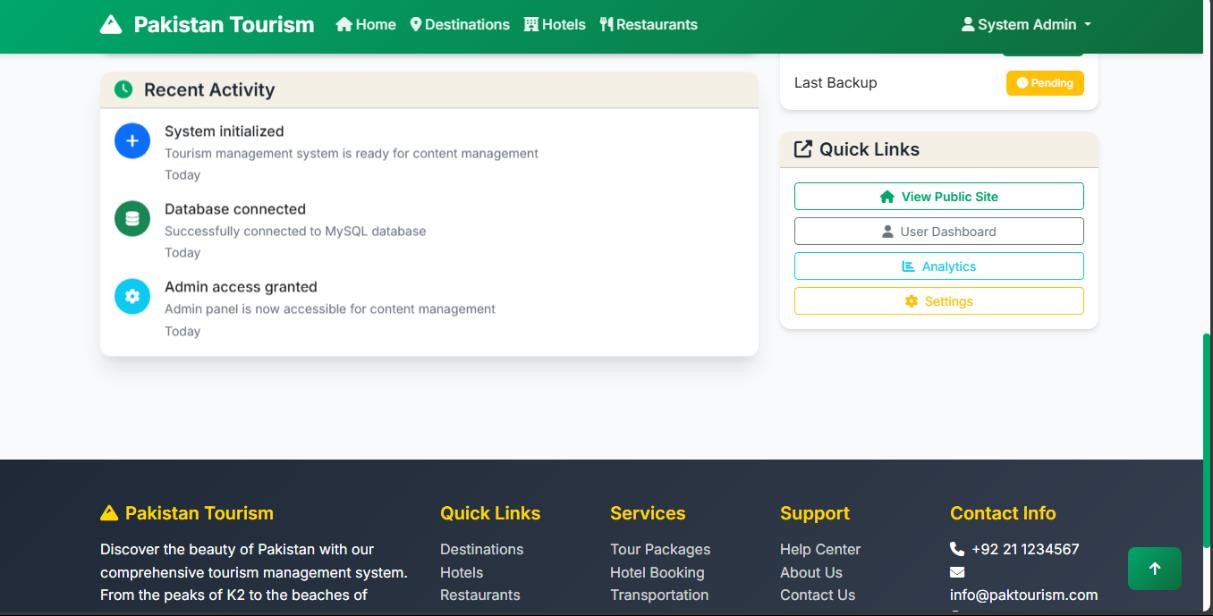
* + - * Admin dashboard



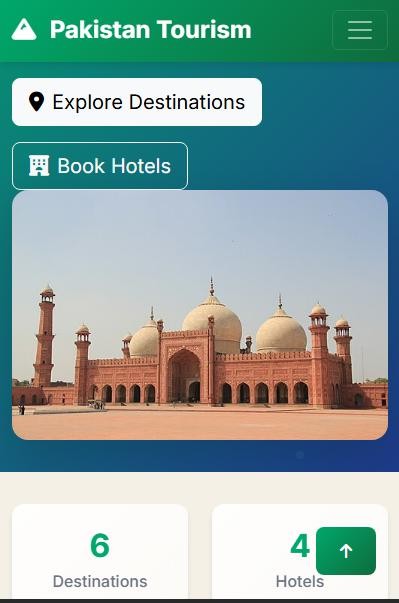
* + - * Admin Profile and Quick Action

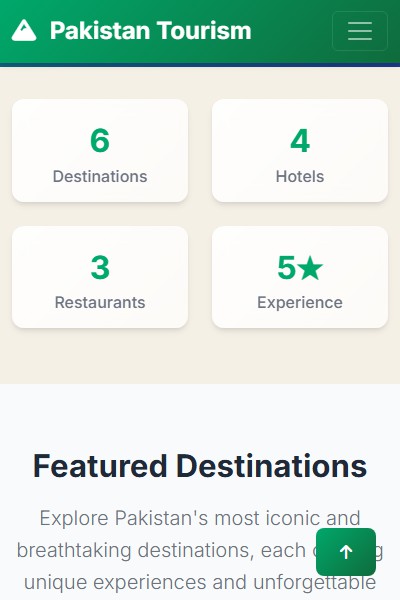


* + - * Admin Recent Activity



#### Mobile Responsiveness

****

## CHAPTER 6: CONCLUSION

## LESSONS LEARNED:

#### Technical Insights

Throughout the project, we gained practical experience in designing and implementing a complete database system. Starting from Entity-Relationship (ER) diagrams, we learned how to convert conceptual designs into relational schemas and then apply normalization (1NF, 2NF, 3NF) to improve efficiency and data integrity. Working with MySQL Server, we practiced using DDL and DML commands, created optimized table structures, and implemented real- time data manipulation. Later, by developing a Graphical User Interface (GUI), we understood how the frontend interacts with the backend and how to manage data through forms and inputs effectively.

#### Project Management Skills

Managing the project taught us how important it is to plan things properly, divide work among team members, and use our time wisely. We split the project into different steps like designing, normalization, making the database, and creating the GUI. We gave each step a deadline to stay on track. This helped us learn how to keep things moving, deal with delays, and finish the work in an organized way. It also showed us how to decide which tasks are most important and stay focused on our main goal.

#### Teamwork Experience

Working as a team was a learning experience. Initially, it was difficult to ensure that everyone was on the same page, especially during the design and schema phases. However, through regular discussions, clear communication, and assigning roles based on strengths, we were able to coordinate our efforts and solve problems together. This helped us build team trust, collaboration, and shared responsibility, which are essential in real-world software development.

#### Documentation Skill

While writing the documentation for our project, we learned how to explain things like ER diagrams, schema design, normalization, SQL queries, and GUI in a simple way. We also learned what things should be added in the documentation, like system design, steps of the work, and final results. This helped us understand how to arrange everything in the right order. We also learned how to use headings, tables, and diagrams to make the document neat and easy to understand. This improved our writing and helped us know how to make a good report.

## CHALLENGES AND SOLUTIONS:

#### Database Design and Relationship Management

##### Challenge:

Designing a relational database that could efficiently handle complex relationships between users, admins, destinations, bookings, reviews, and favorites was a significant challenge.

Ensuring data integrity, supporting cascade deletions, and preventing orphaned records required careful planning.

##### Solution:

We addressed this by thoroughly mapping out entity relationships before implementation, using entity-relationship diagrams, and applying best practices for foreign key constraints and cascading rules. Regular reviews and adjustments were made as new requirements emerged.

#### User Authentication and Authorization

##### Challenge:

Managing separate authentication flows for admins and regular users, and ensuring that only authorized users could access sensitive features, was complex. There were also issues with session management and distinguishing user roles during login.

##### Solution:

We implemented a robust authentication system that clearly separated admin and user roles. Access controls were enforced at both the route and template levels, and thorough testing was conducted to ensure no privilege escalation was possible.

#### Form Validation and Security

##### Challenge:

Handling multiple forms (registration, login, profile editing, etc.) with proper validation and security (such as CSRF protection) was essential to prevent invalid data and security vulnerabilities.

##### Solution:

We used established form libraries that provide built-in validation and CSRF protection. All forms were tested for edge cases, and user feedback was provided for invalid submissions, improving both security and user experience.

#### Responsive and User-Friendly Frontend

##### Challenge:

Ensuring the website was responsive and visually appealing across devices, while also providing a seamless user experience, required careful frontend development and testing.

##### Solution:

We adopted a modern CSS framework (Bootstrap) to ensure responsiveness and consistency. User feedback was incorporated to refine the interface, and cross-browser testing was performed to resolve compatibility issues.

#### Error Handling and User Feedback

##### Challenge:

Providing meaningful error messages and handling unexpected issues gracefully was necessary for both debugging and user satisfaction.

##### Solution:

Comprehensive error handling was implemented throughout the application. Users received clear feedback when something went wrong, and errors were logged for developers to review and address.

#### Testing and Quality Assurance

##### Challenge:

Ensuring the reliability of the system through testing was time-consuming, especially as the project grew in complexity.

##### Solution:

We developed a suite of unit and integration tests for critical features. Manual testing was also performed to catch issues not covered by automated tests, ensuring a stable and reliable application.

## FUTURE WORK AND IMPROVEMENTS:

#### Mobile application development

To make the system more accessible, a mobile app can be developed using frameworks like Flutter or React Native. This app will allow users to book, browse, and manage their accounts from smartphones. It will sync with the existing MySQL database using APIs, keeping data updated and connected across both web and mobile platforms.

#### Payment gateway integration

In the next phase, we can connect the system to secure online payment services like Stripe, PayPal, EasyPaisa, or JazzCash. This will allow users to pay for bookings or services directly from the platform. The integration will include backend payment processing, transaction tracking, and secure checkout flows with encrypted data handling.

#### AI-powered recommendation system

An intelligent system can be added that teaches user preferences and browsing history to suggest popular destinations, hotels, or restaurants. This can be built using machine learning algorithms and connected to the current MySQL database through a data pipeline that collects and processes user activity for generating personalized recommendations.

#### Technical Enhancements

The system can be improved by upgrading the backend logic and structure. This includes implementing modular code design and reducing query execution time. These changes will make the platform more stable and scalable for handling more users and data in the future.

#### Performance optimization

We aim to reduce system loading time and improve speed by optimizing SQL queries, adding caching, and refining database indexes. Regular performance testing will be added to check system health and response time, especially under high traffic.

#### Enhanced security measures

To protect user data, the system can include advanced security practices like two-factor authentication and data encryption. Regular security audits and role-based access control will be implemented to ensure only authorized users can access sensitive features

## FINAL THOUGHTS:

The Pakistan Tourism Management System successfully demonstrates:

#### Robust user management

The system ensures efficient user management by allowing administrators to add, update or remove users easily. Each user is assigned specific roles and permissions which helps in maintaining system integrity and controlling access to sensitive data. The design also supports scalability for managing a growing number of users.

#### Secure authentication

The project implements secure authentication techniques, including encrypted password storage and user session control. Login credentials are validated through a secure process that minimizes the risk of unauthorized access. Additionally, user sessions are managed carefully to ensure that data remains protected during active use. The benefit of encrypted passwords is that crashing a password will be very difficult.

#### Efficient database operations

All the database operations in our project work fast and correctly. We can easily add, see, change, or delete information without any problems. To make things quicker, we added special indexes on important columns like user ID and booking date. This helps the system find data faster. We also made sure that all data is linked properly, so there are no mistakes or missing connections. The database is neat and clean, without any extra or repeated data, which helps it run better. Everything is set up in a way that if we want to add more data or users in the future, it will still work smoothly.

#### Responsive user interface

The user interface is designed to be quick and simple. Options are well organized, helping users complete tasks like booking or searching without confusion. Buttons and Sidebar work smoothly, providing trouble-free experience. The clean layout makes the system easy to understand, even for first-time users.

# 7 REFERENCES

1. World Travel & Tourism Council (WTTC), “Travel & Tourism Economic Impact 2023: Pakistan,” [Online]. Available:

[https://wttc.org](https://wttc.org/)

1. Pakistan Tourism Development Corporation (PTDC), “Tourism Statistics Report 2022,” [Online]. Available:

[https://www.tourism.gov.pk](https://www.tourism.gov.pk/)

## AI PROMPTS:

Below is a list of important AI prompts we used for analysis and improvement of our Tourism Management System database design:

* + - Can you identify possible errors or weaknesses in our ERD and relational schema? Used to validate correctness of entity relationships and structure.
    - Review the functional dependencies in our schema and explain how they affect normalization.
    - Are there any unnecessary composite keys or redundant relationships in our schema?
    - Write proper DDL statements with constraints (PRIMARY, FOREIGN KEY, NOT NULL) for entities like User, Booking, Hotel, and Review.
    - Generate realistic sample INSERT statements for Booking, Review, and Reservation using Pakistani tourism context.
    - Suggest complex SQL queries to analyze booking trends, average ratings, and user activities.
    - How can referential integrity be enforced in our tourism database system using constraints or triggers?
    - What indexing strategies or optimization techniques can improve performance in a tourism-related relational database?
    - How can we scale the Tourism Management System database design to support nationwide or international data?