Manzil

SPROJ Report

Muhammad Mehdi_25100313

Muhammad Usman Arshid_25100320

Umer Inayat_24100199

Omar ibne Sajjad_25100015

Advisor:
School of Science and Engineering
Lahore University of Management Sciences
28th May, 2025

Acknowledgment and Dedication

Certificate

I certify that the senior project titled "**Manzil**" was completed under my supervision by the following students:

_____

_____

_____

and the project deliverables meet the requirements of the program.

---------- Date:

Advisor (Signature)

---------- Date:

Co-advisor (if any)

Table of Contents

1. Introduction
2. System Requirements
3. System Architecture
4. Requirements Specifications
5. Software Development Methodology and Plan
6. Database Design and Web Services
7. System User Interface
8. Project Security
9. Risk Management
10. Testing and Evaluation
11. Deployment Guidelines
12. Conclusion
13. Review Checklist

List of Figures

1. Introduction

a. Introduction

Manzil is a mobile application designed to assist users traveling to major cities in Pakistan, such as Lahore, Karachi, and Islamabad, and the flexibility of "Expanding the city base" to make this a business. Its primary aim is to streamline travel planning by offering a one-stop solution for various needs. These needs include booking hotels, renting vehicles, checking weather updates, and finding local services like restaurants, schools, and hospitals, along with their reviews to help users make better choices. The initial prototype phase focused on establishing fundamental functionalities and use cases, including CRUD operations on user profiles with authentication and authorization. Subsequent development sprints aim to implement features like hotel booking functionality, a dedicated hotel management interface, a search bar for locations, the ability to select a city at the app start, and **AI personalization** into the app - our biggest goal in making this app. The domain of the Manzil application is travel planning and tourism within Pakistan, with a specific focus on major cities. The primary target users of the Manzil application are **travelers** who will benefit from easy access to essential travel information, booking services, and personalized recommendations. The application is also designed for hotel management staff and car rental staff to have different web-based portals/dashboards at their disposal - who can use it to update hotel details and manage customer interactions - and businesses on the Manzil app.

b. Objective and Scope

It aims to be a one-stop platform for accessing essential travel information, booking, and receiving personalized recommendations. This application was chosen to streamline the travel process by centralizing services like hotel and vehicle booking, weather updates, and local service discovery with reviews, thereby improving the overall travel experience. In my opinion, Manzil can **enhance** business operations by providing a direct channel for hotels and other

service providers to connect with travelers, potentially increasing bookings and visibility. The use of machine learning for personalized suggestions can also drive user engagement.

c. Development Methodology

The project is **Agile** methodology (an iterative and incremental approach). The project is structured into sprints. Each sprint focused on implementing a specific set of requirements and the initial focus of a working prototype with core functionalities before moving to more complex features.

d. Contributions

Integrating various essential travel services into a single mobile platform is a key innovative aspect. This includes features for booking hotels, renting vehicles, checking weather updates, and finding local services like restaurants, schools, and hospitals, complete with reviews to aid decision-making. The application leverages machine learning (ML) to offer **personalized** recommendations to users based on their preferences and previous searches. It also supports real-time services such as live weather updates and aims to provide real-time availability data for hotel bookings.

2. System Requirements

a. System Actors

Actor Name	Description
Customer	Customers are the primary users of the app. They can create accounts, log in, search for tourist locations, view details of hotels, hospitals, car rental services, and tourist attractions, book services, and leave reviews for services they've used
Hotel Management/Administration	Hotel administrators handle hotel-related operations. They can create accounts for their hotels, connect with customers, update room availability, upload hotel details and pictures, and view customer reviews, ensuring smooth customer interaction and bookings.
App Administration/Developers	The app administration is responsible for maintaining and managing the backend and overall functionality of the app and resolving any related issues.
Car Rental Staff	Car Rental staff handles the car's inventory. They can add a vehicle, edit its details, accept or reject a ride request, update their company information, give offers to customers, and interact with them.

b. Functional Requirements

List down system requirements. You may group requirements according to actors or modules

#	Requirements
1	As a customer, I want the system to let me create an account with my email.
2	As a customer, I want to log in to my account.
3	As a customer, I want the system to send me a verification code, upon signup.

4	As a customer, I want the system to let me search for locations and tourist sites.
5	As a customer, I want the system to show me a detailed overview of the hospitals, hotels, rent-a-car centers, restaurants, and tourist points.
6	As a customer, I want the system to provide me with AI-based personalized suggestions for hotels, restaurants, and tourist attractions based on my preferences and previous searches.
7	As a customer, I want the system to show me live weather and help me navigate to my destination
8	As a customer, I want the system to link with my calendar and send me notifications reminding me about my hotel check-in time, restaurant reservations, and other trip-related events
9	As a customer, I want the system to let me reserve hotel rooms and have a secure payment method.
10	As a customer, I want the system to let me connect with rent-a-car centers.
11	As a customer, I want the system to show me live weather and help me navigate to my destination.
12	As a customer, I want the system to let me give reviews on hotels, hospitals, and rent-a-car centers.
13	As a hotel managing staff, I want the system to let me create an account for my hotel.
14	As a hotel managing staff, I want the system to let me connect with customers.
15	As a hotel managing staff, I want the system to let me update my hotel records and status.
16	As a hotel managing staff, I want the system to let me upload pictures and details of my hotel.
17	As a hotel managing staff, I want the system to show me the customer reviews for my hotel.
18	As an App Administrator, I want the system to let me update the backend code, with data remaining intact.
19	As an App Administrator, I want the system to use ML models to enhance its performance in multiple aspects

20	As an App Administrator, I want the system to let me block or delete user accounts (customer, hotel management, hospital management, and car vendors) in critical situations.
21	As an App Administrator, I want the system to use ML models to enhance its performance in multiple aspects.
22	As a Car rental company, I want the system to let me create an account for myCompany.
23	As a Car Rental Company, I want the system to let me connect with customers and receive ride/reservation requests.
24	As a Car Rental Company, I want the system to let me update my Company records/inventory and change the status of cars..
25	As a Car Rental Staff, I want the system to let me upload pictures and details of my Company and Cars.
26	As a Car Rental Staff, I want the system to show me the customer reviews for my Company.
27	As a car rental staff, I want the system to let me chat or contact customers.

c. Non-functional Requirements

List down non-functional requirements.

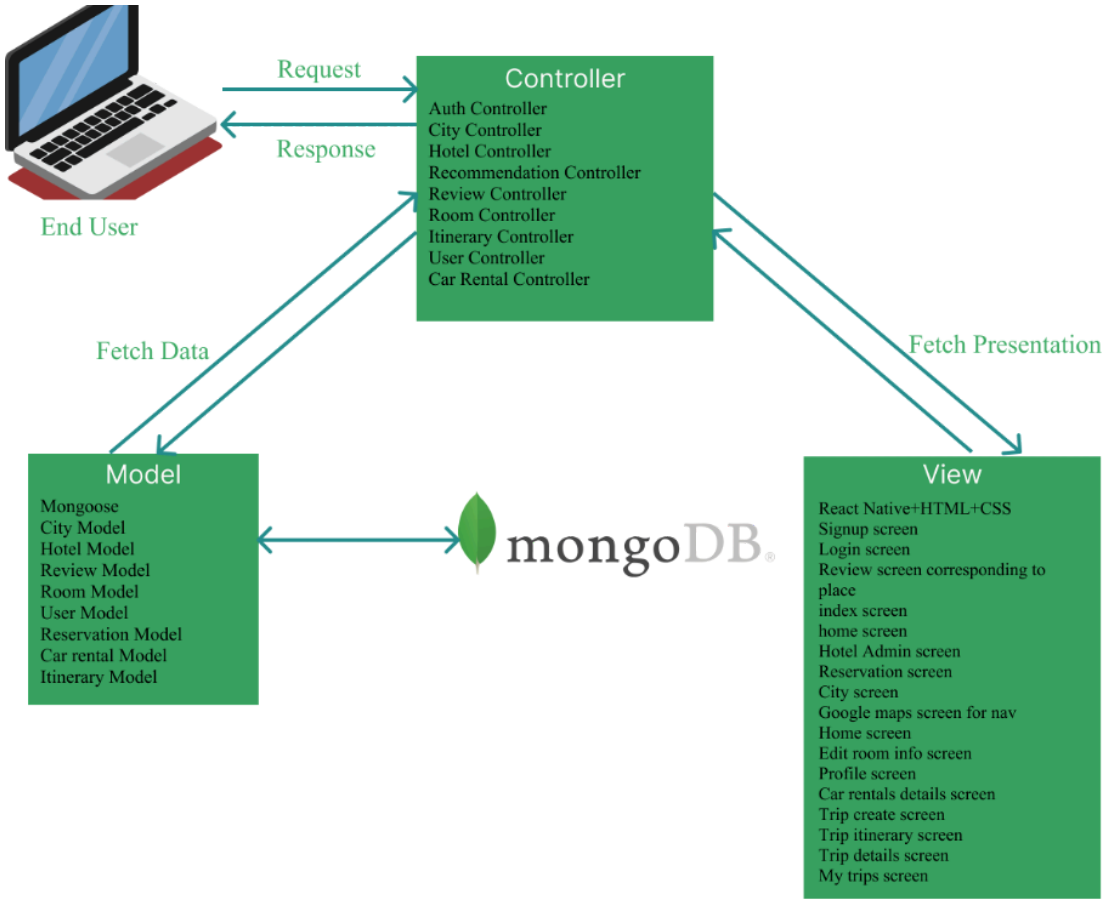
#	**Requirements**
1	The app should not use more than 500 MB of memory when multiple features like maps, weather updates, and hotel reservations are being used simultaneously.
2	The app should load the main dashboard within 3 seconds on a standard 4G network
3	The app should be able to support up to 200 users booking hotels or navigating to destinations simultaneously without slowing or crashing
4	The app should be compatible with mobile devices running Android version 8.0 or higher and IOS version 12.0 or higher
5	The app should ensure that all screens maintain a consistent response time of under 2 seconds with interacting with API calls
6	The app must be available 99.9% of the time over a period of 30 days, excluding planned maintenance
7	The system should recover and resume normal operations in less than 5 minutes in case of a failure
8	The app should display search results for tourist locations and services (e.g., hotels, restaurants) within 2 seconds after the user initiates a search.
9	The app should sync live weather and traffic updates within 2 minutes of change for any destination.
10	The app should provide feedback or error handling within 1 second of an invalid search query or failed booking attempt.
11	The app's user interface should remain responsive (i.e., no lag) even when navigating between different modules like hotel booking, car rentals, and live weather updates.

3. System Architecture

The **Model-View-Controller (MVC)** architecture we utilized in this project is a widely adopted software design pattern that promotes the separation of concerns within an application. It organizes code into three interconnected components:

1. **Model** handles the data,
2. **View** manages the user interface and presentation,
3. **Controller** acts as an intermediary, processing user input and coordinating between the Model and View.

a. Architecture Diagram



b. Architecture Description

Views (React Native Components)

1. **City Screen**: Allows users to view all the famous places in the city.
2. **Review Screen**: Displays and allows users to add destination reviews.
3. **Google Maps Screen**: Shows the map and route details using Google Maps API.
4. **Signup screen**: Allows users to sign up securely into the database.
5. **Login screen**: Allows login for registered users.
6. **Index screen**: Routes to either signup or login screen.
7. **Home** screen: Allows users to select a search bar from the hardcoded cities and features list.
8. **Hotel Admin screen:** This screen allows the hotel administrator to edit the details of hotels and approve bookings.
9. Reservation screen: Allows users to make bookings.
10. **Home screen:** replacement for city screen with better frontend.
11. **Edit Room info:** The screen is dedicated to updating hotel room information for hotel admins.
12. **Profile screen:** let the user edit their profile, view their reservations and reviews, and log out.
13. **Car rentals details screen:** lets users view details of what cars the rental has available and for what pricing and provides information regarding car rental places.
14. **Trip create screen:** allows users to create tailored trips.
15. **Trip itinerary screen:** allows users to view AI planned itinerary.
16. **Trip details screen:** allows users to view the formatted itinerary.
17. **My trips screen:** allows users to view a list of their itineraries.

Backend (Express.js Components)

1. **Routes**: Define API endpoints for communication with the front end.
    - /signup: User Signup.
    - /login: User login.
    - /getReviews
    - /postReviews
    - /getReviewsByEmail
    - /getCities
    - /getRecommendations
    - /getHotels
    - /postHotel
    - /getHotelsByCity
    - /getHotelById
    - /putHotel
    - /getRooms
    - /postRooms
    - /getRoomsByHotel
    - /putEditRoomInfo
    - /getSearchPlaces
    - /postReservations
    - /getAllReservationsByHotelId
    - /getReservationsByEmail
    - /putUpdateReservationsStatus
    - /getUser
    - /putUpdateProfile
    - /postSaveItinerary
    - /getUserItineraries
    - /getTestCarRentalEndpoint

- /getCarRentalCompaniesByCity
- /getCarRentalCompanyById
- /postCreateCarRentalCompany

2. **Controllers**:

Contains business logic.

a. **AuthController**: Handles user authentication and authorization.

b. **ReviewController**: Handles review creation and retrieval.

c. **CityController**: Handles cities retrieval.

d. **HotelController:** Handles hotel creation, retrieval of all hotels, retrieval of hotel through hotel name, and retrieval of the hotel through hotel admin name.

e. **RoomController:** Handles room creation and retrieval by hotel.

f. **RecommendationController:** Handles search and recommendations.

g. **Itinerary Controller:** creates and fetches user itineraries.

h. **User controller:** handles updating profile and fetching user parameters when necessary.

i. Car rental Controller: handles creation, fetch by city, and fetch by ID for car rental companies.

3. **Middleware**:

Intercepts requests for validation and security.

a. **AuthMiddleware**: Verifies authentication tokens.

4. **Database Layer**

- **User Model**: Stores user account information (username, email, password).

· **Review Model**: Stores user reviews (username, place name, comment, rating).

· **Hotel Model**: Stores hotel information (name, location, description, ref to rooms).

· **Room Model**: Stores room information (room type, price, duplicates, number booked, available, ref to hotel).

· **City Model**: Stores information about cities (name, country, description, places, food, photo URL, coordinates).

· **Reservation Model**: stores reservation information for rooms for hotel admin (roomId, customer information, etc.).

· **Car rental model**: stores information about the car rental and the cars they have available.

· **Itinerary model**: store the itinerary of the user.

External Services

1. **Google APIs**: Power navigation and location-based services. As well as search
2. **Gemini and OpenAI API**: AI trip planning.
3. **OpenWeatherAPI**: temperature updates.

c. Justification of the Architecture

Pros:

Separation of Concerns: The app's data handling (Model), UI (View), and logic (Controller) are cleanly separated, making it easy to manage growing features without cluttering the codebase.

Ease of Maintenance and Scalability: Future enhancements, like adding new screens or data models, can be integrated without rewriting major parts of the code. This fits well, as the app is expected to evolve over time.

Parallel Development: Frontend and backend components can be developed simultaneously, saving time. For instance, UI designers can build Views while backend developers work on Models.

Improved Testing: Unit testing becomes easier, especially for the Models and Controllers, ensuring that business logic works correctly even before the Views are finalized.

Reusability of Components: Some parts of the app (like user authentication logic or profile display views) can be reused in different sections without duplication, speeding up development.

Cons:

1. **Initial Complexity and Overhead:** Setting up MVC introduces more files and layers, which can seem unnecessary for simple features or at early stages when the app is small.
2. **Tight Coupling between Controller and View:** In some cases, Controllers become too dependent on specific Views, making it harder to switch UIs without refactoring.
3. **Boilerplate Code:** Managing separate files for Models, Views, and Controllers can result in a lot of boilerplate code, especially for smaller features that don't benefit much from complete MVC separation.

d. Tools and Technologies

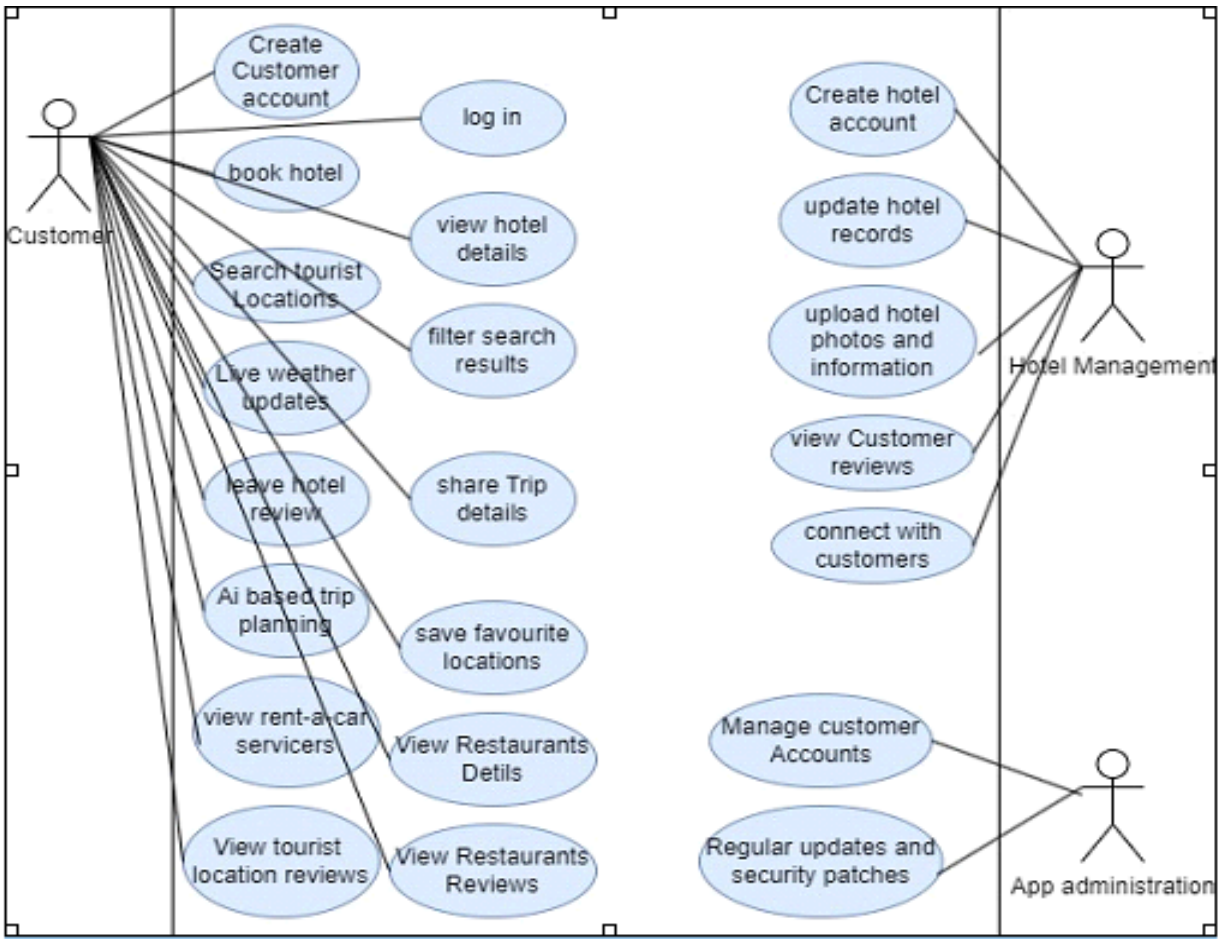
- **React Native/ Expo** (v0.71): For building cross-platform mobile applications.
- **Express** (v4.x): For developing backend REST APIs.
- **Node.js** (v18.x): As the runtime environment for the backend.

- **MongoDB** (v6.x): For database management.
- **Mongoose** (v6.x): For managing MongoDB through object modeling.
- **Amazon EC2:** Hosting services.
- **Gemini**: For itinerary planning.
- **Google APIs**: For map and search functionalities.
- **OpenWeather API:** This is for live weather updates.
- **GitHub:** This is for version control and collaboration.
- **GitHub Boards:** These are for project management and issue tracking.

4. Requirements Specifications

Our System involves 3 main users: primary users, hotel admins, car rental companies. Each of them have their own use cases, but the hotel admin and car rental companies resonate more in terms of their features. The application is basically designed to give the users an all in one platform to plan their trips smoothly. It includes features like online payment, navigation, weather track, hotel reservations, car booking, and contacting the hotel and car rental staff. Apart from these, we have included the other features such as reviewing the details of places, generating trip itineraries through AI integrated, checking g reviews and giving reviews etc.

a. Use Cases



Note: The Car rental company use case would be the same as that of the hotel admin. They would interact with customers and the system in the same way with similar use cases.

10 Core Use Cases:

UC-001

• Create account

Purpose: The customer creates an account to use the app's features.

Pre-conditions: The app must be installed on the customer's device, and the user must have a valid email address.

Post-conditions: The account is created successfully, and the customer is logged in.

Typical Course of Action

- The customer opens the app and selects the "Sign Up" option.
- The app prompts the customer to enter their email, name, and password.
- The customer enters the required information.
- The app sends a verification code to the provided email address.
- The customer enters the verification code.
- The system verifies the code, creates the account, and logs the customer in.
- The use case ends.

Alternate Courses of Action

- In step 5, if the customer does not receive the verification code, they can request the system to resend the code.
- The customer can cancel the account creation at any time and exit the process.

Step# Exception Paths

- In step 4, if the email is already registered, the app displays an error message and requests the customer to use a different email.
- In step 6, if the verification code is incorrect, the app displays an error and asks the customer to re-enter the code.

UC-002

- **Log in**

Purpose: The customer logs in to their account to access app features.

Pre-conditions: The user must already have an account with a valid email and password.

Post-conditions: The customer is logged in successfully.

Typical Course of Action

1. The customer opens the app and selects the "Log In" option.
2. The app prompts the customer to enter their email and password.
3. The customer enters their credentials.
4. The system checks the credentials and logs the customer in.
5. The use case ends.

Alternate Courses of Action

1. If the customer has forgotten their password, they can request a password reset link via email.

Exception Paths

1. If the email or password is incorrect, the system displays an error and asks the customer to retry.

UC-003

• Book hotel

Purpose: The customer books a hotel room for their stay via the app.

Pre-conditions: The customer must be logged into their account. Hotel information (availability, prices) must be up-to-date in the system.

Post-conditions: The hotel room is successfully reserved for the selected dates. The customer receives a booking confirmation.

Typical Course of Action

1. The customer navigates to the "Hotels" section in the app.

2. The customer searches for hotels based on location, check-in, and check-out dates.
3. The app displays a list of available hotels with details like pricing and room availability.
4. The customer selects a hotel and chooses the room type.
5. The app shows the booking summary, including dates, price, and room details.
6. The customer confirms the booking.
7. The system confirms the booking and sends a confirmation to the customer's email or within the app.
8. The use case ends.

Alternate Courses of Action

1. In step 4, if the selected hotel is fully booked, the app suggests alternative hotels in the same location or nearby.
2. The customer can choose to cancel the booking at any time.

Exception Paths

1. In step 8, if the booking cannot be confirmed due to server issues or hotel unavailability, the system notifies the customer and asks them to try again later.

UC-004

View hotel details

Purpose: The customer views detailed information about a hotel, such as room availability, amenities, reviews, and pricing.

Pre-conditions: The customer must be logged into their account. Hotel information (availability, prices) must be up-to-date in the system.

Post-conditions: The customer views the details of the selected hotel and can proceed with further actions, such as booking a room.

Typical Course of Action

1. The customer navigates to the "Hotels" section in the app.
2. The customer searches for hotels by entering a destination and dates.
3. The app displays a list of available hotels.
4. The customer selects a hotel from the list.
5. The app shows detailed information about the selected hotel, including room types and availability, hotel amenities, pricing for rooms, customers reviews, photos of hotels and rooms
6. The customer can scroll through the details or select a specific section to view (e.g., reviews or amenities).
7. The use case ends when the customer finishes viewing the details.
8. The use case ends.

Alternate Courses of Action

1. In step 4, if the hotel is not available or has no details, the app may display a message suggesting alternative hotels in the same location.

Exception Paths

1. In step 3, if no hotels are available for the selected dates or locations, the app displays a "No Results Found" message and offers suggestions for nearby locations or different dates.

UC-005

- **Search Tourist locations**

Purpose: The customer searches for tourist destinations and attractions within a specified location or radius.

Pre-conditions: The customer must be logged into their account. The system should have data on tourist locations.

Post-conditions: The customer is presented with a list of tourist locations matching their search criteria. The customer can select a tourist location to view more details.

Typical Course of Action

1. The customer navigates to the "Tourist Locations" section in the app.
2. The customer enters a destination or selects a location from a list (e.g., city, region).
3. The app retrieves and displays a list of tourist locations that match the search criteria.
4. The customer scrolls through the results and selects a tourist location.
5. The app shows details about the selected tourist location (e.g., description, pictures, reviews).
6. The use case ends when the customer finishes searching or viewing location details.

Alternate Courses of Action

1. The customer can choose to search by categories, such as "Top-rated," "Popular," or "Hidden Gems."

Exception Paths

1. In step 3, if no tourist locations are found based on the search criteria, the app displays a "No Results Found" message and suggests expanding the radius.

UC-006

• Filter search results

Purpose: The customer applies filters to narrow down the search results for hotels, restaurants, tourist locations, or services to meet specific preferences (e.g., price, rating, distance).

Pre-conditions: The customer must be logged into their account. The search results must be available based on initial criteria (e.g., location, dates).

Post-conditions: The customer is presented with filtered search results based on the applied filters.

Typical Course of Action

1. The customer conducts a basic search (e.g., hotels, restaurants, tourist locations) within the app.
2. The app presents an initial list of search results.
3. The customer selects the "Filters" option to refine the results.
4. The app displays a range of filtering options, such as:
   - Price range
   - Customer ratings
   - Distance from the current location
   - Amenities or services (e.g., free Wi-Fi, parking)
   - Categories (e.g., luxury, budget-friendly)
5. The customer selects the desired filters and applies them.
6. The app refreshes the search results based on the applied filters and displays the updated list.
7. The customer views the refined search results or selects a specific result to see more details.
8. The use case ends.

Alternate Courses of Action

1. In step 3, the customer can choose not to apply any filters and continue browsing the original search results. The customer can modify or remove filters at any time and refresh the search results accordingly.

Exception Paths

1. In step 6, if no search results match the applied filters, the app displays a "No Results Found" message and suggests relaxing the filters or changing the search criteria.

UC-007

• Live weather updates

Purpose: The customer receives real-time weather updates for a specific location or along a travel route to plan their journey effectively.

Pre-conditions: The customer must be logged into their account. The app should have access to real-time weather data sources. The customer should have been granted location access to the app (if required for weather updates along the travel route).

Post-conditions: The customer receives updated weather information for the selected location or travel route.

Typical Course of Action

1. The customer navigates to the "Weather" section in the app or selects a location for which they want to view the weather.
2. The customer can either enter a specific location (e.g., city, tourist site) or allow the app to use their current location for weather updates.
3. The app retrieves real-time weather data from external APIs and displays the current conditions, such as:
   - Temperature
   - Humidity
   - Wind speed
   - Weather forecast (e.g., rain, snow, sunny)
   - Alerts for extreme weather conditions
4. The customer can refresh the weather updates or enable auto-refresh to get updates every few minutes.
5. The customer views the weather information and makes decisions accordingly (e.g., continue traveling, delay journey).

6. The use case ends when the customer exits the weather section or navigates to another feature of the app.
7. The use case ends when the customer finishes viewing the details.

Alternate Courses of Action

1. In step 2, the customer can search for weather updates along a travel route (e.g., the weather at multiple points between two cities) instead of just a single location.

Exception Paths

1. In step 3, if there is no real-time data available for the selected location, the app displays a "Weather Information Unavailable" message and suggests trying again later.

UC-008

• Leave Hotel Review

Purpose: The customer provides feedback or a rating on a hotel based on their experience.

Pre-conditions: The customer must be logged into their account. The customer must have used or visited the service/destination they are reviewing.

Post-conditions: The review is saved and visible to other users. The hotel's overall rating is updated based on the new review.

Typical Course of Action

1. The customer navigates to the service or destination (e.g., hotel site) they want to review.
2. The customer selects the "Leave a Review" option.
3. The app prompts the customer to provide:

    • A rating (e.g., 1 to 5 stars)

    • A written review describing their experience

• Optional: Upload images related to their visit (e.g., photos of the hotel or restaurant)

4. The customer submits the review.
5. The app processes and saves the review, updating the overall rating of the hotel.
6. The review becomes visible to other users in the review section.
7. The customer can edit or delete their review later if desired.
8. The use case ends when the review is successfully submitted or saved.

Alternate Courses of Action

1. In step 4, the customer can choose to submit only a rating without writing a detailed review.

Exception Paths

1. In step 5, if there is an issue with the review submission (e.g., network issue), the app displays an error message and asks the customer to retry.

UC-009

• Share trip Details

Purpose: The customer shares their planned or completed trip details (e.g., destinations, accommodations, itinerary) with friends or family through social media, email, or messaging platforms.

Pre-conditions: The customer must be logged into their account. The customer must have created or completed a trip using the app's planning feature.

Post-conditions: The trip details are successfully shared through the selected medium (e.g., email, social media).

Typical Course of Action

1. The customer navigates to their planned or completed trip within the app.
2. The customer selects the "Share" option for the trip details.
3. The app prompts the customer to choose a sharing method, such as Email etc.
4. The customer selects their preferred sharing method and enters the recipient(s) or selects from their contacts.
5. The app generates a shareable link or attaches the trip details (e.g., PDF, itinerary summary) depending on the chosen platform.
6. The customer adds any optional message or note to accompany the trip details.
7. The customer confirms and sends the trip details.
8. The app shows a confirmation message that the trip details have been shared successfully.
9. The use case ends when the trip details are sent.

Alternate Courses of Action

1. In step 3, the customer can choose to share specific portions of the trip (e.g., hotel reservations or restaurant bookings) instead of the entire trip itinerary.

Exception Paths

1. In step 5, if the chosen sharing platform does not support the format (e.g., unsupported file type or link issue), the app displays an error message and suggests an alternative sharing method.

UC-010

• AI-based trip planning

Purpose: The customer uses the app's AI feature to generate a personalized trip plan based on their preferences, budget, and interests (e.g., sightseeing, adventure, relaxation).

Pre-conditions: The customer must be logged into their account. The customer must provide trip-related preferences, such as destination, dates, budget, and interests. The app must have access to AI models and external data (e.g., hotel availability, and weather conditions).

Post-conditions: A detailed and customized trip itinerary is generated and presented to the customer.

Typical Course of Action

1. The customer navigates to the "Trip Planning" section of the app.
2. The customer selects the "AI Trip Planner" option.
3. The app prompts the customer to provide trip details, such as:
   - Desired destination(s)
   - Dates of travel
   - Budget (e.g., per day or total)
   - Preferences (e.g., type of accommodation, activities, food preferences)
   - Mode of transportation (e.g., flight, rental car).
4. The customer submits the trip preferences to the AI engine.
5. The app processes the customer's inputs using machine learning algorithms and external data (e.g., hotel rates, and weather forecasts).
6. The AI-based system generates a personalized trip plan, including:
   - Suggested accommodations
   - Recommended tourist spots and activities
   - Estimated costs
   - Suggested dining options
   - Optimized travel routes and modes of transport
7. The app displays the trip plan for the customer to review.
8. The customer can:

- Save the trip plan for future reference
- Modify or edit the plan (e.g., change hotel, add or remove activities)
- Share the trip plan with others (via email or social media)

9. The use case ends when the customer finalizes or saves the trip plan.

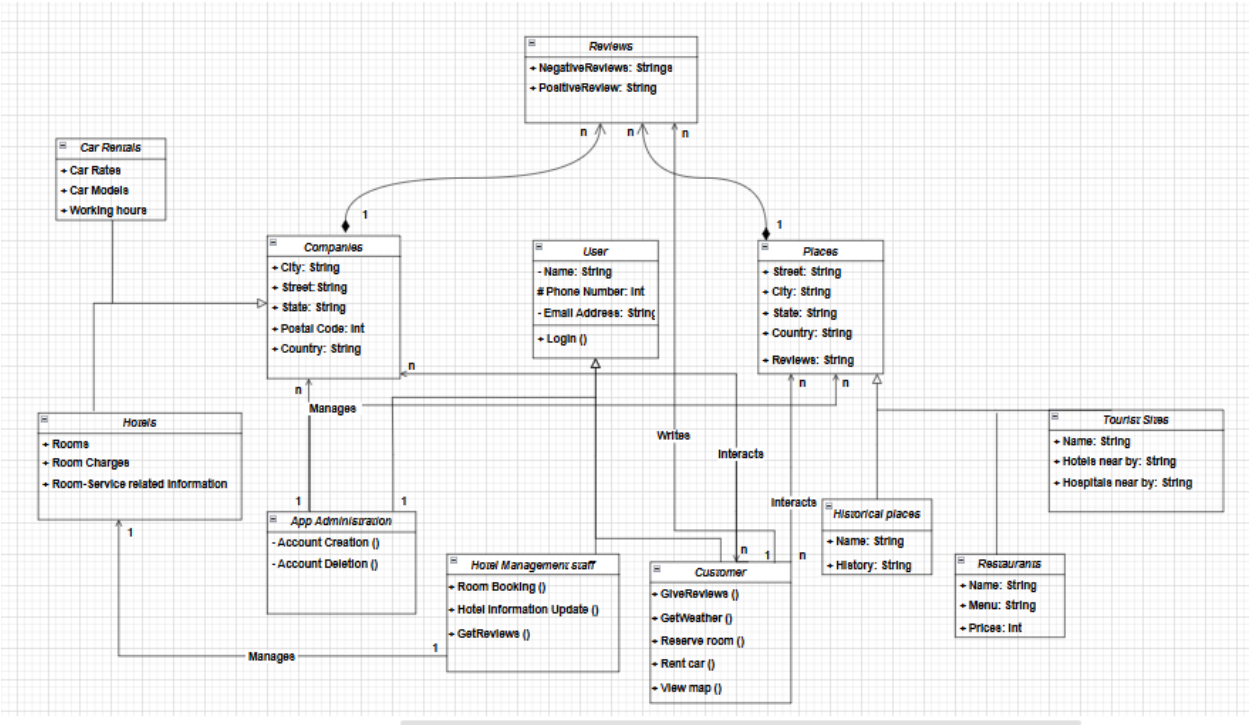
Alternate Courses of Action

1. In step 6, the customer can choose to refine the trip plan by adjusting preferences, such as increasing the budget or focusing on different types of activities (e.g., more sightseeing or adventure).

Exception Paths

1. In step 5, if the AI system cannot generate a trip plan due to missing data (e.g., no available hotels or transportation for the chosen dates), the app displays an error message and suggests alternate options or dates.

b. Class Diagram



Description

There are three major classes:

1. Users
2. Places
3. Companies

Each Major class will have their own child classes which will then have their specific roles.

There is another class called reviews, with which the customer would interact and write reviews of the companies (hotels, car rentals) and places.

- The class **USER** includes all those who will use the application, which includes the customer, the hotel management staff (responsible for managing their hotel updates) and the developers.
- The **CUSTOMER** is the main audience of the mobile application and will use the application for its main purpose. To make the communication smooth and facilitate the

customer, the relation between the customer and companies is bidirectional (n to n), i.e. the customer will be able to interact with the companies and vice versa.

1. The customer will also be able to interact with the class of the place, but this relation is unidirectional (n to n), and only the customer will be able to view the details of the places (historical places, restaurants, etc.)
2. The customer will also be able to write reviews about the hotels and places.

- The class **HOTEL MANAGEMENT STAFF** is also a child class of the user class. This is the management team of a hotel that will be responsible for managing their hotel updates on the application such as posting the latest pictures, room availability information, and other service information.
    1. It has a unidirectional 1 to 1 relation with the class **HOTELS.**

- The class **APP ADMINISTRATION/DEVELOPERS** is the child class of users responsible for managing the accounts and database of the application. They are also responsible for managing the bugs in the application and ensuring true and accurate information.
    - It has a unidirectional relation of 1 to n with the places and companies' classes.

- The class **COMPANIES** is another major class which includes the hotels and car rentals as its child classes.

    1. It has two child classes, i.e. **hotels** and **car rentals.**
    2. It has a unidirectional relation with the customer class, such that the customer will be able to book rooms at the hotels and rent cars from the car rental companies (child classes of the companies).
    3. It has a whole-part composition of 1 to n in relation to the review class, such that the class will hold reviews about hotels and car companies.
    4. The **HOTELS** child class of the **COMPANIES** class will have a 1 to 1 relationship with the **HOTEL MANAGEMENT STAFF** child class of the **USER** class.

- The class **PLACES** is the parent class of the places the customer wants to visit.

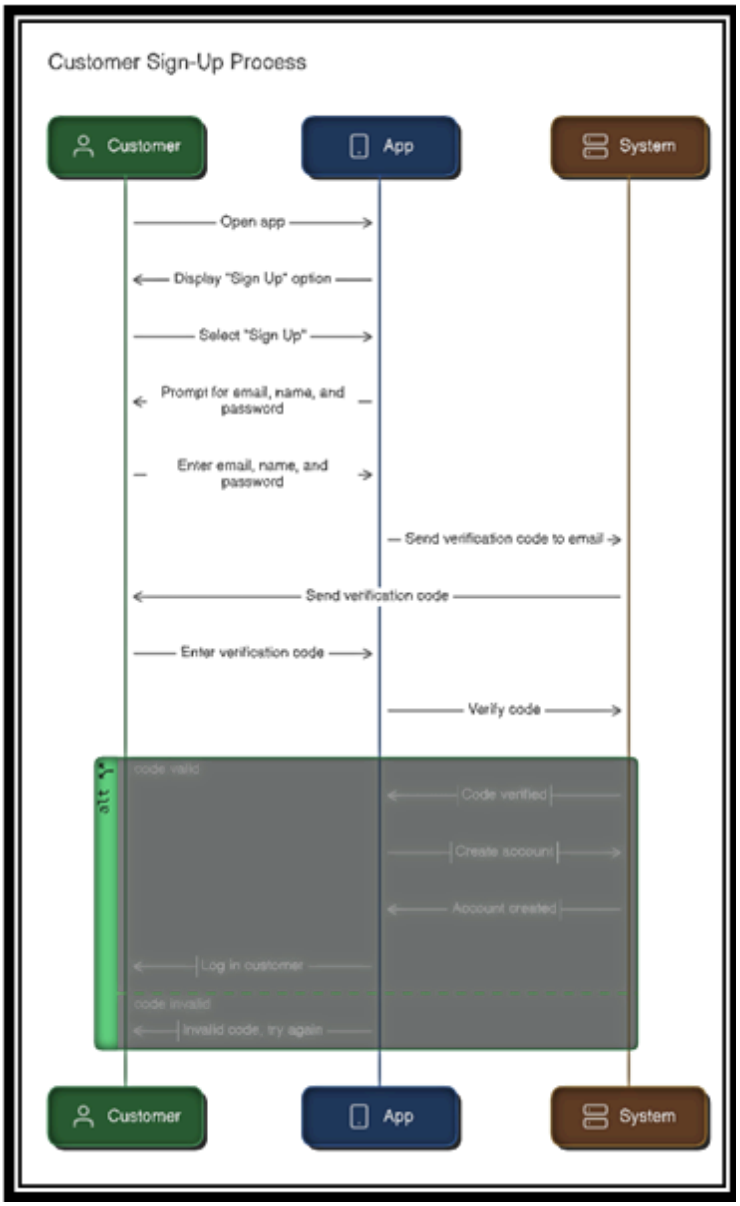
    1. It has three child classes, i.e. **Historical places, restaurants, and tourist sites.**
    2. This class has a unidirectional relation with the user's child classes, i.e. customer and developers.
    3. The customer is capable of searching for places and viewing the location and reviews of tourist sites, restaurants and historical places.
    4. The developers are capable of updating the class.
    5. This class also has a whole-part composition 1 to n relation with the review class, such that the class will hold reviews about the places.

- The **REVIEWS** class, as the name suggests will be keeping the reviews put on by the customers about the places, hotels and car rentals.

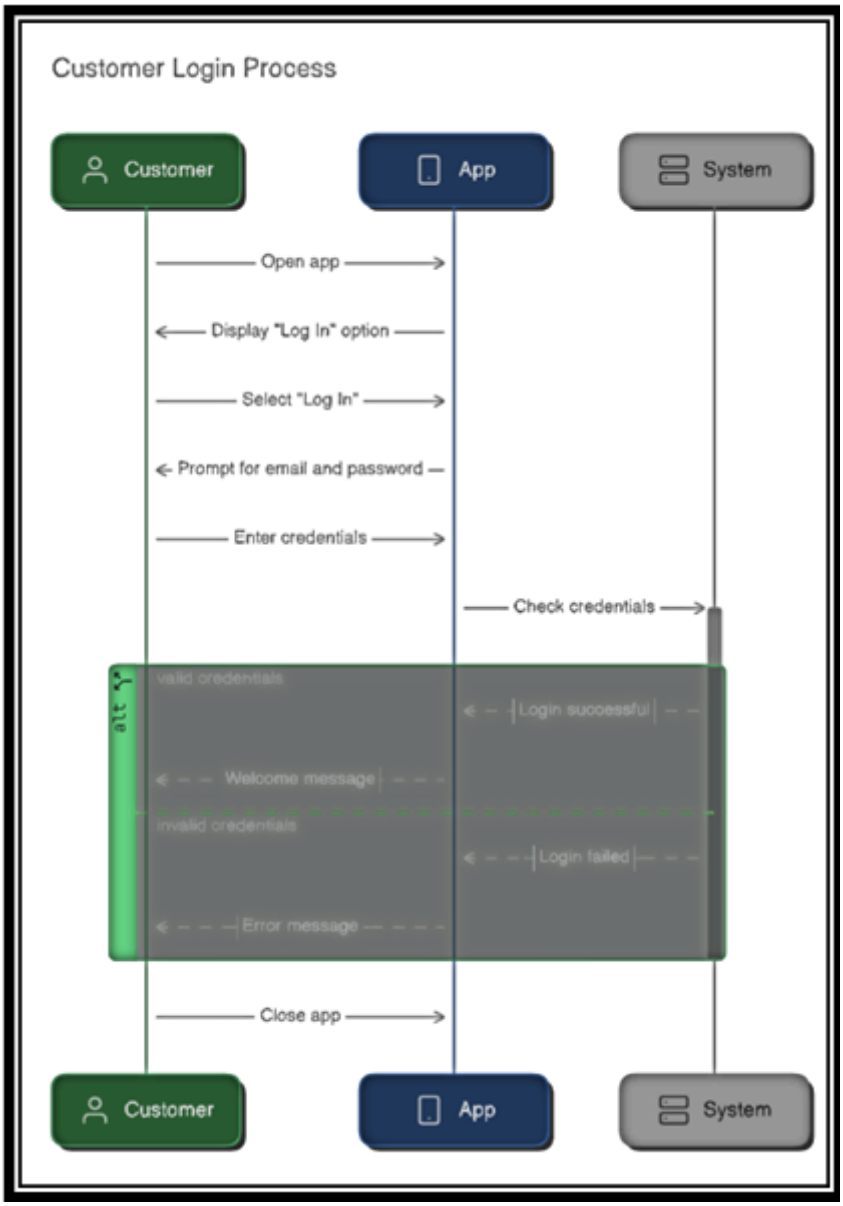
    1. It has a whole-part composition n to 1 relation with the companies and places, i.e. each place, hotel, or car rentals can have multiple reviews.
    2. It also has a n to 1 relation with the customers, i.e. one customer can write multiple reviews about the places and companies.

c. Sequence Diagrams

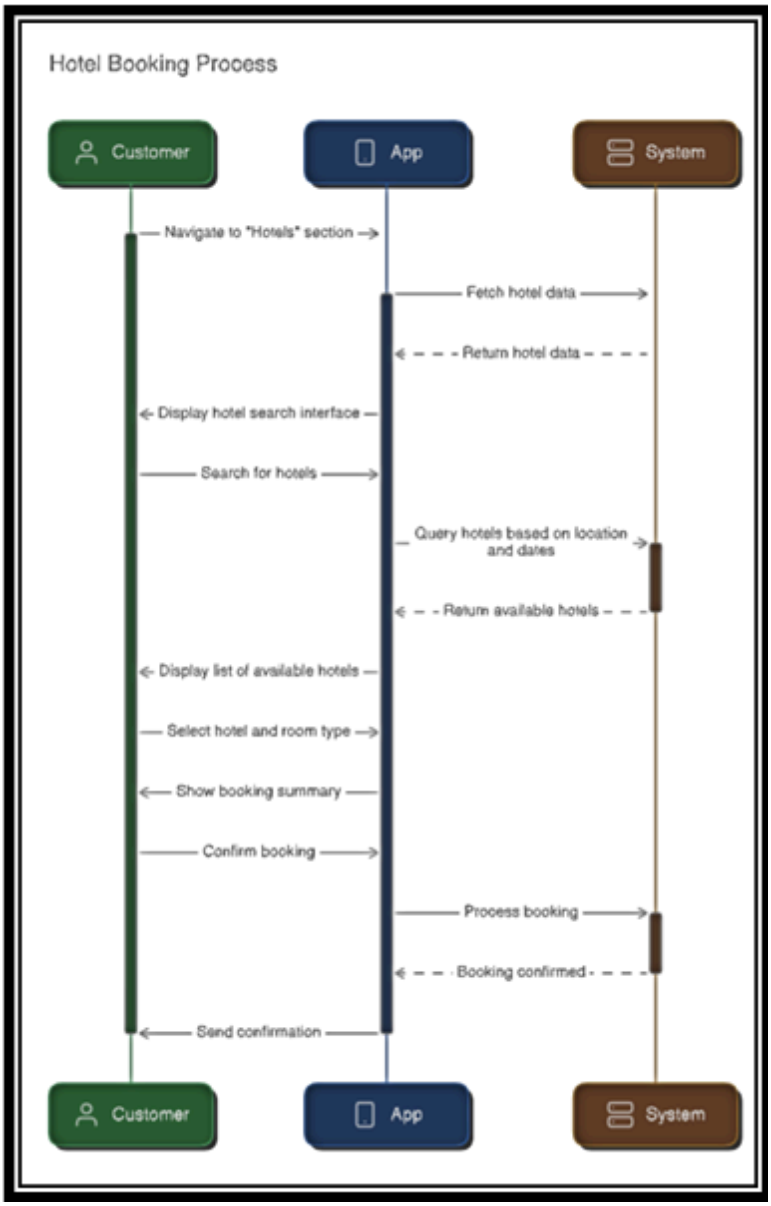
1. Create Account



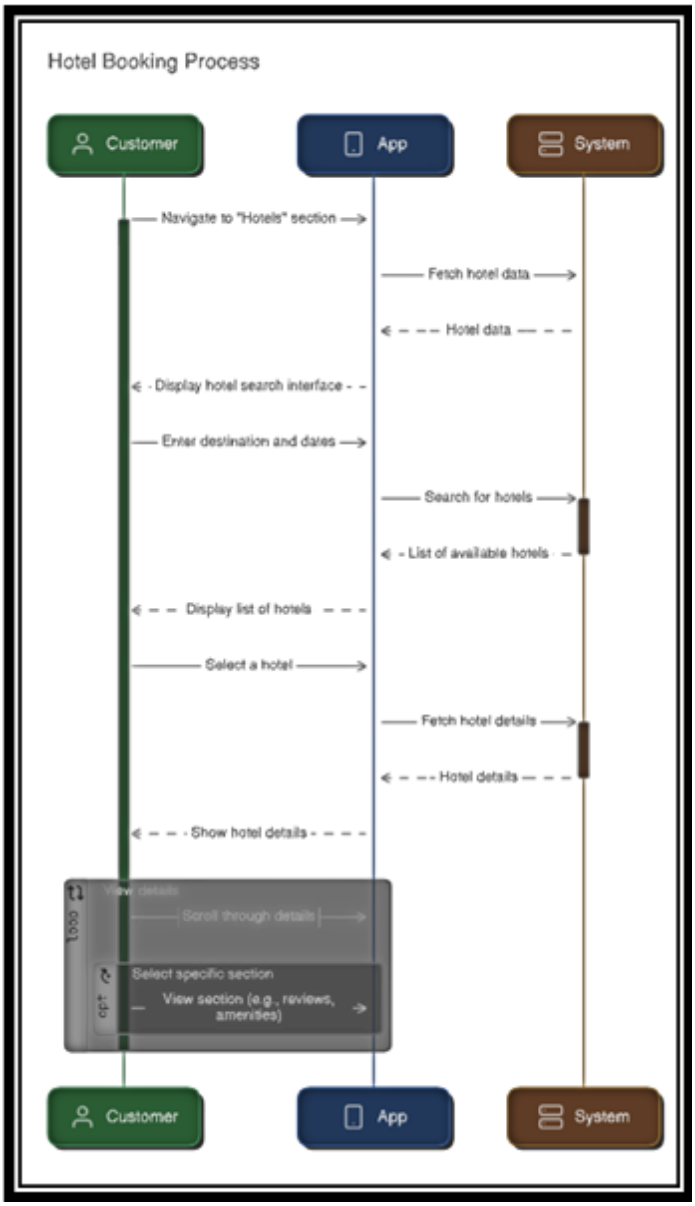
2. Logging in


Customer Login Process
Customer
App
System
Open app
Display "Log In" option
Select "Log In"
Prompt for email and password
Enter credentials
Check credentials
alt
valid credentials
Login successful
Welcome message
invalid credentials
Login failed
Error message
Close app
Customer
App
System

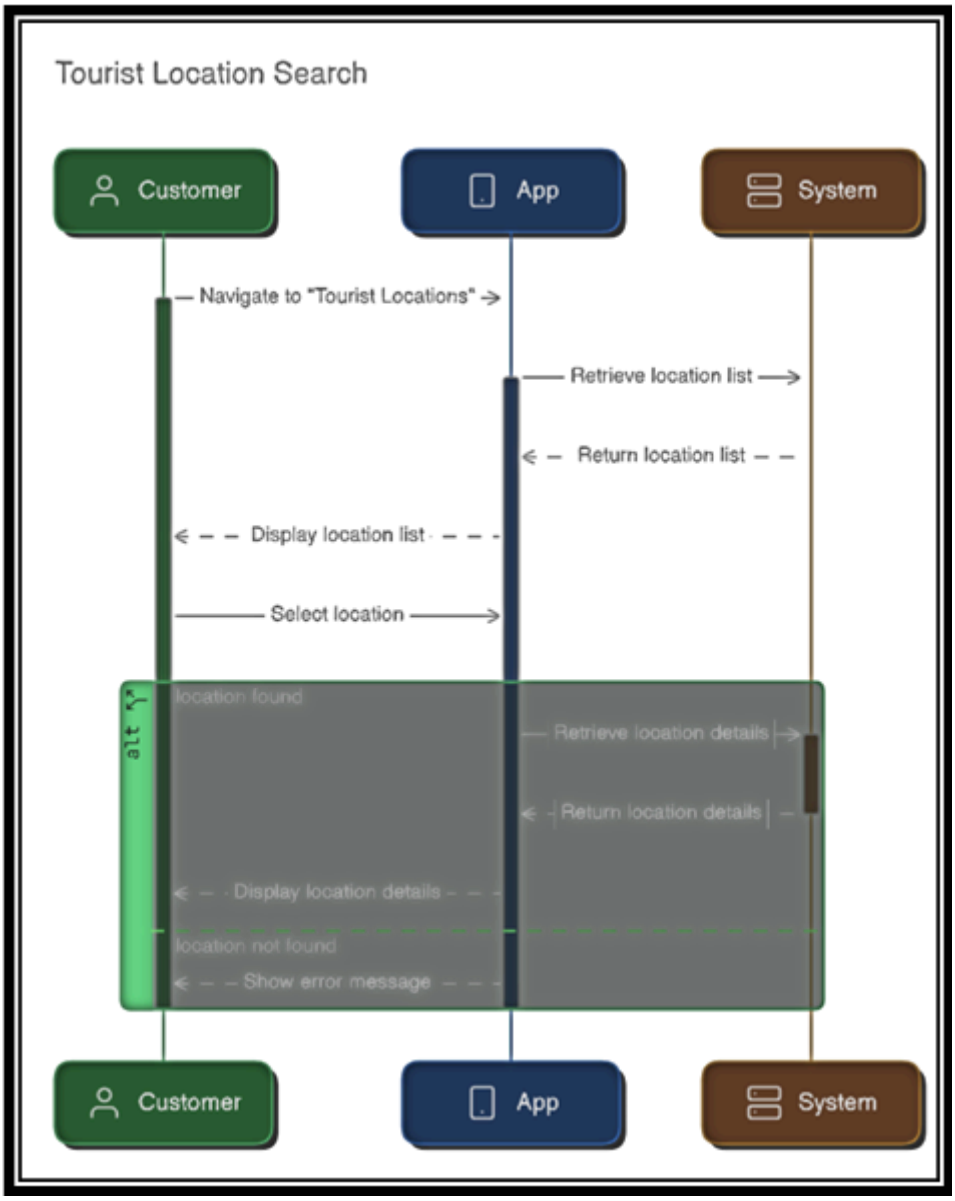
3. Book Hotel


Hotel Booking Process
Customer
App
System
Navigate to "Hotels" section
Fetch hotel data
Return hotel data
Display hotel search interface
Search for hotels
Query hotels based on location and dates
Return available hotels
Display list of available hotels
Select hotel and room type
Show booking summary
Confirm booking
Process booking
Booking confirmed
Send confirmation
Customer
App
System

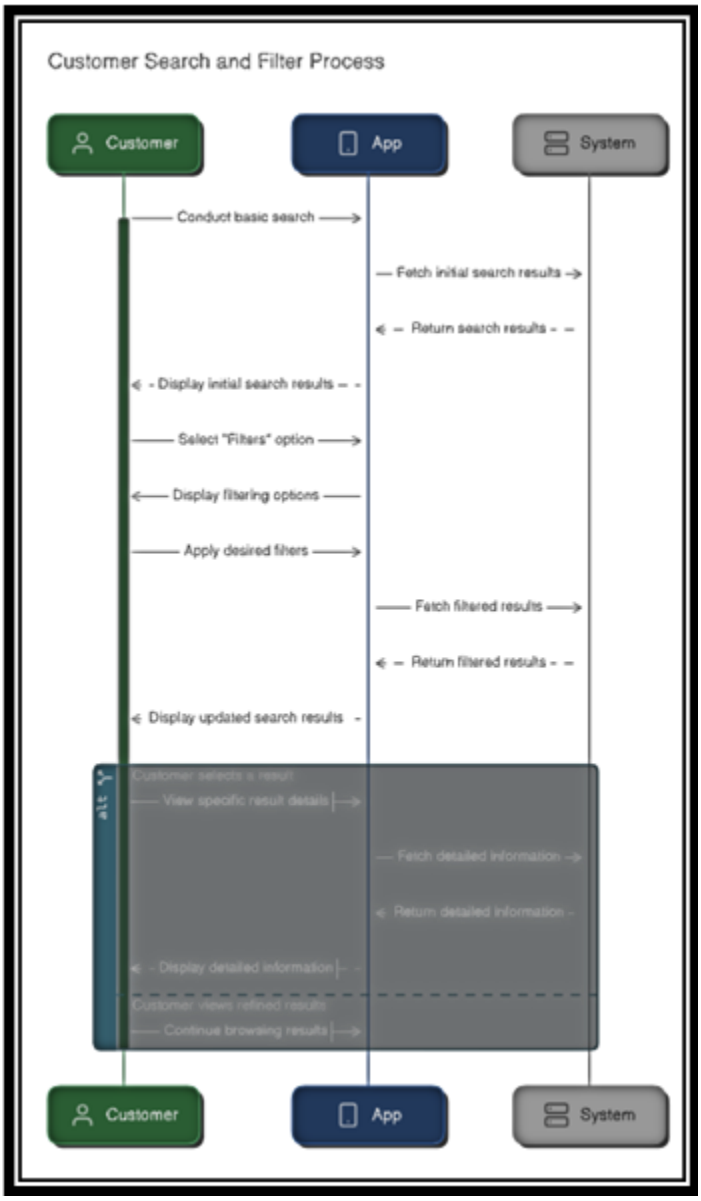
4. Hotel Details



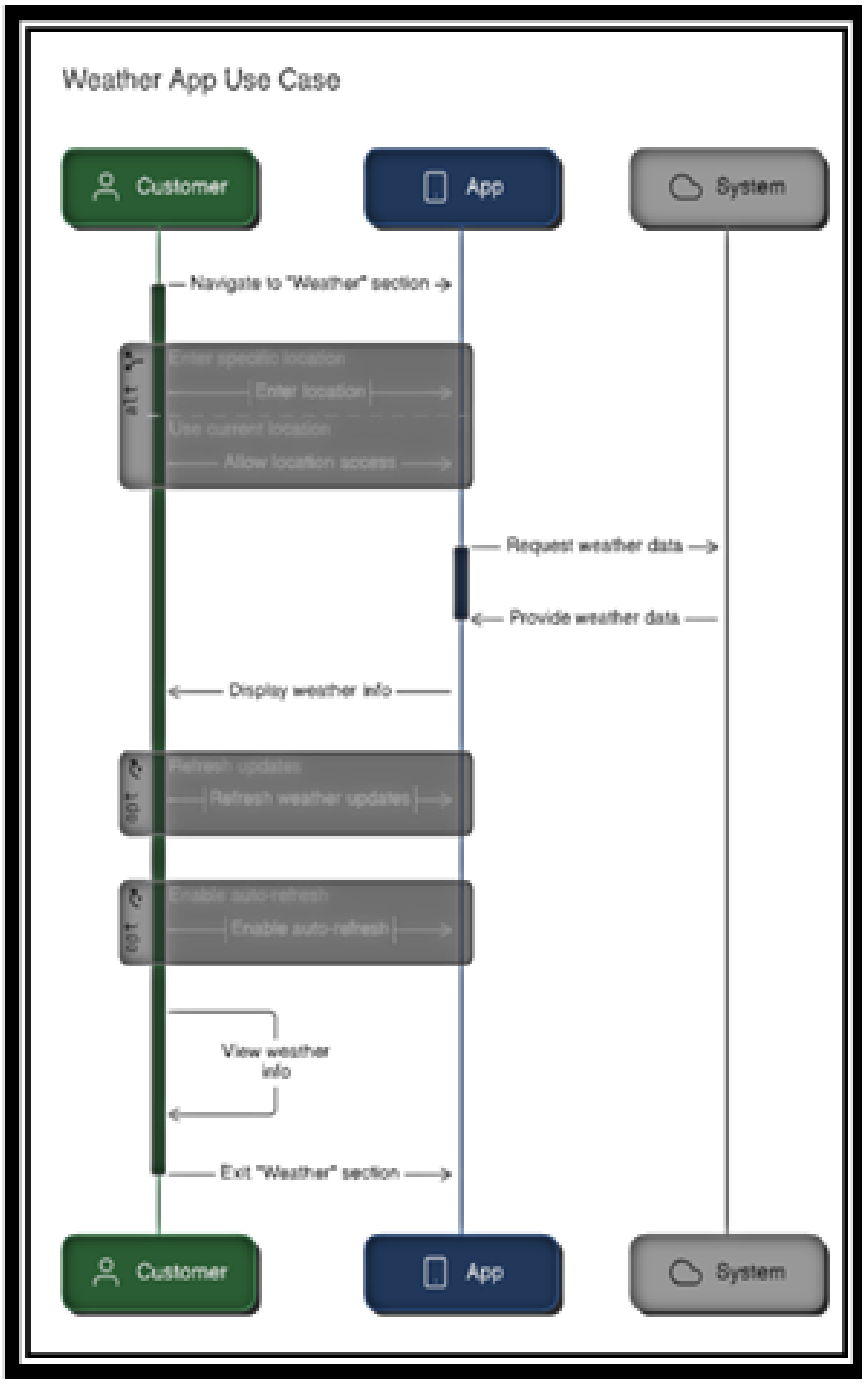
5. Search Tourist Locations



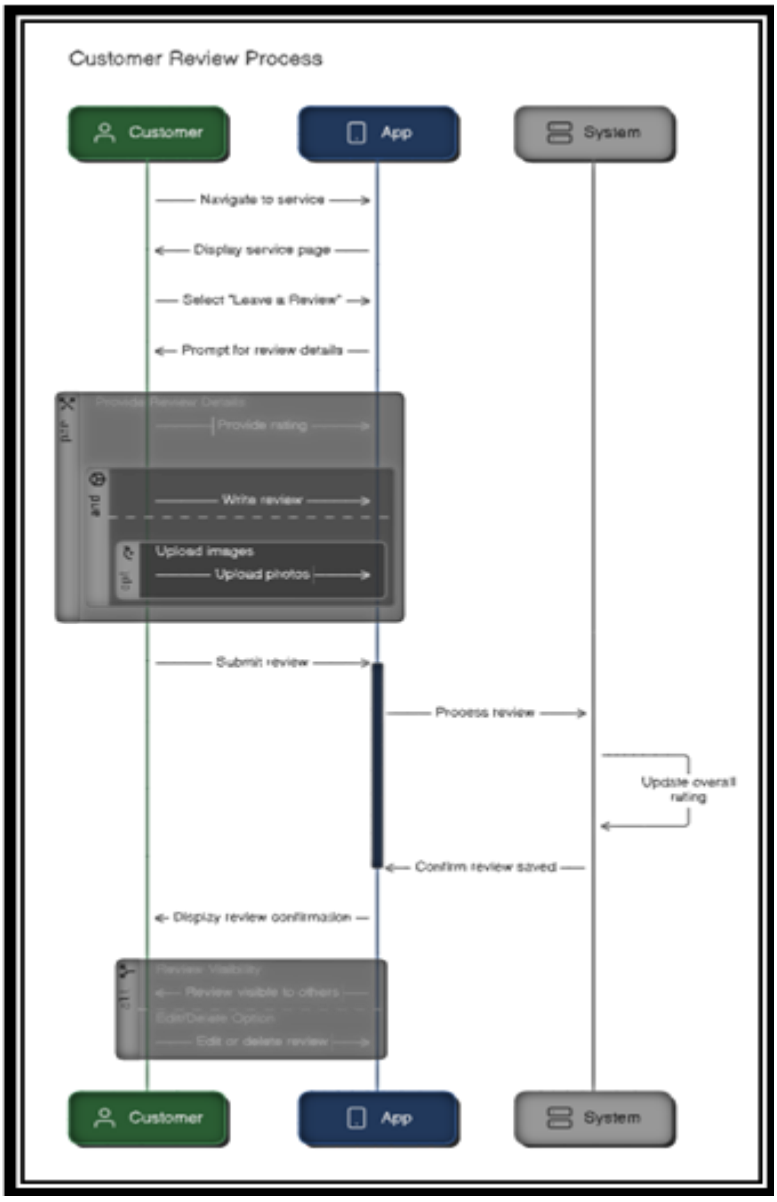
6. Filter Search Results



7. Live weather updates

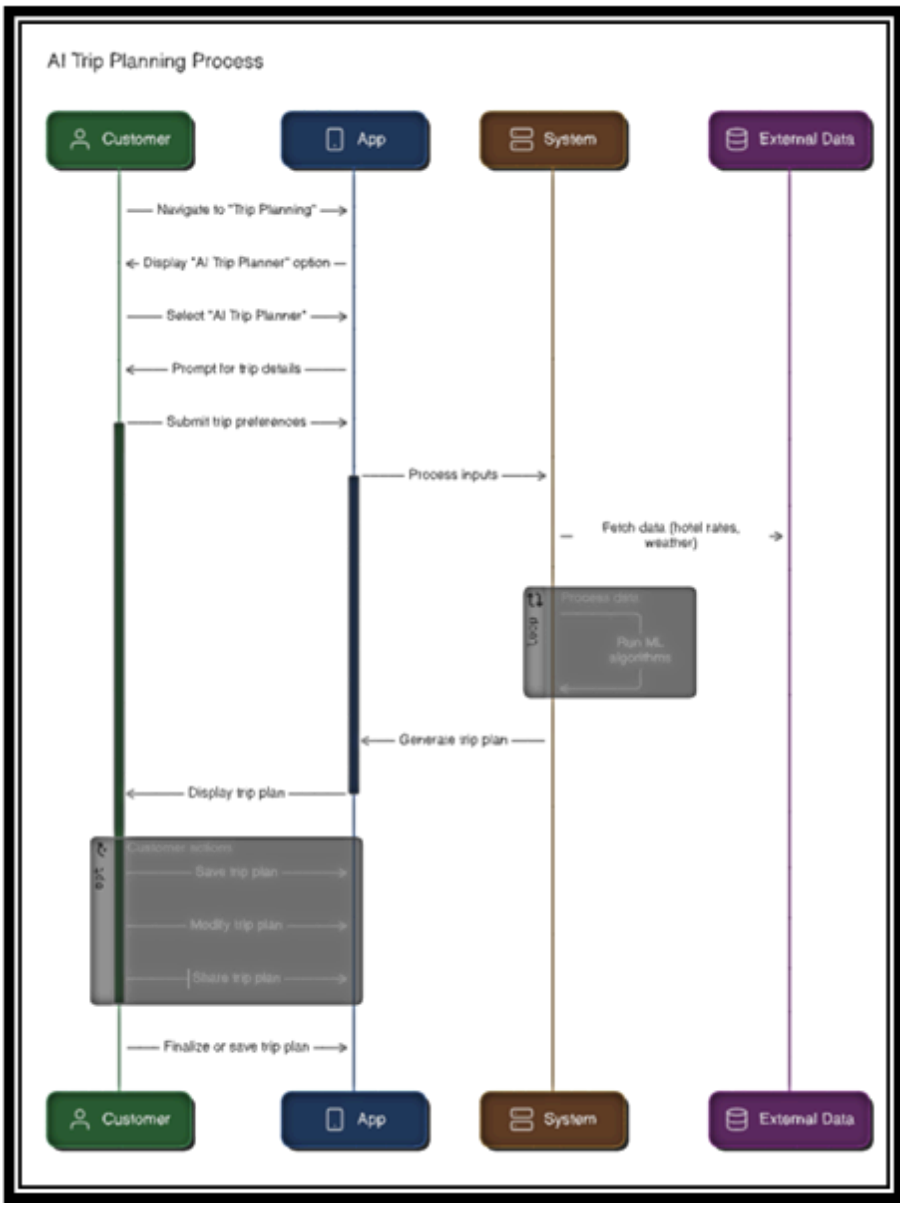


8. Leave Hotel Review



9. Share trip Details

10. AI based Trip Planning



5. Software Development Methodology and Plan

While developing the software, we kept several things in mind to choose the method of development for our project. Our main focus was to deliver software with more features resonating the users needs. As our systems had a wide range of use cases, implementing all of them to the user's requirements needed a plan which should allow the team to do changes as per the users needs.

a. Software Process Selection

A software development process is the set of structured steps involved in the development of any software. It is mainly a series of activities involving the following processes:

1. Specification – defining what the system should do.
2. Design - defining the organization of the system
3. Implementation – implementing/coding the system.
4. Validation – checking that it does what the customer wants.
5. Evolution – changing the system in response to changing customer needs.

In the process of selecting a software process, several things are required to be kept in consideration such as the adaptability of the system to changes in software requirements during the process, the feasibility of developers to work together and being in touch with other stakeholders, software release requirements etc.

Based on the above criteria, a software development process is selected.

Mainly 2 software development processes are used widely:

1. Waterfall Model
2. Agile (Scrum) Model

Each model has its own pros and cons, and based on the software requirements and planning strategies, a development model is selected.

Waterfall Model:

This process model of development involves a step-by-step strategy, where you move to the next step once a step is completed.

The steps involve the following:

1. Requirements analysis and definition
2. Software design
3. Implementation and unit testing
4. Integration and system testing
5. Operation and maintenance

This software model is applicable and best suited to the environment where there is a defined and stable set of system requirements, and the requirements are unlikely to be changed in future. However, it also has its own disadvantages, which makes it unsuitable for certain development environments.

Advantages:

1. **Structured Approach:** The Waterfall model follows a clear, sequential process, which makes it easy to understand and manage.
2. **Well-Defined Stages:** Each phase in the model has specific deliverables and a review process, which helps to maintain focus and accountability.
3. **Easy to Measure Progress:** Progress can be easily tracked as each phase is completed, making it simple to manage timelines.
4. **Ideal for Smaller Projects:** The model is effective for projects with well-defined requirements that are unlikely to change.

Disadvantages:

1. **Inflexibility:** The model works in phases which makes it quite inflexible to adapt to changes in the system requirement at later stages.
2. **Late Testing:** As the process works in phases, the testing phase happens at the end, which can cause extensive costs if bugs are detected.
3. **Limited Customer Involvement:** Stakeholders may have little input during the development process, which can result in a final product that does not meet expectations.
4. **Assumes Predictable Requirements:** Waterfall models is not adaptable for dynamic environments, where changes might evolve during the development phase.

Agile (Scrum) Model:

The Agile (Scrum) model is an iterative and incremental approach to software development that focuses on flexibility, collaboration, and customer feedback.it involves iterations called sprints to deliver functional increments of the product, which allows for continuous improvement and adaptation to changes in requirements.

Advantages:

1. **Flexibility and Adaptability:** This model allows for changes to be made at any stage of the project, which helps accommodate evolving requirements.
2. **Customer Collaboration:** All the stakeholders are kept in a loop during the development phase, and changes are made based on their input, which results in fulfilled customer expectations.

3. **Incremental Delivery:** Features are delivered in small and manageable increments, allowing for earlier detection of issues.
4. **Team Empowerment:** This model involves working in a group which promotes self-organizing teams, which can enhance motivation and creativity.
5. **Enhanced Communication:** Frequent meetings improve team communication and collaboration.

Disadvantages:

1. **Less Predictability:** Due to the flexibility, timelines and budgets can be harder to predict (New requirements can take more time and budget to be implemented), making planning difficult.
2. **Requires Cultural Shift:** Teams may struggle to adapt to Agile practices, especially in traditional environments.
3. **Scope Creep:** The openness to change can lead to continuous changes in scope, which may result in project overload.
4. **Documentation Challenges:** Agile often prioritizes working software over comprehensive documentation, which can create knowledge gaps.
5. **Dependency on Team Dynamics:** The success of Agile relies heavily on team members working well together and understanding Agile principles.

Agile (scrum) model for our project:

Having the pros and cons of the above process models in front, and analyzing the requirements and environment of our project, Agile (Scrum) model suits well in this scope. As the model provides the gap for changes in the requirements list, it will help us to better develop and incorporate changes at later stages in the development phase.

Reasoning:

While keeping the project context in mind, and the development environment in front, the following points resonate more to the Agile (Scrum) development model.

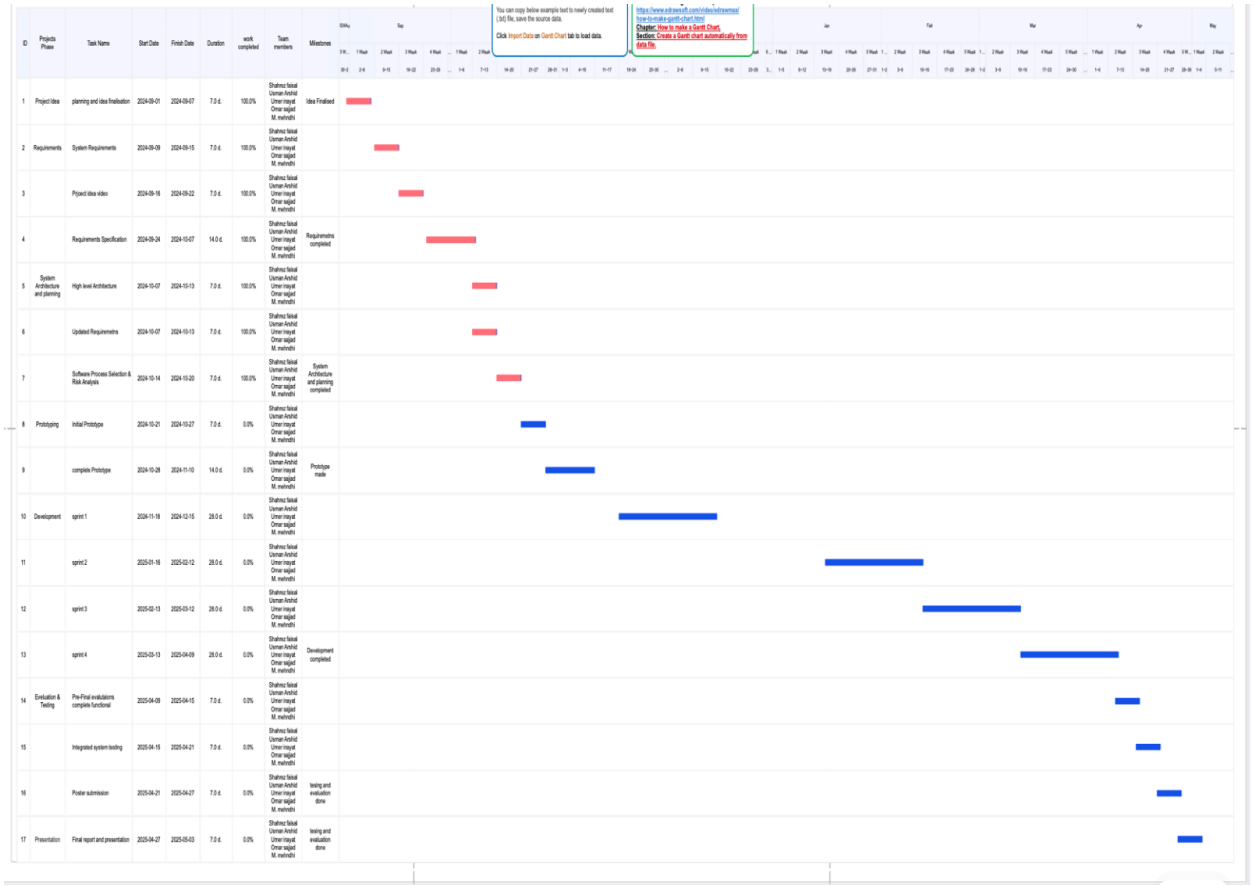
1. As our project has a limited budget and time constraints, there could be a high potential loss in case of bugs or issues found in the system. Thus, the Agile model suits best due to early testing features.
2. The development team is medium skilled developers but are capable of learning and incorporating changes effectively.
3. The team size is less (5), which makes it effective for meetings (Agile model).
4. The organization culture is adaptive to change which makes it suitable for the Agile model.
5. As the course project requires prototype releases, and the project needs to be tested well before the final deadlines, the agile model gives the space for early software release.
6. Also, the biweekly meetings with the project supervisors and keeping them in loop, helps the development team not lose track and continue smoothly.

The above points about the team environment and project requirements resonate with the agile development model, which makes it best suited for the project scope

b. Gantt Chart

Draw a Gantt chart that illustrates your project's schedule. The Gantt chart should show at least the following.

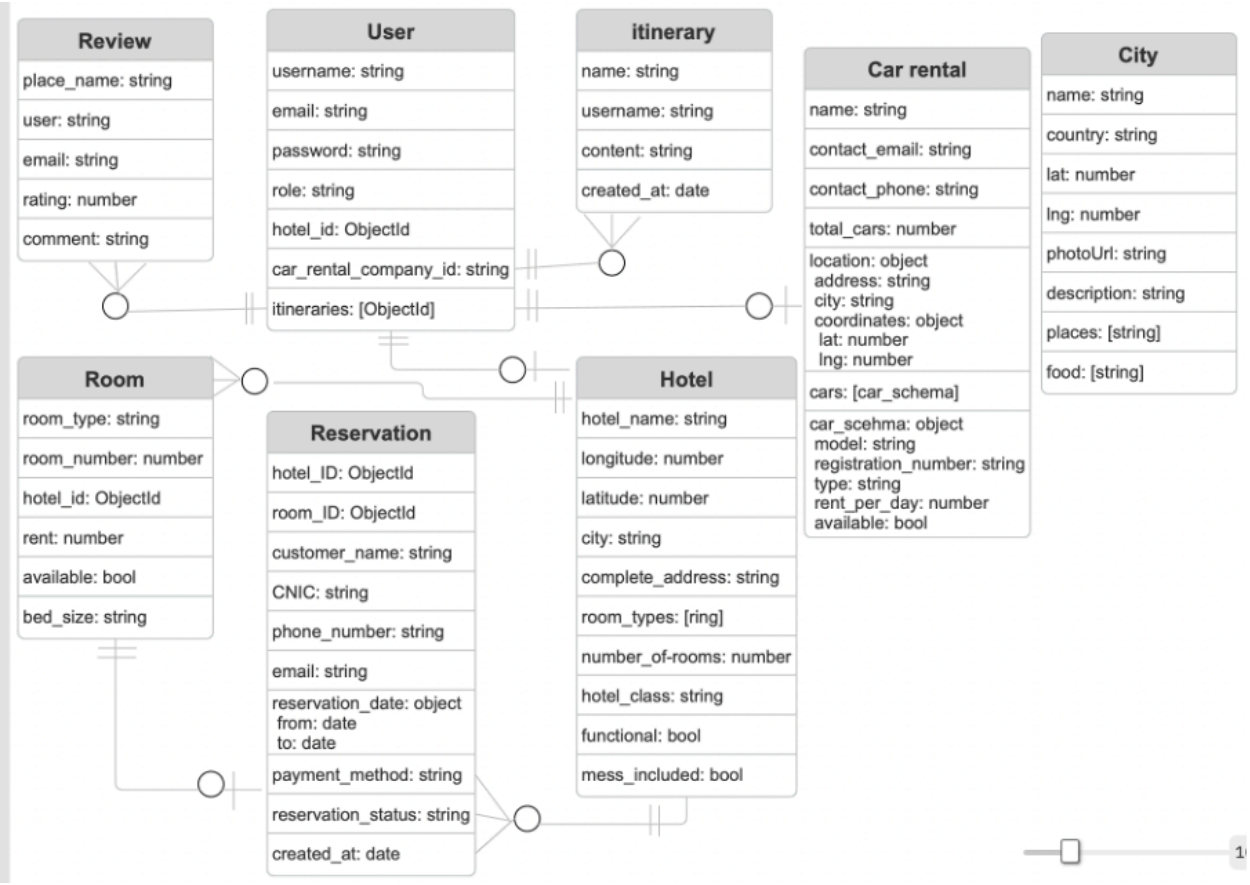
- Tasks (tasks should not be too small or too large)
- Duration (in weeks)
- Milestones
- Team members who have worked on each task.



6. Database Design and Web Services

We used **MongoDB**, a popular NoSQL database that stores data in a flexible, JSON-like format called documents. Unlike traditional relational databases, MongoDB does not require a fixed schema, making it ideal for applications that handle diverse or evolving data structures. Along with this, we used **Mongoose** to make our models more readable.

a. Database Design



User: stores information about each user in the system.

- **Username**: unique identifier for the user.
- **Email**: email address for login, notifications, and profile reference.
- **Password**: stores the encrypted password for authentication.
- **Role**: defines the user's role (e.g., admin, customer, hotel manager).
- **Hotel_id**: if applicable, links the user to a specific hotel (e.g., hotel manager).
- **Car_rental_company_id**: links the user to a car rental company, if applicable.
- **Itineraries**: list of itinerary object IDs created by the user.

Itinerary: user-created travel plans or schedules.

- **Name**: title or name for the itinerary.
- **Username**: links the itinerary to the user who created it.
- **Content**: details or description of the itinerary.
- **Created_at**: timestamp of when the itinerary was created.

Car rental: stores information about car rental companies.

- **Name**: name of the car rental company.
- **Contact_email**: email for communication.
- **Contact_phone**: contact number for the rental service.
- **Total_cars**: total number of cars the company owns.
- **Location**: full address of the rental company.
- **City**: the city in which the rental company is located.
- **Coordinates**: geographic latitude and longitude of the rental company.
- **Cars**: array of car objects with model, type, and availability details.

Car schema (within Car rental): describes each available car.

- **Model**: model name of the car.
- **Registration_number**: car registration plate/number.
- **Type**: type/category of car (e.g., sedan, SUV).
- **Rent_per_day**: cost of renting the car per day.
- **Available**: Boolean indicates if the car is available.

City: Data about various cities is available in the app.

- **Name**: name of the city.
- **Country**: the country the city is located in.
- **Lat**: latitude coordinate of the city.
- **Lng**: longitude coordinate of the city.
- **PhotoUrl**: link to an image representing the city.
- **Description**: brief overview or details about the city.
- **Places**: list of notable places to visit in the city.
- **Food**: list of popular food items or cuisines in the city.

Room: defines the properties of rooms in a hotel.

- **Room_type**: category of room (e.g., deluxe, standard).
- **Room_number**: a unique number used to identify each room.
- **Hotel_id**: foreign key linking the room to a hotel.
- **Rent**: price for renting the room.
- **Available**: Boolean will check if the room is currently available.
- **Bed-size**: bed size (e.g., queen, king, twin).

Reservation: stores details of room bookings made by users.

- **Hotel_ID**: references the hotel being booked.
- **Room_ID**: references the specific room being reserved.
- **Customer_name**: name of the user who made the reservation.
- **CNIC**: user's identity number (e.g., for verification).
- **Phone_number**: contact number of the customer.
- **Email**: customer's email address.
- **Reservation_date**: date range for the reservation (from and to).
- **Payment_method**: the method used for payment (e.g., card, cash).
- **Reservation_status**: current status (e.g., confirmed, canceled).
- **Created_at**: the date when the reservation was made.

Hotel: information about hotels listed in the system.

- **Hotel_name**: name of the hotel.
- **Longitude**: longitude coordinates for the hotel.
- **Latitude**: latitude coordinate for the hotel.
- **City**: the city where the hotel is located.
- **Complete_address**: full address of the hotel.
- **Room_types**: array of room types available in the hotel.
- **Number_of_rooms**: total number of rooms in the hotel.
- **Hotel_class**: star rating or hotel class.
- **Functional**: Boolean will indicate if the hotel is active/functional.
- **Mess_included**: indicates whether meals/mess are included in the booking.

Review: Review left by the user at a destination or place.

b. **Place_name**: to store and display which destination the review was left on.
c. **User**: to store and display the user who made the review.
d. **Email**: The user's email fetches or filters reviews by user profile.
e. **Rating**: out of 5 rating assigned by the user to the place.
f. **Comment**: written feedback or remarks left by the user.

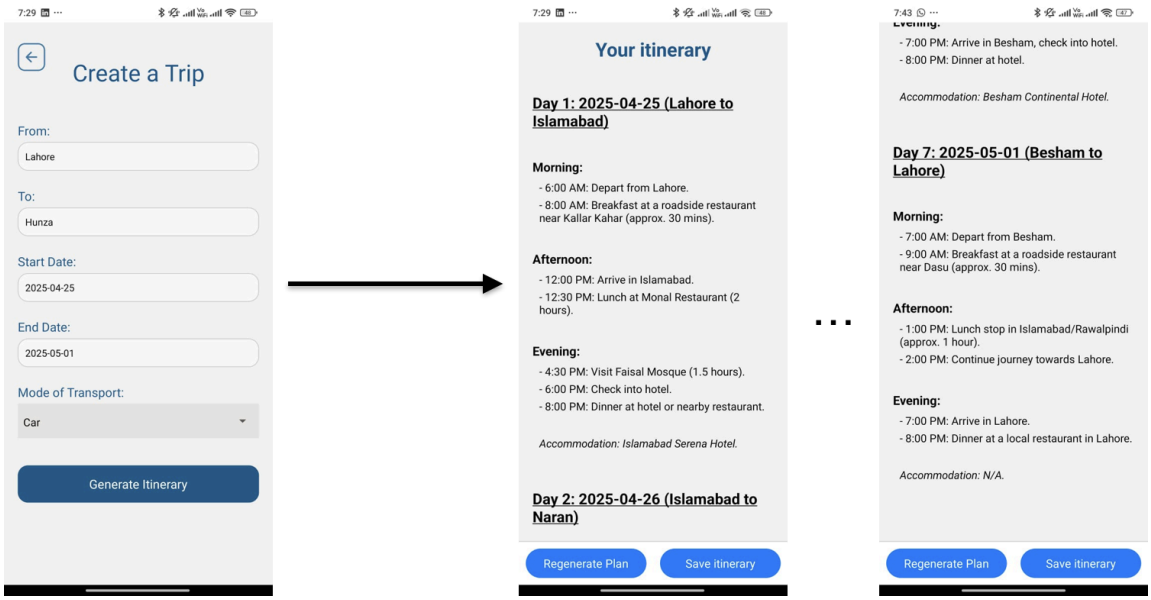
g. API Specification

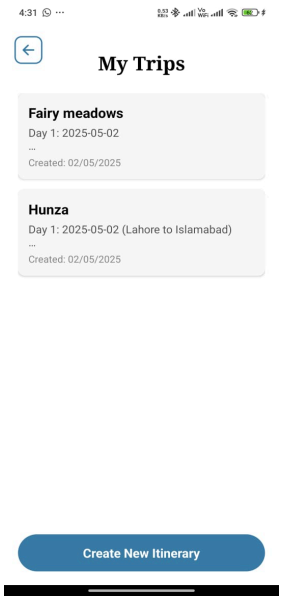
- **Gemini**: For itinerary planning.
- **Google APIs**: For map and search functionalities.
- **OpenWeather API:** This is for live weather updates.

7. System User Interface

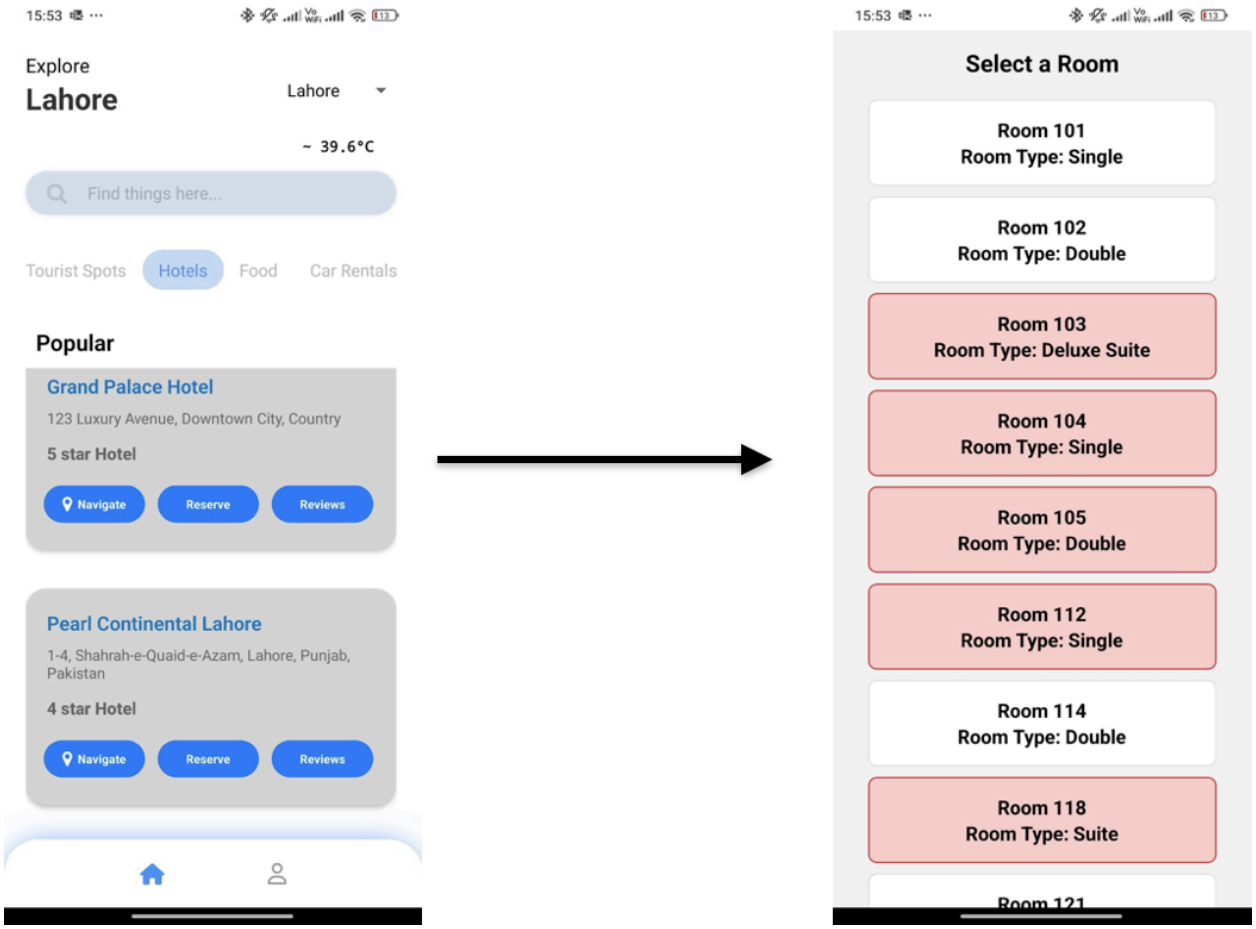
This section showcases the user interface for our main functionality pages. The total developed pages include the ones for the android Manzil app as well as the desktop web applications for hotel and car rental admins.

AI Trip planning

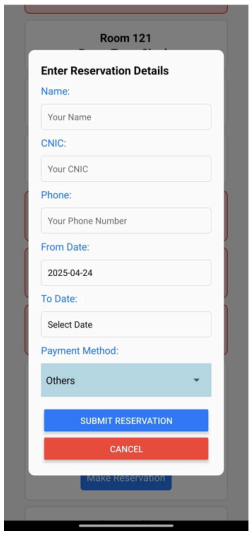


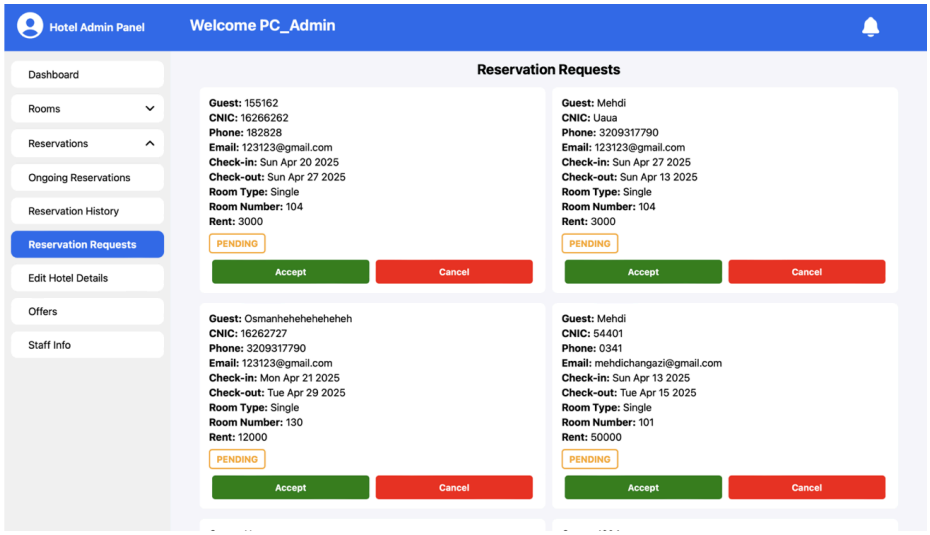


Hotels and hotel rooms

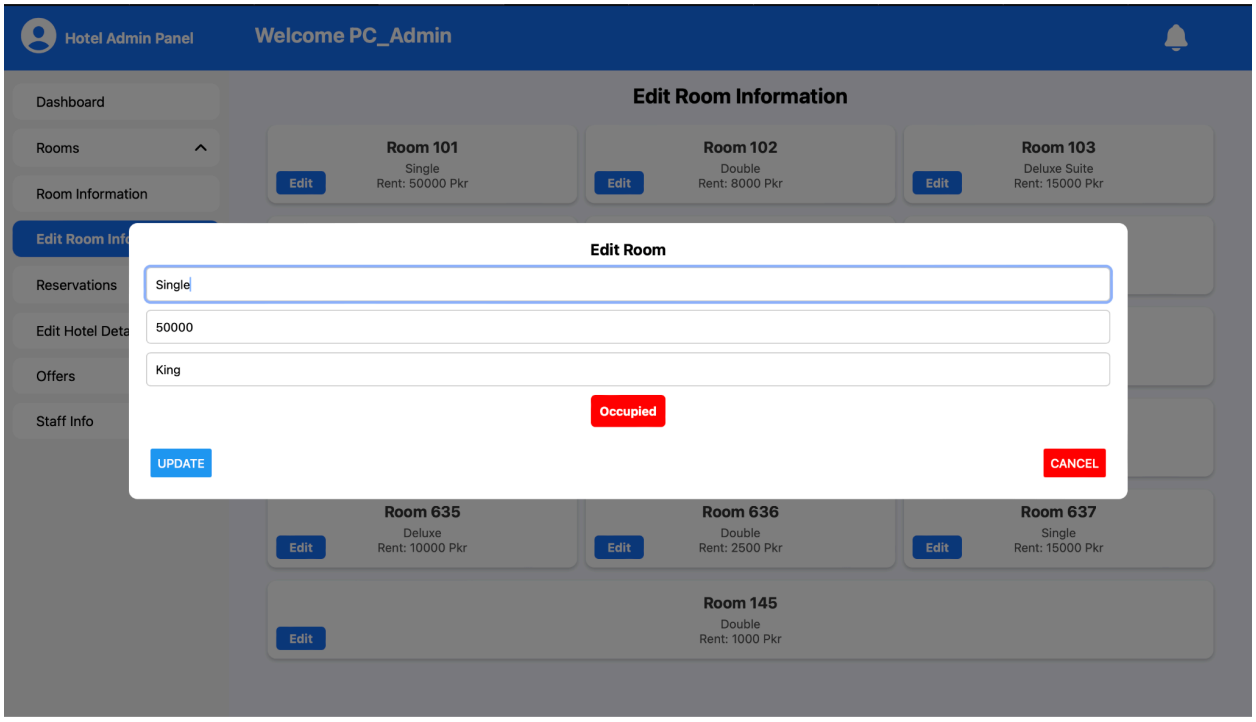


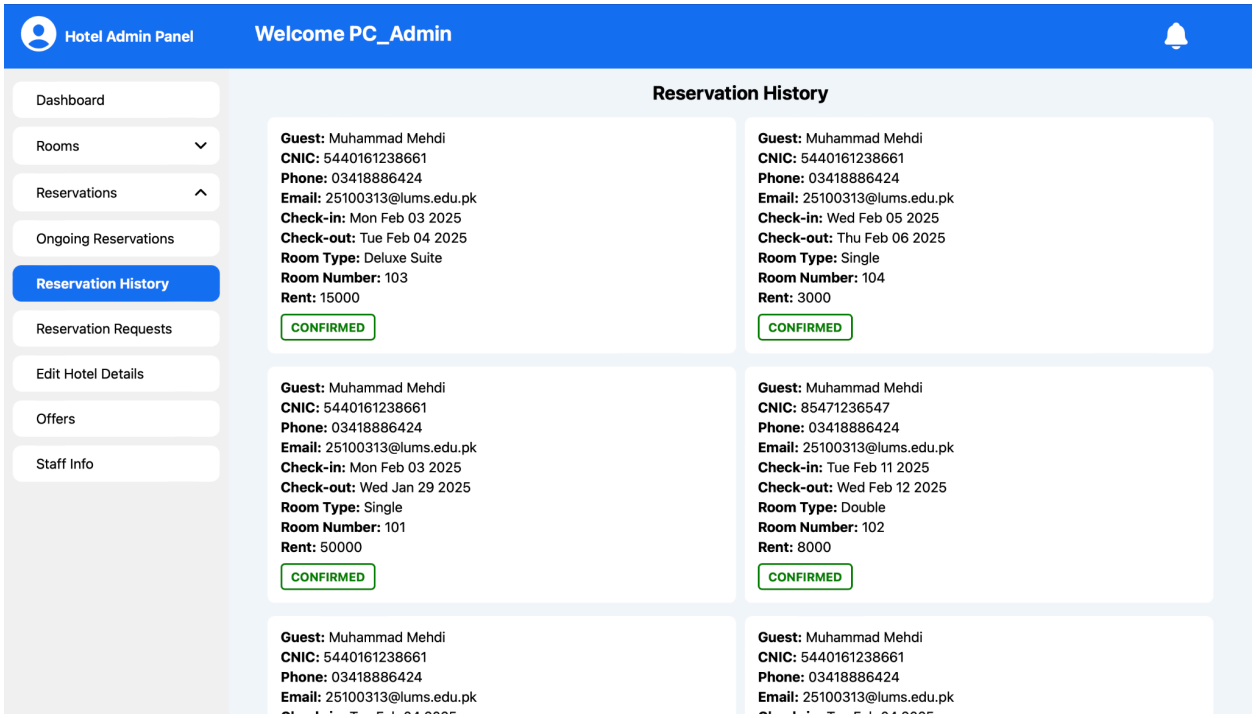
Hotel reservation on user end and admin reservation requests tab

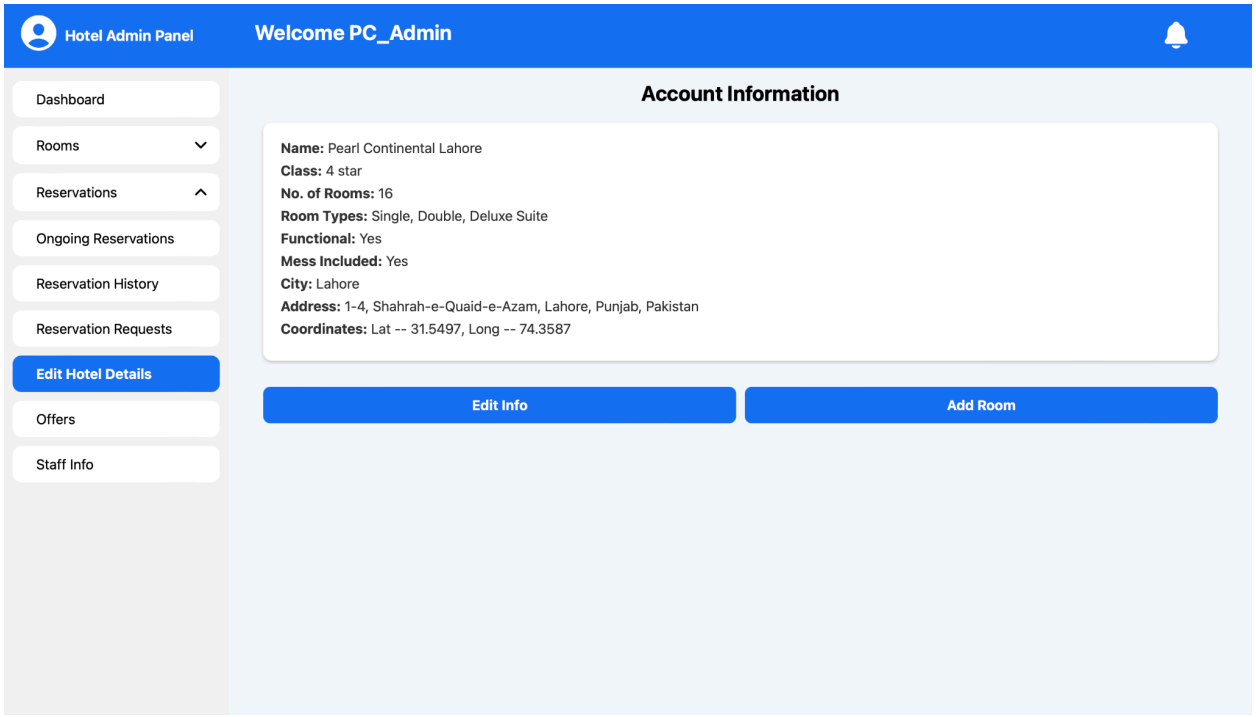




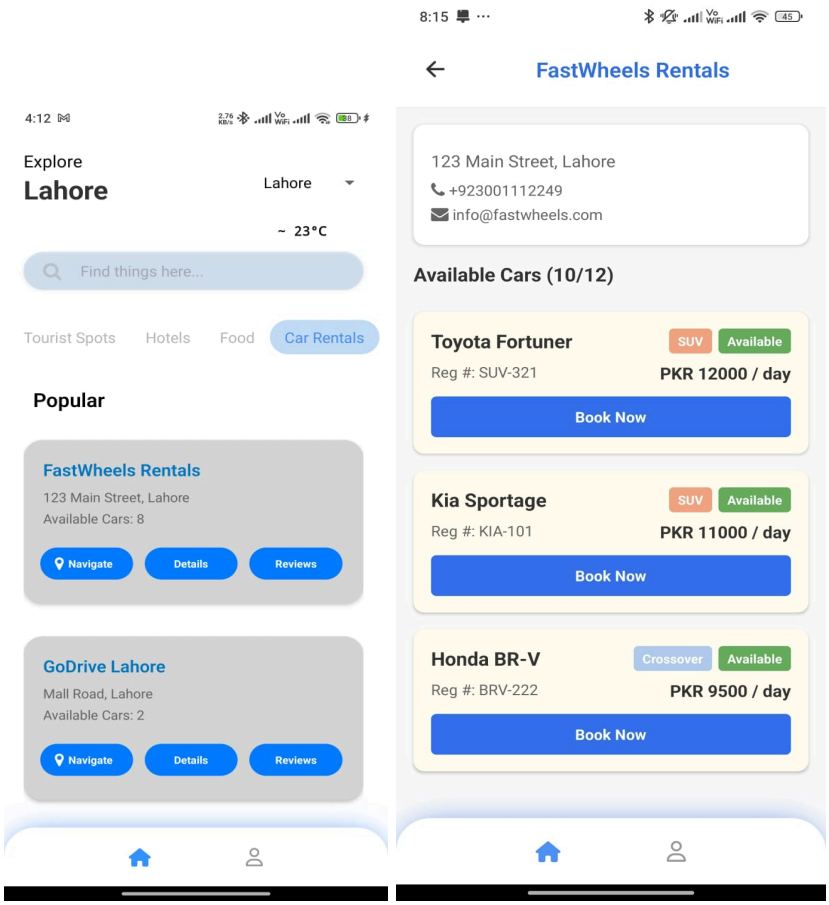
hotel admin portal




Hotel Admin Panel
Welcome PC_Admin
Dashboard
Rooms
Reservations
Ongoing Reservations
Reservation History
Reservation Requests
Edit Hotel Details
Offers
Staff Info
Reservation History
Guest: Muhammad Mehdi
CNIC: 5440161238661
Phone: 03418886424
Email: 25100313@lums.edu.pk
Check-in: Mon Feb 03 2025
Check-out: Tue Feb 04 2025
Room Type: Deluxe Suite
Room Number: 103
Rent: 15000
CONFIRMED
Guest: Muhammad Mehdi
CNIC: 5440161238661
Phone: 03418886424
Email: 25100313@lums.edu.pk
Check-in: Wed Feb 05 2025
Check-out: Thu Feb 06 2025
Room Type: Single
Room Number: 104
Rent: 3000
CONFIRMED
Guest: Muhammad Mehdi
CNIC: 5440161238661
Phone: 03418886424
Email: 25100313@lums.edu.pk
Check-in: Mon Feb 03 2025
Check-out: Wed Jan 29 2025
Room Type: Single
Room Number: 101
Rent: 50000
CONFIRMED
Guest: Muhammad Mehdi
CNIC: 85471236547
Phone: 03418886424
Email: 25100313@lums.edu.pk
Check-in: Tue Feb 11 2025
Check-out: Wed Feb 12 2025
Room Type: Double
Room Number: 102
Rent: 8000
CONFIRMED
Guest: Muhammad Mehdi
CNIC: 5440161238661
Phone: 03418886424
Email: 25100313@lums.edu.pk
Guest: Muhammad Mehdi
CNIC: 5440161238661
Phone: 03418886424
Email: 25100313@lums.edu.pk

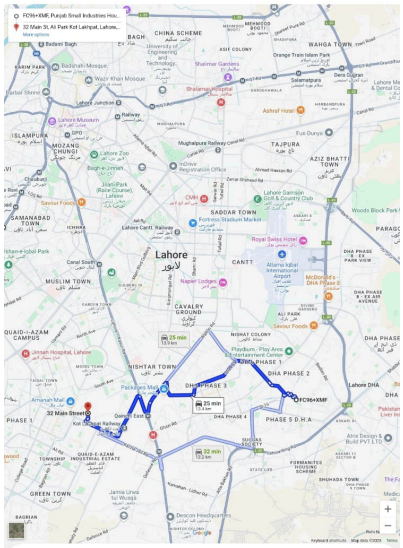

Hotel Admin Panel
Welcome PC_Admin
Dashboard
Rooms
Reservations
Ongoing Reservations
Reservation History
Reservation Requests
Edit Hotel Details
Offers
Staff Info
Account Information
Name: Pearl Continental Lahore
Class: 4 star
No. of Rooms: 16
Room Types: Single, Double, Deluxe Suite
Functional: Yes
Mess Included: Yes
City: Lahore
Address: 1-4, Shahrah-e-Quaid-e-Azam, Lahore, Punjab, Pakistan
Coordinates: Lat -- 31.5497, Long -- 74.3587
Edit Info
Add Room

Car rentals list and car rental page



(similar car rental admin portal as hotels)

Navigation page

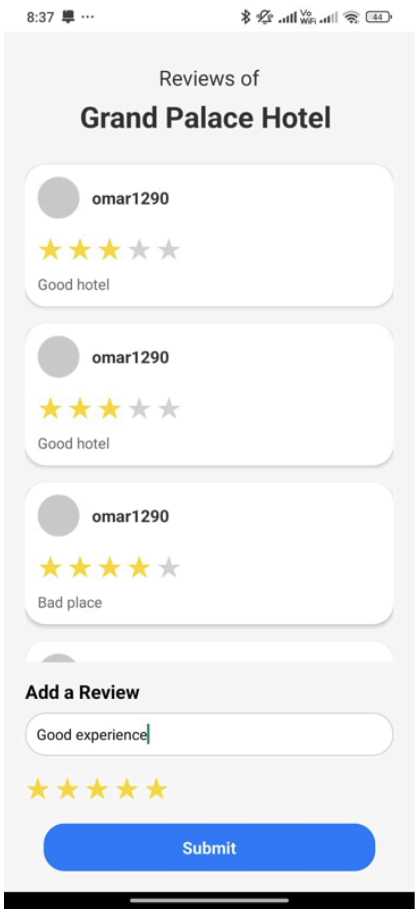


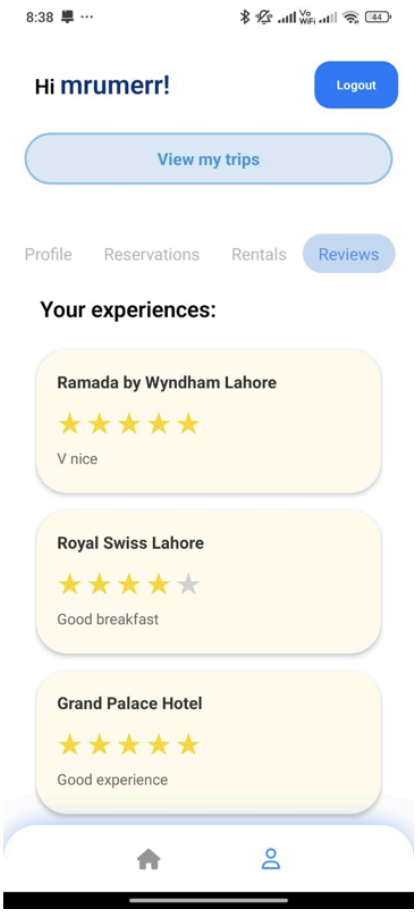
FastWheels Rentals 123 Main Street Lahore

Your Location

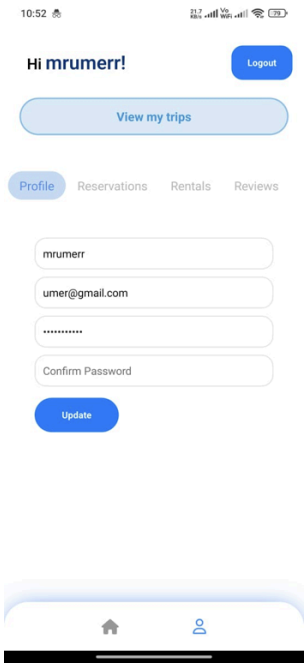
FC96+XMF, Punjab Small Industries Housing Society, Lahore, Pakistan

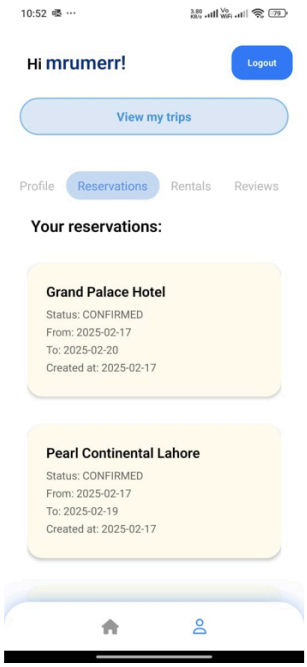
place reviews and your reviews on the profile

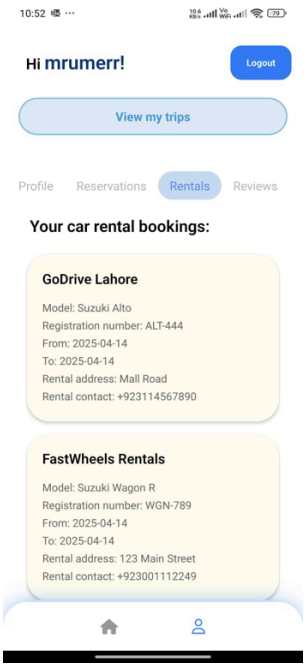




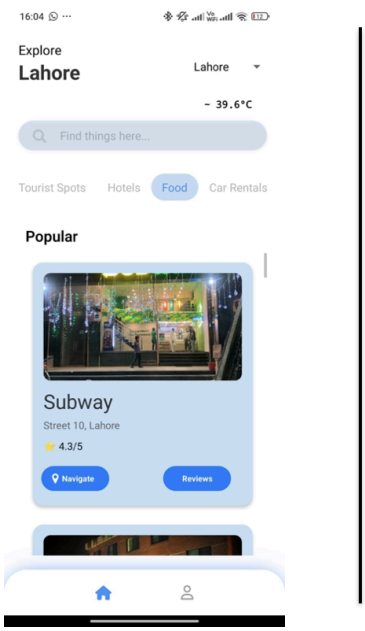
update profile, user reservation history, and user rental booking history

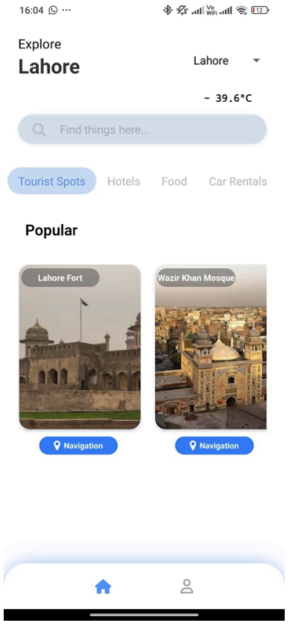


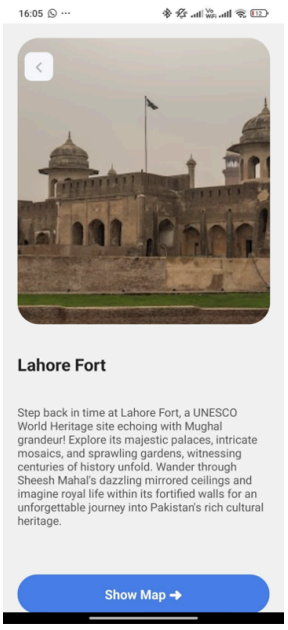




Food and Tourist spots







8. Project Security

Sr#	Security Risks	Potential Losses	Controls
1	A01:2021 - Broken Access Control	**Unauthorized users gaining access to sensitive information** **Loss of customer trust** **Legal consequences due to privacy violations**	**Implement role-based access control (RBAC) to ensure that only authorized users can perform specific actions.** **Ensure access controls are enforced consistently across all applications and API endpoints** **Implement input validation and output encoding to prevent unauthorized data exposure** **Enable audit logging to track access control violations**

2	A02:2021 - Cryptographic failures	**Compromise of sensitive data (e.g., customer details, booking information)** **Financial loss due to data theft attacks** **Non-compliance with data protection regulations such as GDPR**	**Using strong, industry-standard encryption algorithms (e.g., AES-256) for data at rest and in transit** **Implement secure key management practices, ensuring cryptographic keys are stored securely and rotated periodically** **Use secure TLS/SSL protocols for all communications between the client and server** **Avoid exposing sensitive data unnecessarily, especially in logs or error messages**
3	A07:2021 - Identification and Authentication failures	**Identity theft and fraud** **Potential legal repercussions**	**Implement multi-factor authentication (MFA) for users and administrators** **Use secure password storage techniques; bcrypt or Argon2 to hash passwords.** **Implement account lockout mechanisms to mitigate brute force attacks** **Ensure secure session management such as secure cookies, expiring sessions after inactivity, and avoiding session identifiers in URLs**

a. Static and Dynamic Security Scanning Tools

Explore and select **static and dynamic security scanning tools** for your project. The tools should be selected considering the languages and technologies being used in your project.

- ESLint (Static) with security plugins to test for code smells and detect any security issues.
- Postman (dynamic) to test for database injections and broken auth.

9. Risk Management

Potential Risks and Mitigation Strategies

Sr.	Risk Description	Mitigation Strategy
1.	**System Downtime**: The application may experience downtime due to server overload, particularly during peak tourist seasons in Pakistan (e.g., summer or winter vacations).	Implement load balancing and auto-scaling on AWS EC2 to dynamically allocate resources based on traffic demand. Use AWS CloudWatch to monitor server performance and set up alerts for proactive issue resolution.
2.	**Data Breach or Security Flaws**: Sensitive user data (e.g., payment details, personal information) could be compromised due to vulnerabilities in the application or database.	Use end-to-end encryption (SSL/TLS for transmission, AES-256 for storage) and implement multi-factor authentication (MFA) for user logins. Perform post-deployment security audits using tools like OWASP ZAP to identify and address vulnerabilities.
3.	**Gemini API Dependency and Failures**: The trip planning feature relies on Gemini API calls, which may fail due to rate limits, downtime, or incorrect prompt outputs, disrupting the user experience.	Implement fallback mechanisms (e.g., cached trip plans or default suggestions) for API failures. Monitor API performance and response quality using logging tools. Refine prompts through testing to ensure consistent and relevant trip plan outputs.

4.	**Outdated Hotel or Service Data**: Hotel listings, pricing, or reviews may become obsolete, leading to user dissatisfaction or incorrect bookings.	Web-based portal for hotel managers to update listings easily. Implement automated reminders for service providers to refresh data and display a "last updated" timestamp to inform users of data freshness.
5.	**Team Coordination Challenges in Maintenance**: Post-development maintenance (e.g., bug fixes and updates) may face challenges due to team members' reduced availability or lack of coordination.	Document all code, APIs, and deployment processes thoroughly in the project repository (e.g., GitHub README). Assign clear maintenance roles and establish a communication channel for ongoing collaboration.
6.	**Slow Application Performance**: The app may experience slow response times due to inefficient code, unoptimized database queries, or large data transfers (e.g., high-resolution images).	Optimize MongoDB queries with indexing and implement caching (e.g., Redis) for frequently accessed data. Compress images and use lazy loading to reduce load times. Conduct performance testing post-deployment to identify bottlenecks.
7.	**Limited User Adoption**: The app may struggle to attract users due to competition from established platforms (e.g., Booking.com, TripAdvisor) or lack of awareness.	Launch a targeted marketing campaign via social media (e.g., Instagram, X) and local travel blogs, emphasizing Manzil's focus on Pakistani cities. Offer initial discounts or promotions for hotel bookings to attract early adopters.
8.	**Insufficient User Feedback Post-Launch**: Limited feedback from real users may hinder the identification of usability issues or feature gaps, reducing user satisfaction.	Integrate an in-app feedback form and prompt users to rate the app after key actions (e.g., booking a hotel). Analyze user reviews on app stores and conduct post-launch surveys to gather insights for future updates.

9.	**Scope Creep in Future Updates**: Requests for new features (e.g., support for additional cities, advanced integrations) may overwhelm the team during post-launch maintenance, delaying critical updates.	Establish a feature prioritization framework based on user feedback and business impact.
10.	**Cost Overruns on AWS**: Post-free-tier usage of AWS EC2 may lead to unexpected costs if traffic exceeds projections or resources are not optimized.	Monitor AWS usage with cost explorer tools and set budget alerts to prevent overruns. Optimize resource allocation (e.g., use smaller EC2 instances for low-traffic periods) and explore cost-effective alternatives like AWS Lightsail for future scaling.

10. Testing and Evaluation

The testing strategy for Manzil involves the following

- Performing unit and integration testing of React Native components like HomeScreen and Index (initially to test the actual code to see if it works).
- Mocking external dependencies such as expo-router, Axios, and @react-native-async-storage/async-storage to isolate components during testing.
- Simulating user interactions using fireEvent from @testing-library/react-native to verify component behavior.
- Testing navigation by checking if routing functions are called correctly.
- Verifying API calls by ensuring Axios.get is invoked with the expected URLs.
- Utilizing snapshot testing to detect unintended changes in component rendering.
- Handling asynchronous operations using async/await and act.

Sample test cases:

- Verifying that the HomeScreen component renders correctly.
- Checking if clicking the "Hotels" tab triggers the correct API call.
- Ensuring navigation to the "Profile" screen occurs when the profile button is pressed.
- Confirming that the city is updated when a different city is selected.
- Verifying that the "Car Rentals" tab renders with rental companies and that the correct API endpoint is called.
- Check if the "Navigate" button in the "Car Rentals" tab navigates to the GoogleMapScreen with the correct parameters.
- Ensuring navigation to the CarRentalDetailsPage when the "Details" button is clicked.
- Confirm that the <Index /> component renders correctly and contains the expected text and buttons.

Automation tools:

- Jest: The primary testing framework.

- @testing-library/react-native: This is for rendering and interacting with React Native components.
- expo-router: Mocked for testing navigation.
- axios: Mocked for testing API calls.
- @react-native-async-storage/async-storage: Mocked for testing local storage interactions

11. Deployment Guidelines

then mention all the steps for deployment in a production environment.

The deployment of the Manzil system involves the following steps:

- **Access the Code:**

◦ The code for the Manzil is located in two folders on GitHub repositories.

1st Official Development Repository: https://github.com/Shahrexz/P04

2nd Development Repository: https://github.com/Muhammad-Mehdi-Changazi/Manzil

3rd Hotel Management portal Repository:
https://github.com/Muhammad-Mehdi-Changazi/Manzil_Hotel_Admin

4th Car Rental Management portal Repository:
https://github.com/Muhammad-Mehdi-Changazi/Manzil_Car_Rental

- **Build the Application:**

◦ Utilise the chosen development tools (likely React Native) to build a production-ready bundle or application file.

◦ First, you have to use “npm install” on the first Official Repo or the 2nd Repo of Mehdi, and then, once the node modules have been developed, you run “npm start” in the command line

- **Deployment to Hosting Platform:**

◦ Deploy the Staff portal → which included Hotel Management admin and Car Rental admin portals for keeping their system running for users to the online hosting platform of Amazon - AWS

- **Testing on the Deployed Environment:**

◦ Thoroughly tested the Project on **JEST**, and the deployed application portals to ensure all implemented functionalities work correctly in the production-like environment.

- **Online Links:**

◦ The online links where the application Staff portals are hosted are:

Hotel Management: http://myexpoapp-hoteladmin.s3-website-us-east-1.amazonaws.com/

Car Rental Management: http://manzil-carrentals.s3-website-us-east-1.amazonaws.com/

- **Access Information:**

User:

◦ Username: mehdichangazi@gmail.com

◦ Password: Hi@12345

Car Rental Portal:
◦ Username: staff@fastwheels.com

◦ Password: password123

Staff Management Portal:
◦ Username: @pchotel.com

◦ Password: pcadminpass123

12. Conclusion

a. Summary

This project involved the development of Manzil, a mobile application designed to streamline travel planning within Pakistan, specifically for northern tourist destinations and major cities such as Lahore, Karachi, and Islamabad. The goal is to provide users with a seamless, one-stop platform for essential travel information, bookings, and personalized recommendations. Manzil aims to simplify processes like booking accommodations, renting vehicles, exploring attractions, and receiving real-time weather and road updates. The learning was a pretty linear curve, and we learned the routing of React Native and Sockets, including APK production in Android.

A crucial part of the development process was rigorous testing, primarily using Jest as the testing framework and @testing-library/react-native for interacting with components. This involved mocking external dependencies like expo-router, Axios, and @react-native-async-storage/async-storage to ensure components could be tested in isolation. We also simulated user interactions using fireEvent and handled asynchronous operations with the act. Snapshot testing was employed to capture component rendering.

b. Challenges

Developing the Manzil prototype presented several significant challenges, both technical and non-technical. Technically, integrating complex third-party APIs for real-time data like Google Maps, reviews, weather, and potentially others mentioned for broader features required careful handling and mocking for testing. Implementing features like automated hotel bookings and search filters and managing diverse place categories posed distinct technical hurdles. Ensuring the application's scalability and security on Amazon EC2 involved infrastructure configuration challenges. A significant technical effort was dedicated to rigorous testing, particularly unit testing with Jest, which necessitated extensive mocking of dependencies like expo-router, Axios, and AsyncStorage and managing asynchronous operations with the act.

Furthermore, fixing backend vulnerabilities was an explicit requirement in Sprint-4, indicating security was an ongoing concern. Non-technically, the primary challenge was selecting and prioritizing a subset of system requirements from a larger list (spanning potential features in Sprints 1-4) to fit within the prototype's limited timeframe (initially three weeks). Adhering to these timelines and coordinating internal reviews also required effective project management.

c. Future

The Manzil prototype provides a strong foundation upon which the whole application can be built. Future work would focus on taking this to a whole new level, where Mehdi and Usman are thinking of making this a viable business with some modifications and introducing "personalization" into the idea of this whole trip business market. This involves fully developing core features hinted at or partially implemented, such as comprehensive hotel booking functionality with online payment, including secure online payment, a dedicated hotel management interface, a fully functional AI trip planner, and the reservations tab. Enhancements are needed for the search bar with improved filters and city selection, detailed pages for hotels and rentals, better navigation integration, and more robust weather updates. Further development of the machine learning capabilities is needed to provide richer, personalized recommendations. Scaling and optimizing the application on AWS EC2 according to the detailed technical architecture defined after the prototype phase is crucial for a production environment. Ongoing security maintenance and potential expansion to cover more geographic locations within Pakistan represent further avenues for future development.

13. Review checklist

Before submission of this report, the team must perform an internal review. Each team member will review one or more sections of the deliverable.

Chapter/Section Name	Reviewer Name(s)
Chap 1	M. Usman Arshid
Chap 10	M. Usman Arshid
Chap 11	M. Usman Arshid
Chap 12	M. Usman Arshid
Chap 3	Umer Inayat
Chap 6	Umer Inayat
Chap 7	Umer Inayat
Chap 2	Muhammad Mehdi
Chap 4	Muhammad Mehdi
Chap 5	Muhammad Mehdi
Chap 8	Omar Ibne Sajjad
Chap 9	Omar Ibne Sajjad