Project Planning Phase

Project Planning Template (Product Backlog, Sprint Planning, Stories, Story points)

Date	5 July 2024
Team ID	SWTID1720162063
Project Name	Project - Cab Booking App
Maximum Marks	4 Marks

Product Backlog Table

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my name, email, and password.	2	High	[Jashwanth]
Sprint-1		USN-2	As a user, I will receive a confirmation email once I have registered for the application.	1	High	[Mothish]
Sprint-1	Login	USN-3	As a user, I can log into the application by entering my email and password.	1	High	[Rishi Varma]
Sprint-2	Profile Management	USN-4	As a user, I can update my profile information such as email, name, and password.	3	Medium	[Prasad]
Sprint-2	Ride Booking	USN-5	As a user, I can browse available cabs and book a ride by specifying pick-up and drop-off locations.	5	High	[Jashwanth]
Sprint-2	Booking Confirmation	USN-6	As a user, I receive a confirmation with ride details after booking.	2	High	[Mothish]

Sprint-3	Ride History	USN-7	As a user, I can view my past and upcoming ride bookings.	3	Medium	[Rishi Varma]
Sprint-3	Feedback	USN-8	As a user, I can provide feedback and ratings for rides and drivers.	2	Medium	[Prasad]
Sprint-3	Logout	USN-9	As a user, I can log out of the application.	1	Low	[Mothish]
Sprint-4	System Management (Admin)	USN-10	As an admin, I have full control over the system functionalities and configurations.	5	High	[Rishi Varma]
Sprint-4	Rider Management (Admin)	USN-11	As an admin, I can manage rider accounts and ratings.	4	Medium	[Jashwanth]
Sprint-4	Cab Management (Admin)	USN-12	As an admin, I can manage cab listings including adding, updating, and removing cabs.	3	Medium	[Prasad]
Sprint-4	Driver Management (Admin)	USN-13	As an admin, I can manage driver profiles and ratings.	3	Medium	[Rishi Varma]
Sprint-4	Logout (Admin)	USN-14	As an admin, I can log out of the application.	1	Low	[Rishi Varma]

Ideation Phase
Define the Problem Statements

Date	05July 2024
Team ID	SWTID1720162063
Project Name	Cab Booking App
Maximum Marks	3 Marks

Customer Problem Statement Template:

Customer Problem Statement Table

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	I am a busy professional	I'm trying to book a cab easily and quickly through an app.	But the current app requires too many steps to complete a booking.	Because I don't have time to navigate through complicated processes.	Which makes me feel frustrated and stressed, leading me to consider other alternatives.
PS-2	I am a frequent traveler who values safety and reliability.	I'm trying to ensure that the cab I book is safe and the driver is trustworthy.	But I often find it difficult to verify driver credentials and cab conditions through the app.	Because the app does not provide sufficient details or ratings about drivers and cabs.	Which makes me feel anxious and insecure about my ride, affecting my overall experience.
PS-3	I am a tech-savvy user who enjoys seamless digital experiences.	I'm trying to manage my ride bookings, profile, and feedback all in one place.	But the app's interface is cluttered and lacks integration.	Because the app has too many separate screens and inconsistent user experiences.	Which makes me feel overwhelmed and discouraged from using the app regularly.
PS-4	I am a budget-conscious rider who looks for	I'm trying to find and compare ride options to get the best deal.	But the app doesn't provide clear information on fare estimates	Because the pricing and options are hidden or confusing.	Which makes me feel frustrated and uncertain about the cost, leading me to look for other

	cost-effective options.		or ride options.		apps with better transparency.

Project Design Phase-I
Proposed Solution Template

Date	05 July 2024
Team ID	SWTID1720162063
Project Name	Project -Cab Booking App
Maximum Marks	3 Marks

Proposed Solution Template:

Solution for PS-1

S. No.	Parameter	Description
1	Problem Statement (Problem to be solved)	The current app requires too many steps to complete a booking, causing frustration and stress for busy professionals.
2	Idea / Solution description	Simplify the booking process by reducing the number of steps required to complete a booking. Implement a quick booking feature that allows users to book a cab with a single tap.
3	Novelty / Uniqueness	The quick booking feature is unique as it leverages user preferences and previous booking history to streamline the process.
4	Social Impact / Customer Satisfaction	This solution will reduce user frustration, increase satisfaction, and encourage more frequent use of the app.
5	Business Model (Revenue Model)	Increased user engagement and retention will lead to higher ride bookings, thus boosting revenue through ride commissions.

6	Scalability of the Solution	The solution can be easily scaled to accommodate more users by enhancing backend systems and leveraging cloud services for load balancing.

Solution for PS-2

S. No.	Parameter	Description
1	Problem Statement (Problem to be solved)	Difficulty in verifying driver credentials and cab conditions, leading to anxiety and insecurity for frequent travelers.
2	Idea / Solution description	Implement a detailed driver and cab profile feature that includes driver ratings, past ride reviews, and verified credentials. Additionally, add a safety certification badge for cabs.
3	Novelty / Uniqueness	The safety certification badge and detailed profiles provide a unique selling point by enhancing transparency and trust.
4	Social Impact / Customer Satisfaction	This solution will significantly improve user trust and security, resulting in higher customer satisfaction and repeat usage.
5	Business Model (Revenue Model)	Enhanced user trust and satisfaction will lead to increased ride bookings. A premium service can be introduced for users who want additional safety features.
6	Scalability of the Solution	The solution can be scaled by integrating with third-party verification services and expanding the safety features across all operational regions.

Solution for PS-3

S. No.	Parameter	Description

1	Problem Statement (Problem to be solved)	The app's interface is cluttered and lacks integration, overwhelming tech-savvy users.
2	Idea / Solution description	Redesign the user interface to be more intuitive and streamlined, integrating all functionalities (ride booking, profile management, feedback) into a cohesive experience.
3	Novelty / Uniqueness	The new design will use advanced UI/UX principles to create a seamless and engaging user experience.
4	Social Impact / Customer Satisfaction	A more intuitive and integrated interface will enhance user satisfaction and engagement, making the app more user-friendly.
5	Business Model (Revenue Model)	Improved user experience will lead to increased user retention and higher frequency of ride bookings, boosting revenue.
6	Scalability of the Solution	The redesigned interface can be scaled and customized for different user demographics and regions, ensuring broad applicability.

Solution for PS-4

S. No.	Parameter	Description
1	Problem Statement (Problem to be solved)	Lack of clear information on fare estimates and ride options, causing frustration and uncertainty for budget-conscious riders.
2	Idea / Solution description	Implement a transparent pricing model that provides clear fare estimates and detailed ride options upfront. Include a comparison tool for ride options.
3	Novelty / Uniqueness	The transparent pricing model and comparison tool offer a unique value proposition by enhancing clarity and trust.
4	Social Impact / Customer Satisfaction	This solution will reduce user frustration and uncertainty, increasing trust and satisfaction with the app.
5	Business Model (Revenue Model)	Transparent pricing will attract more budget-conscious users, leading to increased ride bookings. Additionally, introducing a subscription model for frequent riders can generate additional revenue.

6	Scalability of the Solution	The solution can be scaled by continuously updating pricing algorithms and expanding ride options to meet the needs of different markets.

Requirement Gathering and Analysis Phase
Solution Architecture

Date	*05july2024*
Team ID	SWTID1720162063
Project Name	*Project - Cab Booking App*
Maximum Marks	

Overview

The RideEase cab booking app aims to provide an efficient, user-friendly platform for booking cab rides, managing driver information, and ensuring secure and reliable transactions. The solution architecture will leverage the MERN stack (MongoDB, Express.js, React.js, Node.js) to build a scalable and robust application.

Components

1. **User Interface**: React.js-based web application providing an intuitive and responsive user experience.
2. **Web Server**: Node.js with Express.js handling server-side logic and API requests.
3. **API Gateway**: Manages client requests and routes them to appropriate services.
4. **Authentication Service**: Ensures secure user authentication and authorization using JWT.
5. **Database**: MongoDB for storing user data, ride details, and other critical information.
6. **Cloud Database**: MongoDB Atlas for scalable and reliable cloud storage.
7. **File Storage**: AWS S3 for storing user profile images and ride history.
8. **External APIs**:
    - Payment Gateway: Stripe and PayPal for processing payments.
    - Notification Services: Twilio and SendGrid for email and SMS notifications.
    - Map Services: Google Maps and Mapbox for location and navigation.
9. **Machine Learning Models** (Optional): TensorFlow and Scikit-Learn for analyzing driver behavior.
10. **Infrastructure**: Kubernetes and Docker for container orchestration and deployment on AWS or IBM Cloud.

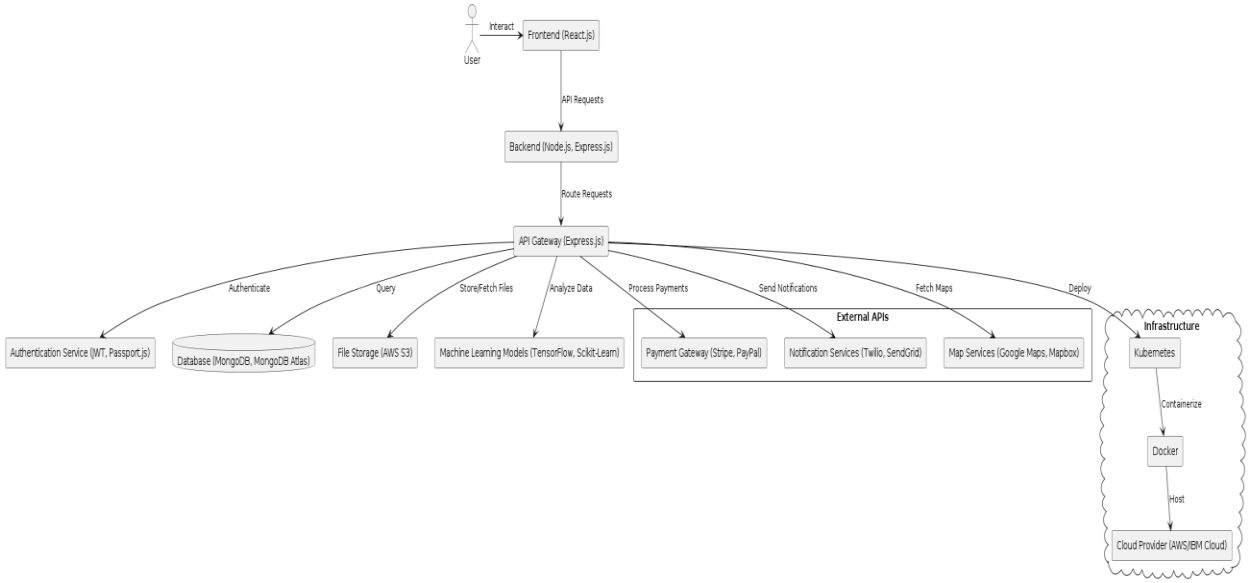


Figure 1: Architecture and data flow of the cab booking application

Requirement Gathering and Analysis Phase
Solution Requirements (Functional & Non-functional)

Date	05 july 2024
Team ID	SWTID1720162063
Project Name	Project - Cab Booking App
Maximum Marks	

Functional Requirements Table

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form
		Registration through Gmail
		Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email
		Confirmation via OTP
FR-3	User Profile Management	Update Email
		Update Name
		Update Password
FR-4	Ride Booking	Browse Available Cabs
		Specify Pick-up and Drop-off Locations
		Choose Ride Options (e.g., Ride-sharing, Premium)
FR-5	Booking Confirmation	Display Cab Information
		Display Driver Details
		Provide Booking ID
FR-6	Ride History	View Past Rides
		View Upcoming Rides
FR-7	Feedback	Provide Ride Ratings
		Provide Driver Ratings
FR-8	User Logout	Log out from the Application
FR-9	Admin System Management	Control System Functionalities
		Configure System Settings
FR-10	Admin Rider Management	Create Rider Accounts
		Update Rider Accounts
		Delete Rider Accounts
FR-11	Admin Cab Management	Add New Cabs
		Update Cab Details
		Remove Inactive Cabs
FR-12	Admin Driver Management	Add New Drivers
		Update Driver Profiles
		Manage Driver Ratings
FR-13	Admin Logout	Log out from the Application

Non-functional Requirements

We can provide descriptions for the non-functional requirements to ensure they meet the overall system standards.

Non-functional Requirements Table

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The application should have an intuitive and user-friendly interface, ensuring ease of use for both riders and admins.
NFR-2	Security	The system must ensure data protection and privacy through encryption, secure authentication methods, and regular security audits.
NFR-3	Reliability	The application should be reliable and consistently perform its intended functions without failure. Regular backups and recovery procedures should be in place.
NFR-4	Performance	The application should provide fast response times and handle multiple transactions efficiently. It should be optimized for both mobile and web platforms.
NFR-5	Availability	The system should have high availability, with minimal downtime, and should be accessible to users 24/7.
NFR-6	Scalability	The system should be scalable to accommodate an increasing number of users and transactions without performance degradation. This includes the ability to add more servers or services as needed.

Requirement Gathering and Analysis Phase
Technology Stack (Architecture & Stack)

Date	05july2024
Team ID	SWTID1720162063
Project Name	Project - Cab Booking Ap
Maximum Marks	

Technical Architecture:

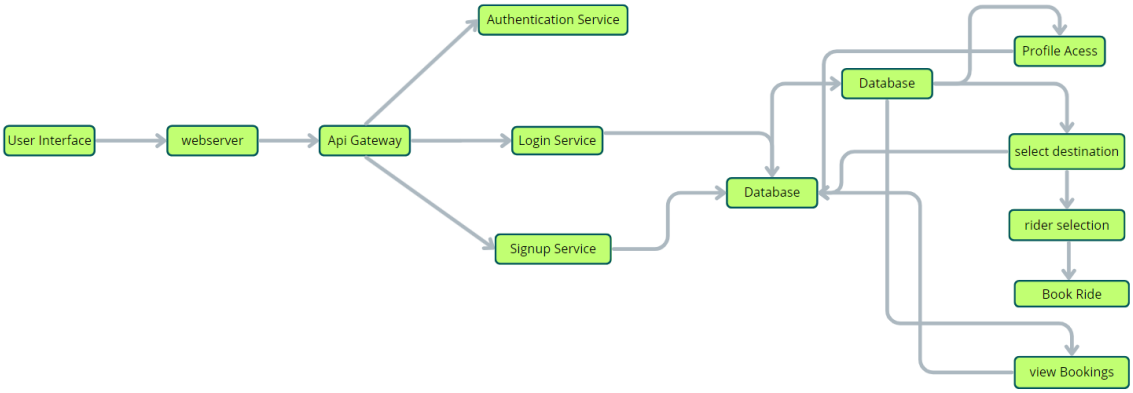


Table-1: Components & Technologies

S.No	Component	Description	Technology
1	User Interface	How users interact with the application	HTML, CSS, JavaScript, React.js
2	Web Server	Hosts the user interface, serving dynamic web pages	Node.js, Express.js
3	API Gateway	Central entry point for client requests	Express.js
4	Authentication Service	Manages user authentication and authorization	JWT, Passport.js

5	Database	Stores persistent data related to rides and users	MongoDB
6	Cloud Database	Cloud-based database service	MongoDB Atlas
7	File Storage	Storing user profile images and ride history	AWS S3
8	External API-1	Payment gateway integration	Stripe API, PayPal API
9	External API-2	Email and SMS notification service	Twilio API, SendGrid API
10	External API-3	Map and location services	Google Maps API, Mapbox API
11	Machine Learning Model	Driver behavior analysis (optional)	TensorFlow, Scikit-Learn
12	Infrastructure (Server)	Local and cloud server deployment configuration	Kubernetes, Docker, AWS EC2, IBM Cloud

Table-2: Application Characteristics

S.No	Characteristics	Description	Technology
1	Open-Source Frameworks	List the open-source frameworks used	React.js, Node.js, Express.js, Mongoose
2	Security Implementations	Security and access controls, encryption methods	JWT, Passport.js, SSL/TLS, OAuth, IAM
3	Scalable Architecture	Justify the scalability of architecture (microservices, containerization)	Docker, Kubernetes, Microservices
4	Availability	Ensure high availability with load balancers and distributed servers	AWS ELB, NGINX, HAProxy
5	Performance	Design considerations for performance (caching, CDNs, number of requests per sec)	Redis Cache, AWS CloudFront, CDN

Frontend Development Report

Date	21-07-2024
Team ID	SWTID1720162063
Project Name	Cab Booking App
Maximum Marks	

Project Title: [Cab Booking Application]

Date: [21-07-2024]

Prepared by: [Cab Booking App Team]

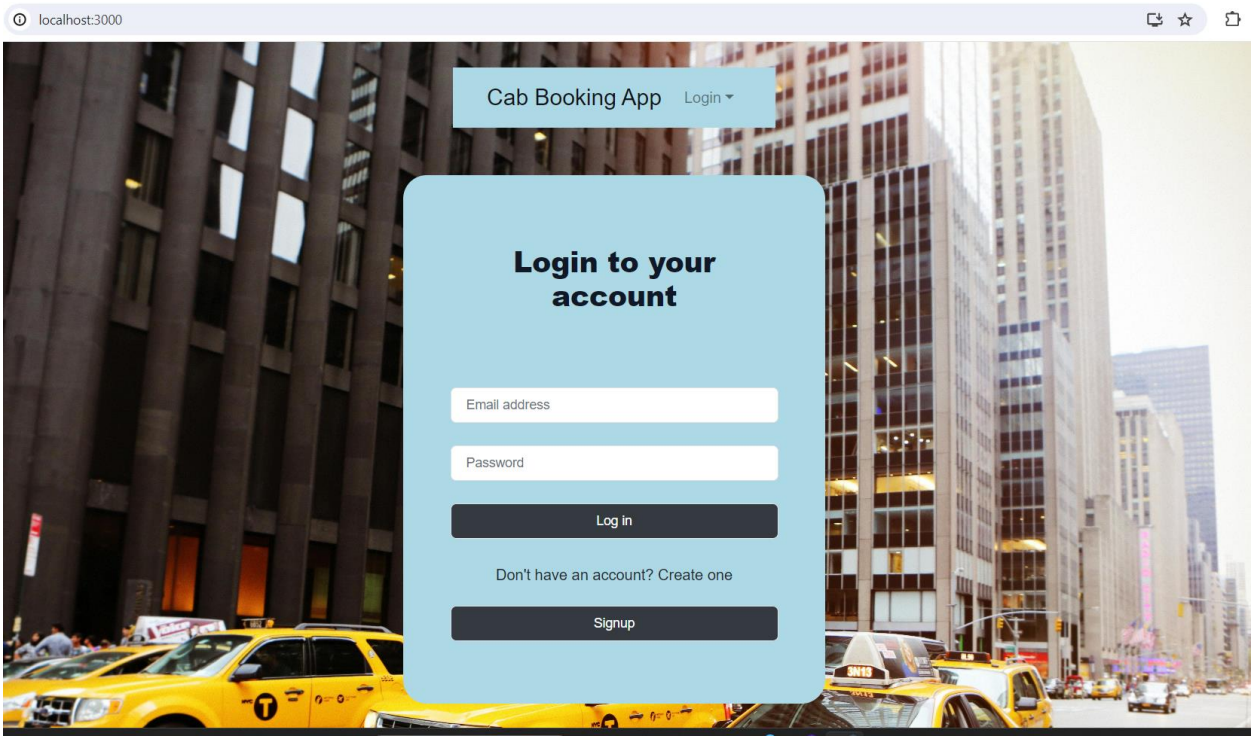
Objective

The objective of this report is to document the frontend development progress and key aspects of the user interface implementation for the Cab Booking App project. This project aims to provide users with a seamless experience for booking rides, managing their bookings, and interacting with the platform. The focus is on creating a responsive, user-friendly interface using modern web development technologies.

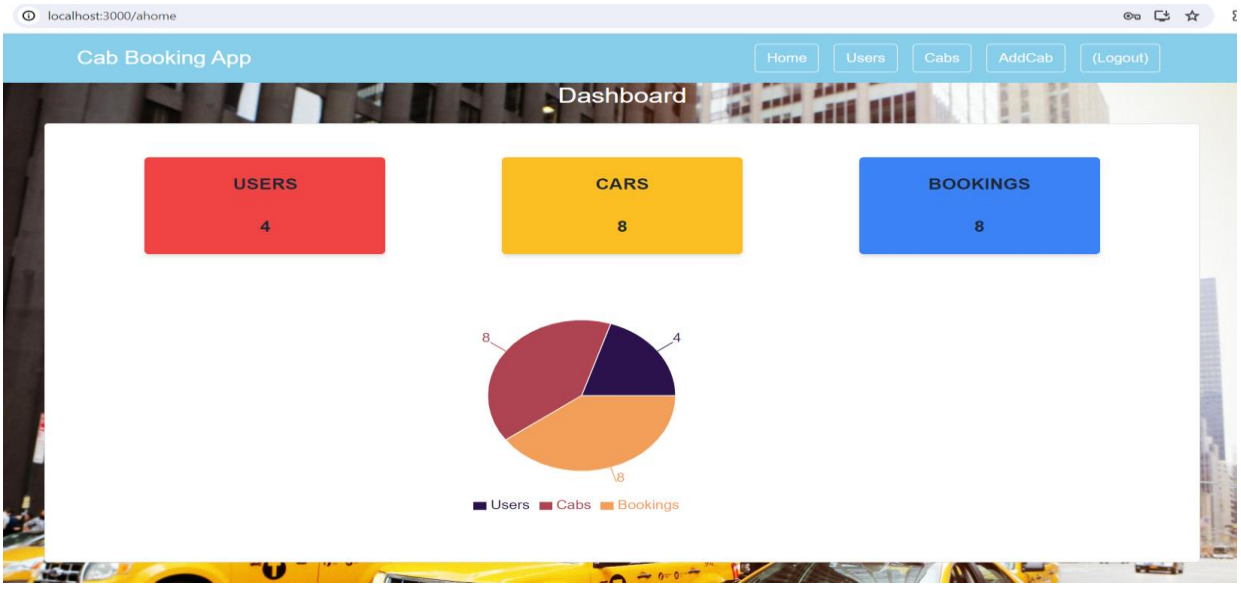
Technologies Used

- **Frontend Framework**: React.js
- **State Management**: Context API
- **UI Framework/Libraries**: Bootstrap
- **API Libraries**: Axios
- **Languages and Tools**: HTML, CSS, JavaScript, Node.js, MongoDB, Express

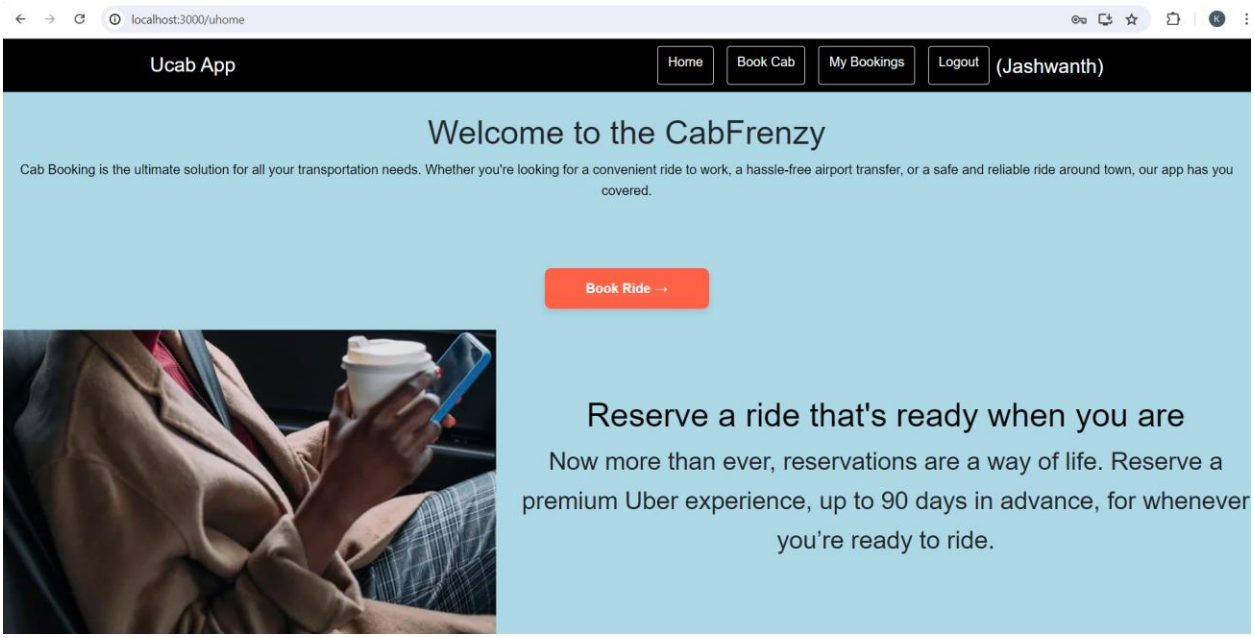
Project Structure



Admin HomePage:



User HomePage:



- **frontend/public**: This directory contains the static files and assets such as the HTML template and images.
- **frontend/src**: The main source directory for the React application.
    - **components**: Contains reusable UI components used across the application.
    - **pages**: Includes different pages for the web app, each representing a different route.
    - **App.js**: Responsible for routing and the main application layout.
    - **index.js**: Entry point for the React application.

Key Components

App.js

App.js is responsible for routing and the main application layout. It includes the setup for React Router and defines the main routes for the application.

/components

The components directory contains reusable UI components used across the application. These components are designed to be modular and can be easily integrated into different parts of the application.

/pages

The pages directory includes different pages for the web app, each corresponding to a specific route. Examples of pages might include the home page, dashboard, and user profile page.

Routing

Routing is managed using React Router. Here are the main routes:

- **/home**: Landing page of the application.
- **/dashboard**: Dashboard displaying user data and statistics.
- **/profile**: User profile management.

State Management

State management is achieved using Context API. This allows for managing global state across the application and provides a way to share data between components without having to pass props manually at every level.

Integration with Backend

The frontend communicates with the backend APIs hosted on http://backend-url. Key endpoints include:

- **GET /api/data**: Retrieves data for display.
- **POST /api/user/login**: Handles user authentication.

User Interface (UI) Design

The UI design follows a user-centric approach, prioritizing usability and accessibility. The design principles include:

- **Consistency**: Ensuring a uniform look and feel across the application.
- **Responsive Design**: Making the application accessible on various devices and screen sizes.
- **Simplicity**: Keeping the interface clean and intuitive to enhance user experience.

Implemented using Bootstrap, the UI leverages pre-designed components and styles to streamline development and maintain visual coherence.

Third-Party Integrations (If any)

- **Axios**: Used for making API requests.
- **React Router**: Used for routing within the React application.

API Development and Integration Report

Date	22-07-2024
Team ID	SWTID1720162063
Project Name	Cab Booking Application
Maximum Marks	

Project Title: [Cab Booking Application]
Date: [22-07-24]
Prepared by: [Jaswanth(Team lead),Mothish(Team Member),Rishi(Team Member),Prasad(Team member).]

Objective

The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Cab Booking Application project. This includes a detailed overview of the technologies used, the project structure, and the design of API endpoints. The report aims to provide a comprehensive understanding of how different components of the backend interact to deliver a seamless and secure cab booking experience. Additionally, it covers the integration points with the frontend application, error handling strategies, validation mechanisms, and security measures to ensure the reliability and integrity of the system.

Technologies Used

- **Backend Framework:** Node.js with Express.js
- **Database:** MongoDB
- **Authentication:** JWT

Project Structure

Provide a screenshot of the backend project structure with explanations for key directories and files.

Key Directories and Files

11. **/controllers**
    - Contains functions to handle requests and responses.
12. **/models**
    - Includes Mongoose schemas and models for MongoDB collections.
13. **/routes**
    - Defines the API endpoints and links them to controller functions.
14. **/middlewares**
    - Custom middleware functions for request processing.
15. **/config**
    - Configuration files for database connections, environment variables, etc.

API Endpoints

A summary of the main API endpoints and their purposes:

User Authentication

- **POST /api/user/register** - Registers a new user.
- **POST /api/user/login** - Authenticates a user and returns a token.

User Management

- **GET /api/user/**

  - Retrieves user information by ID.

- **PUT /api/user/**

  - Updates user information by ID.

Cab Management

- **GET /api/cabs** - Retrieves all cabs.
- **POST /api/cabs** - Adds a new cab.
- **PUT /api/cabs/**

- Updates cab details by ID.

- **DELETE /api/cabs/**

  - Deletes a cab by ID.

Driver Management

- **GET /api/drivers** - Retrieves all drivers.
- **POST /api/drivers** - Adds a new driver.
- **PUT /api/drivers/**

  - Updates driver details by ID.

- **DELETE /api/drivers/**

  - Deletes a driver by ID.

Booking Management

- **GET /api/bookings** - Retrieves all bookings.
- **POST /api/bookings** - Creates a new booking.
- **GET /api/bookings/**

  - Retrieves booking details by ID.

- **PUT /api/bookings/**

  - Updates booking details by ID.

- **DELETE /api/bookings/**

  - Cancels a booking by ID.

Integration with Frontend

The backend communicates with the frontend via RESTful APIs. Key points of integration include:

- **User Authentication:** Tokens are passed between frontend and backend to handle authentication.

- **Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction.

Error Handling and Validation

Describe the error handling strategy and validation mechanisms:

- **Error Handling:** Centralized error handling using middleware.
- **Validation:** Input validation using libraries like Joi or express-validator.

Security Considerations

Outline the security measures implemented:

- **Authentication:** Secure token-based authentication.
- **Data Encryption:** Encrypt sensitive data at rest and in transit.

Database Design and Development Report

Date	22-07-2024
Team ID	SWTID1720162063
Project Name	Cab Booking Application
Maximum Marks	

Project Title: [Cab Booking Application]
Date: [22-07-24]
Prepared by: [Jaswanth(Team lead),Mothish(Team Member),Rishi(Team Member),Prasad(Team member).]

Objective

The objective of this report is to outline the database design and implementation details for the Cab Booking Application project, including schema design and database management system (DBMS) integration.

Technologies Used

- **Database Management System (DBMS):** MongoDB
- **Object-Document Mapper (ODM):** Mongoose

Design the Database Schema

The database schema is designed to accommodate the following entities and relationships:

16. **Users**
    - Attributes: _id, name, email, password, createdAt, updatedAt
17. **Cabs**
    - Attributes: _id, cabNumber, model, capacity, available, createdAt, updatedAt
18. **Drivers**
    - Attributes: _id, name, licenseNumber, cab (references Cab), createdAt, updatedAt
19. **Bookings**
    - Attributes: _id, user (references User), cab (references Cab), driver (references Driver), pickupLocation, dropLocation, pickupTime, status, createdAt, updatedAt

Implement the Database using MongoDB

The MongoDB database is implemented with the following collections and structures:

Database Name: cab_booking_app

20. **Collection: users**

```
{
  "_id": ObjectId,
  "name": String,
  "email": String,
  "password": String,
  "createdAt": Date,
  "updatedAt": Date
}
```

2. Collection: cabs

{

"_id": ObjectId,

"cabNumber": String,

"model": String,

"capacity": Number,

"available": Boolean,

"createdAt": Date,

"updatedAt": Date

}

3. Collection: drivers

{

"_id": ObjectId,

"name": String,

"licenseNumber": String,

"cab": ObjectId (references cabs),

"createdAt": Date,

"updatedAt": Date

}

21. Collection: bookings

{

"_id": ObjectId,

"user": ObjectId (references users),

"cab": ObjectId (references cabs),

"driver": ObjectId (references drivers),

"pickupLocation": String,

"dropLocation": String,

"pickupTime": Date,

"status": String,

"createdAt": Date,

"updatedAt": Date

}

Integration with Backend

- **Database connection:** Provide a screenshot of the database connection done using Mongoose.
- The backend APIs interact with MongoDB using Mongoose ODM. Key interactions include:
    - **User Management:** CRUD operations for users.
    - **Cab Management:** CRUD operations for cabs, with availability status updates.
    - **Driver Management:** CRUD operations for drivers and their assigned cabs.
    - **Booking Management:** CRUD operations for bookings associated with users, cabs, and drivers.

User Acceptance Testing (UAT) Template

Date	22 – 07 - 2024
Team ID	SWTID1720162063
Project Name	Cab Booking App
Maximum Marks	

Project Overview:

Project Name: Cab Booking App

Project Description: The Cab Booking App is a comprehensive platform designed to streamline the process of booking rides and managing transportation logistics. Users can easily register and log in, book rides. The application also provides an admin interface where administrators can log in to manage cars and drivers.

Project Version: 1.0

Testing Period: 19 – 07 - 2024 to 22 – 07 - 2024

Testing Scope:

22. Features to be Tested:

- User Registration and Login

- Ride Booking
- Admin Login
- Car Management (Admin)
- Driver Management (Admin)

23. User Stories/Requirements to be Tested:

- As a user, I can register and log in.
- As a user, I can book a ride.
- As an admin, I can log in.
- As an admin, I can add, edit, and delete cars.
- As an admin, I can add, edit, and delete drivers.

Testing Environment:

URL/Location: [Web URL or Application Location]

Credentials (if required): [Username/Password]

Test Cases:

Test Case ID	**Test Scenario**	**Test Steps**	**Expected Result**	**Actual Result**	**Pass/Fail**
TC-001	User Registration	Step 1: Open registration page. Step 2: Enter details. Step 3: Submit.	User registered successfully	User Registered	Pass
TC-002	User Login	Step 1: Open Login page. Step 2: Enter credentials and submit.	User LoggedIn successfully	User LoggedIn	Pass

TC-003	Book a Ride	Step 1: Login as user. Step 2: Enter ride details. Step 3: Confirm booking.	Ride booked successfully.	Ride booked successfully.	Pass
TC-004	Admin Login	Step 1: Open admin login page. Step 2: Enter credentials.	Admin logged in successfully.	Admin logged in successfully.	Pass
TC-005	Add a Car (Admin)	Step 1: Login as admin. Step 2: Open car management. Step 3: Enter car details.	Car added successfully.	Car added successfully.	Pass
TC-006	Edit a Car (Admin)	Step 1: Login as admin. Step 2: Open car management. Step 3: Select car to edit. Step 4: Edit details. Step 5: Submit.	Car edited successfully.	Car cannot be edited	Fail
TC-007	Delete a Car (Admin)	Step 1: Login as admin. Step 2: Open car management. Step 3: Select car to delete.	Car deleted successfully.	Car deleted successfully.	Pass

		Step 4: Confirm deletion.			
TC-008	Add a Driver (Admin)	Step 1: Login as admin. Driver added successfully. Step 2: Open driver management. Step 3: Enter driver details. Step 4: Submit.	Driver added successfully.	Driver added successfully.	Pass
TC-009	Edit a Driver (Admin)	Step 1: Login as admin. Step 2: Open driver management. Step 3: Select driver to edit. Step 4: Edit details. Step 5: Submit.	Driver edited successfully.	Driver edited successfully.	Pass
TC-010	Delete a Driver (Admin)	Step 1: Login as admin. Step 2: Open driver management. Step 3: Select driver to delete. Step 4: Confirm deletion.	Driver deleted successfully.	Driver cannot be deleted.	Fail

Bug Tracking:

Bug ID	Bug Description	Steps to reproduce	Severity	Status	Additional feedback
BG-001	Car editing not saving changes	1. Login as admin 2. Open car management 3. Select a car to edit 4. Edit car details 5. Click "Submit"	High	Resolved And closed	Changes not saving due to a server-side validation issue. Fixed and now changes are saved.
BG-002	Error message when deleting driver	1. Login as admin 2. Open driver management 3. Select a driver to delete 4. Click "Delete"	Medium	Resolved	Error due to foreign key constraint. Issue fixed and drivers can now be deleted.

Sign-off:

Tester Name: M MOTHISH

Date: 22 – 07 - 2024

Signature: mothish

Notes:

- Ensure that all test cases cover both positive and negative scenarios.
- Encourage testers to provide detailed feedback, including any suggestions for improvement.
- Bug tracking should include details such as severity, status, and steps to reproduce.
- Obtain sign-off from both the project manager and product owner before proceeding with deployment.

Project Documentation format

1. Introduction

Project Title: Cab Booking Application

Team Members:

- Jaswanth: Project Manager
- Mothish: Frontend Developer
- Rishi: Backend Developer
- Prasad: Database Administrator

2. Project Overview

Purpose:
The Cab Booking Application aims to provide a seamless and user-friendly platform for booking cabs. It allows users to book rides, view ride history, and rate drivers, while admins can manage cab and driver details.

Features:

- User and Admin login with role-based access.
- Cab booking and ride history for users.
- Admin dashboard for managing cabs and drivers.
- Real-time ride status updates.
- Rating and feedback system for users.

3. Architecture

Frontend:
The frontend is built using React, providing a responsive and interactive user interface. Key components include the login page, user dashboard, booking interface, and admin management pages.

Backend:
The backend is powered by Node.js and Express.js, handling API requests, authentication, and business logic. It serves as the intermediary between the frontend and the database.

Database:
MongoDB is used for storing user data, cab details, ride history, and driver information. The database schema is designed to efficiently manage relationships and queries.

4. Setup Instructions

Prerequisites:

- Node.js
- MongoDB

Installation:

*Clone the repository:

git clone https://github.com/your-repo/cab-booking-app.git

*Navigate to the project directory:

cd cab-booking-app

*Install dependencies for both frontend and backend:

cd client

npm install

cd ../server

npm install

*Set up environment variables in .env files for both client and server.

5. Folder Structure

Client:

- **/src**
  - **components**: Contains React components.
  - **pages**: Contains different page components like Login, Dashboard, Booking, Admin.
  - **services**: Contains service files for API calls.
  - **styles**: Contains CSS files.

Server:

- **/src**
    - **controllers**: Handles request logic.
    - **models**: Contains Mongoose schemas.
    - **routes**: Defines API routes.
    - **middlewares**: Contains middleware functions for authentication, error handling, etc.
    - **utils**: Utility functions and constants.

6. Running the Application

Commands to start the servers:

- **Frontend:**

  ```
  cd client
  npm start
  ```

- **Backend:**

  ```
  cd server
  npm start
  ```

7. API Documentation

Endpoints:

- **User Authentication:**
    - **POST /api/auth/login:** Authenticate user/admin.
    - **POST /api/auth/register:** Register a new user.
- **Cab Management (Admin):**
    - **GET /api/cabs:** Retrieve all cabs.
    - **POST /api/cabs:** Add a new cab.
    - **PUT /api/cabs/:id:** Update cab details.
    - **DELETE /api/cabs/:id:** Delete a cab.
- **Driver Management (Admin):**
    - **GET /api/drivers:** Retrieve all drivers.
    - **POST /api/drivers:** Add a new driver.

        - **PUT /api/drivers/:id:** Update driver details.
        - **DELETE /api/drivers/:id:** Delete a driver.
- **Booking (User):**
    - **POST /api/bookings:** Create a new booking.
    - **GET /api/bookings/:userId:** Get bookings for a user.
    - **PUT /api/bookings/:id:** Update booking status.

8. Authentication

Authentication and Authorization:

- JWT tokens are used for authentication.
- Middleware checks token validity and user roles.
- Sessions are managed using cookies and local storage.

9. Known Issues

- Occasionally, booking status updates may lag.
- Admin dashboard filtering options need improvement.

10. Future Enhancements

- Implement real-time cab tracking using GPS.
- Enhance user interface with more interactive elements.
- Add support for multiple payment options.
- Improve the rating and feedback system with detailed analytics.