

FSD Project Documentation :

QuickCab – Online Cab Booking Platform

Table of Contents

1. Introduction	1
• Project Title	
• Team Members	
2. Project Overview	2
• Purpose	
• Features	
3. Architecture	3
• Frontend	
• Backend	
• Database	
4. Setup Instructions	5
• Prerequisites	
• Installation	
5. Folder Structure	6
• Client	
• Server	
6. Running the Application	7
• Frontend	
• Backend	
7. API Documentation	8
• Endpoints	
• Request Methods and Parameters	
• Example Responses	
8. Authentication	10
• JWT and Session Handling	
• Role-Based Access Control	
9. User Interface	11
• Screenshots or GIFs of Key Features	
10. Testing	12
• Strategy	
• Tools Used	

11. Screenshots or Demo	13
• Visual Walkthrough	
• Demo Link	
12. Known Issues	14
• Bugs and Limitations	
13. Future Enhancements	15
• Planned Features	
• Optimization Ideas	

1. INTRODUCTION

Project Title: QuickCab – Online Cab Booking Platform

Team Members and Roles:

NAME	Reg. No.	Roles
Akhand Singh	22BCE10587	Backend/Server
Om Aditya Pandey	22BCE10884	Backend / Server
Ansh Agarwal	22BCE11668	Frontend/Client
Siddharth Jha	22BCE10671	Frontend/Client

2. Project Overview

Purpose

QuickCab is a comprehensive MERN-stack-based cab booking web application designed to streamline the process of booking, managing, and tracking cabs online. It caters to both users seeking to book rides and administrators overseeing ride logistics. The primary objectives of QuickCab are:

- **User Convenience:** Provide a seamless and intuitive platform for users to book cabs with ease.
- **Administrative Efficiency:** Equip administrators with tools to manage drivers, monitor bookings, and oversee operations effectively.
- **Real-Time Tracking:** Enable users to track their rides in real-time, enhancing transparency and trust.
- **Secure Transactions:** Ensure secure user authentication and data handling throughout the application.

Key Features

- **User Registration and Authentication:** Secure sign-up and login functionalities using JWT (JSON Web Tokens) for session management.
- **Role-Based Dashboards:** Distinct dashboards for users and administrators, tailored to their specific needs and functionalities.
- **Cab Booking Interface:** Users can input pickup and drop-off locations to book cabs seamlessly.
- **Real-Time Ride Status:** Live tracking of ride status, providing users with updates on their ride's progress.
- **Ride History and Transactions:** Comprehensive logs of past rides and transaction records for user reference.
- **Admin Panel:** Tools for administrators to manage drivers, monitor user activities, and oversee bookings.
- **Google Maps Integration:** Visual representation of routes and locations using Google Maps API.

3. Architecture

Frontend

- **Technologies Used:** React.js, Tailwind CSS, JavaScript
- **Design Principles:**
 - **Single Page Application (SPA):** Ensures a fluid user experience without full page reloads.
 - **Routing:** Implemented using React Router for efficient navigation between components.
 - **State Management:** Utilizes Context API or Redux for managing global state across the application.
 - **Form Validation:** Ensures data integrity and user input validation during registration and booking processes.
 - **Responsive Design:** Tailwind CSS facilitates a mobile-first, responsive design approach.

Backend

- **Technologies Used:** Node.js, Express.js
- **Core Functionalities:**
 - **RESTful APIs:** Structured endpoints for handling user actions, bookings, and administrative tasks.
 - **Authentication Middleware:** JWT-based middleware to protect routes and ensure secure access.
 - **CRUD Operations:** Comprehensive Create, Read, Update, Delete functionalities for users, rides, and bookings.
 - **Error Handling:** Standardized error responses to maintain consistency and aid in debugging.

Database

- **Technology Used:** MongoDB
- **Schema Designs:**

■ **User Model:**

- {
 name: String,
 email: String,
 password: String (hashed),
 role: { type: String, enum: ['user', 'admin'] }
}

■ **Driver Model:**

- {
 name: String,
 email: String,
 phone: String,
 status: { type: String, enum: ['available', 'unavailable'] },
 location: {
 type: { type: String, default: 'Point' },
 coordinates: [Number]
 }
}

■ **Booking Model:**

- {
 userId: ObjectId,
 driverId: ObjectId,
 pickupLocation: String,
 dropLocation: String,
 status: { type: String, enum: ['pending', 'accepted', 'completed'] },
 fare: Number,
 date: Date
}

■ **Ride History Model:**

- {
 bookingId: ObjectId,

```
    userId: ObjectId,  
    driverId: ObjectId,  
    pickupLocation: String,  
    dropLocation: String,  
    fare: Number,  
    date: Date  
  }
```

Integration Points

- **Google Maps API:** Facilitates route visualization, distance calculations, and location-based services.
- **JWT (JSON Web Token):** Manages user sessions and implements role-based access control.
- **Nodemailer:** Sends email notifications for booking confirmations, ride updates, and other alerts.

4.Setup Instructions

a. Prerequisites

To deploy and execute the QuickCab application, it is imperative to install the following dependencies and runtime environments:

- i. **Node.js (v16 or higher)**: Provides the JavaScript runtime for both frontend and backend execution.
- ii. **MongoDB (v4.4 or higher)**: A document-oriented NoSQL database system used for persistent data storage.
- iii. **Git**: Distributed version control system for cloning and managing source code.
- iv. **npm**: Node Package Manager for dependency management.

b. Installation

```
# Clone the repository
$ git clone https://github.com/yourusername/quickcab.git

# Navigate to project directory
$ cd quickcab

# Install frontend dependencies
$ cd client
$ npm install

# Install backend dependencies
$ cd ../server
```



```
$ npm install
```

Execute the commands below to set up the environment and install necessary modules:

Listing 1: Installation Steps

C. Environment Configuration

```
PORT =5000  
MONGO_URI=your_mongodb_connection_string JWT_SECRET =your_jwt_secret_key
```

To securely manage sensitive credentials, create a .env file within the server/ directory:

Listing 2: .env Configuration for Backend

Optionally, you may also configure frontend-specific API endpoints in the client folder if using a proxy for local development.

5.Folder Structure

a.Client (React.js)

The client directory contains all frontend logic using ReactJS. The structure adheres to component-based design principles:

Listing 3: React Frontend Directory Layout

<u>client/</u>		
<u>public/</u>		<i># Static assets like <u>index.html</u></i>
<u>src/</u>		
<u>components/</u>		<i># Reusable UI <u>components</u> (</i>
<u>Navbar</u> , <u>Cab Card</u>)		
<u>pages/</u>		<i># React <u>pages</u> like <u>HomePage</u> ,</i>
<u>Booking Page</u> , <u>Profile Page</u>		
<u>contexts/</u>		<i># React <u>Contexts</u> (e.g.,</i>
<u>Auth Context</u> , <u>Ride Context</u>)		
<u>services/</u>		<i># <u>Axios</u> API functions to</i>
<u>interact with backend</u>		
<u>App.js</u>		<i># <u>Main component</u> managing</i>
<u>routes</u>		
<u>index.js</u>		<i># React DOM entry point</i>

b.Server (Node.js+ Express)

The server directory implements RESTful API routes and manages businesslogic and data persistence.

Listing 4: Backend Directory Structure

<u>server/</u>	
<u>controllers/</u>	<i># Contains logic for user <u>auth</u>, ride booking, admin controls</i>
models/	<i># Mongoose schemas for User,</i>
Ride, Driver, Booking	
routes/	<i># API endpoints grouped into</i>
modular files	
middleware/	<i># JWT auth, error handler, role-based</i>
guards	
config/	<i># MongoDB connection config</i>
utils/	<i># Custom utilities (e.g., fare</i>
calculation)	
server.js	<i># Main entry point of the</i>
Express app	
.env	<i># Environment configuration</i>

6. Running the Application

a. Frontend Execution

To launch the React development server:

Listing 5: Launch Frontend Server

```
$ cd client  
$ npm start
```

b.Backend Execution

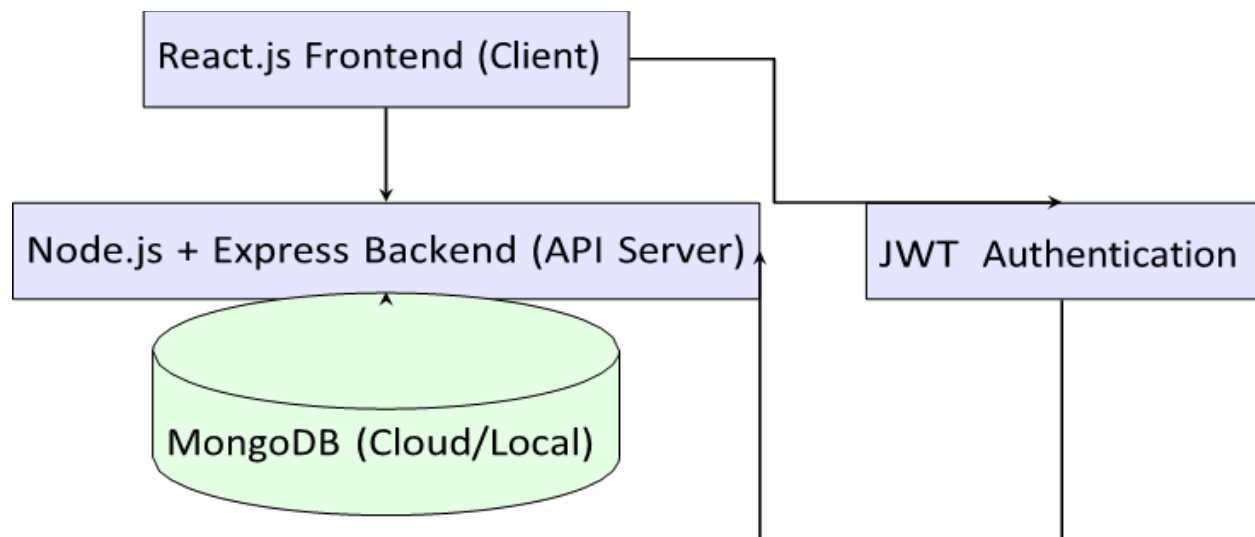
To start the backend Expressserver:

Listing 6: Launch Backend Server

```
$ cd server  
$ npm start
```

Ensure MongoDB is running and the backend is correctly connected to the database URI.

System Architecture Diagram



7 API DOCUMENTATION

Resource	Endpoint	Method	Description	Request Body	Example Response
Auth	/api/auth	GET	Get logged in user	None	{ "id": "123", "name": "John Doe", "email": "john@example.com", "phone": "9876543210" }
Auth	/api/auth/login	POST	Authenticate user & get token	{ "email": "john@example.com", "password": "password123" }	{ "token": "..." }
User	/api/users	POST	Register a new user	{ "name": "John Doe", "email": "john@example.com", "password": "password123", "phone": "9876543210" }	{ "token": "..." }
User	/api/users/me	GET	Get current user profile	None	{ "id": "123", "name": "John Doe", "email": "john@example.com", "phone": "9876543210" }
User	/api/users/me	PUT	Update user profile	{ "name": "John Smith", "phone": "9876543211" }	{ "id": "123", "name": "John Smith", "email": "john@example.com", "phone": "9876543211" }
Cab	/api/cabs	GET	Get all cab types	None	[{ "id": "1", "name": "Economy", ... }]
Cab	/api/cabs/:id	GET	Get cab by ID	None	{ "id": "1", "name": "Economy", ... }
Cab	/api/cabs	POST	Create a cab type	{ "name": "Premium", ... }	{ "id": "2", "name": "Premium", ... }
Booking	/api/bookings	POST	Create a	{ "pickup": { ... },	{ "id": "123", "status":

			new booking	"destination": { ... }, "cabType": "1", "fare": 2500, ... }	"pending", ... }
Booking	/api/bookings/:id	GET	Get booking by ID	None	{ "id": "123", "status": "confirmed", ... }
Booking	/api/bookings/my-bookings	GET	Get all bookings for current user	None	[[{ "id": "123", "status": "completed", ... }]]
Booking	/api/bookings/:id/status	PUT	Update booking status	{ "status": "completed" }	{ "id": "123", "status": "completed", ... }
Booking	/api/bookings/:id/rating	PUT	Add rating and feedback to booking	{ "rating": 5, "feedback": "Great service!" }	{ "id": "123", "rating": 5, "feedback": "Great service!", ... }
Driver	/api/drivers	POST	Register a new driver	{ "name": "Rahul Kumar", ... }	{ "token": "..." }
Driver	/api/drivers/login	POST	Authenticate driver & get token	{ "email": "rahul@example.com", "password": "password123" }	{ "token": "..." }
Driver	/api/drivers/me	GET	Get current driver profile	None	{ "id": "123", "name": "Rahul Kumar", ... }
Driver	/api/drivers/me	PUT	Update driver profile	{ "isAvailable": false }	{ "id": "123", "isAvailable": false, ... }
Driver	/api/drivers/nearby	GET	Get nearby available drivers	{ "lat": 28.6139, "lng": 77.209, "radius": 5 } (query params)	[[{ "id": "123", "name": "Rahul Kumar", ... }]]
Admin	/api/admin/users	GET	Get all users	None	[[{ "id": "123", "name": "John Doe", ... }]]
Admin	/api/admin/users/:id	PUT	Update user status	{ "status": "blocked" }	{ "id": "123", "status": "blocked", ... }
Admin	/api/admin/users/:id	DELETE	Delete a user	None	{ "msg": "User removed" }
Admin	/api/admin/drivers	GET	Get all drivers	None	[[{ "id": "123", "name": "Rahul

					Kumar", ... }}
Admin	/api/admin/drivers/:id	PUT	Update driver status	{ "status": "blocked" }	{ "id": "123", "status": "blocked", ... }
Admin	/api/admin/drivers/:id	DELETE	Delete a driver	None	{ "msg": "Driver removed" }
Admin	/api/admin/bookings	GET	Get all bookings	None	[{ "id": "123", "status": "completed", ... }]
Admin	/api/admin/stats	GET	Get system statistics	None	{ "totalUsers": 250, "totalDrivers": 100, "totalBookings": 1500, ... }
Error	Any	Any	Possible errors	Depends	{ "msg": "Error message" }

8 AUTHENTICATION

Authentication in QuickCab is implemented using secure, stateless mechanisms to protect access across all user roles: Passengers, Drivers, and Admins.

Overview:

- Type: Token-Based Authentication
- Method: JSON Web Tokens (JWT)
- Password Security: bcrypt.js for hashing
- Route Protection: Middleware-based authorization
- Role Support: Multi-role (User, Driver, Admin)

Authentication Flow

1. Registration
 - A user (or driver) registers via /api/users or /api/drivers.
 - Password is hashed using bcrypt before storing in MongoDB.
2. Login
 - A user logs in via /api/auth/login or /api/drivers/login.
 - If credentials are valid, a signed JWT is returned.
 - The token contains user ID, role, and expiration time.
3. Token Storage
 - On the frontend, the token is typically stored in localStorage or HTTP-only cookies (depending on client implementation).
4. Authenticated Requests
 - The client sends the JWT in the Authorization header with each protected request: Authorization: Bearer <token>
5. Middleware Validation
 - The backend checks:
 - Token presence and validity
 - Expiration
 - Role-based access (e.g., admin-only routes)
 - If valid, the request proceeds; otherwise, a 401 or 403 is returned.

9 USER INTERFACE

The QuickCab UI is designed using React.js and Tailwind CSS with a mobile-first responsive layout. It emphasizes simplicity, usability, and clarity to enhance the experience across all user types.

A. Passenger (User) Interface

1. Login / Register Page
 - Fields: Email, Password, Name, Phone
 - Buttons: Login, Register, Forgot Password
2. Dashboard (Home)
 - Welcome message & user avatar
 - Cab category selection (Economy, Premium, SUV)
 - Map component with current location pin (Leaflet.js)
 - Pickup & destination selectors with autocomplete
 - "Estimate Fare" and "Book Ride" buttons
3. Booking Confirmation
 - Live ride status: searching, accepted, en route, completed
 - Driver details: name, rating, contact option
 - Real-time map tracking with route overlay
4. Booking History
 - List of past rides with date, fare, and status
 - Clickable details view for feedback/rating
5. Profile Page
 - Update user details (name, phone, etc.)
 - Logout button

B. Driver Interface

1. Login / Register Page
 - Driver-specific fields: name, license number, vehicle details
2. Dashboard
 - Toggle availability (Online / Offline)
 - New ride request modal with accept/decline options
 - Ride in-progress display with route navigation aid
3. Ride Management
 - Start/Complete ride button
 - Fare summary & payment status (pending/future use)
4. Earnings Overview

- Weekly/monthly earnings summary
- List of completed rides
- 5. Profile Page
 - Driver details, vehicle info
 - Update availability, status

C. Admin Interface

1. Admin Login
 - Email & password authentication
2. Dashboard Overview
 - Summary cards: Total Users, Drivers, Bookings
 - Charts: Daily bookings, Top drivers, Revenue trends (Chart.js/Recharts)
3. User Management
 - View/edit/delete users
 - Block/unblock user accounts
4. Driver Management
 - View driver status
 - Block/unblock/remove drivers
5. Booking Oversight
 - View all rides with status filter (pending, completed, canceled)
 - Search and sort by user, driver, date
6. System Settings (Future Scope)
 - Update fare rates, cab types, promotions

10. TESTING

The testing strategy for **QuickCab** ensures the reliability, performance, and security of the application across all user roles — Passengers, Drivers, and Administrators. The testing lifecycle includes multiple levels of tests to catch issues early and deliver a robust application experience.

• Unit Tests

Testing individual components, functions, and utilities in isolation to validate logic and correctness.

- **Scope:**
 - React components (e.g., booking forms, maps)
 - Backend utility functions (e.g., fare calculator, Haversine formula)
 - API service layers (e.g., Axios calls)
- **Tools:**
 - `Jest`: For testing JavaScript logic and React components.
 - `Mocha`: For backend logic and Express routes.

• Integration Tests

Testing the interaction between components, APIs, and the database to ensure proper integration.

- **Scope:**
 - User registration → JWT token generation
 - Booking a ride → cab availability → storing in `Bookings` collection
 - Driver status toggle → ride assignment flow
- **Tools:**
 - `Jest`: Extended tests for chained logic and component communication.
 - `Mocha + Chai + Supertest`: For testing Express API routes and MongoDB integration.

• End-to-End (E2E) Tests

Validating complete user workflows from frontend to backend to simulate real-world scenarios.

- **Scope:**
 - Passenger logs in → books a cab → tracks ride → rates driver

- Driver accepts ride → completes ride → views earnings
 - Admin views analytics and manages users
- **Tools:**
 - Cypress: Automated browser tests simulating user flows
 - Selenium: For browser compatibility and behavioral testing

• Performance Tests

Evaluating how the application behaves under different traffic conditions and identifying bottlenecks.

- **Scope:**
 - Concurrent cab booking requests
 - Multiple drivers updating statuses
 - Admin dashboard under data-heavy usage
- **Tools:**
 - Apache JMeter: Load testing backend APIs and database performance
 - LoadView: For end-user experience simulation under load

• Security Tests

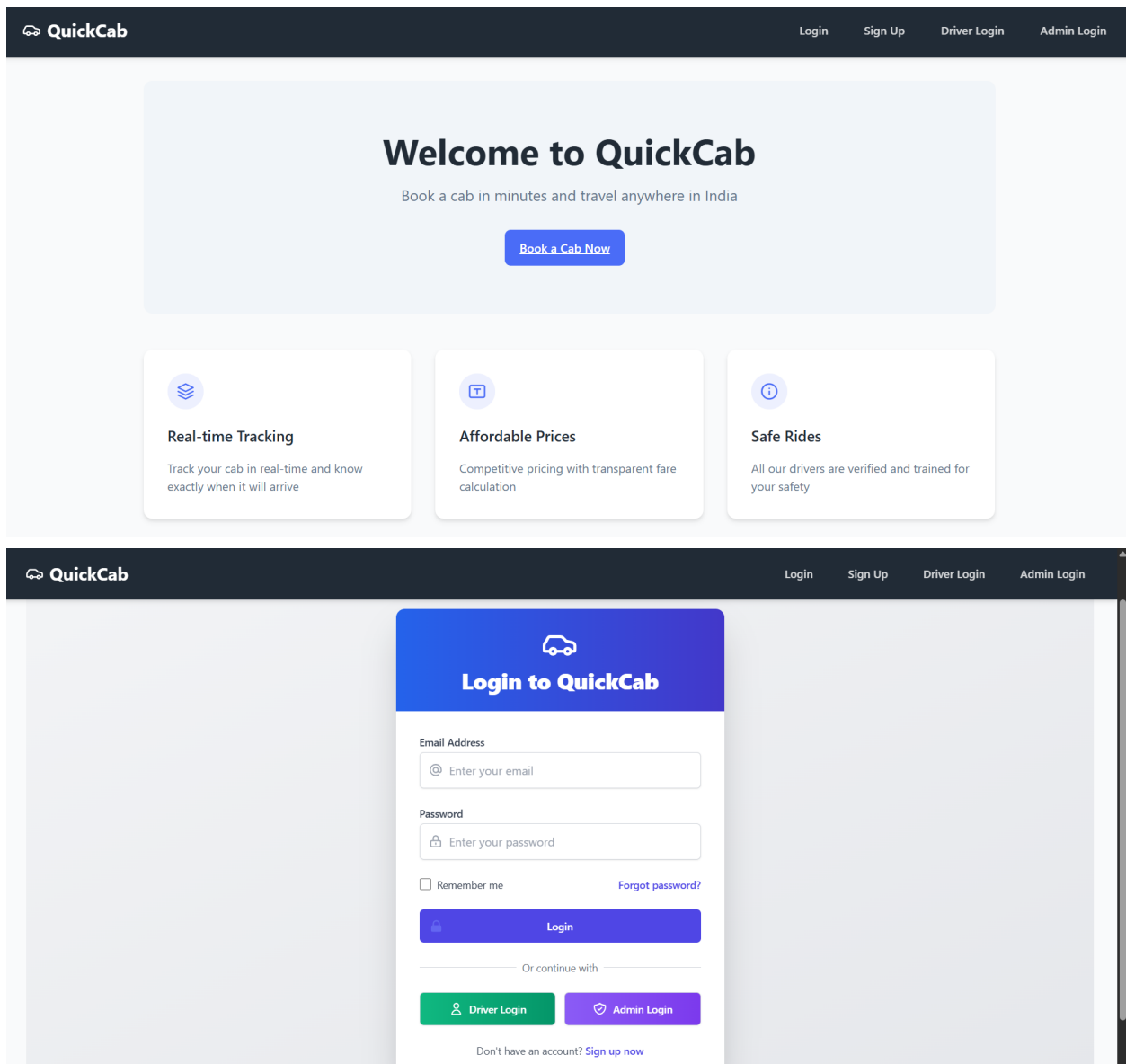
Ensuring the application is safe from vulnerabilities such as XSS, SQL Injection, and session hijacking.

- **Scope:**
 - Input validation and sanitization
 - JWT token protection and route access control
 - Role-based access enforcement (user/driver/admin)
- **Tools:**
 - OWASP ZAP: For automated security scanning and vulnerability detection
 - Nessus: For identifying deeper network and configuration vulnerabilities


CI/CD Integration


- Tests can be automated using **GitHub Actions** or **GitLab CI** to ensure tests are run on every push and pull request.
- Code coverage metrics monitored using tools like **Coveralls** or **Codecov**.


11. Screenshots and Demo





Admin Dashboard

Total Users
25
[View All](#)

Total Drivers
10
[View All](#)

Total Rides
0
[View All](#)


Active Rides
0
[View All](#)


Total Revenue
₹0


Recent Bookings


No bookings found.
[View All Bookings](#)

Quick Actions

Add New User

Add New Driver

Create New Booking

Export Reports

Driver Dashboard

Your Status

Name: rahul

Vehicle: swift (up32fa2130)

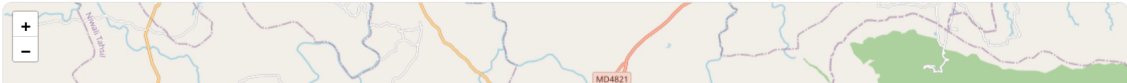
Rating: 5.0 ★

Total Rides: 0

[Go Offline](#)

[Available](#)

Your Current Location



Book a Cab

Pickup City

Destination City

Find Cabs

Show Map

Pickup City

Bhopal, Madhya Pradesh

Destination City

Indore, Madhya Pradesh

Find Cabs

Hide Map

I



Ride Details

Confirmed - Driver On The Way

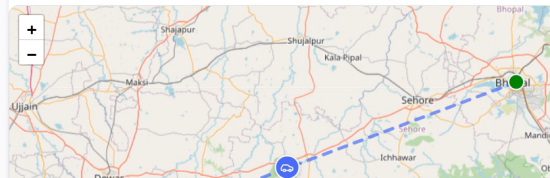
End Ride



Rahul Kumar

Maruti Swift - DL-01-AB-3184

Live Driver Location



12. KNOWN ISSUES

Despite a solid foundation, the current version of QuickCab has some limitations and areas for improvement:

- **No Payment Gateway:** Currently, there is no real-time payment integration. Users can book rides, but fare collection is not functional.
- **No Push Notifications:** Ride request alerts for drivers or ride status updates for passengers are not implemented.
- **Limited Real-Time Features:** While maps and status updates exist, true WebSocket-based real-time communication is not yet integrated.
- **Driver Verification:** There's no background verification or document upload mechanism for validating driver identity.
- **Admin Analytics:** Analytics are basic, with limited visualization or data filtering capabilities.
- **Ride Matching Algorithm:** Ride assignment is simplistic; there's no proximity-based smart matching between driver and passenger.
- **Lack of Multi-language Support:** Currently only available in English, limiting accessibility for diverse user groups.
- **Basic Error Handling:** Some frontend/backend components lack graceful handling of API/network failures.
- **Security Improvements Needed:** Although basic security is implemented, deeper threat modeling (e.g., rate limiting, brute-force detection) is missing.
- **Scalability Limitations:** The current backend is not optimized for horizontal scaling in high-traffic scenarios.

13. FUTURE SCOPE

The following enhancements can significantly boost the platform's usability, scalability, and feature richness:

Functional Enhancements

- **Payment Gateway Integration:** Integrate Razorpay, Stripe, or PayPal for seamless fare transactions.
- **Push Notification System:** Implement Firebase Cloud Messaging (FCM) for real-time alerts.
- **Real-Time Communication:** Use WebSockets or Socket.io for live ride updates, request notifications, and chat support.
- **Advanced Ride Matching:** Add geospatial querying (e.g., MongoDB's GeoJSON) for smarter driver allocation.
- **In-App Chat System:** Allow secure messaging between driver and passenger.

User Experience

- **Multi-language Support:** Use i18n libraries to support different languages based on user preference.
- **Dark Mode:** Add a toggle for light/dark UI themes for better accessibility.
- **Driver Verification System:** Enable KYC, license upload, and admin approval workflows.
- **Promo Codes & Loyalty Programs:** Offer discounts, cashback, and reward points.

Admin & Analytics

- **Advanced Dashboard:** Include charts, heatmaps, and ride trends using tools like Chart.js or Recharts.
- **Role-based Permissions:** Add granular access control for different admin staff roles (e.g., analytics viewer, operations manager).

Technical Enhancements

- **Microservices Architecture:** Refactor backend into scalable services (auth, ride, user, analytics).
- **Caching Layer:** Introduce Redis or similar for caching frequently accessed data (like cab types, fare rates).
- **Logging and Monitoring:** Use tools like ELK Stack or Prometheus + Grafana for real-time

system monitoring.

- **Load Balancing:** Implement NGINX or AWS ALB for better traffic distribution.