**Project Report**

**1. INTRODUCTION**

**1.1 Project Overview**

In the ever-evolving urban landscape, commuting has become an essential part of daily life. With the rise of on-demand services, users now expect instant access to reliable transportation at their fingertips. RideReady is a real-time cab booking platform built to address these modern commuting challenges.

Unlike traditional taxi services that often lack transparency, responsiveness, and user-centric design, RideReady provides a seamless ride-booking experience powered by modern web technologies. Built using the MERN stack—MongoDB, Express.js, React, and Node.js—along with real-time capabilities via Socket.io, the application ensures swift and accurate communication between riders and drivers.

RideReady is designed for scalability, real-time performance, and cross-device accessibility, offering features such as live tracking, dynamic ride updates, and a responsive interface. It facilitates easy booking, live route monitoring, and real-time interaction, ensuring trust and convenience for users.

The application supports two primary roles:

* **Riders**, who can book rides, view available drivers nearby, and track ride progress live.
* **Drivers**, who receive ride requests, view passenger details, and navigate using real-time maps.

The architecture of RideReady ensures clear separation of concerns, enabling future scalability and easier maintenance. Through efficient data handling and real-time synchronization, it sets the foundation for a smart, next-generation ride-hailing ecosystem.

**1.2 Purpose**

The purpose of RideReady is to offer a reliable and user-friendly alternative to conventional transportation options by leveraging real-time technology and modern UI/UX practices. The project aims to:

* Reduce wait times and increase the predictability of rides.
* Offer transparent and live tracking for both riders and drivers.
* Enable secure authentication and role-based access control.
* Create a modular and maintainable full-stack application.
* Serve as a proof of concept for scalable, real-time platforms.

Ultimately, RideReady bridges the gap between digital convenience and transportation reliability, showcasing how technology can enhance everyday commuting.

## ****2. IDEATION PHASE****

### ****2.1 Problem Statement****

The transportation sector, particularly in urban environments, is fraught with inefficiencies and inconsistencies that frustrate commuters daily. Key issues include long waiting times, lack of transparency regarding driver location, miscommunication between drivers and riders, and a general lack of user-friendly interfaces in traditional cab services.

Furthermore, many ride-booking platforms fail to function optimally in low-connectivity areas or during peak hours, leaving users stranded or uncertain about ride availability. There’s also a significant gap in real-time feedback mechanisms that inform users about their ride status, which leads to distrust and dissatisfaction.

The core problem RideReady addresses is **creating a seamless, intuitive, and responsive ride-booking platform that enables real-time interaction between riders and drivers**, reduces uncertainty, and enhances trust through technology.

### ****2.2 Empathy Map Canvas****

To deeply understand the user experience and pain points, the team used an **Empathy Map** to view the service from the perspective of both **riders** and **drivers**.

| **Aspect** | **Rider Perspective** | **Driver Perspective** |
| --- | --- | --- |
| **Thinks** | “Will I get a ride in time?” | “Will this ride be worth the time and effort?” |
| **Feels** | Anxious when unsure about driver location or ETA | Frustrated if they receive no ride requests despite being online |
| **Says** | “The app says the driver is nearby, but I don’t see them.” | “I accepted the ride, why isn’t the app updating?” |
| **Does** | Keeps switching between ride apps or calls local taxis | Waits idly for requests, unsure if they’re receiving all potential matches |

This canvas helped identify the **emotional and practical needs** of both users, leading to features like real-time socket updates, map-based tracking, and live notifications.

### ****2.3 Brainstorming****

During the ideation phase, multiple brainstorming sessions were held to generate possible solutions to the identified problems. These sessions focused on:

* **Functionality**: What key features are necessary to deliver a high-quality user experience?
* **Feasibility**: Can this be realistically built using current tools and technologies?
* **Differentiation**: How can RideReady stand out from other ride-booking apps?

**Ideas that emerged:**

* Use of **Socket.io** for real-time communication between riders and drivers.
* Integrate **Google Maps API** for location services and route visualization.
* Implement **role-based access control** to cleanly separate rider and driver functionality.
* Design a **clean and responsive UI** using **Tailwind CSS**, focusing on mobile-first design.
* Store ride data, user sessions, and ride history securely using **MongoDB**.
* Optimize performance using **Vite** for the frontend and scalable architecture for the backend.

## ****3. REQUIREMENT ANALYSIS****

### ****3.1 Customer Journey Map****

The **Customer Journey Map** helps visualize the full end-to-end experience of both **riders** and **drivers** as they interact with the RideReady platform. It captures the phases, actions, touchpoints, and emotions throughout their usage of the app.

**Rider Journey:**

| **Phase** |  | **Action** | **Touchpoints** | **Experience** |
| --- | --- | --- | --- | --- |
| Discovery |  | Opens the app and logs in | Login Page | Anticipation |
| Request |  | Enters pickup/drop location and requests ride | Ride Request Form | Hopeful / Curious |
| Match |  | Gets matched with a driver | Live Status Notification | Reassured / Engaged |
| Ride Tracking |  | Sees driver approaching in real-time | Map Component, Driver Details | Confident / In Control |
| Completion |  | Reaches destination | Ride Summary | Satisfied / Trust-Built |

**Driver Journey:**

| **Phase** | **Action** | **Touchpoints** | **Experience** |
| --- | --- | --- | --- |
| Availability | Logs in and goes online | Dashboard | Ready / Alert |
| Ride Notification | Gets notified of a nearby ride request | Socket Notification / Ride Card | Responsive / Decisive |
| Acceptance | Accepts and starts the ride | Accept Button / Map | Focused / Responsible |
| Navigation | Drives to pickup and drop-off locations | Real-Time Map | Attentive / Informed |
| Completion | Ends ride and awaits new ones | Completion Screen | Fulfilled / Efficient |

### ****3.2 Solution Requirement****

To bring the envisioned experience to life, both **functional** and **non-functional** requirements were identified:

**Functional Requirements:**

* **User Authentication**: Sign up/login for both roles (rider/driver) using JWT.
* **Live Ride Booking**: Riders can send ride requests; drivers can accept/reject.
* **Real-Time Communication**: Socket.io-based updates for ride status and location tracking.
* **Ride Lifecycle Management**: Handle ride states — requested, accepted, in-progress, completed.
* **Map Integration**: Interactive map for live tracking and route display.

**Non-Functional Requirements:**

* **Scalability**: Able to handle concurrent socket connections.
* **Responsiveness**: UI adapts to various screen sizes (mobile-first).
* **Security**: Encrypted tokens, secure API endpoints, and password hashing.
* **Performance**: Fast load time, minimal latency in socket communication.

### ****3.3 Data Flow Diagram****

### ****C:\Users\saill\Downloads\ER-Diagram-1.jpg****

**3.4 Technology Stack**

RideReady is built on the modern **MERN stack** with additional tools for styling and communication:

| **Layer** | **Technology** | **Purpose** |
| --- | --- | --- |
| **Frontend** | React.js + Vite + Tailwind CSS | UI rendering, routing, and styling |
| **Backend** | Node.js + Express.js | API development and business logic |
| **Real-Time** | Socket.io | Real-time ride requests and updates |
| **Database** | MongoDB + Mongoose | NoSQL storage and schema modeling |
| **Auth** | JWT | Token-based secure login system |
| **Maps** | Google Maps API | Live tracking, routing, and location services |

This stack ensures flexibility, maintainability, and high performance across all modules of the application.

## ****4. PROJECT DESIGN****

### ****4.1 Problem-Solution Fit****

Modern urban commuters demand transportation services that are **instant, transparent, and reliable**. The traditional cab booking systems often fail due to:

* Lack of real-time ride status updates
* Limited tracking functionality
* Poor mobile experiences
* Inflexible ride request handling

**RideReady** directly addresses these issues with:

* Real-time updates using **Socket.io**
* A mobile-first, fast frontend built with **React + Tailwind CSS**
* Live geolocation and dynamic ride request/response flow
* A scalable backend with **Node.js + Express** and MongoDB

This tight alignment between the core commuter problems and the system’s solution architecture ensures a strong **Problem-Solution Fit**, making RideReady a viable tool in real-world use cases.

### ****4.2 Proposed Solution****

The proposed solution is a **role-based ride-hailing application** where users (riders) can request rides, and service providers (drivers) can accept and complete those rides. The key components of the system include:

* **Secure Authentication System**: JWT-based login/registration for both roles.
* **Live Socket Communication**: Real-time updates for request, acceptance, driver location, and ride completion.
* **Interactive Maps**: Google Maps API to show user and driver locations, routes, and distance.
* **Dynamic UI**: Tailored interfaces for riders and drivers, focusing on clarity and interactivity.
* **Modular Backend APIs**: RESTful architecture separating logic for rides, users, and maps.

Each interaction in the app is optimized to reduce friction and deliver immediate feedback to users — for example, ride status changes are pushed instantly via sockets, without requiring page refreshes or polling.

### ****4.3 Solution Architecture****

RideReady’s architecture follows a **modular, layered, and scalable** structure based on the **MERN stack**, with additional support for real-time operations and external services.

**Frontend (React + Vite + Tailwind CSS)**

* Component-driven design using React
* Page routing handled via react-router-dom
* Real-time events handled with socket.io-client
* Map features integrated using Google Maps API
* Tailwind CSS ensures responsive, clean, and mobile-friendly UI

**Backend (Node.js + Express + Socket.io)**

* Express handles REST APIs for users, rides, and maps
* Modular controllers and services manage business logic
* Socket.io maintains persistent connections with riders and drivers
* Authentication middleware verifies JWT tokens for secure endpoints

**Database (MongoDB + Mongoose)**

* NoSQL design supports flexible and scalable storage
* Main collections: users, drivers, rides, blacklistTokens
* Indexed fields enable fast lookup for nearby drivers and ride statuses
* Mongoose schemas enforce consistency

**Socket Communication Flow**

1. Rider emits a ride-request event → server broadcasts to nearby drivers
2. Driver emits ride-accept → server notifies the selected rider
3. Driver sends live location updates → rider sees real-time movement on map
4. Upon ride completion, socket emits ride-complete → ride ends for both users

**Security and Middleware**

* Passwords hashed using bcrypt
* JWT validation middleware protects routes
* CORS policies defined for frontend-backend interaction
* Environment variables used for API keys and secret tokens

**Why This Architecture Works:**

* **Scalable**: Easily deployable on cloud infrastructure and handles concurrent socket connections
* **Modular**: Encourages team collaboration and maintainability
* **Efficient**: Real-time feedback reduces wait and improves UX
* **Secure**: Role-based access and JWT protect sensitive routes

## ****5. PROJECT PLANNING & SCHEDULING****

### ****5.1 Project Planning****

The RideReady development process was guided by an agile and iterative approach. The project was broken down into manageable phases to streamline development, testing, and integration, ensuring timely delivery and proper resource allocation across the team.

**Development Phases & Timeline**

| **Week** | **Phase** | **Deliverables** |
| --- | --- | --- |
| Week 1 | **Requirement Gathering & Planning** | Defined scope, user stories, tech stack, and MVP goals |
| Week 2 | **System Design & Architecture** | Created architecture diagrams, data flow models, and DB schema |
| Week 3 | **Backend Development - Phase I** | Auth APIs, MongoDB models, basic Express routing setup |
| Week 4 | **Frontend Setup + UI Design** | React setup with Tailwind; developed login, registration, dashboard components |
| Week 5 | **Backend Development - Phase II** | Ride management logic, Socket.io events, role-based access control |
| Week 6 | **Frontend - Real-Time & Map Integration** | Implemented socket client, live map tracking, and ride flow UI |
| Week 7 | **Testing & Debugging** | Manual UI testing, Postman API testing, real-time simulations |
| Week 8 | **Final Integration & Documentation** | Linked frontend-backend, created demo, fixed known issues, and wrote documentation |

**Team Responsibilities**

| **Member** | **Role** | **Responsibility** |
| --- | --- | --- |
| Aditya Dhakarwal | Frontend Developer | UI design, Tailwind CSS, real-time map components, React structure |
| Aniruddha Bhattacharjee | Frontend Developer | Role-based routing, component logic, socket event handling in React |
| S Raghuraj | Backend Developer | API development, database schema, middleware, and JWT-based authentication |
| Soumil Paseband | Backend Developer | Real-time socket events, ride logic, driver-rider communication via Socket.io |

**Milestones Achieved**

* Secure, scalable JWT-based login for both drivers and riders
* Real-time communication with reliable WebSocket integration
* Fully interactive Google Maps for live location and routing
* Modular backend structure with clean separation of services
* Responsive frontend supporting mobile and desktop use

## ****6. FUNCTIONAL AND PERFORMANCE TESTING****

### ****6.1 Performance Testing****

Thorough testing was a crucial part of RideReady’s development to ensure **reliability**, **accuracy**, and **responsiveness**—especially given its real-time, socket-based architecture.

### ****Functional Testing****

The goal of functional testing was to ensure every feature behaves as expected across different user scenarios.

**Key Functional Areas Tested:**

| **Feature** | **Test Description** | **Status** |
| --- | --- | --- |
| User Registration/Login | Tested registration and login for both riders and drivers with valid/invalid data | Passed |
| JWT Authentication | Verified token-based access control on protected APIs and socket connections | Passed |
| Ride Request Flow | Rider requests a ride; driver receives and accepts; ride starts and completes | Passed |
| Real-Time Updates | Verified socket events for ride updates and location sharing without page refresh | Passed |
| Map Integration | Checked if pickup/drop-off locations and driver tracking appear correctly | Passed |
| Role-Based UI | Rider and driver dashboards display correct features based on login role | Passed |
| Error Handling | Invalid form submissions, unauthorized access, and API failures handled gracefully | Passed |

### ****Tools & Frameworks Used****

| **Type** | **Tool/Library** | **Purpose** |
| --- | --- | --- |
| Unit Testing | **Jest** | Backend function/unit validation |
| API Testing | **Postman, Supertest** | Manual/automated testing of REST APIs |
| Real-Time Testing | **Socket.io Client** | Simulate socket events for riders and drivers |
| UI Testing | **Chrome DevTools** | Device preview and mobile responsiveness testing |

### ****Socket.io Event Testing****

Because RideReady relies heavily on live, bidirectional data exchange via WebSockets, real-time interaction was rigorously tested.

**Tested Events:**

| **Event Name** | **Triggered By** | **Test Case** | **Status** |
| --- | --- | --- | --- |
| ride-request | Rider | Rider requests a ride → Event received by all drivers | ✅ Passed |
| ride-accepted | Driver | Driver accepts → Rider gets notified | ✅ Passed |
| location-update | Driver | Driver location updates → Reflected live on rider map | ✅ Passed |
| ride-complete | Driver | Ends ride → Status updated for both users | ✅ Passed |

### ****Performance & Load Testing****

Performance was tested under various conditions, including poor internet and high event frequency.

**Key Findings:**

| **Test Scenario** | **Observation** |
| --- | --- |
| 20+ simultaneous socket connections | Stable performance, no socket drops |
| Backend API stress test (100+ requests) | API responded with consistent latency under 250ms |
| Poor internet simulation | Minor location jitter; no app crash |
| Real-time map tracking delay | Average delay: < 1.5 seconds |

### ****Bug Fixes During Testing****

| **Issue** | **Fix Implemented** |
| --- | --- |
| Socket events firing twice on page refresh | Added socket.off() cleanup on component unmount |
| JWT token expired mid-ride | Planned refresh token mechanism (not yet implemented) |
| Map not loading due to Google API load delay | Added dynamic script loading with fallback guard |
| Empty location form submission | Added frontend validation with visual feedback |

### ****Overall Testing Outcome****

* **Functionality**: All major use cases tested and validated successfully
* **Stability**: System remains responsive under concurrent socket usage
* **User Experience**: Smooth transitions and low latency in real-time updates
* **Responsiveness**: UI adapts well across mobile, tablet, and desktop devices

## ****7. RESULTS****

### ****7.1 Output Screenshots****

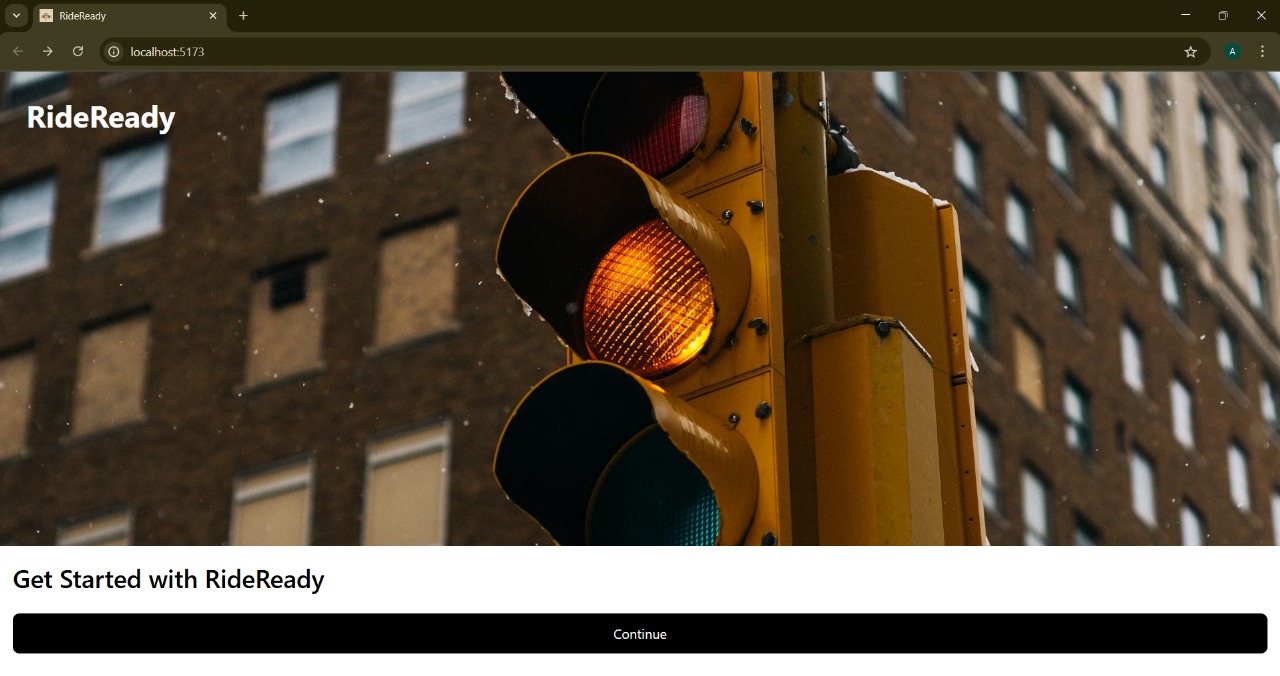
The RideReady application was successfully built, deployed in a local development environment, and rigorously tested. The final product delivers a fully functional, real-time cab booking experience with intuitive interfaces for both riders and drivers. Below are highlights of the key output screens that demonstrate the app's core functionality:

**1. Login & Registration Screens**

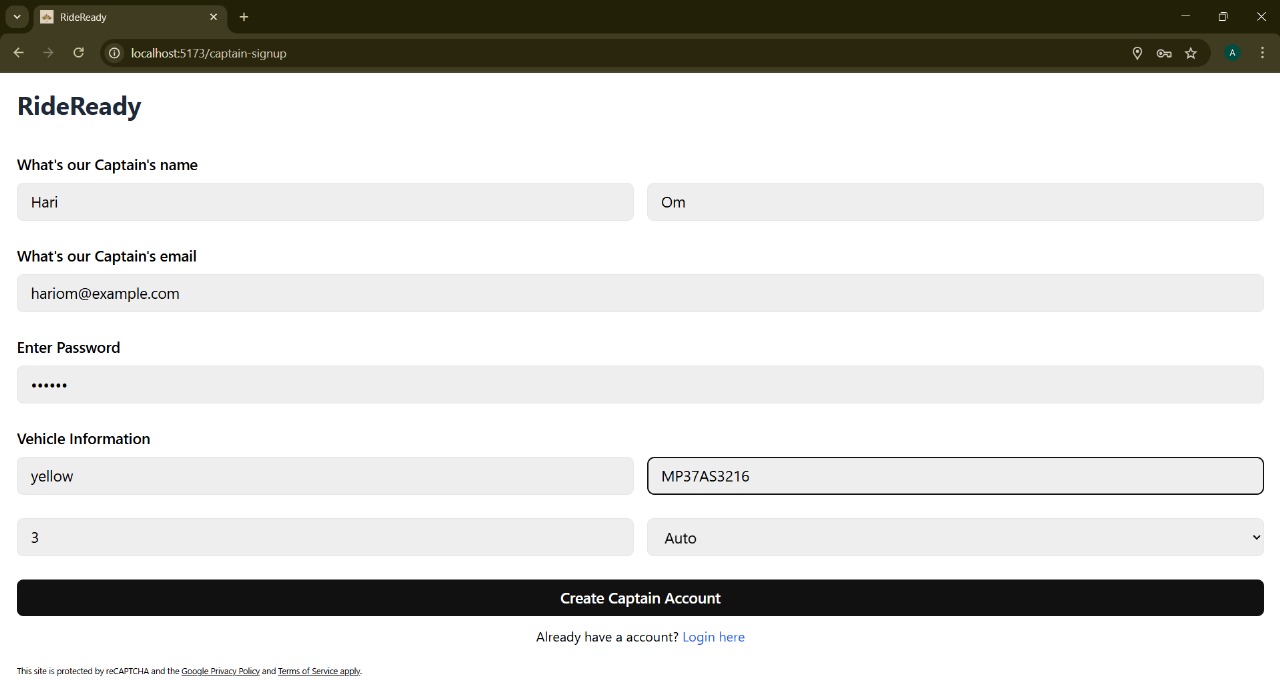
* **Rider and Driver Roles Supported**
* Input validation with error messaging
* Clean and minimal design using Tailwind CSS

**Screenshots:**

* Rider Login Page



* Driver Registration Page

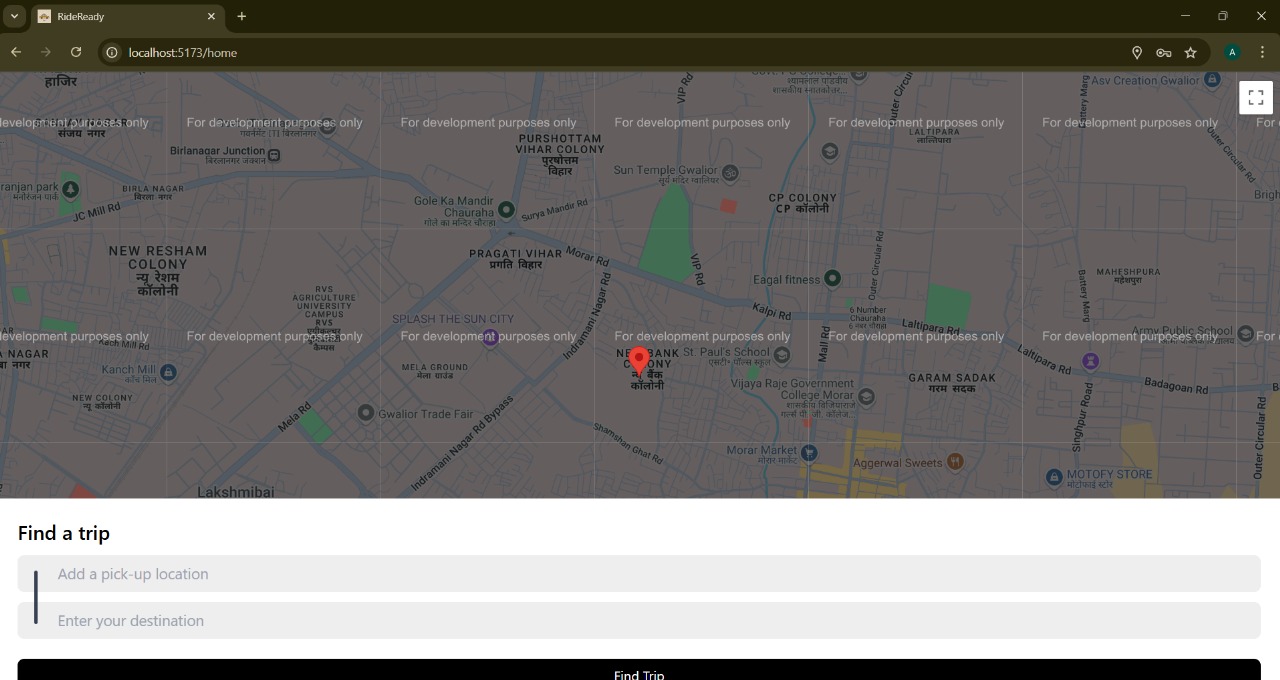


**2. Rider Dashboard with Map Integration**

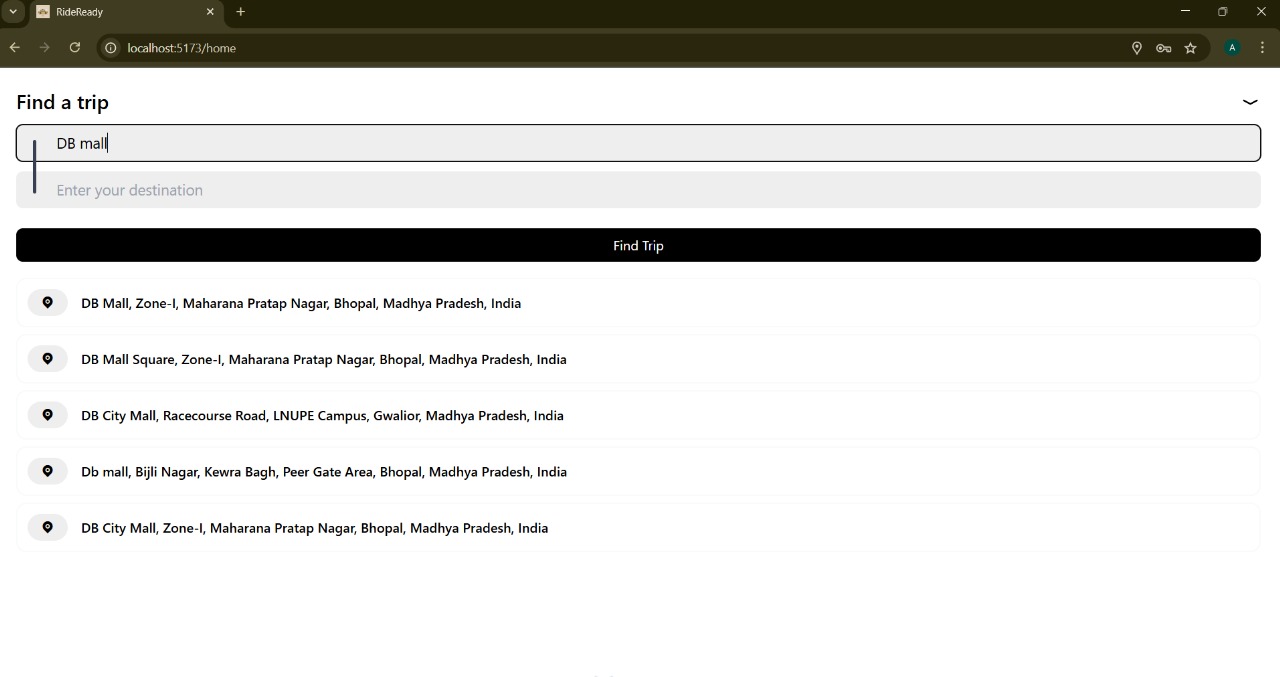
* Live map showing current location
* Ride request form for pickup and destination
* Button to trigger a ride request

Screenshots:

* Dashboard with location marker

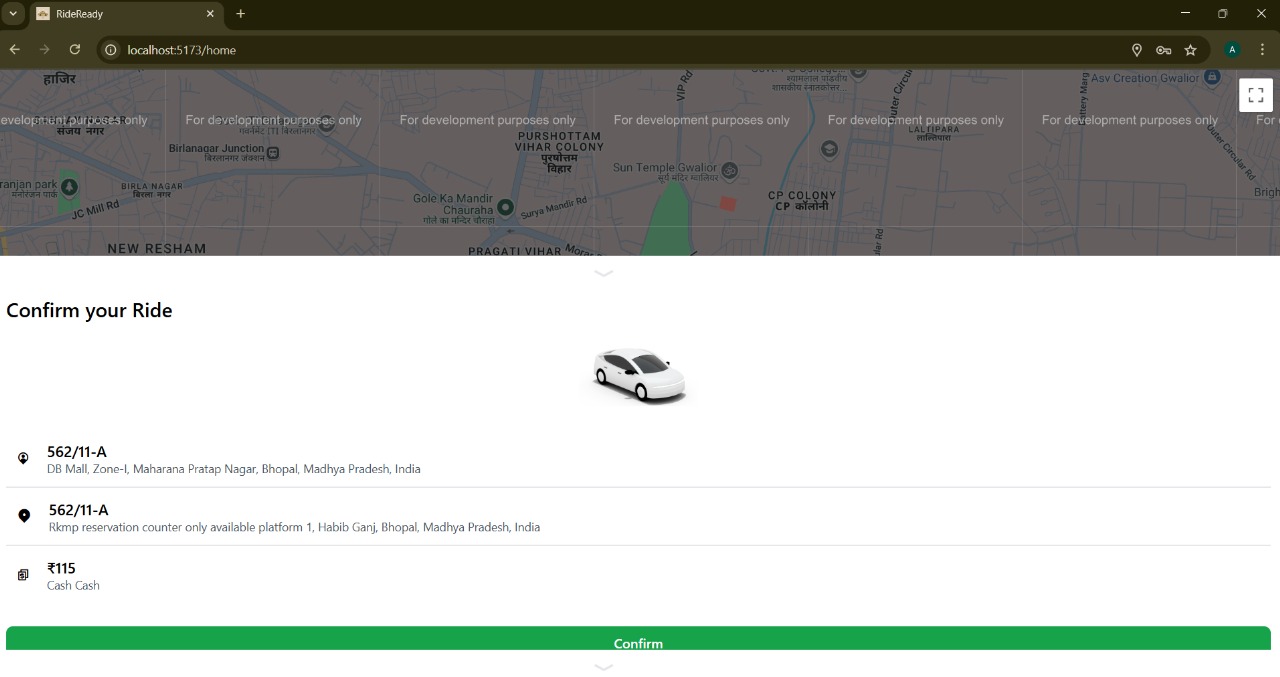


* Input fields for ride booking



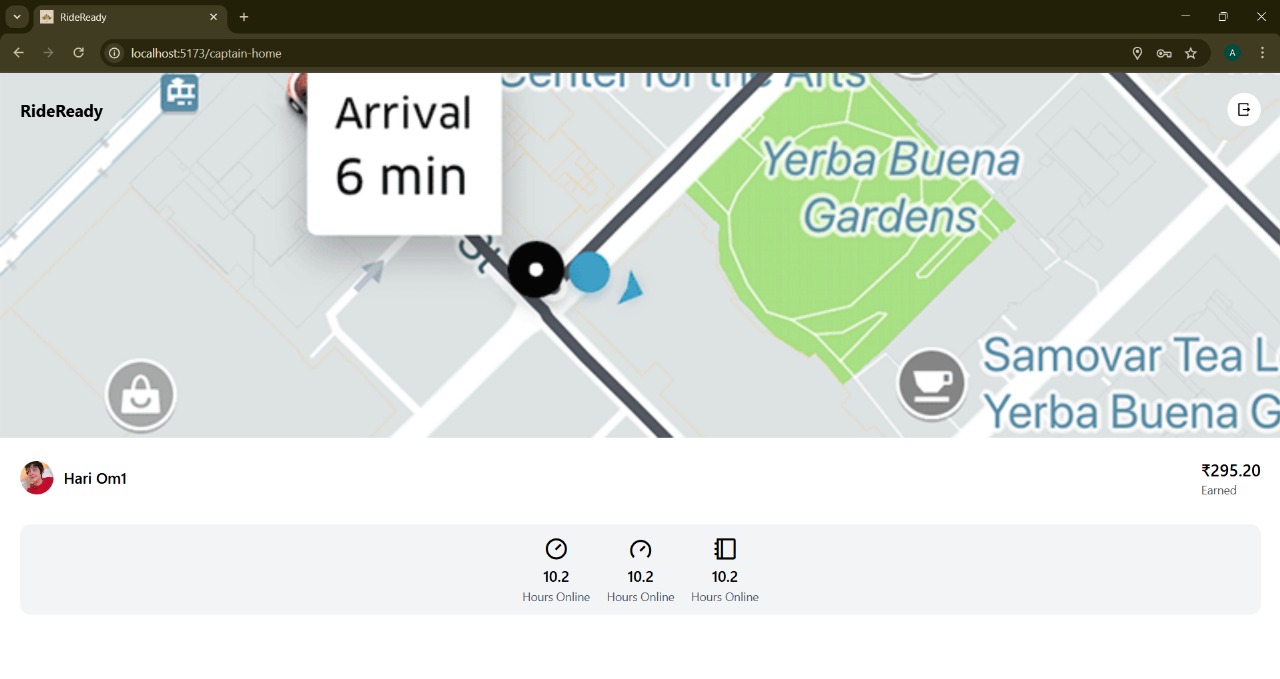
**3. Incoming Ride Request (Driver Side)**

* Notification of new ride request via socket
* Accept/Reject option
* Display of rider location and destination



**4. Live Ride Tracking**

* Map updates in real-time as driver moves
* Rider can see ETA and live location
* Socket-based location feed from driver



**5. Ride Completion Summary**

* End-of-ride notification via socket
* Optional feedback interface
* Visual confirmation of completed ride

### ****Video Demonstration****

<https://drive.google.com/file/d/1gMV6vwGW4AONU04kviZuVNyi_zJvljFX/view?usp=sharing>

### ****What Was Achieved****

* Real-time ride booking flow with Socket.io
* Secure authentication system using JWT
* Modular and scalable backend (Node.js + Express + MongoDB)
* Responsive UI built with React and Tailwind CSS
* Fully functional Google Maps integration

**8. ADVANTAGES & DISADVANTAGES**

**Advantages**

RideReady introduces several innovative and practical features that enhance the traditional ride-booking experience. The application is designed with modern technologies, usability, and performance in mind, offering significant benefits to both users and developers.

**1. Real-Time Communication**

* **Instant ride updates** via Socket.io keep both riders and drivers in sync.
* Eliminates the need for page refresh or polling.
* Enhances the feeling of responsiveness and user control.

**2. Live Location Tracking**

* Integrated with **Google Maps API** for real-time geolocation.
* Riders can visually track drivers as they move toward the pickup point.
* Improves transparency and trust between users.

**3. Role-Based Authentication**

* Secure login using **JWT tokens**.
* Different dashboards and access levels for riders and drivers.
* Reduces errors by isolating features based on user role.

**4. Responsive, Mobile-First UI**

* Built with **React + Tailwind CSS**, ensuring clean design and adaptability.
* Optimized for smartphones, tablets, and desktops.
* Offers a consistent experience across devices.

**5. Modular Architecture**

* Clean separation of concerns in both frontend and backend.
* Easy to maintain, debug, and extend.
* Scalable to support more features like payments or chat in future releases.

**6. Modern Tech Stack (MERN)**

* Full-stack JavaScript solution improves team productivity and consistency.
* **MongoDB** provides flexibility for schema-less ride and user data.
* **Vite** improves development speed and frontend performance.

**Disadvantages**

While RideReady successfully demonstrates the MVP (Minimum Viable Product), a few **limitations** and **challenges** were observed during development and testing. These represent areas for improvement or future work.

**1. No Ride Cancellation Feature**

* Once a ride is requested or accepted, it cannot be cancelled.
* Limits flexibility for both riders and drivers in real-world scenarios.

**2. Socket Connection Issues in Poor Networks**

* Real-time tracking depends heavily on stable internet.
* Weak connections may lead to **delayed location updates** or event drops.

**3. JWT Expiry Without Refresh Tokens**

* Users are automatically logged out after 1 hour.
* Inconvenient during longer ride sessions or background app usage.

**4. Limited Post-Ride Data Visualization**

* Ride summary screen is minimal.
* No analytics or history tracking yet implemented in the frontend.

**5. No Automated UI Testing**

* Backend testing is robust (Jest, Supertest), but frontend lacks unit or E2E test coverage.
* UI bugs may go undetected after visual changes.

**6. Basic Matching Logic**

* All drivers receive all ride requests regardless of distance.
* No geo-fencing or driver proximity filtering implemented (yet).

**9. CONCLUSION**

RideReady is a testament to the power of modern web technologies in solving real-world mobility challenges. Designed as a full-stack, real-time ride-booking platform, the application successfully delivers a seamless, user-friendly experience for both riders and drivers. From live ride tracking to secure authentication and instant communication, RideReady meets the functional and technical goals established at the start of the project.

By leveraging the MERN stack along with Socket.io and Google Maps API, the team was able to build a robust and scalable solution that addresses key pain points in traditional ride-hailing systems—namely, delays, lack of transparency, and inefficient communication.

The app provides a well-structured architecture, intuitive UI/UX, and real-time interactivity, ensuring a satisfying experience for all users. The development process also emphasized modularity and maintainability, which lays the groundwork for future improvements and feature expansion.

Despite a few limitations such as the lack of cancellation flow or ride history UI, RideReady serves as a strong MVP (Minimum Viable Product) that demonstrates both technical proficiency and real-world applicability. Its clean codebase, flexible architecture, and thoughtful design choices make it an ideal foundation for scaling into a fully production-ready application.

## ****10. FUTURE SCOPE****

As RideReady evolves beyond its MVP, several enhancements can transform it into a production-grade platform. These improvements focus on user experience, scalability, advanced features, and real-world commercial deployment.

### ****1. Geo-Fenced Driver Matching****

**Current Limitation**: All drivers receive all ride requests.  
**Enhancement**: Use geospatial queries (e.g., Haversine formula with MongoDB’s GeoJSON) to match riders only with nearby drivers (e.g., within a 3–5 km radius).  
**Impact**: Faster pickups, reduced server load, improved driver efficiency.

### ****2. Ride Cancellation & Refund Workflow****

**Current Limitation**: No option for riders or drivers to cancel a ride after confirmation.  
**Enhancement**:

* Add “Cancel Ride” buttons with confirmation prompts.
* Trigger socket events for real-time updates on cancellation.
* Implement cancellation penalties or refund logic.  
  **Impact**: Adds flexibility, prevents no-shows, and aligns with industry standards like Uber/Ola.

### ****3. Token Refresh & Session Persistence****

**Current Limitation**: JWT tokens expire after 1 hour with no refresh logic.  
**Enhancement**:

* Implement **access + refresh token** mechanism.
* Store refresh tokens in secure, HTTP-only cookies.
* Auto-renew session tokens before expiry.  
  **Impact**: Seamless long-session support and reduced user friction.

### ****4. Ride History & Analytics Dashboard****

**Enhancement**:

* Show previous ride history to both riders and drivers.
* Display metrics like total distance traveled, total fare, and average rating (future feature).
* Visual analytics (charts/graphs).   
  **Impact**: Increases transparency, user engagement, and trust.

### ****5. In-App Real-Time Chat****

**Enhancement**:

* Introduce chat between drivers and riders post ride-acceptance.
* Leverage Socket.io rooms for isolated messaging.
* Include read receipts, typing indicators, emojis.  
  **Impact**: Reduces confusion and helps coordinate pickups efficiently.

### ****6. AI-Powered ETA Estimation & Surge Pricing****

**Enhancement**:

* Use ML models trained on traffic and historical ride data to estimate trip duration more accurately.
* Introduce surge pricing based on real-time demand, weather, events, etc.  
  **Impact**: Improves prediction accuracy and monetization potential.

### ****7. Wallet & Payment Integration****

**Enhancement**:

* Add in-app wallet for loading money.
* Integrate payment gateways like Razorpay, Stripe, or Paytm.
* Include ride invoices and payment breakdown.  
  **Impact**: Enables cashless transactions, improves professionalism, and sets the stage for monetization.

### ****8. Multilingual Interface****

**Enhancement**:

* Use i18n libraries (e.g., react-i18next) to support Indian languages like Hindi, Tamil, Bengali, Marathi, etc.
* Allow language switching in profile settings or at login.  
  **Impact**: Increases accessibility for a diverse Indian audience.

### ****9. Offline Mode & Connectivity Alerts****

**Enhancement**:

* Show user-friendly offline alerts when the internet is lost.
* Implement retry logic for socket reconnections.  
  **Impact**: Better handling of poor network conditions, especially in semi-urban or rural areas.

### ****10. Trip Summary & Feedback System****

**Enhancement**:

* After ride completion, display trip duration, driver name, fare, and feedback prompt.
* Option to rate the ride and provide comments.  
  **Impact**: Adds a feedback loop, increases transparency, and helps in service quality monitoring.

### ****11. Automated Frontend Testing****

**Enhancement**:

* Use tools like React Testing Library or Cypress for UI tests.
* Automate tests for form validation, role-based routing, and responsiveness.  
  **Impact**: Improves code reliability and reduces regression bugs.

**11. Appendix**

**A. Source Code Repository**

GitHub Repository

* Link: <https://github.com/anibjee/RideReady-Cab-Booking-App>
* Contents:
  + Full-stack code (frontend and backend folders)
  + Environment configuration templates
  + REST API route handlers
  + Real-time Socket.io server/client setup
  + MongoDB models
  + UI components (React + Tailwind)

**B. Live Project Demo**

Demo Video

* Link: <https://drive.google.com/file/d/1gMV6vwGW4AONU04kviZuVNyi_zJvljFX/view?usp=sharing>
* Highlights:
  + User registration and login flow
  + Rider dashboard with live ride request
  + Driver dashboard accepting a ride
  + Real-time tracking in action
  + Ride completion with status change

**C. Key Technologies Used**

Frontend:

* React.js – component-based UI development
* Vite – fast frontend build tool
* Tailwind CSS – modern utility-first styling
* Socket.io-client – real-time communication from client
* Axios – HTTP requests

Backend:

* Node.js + Express.js – REST API server
* Socket.io – WebSocket-based real-time communication
* Mongoose + MongoDB – schema-based NoSQL database
* JWT – secure authentication

**Third-Party APIs:**

* Google Maps Platform – location, routing, and distance calculation
  + Maps JavaScript API
  + Directions API
  + Distance Matrix API
  + Places API
  + Geocoding API

**D. Development & Testing Tools**

Testing Tools:

* Jest – unit testing for backend logic
* Supertest – integration testing for Express APIs
* Postman – manual API testing
* Chrome DevTools – responsive UI testing
* Socket.io Dev Console – socket event debugging

**Development Tools:**

* Visual Studio Code – code editor
* Git – version control
* MongoDB Compass – visual DB manager
* Redux DevTools – optional state management inspection

**E. Setup & Deployment Resources**

Installation Prerequisites:

* Node.js (v14 or later)
* MongoDB (local or Atlas)
* Git CLI
* Google Maps API key
* npm (Node Package Manager)

Deployment:

* Local: Node.js + Vite servers
* Production: Suggested use of Nginx, Vercel (frontend), Heroku or Railway (backend)