Software Requirements Specification

for

APNI RIKSHA

Version 1.0 approved

Prepared by
KAUSTAV DAS (23DCE020)
KURESH GARBADA (23DCE032)
BHAVYA GODHAVIYA (23DCE036)
HARSH GOSWAMI (23DCE037)

Devang Patel Institute of Advance Technology and Research

01-02-2025

Table of Contents

Revision History

Name	*Date*	*Reason For Changes*	*Version*

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements for the development of Apni Riksha, an auto-rickshaw booking system designed to simplify the process of booking auto-rickshaws for hostel students and local residents. This document covers the scope of the product, including its features, user interactions, and system architecture. The purpose of this document is to serve as a reference for developers, testers, project guide to ensure that the software meets the intended objectives and delivers a seamless user experience.

1.2 Document Conventions

This document follows the IEEE 830-1998 standard for writing Software Requirements Specifications. The following conventions are used throughout the document:

- ***Bold Text**: Used for section headings and subheadings.*
- *Italic Text: Used for emphasis or to highlight key terms.*
- *Monospace Font: Used for technical terms, code snippets, or file names.*
- ***Priorities**: Requirements are prioritized as follows:*
  - ***High (H)**: Must be implemented in the current release.*
  - ***Medium (M)**: Should be implemented if time permits.*
  - ***Low (L)**: Can be deferred to future releases.*

1.3 Intended Audience and Reading Suggestions

This document is intended for the following audiences:

1. ***Developers**: To understand the system requirements and design the software accordingly.*
2. ***Testers**: To create test cases and validate the system against the specified requirements.*
3. ***Project Guide**: To plan and monitor the project's progress and ensure alignment with the requirements.*
4. ***Documentation Writers**: To create user manuals and technical documentation.*

.

1.4 Project Scope

Apni Riksha is a mobile web-based auto-rickshaw booking system designed to provide a convenient and reliable solution for hostel students and local residents to book auto-rickshaws without hassle. The system will allow users to register, log in, and book rides in real-time, while drivers can register themselves, accept ride requests. The platform will include features such as real-time tracking and fare estimation.
The primary objectives of this project are:

- *To simplify the auto-rickshaw booking process for users.*
- *To provide drivers with a steady source of income and flexibility in their work.*
- *To reduce the time and effort required to book rides during emergencies.*
- *To create a scalable and user-friendly platform using the MERN stack.*

This project aligns with the broader goal of leveraging technology to improve local transportation and enhance the quality of life for students and residents.

1.5 References

The following documents and resources were referenced during the creation of this SRS:

1. ***IEEE 830-1998**: IEEE Recommended Practice for Software Requirements Specifications.*
2. ***Google Maps API Documentation**: https://developers.google.com/maps/documentation*
3. ***Socket.IO Documentation**: https://socket.io/docs/*
4. ***MERN Stack Overview**: https://www.mongodb.com/mern-stack*
5. ***React.js Documentation**: https://reactjs.org/docs/getting-started.html*
6. ***Node.js Documentation**: https://nodejs.org/en/docs/*
7. ***Express.js Documentation**: https://expressjs.com/*
8. ***MongoDB Documentation**: https://docs.mongodb.com/*

2. Overall Description

2.1 Product Perspective

- ***Apni Riksha** is a **new, self-contained product** designed to address the challenges faced by hostel students and local residents in booking auto-rickshaws conveniently and efficiently. The system is not part of an existing product family or a replacement for any current system but is an innovative solution tailored to meet the specific needs of its target audience.*

- ***Context and Origin***
  *The idea for **Apni Riksha** originated from the frequent difficulties faced by hostel students and local residents in securing auto-rickshaws during emergencies or peak hours. Traditional methods of booking auto-rickshaws, such as hailing them on the street or relying on phone calls, are often unreliable and time-consuming.*

2.2 Product Features

***Apni Riksha** is designed to provide a seamless and user-friendly experience for booking auto-rickshaws. The system offers a range of features for users, drivers, and administrators, ensuring efficient ride management and real-time communication. Below is a high-level summary of the major features:*

- ***User Features***

  - ***User Registration and Login**:*
    Users can create an account and log in using their email or phone number with OTP-based verification.
  - ***Ride Booking**:*
    Users can book an auto-rickshaw by specifying their pickup and drop locations. The system provides real-time fare estimation.

- ***Real-Time Tracking***:
  Users can track the location of their booked auto-rickshaw in real-time using Google Maps integration.
- ***Emergency Booking***:
  Priority booking feature for urgent ride requests.

- ***Driver Features***

  - ***Driver Registration and Login***:
    Drivers can register themselves by providing necessary documents (license, vehicle details, etc.) and log in to the system.
  - ***Ride Requests***:
    Drivers receive real-time ride requests via ***Socket.IO*** *and can accept or reject them based on availability.*
  - ***Earnings Dashboard***:
    Drivers can track their daily/weekly earnings and manage withdrawals.
  - ***Navigation Assistance***:
    Integrated Google Maps for optimal route navigation.

- ***System Features***

  - ***Real-Time Communication:***
    Real-time ride requests and notifications using Socket.IO.
  - ***Google Maps Integration:***
    For location tracking, route optimization, and fare estimation.

2.3 User Classes and Characteristics

Apni Riksha caters to multiple user classes, each with distinct characteristics and requirements. These user classes are differentiated based on their roles, frequency of use, technical expertise, and access levels. Below is a detailed description of the user classes and their pertinent characteristics:

- ***Hostel Students and Local Residents (Primary Users):***

  - *Characteristics:*
    - *Frequency of Use: High (daily or weekly use for commuting).*
    - *Technical Expertise: Basic to intermediate (familiar with mobile/web applications).*
    - *Primary Functions: Ride booking, real-time tracking, payment, and ride history.*

- ***Auto Rickshaw Drivers (Service Providers)***

  - *Characteristics:*
    - *Frequency of Use: High (daily or weekly use for commuting).*
    - *Technical Expertise: Basic (may require training to use the system effectively).*

- *Primary Functions: Ride requests, navigation, earnings tracking, and ride history.*

- ***Developers and Testers (Technical Team)***

  - *Characteristics:*
    - *Frequency of Use: High (daily use during development and testing phases).*
    - *Technical Expertise: Advanced (proficient in software development and testing).*
    - *Primary Functions: System development, debugging, and testing.*
    - *Educational Level: College degree or higher (technical background).*

2.4 Operating Environment

*The **Apni Riksha** auto-rickshaw booking system is designed as a **mobile web-based application** that operates across various environments. The software will be compatible with multiple devices and platforms to ensure accessibility and seamless functionality.*

1. ***Hardware Requirements***

- ***Client Devices:** Smartphones, tablets, laptops, and desktops.*
- ***Server Requirements:***
  - *Cloud-based or dedicated server for backend deployment.*
  - *Minimum **4-core CPU, 8GB RAM, 100GB storage** for hosting services.*
- ***Network:** Stable **internet connection (4G/5G, Wi-Fi)** for smooth operation.*

2. ***Software Requirements***

- ***Frontend:** React.js, Tailwind CSS (for UI design).*
- ***Backend:** Node.js with Express.js.*
- ***Database:** MongoDB (NoSQL database).*
- ***Real-Time Features:** Socket.io for instant updates.*
- ***API Integration:***
  - *Maps API for live tracking.*
  - *Payment gateway (future enhancement).*

3. ***Operating System Compatibility***

- ***Client Side:***
  - *Windows, macOS, Linux (for web access).*
  - *Android & iOS (mobile browsers like Chrome, Safari).*

- ***Server Side:***
  - *Ubuntu 20.04+ / Windows Server (for backend deployment).*
  - *Node.js runtime environment (v16+).*

4. Other Dependencies & Tools

- ***Version Control:*** *Git & GitHub for code management.*
- ***Development & Testing:*** *Postman (API testing), Jest (for unit testing).*
- ***Hosting Services:*** *Railway, Vercel, or AWS for deployment.*

2.5 Design and Implementation Constraints

*The development of **Apni Riksha**, a mobile web-based auto-rickshaw booking system, is subject to several **constraints and limitations** that must be considered during the design and implementation phases.*

1. ***Regulatory & Compliance Constraints***
   - Must comply with **local transportation laws** and **data privacy regulations** (e.g., **IT Act, GDPR** for user data protection).
   - Ensuring **secure payment transactions** if integrated with third-party gateways.

2. ***Hardware & Performance Constraints***
   - *The application must run efficiently on low-end devices with limited CPU/RAM capacity.*
   - *Real-time tracking and socket-based communication should work with minimal network latency.*

3. ***Software & Technology Constraints***
   - *Android & iOS (mobile browsers like Chrome, Safari).*
   - *Maps API limitations, such as rate limits on geolocation requests.*
   - ***Limited backend resources** on cloud hosting (e.g., free-tier restrictions on Railway or Vercel).*

2.6 User Documentation

*The following **user documentation** will be provided with the **Apni Riksha** system to ensure smooth onboarding and usability:*

1. ***User Manual** – A step-by-step guide on booking rides, payment, and driver registration.*
2. ***Online Help Section** – FAQs and troubleshooting tips within the app.*
3. ***Video Tutorials** – Short demo videos explaining key features.*

4. ***API Documentation*** *– For future integrations and developer reference.*

2.7 Assumptions and Dependencies

- ***Assumptions***
  - 📶 ***Users will have stable internet access*** *(Wi-Fi, 4G, or 5G) for smooth booking and tracking.*
  - 📍 ***Google Maps API will be available*** *without major disruptions for location services.*
  - 📱 ***Users will access the system via modern browsers*** *(Chrome, Safari, Edge) on mobile and desktop.*
  - 🛠️ ***Auto drivers will be willing to adopt digital booking technology*** *for better income opportunities.*
  - 🔐 ***User authentication (JWT, OAuth) will be secure and functional*** *for login and ride requests.*

- ***Dependencies***

  - ***Google Maps API*** *– For real-time tracking and navigation.*
  - ***MERN Stack Components*** *– MongoDB, Express.js, React.js, and Node.js for full-stack development.*
  - ***Cloud Hosting (Railway/Vercel/AWS)*** *– For backend and frontend deployment.*
  - ***Socket.io*** *– For real-time updates between drivers and passengers.*
  - ***Third-Party Payment Gateway (Future Enhancement)*** *– For digital transactions.*

3. System Features

3.1 Ride Booking System

- ***3.1.1 Description and Priority***

- *Allows users to book auto-rickshaws in real time.*
- ***Priority:*** *High*

- ***3.1.2 Stimulus/Response Sequences***

1. *User enters pickup and drop locations.*
2. *System finds nearby auto drivers.*
3. *User confirms booking; driver gets notified.*
4. *Driver accepts, and ride details are shared.*

- ***3.1.3 Functional Requirements***

- ***REQ-1:*** *Users can search for rickshaws based on location.*
- ***REQ-2:*** *System notifies available drivers in real time.*
- ***REQ-3:*** *Ride details (fare, driver info) are displayed before confirmation.*

3.2 Live Tracking

- ***3.2.1 Description and Priority***

- *Enables users to track their booked rickshaw in real time.*
- ***Priority:*** *High*

- ***3.2.2 Stimulus/Response Sequences***

1. *User confirms booking.*
2. *System shows driver's live location.*
3. *Ride status updates in real time*

.

- ***3.2.3 Functional Requirements***

- ***REQ-1:*** *System integrates Google Maps for tracking.*
- ***REQ-2:*** *Users receive ETA updates dynamically.*
- ***REQ-3:*** *Drivers can update ride status (Arrived, On Trip, Completed).*

3.3 Driver Registration & Ride Management

- ***3.3.1 Description and Priority***

- *Allows auto drivers to sign up, receive bookings, and manage rides.*
- ***Priority:*** *High*

- ***3.3.2 Stimulus/Response Sequences***

1. *Driver registers with required details.*
2. *System verifies identity.*
3. *Drivers get ride requests and accept bookings*

.

- ***3.3.3 Functional Requirements***

- ***REQ-1:*** *Drivers can register and log in securely.*
- ***REQ-2:*** *System verifies driver documents (TBD).*
- ***REQ-3:*** *Drivers receive and respond to ride requests.*

4. External Interface Requirements

4.1 User Interfaces

- *Responsive UI for mobile and web, designed using React.js + Tailwind CSS.*
- *Standard UI elements:*
  - *Navigation bar (Home, Book Ride, My Rides, Profile).*
  - *Buttons: Book Now, Cancel, Track Ride.*
  - *Error Handling: Pop-up alerts for invalid inputs.*
  - *Dark & Light Mode for accessibility.*
- *Map Integration: Google Maps for ride tracking and location selection.*

4.2 Hardware Interfaces

- *Supported Devices: Smartphones, tablets, and desktops.*
- *GPS Integration: For real-time location tracking on user and driver devices.*
- *Cloud Server Connectivity: For handling ride requests and data storage.*

4.3 Software Interfaces

- *Frontend: React.js with Tailwind CSS.*
- *Backend: Node.js with Express.js.*
- *Database: MongoDB for storing user and ride data.*
- *External APIs:*
  - *Google Maps API (Location & Navigation).*
  - *Socket.io (Real-time communication).*
- *Authentication: JWT for secure login sessions.*

4.4 Communications Interfaces

- *Web Communication: HTTPS for secure data transfer.*
- *Real-Time Updates: WebSockets (Socket.io) for ride status updates.*
- *API Protocols: RESTful APIs for client-server interactions.*
- *Security: Encrypted data transmission for user authentication and ride details.*

5. Other Non-functional Requirements

- ***5.1 Performance Requirements***

- *The system should process ride requests within 2 seconds.*
- *Real-time location tracking should update every 5 seconds.*
- *Must support at least 100 concurrent users initially, scalable as needed.*

- ***5.2 Safety Requirements***

- *User data should be encrypted to prevent unauthorized access.*
- *Driver verification required before account activation.*
- *Emergency contact feature for passengers in case of danger.*

- ***5.3 Security Requirements***

- *JWT-based authentication for users and drivers.*
- *End-to-end encryption for ride details and transactions.*
- *Role-based access control (RBAC) to protect admin functionalities.*

- ***5.4 Software Quality Attributes***

- *Reliability: 99.9% uptime with minimal downtime.*
- *Usability: Intuitive UI for both passengers and drivers.*
- *Scalability: Designed to handle growing user base efficiently.*
- *Maintainability: Modular code structure for easy updates.*
- *Interoperability: Compatible with different devices and browsers.*

6. Other Requirements

- *Database Requirements: MongoDB with optimized indexing for fast queries.*
- *Internationalization: Initially supports English; expandable to regional languages.*
- *Legal Compliance: Follows data protection laws (e.g., GDPR, IT Act of India).*
- *Reuse Objectives: Modular components for potential expansion (e.g., cab services).*

6.1 Appendix A: Glossary

- *MERN Stack: MongoDB, Express.js, React.js, Node.js.*
- *JWT: JSON Web Token for authentication.*
- *RBAC: Role-Based Access Control for managing permissions.*
- *API: Application Programming Interface for external integrations.*

6.2 Appendix B: Analysis Models

- *ER Diagram – Represents database relationships.*
- *Use Case Diagram – Shows interactions between users and the system.*
- *Data Flow Diagram (DFD) – Maps the flow of data across modules.*

6.3 Appendix C: Issues List

- *TBD: Payment gateway integration.*
- *Pending: Finalizing fare calculation model.*
- *Decision Needed: Driver verification process (manual vs. automated).*