📄Software Requirements Specification (SRS)

1. Introduction

Project Title: LocalServe – Community-Based Local Services Platform

Purpose:
To create a web-based platform that connects users with local service providers (plumbers, electricians, tutors, cleaners, etc.), enabling **service discovery, bookings, secure payments, real-time communication, and reviews**.

Scope:
- **Users (Customers):** Search and filter services, book services, make payments, chat with providers, and leave reviews.
- **Providers (Service Providers):** Register and manage services, update availability, manage bookings, receive payments, and view feedback.
- **Admin:** Manage users and providers, monitor bookings, oversee payments, and view analytics/statistics.

---

2. Functional Requirements

• **Users**
• Register/Login (JWT Authentication)
• Browse/Search/Filter services
• Book services and track booking status
• Chat with providers in real-time
• Make payments via Razorpay
• Write reviews with ratings and images

• Manage personal bookings (cancel/reschedule)

• **Providers**

• Register/Login as a provider
• Add, update, delete services
• View and manage bookings
• Chat with customers in real-time
• Track payments and earnings

• View reviews submitted on their services

• **Admins**

• Manage users and providers (verify, suspend, delete)
• Manage services and bookings

- Monitor payments and disputes
- View site statistics and analytics dashboard

---

3. Non-Functional Requirements

- **Performance:** Support at least 100+ concurrent users.
- **Security:** JWT authentication, password hashing, secure payment flow with Razorpay.
- **Scalability:** Modular REST APIs + real-time communication with Socket.io.
- **Usability:** Responsive design (mobile-friendly), intuitive dashboard for all roles.
- **Reliability:** MongoDB indexes for fast lookups, error handling for failed transactions.

---

4. System Design Overview

- **Frontend:** Vue.js + Vite + Pinia + TailwindCSS
- **Backend:** Node.js + Express.js
- **Database:** MongoDB (with Mongoose ORM)
- **Authentication:** JWT (Access + Refresh tokens)
- **Real-Time:** Socket.io (chat + notifications)
- **Payments:** Razorpay
- **Media Storage:** Cloudinary
- **Email Notifications:** Nodemailer
- **Deployment:** Vercel (Frontend), Render/Heroku (Backend)

---

5. UML Diagrams (Overview)

- **Use Case Diagram:**
- **Actors:** User, Provider, Admin

- **Use Cases:**

    - User → Search, Book, Pay, Chat, Review
    - Provider → Add/Manage services, Manage bookings, Chat, View reviews
    - Admin → Manage users, services, bookings, and view analytics

- **Class Diagram (Simplified):**

- **User(userId, name, email, password, role)**
- **Service(serviceId, providerId, name, category, price, availability, tags)**
- **Booking(bookingId, userId, serviceId, status, date, paymentId)**
- **Payment(paymentId, bookingId, amount, status, transactionDetails)**
- **Review(reviewId, userId, serviceId, rating, comment, images)**
- **Notification(notificationId, userId, title, body, type, isRead)**

• **Message(messageId, conversationId, senderId, text, attachments, readBy)**

• **Sequence Diagram (Booking Flow):**

• User searches → System shows services → User selects service
• User books → Provider gets notification → Provider accepts/rejects
• If accepted → User makes payment → Booking confirmed

• System sends notifications and updates booking status

• **ER Diagram (Entities & Relationships):**

• **Users ↔ Services** (1–M)
• **Users ↔ Bookings** (1–M)
• **Services ↔ Bookings** (1–M)
• **Bookings ↔ Payments** (1–1)
• **Services ↔ Reviews** (1–M)
• **Users ↔ Notifications** (1–M)
• **Users ↔ Messages** (M–M via Conversation)

📌**End of Document**