📘 1️⃣ SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

1. Introduction

1.1 Purpose

The purpose of this project is to develop a Local Service System that connects users with nearby service providers such as plumbers, electricians, and mechanics during emergencies.

1.2 Scope

The system allows:

- Users to raise emergency service requests.
- Providers to view and accept requests.
- Providers to complete assigned jobs.
- Users to track request status.

---

2. Overall Description

2.1 Product Perspective

This is a web-based application developed using:

- Frontend: HTML, CSS, JavaScript
- Backend: Python (Flask)
- Database: MySQL

2.2 Product Functions

- User Registration (optional future scope)
- Raise Emergency Request
- Provider Dashboard
- Accept Request
- Complete Request
- Track Request Status
- Provider Availability Management

---

3. Functional Requirements

FR1 – Raise Emergency

User shall be able to submit:

- Name
- Phone
- Service Type
- Description

FR2 – View Emergency Requests

Provider shall be able to:

- View pending requests by service type

FR3 – Accept Request

Provider shall be able to:

- Accept a request
- System shall change status to ASSIGNED
- Provider availability becomes BUSY

FR4 – Complete Request

Provider shall:

- Mark job as completed
- System changes status to COMPLETED
- Provider becomes AVAILABLE

FR5 – Track Request

User shall:

- Enter phone number
- View latest request status

---

4. Non-Functional Requirements

- System shall be responsive.
- System shall support multiple service types.
- System shall ensure data integrity.
- System shall provide smooth UI interaction.
- System shall update status in real-time.

---

2 SYSTEM DESIGN PHASE

1. High Level Design (HLD)

Architecture Type: Client-Server Architecture

Components:

- Frontend (User Interface)
- Backend (Flask API Server)
- Database (MySQL)

Flow:

User → Frontend → Backend API → Database → Response → Frontend

---

2. Low Level Design (LLD)

Modules:

1. User Module
   - Raise Emergency
   - Track Status
2. Provider Module
   - View Requests
   - Accept Request
   - Complete Request
3. Database Module
   - Store service requests
   - Store provider details
   - Manage status updates

3 SYSTEM ARCHITECTURE

Architecture Model: 3-Tier Architecture

1. Presentation Layer (HTML/CSS/JS)
2. Business Logic Layer (Flask Backend)
3. Data Layer (MySQL Database)

4 SYSTEM ARCHITECTURE

Architecture Model: 3-Tier Architecture

1. Presentation Layer (HTML/CSS/JS)
2. Business Logic Layer (Flask Backend)
3. Data Layer (MySQL Database)

Field	Type	Description
id	INT (PK)	Unique ID
user_name	VARCHAR	User name
user_phone	VARCHAR	User phone
service_type	VARCHAR	Type of service
description	TEXT	Problem details
status	VARCHAR	PENDING / ASSIGNED / COMPLETED
assigned_provider	INT	Provider ID

Table 2: providers

Field	Type	Description
id	INT (PK)	Provider ID
name	VARCHAR	Provider name
service_type	VARCHAR	Service category
availability	VARCHAR	AVAILABLE / BUSY

📊 1️⃣ DATA FLOW DIAGRAM (DFD)

🔹 DFD Level 0 (Context Diagram)

Description:

In Level 0 DFD, the entire system is represented as a single process interacting with external entities.

External Entities:

- User
- Provider
- Database

Process:

- Local Service System

Data Flow:

- User → Emergency Request → System
- System → Status Update → User
- Provider → Accept/Complete → System
- System ↔ Database

✍️ Journal Writing Format:

The Level 0 DFD represents the Local Service System as a single process. The user sends emergency requests to the system. The provider receives and updates request status. The system stores and retrieves data from the database.

🔹 DFD Level 1

Now system is divided into processes:

Processes:

1. Raise Emergency
2. Assign Provider

3. Complete Service
4. Track Request

Data Store:

- Service Requests Table
- Providers Table

Flow Explanation:

User → Raise Emergency → Service Requests DB
Provider → View Requests → Assign Provider
Provider → Complete Job → Update Status
User → Track Request → View Status

👥 2️⃣ USE CASE DIAGRAM CONTENT

Actors:

1. User
2. Provider
3. Admin (optional future scope)

Use Cases – User

- Raise Emergency
- Track Request

Use Cases – Provider

- View Requests
- Accept Request
- Complete Request

Journal Format:

The Use Case Diagram identifies two main actors: User and Provider.
The user can raise emergency requests and track service status.
The provider can view available requests, accept them, and mark them as completed.

🗂️ 3️⃣ ER DIAGRAM EXPLANATION

Entities:

1. Service_Request
2. Provider

Entity 1: Service_Request

Attributes:

- id (Primary Key)
- user_name
- user_phone
- service_type
- description
- status
- assigned_provider (Foreign Key)

Entity 2: Provider

Attributes:

- id (Primary Key)
- name
- service_type
- availability

Relationship:

One Provider can handle many Service Requests.
One Service Request is assigned to only one Provider.

Relationship Type:
One-to-Many (Provider → Service_Request)

Journal Writing:

The ER Diagram consists of two entities: Service_Request and Provider.
A one-to-many relationship exists between Provider and Service_Request, where one provider can handle multiple service requests.

🔄 4️⃣ SEQUENCE DIAGRAM EXPLANATION

Case 1 – Raise Emergency

User → Frontend
Frontend → Backend API
Backend → Database
Database → Backend
Backend → Frontend
Frontend → User

Case 2 – Accept Request

Provider → Frontend
Frontend → Backend
Backend → Update DB
Backend → Frontend
Frontend → Provider

Journal Format:

The sequence diagram shows the interaction between User, Frontend, Backend, and Database during emergency request submission and provider assignment.

🔁 5️⃣ ACTIVITY DIAGRAM EXPLANATION

Activity Flow – User

Start
↓
Open Website
↓
Raise Emergency
↓
Submit Form
↓
System Saves Request
↓
Track Status
↓
End

Activity Flow – Provider

Start
↓
Login / Open Dashboard
↓
View Requests
↓
Accept Request
↓
Complete Service
↓
System Updates Status
↓
End