# Software Requirements Specification (SRS)

The purpose of this SRS document is to define the functional and non-functional requirements of a charity management platform that facilitates the donation and distribution of medicines and essential supplies to underprivileged individuals, especially children. The system enables donors to contribute items, field workers to log distribution, and administrators to manage data, generate reports, and monitor impact.

## Overview

This SRS includes the following:

- Functional requirements outlining system behaviour
- Non-functional requirements addressing security, performance, and scalability
- External and internal interfaces
- Data requirements and system constraints

## Overall Description

### Product Perspective

The system is a standalone-distributed solution integrating:

- Frontend: Next.js Web App and Flutter-based Mobile App
- Backend: RESTful APIs via Spring Boot
- Database: PostgreSQL
- Deployment: Render/Netlify/Firebase

### Product Functions

- **User Authentication:** Secure login using JWT tokens
- **Donor and Life Member Management:** Add/edit donors, filter by parish

- **Client Management:** Add clients, review status (pending/approved)
- **Donation Recording:** Link donations to donors/life members with payment details
- **Activity Logging:** Track field worker visits, equipment distribution, training, etc.
- **Voucher System:** Create and approve payment vouchers
- **Analytics Dashboard:** Real-time visual reports on donations, clients, and activities
- **Prescription Scanning (optional):** OCR-based image-to-text extraction

## User Classes and Characteristics

| **User Role** | **Characteristics** |
|---|---|
| **Admin** | Full access to all modules |
| **Treasurer** | Manages donations and vouchers |
| **Secretary** | Manages life members |
| **Community Worker (CW)** | Field data collection, activity logging |
| **Viewer** | Read-only access to dashboard |
| **Manager** | Reviews/approves client submissions |

### Operating Environment

- Web: Latest Chrome/Firefox, 64-bit OS
- Mobile: Android (8.0+), iOS (13+)
- Backend: Spring Boot 3.x, PostgreSQL 14+
- Hosting: Render (Backend), Netlify/Vercel (Frontend)

**Constraints**

- Only authorized users may access the system.
- Data must comply with privacy standards.
- Mobile app must support low-bandwidth regions (offline-first design).
- OCR accuracy depends on image clarity and lighting.

## Specific Requirements

### Functional Requirements

#### User Authentication and Authorization

- FR1.1: Users must log in using email and password.
- FR1.2: System must issue and validate JWT tokens.
- FR1.3: Role-based access must be enforced across modules.

#### Life Member and Donor Management

- FR2.1: Admins can add/edit/delete life members.
- FR2.2: Secretary can search and filter life members by name, status, or parish.

#### Client Management

- FR3.1: CWs can submit new client entries with full details.
- FR3.2: Managers/Admins can view, approve, or reject client requests.
- FR3.3: Clients must have statuses: *pending*, *approved*, *rejected*.

#### Donation Recording

- FR4.1: Treasurer can record donation details: amount, donor, payment mode.
- FR4.2: System must allow search and export of donation history.

**Activities Logging**

- FR5.1: CWs can submit field activities with client ID, activity type, date, and remarks.
- FR5.2: CWs can view/edit their submitted activities.
- FR5.3: Activity types include *Field Visit*, *Education*, *Training*, *Financial Help*, *and Equipment*.

**Analytics Dashboard**

- FR7.1: Admin can view donation trends (bar chart), client approvals (pie chart), activity logs (line chart).
- FR7.2: Viewer has read-only access to dashboard.

**OCR and Prescription Scanning (Optional)**

- FR8.1: CW can capture/upload prescription image.
- FR8.2: App extracts medicine name, dosage, and quantity using OCR.
- FR8.3: CW can review extracted data before final submission.

**External Interface Requirements**

**User Interfaces**

- Web UI (React.js): Form-based CRUD, search filters, toast alerts, role-based menus
- Mobile UI (Flutter): Tabbed navigation, dropdown selectors, dark/light theme
- Responsive layouts, minimal latency, and offline caching for mobile

**Hardware Interfaces**

- Mobile: Access to camera, storage
- Web: Desktop or laptop with modern browser

**Software Interfaces**

- API: REST endpoints via Spring Boot backend
- Mobile HTTP Client: Dio package

- OCR Library: Google ML Kit
- Auth: JWT-based token headers (Authorization: Bearer <token>)

## Data Requirements

### Database Schema Highlights (PostgreSQL)

- **Users**: id, name, email, password, role
- **LifeMembers**: id, name, contact, parish, status
- **Donors**: id, name, linked_lm_id, is_life_member
- **Clients**: id, name, illness, contact, status, submitted_by
- **Activities**: id, client_id, type, remarks, submitted_by
- **Vouchers**: id, amount, description, status, approved_by
- **Prescriptions**: id, client_id, image_url, medicines (linked table)

### Data Validation Rules

- Required fields must not be null.
- Status must follow predefined enums.
- Phone numbers must be 10 digits.
- Uploaded image size < 5MB (JPEG/PNG only).

## Non-Functional Requirements

### Performance

- Mobile sync response time < 5 sec under 4G
- Web dashboard load time < 3 sec
- Backend handles 100+ concurrent API requests

**Security**

- Passwords stored using bcrypt hashing
- All API calls over HTTPS
- Role-based filtering on every protected endpoint
- Uploaded images stored in secured cloud storage (e.g., Firebase/Cloudinary)

**Reliability & Availability**

- Uptime > 99.5% for backend services
- Retry mechanism on mobile API failures
- Data consistency through atomic database operations

**Usability**

- Simple, clean UI for both web and mobile
- Form validations with proper error messages
- Language toggles and accessibility options in future scope

**Maintainability**

- Modular frontend and backend with folder-based architecture
- Clear version control (Git) with documentation and changelogs

**Scalability**

- Can onboard additional NGOs or organizations in future
- Supports migration to NoSQL or shared SQL for large-scale use
- Stateless backend suitable for horizontal scaling

**Assumptions and Dependencies**

- Internet connectivity is required for real-time sync
- OCR accuracy depends on image clarity and model performance
- Users are pre-verified before account creation
- Backend must be deployed prior to frontend or app testing