

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

SYSTEM NAME: DF REACH SYSTEM



July 23, 2025

Deshpande Foundation

Contents

[1. Introduction 1](#_Toc204185095)

[2. User Roles and Pages 1](#_Toc204185096)

[2.1. User Roles 1](#_Toc204185097)

[2.2. Application Pages Overview 1](#_Toc204185098)

[3. Presentation Layer (Frontend) 2](#_Toc204185099)

[4. Application Layer (Backend) 3](#_Toc204185100)

[5. Data Layer (Database) 4](#_Toc204185101)

[5.1. Database Overview 4](#_Toc204185102)

[5.2. Core Tables by Function 4](#_Toc204185103)

[5.3. Table Descriptions 5](#_Toc204185104)

[6. Data Flow 7](#_Toc204185105)

[7. Software Requirements 8](#_Toc204185106)

## 1. Introduction

This document outlines the architecture and data flow for the DF Reach application, which is designed with a clear separation of concerns across three main layers: **Presentation**, **Application**, and **Data**. It provides a comprehensive overview of how various system components interact to deliver a robust and scalable solution for beneficiary management, program enrollment, and administrative oversight.

## 2. User Roles and Pages

The application is designed to cater to different user roles, each with specific functionalities and access levels to various pages:

### 2.1. User Roles

* **Field Facilitator:**
  + **Responsibilities:** Primarily responsible for entering and managing beneficiary details. They enroll beneficiaries in different programs and sub-programs.
  + **Accessible Pages:** Dashboard, Beneficiary, Programs, Contact Us.
* **Manager:**
  + **Responsibilities:** Oversees the activities of field facilitators and can view analytics related to beneficiary enrollment and program performance.
  + **Accessible Pages:** Dashboard, Programs, Analytics, Contact Us.
* **Admin:**
  + **Responsibilities:** Has full access to the system, including configuration settings, user management, and comprehensive analytics.
  + **Accessible Pages:** Dashboard, Programs, Analytics, Contact Us, Config Page.

### 2.2. Application Pages Overview

* **Dashboard:** A central overview page providing relevant summaries and quick actions based on the user's role.
* **Beneficiary:** A dedicated page for Field Facilitators to input and manage beneficiary information, including their enrollment in programs and sub-programs.
* **Programs:** This page lists all available programs with short descriptions. Users can click on a program to view its associated sub-programs and their detailed descriptions.
* **Analytics:** Provides managers and admins with insights into program performance, beneficiary enrollment trends, and field facilitator activities.
* **Contact Us:** An informational page providing contact details for support or inquiries.
* **Config Page:** An administrative page accessible only to Admins, allowing for system configuration, user role management, and master data setup (e.g., adding/editing programs and sub-programs).

## 3. Presentation Layer (Frontend)

* **Technology Stack:** Developed using **ReactJS**.
* **Purpose:** This layer is responsible for the entire user interface (UI), handling user interactions, and displaying data in a user-friendly manner. It is the client-side component of the application.
* **Key Responsibilities:**
  + **User Interface (UI):** Renders all visual elements, forms, dashboards, and interactive components.
  + **User Interaction:** Captures user input, clicks, and gestures.
  + **Data Display:** Presents information retrieved from the Application Layer to the user.
  + **Communication:** Communicates with the Application Layer via **RESTful API** calls using **Axios**. Axios is used for making asynchronous HTTP requests from the browser to the backend.
  + **Client-Side Routing:** Handles client-side routing within the single-page application (SPA) without full page reloads, providing a seamless user experience.
  + **Form Validations:** Performs initial client-side validation of user input, providing immediate feedback to the user and reducing unnecessary backend requests.
  + **State Management:** Manages the application's UI state, ensuring data consistency and responsiveness.
  + **User Feedback:** Utilizes libraries like **SweetAlert2** for enhanced user notifications (e.g., success messages, error alerts).

## 4. Application Layer (Backend)

* **Technology Stack:** Developed using **Node.js** with **Express.js**.
* **Purpose:** This layer serves as the core business logic processor. It acts as an intermediary between the Presentation Layer and the Data Layer, enforcing business rules and managing application workflows.
* **Key Responsibilities:**
  + **API Endpoint Management:** Receives and processes HTTP requests (GET, POST, PUT, DELETE) from the Presentation Layer via **Express.js** routes.
  + **Core Business Logic:** Contains all the core business rules, calculations, and complex workflow management for processes such as Reimbursement, Procurement, and Advance.
  + **Authentication:** Handles user authentication using **JSON Web Tokens (JWT)**. It validates tokens received from the frontend to ensure secure access to protected resources.
  + **Authorization:** Implements access control mechanisms to ensure users can only perform actions and access data for which they have explicit permissions, based on their assigned roles.
  + **Data Validation:** Performs comprehensive server-side validation of incoming data using libraries like **express-validator**, ensuring data integrity and security before any database interaction.
  + **File Uploads:** Manages file uploads from the frontend using **Multer**, securely processing and storing files as required by the application.
  + **Database Interaction:** Interacts directly with the Data Layer (MySQL Database) to store, retrieve, update, and delete data based on business logic and user requests.
  + **External Service Integration:**
    - **Email Notifications:** Integrates with **Nodemailer** to send automated transactional emails (e.g., confirmation, status updates, password resets).
    - **Document Generation:** Responsible for generating various documents such as Purchase Orders (PO), Work Orders (WO), or other official reports.
    - **Financial Integrations:** Potentially integrates with external financial systems like Tally or directly with bank APIs for payment processing, reconciliation, or data synchronization.

## 5. Data Layer (Database)

* **Technology Stack:** Utilizes **MySQL** as the relational database management system.
* **Purpose:** This layer is responsible for the persistent storage and efficient retrieval of all application data.
* **Key Responsibilities:**
  + **Data Storage:** Stores all structured data, including but not limited to:
    - User and Role Management: employees, roles, emp\_roles, employee\_history
    - Beneficiary Information: master\_beneficiary, gender, master\_education\_details, master\_education\_levels, income, earning\_range, master\_occupation, master\_certification
    - Program and Enrollment Data: programs, sub\_programs, beneficiary\_programs, master\_outreach\_channels
    - Geographical Data: state, district, taluka, village, address
    - System Configurations: master\_status, menus, role\_actions, role\_menu
  + **Data Integrity:** Ensures data consistency, accuracy, and reliability through the implementation of relational constraints (e.g., foreign keys, primary keys), indexes for optimized queries, and transaction management.
  + **Data Retrieval:** Provides efficient mechanisms for the Application Layer to query and retrieve data, supporting complex joins and filtering.
  + **Data Persistence:** Ensures that all application data is stored reliably and can be recovered in case of system failures.
  + **Access:** The MySQL database is accessed exclusively by the Application Layer. The Presentation Layer does not have direct access to the database, ensuring a secure and controlled data flow.

### 5.1. Database Overview

The database schema is designed to support the core functionalities of the DF Reach application, including beneficiary management, program enrollment, employee roles, and geographical data. It follows a relational model to ensure data integrity and efficient querying.

### 5.2. Core Tables by Function

* **User & Role Management:**
  + employees: Stores details of all employees, including field facilitators and managers.
  + roles: Defines different roles within the system (e.g., Field Facilitator, Manager, Admin).
  + emp\_roles: Maps employees to their respective roles.
  + employee\_history: Logs historical changes to employee records.
* **Beneficiary & Related Details:**
  + master\_beneficiary: Core table for individual beneficiary details.
  + gender: Master data for gender.
  + master\_education\_details: Stores educational qualifications of beneficiaries.
  + master\_education\_levels: Master data for education levels.
  + income: Stores income-related details for beneficiaries.
  + earning\_range: Master data for income ranges.
  + master\_occupation: Master data for occupations.
  + master\_certification: Stores certification details for beneficiaries.
* **Program & Enrollment:**
  + programs: Defines the main programs offered.
  + sub\_programs: Defines sub-programs associated with main programs.
  + beneficiary\_programs: Links beneficiaries to the programs and sub-programs they are enrolled in.
  + master\_outreach\_channels: Master data for how beneficiaries are reached.
* **Geographical Data:**
  + state: Master data for states.
  + district: Master data for districts, linked to states.
  + taluka: Master data for talukas (sub-districts), linked to districts.
  + village: Master data for villages, linked to talukas.
  + address: Stores detailed address information, linked to geographical entities.
* **System Configurations & Utilities:**
  + master\_status: Generic master data for status (e.g., active, inactive, enrolled).
  + menus: Defines the application's menu structure.
  + role\_actions: Maps roles to specific actions they can perform.
  + role\_menu: Maps roles to accessible menu items.

### 5.3. Table Descriptions

* address: Stores detailed address information including state, district, taluka, village, and pincode.
* beneficiary\_programs: Records the enrollment of a beneficiary in a specific program and sub-program, along with the enrolling associate and outreach channel.
* master\_outreach\_channels: Lists various channels through which beneficiaries are reached.
* district: Contains district names and their associated states.
* emp\_roles: Links employees to their assigned roles.
* employee\_history: Maintains a historical record of changes to employee information.
* employees: Stores core employee details such as name, email, mobile, job title, and manager.
* gender: Defines gender types.
* income: Details about beneficiary's annual income, earning members, and source of income.
* earning\_range: Defines predefined ranges for annual income.
* master\_beneficiary: Comprehensive details about each beneficiary, including personal, address, education, and income information.
* master\_education\_details: Captures specific educational details like institute, field of study, and passing year.
* master\_education\_levels: Lists various educational qualification levels.
* master\_certification: Stores information about any certifications obtained by beneficiaries.
* master\_status: A generic table for defining various status types used across the application.
* menus: Defines the navigation menu items and their hierarchy.
* master\_occupation: Lists different types of occupations.
* programs: Stores details about the main programs offered by the organization.
* role\_actions: Specifies which actions are permitted for each role.
* role\_menu: Controls which menu items are visible to each role.
* roles: Defines the different user roles in the system.
* state: Contains state names.
* sub\_programs: Stores details about sub-programs, linked to their parent programs.
* taluka: Contains taluka names, linked to districts.
* village: Contains village names and pincodes, linked to talukas.

## 6. Data Flow

The following steps illustrate the typical data flow within the application, from user interaction to database persistence and back:

1. **User Interaction:** A user interacts with the ReactJS Frontend. For example, a field facilitator fills out a form to submit a new reimbursement request.
2. **API Request:** The Frontend sends an asynchronous REST API request (e.g., POST /api/reimbursements) to the Node.js Backend using Axios. This request includes the reimbursement data (e.g., amount, description, attached documents) and typically a JWT in the authorization header for authentication.
3. **Backend Processing:**
   * The Node.js Backend receives the request via Express.js.
   * It first authenticates the user by validating the provided JWT. If the token is invalid or missing, the request is rejected with an authentication error.
   * It then validates the incoming data using express-validator to ensure all required fields are present, data types are correct, and values adhere to business rules.
   * If the request involves file uploads (e.g., receipts for reimbursement), Multer handles the file processing, saving them to a designated storage location.
   * The backend's business logic processes the request. This may involve:
     + Retrieving additional user or workflow-related data from the MySQL Database.
     + Storing or updating the reimbursement request details in the relevant tables (e.g., reimbursement\_requests, documents) within the MySQL Database.
     + Triggering an email notification to the user, their manager, or an accounting department via Nodemailer to inform them of the new request or its status change.
     + Potentially generating a unique ID or reference number for the request.
4. **Database Interaction:** The Backend performs CRUD (Create, Read, Update, Delete) operations on the MySQL Database to persist the new reimbursement request, update its status, or retrieve information for processing.
5. **API Response:** The Backend sends a JSON response back to the ReactJS Frontend. This response indicates the success or failure of the operation (e.g., HTTP status code 200 for success, 400/500 for errors) and may include any relevant data (e.g., the ID of the newly created reimbursement request, a success message).
6. **Frontend Update:** The ReactJS Frontend receives the JSON response.
   * If the response indicates success, the Frontend updates the UI accordingly (e.g., shows a success message using SweetAlert2, clears the form, or updates a list of pending reimbursement requests).
   * If the response indicates an error, the Frontend displays an appropriate error message to the user using SweetAlert2, guiding them on how to resolve the issue.

This layered architecture ensures modularity, scalability, and maintainability, allowing independent development and deployment of each component while providing a robust and secure application environment.

## 7. Software Requirements

This section lists the key tools and technologies required for the development and operation of the application.

|  |  |
| --- | --- |
| **Tool/Technology** | **Purpose** |
| **VS Code** | Primary Integrated Development Environment (IDE) for coding. |
| **Multer** | Middleware for handling multipart/form-data, primarily used for file uploads in the backend. |
| **REST API** | Architectural style for communication between the frontend and backend, enabling stateless, client-server interactions. |
| **Postman** | Tool for testing and validating RESTful APIs, allowing developers to send requests and inspect responses. |
| **Kelvin CSS** | Frontend styling framework (Note: Assuming this is a custom or specific CSS framework, as "Kelvin CSS" is not a widely known standard framework like Tailwind CSS or Bootstrap). |
| **JWT (JSON Web Token)** | Standard for securely transmitting information between parties as a JSON object, used for authentication and authorization. |
| **Axios** | Promise-based HTTP client for the browser and Node.js, used by the frontend for making API calls. |
| **SweetAlert2** | A beautiful, responsive, customizable, accessible (WAI-ARIA) replacement for JavaScript's popup boxes, used for displaying alert messages. |
| **Nodemailer** | Module for Node.js applications to allow easy email sending from the backend. |
| **Express.js** | Fast, unopinionated, minimalist web framework for Node.js, used for building the backend routing and API endpoints. |
| **express-validator** | Middleware for Express.js that wraps validator.js, used for backend data validation. |
| **Bcrypt** | Library for hashing passwords, ensuring secure storage of user credentials by encrypting them. |