Software Requirements Specification (SRS)

System Name: DF Ticketing System

Company: DESHPADE FOUNDATION

Version: 1.0

Date: 19-06-2025

Contents

[1. Introduction 1](#_Toc202522770)

[1.1 Purpose 1](#_Toc202522771)

[1.2 Intended Audience 1](#_Toc202522772)

[1.3 Intended Use 1](#_Toc202522773)

[1.4 Product Scope 1](#_Toc202522774)

[1.5 Definitions and Acronyms 2](#_Toc202522775)

[2. Overall Description 2](#_Toc202522776)

[2.1 User Needs 2](#_Toc202522777)

[2.2 Assumptions and Dependencies 2](#_Toc202522778)

[3. System Features and Requirements 3](#_Toc202522779)

[3.1 Functional Requirements 3](#_Toc202522780)

[3.2 External Interface Requirements 4](#_Toc202522781)

[3.3 System Features 4](#_Toc202522782)

[3.4 Use Cases 4](#_Toc202522783)

[4. Module Breakdown by Role 6](#_Toc202522784)

[Home Module 6](#_Toc202522785)

[Report Module 6](#_Toc202522786)

[5. Non-Functional Requirements 6](#_Toc202522787)

[6. Deployment Architecture 7](#_Toc202522788)

[6.1 Logical Architecture 7](#_Toc202522789)

[6.2 Data Flow 8](#_Toc202522790)

[7. Database Overview 8](#_Toc202522791)

[7.1 Core Tables by Function 8](#_Toc202522792)

[7.2 Table Descriptions 9](#_Toc202522793)

[8. Software Requirements 13](#_Toc202522794)

# 1. Introduction

### 1.1 Purpose

This document defines the functional and non-functional requirements for the DF\_Ticketing System. It supports workflows for Reimbursement, Procurement, and Advance ticketing, enabling employees to raise requests, managers and finance teams to approve them, and finance to process payments with tracking and audit.

### 1.2 Intended Audience

* Employees of:
  + Deshpande Foundation
  + Deshpande Educational Trust
  + Foundation for Sandboxstartup Initiatives
  + USF Members
* Development, QA, and Operations teams
* HR and Finance departments
* System administrators

### 1.3 Intended Use

The system allows:

* Raising and tracking tickets
* Routing for approval
* Managing documents like PR, PO, WO, UTR, and bills
* Exporting data for Tally and bank formats
* Generating PO (items) or WO (services)

### 1.4 Product Scope

* Web-based system
* Hosted on AWS EC2
* Built with ReactJS (frontend) and NodeJS (backend)
* Database: MySQL (hosted on EC2 RDP)
* Tally and bank integrations
* Role-based modules and actions

### 1.5 Definitions and Acronyms

|  |  |
| --- | --- |
| **Term** | **Description** |
| PR | Purchase Request |
| PO | Purchase Order (for items) |
| WO | Work Order (for services) |
| UTR | Unique Transaction Reference |
| Tally | Accounting Software |
| RDP | Remote Desktop Protocol |

# 2. Overall Description

### 2.1 User Needs

* Create and manage financial requests
* View ticket status and history
* Upload and view documents
* The finance team can process, approve, and track payments

### 2.2 Assumptions and Dependencies

* Application hosted on AWS
* Authenticated access
* Data stored in MySQL
* Integration with Tally and bank formats
* Vehicles, travel, food, and accommodation apply to both procurement and reimbursement

# 3. System Features and Requirements

### 3.1 Functional Requirements

#### Reimbursement Workflow

* User uploads bill and creates ticket
* Manager reviews and approves/rejects
* Finance Processor will verify and processed, then it will go to Finance Approver, then after approving it will come back to Finance Processor for Payments and UTR upload will happen and ticket close

#### Procurement Workflow

* **< ₹20,000:**
  + User raises request → approval → payment → closure
* **≥ ₹20,000 (Items):**
  + The Ops team raises a ticket
  + Generate PR → add vendors → collect quotations
  + Select vendor → generate PO
  + Payment and invoice upload → UTR upload → ticket closure
* **≥ ₹20,000 (Services):**
  + As above, but generate WO instead of PO

#### Advance Workflow

* User requests an Advance
* Paid → bills uploaded later
* Reconciliation → balance settled → closure

#### Finance Panel

* List of pending payments
* Upload UTRs
* Download Excel for Tally and Bank

#### Document Management

* Uploads: PR, PO, WO, invoices, bills, quotations, UTRs

### 3.2 External Interface Requirements

* Frontend: ReactJS
* Backend: NodeJS
* Database: MySQL on AWS EC2 RDP
* Notifications: SMTP or AWS SES
* Downloads: Excel formats for Tally, bank

### 3.3 System Features

* Role-based access
* Ticket lifecycle with audit trail
* Document uploads and tracking
* UTR upload for payment closure
* Tally and bank export options
* Dashboard with recent reports
* Filtering and search options

### 3.4 Use Cases

This section describes typical interactions between users and the DF\_Ticketing System.

#### Use Case 3.4.1: Employee Submits Reimbursement Request

* **USER:** Normal User (Employee)
* **Precondition:** The Employee is logged into the system.
* **Description:**
  1. Employee navigates to the "Report" module.
  2. Employee clicks "+ New Report" and selects "Reimbursement".
  3. Employee fills in necessary details (e.g., expense type, amount, description).
  4. Employee uploads relevant bills/invoices.
  5. Employee saves the reimbursement request ticket.
* **Postcondition:** A new reimbursement ticket is created with "Initialized" status and once the user submit the report status will be "Manager Pending" and is visible in the employee's "Report" list.

#### Use Case 3.4.2: Manager Approves a Ticket

* **USER:** Manager
* **Precondition:** Manager is logged into the system and has pending tickets for approval in "MY TEAM".
* **Description:**
  1. The manager navigates to the "MY TEAM" module.
  2. The manager views the list of tickets awaiting their approval.
  3. The manager selects a specific ticket to review its details and attached documents.
  4. The manager decides to "Approve" or "Reject" the ticket.
  5. If approved, the manager may add comments.
  6. If rejected, the manager must provide a reason.
* **Postcondition:** The ticket status is updated to "Manager Approved " (if approved) or "Manager Rejected " (if rejected). The ticket moves to the next stage (Finance Processor) or back to the employee.

#### Use Case 3.4.3: Finance Processes Payment and Uploads UTR

* **USER:** Finance
* **Precondition:** Finance is logged into the system and has tickets pending payment processing in "FINANCE APPROVAL" or "FINANCE PANEL".
* **Description:**
  1. Finance navigates to "FINANCE APPROVAL" or "FINANCE PANEL".
  2. Finance reviews tickets requiring payment processing.
  3. Finance verifies the details, including the approved amount and attached documents.
  4. Finance processes the payment through the bank or the relevant system.
  5. Finance uploads the Unique Transaction Reference (UTR) for the payment.
  6. Finance marks the ticket as "Closed".
* **Postcondition:** The ticket status is updated to "Closed", and the UTR is recorded, providing an audit trail.

# 4. Module Breakdown by Role

|  |  |
| --- | --- |
| **Role** | **Modules** |
| Normal User | HOME, REPORT, CONTACT US |
| Manager | HOME, REPORT, MY TEAM, CONTACT US |
| Finance | HOME, REPORT, FINANCE APPROVAL, FINANCE PANEL, SUMMARY, CONTACT US |
| HR | HOME, REPORT, HRPORTAL, CONTACT US |
| Admin | HOME, REPORT, HRPORTAL, MY TEAM, FINANCE APPROVAL, FINANCE PANEL, SUMMARY, CONFIG, CONTACT US |

### Home Module

* Top summary of report and ticket types with counts
* The latest 5 tickets are displayed
* "+ New Report" button in Report tab

### Report Module

* Full list of reports
* Filter by status, type, or date
* Document upload and status tracking

# 5. Non-Functional Requirements

* Supports ≥ 500 concurrent users
* Role-based access with data-level security
* SSL encryption
* 99.5% uptime target
* Easily maintainable frontend/backend
* Compatible with major browsers
* Mobile responsive (optional)

# 6. Deployment Architecture

|  |  |  |
| --- | --- | --- |
| **Component** | **Technology** | **Hosting** |
| Frontend | ReactJS | AWS EC2 |
| Backend | NodeJS | AWS EC2 |
| Database | MySQL | EC2 RDP |
| Email Service | GOOGLE SMTP | External |
| File Storage | EC2 / AWS S3 | Optional |

### 6.1 Logical Architecture

The DF\_Ticketing System follows a three-tier logical architecture:

* **Presentation Layer (Frontend):**
  + Developed using **ReactJS**.
  + Responsible for the user interface, user interaction, and displaying data.
  + Communicates with the Application Layer via RESTful API calls using Axios.
  + Handles client-side routing, form validations, and state management.
* **Application Layer (Backend):**
  + Developed using **NodeJS** with **Express.js**.
  + Acts as the core business logic processor.
  + Receives requests from the Presentation Layer, processes them, and interacts with the Data Layer.
  + Handles authentication (JWT), authorisation, data validation (express-validator), and file uploads (Multer).
  + Manages workflows (Reimbursement, Procurement, Advance) and business rules.
  + Interacts with external services like email (Nodemailer) and potentially Tally/bank integrations.
* **Data Layer (Database):**
  + Utilises **MySQL** as the relational database.
  + Stores all persistent data related to users, roles, tickets, workflows, payments, documents, and system configurations.
  + Accessed by the Application Layer to store and retrieve data.

### 6.2 Data Flow

1. **User Interaction:** A user interacts with the **ReactJS Frontend** (e.g., submitting a new reimbursement request).
2. **API Request:** The Frontend sends an asynchronous **REST API** request (via Axios) to the **NodeJS Backend**. This request often includes a JWT for authentication.
3. **Backend Processing:**
   * The Backend receives the request via **Express.js**.
   * It authenticates the user using **JWT**.
   * It validates the incoming data using **express-validator**.
   * If it's a file upload, **Multer** handles the file processing.
   * The backend business logic processes the request, which may involve:
     + Retrieving data from the **MySQL Database**.
     + Storing or updating data in the **MySQL Database**.
     + Triggering an email notification via **Nodemailer** (if applicable).
     + Generating documents (e.g., PO/WO).
4. **Database Interaction:** The Backend performs CRUD (Create, Read, Update, Delete) operations on the **MySQL Database**.
5. **API Response:** The Backend sends a JSON response back to the **ReactJS Frontend**, indicating success or failure and any relevant data.
6. **Frontend Update:** The Frontend receives the response and updates the UI accordingly.

# 7. Database Overview

### 7.1 Core Tables by Function

* **Users & Roles**: users, employees, user\_roles, roles, login\_details, employee\_history, user\_history
* **Tickets & Workflow**: tickets, reports, ticket\_logs, ticket\_history, report\_logs, report\_history, re\_ticket\_details, reimb\_others
* **Finance**: payments, payment\_type, payment\_mode, payment\_route, organization\_bank, user\_bank, tally\_pay\_bank, tally\_payment, tally\_booking
* **Shared**: expense, expense\_category, bills, vehicles, travels, accommodation, food, vendors
* **System Configuration**: departments, sub\_departments, cost\_center, entities, academic\_cost\_center, academic\_entity, academic\_organization, academic\_year, budget\_type, categories, contribution\_type, levels, menus, role\_menu, role\_actions, master\_status

### 7.2 Table Descriptions

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| academic\_cost\_center | Links cost centres to academic years and entities, managing their status within specific academic contexts. |
| academic\_entity | Maps entities to academic years, indicating their active status for particular academic periods. |
| academic\_expense\_category | Associate expense categories with academic years, useful for tracking expenses relevant to specific academic cycles. |
| academic\_organization | Connects organisations to academic years, managing their status for defined academic periods. |
| academic\_year | Defines academic years with start/end dates and status, used for various academic-related configurations. |
| accommodation | Stores details of accommodation expenses, linked to reimbursement tickets. |
| adv\_ticket\_details | Holds specific details for advance payment tickets, including amount and description. |
| bills | Stores information about bills and invoices uploaded, linked to tickets and expense categories. |
| budget\_type | Defines different types of budgets (e.g., Revenue, Capex) with their names and status. |
| categories | Classifies different expense categories like Food, Travel, Accommodation, and Others. |
| contribution\_type | Categorises types of financial contributions (e.g., FC, NFC). |
| cost\_center | Defines cost centres within the organisation for financial tracking. |
| departments | Stores information about various departments within the organisation. |
| employee\_history | Maintains a historical record of changes to employee details. |
| employees | Contains detailed personal and professional information for all employees. |
| entities | Represents different legal entities or branches within the organisation. |
| expense | Defines broad expense types, such as Program Expense or General & Administrative Expense. |
| expense\_category | Categorises the type of expense (e.g., Reimbursement, Procurement, Advance). |
| food | Stores details of food-related expenses, linked to reimbursement tickets. |
| ledgers | Manages different ledger accounts with their codes, names, types, and associated categories. |
| levels | Defines the different approval levels and stages within the ticketing workflow. |
| login\_details | Log user login and logout times, providing an audit trail of system access. |
| master\_status | A master table defining various status codes used throughout the system (e.g., Active, Pending, Approved, Rejected). |
| menus | Defines the navigation menu items and their properties within the application. |
| organization | Stores core information about the main organisation. |
| organization\_bank | Contains bank account details for the organisation and its entities. |
| payment\_mode | Lists available payment modes (e.g., Online, Cash). |
| payment\_route | Defines the possible routes for payments (e.g., Organisation to User, Organisation to Vendor). |
| payment\_type | Specifies types of electronic payments (e.g., NEFT, IMPS, RTGS, IFT). |
| payments | Records details of all financial payments made, linked to tickets and payment modes. |
| pf\_details | Store Provident Fund (PF) related information for employees. |
| pro\_ticket\_details | Contains specific details for procurement tickets, including procurement type and particulars. |
| re\_ticket\_details | Stores detailed information for reimbursement tickets, including expense categories and granted amounts. |
| reimbursement\_history | Keeps a historical record of changes and details related to reimbursement tickets, including travel and accommodation. |
| reimb\_others | Stores details for miscellaneous reimbursement expenses not covered by other specific categories. |
| report\_history | Tracks changes made to reports over time. |
| report\_logs | Logs status changes and actions taken on reports, providing an audit trail for the entire report lifecycle. |
| reports | Represents a collection of tickets, often created by a user, with overall status and details. |
| role\_actions | Defines specific actions that each role is permitted to perform within the system. |
| role\_menu | Maps roles to specific menus, controlling what each user role can see in the navigation. |
| roles | Defines different user roles within the system (e.g., Admin, Finance, Manager, User). |
| sub\_departments | Stores information about sub-departments, linked to main departments. |
| tally\_booking | Stores details related to Tally accounting software bookings, linked to tickets and payments. |
| tally\_pay\_bank | Records bank payment details for Tally integration, including routing and associated bank information. |
| tally\_payment | Contains general payment details for Tally integration, including voucher type and ledger information. |
| ticket\_history | Maintains a historical record of changes to individual ticket details. |
| ticket\_logs | Logs status changes and actions taken on individual tickets, providing an audit trail for each ticket's lifecycle. |
| tickets | Represents individual financial requests (reimbursement, procurement, advance) with their current status and key attributes. |
| travels | Stores details of travel expenses, including vehicle, distance, and travel dates, linked to reimbursement tickets. |
| user\_bank | Stores bank account details for individual users. |
| user\_history | Maintains a historical record of changes to user profiles. |
| user\_roles | Assigns roles to users, managing their permissions and access levels. |
| users | Stores basic user account information, linked to employee and other related details. |
| vehicles | Defines different vehicle types and their associated per-kilometre amounts for travel expenses. |
| vendor\_bank | Stores bank account details for vendors. |
| vendors | Contains information about registered vendors, including contact and GST details. |

# 8. Software Requirements

|  |  |
| --- | --- |
| **Tool/Technology** | **Purpose** |
| VS Code | Development platform |
| Multer | File upload handling in the backend |
| REST API | Communication between the frontend and the backend |
| Postman | API testing and validation |
| Telvin CSS | Frontend styling |
| JWT (JSON Web Token) | Secure authentication |
| Axios | Making API calls from the frontend |
| SweetAlert2 | Displaying alert messages |
| Nodemailer | Sending emails from the backend |
| Express.js | Backend routing |
| express-validator | Backend validation |
| Bcrypt | Password hashing and encryption |