Software Requirements Specification (SRS)

Project Title: Canteen Management System
Prepared By: Sameer
Date: April 30, 2025

1. Introduction

1.1 Purpose

This document specifies the requirements for the Canteen Management System for NBCC India Limited. It serves as a reference for developers, testers, and stakeholders to understand the system functionalities and constraints. The goal is to streamline meal management, allow user feedback, and support administrative operations like menu uploads and meeting requests.

1.2 Scope

The system is a web-based application supporting three roles: **Users**, **Canteen Managers**, and **Admins**. Key functionalities include:

- Viewing and uploading weekly menus.
- Scheduling and handling meetings.
- Submitting and managing meal feedback (with image support).
- Changing usernames and passwords.
- Admin dashboard for user insights.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS** – Software Requirements Specification
- **JWT** – JSON Web Token
- **API** – Application Programming Interface
- **UI** – User Interface

1.4 Overview

The remainder of this document is organized into system description, functional and non-functional requirements, use cases, and design constraints.

2. Overall Description

2.1 Product Perspective

This system is independent but modular, using a React frontend and Django REST backend. It follows the MVC (Model View Controller) architecture and REST principles.

2.2 Product Functions

- User Registration and Login (JWT)
- Weekly Menu Viewing (User) and Upload and Changes (Admin/Manager)
- Feedback with Image Upload
- Meeting Scheduling & Handling
- Dashboard Views (role-specific)
- Profile Management (Username & Password)

2.3 User Classes and Characteristics

- **User**: Can view menus, submit feedback, and request meetings.
- **Manager**: Can view feedbacks, manage menus, and handle meetings.
- **Admin**: Can upload menus, view any changes in the menu.

2.4 Operating Environment

- **Frontend**: ReactJS
- **Backend**: Django REST Framework
- **Database**: MS SQL
- **Hosting**: Localhost (Currently)

2.5 Design and Implementation Constraints

- JWT-based authentication
- Django Media configuration for file/image handling
- Bootstrap for responsive design
- Stateless API interactions

2.6 Assumptions and Dependencies

- Internet connectivity for users
- Users have access to modern browsers
- LocalStorage is supported for token management

3. Specific Requirements

3.1 Functional Requirements

3.1.1 User Registration & Authentication

- Users must be able to register using a unique email.
- JWT tokens will be issued on login and used for authentication.

3.1.2 Weekly Menu Management

- Admins can upload a weekly menu via Excel/CSV.
- Users can only view one active weekly menu sorted from Monday to Sunday.
- Any changes made by the Canteen Manager will be displayed in Today's Menu section of User Dashboard.

3.1.3 Feedback System

- Users can rate meals and provide textual feedback.
- Users can optionally upload an image of the meal.
- Managers can view all feedbacks along with images.

3.1.4 Meeting Scheduling

- Users can request meetings with managers.
- Managers can accept or decline meeting requests.

3.1.5 User Profile Management

- Users can update their username and password.

3.1.6 Analytics

- Admin can view changes in menu made by the Canteen Manager along with reasons for the same.
- Canteen Manager should generate reports based on the Feedbacks provided by the Users.

3.2 Non-Functional Requirements

3.2.1 Usability

- The UI should be intuitive and responsive.

3.2.2 Security

- JWT for authentication.

- Encrypted password storage using Django's default mechanisms.

3.2.3 Performance

- The system should support concurrent access with minimal latency.

3.2.4 Maintainability

- Clear separation of concerns between frontend and backend.
- Modular code design.

4. System Features

4.1 User Registration & Authentication

- **Description**: Allows new users to register and existing users to log in using JWT-based authentication.
- **Actors**: All users (Students, Managers, Admins)
- **Features**:
  - Secure JWT-based login and token refresh.
  - Role-based access after login.
  - Password encryption.
  - Session handling with token expiry.

4.2 Role-Based Dashboards

- **Description**: Provides tailored dashboards based on user roles (User, Manager, Admin).
- **Actors**: All roles
- **Features**:
  - Normal Users see weekly menu, give feedback, request meetings.
  - Managers handle menus, feedbacks, and meetings.
  - Admins can upload weekly menu and see changes in menu with justifications.

4.3 Weekly Menu Management

- **Description**: Enables managers to upload, edit, and update weekly meal menus.
- **Actors**: Managers, Users
- **Features**:
  - CSV-based menu upload (Monday to Sunday).
  - Auto-replace old menus with new ones.
  - Menu display sorted by weekdays.
  - Separate API endpoint for users and managers.

4.4 Menu Change Request System

- **Description**: Allows managers to change menu items with justifications.
- **Actors**: Managers
- **Features**:
    - Submit menu changes with justifications.
    - View history of past menu change requests.

4.5 Feedback System

- **Description**: Users can submit meal feedback along with an optional image.
- **Actors**: Users, Managers
- **Features**:
    - Rate meal (1 to 5 stars).
    - Add text-based remarks.
    - Upload food image for visual reference.

4.6 Feedback Management (Manager View)

- **Description**: Managers can view all submitted feedback in a structured format.
- **Actors**: Managers
- **Features**:
    - Feedback displayed with images, ratings, remarks.
    - Downloadable feedback report in PDF.
    - Sorted by submission date.
    - Includes user's name and meal type.

4.7 User Profile Management

- **Description**: Users can update their profile credentials.
- **Actors**: All users
- **Features**:
    - Change username.
    - Change password.
    - Auto-update across session.

4.9 Responsive UI with React

- **Description**: Provides user-friendly interface using React and Bootstrap.
- **Actors**: All users
- **Features**:
    - Mobile-friendly layout.
    - Dynamic components (menus, modals).
    - Toast notifications for feedback.

4.10 Secure Backend API

- **Description**: Backend exposed via Django REST Framework with role-level access control.

- **Actors**: Developers, System
- **Features**:
  - Authenticated API access.
  - CSRF, CORS, and role-based API permission classes.
  - File uploads handled securely.

4.11 Report Generation

- **Description**: Managers can download user feedback data in PDF.
- **Actors**: Managers
- **Features**:
  - Styled tabular report with meals, ratings, and comments.
  - Auto-time stamped filename.
  - Print-ready formatting.

5. Use Case Models

5.1 Use Case: Submit Feedback

- **Actor**: User
- **Description**: The user submits feedback for a meal, including a rating, remarks, and optionally an image.
- **Precondition**: User must be logged in.
- **Post-condition**: Feedback is stored in the system and visible to the manager.

Basic Flow:

1. User selects the meal type (Breakfast, Lunch, Snacks).
2. User selects a rating (1–5 stars).
3. User enters remarks.
4. User optionally uploads an image.
5. User submits the form.
6. System validates inputs and stores feedback.

Alternate Flow:

- If any required field is empty, the form will show a validation error.

5.2 Use Case: View Feedback (Manager)

- **Actor**: Manager
- **Description**: Managers can view all submitted feedback with ratings, remarks, and images.
- **Precondition**: Manager must be authenticated.
- **Post-condition**: Feedback entries are displayed in a list view, optionally exportable as PDF.

Basic Flow:

1. Manager logs in.
2. Manager navigates to the feedback section.
3. System fetches all feedback entries.
4. Manager views individual feedback cards with image (if any).
5. Manager may export feedback as a report.

5.3 Use Case: Upload Weekly Menu

- **Actor**: Admin / Manager
- **Description**: Uploads weekly meal plans which are visible to users.
- **Precondition**: User must be Admin or Manager.
- **Post-condition**: Menu is saved and displayed in chronological order.

Basic Flow:

1. Admin/Manager logs in.
2. Navigates to the menu upload section.
3. Uploads an Excel or fills a form with the weekly menu.
4. System validates and stores the menu.
5. Old menu is replaced if one already exists.

Alternate Flow:

- If file is in the wrong format, system displays an error.

5.4 Use Case: View Weekly Menu

- **Actor**: User
- **Description**: Users view the currently uploaded weekly menu.
- **Precondition**: User is logged in.
- **Post-condition**: Menu is rendered in a table from Monday to Sunday.

Basic Flow:

1. User logs in.
2. Navigates to the "Weekly Menu" section.
3. System displays the menu for each day.

5.5 Use Case: Request Meeting

- **Actor**: User
- **Description**: Users can request for snacks and other items for a meeting.
- **Precondition**: User is authenticated.
- **Post-condition**: Meeting request is stored as "Pending".

Basic Flow:

1. User navigates to the meeting request form.
2. Fills in reason and preferred time.
3. Submits the request.
4. System saves the request for manager review.

5.6 Use Case: Handle Meeting Requests

- **Actor**: Manager
- **Description**: Manager views pending meeting requests and can accept or decline them.
- **Precondition**: Manager is logged in.
- **Post-condition**: Meeting request status is updated accordingly.

Basic Flow:

1. Manager accesses the meeting management page.
2. Sees a list of pending requests.
3. Clicks accept or decline.
4. System updates request status.

5.7 Use Case: Change Password

- **Actor**: Any authenticated user
- **Description**: Allows users to securely update their password.
- **Precondition**: User is logged in.
- **Post-condition**: Password is updated after validation.

Basic Flow:

1. User accesses "Change Password" form.
2. Enters current and new passwords.
3. System validates and updates credentials.

5.8 Use Case: Change Username

- **Actor**: Any authenticated user
- **Description**: Allows users to update their username.
- **Precondition**: User is logged in.
- **Post-condition**: Username is updated in the system.

Basic Flow:

1. User opens username change modal/form.
2. Inputs new username.
3. System updates it in the database.

6. User Interface Design

- **Login Page**
- **Registration Page**
- **Dashboard Views (User/Manager/Admin)**
- **Feedback Form with Image Upload(User/Manager)**
- **Weekly Menu Table**
- **Meeting Schedule and Review Panel**
- **Change Menu**
- **Change Username**
- **Change Password**

7. References

- ReactJS Documentation
- Django & Django REST Framework
- MS SQL
- RESTful API Design