Software Requirements Specification

for

iVMS Web Application

Contents

1 Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document outlines the requirements for the Intelligent Visitor Management System (iVMS), a full-stack web application developed for Kristellar Aerospace Pvt Ltd. The application manages visitor and courier data, facilitates conference hall bookings, and provides administrative dashboards for user and data management. It is designed for intranet use within the company to streamline visitor/courier tracking and resource allocation.

1.2 Scope

The iVMS is a web-based application accessible via the intranet URL 'ivms.local'. It enables employees to register visitors and couriers, book conference halls, and manage related data through dashboards. Key features include user authentication, visitor/courier data entry with face recognition, hall booking with floor/room selection, and administrative controls for user management and booking oversight. The system uses React with Vite for the frontend, Node.js and Python (for face recognition) for the backend, and PostgreSQL for data storage. This SRS defines functional and non-functional requirements, system architecture, and constraints.

1.3 Definitions, Acronyms, and Abbreviations

- **iVMS**: Intelligent Visitor Management System.
- **React**: JavaScript library for building user interfaces.
- **Vite**: Frontend build tool for development and bundling.
- **Node.js**: JavaScript runtime for backend services.
- **Python**: Programming language used for face recognition.
- **PostgreSQL**: Relational database management system.
- **API**: Application Programming Interface.
- **UI**: User Interface.
- **UX**: User Experience.
- **WCAG**: Web Content Accessibility Guidelines.

2 Overall Description

2.1 Product Perspective

The iVMS is a full-stack intranet application hosted on an internal server (accessible at 'ivms.local') using NGINX for routing. The frontend, built with React and Vite, interacts with a Node.js backend for core operations and a Python backend for face recognition. PostgreSQL stores all data, including users, visitors, couriers, and booking records. The system integrates with a webcam for face recognition to verify visitors/couriers.

2.2 Product Functions

The application provides the following major functions:

- User authentication (login, registration, logout).
- Visitor data entry with face recognition and storage.

- Courier data entry and storage.
- Conference hall booking with floor and room selection.
- Dashboards for viewing/managing visitors, couriers, and bookings.
- Administrative panel for user management (create, view, delete users).
- Email notifications for booking confirmations and status updates.

2.3 User Classes and Characteristics

- **Employees**: Register visitors/couriers, book conference halls, and view dashboards. They require intuitive UI and reliable data entry.
- **Administrators**: Manage user accounts, oversee bookings, and access all data. They need secure access to administrative controls.

2.4 Operating Environment

The application runs on modern web browsers (Chrome, Firefox, Edge) within the company's intranet. It requires a stable internal network connection and a webcam for face recognition. The frontend uses React 18.x and Vite 5.x, the backend uses Node.js 20.x and Python 3.x, and the database uses PostgreSQL 15.x. The system is deployed on an NGINX server with PM2 for backend process management.

2.5 Design and Implementation Constraints

- The application must use React with Vite for the frontend, Node.js and Python for the backend, and PostgreSQL for the database.
- Must operate within the company intranet("ivms.local").
- Face recognition requires a compatible webcam and Python libraries (e.g., face_recognition).
- Must adhere to company security policies for data handling.

2.6 Assumptions and Dependencies

- Users have access to intranet-connected devices with modern browsers.
- Webcam hardware is available and functional for face recognition.
- PostgreSQL database is hosted on an internal server with sufficient capacity.
- Stable intranet connection for API and HTTP POST communication.

3 Functional Requirements

3.1 User Authentication

- **FR1**: The system shall allow employees to register with an username, password, and role.
- **FR2**: The system shall allow employees to log in using valid credentials.
- **FR3**: The system shall provide a logout option.
- **FR4**: Administrators shall manage user accounts (create, view, delete).

3.2 Visitor Management

- **FR5**: The system shall allow employees to enter visitor data (e.g., name, contact, purpose to visit, phone number, email id, photo, etc).
- **FR6**: The system shall capture visitor photos via webcam and process them with face recognition.
- **FR7**: The system shall store visitor data in PostgreSQL.
- **FR8**: The system shall display a visitor list in a dashboard with search/filter options.

3.3 Courier Management

- **FR9**: The system shall allow employees to enter courier data (e.g., name, contact, delivery details).
- **FR10**: The system shall store courier data in PostgreSQL.

3.4 Hall Booking

- **FR11**: The system shall allow employees to book conference halls by selecting date, time, floor (e.g., 3rd, 4th), room, and purpose.
- **FR12**: The system shall validate booking inputs (e.g., no overlapping bookings).
- **FR13**: The system shall display bookings in a dashboard with calendar view (using React Big Calendar).
- **FR14**: The system shall send email notifications for booking confirmations and status updates.

3.5 Administrative Dashboards

- **FR15**: The system shall provide dashboards to view and manage visitors, couriers, and bookings.
- **FR16**: Administrators shall update booking statutes (e.g., approve, reject).
- **FR17**: The system shall allow searching/filtering by email, or phone number only for visitors.

4 Non-Functional Requirements

4.1 Performance

- **NFR1**: The application shall load the initial page in under 2 seconds on the intranet.
- **NFR2**: API response times shall not exceed 5 second under normal conditions.
- **NFR3**: Face recognition processing shall complete within 5 seconds per image.

4.2 Security

- **NFR4**: User data and API communications shall be encrypted using HTTPS.
- **NFR5**: Passwords shall be hashed (e.g., using bcrypt) before storage in PostgreSQL.
- **NFR6**: Access to administrative features shall be restricted to authorized users.

4.3 Usability

- **NFR7**: The UI shall be responsive for screen sizes from 320px to 1920px.
- **NFR8**: The application shall provide intuitive navigation and form validation feedback.

4.4 Reliability

- **NFR9**: The application shall have 99.9% uptime within the intranet, excluding maintenance.
- **NFR10**: Face recognition shall achieve at least 95% accuracy under optimal conditions.

5 System Architecture

The iVMS uses a client-server architecture:

- **Frontend**: React 18.x with Vite 5.x, hosted on NGINX, accessible at 'ivms.local'.
- **Backend**:
- Node.js 20.x for core operations (authentication, data management, hall booking).
- Python 3.x with Flask for face recognition, using HTTP POST for real-time processing.
- **Database**: PostgreSQL 15.x for storing user, visitor, courier, and booking data.
- **APIs**: RESTful endpoints (Node.js) for data operations; HTTP POST (Python) for face recognition.

6 External Interface Requirements

6.1 User Interfaces

The application provides a web-based UI with:

- **Login Page**: Form with email, password, and icons (e.g., UserIcon, LockIcon).
- **Visitor/Courier Forms**: Fields for name, contact, purpose, and photo capture.
- **Hall Booking Form**: Fields for date, time range, floor, room, and purpose.
- **Dashboards**: Tables and calendar views for visitors, couriers, and bookings.
- **Admin Panel**: User management and booking oversight. Styling uses Tailwind CSS and Lucide-React icons (e.g., ArrowRight, CheckCircle).

6.2 Hardware Interfaces

- Webcam for capturing visitor/courier photos.
- Intranet-connected devices for accessing 'ivms.local'.

6.3 Software Interfaces

- **React**: Version 18.x for UI components.
- **Vite**: Version 5.x for development and bundling.
- **Node.js**: Version 20.x with Express for REST APIs.
- **Python**: Version 3.x with face recognition, and OpenCV for face recognition.
- **PostgreSQL**: Version 15.x for data storage.
- **Other**: React Big Calendar for booking visualization, Nodemailer for email notifications, Axios for API calls, React Router for navigation, MSG91 for OTP verification.

6.4 Communications Interfaces

- HTTPS for secure API communication between frontend and Node.js backend.
- HTTP POST for real-time face recognition with Python backend.
- SMTP for email notifications via Nodemailer.

7 Other Requirements

7.1 Database

PostgreSQL schema includes:

- **Users**: Fields for email, password (hashed), role.
- **Visitors**: Fields for name, contact, purpose, photo, face encoding.
- **Couriers**: Fields for name, contact, delivery details.
- **Bookings**: Fields for request ID, date, time, floor, room, purpose, status.

7.2 Legal

The application shall comply with company data privacy policies and intranet security standards.

7.3 Maintenance

The system allows updates to forms, dashboards, and backend logic without downtime, using PM2 for process management.