Functional Requirements Specification (FRS)

for

iVMS Web Application

1. Introduction

1.1 Purpose

This Functional Requirements Specification (FRS) outlines the functional requirements for the iVMS (Intelligent Visitor Management System) 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 booking management within the company's intranet. It includes a face recognition module for visitor verification. This document serves as a reference for stakeholders, developers, and testers to ensure the application meets user needs.

1.2 Scope

The iVMS web application provides the following key functionalities:

- User authentication (registration, login, logout) for employees and administrators.
- Visitor registration and management with face recognition for verification.
- Courier data entry and management.
- Hall booking with floor and room selection.
- Administrative dashboards for managing users and bookings.
- Responsive design for access on desktop and mobile devices within the intranet (accessible at ivms.local).

The application uses React with Vite for the frontend, Node.js with PostgreSQL for the primary backend, and a Python Flask server for face recognition. It is deployed on an intranet server (e.g., Apache or NGINX) and integrates with a PostgreSQL database.

1.3 Definitions, Acronyms, and Abbreviations

- **iVMS**: Intelligent Visitor Management System.
- **React**: JavaScript library for building user interfaces.
- **Vite**: Build tool for fast development and optimized builds.
- **Node.js**: JavaScript runtime for the primary backend.
- **Flask**: Python framework for the face recognition backend.
- **PostgreSQL**: Relational database for storing visitor, courier, booking, and user data.
- **CRUD**: Create, Read, Update, Delete operations.
- **API**: Application Programming Interface.

2. Functional Requirements

2.1 User Authentication

2.1.1 User Registration

- **ID**: FR-1
- **Description**: Administrators shall register new users (employees or admins) via a form.
- **Input**: Email, username, password, role (employee or admin).
- **Output**: Success message or error (e.g., "Email already exists").
- **Dependencies**: Node.js backend API (/api/register), PostgreSQL users table.

2.1.2 User Login

- **ID**: FR-2
- **Description**: Users shall log in using email and password at ivms.local.
- **Input**: Email, password.
- **Output**: Redirect to dashboard (admin or employee) or error (e.g., "Invalid credentials").
- **Dependencies**: Node.js backend API (/api/login), React components (Login.jsx).

2.1.3 User Logout

- **ID**: FR-3
- **Description**: Users shall log out, clearing their session.
- **Input**: Click logout button.
- **Output**: Redirect to login page (ivms.local).
- **Dependencies**: Node.js session management.

2.2 Visitor Management

2.2.1 Visitor Registration

- **ID**: FR-4
- **Description**: Employees shall register visitors via a form, capturing details and a photo for face recognition.
- **Input**: Visitor name, contact, purpose, photo (uploaded to /uploads) and etc.
- **Output**: Success message, visitor ID, or error.
- **Constraints**: Photo must be a valid image file (.jpg, .jpeg, .png).
- **Dependencies**: Node.js backend API, PostgreSQL visitor table, Python Flask server for photo storage.

2.2.2 Face Recognition Verification

- **ID**: FR-5

- **Description**: The system shall verify a visitor's identity by comparing a captured photo against stored images.
- **Input**: Photo uploaded to /uploads/temp.jpg.
- **Output**: JSON response with matchId (visitor ID) or null if no match.
- **Constraints**: Face recognition must use face_recognition library; match threshold < 0.5.
- **Dependencies**: Python Flask server (/compare endpoint), face_recognition library.

2.2.3 View Visitor Data

- **ID**: FR-6
- **Description**: Admins shall view a list of registered visitors.
- **Output**: List of visitors with details (name, contact, visit date).
- **Dependencies**: Node.js backend API, PostgreSQL visitor table.

2.3 Courier Management

2.3.1 Courier Registration

- **ID**: FR-7
- **Description**: Employees shall register courier details via a form.
- **Input**: Courier sender, receiver, description.
- **Output**: Success message or error.
- **Dependencies**: Node.js backend API, PostgreSQL courier table.

2.3.2 View/Update/Delete Courier

- **ID**: FR-8
- **Description**: Admins shall view courier records.
- **Output**: List of couriers or success/error message.
- **Constraints**: Only admins/user roles can view records.
- **Dependencies**: Node.js backend API, PostgreSQL courier table.

2.4 Hall Booking

2.4.1 Book Hall

- **ID**: FR-9
- **Description**: Employees shall book a conference hall via a form with floor and room selection.
- **Input**: Name, email, date, time range, purpose, floor (e.g., 3rd, 4th), room (dependent on floor).
- **Output**: Success message or error (e.g., "Room unavailable").

- **Constraints**: Time range must be valid; room availability checked.
- **Dependencies**: Node.js backend API, PostgreSQL booking table.

2.4.2 View/Update/Delete Booking

- **ID**: FR-10
- **Description**: Office Executive roles shall view, accept, or reject bookings on the dashboard.
- **Output**: List of bookings in a managed way.
- **Dependencies**: Node.js backend API, PostgreSQL booking table.

2.5 Administrative Dashboard

2.5.1 Manage Users

- **ID**: FR-11
- **Description**: Admins shall view, create, or delete user accounts.
- **Input**: User details or action (create/view/delete).
- **Output**: Updated user list or success/error message.
- **Constraints**: Only admins can access.
- **Dependencies**: Node.js backend API, PostgreSQL users table.

2.5.2 Manage Bookings and Data

- **ID**: FR-12
- **Description**: Admins shall view and manage all visitor, courier user accounts.
- **Output**: Dashboard with data tables and management options.
- **Dependencies**: Node.js backend API, PostgreSQL tables.

2.6 Responsive Design

2.6.1 Cross-Device Compatibility

- **ID**: FR-13
- **Description**: The application shall be responsive for desktop at ivms.local.
- **Constraints**: Support for modern browsers (Chrome, Firefox) and screen sizes (320px–1920px).
- **Dependencies**: React components, Tailwind CSS (assumed from prior discussions).

3. Non-Functional Requirements

- **Performance**: Page load time < 2 seconds; face recognition response < 5 seconds.
- **Scalability**: Support up to 100 concurrent users within the intranet.
- **Security**: HTTPS for data in transit; encrypted storage in PostgreSQL.

- **Accessibility**: Comply with WCAG 2.1 Level AA.

4. Assumptions and Constraints

- **Assumptions**:
    - Intranet server (NGINX) is configured for ivms.local.
    - PostgreSQL database is accessible to the Node.js backend.
    - Python Flask server runs on port 5002 for face recognition.
    - Users have modern browsers and intranet access.
- **Constraints**:
    - Limited to React with Vite, Node.js, Python Flask, and PostgreSQL.
    - Face recognition relies on face_recognition library and image files in /uploads.

5. System Interfaces

- **User Interface**: React SPA with components (Login.jsx, VisitorForm.jsx, HallBookingForm.jsx, etc.) and icons (e.g., EyeIcon, UserIcon etc).
- **API Interfaces**:
    - Node.js API for authentication, visitor, courier, and booking operations.
    - Python Flask API (/compare) for face recognition.
- **External Libraries**:
    - Frontend: React, Vite, Tailwind CSS, lucide-react for icons.
    - Backend: Node.js, PostgreSQL, Python (face recognition, flask).

6. Acceptance Criteria

- Users can register, log in, and log out at ivms.local.
- Visitors can be registered with photos and verified via face recognition.
- Couriers and hall bookings can be created and managed.
- Admins can view and manage all data via dashboards.
- Application is responsive and accessible within the intranet.