TAMIL S P

S.NO : 90

PID 30

Industry Relation Programme

1.System Overview

This document outlines the design of the Industrial Relationship Program Management System, a web-based application built using the MERN stack (MongoDB, Express.js, React.js, Node.js). The system aims to streamline the management of industrial relationships for educational institutions by providing a centralized platform for storing, organizing, and accessing data related to industrial visits (both inbound and outbound).

2. Problem Statement

Traditionally, managing industrial relationships has relied on manual techniques or spreadsheet-based solutions. These methods pose limitations in data scalability, searchability, and accessibility. Retrieving specific information about past or upcoming industrial interactions becomes cumbersome, hindering effective program management.

3. System Objectives

The primary objective of this system is to develop a robust Industrial Relationship Program Management System. This application will address the aforementioned challenges by offering the following functionalities:

- **Admin User Management:** Create, edit, and delete admin accounts for secure system access.
- **Visitor Management:** Manage comprehensive visitor profiles encompassing personal details (name, email, phone number), professional details (company, position, domain, LinkedIn profile URL), and visit-specific information.

- **Search and Filter:** Implement a search function with various criteria (name, company, domain, etc.) for efficient data retrieval. Integrate filter options to refine search results based on specific parameters.

4. System Architecture

The system adopts a Model-View-Controller (MVC) design pattern for modularity and maintainability.

- **Frontend (React.js):** This layer handles user interactions and visual presentation of data. React components will be responsible for rendering the user interface, handling user input, and communicating with the backend API.
- **Backend (Express.js):** This layer serves as the application's core, handling business logic and data persistence. Express.js routes will handle API requests from the frontend, interact with the database, and perform necessary operations on visitor data.
- **Database (MongoDB):** The system will leverage MongoDB, a NoSQL document database, to store visitor information and other relevant data in a flexible and scalable manner. Mongoose ODM (Object Data Modeling) will likely be used for data modeling and interacting with the MongoDB database from the backend.

5. System Requirements

5.1. Functional Requirements

- **User Authentication:** Secure login system for authorized admins using industry-standard password hashing techniques.
- **Admin Dashboard:** Provide a user-friendly dashboard for admins to manage visitor profiles.
- **Visitor Management:**
    - Create new visitor profiles with comprehensive personal and professional details.
    - Edit existing visitor information to maintain data accuracy.
    - Delete visitor profiles when necessary.
- **Search Functionality:** Implement a search bar allowing admins to search for visitors based on various criteria like name, company, domain, etc.
- **Filter Functionality:** Integrate filter options to refine search results based on specific parameters (e.g., company type, domain, visit type).

5.2. Non-Functional Requirements

- **Security:** Ensure secure data storage practices, user authentication mechanisms, and authorization controls to protect sensitive information.

- **Performance:** Maintain optimal system responsiveness and scalability to accommodate a growing user base and data volume.
- **Availability:** Strive for high system availability to minimize downtime and ensure continuous accessibility for authorized users.
- **Responsiveness:** Design the application for responsiveness across various devices (desktops, tablets, smartphones) to provide a seamless user experience.

6. Data Flow

- **Data Input:** Authorized admins will create, edit, and delete visitor profiles through the user interface.
- **Data Processing:** The backend API will handle data validation, sanitization, and interaction with the MongoDB database.
- **Data Storage:** Visitor information will be stored in MongoDB documents with appropriate schema definitions.
- **Data Retrieval:** Admins can search and filter visitor data using the implemented functionalities. Search queries and filters will be processed by the backend API to retrieve relevant data from the database.
- **Data Output:** The retrieved visitor information will be presented to admins through the user interface.

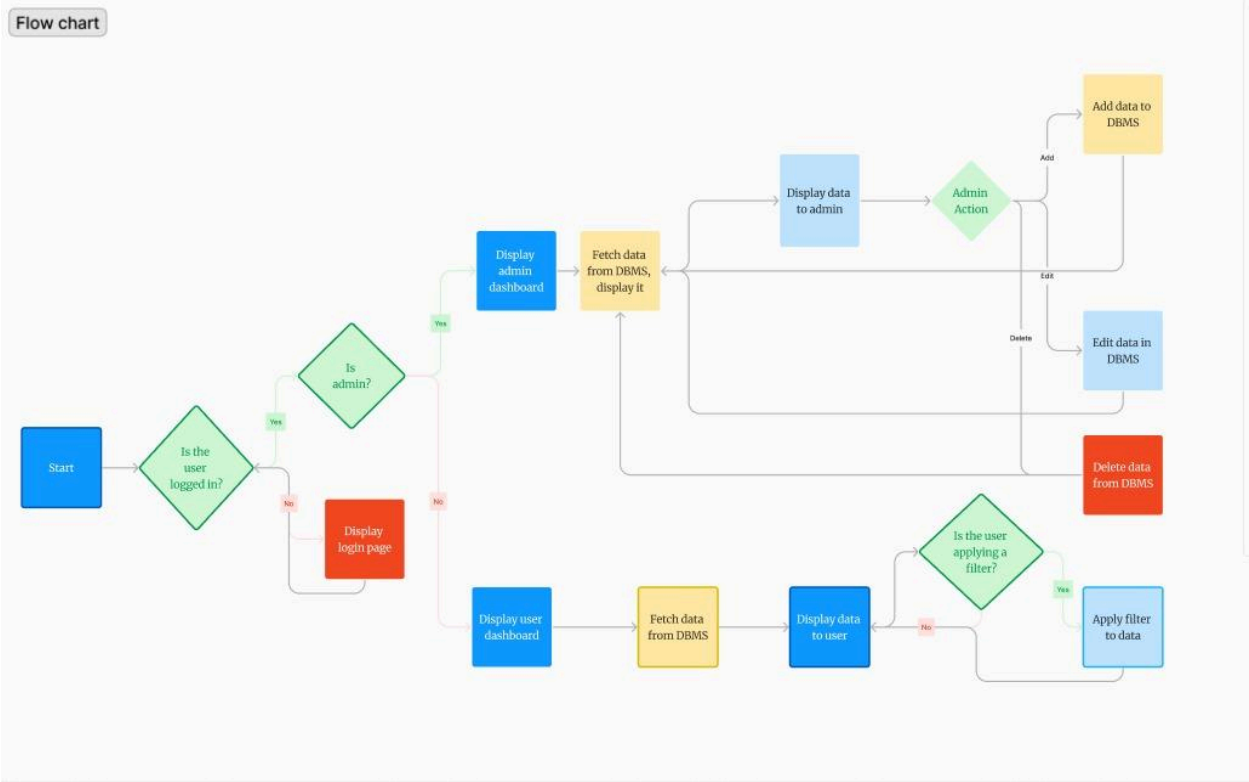
7. Technologies

- **Frontend:** React.js (or a suitable alternative frontend framework)
- **Backend:** Express.js (Node.js framework)
- **Database:** MongoDB
- **Database ODM:** Mongoose
- **Additional Libraries:** Potential use of frontend libraries or UI component kits for enhanced user interface development.

8. Future Enhancements

- **Notes and Follow-up Features:** Implement functionalities for adding notes and tracking follow-up actions related to

FLOWCHART


Flow chart
Start
Is the user logged in?
Yes
No
Is admin?
Yes
No
Display login page
Display admin dashboard
Fetch data from DBMS, display it
Display data to admin
Admin Action
Add
Edit
Delete
Add data to DBMS
Edit data in DBMS
Delete data from DBMS
Display user dashboard
Fetch data from DBMS
Display data to user
Is the user applying a filter?
Yes
No
Apply filter to data