**Project Documentation - Organisation**

**1. Introduction**

1.1 Overview

The Organisation Project is a full-stack web application that allows administrator to manage organizations, employees, and their details. The technology stack includes React JS for the front end, Java-Spring Boot for the back end and MongoDB for the database.

# Organisation project is a full stack Project.

# React JS -> https://github.com/chinnamrajukasina/Visual-tudio/tree/main/React/organization-ui

# Java-> https://github.com/chinnamrajukasina/Java/tree/main/organisation

**2. Objective**

* The application ensures seamless communication between the frontend and backend, allowing the administrator to perform dynamic CRUD operations on organizations and employees.
* Administrators can create organizations by providing essential information such as the organization's name and details, fostering a flexible and user-friendly interface.
* Within each organization, administrators have the capability to add employees along with their specific details, contributing to a comprehensive record-keeping system.
* The project emphasizes real-time updates, empowering administrators to modify existing data promptly. Whether it's updating an employee's information or refining organization details, the system facilitates efficient and accurate data management.
* For enhanced flexibility, the administrator retains the ability to remove specific employees or entire organizations as needed. This feature streamlines data maintenance and ensures the database remains accurate and up to date.
* Responsive design principles have been implemented to guarantee that administrators can manage organizations and employees seamlessly across various devices and screen sizes.
* Documentation is comprehensive, offering clear insights into the codebase, API endpoints, and overall functionality, facilitating collaboration among developers and supporting future enhancements.

**2.1 Screenshots**

**A screenshot of a computer

Description automatically generated**

**A computer screen shot of a chat box

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Current State**: As on date 30/11/2023 connected successfully to the mongo dB database and retrieve the date in the database. Changes made at Organisation level are reflected to database and for Employees APIs to be create.

**Possible Improvements:**

* Implementing more API calls for CRUD operations.
* Adding form validations and error handling.
* Enhancing user interface and experience.
* With a robust logging and auditing system in place, the application provides administrators with a detailed history of events and actions, aiding in tracking changes and diagnosing potential issues.
* The application prioritizes a secure and authenticated experience, allowing only authorized administrators to access and manipulate sensitive data.
* To augment user experience, the system incorporates intuitive form validations, error handling, and user notifications, ensuring administrators are guided through their interactions with the application.

**3. Front-end (React JS)**

**3.1.1 Project Structure**

1. src: The main source code directory.
   * components: Contains reusable React components used across the application.
   * containers: High-level components that represent entire views or pages.
   * services: Utility functions or services for making API calls or handling data logic.
   * styles: Global styles or styling variables for the application.
2. public: Publicly accessible files.
   * index.html: The main HTML file where the React application is mounted.
3. node\_modules: Dependencies installed via npm.

**3.1.2 Files**

1. package.json: Configuration file that includes project metadata, dependencies, and scripts.
2. package-lock.json: Lock file that ensures consistent package installations across different environments.
3. .gitignore: Specifies files and directories that should be ignored by Git.
4. README.md: Project documentation containing essential information about the project, setup instructions, and usage guidelines.
5. .git: Git configuration and repository metadata.
6. public/index.html: The main HTML file where the React application is mounted.
7. src/index.js: Entry point of the React application, where the root component is rendered.
8. src/App.js: The main application component.
9. src/index.css or App.css: Global styles for the application.

**3.2 Installation**

Node.js and npm Installation: Make sure you have Node.js and npm installed on your machine. You can download them from the official website: [Node.js Downloads](https://nodejs.org/)

**3.3 Navigate to the Project:**

git clone https://github.com/chinnamrajukasina/Visual-tudio/tree/main/React/organization-ui

Move into the newly created directory: cd organization-ui

npm start.

**3.4 Features**

* UI is implemented using ReactJS and each functionality is grouped by components.

1. **App (App.js):**
   * Main component rendering the **admin** component.
   * Imports **data** and passes it as a prop to the **admin** component.
2. **Admin (Admin.js):**
   * Receives **orgs** as a prop and renders the **OrgList** component.
   * Imports **OrgList** and passes **orgs** as a prop to it.
3. **OrgList (OrgList.js):**
   * Displays a list of organizations (**orgs**).
   * Uses the **Org** component to render individual organization cards.
   * Utilizes **useState** to manage the state of organizations.
4. **Org (Org.js):**
   * Represents an organization card.
   * Allows updating, removing, and adding employees to the organization.
   * Uses the **EmpList** component to render a list of employees.
5. **EmpList (EmpList.js):**
   * Displays a list of employees for a given organization.
   * Uses the **Emp** component to render individual employee cards.
6. **Emp (Emp.js):**
   * Represents an employee card.
   * Allows updating and removing an employee.

**Functionality:**

* **Updating Organization (handleUpdateOrg):**
  + Logs update operations for an organization.
  + Updates the organization's name and address.
* **Removing Organization (handleRemoveOrg):**
  + Logs remove operations for an organization.
  + Filters out the organization with the specified ID.
* **Adding Employee to Organization (handleAddEmployee):**
  + Adds a new employee to the organization with an auto-generated ID, name, and email.
* **Updating Employee (handleUpdate in Org.js and Emp.js):**
  + Logs update operations for an employee.
  + Updates the employee's name and email.
* **Removing Employee (handleRemoveEmp in EmpList.js and Emp.js):**
  + Logs remove operations for an employee.
  + Filters out the employee with the specified ID.

**Styling:**

* Styling is applied using inline styles in the components.
* **Card** components from the Ant Design library are used for organization and employee cards.
* Flexbox is used to arrange organization cards side by side with a 20px gap.

A screenshot of a computer

Description automatically generated

**4. Back-end (Java-Spring Boot)**

**4.1 Project Structure**

4.1.1 Packages

1. com.example.organization:
   * Application.java: The main class with the main method to run the Spring Boot application.
   * config: Configuration classes for Spring, such as database configuration or security settings.
2. com.example.organization.controller:
   * OrgController.java: Controller class handling HTTP requests related to organizations.
   * EmpController.java: Controller class handling HTTP requests related to employees.
3. com.example.organization.model:
   * Organization.java: Entity class representing an organization with details like name and address.
   * Employee.java: Entity class representing an employee with details like name and email.
4. com.example.organization.repository:
   * OrganizationRepository.java: Interface extending Spring Data JPA's JpaRepository for organization CRUD operations.
   * EmployeeRepository.java: Interface extending Spring Data JPA's JpaRepository for employee CRUD operations.
5. com.example.organization.service:
   * OrgService.java: Service class containing business logic related to organizations.
   * EmpService.java: Service class containing business logic related to employees.
6. com.example.organization.exception:
   * CustomException.java: Custom exception class for handling application-specific exceptions.
7. com.example.organization.dto:
   * OrganizationDTO.java: Data Transfer Object (DTO) for transferring organization-related data between layers.
   * EmployeeDTO.java: DTO for transferring employee-related data.
8. com.example.organization.mapper:
   * OrgMapper.java: Mapper class for converting between Organization entities and OrganizationDTOs.
   * EmpMapper.java: Mapper class for converting between Employee entities and EmployeeDTOs.
9. com.example.organization.security:
   * SecurityConfig.java: Configuration class for Spring Security settings.
10. com.example.organization.util:
    * ApiResponse.java: Utility class for creating standardized API responses.
    * Constants.java: Constants used throughout the application.

4.1.2 Files

1. pom.xml: Maven configuration file containing project dependencies and build settings.
2. application.properties: Configuration file for application-specific properties like database connection details.
3. MainApplicationTests.java: JUnit test class for testing the main application.
4. MainApplicationIntegrationTests.java: Integration tests for testing the application's components together.

**4.2 Installation**

Go to the official Spring Tool Suite download page: [Spring Tools 4 - for Eclipse](https://spring.io/tools).

Clone the Repository using the command in the terminal:

git clone https://github.com/chinnamrajukasina/Java/tree/main/organisation

cd organisation

**4.3 Configuration**

* configuration settings for database connection for mongo DB .

**5 Contact Information**

Name: K S K Chinnam Raju

Email: [chinnamrajukasina@gmail.com](mailto:chinnamrajukasina@gmail.com)

GitHub: https://github.com/chinnamrajukasina