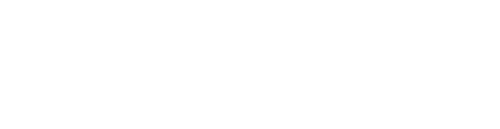


|  |
| --- |
| Technical Assessment Documentation |
|  |
| September 6  EPI-USE  Authored by: Vian Reynecke |

**

|  |
| --- |
| **Employee Hierarchy Management System -** [https://vian-epiuse.vercel.app/](https://vian-epiuse.vercel.app/%20)OverviewThis cloud-hosted application manages an employee hierarchy. It allows users to create, read, update and delete employee data, control the reporting structure and visually represent the organizational hierarchy. Gravatar is integrated for employee avatars and the system features both list and hierarchy views for managing employees. The application is deployed with Vercel linked to Supabase, ensuring data persistence and accessibility.Key Features **1. Landing Page**  The landing page provides a brief description of the project. It welcomes users and explains the main features of the application.  **2. Home Page**   * *2.1 Create Tab (A*llows users to create new employees, with validation on all inputs)   + **Reporting Line Manager:** Users cannot select a reporting manager if the employee is the CEO. Additionally, only one CEO can exist in the system.   + **Validation:** The system checks for duplicate emails , missing input and input format.   + **Error Handling:** Relevant error messages are displayed for invalid input or missing data. * *2.2. Manage Tab* * 2.2. Hierarchy View (Displays employee hierarchy in a tree structure.) * **Search:** Users can search by name, surname or role and the system will highlight the found employee in the tree. * **Edit Functionality:** Users can edit an employee’s data by clicking on the edit button next to the employee’s name. Validation includes:   + Users cannot change their role if others report to them.   + Users cannot assign themselves as their own manager.   + Users cannot assign their subordinates as their managers (cannot report to subordinates).   + Users cannot be deleted if they have subordinates.   + Includes all other validation on inputs from the create tab. * 2.2.2 List View (Displays a sortable and filterable list/table view of employees.) * **Sort & Filter:** Users can sort by name, role, email and manager. Sorting can be combined with filtering. * **Search:** Users can search for employees using the same criteria as in the hierarchy view. |
| 3. Hierarchy Page   * Displays the hierarchy in tree form with nodes that can be expanded or collapsed for ease of use. * User nodes are represented by + and - symbols to control expansion and collapse for better viewing.   **4. About Page**   * A simple page providing information about the project and a brief bio of myself * Links: Includes links to my GitHub, BlendWeb and LinkedIn profiles.   **5. Dark and Light Mode Styling**  The application automatically adapts to the user's current system mode (dark or light), ensuring a seamless user experience regardless of device settings. Feel free to test this out!  **6. Additional Features**   * **Gravatar Integration:** Employees are assigned a profile picture based on their email using Gravatar, I hashed the email using md5 hashing to safely obtain the URL and not exposing the user’s email. * **Validation:** Extensive validation is implemented to ensure data integrity (e.g., unique emails, role restrictions). * **Role Restrictions:** Users cannot assign certain roles if it would break the hierarchy (e.g., assigning a subordinate as a manager).  Technical Overview The application follows an **API Gateway architecture**, where all client requests are routed through a central API gateway that connects to the necessary services. This allows for a clear separation of concerns between the frontend and backend while centralizing request handling and security.   * **Next.js** is used for the frontend, making server-side rendering (SSR) and static site generation (SSG) possible, improving performance and user experience. * **Supabase** acts as the backend service, providing a real-time database and API endpoints for managing employee data and performing CRUD operations.   **1. Technologies**   * *Frontend:*   + **Next.js:** A powerful React framework that handles routing, server-side rendering and API route creation. It supports JavaScript and TypeScript, offering type safety and better maintainability.   + **Next UI:** A UI library that simplifies interface development by offering pre-built components for forms, tables and the hierarchy view.   + **Tailwind CSS:** Used for styling and ensuring responsive design, including support for light and dark mode, which adapts to the user's system preferences. * *Backend:*   + **API Gateway:** Centralized entry point for all client requests, providing access control and routing to the backend services hosted on Supabase. It abstracts complexity and ensures security.   + **Supabase:** Provides the API and handles database operations. With PostgreSQL at its core, it ensures data integrity and offers real-time capabilities for managing employee records and relationships.   + **Gravatar:** Integrated to fetch profile pictures for employees using their email. * *Deployment:* * **Vercel**: The application is deployed on Vercel, offering a streamlined CI/CD pipeline linked to the main branch of the GitHub repository. This provides automatic scaling, fast builds and a globally available app.   **2. Justification for Technology Choices**   * **API Gateway:** Ensures centralized request handling, which simplifies security and request routing between the frontend and backend. * **Next.js:** Chosen for its ability to handle both client-side and server-side rendering, which enhances application performance, especially in a data-driven environment. * **TypeScript:** Ensures robust type-checking, improving code reliability and preventing common bugs during development. * **Supabase:** Provides a powerful backend-as-a-service solution, simplifying API creation, database management and real-time capabilities. * **Gravatar Integration:** Reduces complexity by fetching profile pictures based on employee email addresses. * **Vercel:** Provides seamless deployment and hosting for Next.js applications, ensuring fast performance, scalability and simple CI/CD integration. |

3. Design Patterns

* 3.1 Repository Pattern
  + Use: Abstracts data access logic through services or utility functions for employee records.
  + Benefit: Separates data access from business logic, enhancing maintainability and testability.
* 3.2 Observer Pattern
  + Use: Utilizes reactive programming or state management to update UI components when employee data changes.
  + Benefit: Ensures the UI automatically reflects data changes, keeping the design decoupled.
* 3.3 Factory Pattern
  + Use: Defines functions or methods to create complex components or views for different employee roles.
  + Benefit: Centralizes creation logic, providing flexibility and consistency in component management.

## Database Setup Overview

Database Management System: PostgreSQL

1. Table: *employees*

* Purpose: Stores comprehensive information about employees within the organization, including personal details, employment information and role assignments.
* Table Structure:
  + id (UUID): Unique identifier for each employee, serves as the primary key.
  + name (VARCHAR): First name of the employee.
  + surname (VARCHAR): Last name of the employee.
  + birth\_date (DATE): Date of birth of the employee.
  + employee\_id (VARCHAR): Unique internal identifier for each employee, used for internal tracking.
  + salary (NUMERIC or DECIMAL): The salary of the employee.
  + email (VARCHAR): Email address of the employee, utilized for Gravatar integration and communication.
  + updated\_at (TIMESTAMP): Timestamp indicating when the record was last updated.
  + created\_at (TIMESTAMP): Timestamp indicating when the record was created.
  + role (role): The employee’s role, constrained by the custom ENUM type role.
  + reporting\_line\_manager: The role of the employee’s manager, constrained by the custom ENUM type role.
  + reporting\_id: (INTEGER): Foreign key referencing the id of the employee’s manager, establishing hierarchical relationships.

2. Custom ENUM Type: role

Definition: ENUM type defining a set of predefined roles within the organization.

*Values:*

* HR Manager
* IT Manager
* Finance Manager
* IT Intern
* HR Specialist
* Financial Analyst
* Accountant
* Senior Developer
* System Administrator
* Database Administrator
* Recruiter

3. Design Considerations

* Role Management: The reporting\_line\_manager field of type role captures the role of the manager, while reporting\_id links to the actual manager’s ID, allowing for both role-based and ID-based references in hierarchical relationships.
* Data Integrity: Foreign key constraints ensure that the reporting\_id refers to a valid employee, maintaining accurate hierarchical relationships.
* Consistency: The role ENUM type enforces valid role values, ensuring consistency across role assignments.