hierarKEY

Technical Document

Overview

This document outlines the architecture, design patterns and technologies used in the development of the hierarKEY. The aim is to ensure that the solution aligns with the project's functional requirements.

Architecture

The solution is based on a Client-Server and Model-View-Controller (MVC) architecture, combining the strengths of both to create a maintainable and modular system. These architecture styles were chosen due to:

- **Clear Separation of Concerns:** Client-Server separates frontend and backend responsibilities, while MVC structures the solution into 3 distinct layers:
  - Model: Handles data management and business logic, interacting with the database via Supabase RPC.
  - Controller: Processes incoming requests and responses.
  - View: The frontend React application renders data received from the backend.
- **Security:** Sensitive business logic is securely handled on the server side, thus minimizing exposure on the client.
- **Reusability:** Backend APIs can serve multiple clients and MVC's separation enables easy modifications to one part without impacting others.
- **Maintainability:** MVC enables easier updates by separating data access, business logic and UI, while Client-Server architecture supports independent deployment cycles.

Design Patterns

- **Singleton:** The Supabase client is implemented as a singleton by exporting a single shared instance from a dedicated file. This ensures that all parts of the application use the same connection, reducing resource overhead and maintaining consistency when interacting with the backend services.
- **Facade:** Supabase RPC (Remote Procedure Call) functions are used to encapsulate SQL logic. These functions act as a facade, exposing a clean and unified interface to the application while hiding the underlying database complexity. This simplifies the codebase and reduces coupling between the application and database layers.
- **Proxy:** Proxy is applied through API route handlers that sit between the client and the database. These handlers include additional logic such as authentication checks before passing requests to the underlying Supabase functions, effectively acting as intermediaries that control access to the real operations.
- **Strategy:** Role-based access control is implemented using a strategy-like approach. Different user roles (editor and viewer) determine the behavior and available actions within the application. For instance, only editors are permitted to add employees.

Technologies and Justification

Frontend: React with Material UI

- React was chosen for its component-based architecture, fast rendering with the virtual DOM, and strong ecosystem support. It allows for building dynamic, modular and reusable UI components.
- Material UI (MUI) provides a comprehensive set of prebuilt, responsive components. It significantly accelerates UI development while maintaining consistency and accessibility across the interface.

Backend: Node.js with Express

- Node.js was chosen for the backend because it uses JavaScript, which matches the language used in the React frontend. This makes it easier to work on both the frontend and backend. Node.js also works well for building lightweight, fast applications.
- Express was used as the web framework for Node.js. It is simple to set up and makes it easy to define API routes, handle requests and connect with the database.

Database: Supabase

- Supabase provides a hosted PostgreSQL database with built-in authentication, real-time updates and file storage. It simplifies backend development by offering RPCs (Remote Procedure Calls) and row-level security. The use of SQL was essential for this application due to the relationships between employees, managers and organisations, which require relational data modeling. The comprehensive feature set allows for rapid development and secure data management while reducing the need for extensive backend infrastructure setup and maintenance.

Hosting: GitHub Pages and Render

- GitHub Pages is used to host the frontend. Deployment is simple and automatic, with updates pushed directly from the GitHub repository. Additionally, GitHub Pages is free for public projects and requires little maintenance. It also includes version control and built-in HTTPS, keeping the site secure without extra setup.
- The backend API runs on Render, a cloud platform offering fully managed hosting. Render automates deployment from Git repositoriess. It scales resources based on demand, ensuring the app runs smoothly under varying loads. Render also manages server uptime, health monitoring, and automatic HTTPS certificates.