EPI-USE Employee Hierarchy Platform

Overview

This is a cloud-hosted web application built to manage EPI-USE Africa's employee hierarchy. It handles employee data, reporting relationships, and provides different access levels based on user roles.

Live Application: https://epiuse-employee-hierarchy.vercel.app/

What I Built

Employee Management

- Full CRUD - you can add, view, edit, and delete employees
- All the required fields are there: name, surname, birth date, employee number, salary, position, and who they report to
- Made sure employees can't be their own manager (that would be weird)
- CEO doesn't need a manager, which makes sense

Visual Stuff

- Interactive org chart that shows the whole company structure as a tree
- You can click on people to see their details
- Search works across everything - find anyone by any field
- Employee table where you can sort and filter however you want

Gravatar & Pictures

- Pulls profile pictures from Gravatar automatically
- Also added custom picture uploads since that was mentioned as a nice bonus

Hosting & Data

- Everything's live on the cloud (Vercel for the frontend, Railway for backend and database)
- No fake data - everything goes to a real PostgreSQL database
- You can access it right now at the URL above

Documentation

- User guide shows how to actually use the thing
- This README explains how I built it and why I made certain choices

Extra Stuff I Added

Better Security

- Different permission levels - admins can do everything, managers can only mess with their department, employees can just view stuff
- Proper login system with password resets
- Made sure people can't break things by entering bad data

Smarter Manager Assignment

- Managers can assign people to other managers in their department, not just themselves
- The UI shows you which managers you can pick from
- Won't let you assign someone to the wrong department by mistake

Nice-to-Have Features

- Dashboard that shows useful stats about the organization
- Export employee data to CSV or JSON files
- Custom profile picture uploads (the spec mentioned this as optional)
- Org chart that you can actually navigate around, zoom in/out

Technical Stuff

- Used TypeScript everywhere so there's fewer bugs
- Database queries are type-safe with Prisma
- Frontend caches data intelligently so it feels fast
- Works on phones and tablets too

Architecture & Design Decisions

Technology Stack

Frontend: Next.js + React

- Went with Next.js because it handles server-side rendering automatically, which makes pages load faster
- React makes it easy to build reusable components
- TypeScript catches bugs while I'm coding instead of when users are using the app

Backend: Node.js + Express

- Express is simple and lightweight - didn't need anything fancy
- Keeping everything in JavaScript means I don't have to switch between languages
- Built it as a REST API so other systems can easily connect to it later

Database: PostgreSQL + Prisma

- PostgreSQL is great for complex relationships, which you need for employee hierarchies
- Prisma gives me type-safe database queries and handles all the migration stuff
- Used a self-referential relationship so the org chart can be as deep as needed

Authentication: JWT

- JWT tokens don't need server-side sessions, so the app can scale better

- The tokens include role info, so I can check permissions without hitting the database every time

How I Structured Things

Permissions System I set up three user types:

- Admin: Can do anything across the whole company
- Manager: Can manage their department and assign people to other managers in the same department
- Employee: Can view stuff and update their own profile

Database Design For the employee hierarchy, I used a simple approach where each employee has a `managerId` field pointing to their boss. It's not the fanciest way to do it, but it's easy to understand and Prisma handles the recursive queries well enough.

Project Layout Put the frontend and backend in separate folders but kept them in the same repo. This way I can share TypeScript types between them and deploy them separately.

Why I Think This Works

The whole thing is pretty straightforward - no over-engineering. The API doesn't store sessions so it can scale horizontally if needed. TypeScript catches most bugs before they become problems. React Query makes the frontend feel snappy by caching data and updating optimistically.

The permission system is simple but covers what you'd actually need in a real company - admins can do everything, managers can handle their teams, and employees can't break anything.

Database Schema

```
Users Table:
- id, email, password, role, resetToken, timestamps

Employees Table:
- id, firstName, lastName, email, birthDate, employeeNumber
- salary, position, department, managerId (self-reference)
- profilePicture, timestamps
```

The `managerId` creates the hierarchy - each employee can have one manager, and managers can have multiple subordinates.

Key Features Implemented

Employee Management

- Full CRUD operations with role-based restrictions
- Managers can assign employees to other managers within their department (recent addition)
- Prevents invalid operations (like deleting managers with subordinates)

Organizational Visualization

- Interactive org chart using react-d3-tree
- Real-time dashboard with role-specific metrics

Data Export

- CSV/JSON export with permission-based filtering
- Admins see everything, managers see their teams only

Security

- Password reset via email tokens
- Rate limiting to prevent abuse
- Input validation on both frontend and backend

Recent Enhancement: Flexible Manager Assignment

Originally, managers could only assign employees to themselves. I enhanced this so managers can assign employees to other managers within their department.

Backend Changes: Modified the authorization logic to allow intra-department assignments while maintaining security boundaries.

Frontend Changes: Added dropdown selections showing available managers with clear visual indicators.

This gives managers more flexibility in organizing their teams while maintaining proper access controls.

Development Setup

```
# Frontend
cd apps/frontend && npm install && npm run dev

# Backend
cd apps/backend && npm install && npm run dev
```

Environment Variables Needed:

- `DATABASE_URL`: PostgreSQL connection string
- `JWT_SECRET`: Token signing key
- `EMAIL_*`: SMTP configuration for password resets

Deployment

- **Frontend**: Vercel (automatic deployments from main branch)
- **Backend + Database**: Railway (handles both API and PostgreSQL)

This setup gives good performance, automatic scaling, and minimal operational overhead.

For end-user instructions, see the User Guide.