Project Report: Employee & Project Management Platform

1. Project Overview

This project is a **Serverless Full-Stack Application** designed for modern deployment infrastructure (Vercel) and scalable data storage (MongoDB Atlas). It streamlines the management of organizational projects, employees, and their assignments.

Key Capabilities

- **Dashboard Analytics**: Real-time view of project statuses, employee workforce, and assignment gaps.
- **Project Management**: Create, update, and archive projects. Track lifecycle from "Planning" to "Active" to "Archived".
- **Employee Directory**: Manage workforce details and roles.
- **Readiness Checklists**: Track pre-kickoff requirements for projects with granular status tracking.
- **Team Assignments**: Map employees to projects with active/inactive status periods.

---

2. Technology Stack

Frontend (User Interface)

- **Framework**: **Angular 18**
  - Uses **Standalone Components** to minimize module boilerplate.
  - Adopts modern **Signals** for reactive state management (e.g., in `AppComponent`).
  - Retains traditional **RxJS** patterns (Subscriptions) in complex components (`DashboardComponent`).
- **Styling**: **Tailwind CSS** for utility-first styling.
- **UI Components**: Custom components inspired by `shadcn`/modern aesthetics, using gradients, glassmorphism (`backdrop-blur`), and responsive grids.

Backend (API & Data)

- **Architecture**: **Function-as-a-Service (FaaS)** pattern.
  - Single entry point: `api/employee-management/handler.mjs` acts as a central request dispatcher.
  - Designed for Vercel Serverless Functions.
- **Database**: **MongoDB Atlas** (NoSQL).
- **Data Access Layer**:

- **Prisma ORM**: Defines the data schema (`prisma/schema.prisma`) and handles standard CRUD operations.
- **MongoDB Native Driver**: Used for high-performance data seeding and atomic counter operations.

Developer Tools

- **Local API Emulation**: `tools/dev-api-server.mjs` spins up a Node.js server (Port 4310) to mimic Vercel's behavior locally.
- **Data Seeding**: `DATA_SYN/populate_data.py` (Python) generates realistic test data (employees, projects, departments) ensuring all relationships are valid.

---

3. Project Structure Breakdown

Reference Map

```
e:/FSD_FINAL/
  api/                    # Serverless Functions (Backend)
    employee-management/
        handler.mjs       # Main API Dispatcher
  DATA_SYN/               # Data Generation
    populate_data.py      # Python Script for Seeding MongoDB
  prisma/                 # Database Configuration
    schema.prisma         # Data Models & Relations
    seed.ts               # (Legacy) TS Seeder
  src/                    # Frontend Source Code
    app/
        model/            # TS Interfaces & Classes
        pages/            # Application Views
            dashboard/    # Dashboard Component
            employee/     # Employee List
            project*/     # Project Management Pages
        service/          # API Communication
            master.service.ts
  tools/                  # Developer Utilities
    dev-api-server.mjs    # Local Node Server for API
  project_report.md       # This file
```

Key Directories Explained

- **`src/app/pages`**: Contains all the "Screens" of the application. Each folder (e.g., `dashboard`, `login`) is a standalone component module.
- **`src/app/service`**: The "Network Layer". `MasterService` handles all HTTP calls, headers, and error handling.

- `api/`: The "Backend". This folder logic is what Vercel executes when an HTTP request hits `/api/...`.

---

4. Architecture & Data Flow

Request Lifecycle

1. **Client (Angular)**: `MasterService` allows the frontend to communicate with the backend. It effectively abstracts:
   - **API URLs**: Toggles between local dev and production endpoints.
   - **Routing Quirks**: Handles Vercel-specific routing needs (e.g., passing IDs via query params `?id=123` instead of URL paths `/123` for certain retrievals).
2. **Dispatcher**: The `handler.mjs` script receives all requests relative to `/api/employee-management/`.
3. **Controller**: It parses the request method and path to invoke the correct logic (e.g., `GetEmployees`, `CreateProject`).
4. **DB Interaction**: Prisma or Native Driver executes the query against MongoDB.

Data Model (Key Entities)

- **Department**: Organized hierarchically (Parent -> Children).
- **Employee**: The core workforce unit.
- **Project**: The central object of work, containing metadata like `leadByEmpId` (Team Lead), `clientName`, and `readiness` scores.
- **ProjectEmployee**: A many-to-many relationship table that tracks **Active** vs **Inactive** tenure on a project.

---

5. Feature Deep Dive

A. Dashboard Logic

The `DashboardComponent` is "thick" on the client side. Instead of asking the API for "Total Active Projects", it: 1. Fetches **ALL** Projects. 2. Fetches **ALL** Assignments. 3. Calculates totals, active/inactive ratios, and utilization percentages directly in the browser (`calculateProjectStatistics()` method). *Pro*: Reduces backend complexity. *Con*: Will get slower as data grows to thousands of records.

B. Readiness Checklist

Located in `ProjectFormComponent`, this feature allows managers to audit project prep. - **Scoring**: Each item (Scope, Budget, Team) has a point weight.

- **Reactive Status**: Updating an item immediately recalculates the total "Readiness Score" (0-100%) using Angular Signals.

C. Authentication

- **Login**: Simple credential check against the `Employee` collection (checking `password` field).
- **Layouts**: The app swaps between `auth` layout (Login page) and `private` layout (Sidebar + Content) based on the current Route data.

---

6. Recent Bug Fixes & Improvements

We have recently applied three critical fixes to improve the user experience:

A. Dashboard Department Logos

- **Issue**: Department cards displayed duplicate text because logo images were missing/null.
- **Fix**: Implemented a **Frontend Fallback Strategy**. The dashboard now maps standard names (Engineering, HR, etc.) to local SVG assets (`engineering.svg`, `hr.svg`).
- **Commit**: `fix(dashboard): resolve duplicate department logos...`

B. Readiness Checklist Visibility

- **Issue**: Dropdown menus had white text on a white background.
- **Fix**: Added `bg-slate-800` (dark mode background) to the `<option>` elements.
- **Commit**: `fix(ui): contrasting background for readiness checklist...`

C. Dashboard Navigation

- **Issue**: Statistic cards were static.
- **Fix**: Added `[routerLink]` directives to map cards to their respective pages.
- **Commit**: `feat(dashboard): add navigation links...`

---

7. Setup & Deployment Guide

Prerequisites

- Node.js (v18+)
- Python 3.x (for data seeding)

- MongoDB Atlas Connection String

Local Installation

1. **Clone & Install**: `bash git clone <repo-url> cd FSD_FINAL npm install`
2. **Environment Setup**: Create a `.env` file in the root: `env DATABASE_URL="mongodb+srv://<user>:`
3. **Populate Data**: `bash # Generates fresh data for departments, employees, and projects python DATA_SYN/populate_data.py`
4. **Run Development Servers**:

   ```
   # Terminal 1: Frontend (Angular)
   npm start

   # Terminal 2: API (Local Node Server)
   npm run api:dev
   ```

   Access the app at `http://localhost:4200`.

Vercel Deployment

1. Push code to GitHub Main branch.
2. Import project into Vercel.
3. Add `DATABASE_URL` to Vercel Environment Variables.
4. Deploy. Vercel will auto-detect the Angular framework and Serverless Functions.

---

8. Future Roadmap (Recommended)

1. **Server-Side Aggregation**: Create a `/api/dashboard-stats` endpoint to offload the heavy math from the browser.
2. **Real Authentication**: Replace simple password checks with JWT (JSON Web Tokens) or NextAuth for security.
3. **Modernize Angular**: Fully migrate from `*ngIf` to the new `@if` syntax for cleaner templates.