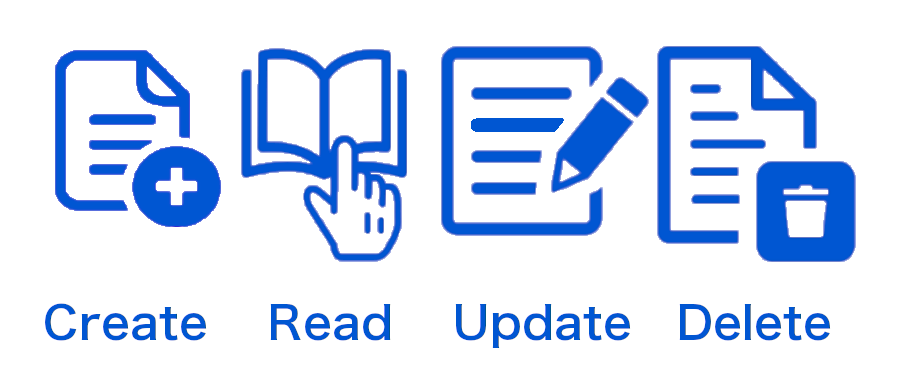
# The Project Scenario

****



## Overview

In this project, you will step into the shoes of an entry-level full-stack developer at Apollonia Dental Practice, helping the team with developing a new employee management web application for the company.

## Project Scenario

The team at Apollonia Dental Practice is trying to set the foundations to digitalize their employee management.In time, they'll want to be able to have digital records of their employees and departments, assign patients to each member of their medical staff, assign projects to each staff member, and more. To do this, they want to begin creating a basic CRUD web app, storing and managing information about their employees and departments, as a foundation to further develop the app upon in the future.

They want you to develop a database and web app reflecting their current organization, staff members and departments. The team will use your development to begin creating a new digital employee management system in order to better manage their dental practice and its workforce.

## Project Objectives

1. Create a database with employees and departments
2. Create a web UI for employees and departments management
3. Perform CRUD (Create, Read, Update, Delete) operations on the employees and departments database using the web UI

## Your Challenge

Your task will be to create a fundamental employee management CRUD web app to store information about employees and departments, and perform CRUD operations on the employee data, using Node.js and MongoDB. To do this, you will create a Node.js server and MongoDB database, create employees and departments collections, write a UI in HTML, CSS and JavaScript, and let users perform CRUD operations on the database via a REST API through the web UI.

In this project, we'll use organizational data from the Apollonia Dental Practice, and the following technologies: HTML, CSS, JavaScript, Node.js, ExpressJS, MongoDB, REST API.

After you perform your development, you will share your outcome, packaging it with Docker.

Chatgpt folder structure

Here’s a clean and modular **folder structure** for a MEAN (MongoDB, Express.js, Angular, Node.js) stack application that is containerized and deployable on **GCP** or **AWS** platforms.

**Folder Structure**

plaintext

Copy code

mean-app/

├── backend/ # Backend (Node.js + Express.js API)

│ ├── src/ # Source code for backend

│ │ ├── controllers/ # Controller logic for API routes

│ │ ├── models/ # MongoDB models (Mongoose schemas)

│ │ ├── routes/ # API route definitions

│ │ ├── middlewares/ # Custom middleware (auth, validation, etc.)

│ │ ├── services/ # Business logic and reusable backend services

│ │ ├── utils/ # Utility functions/helpers

│ │ ├── app.js # Express.js app setup

│ │ └── server.js # Server entry point

│ ├── .env # Environment variables (backend)

│ ├── Dockerfile # Dockerfile for backend container

│ ├── package.json # Backend dependencies

│ └── package-lock.json # Backend lock file

│

├── frontend/ # Frontend (Angular application)

│ ├── src/ # Angular source code

│ │ ├── app/ # Main application module

│ │ │ ├── components/ # Angular components

│ │ │ ├── services/ # Angular services (HTTP, shared logic)

│ │ │ ├── guards/ # Route guards for authentication/authorization

│ │ │ └── models/ # TypeScript models/interfaces

│ │ ├── assets/ # Static assets (images, styles)

│ │ ├── environments/ # Angular environment settings (prod/dev)

│ │ ├── main.ts # Angular entry point

│ │ └── index.html # Main HTML file

│ ├── angular.json # Angular CLI configuration

│ ├── Dockerfile # Dockerfile for frontend container

│ ├── package.json # Frontend dependencies

│ └── package-lock.json # Frontend lock file

│

├── database/ # MongoDB setup (if using Dockerized DB)

│ ├── init.js # Initialization script for MongoDB

│ ├── Dockerfile # Dockerfile for MongoDB container (optional)

│ └── docker-compose.yml # Compose file for MongoDB and other services

│

├── deploy/ # Deployment files/scripts

│ ├── gcp/ # GCP-specific deployment files

│ │ ├── app.yaml # App Engine deployment config

│ │ ├── cloudbuild.yaml # GCP Cloud Build configuration

│ │ └── terraform/ # Terraform scripts for GCP infrastructure

│ ├── aws/ # AWS-specific deployment files

│ │ ├── ecs-task-def.json # ECS task definition for AWS

│ │ ├── eb/ # Elastic Beanstalk deployment config

│ │ └── terraform/ # Terraform scripts for AWS infrastructure

│ └── kubernetes/ # Kubernetes deployment files

│ ├── backend-deployment.yaml # Deployment for backend

│ ├── frontend-deployment.yaml # Deployment for frontend

│ └── mongodb-deployment.yaml # Deployment for MongoDB

│

├── docker-compose.yml # Multi-container setup for local development

├── .dockerignore # Exclude files from Docker builds

├── README.md # Project overview and setup instructions

└── .gitignore # Ignore unnecessary files in Git

**Key Components Explained**

1. **Backend (backend/)**:
   * **Express.js server** to handle APIs and integrate with MongoDB.
   * Uses environment variables (.env) for secure configuration.
   * Modular structure for controllers, routes, and models.
2. **Frontend (frontend/)**:
   * Angular app to provide a user-friendly interface.
   * Organized into components, services, and guards for modularity.
   * Environment configurations (environments/) for production and development.
3. **Database (database/)**:
   * Dockerized MongoDB setup for local and production environments.
   * init.js to seed initial data if needed.
4. **Deployment (deploy/)**:
   * Separate folders for **GCP**, **AWS**, and **Kubernetes** configurations.
   * Includes Terraform scripts for IaC (Infrastructure as Code) on both platforms.
5. **Dockerized Setup**:
   * Each service (backend, frontend, MongoDB) has its own Dockerfile.
   * docker-compose.yml for local multi-container development.
6. **Cross-Platform Deployment**:
   * **GCP**: App Engine, Cloud Build, or Kubernetes Engine.
   * **AWS**: Elastic Beanstalk, ECS, or EKS.
   * **Kubernetes**: YAML files for deployments and services.

**Sample docker-compose.yml**

For local development:

yaml

Copy code

version: '3.8'

services:

backend:

build: ./backend

ports:

- "5000:5000"

environment:

- MONGO\_URI=mongodb://database:27017/mean\_app

depends\_on:

- database

frontend:

build: ./frontend

ports:

- "4200:80"

depends\_on:

- backend

database:

image: mongo:latest

container\_name: mongodb

ports:

- "27017:27017"

volumes:

- ./database:/data/db

**Deployment Notes**

* **AWS ECS**:
  + Use ecs-cli or Terraform to deploy containers to ECS (Elastic Container Service).
* **GCP App Engine**:
  + Include app.yaml for deploying containers using App Engine Flex.
* **Kubernetes**:
  + YAML files (kubernetes/) define backend/frontend deployments, services, and persistent volumes.