Based on the provided assignment, here's a **Node.js + TypeScript** project structure with a **layered architecture** for the **Employee Leave Management System (ELMS)** using **Express** and **SQL Server**.

### **Project Directory Structure**

employee-leave-management/

│── src/

│ ├── controllers/

│ │ ├── employee.controller.ts

│ │ ├── manager.controller.ts

│ │ ├── admin.controller.ts

│ ├── services/

│ │ ├── employee.service.ts

│ │ ├── manager.service.ts

│ │ ├── admin.service.ts

│ ├── repositories/

│ │ ├── employee.repository.ts

│ │ ├── manager.repository.ts

│ │ ├── admin.repository.ts

│ ├── models/

│ │ ├── employee.model.ts

│ │ ├── leave.model.ts

│ ├── routes/

│ │ ├── employee.routes.ts

│ │ ├── manager.routes.ts

│ │ ├── admin.routes.ts

│ ├── database/

│ │ ├── db.config.ts

│ │ ├── db.connection.ts

│ ├── config/

│ │ ├── app.config.ts

│ ├── middleware/

│ │ ├── error.middleware.ts

│ ├── app.ts

│ ├── server.ts

│── postman\_collection.json

│── sql/

│ ├── schema.sql

│── .env

│── .gitignore

│── package.json

│── tsconfig.json

│── README.md

## **1. Installing Dependencies**

Run the following command to set up a **Node.js + TypeScript** project with necessary dependencies:

npm init -y

npm install express dotenv mssql

npm install --save-dev typescript ts-node @types/node @types/express nodemon

## **2. Configuration Files**

### tsconfig.json **(TypeScript Configuration)**

{

"compilerOptions": {

"target": "ES6",

"module": "CommonJS",

"outDir": "./dist",

"rootDir": "./src",

"strict": true,

"esModuleInterop": true

}

}

### .env

PORT=5000

DB\_SERVER=your\_sql\_server

DB\_USER=your\_username

DB\_PASSWORD=your\_password

DB\_NAME=employee\_leave\_management

### src/config/app.config.ts

import dotenv from "dotenv";

dotenv.config();

export default {

port: process.env.PORT || 5000,

db: {

server: process.env.DB\_SERVER || "",

user: process.env.DB\_USER || "",

password: process.env.DB\_PASSWORD || "",

database: process.env.DB\_NAME || "",

},

};

## **3. Database Configuration**

### src/database/db.connection.ts

import sql from "mssql";

import config from "../config/app.config";

const dbConfig = {

user: config.db.user,

password: config.db.password,

server: config.db.server,

database: config.db.database,

options: {

encrypt: false,

enableArithAbort: true,

},

};

const poolPromise = new sql.ConnectionPool(dbConfig)

.connect()

.then(pool => {

console.log("Connected to SQL Server");

return pool;

})

.catch(err => {

console.error("Database Connection Failed!", err);

process.exit(1);

});

export { sql, poolPromise };

## **4. Models**

### src/models/employee.model.ts

export interface Employee {

id: number;

name: string;

email: string;

role: "Employee" | "Manager" | "Admin";

}

### src/models/leave.model.ts

export interface Leave {

id: number;

employee\_id: number;

start\_date: string;

end\_date: string;

leave\_type: "Sick" | "Vacation";

status: "Pending" | "Approved" | "Rejected";

reason: string;

}

## **5. Repositories**

### src/repositories/employee.repository.ts

import { poolPromise, sql } from "../database/db.connection";

export class EmployeeRepository {

async applyLeave(employee\_id: number, start\_date: string, end\_date: string, leave\_type: string, reason: string) {

const pool = await poolPromise;

await pool.request()

.input("employee\_id", sql.Int, employee\_id)

.input("start\_date", sql.Date, start\_date)

.input("end\_date", sql.Date, end\_date)

.input("leave\_type", sql.VarChar, leave\_type)

.input("reason", sql.VarChar, reason)

.query("INSERT INTO LeaveTable (employee\_id, start\_date, end\_date, leave\_type, status, reason) VALUES (@employee\_id, @start\_date, @end\_date, @leave\_type, 'Pending', @reason)");

}

}

## **6. Services**

### src/services/employee.service.ts

import { EmployeeRepository } from "../repositories/employee.repository";

export class EmployeeService {

private employeeRepo = new EmployeeRepository();

async applyLeave(employee\_id: number, start\_date: string, end\_date: string, leave\_type: string, reason: string) {

return await this.employeeRepo.applyLeave(employee\_id, start\_date, end\_date, leave\_type, reason);

}

}

## **7. Controllers**

### src/controllers/employee.controller.ts

import { Request, Response } from "express";

import { EmployeeService } from "../services/employee.service";

const employeeService = new EmployeeService();

export const applyLeave = async (req: Request, res: Response) => {

try {

const { employee\_id, start\_date, end\_date, leave\_type, reason } = req.body;

await employeeService.applyLeave(employee\_id, start\_date, end\_date, leave\_type, reason);

res.status(201).json({ message: "Leave request submitted" });

} catch (error) {

res.status(500).json({ error: error.message });

}

};

## **8. Routes**

### src/routes/employee.routes.ts

import express from "express";

import { applyLeave } from "../controllers/employee.controller";

const router = express.Router();

router.post("/leaves", applyLeave);

export default router;

## **9. App Entry Point**

### src/app.ts

import express from "express";

import employeeRoutes from "./routes/employee.routes";

const app = express();

app.use(express.json());

app.use("/api", employeeRoutes);

export default app;

### src/server.ts

import app from "./app";

import config from "./config/app.config";

const PORT = config.port;

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

## **10. Running the Project**

### **Start the Server**

npx ts-node src/server.ts

### **Run with Nodemon (for development)**

npx nodemon src/server.ts

## **11. SQL Schema (**sql/schema.sql**)**

CREATE TABLE EmployeeTable (

id INT IDENTITY(1,1) PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100),

role VARCHAR(20)

);

CREATE TABLE LeaveTable (

id INT IDENTITY(1,1) PRIMARY KEY,

employee\_id INT FOREIGN KEY REFERENCES EmployeeTable(id),

start\_date DATE,

end\_date DATE,

leave\_type VARCHAR(20),

status VARCHAR(20),

reason TEXT

);

This setup follows the **layered architecture** and **REST API best practices**, ensuring separation of concerns and maintainability. Let me know if you need any modifications!