**Employee Management System Documentation**

This documentation provides comprehensive instructions for setting up and using the Employee Management System. The system includes a .NET Core API backend for managing employee data and an Angular frontend application for user interaction. The documentation is divided into three main sections:

1. **Setup Instructions**
2. **API Endpoints**
3. **Frontend User Guide**

## 1. Setup Instructions

### 1.1 Backend Setup - .NET Core API

1. **Prerequisites**
   * .NET Core
   * SQL Server
   * Visual Studio
2. **Configure the Database**

**SQL Server**: Update the appsettings.json file with your SQL Server connection string

"ConnectionStrings": {

"YourConnectionString": "Server=DESKTOP- RGCAH62;Database=EmployeeDB;Trusted\_Connection=True;TrustServerCertificate=True;"

}

1. **Verify API is Running** Open https://localhost:7035/in your browser or use a tool like Postman to test the API endpoints.

1.2 Frontend Setup - Angular Application

**Check Node Installation**

Open a terminal window and run  
node –v  
  
**Install Angular CLI**

* Run the following command to install Angular CLI globally:

npm install -g @angular/cli

**Generate a New Angular Project**

* Run the following commands in your terminal:

ng version

ng new my-project

cd my-project

ng serve

 **Create a Standalone TypeScript Angular Project**

* Open the Package Manager Console from Tools > NuGet Package Manager > Package Manager Console.
* Install necessary Entity Framework Core packages:

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Design

Install-Package Microsoft.EntityFrameworkCore.Tools

 **Scaffold the Database Context**

* Run the following command to scaffold the database context:

## Scaffold-DbContext "Server=DESKTOP-RGCAH62;Database=EmployeeDB;Trusted\_Connection=True;TrustServerCertificate=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context MyDbContext

## 2. API Endpoints

### 2.1 Base URL

* **Development**: https://localhost:7035/api

### 2.2 Endpoints

#### 2.2.1 Get All Employees

* **Method**: GET
* **Endpoint**: /employees
* **Description**: Retrieve a list of all employees.
* **Response**: 200 OK with a list of employee objects.

#### 2.2.2 Get Employee by ID

* **Method**: GET
* **Endpoint**: /employees/{id}
* **Description**: Retrieve an employee by their unique ID.
* **Response**: 200 OK with employee object; 404 Not Found if employee does not exist.

#### 2.2.3 Create Employee

* **Method**: POST
* **Endpoint**: /employees
* **Description**: Create a new employee.
* **Request Body**: Employee object.
* **Response**: 201 Created with the newly created employee object.

#### 2.2.4 Update Employee

* **Method**: PUT
* **Endpoint**: /employees/{id}
* **Description**: Update an existing employee.
* **Request Body**: Employee object with updated fields.
* **Response**: 200 OK with the updated employee object; 404 Not Found if employee does not exist.

#### 2.2.5 Delete Employee

* **Method**: DELETE
* **Endpoint**: /employees/{id}
* **Description**: Delete an employee by their unique ID.
* **Response**: 204 No Content if deletion is successful; 404 Not Found if employee does not exist.

### 2.3 Error Handling

* **4XX Client Errors**: Include status codes such as 400 Bad Request, 404 Not Found, etc. The response body will contain an error message detailing the issue.
* **5XX Server Errors**: Include status codes such as 500 Internal Server Error. The response body will contain a generic error message.

 **Update Program.cs**

* Add the following code to configure the database context:

// Add services to the container.

builder.Services.AddDbContext<MyDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("YourConnectionString")));

builder.Services.AddCors(options =>

{

options.AddPolicy("Development",

builder =>

{

builder.AllowAnyOrigin()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

 **Update appsettings.json**

* Ensure the connection string is correctly added:

## "ConnectionStrings": {

## "YourConnectionString": "Server=DESKTOP-RGCAH62;Database=EmployeeDB;Trusted\_Connection=True;TrustServerCertificate=True;"

## }3

## . Frontend User Guide

### 3.1 Overview

The Angular application provides a user-friendly interface for managing employee data. Users can perform CRUD operations through the web interface.

### 3.2 Navigation

1. **Home Page**
   * Displays a list of employees.
   * Options to add a new employee or navigate to the edit form of existing employees.
2. **Add Employee Form**
   * **Path**: /add-employee
   * **Description**: Allows users to add a new employee.
   * **Fields**: Name, Email, Mobile Number, Home Address, Photo URL.
   * **Validation**: Required fields, valid email format.
3. **Edit Employee Form**
   * **Path**: /edit-employee/:id
   * **Description**: Allows users to edit existing employee details.
   * **Fields**: Pre-filled with existing employee data; users can modify and save changes.
   * **Validation**: Same as Add Employee Form.
4. **Employee Details**
   * **Path**: /employee/:id
   * **Description**: Displays detailed information about a specific employee.
   * **Fields**: Name, Email, Mobile Number, Home Address, Photo URL.
5. **Back Button**
   * **Path**: Back button on forms navigates users to the employee list page.

### 3.3 Angular Concepts

1. **Observable and HttpClient**
   * Use HttpClient to make API requests (GET, POST, PUT, DELETE).
   * Use Observables to handle asynchronous data streams.
2. **Router-Outlet**
   * Acts as a placeholder for dynamically loaded components based on route configuration.
3. **Single Page Application (SPA)**
   * The Angular app functions as a single-page application where content is dynamically loaded without reloading the entire page.
4. **Index.html**
   * The main entry point of the Angular application.
5. **Routes in app.module.ts**
   * Configure routes for navigation between different views.
6. **Service Layer**
   * Services handle HTTP requests and data manipulation. Example methods include get(), post(), put(), and delete().
7. **Component Selectors**
   * Define the HTML tags for components. Example: <app-add-employee>.
8. **Interfaces**
   * Define data structures to match the API data models.
9. **ngOnInit**
   * Lifecycle hook called when the component is initialized or refreshed.
10. **Imports**
    * Import data or services from other modules or components.
11. **Error Handling**
    * Catch and handle errors in the subscribe() method of Observables.

## 4. Additional Information

### 4.1 Project Structure

* **Backend**:
  + Controllers: Handle HTTP requests.
  + Services: Contain business logic.
  + Models: Define the data structure.
  + Data Access Layer: Interacts with the database.
* **Frontend**:
  + Components: Angular components for different views and forms.
  + Services: Angular services for API interactions.
  + Routing: Configurations for navigation between views.

### 4.2 Contact Information

* **Project Lead**: [Nsreen Nedal Kamel Al-Daraghmeh]
* **Email**: [nsreendaraghmeh@yahoo.com]
* **GoogleDriveLink**: [https://drive.google.com/drive/folders/169AkxZkAl\_XJB-ZiONrBDrrhPBVapwrm?usp=drive\_link]