Employee Management Project Documentation

Table of Contents

Introduction

This is a simple FastAPI project for employee management. The project allows you to perform CRUD operations on employee records, storing data in JSON format.

Prerequisites:

Make sure you have the following installed on your system:

- fastapi==0.103.0
- uvicorn==0.22.0
- requests==2.31.0
- python-multipart==0.0.6
- python-dotenv==1.0.0

Installation:

1. Download the requirements.txt file.
2. Open a terminal window.
3. Navigate to the directory containing the requirements.txt file.

Run the following command:

```
pip3.9 install -r requirements.txt
```

This command will install all the necessary dependencies listed in the requirements.txt file.

Manual Installation

1. Install Python:
Make sure you have Python installed on your system. You can download it from the official Python website: https://www.python.org/downloads/.
Set environment variables: https://docs.python.org/3/using/windows.html

2. Install FastAPI:

- Open a terminal window.
- Run the following command:

```
pip install fastapi
```

3. Install Uvicorn:

- Open a terminal window.
- Run the following command:

```
pip install uvicorn
```

4. Install Pydantic:

- Open a terminal window.
- Run the following command:

```
pip install pydentic
```

Project Structure

```
employee_management/
|
|── routes
|    |──__pycache__
|    |──api.py
|
├── src/
|    ├── schema/
|    |    ├── __pycache__
|    |    ├── employee_schema.py
|    |
|    ├── endpoints/
|    |    ├── __pycache__
|    |    ├──employe_managemant.py
|    |
|    ├── response/
|    |    ├── __pycache__
|    |    ├── employee_response.json
|
├── main.py              # Main FastAPI application
├── requirements.txt     # Dependency specifications
|──__pycache__
```

Note: The Folder **__pycache__** is automatically created the first time a module or script is imported.It’s a valuable feature of Python that improves program performance and efficiency. It's essential for production environments where program startup time needs to be minimized.

The project is organized as follows:

- `employee_management/`: Main project directory.
- `src/`:
  - `schema/`: Stores data models (e.g., `employee_schema.py`).
  - `endpoints/`: Contains logic for CRUD operations (e.g., `employee_management.py`).
  - `response/`: Defines response models (e.g., `employee_response.json`).
- `main.py`: Main entry point for the FastAPI application.
- `requirements.txt`: Lists required dependencies.
- `__pycache__`: Automatically generated cached bytecode for improved performance.
-

Running the Project:

1. Open project folder (employee_management) in VS code
2. Open Terminal
3. Navigate to the directory containing the file main.py.
4. Execute the command to Run the FastAPI application with Uvicorn:
   ```
   uvicorn main:app
   ```

NOTE: Visit http://127.0.0.1:8000/docs to access the Swagger documentation and test the API endpoints.
API Endpoints

OR

Optional: You can specify the host and port using the `--host` and `--port` options. For example, to run the application on all network interfaces and port 9004, use:

```
uvicorn main:app --host 0.0.0.0 --port 9004
```

Note: The Swagger documentation will be available at http://[host]:[port]/docs.

Create Employee

Endpoint: `POST /employees/`

Request Body:
*Name (str)
*email (str)
*mobile_number (int)
*department (str)

Response Body:
*id (str)
*Name (str)
*email (str)
*mobile_number (int)
*department (str)

Read Employee

Endpoint: `GET /employees/{employee_id}`

Query Parameter:
*employee_id (int)

Response Body:
*id (int)
*Name (str)
*email (str)
*mobile_number (str)
*department (str)

Update Employee

Endpoint: `PATCH /employees/{employee_id}`

Query Parameter:
*employee_id (int)

Response Body:
*id (int)
*Name (str)
*email (str)
*mobile_number (str)
*department (str)

Delete Employee

Endpoint: `DELETE /employees/{employee_id}`

Query Parameter:
*employee_id (int)

Response Body:
*id (int)
*Name (str)
*email (str)
*mobile_number (str)
*department (str)

List Employees

Endpoint: `GET /find_employees/`

Query Parameter:
Name (str)
department (str, Enum)
offse(int)
limit(int)

Response Body:
List of EmployeeResponse models
*id (int)
*Name (str)
*email (str)
*mobile_number (str)
*department (str)

Data Models

- Employee Model (schema/employee_schema.py): Defines the structure of employee data for internal processing.
- Employee Response Model (response/employee_response.json): Defines the format of data returned by the API.

Employee Model

```
json

{ "Name": "John Doe",

 "email": "john.doe@example.com",

 "mobile_number": "1234567890",

 "department": "Engineering"

}
```

Employee Response Model

```
json

{
"id": 1,
"Name": "John Doe",
"email": "john.doe@example.com",
"mobile_number": "1234567890",
"department": "Engineering"
}
```

STEPS FOR TESTING APIS:

Step 1: Preparation of test cases in spreadsheet.

- Unit level test cases
- Scenario based test cases.

Step 2: Running those test cases in PostMan Tool.

- Creating a workspace for team having common project
- Creating collection of Set of related API's
- Creating a folder having **POST, GET, PATCH, DELETE**
- Testing those each API with positive and negative Scenarios

Step 3: Identify which test cases **passed** and **failed** and update the spreadsheet accordingly.