Task Manager

* Project Documentation
  + Project Directory Structure:

D:\Task-manager

└───code

├───app

│ ├───config

│ ├───helpers

│ ├───libs

│ ├───middleware

│ ├───models

│ ├───repository

│ ├───routes

│ └───services

├───env

└───\_\_pycache\_\_

* + **app**
  + This is the main application directory that typically contains all the code related to the application's core functionality.
  + **config**: This folder usually holds configuration files for the application, such as settings for databases, environment variables, and any other configurations necessary for the application to run.
  + **helpers**: Contains utility functions or classes that provide common functionalities used across the application. This could include data processing, formatting functions, etc.
  + **libs**: This folder is often used for third-party libraries or custom modules that extend the functionality of the application.
  + **middleware**: Contains middleware components that process requests and responses in the application. Middleware is often used for tasks like authentication, logging, or modifying requests and responses.
  + **models**: This directory holds the data models for the application. Models define the structure of the data and the relationships between different data entities, often corresponding to database tables.
  + **repository**: This folder typically contains code that handles data access and manipulation, acting as an abstraction layer between the application and the database.
  + **routes**: Contains route definitions for the application, usually mapping URLs to specific functions or classes that handle the incoming requests.
  + **services**: This directory usually contains business logic or service classes that encapsulate complex operations and workflows, interacting with models and repositories.
  + **env**
  + This folder contains the virtual environment for the project. It holds the Python interpreter and installed packages specific to this project, allowing for isolated dependencies.
  + **pycache**
  + This directory is automatically created by Python to store compiled bytecode of modules. It speeds up loading times for modules by caching the compiled versions of Python files.
  + **Summary**

This project structure is organized and modular, following common conventions in Python web applications (especially those built with frameworks like FastAPI,Django or Flask). Each folder has a specific purpose, making the codebase easier to navigate and maintain. This organization supports scalability, as new features or components can be added without disrupting the existing code structure.

**Instructions to run code locally**

**D:**

**cd Task-manager**

**cd code**

**.\env\Scripts\activate**

**cd app**

**pip install -r requirements.txt # (optional if already installed)**

**uvicorn main:app –reload**

"""

Note: Temporary Authorization Mechanism

---------------------------------------

For the time being, the API uses a simple authorization mechanism where any value can be passed in the `Authorization` header.

The header is validated using the `api\_key\_auth` function, which checks if the `Authorization` token is present.

This is a placeholder implementation and is subject to change as a more secure and robust authorization process will be implemented in the future.

Example:

APIKey = APIKeyHeader(name='Authorization')

Usage:

def api\_key\_auth(x\_api\_key: str = Depends(APIKey)):

if not x\_api\_key: # Check if x\_api\_key is empty or None

response = {

'status': 'error',

'error\_code': 100,

'message': "Unauthorized Access, Invalid Authorization token."

}

return JSONResponse(response, status\_code=status.HTTP\_401\_UNAUTHORIZED)

return True

"""