AI-Powered Task Manager with Python

This project is designed for showcasing expertise in Python, AI integration, REST API development, concurrency, and design patterns. It features AI-based task prioritization using OpenAI's API, Flask-based RESTful API, SQLite database management, and multithreading for efficient execution.

Key Features:

- ✔ AI-Powered Task Prioritization using OpenAI GPT-4
- ✔ REST API for CRUD operations (Flask-based)
- ✔ Multithreading for parallel processing
- ✔ Singleton Pattern for database management
- ✔ Factory Pattern for structured task creation
- ✔ Logging and Unit Testing for maintainability

Full Python Code:

```
import threading
import time
import sqlite3
import openai
import logging
import unittest
from flask import Flask, request, jsonify
from queue import PriorityQueue
from abc import ABC, abstractmethod

app = Flask(__name__)
logging.basicConfig(filename='task_manager.log', level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s')

class Database:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(Database, cls).__new__(cls)
            cls._instance.init_db()
        return cls._instance
```

```
    def init_db(self):
        conn = sqlite3.connect("tasks.db")
        cursor = conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS tasks (
                          id INTEGER PRIMARY KEY AUTOINCREMENT,
                          description TEXT,
                          priority INTEGER,
                          status TEXT)''')
        conn.commit()
        conn.close()

    def execute_query(self, query, params=()):
        conn = sqlite3.connect("tasks.db")
        cursor = conn.cursor()
        cursor.execute(query, params)
        conn.commit()
        conn.close()

openai.api_key = "your_api_key_here"

def get_task_priority(description):
    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[{"role": "system", "content": "Classify task priority as High (3), Medium
(2), or Low (1)."},
                  {"role": "user", "content": description}]
    )
    priority_map = {"High": 3, "Medium": 2, "Low": 1}
    return priority_map.get(response["choices"][0]["message"]["content"].strip(), 2)

class TaskFactory(ABC):
    @abstractmethod
    def create_task(self, description):
        pass

class ConcreteTaskFactory(TaskFactory):
    def create_task(self, description):
        priority = get_task_priority(description)
        return {"description": description, "priority": priority, "status": "Pending"}
```

```
task_queue = PriorityQueue()

class TaskManager:
    def __init__(self):
        self.db = Database()

    def add_task(self, description):
        factory = ConcreteTaskFactory()
        task = factory.create_task(description)
        task_queue.put((-task["priority"], task))
        self.store_task(task)
        return task

    def store_task(self, task):
        self.db.execute_query("INSERT INTO tasks (description, priority, status) VALUES
(?, ?, ?)",
                              (task["description"], task["priority"], task["status"]))

    def process_tasks(self):
        while not task_queue.empty():
            _, task = task_queue.get()
            logging.info(f"Processing: {task['description']} (Priority: {task['priority']})")
            time.sleep(2)
            self.update_task_status(task, "Completed")

    def update_task_status(self, task, status):
        self.db.execute_query("UPDATE tasks SET status=? WHERE description=?",
(status, task["description"]))

task_manager = TaskManager()

@app.route('/add_task', methods=['POST'])
def add_task():
    data = request.get_json()
    description = data.get('description', '')
    task = task_manager.add_task(description)
    return jsonify(task)

@app.route('/process_tasks', methods=['GET'])
def process_tasks():
```

```
    threading.Thread(target=task_manager.process_tasks).start()
    return jsonify({"message": "Task processing started."})

@app.route('/tasks', methods=['GET'])
def view_tasks():
    conn = sqlite3.connect("tasks.db")
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM tasks")
    tasks = cursor.fetchall()
    conn.close()
    return jsonify(tasks)

class TestTaskManager(unittest.TestCase):
    def setUp(self):
        self.manager = TaskManager()

    def test_add_task(self):
        task = self.manager.add_task("Write a Python script")
        self.assertEqual(task["status"], "Pending")

    def test_task_priority(self):
        high_priority_task = get_task_priority("Fix urgent security vulnerability")
        low_priority_task = get_task_priority("Watch tutorial on Python basics")
        self.assertGreater(high_priority_task, low_priority_task)

if __name__ == '__main__':
    unittest.main(exit=False)
    app.run(debug=True)
```

How to Use:

- ➡ Install dependencies using `pip install flask openai sqlite3`
- ➡ Run the API Server using `python task_manager.py`
- ➡ Test API Endpoints using Curl or Postman