Project Title: Task Manager with REST API

Description: Create a task management web application with a REST API using Django. The application should allow multiple users to create, view, update, and delete tasks. Utilize Django templates for rendering views, PostgreSQL for the database, and Django ORM for managing database relations. Additionally, use virtual environments, environment variables, and Git for proper development practices.

Requirements:

1. Set up the Django project:
    - Create a new Django project named "task_manager", or any name you prefer.
    - Set up a virtual environment for the project.
    - Install the required dependencies (Django, djangorestframework, psycopg2, etc.).
    - Set up the PostgreSQL database configuration.
2. Create a Django app:
    - Create a new Django app named "tasks".
3. Features/Functionalities:
    - Need to have User Authentication (Registration, Login).
    - Nice to have a password reset option (Optional).
    - After successful login, the home page should show the tasks list.
    - The Task should include the following properties:
        - A title, a description, and a due date.
        - Multiple photos add/delete options.
        - Priority (Low, medium, high)
        - Option to mark as complete.
        - Store date time of Creation.
        - Store date time of last Update (Optional).
    - On the list page, tasks should be **searched by title**, and filtered by:
        - Creation date, Due date, Priority, Whether to complete or not.
    - A details page for a task where all info is shown with photos.
    - Option to update the task with all the fields including, priority, due date, etc.
    - Task deletion with confirmation.
    - Make any additional fields/models if you need them.
4. Set up the database relations:
    - Define appropriate relations between models (e.g., ForeignKey, ManyToManyField, or others) to represent any other entities required for the project (e.g., User).
5. Include Admin in models:
    - Add all models in Admin. Create proper functionalities to CRUD all models in Admin.
    - Show **only the necessary fields** in the list for each model (Necessity depends on you).
    - **Sort the tasks by Priority by default.**
6. Create Django templates:
    - **Create a base template** sharing common elements and extend it in all other templates.
    - Create templates for task list, task creation, task details, task update, and task deletion.
    - Utilize Django template tags and filters as needed for rendering dynamic content.
    - Ensure the templates are responsive and visually appealing. **Free to use CSS frameworks and libraries.**
7. Implement Django views and URL patterns:
    - Create **Class Based View**s to display the task list, task creation, task details, task update, and task deletion.
    - Define URL patterns for each view mentioned in the previous step.

    - Map the URLs to their respective views.
8. Implement the functionality:
    - Retrieve tasks from the database and display them on the task list view.
    - Add a form on the task creation and task update views to capture task details.
    - Implement form validation and handle errors appropriately.
    - Implement logic for creating, updating, and deleting tasks database using Django ORM.
9. Create the REST API views:
    - Create API views to list all tasks, retrieve a single task, create a new task, update an existing task, and delete a task.
    - Implement appropriate serializers for converting data to and from JSON format.
    - Ensure the API views handle appropriate HTTP methods (GET, POST, PUT/PATCH, DELETE). Validate input data and handle errors appropriately.

Work procedure (Very important to follow):

1. Use Git for version control:
    a. Initialize an empty Git repository, and have an initial commit with a README.md
    b. Update .gitignore to ignore IDE files, variable files, bytecodes, or others.
    c. Commit your code regularly as you progress through the project.
2. Use environment variables for sensitive settings:
    a. Create a separate file for environment variables (e.g., `.env` or `.env.example`).
    b. Include settings such as database credentials, API keys, or any other sensitive information in the environment variable file.
    c. Configure Django to read these environment variables for the project's settings.
3. Documentation:
    a. Populate the README file that explains how to set up and run the project.
    b. Include instructions on how to set up the environment variables and their usage.
    c. Document the API endpoints and their usage in the README file.
4. Finally:
    a. Create at least three users and a few tasks for each user having tasks with various priorities and images. Use realistic data for all fields.
    b. Export all data from the database to JSON fixtures and include them in the git commit.

Submission Guidelines (Must follow, otherwise won't be judged):

- After doing all of the above in the last section, create a screen record video, upload it to Google Drive, and share the link with us. The video should include:
    - All commits in GitHub, then clone from GitHub and set up everything in a new directory.
    - Show the functionalities one by one you achieved, specified in the **Requirements.**
- Provide a link to a Git repository containing the Django project.

Note: This project aims to assess the developer's understanding of Python, Django, Django templates, database relations and Django ORM, Django REST API implementation, PostgreSQL integration, virtual environments, environment variables for sensitive configuration, and version control with Git. You can expand on it or add additional features if desired. **Also, it's nice to leave all the API things and some small parts, if you are not capable/times up. Taking additional time will not be a penalty.**

Best of luck!