Use to ChatGPT / AI / External source is totally discouraged, if use of any such medium is observed such test submission will be considered as invalid.

Python Developer Assignment: Creating APIs with Django
Objective:

Evaluate the skills of a Python developer by creating a Django application that includes several APIs with specific functionalities and form validation requirements.

Project Requirements:

1.GET API to Greet a User:

1. An API that accepts a GET parameter called name and returns a greeting message in the format "Hello <name>".

2.POST API to Process HTML Form:

1. An API that accepts a POST HTML form with fields for name, age, and salary.
2. The API should return a sentence in the format: "<name> is <age> years old and earns <salary> every month".
3. Implement form validation to ensure:
    1. name is not empty and is a string.
    2. age is a positive integer.
    3. salary is a positive number.

3.POST API to Fetch and Display Jokes:

1. An API that accepts a POST HTML form with a count field.
2. The API should call the external API https://api.api-ninjas.com/v1/jokes?limit= to fetch the specified number of jokes.
3. Display the fetched jokes in an HTML format, centered horizontally on the page.

Steps to Complete the Assignment:
1. Set Up Django Environment

- **Install Django:**
    - Use pip to install Django.
    - Create a new Django project and a new application within it.

2. Create the APIs
a. GET API to Greet a User
Create View:

- Define a view that accepts a name parameter from a GET request.
- Return a JSON response with a greeting message.
-

URL Configuration:

- Map a URL to the view in the urls.py file.

b. POST API to Process HTML Form

Create HTML Form:

- Design an HTML form with fields for name, age, and salary.

Create View:

- Define a view that processes the form data from a POST request.
- Validate the form data ensuring:
  - name is a non-empty string.
  - age is a positive integer.
  - salary is a positive number.
- Return a response with a sentence in the required format.

URL Configuration:

- Map a URL to the view in the urls.py file.

c. POST API to Fetch and Display Jokes

Create HTML Form:

- Design an HTML form with a count field.

Create View:

- Define a view that processes the form data from a POST request.
- Fetch jokes from the external API using the count parameter.

- Render the jokes in an HTML template, centered horizontally on the page.

URL Configuration:

- Map a URL to the view in the urls.py file.

3. Implement Form Validation

- **Create Django Form Classes:**
  - Define form classes in forms.py to handle and validate the form data for both the POST APIs.
  - Use Django's built-in validators and custom validation methods as required.

4. Testing

- **Test APIs:**
  - Test the GET API by passing different name values and verifying the responses.
  - Test the POST API by submitting the HTML forms with various inputs and verifying the responses and validation.

5. Documentation

- **Provide Clear Instructions:**
  - Document the steps to set up and run the Django application.
  - Include instructions on how to interact with each API and test their functionalities.

Deliverables:

1.Django Application:

- Django project and application code containing the APIs and HTML templates.

2.Documentation:

- Detailed documentation on setting up, running, and testing the Django application.
- Explanation of the form validation logic and API integration.

Tips:

- **Follow Django Best Practices:**
  - Use Django's class-based views and forms for a clean and maintainable codebase.
  - Implement error handling for API requests and form validations.
- **Use Third-Party Libraries:**
  - Use the requests library to call the external jokes API.
- **Focus on User Experience:**
  - Design intuitive and user-friendly HTML forms.
  - Ensure the jokes are displayed in a visually appealing manner.