Building a Flask App: A Step-by-Step Guide

1. Setting Up the Environment

Install Flask

Flask is a lightweight web framework for Python. To use it, ensure you have Python installed and then install Flask using pip:

```
pip install flask
```

This will install Flask and its dependencies, enabling you to create web applications.

2. Project Structure

A well-structured project makes development and maintenance easier. Organize your project as follows:

```
/flask_app
    |-- app.py  # Main application file containing routes and logic
    |-- templates/
    |   |-- index.html  # HTML templates for rendering web pages
    |-- static/  # Contains static files like CSS, JavaScript, and images
    |-- grades.json  # JSON file used for storing student grades
```

This structure keeps the project modular and manageable.

3. Writing the Flask App

Import Necessary Modules

Flask provides various modules that help in handling web requests and responses. The following are essential for our app:

```
from flask import Flask, jsonify, request, render_template
import json
import os
```

- `Flask`: Initializes and runs the web application.

- `jsonify`: Converts Python dictionaries into JSON responses.
- `request`: Handles incoming HTTP requests.
- `render_template`: Renders HTML templates.
- `json` and `os`: Used for file handling and managing the JSON storage.

Initialize the App

```
app = Flask(__name__)
```

This initializes the Flask application, which will handle incoming HTTP requests.

Data Handling Functions

To manage student grades, we need functions that can read and write to a JSON file:

```
GRADES_FILE = 'grades.json'
```

Load Grades

```
def load_grades():
    if not os.path.exists(GRADES_FILE):
        with open(GRADES_FILE, 'w') as file:
            json.dump({}, file)

    with open(GRADES_FILE, 'r') as file:
        return json.load(file)
```

This function:

- Checks if the file exists.
- If not, creates an empty JSON file.
- Reads and loads the data into a dictionary.

Save Grades

```
def save_grades(grades):
    with open(GRADES_FILE, 'w') as file:
        json.dump(grades, file, indent=4)
```

This function saves the current student grades into `grades.json` in a readable format.

4. Creating Routes

Flask uses **routes** to define the various endpoints of the application.

Home Page Route

The homepage serves an HTML template.

```
@app.route('/')
def index():
    return render_template('index.html')
```

This function renders `index.html` when a user visits the root URL (`/`).

Retrieve All Grades

To fetch all stored grades, we create a GET request route:

```
@app.route('/grades', methods=['GET'])
def get_grades():
    grades = load_grades()
    return jsonify(grades)
```

This function reads grades from the JSON file and returns them as a JSON response.

Retrieve a Specific Student's Grade

If we need to fetch a particular student's grade, we use:

```
@app.route('/grades/<string:name>', methods=['GET'])
def get_grade(name):
    grades = load_grades()
    grade = grades.get(name)
    if grade is not None:
        return jsonify({name: grade})
    else:
        return jsonify({"error": "Student not found"}), 404
```

This function:

- Retrieves the grade for a given student.
- Returns an error if the student is not found.

Add a New Grade

A **POST** request allows adding a new student and their grade:

```
@app.route('/grades', methods=['POST'])
def add_grade():
    grades = load_grades()
    data = request.json
    name = data.get('name')
    grade = int(data.get('grade'))
```

- The `request.json` retrieves incoming JSON data.
- The `name` and `grade` fields are extracted.

To ensure data validity:

```
    if len(name) > 32 or any(char in name for char in "<>'\""):
        return jsonify({"error": "Student name invalid"}), 400

    if name in grades:
        return jsonify({"error": "Student already exists"}), 400
```

- Checks for invalid characters.
- Prevents duplicate student names.

Finally, we store the new student's grade:

```
    grades[name] = grade
    save_grades(grades)
    return jsonify(grades), 201
```

Edit a Grade

A **PUT** request updates an existing student's grade:

```
@app.route('/grades/<string:name>', methods=['PUT'])
def edit_grade(name):
    grades = load_grades()
```

```
    if name not in grades:
        return jsonify({"error": "Student not found"}), 404
```

If the student exists, we update their grade:

```
    data = request.json
    new_grade = data.get('grade')
    grades[name] = new_grade
    save_grades(grades)
    return jsonify(grades)
```

Delete a Grade

A **DELETE** request removes a student's grade:

```
@app.route('/grades/<string:name>', methods=['DELETE'])
def delete_grade(name):
    grades = load_grades()
    if name not in grades:
        return jsonify({"error": "Student not found"}), 404

    deleted_grade = grades.pop(name)
    save_grades(grades)
    return jsonify(grades)
```

This:

- Checks if the student exists.
- Removes their entry from the JSON file.

5. Running the Flask App

To start the server, run:

```
if __name__ == '__main__':
    app.run(debug=True)
```

This will start a local development server.

6. Testing the API

Use `curl` or Postman to test the API endpoints:

- **Get all grades**:

```
    curl -X GET http://127.0.0.1:5000/grades
    ```
- **Add a grade**:
-
```bash
    curl -X POST http://127.0.0.1:5000/grades -H "Content-Type:
application/json" -d '{"name": "John", "grade": 85}'
    ```

7. Deployment

For production, use `gunicorn`:

```bash
pip install gunicorn
```

Run:

```
gunicorn -w 4 app:app
```

Deploy to AWS, Heroku, or Render for cloud hosting.

Conclusion

This guide provides a structured approach to building a Flask application for managing student grades. Future improvements could include authentication, database integration, and UI enhancements.