RESTful API Assignment

1. What is a RESTful API?

A RESTful API (Representational State Transfer) is a web service that follows REST principles, using standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources, which are identified by URLs.

2. Explain the concept of API specification

An API specification defines how an API should behave, including endpoints, request/response formats, parameters, status codes, and authentication. It helps developers understand and use the API correctly. Examples: OpenAPI, Swagger.

3. What is Flask, and why is it popular for building APIs?

Flask is a lightweight Python web framework that is easy to use, flexible, and minimal. It’s popular for building APIs because of its simplicity, built-in development server, and support for extensions like Flask-RESTful.

4. What is routing in Flask?

Routing in Flask maps URLs to specific Python functions. It determines what logic is executed when a user accesses a specific endpoint.

5. How do you create a simple Flask application?

Python
```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return 'Hello, World!'

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

6. What are HTTP methods used in RESTful APIs?

- `GET`: Retrieve data
- `POST`: Create new data
- `PUT`: Update existing data
- `DELETE`: Delete data
- `PATCH`: Partially update data

7. What is the purpose of the @app.route() decorator in Flask?

It defines the route (URL) and binds it to a specific view function that handles the request.

8. What is the difference between GET and POST HTTP methods?

- `GET`: Requests data from the server, parameters are in the URL.
- `POST`: Sends data to the server to create something, parameters are in the request body.

9. How do you handle errors in Flask APIs?

Use error handlers:

```python
@app.errorhandler(404)
def not_found(error):
    return {'error': 'Not found'}, 404
```

10. How do you connect Flask to a SQL database?

Using Flask-SQLAlchemy:

```python
from flask_sqlalchemy import SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///data.db'
db = SQLAlchemy(app)
```

11. What is the role of Flask-SQLAlchemy?

It is an ORM extension for Flask that simplifies working with databases using Python classes instead of raw SQL.

12. What are Flask blueprints, and how are they useful?

Blueprints are a way to organize Flask apps into smaller, reusable modules. They help with modular development.

13. What is the purpose of Flask's `request` object?

The `request` object holds data sent by the client in an HTTP request, such as form data, JSON, headers, etc.

14. How do you create a RESTful API endpoint using Flask?

```python
@app.route('/api/data', methods=['GET'])
def get_data():
    return {'data': 'Sample'}
```

15. What is the purpose of Flask's `jsonify()` function?

`jsonify()` converts Python dictionaries/lists to JSON format and sets the correct MIME type (`application/json`).

---

16. Explain Flask’s `url_for()` function

`url_for()` generates a URL for a given view function. It's useful for dynamic routing:

```python
url_for('home')
```

17. How does Flask handle static files (CSS, JavaScript, etc.)?

Flask serves static files from the `/static` directory. You can link them like this:

```html
<link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
```

18. What is an API specification, and how does it help in building a Flask API?

It defines how the API behaves. It ensures consistency, helps front-end and back-end developers collaborate, and is often used to auto-generate documentation.

19. What are HTTP status codes, and why are they important in a Flask API?

Status codes inform the client about the result of the request:

- `200 OK`: Success
- `201 Created`: Resource created
- `400 Bad Request`: Invalid input
- `404 Not Found`: Resource not found
- `500 Internal Server Error`: Server issue

They help in debugging and proper client behavior.

20. How do you handle POST requests in Flask?

```python
from flask import request

@app.route('/api/data', methods=['POST'])
def post_data():
    data = request.json
    return {'received': data}, 201
```

21. How would you secure a Flask API?

- Use HTTPS
- Input validation and sanitization
- Authentication (e.g., JWT, OAuth)
- Rate limiting
- API keys
- Use Flask extensions like Flask-JWT or Flask-Login

22. What is the significance of the Flask-RESTful extension?

It simplifies building REST APIs in Flask by providing classes for resources, better routing, and built-in support for input parsing and error handling.

23. What is the role of Flask's `session` object?

`session` stores user-specific data across requests, like login state. It uses cookies and is signed to prevent tampering.