REST API Study Notes

What is an API?

An API (Application Programming Interface) allows two pieces of software to communicate. This is essential because each software system can be different, and a standardized way of communication ensures compatibility.

Why Use JSON?

JSON (JavaScript Object Notation) is commonly used in APIs as a lightweight data-interchange format. It is easy for humans to read and write and easy for machines to parse and generate. JSON uses name/value pairs and ordered lists, making it suitable for various programming languages.

How REST Works

REST (Representational State Transfer) is a set of rules that developers follow when creating APIs. It uses standard web protocols, primarily HTTP, and focuses on stateless communication, meaning each request from a client to server must contain all the information the server needs to fulfill that request.

API Endpoints

Endpoints are specific paths on a server where data can be accessed or modified. For example, `/drinks/<id>` might be an endpoint to get information about a particular drink, where `<id>` is the identifier for the drink.

Why the Complexity?

- **Security:** APIs can implement security measures like authentication to ensure only authorized users can access data.
- **Versatility:** A single backend can serve multiple frontends (web, mobile, desktop) consistently.
- **Modularity:** The backend can be updated or replaced without affecting the frontend, as long as the API remains the same.
- **Interoperability:** Public APIs allow different developers to create diverse applications using the same data.

HTTP Methods in REST APIs

- **GET:** Retrieve data from the server (e.g., viewing a webpage).
- **POST:** Send data to the server to create or update resources.
- **PUT:** Update or replace existing resources. It is idempotent, meaning making the same request multiple times will have the same effect.
- **DELETE:** Remove a resource from the server.
- **PATCH:** Update partial resources.

Consuming an API

To use an API, you can make requests using URLs and methods like GET or POST. For instance, accessing `https://api.example.com/items` with a GET request retrieves data, while a POST request to the same endpoint might add a new item.

Example: Using a REST API with Python

```
import requests
response = requests.get("https://api.example.com/items")
data = response.json()
for item in data:
    print(item['name'])
```

This script makes a GET request to an API, converts the response to JSON, and prints out each item's name.

Building a REST API with Flask

Flask is a Python web framework that makes it easy to create APIs. Here’s a simple example:

```
from flask import Flask, request, jsonify
app = Flask(__name__)

@app.route('/items', methods=['GET'])
def get_items():
    items = [{"name": "Item1"}, {"name": "Item2"}]
    return jsonify(items)

@app.route('/items', methods=['POST'])
def add_item():
    new_item = request.get_json()
    # Logic to add item to database goes here
    return jsonify(new_item), 201

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

This code sets up a basic Flask application with endpoints to get and add items using GET and POST methods.

Conclusion

REST APIs are powerful tools that enable different software systems to communicate efficiently and securely over the internet. By understanding the basics of HTTP methods, JSON, and how to consume and create APIs, developers can build robust applications that interact seamlessly with various data sources.