Assignment Submission: Demystifying APIs - The Digital Connectors

Author: Rana Sujeet Kumar
Role: Technical Trainee Applicant
Date: 21/08/2025

---

1. Introduction: What is an API? (The "Waitress" Analogy)

An **API (Application Programming Interface)** is a set of defined rules and protocols that allows different software applications to communicate with each other. It is a contract between a service provider and a service user, specifying how to request data, how to perform actions, and what data will be returned.

Real-Life Analogy: The Restaurant Waitress
Imagine you are a customer (User Application) sitting at a table in a restaurant. The kitchen (Server Application) is where the food (data/functionality) is prepared.

- You cannot go into the kitchen yourself to place your order or get your food.
- Instead, you interact with the waitress (the **API**).
- The waitress provides you with a **menu (API documentation)**, which lists what you can order, how to order it (e.g., "Burger, no onions"), and what you can expect to receive.
- You give your order (API Request) to the waitress.
- The waitress takes your request to the kitchen.
- The kitchen prepares the food and gives it to the waitress.
- The waitress then brings your food (API Response) back to you.

The API, like the waitress, abstracts the complexity of the kitchen's inner workings. You don't need to know how the chef cooks the steak; you just need to know how to order it correctly using the menu.

---

2. Different Types and Purposes of APIs

APIs can be categorized based on their scope of use (who can access them) and their architectural style (how they are built).

Type	Based On	Purpose & Description	Real-Life Example
Open/Public APIs	Scope	Available to any external developer. Used to foster innovation, extend brand reach, and create ecosystems.	**Twitter API:** Allows developers to integrate tweet posting or fetching into their apps.
Partner APIs	Scope	Exposed only to specific strategic business partners. Not available to the public. Often involve authentication and authorization.	**Netflix ISP Partner API:** Allows Internet Service Providers like Comcast to integrate Netflix seamlessly into their set-top boxes.
Internal/Private APIs	Scope	Used within a single organization to connect systems and departments. Improve efficiency and reuse of services.	A bank's internal API connecting its customer portal (front-end) to its legacy database system (back-end).
Composite APIs	Architecture	Combine multiple API calls (often to different endpoints or services) into a single call. Reduces server load and network traffic for the client.	An e-commerce checkout process that uses one API call to (1) update inventory, (2) process payment, and (3) create a shipping label.
REST API	Architecture	**Representational State Transfer.** Uses standard HTTP methods (GET, POST, PUT, DELETE) and is stateless. Data is usually in JSON/XML format.	**Facebook Graph API:** A vast majority of interactions with Facebook's platform (getting user info, posting photos) use RESTful principles.
SOAP API	Architecture	**Simple Object Access Protocol.** A strict, standardized protocol that uses XML for messaging. Has built-in security (WS-Security) and transaction compliance.	**PayPal API:** Historically used SOAP for its robust security features, ensuring safe financial transactions.

3. REST API

Key Principles:

- **Stateless:** Each request from the client must contain all the information the server needs to fulfill it. The server does not store any client context between requests.
- **Client-Server:** Separation of concerns. The client handles the user interface, and the server handles data storage, improving portability.
- **Uniform Interface:** Resources (like user, product) are identified by URLs (URIs). Uses standard HTTP verbs:
    - GET - Retrieve a resource.
    - POST - Create a new resource.
    - PUT - Update a resource.
    - DELETE - Remove a resource.
- **Cacheable:** Responses must define themselves as cacheable or not to improve performance.
- **Layered System:** The client cannot tell if it is connected directly to the end server or to an intermediary (like a load balancer).

Real-Life Example: GitHub REST API

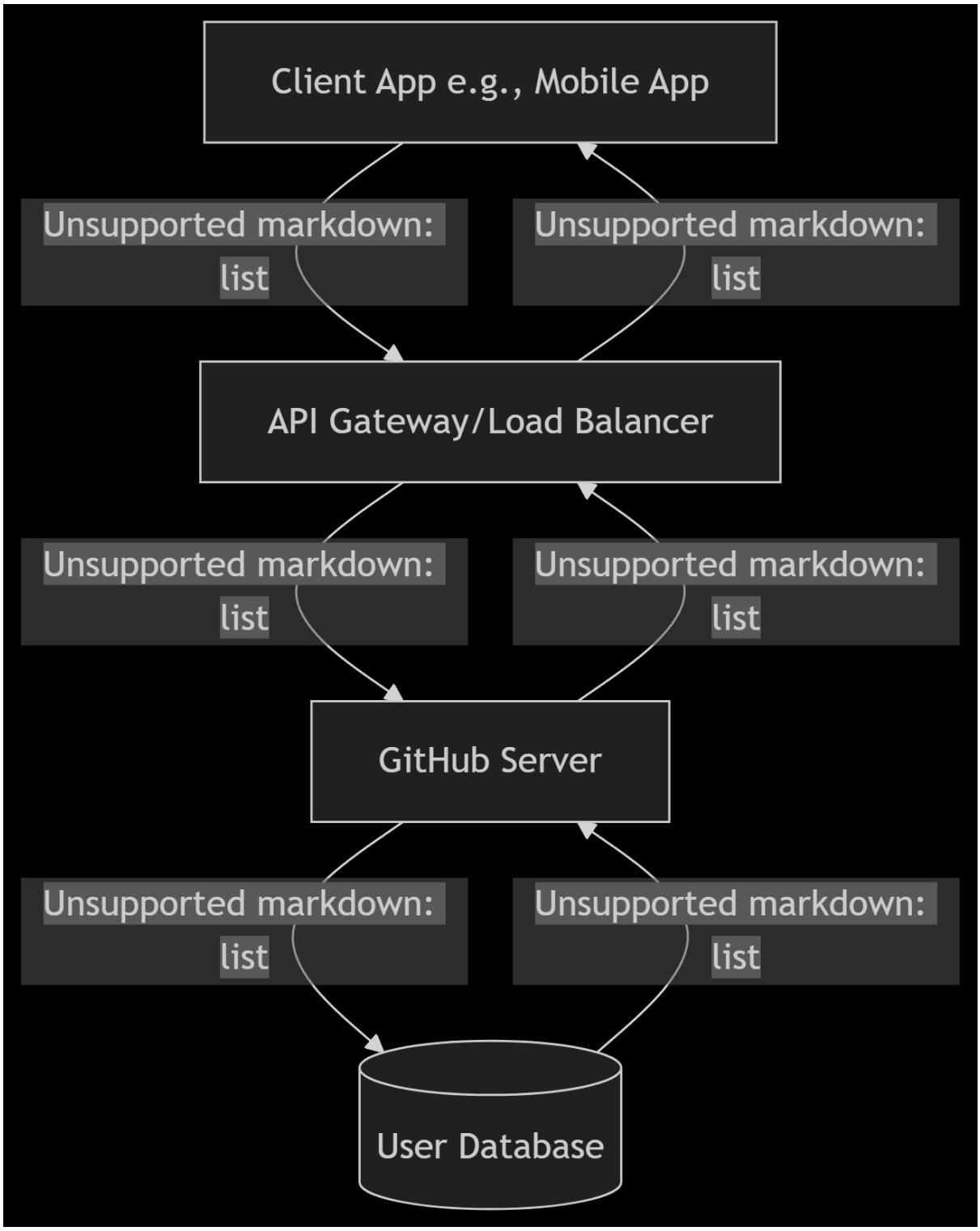
Let's say a developer wants to get a list of all repositories for a user named "janedoe".

1. **API Request:**
    - **HTTP Method:** GET
    - **Endpoint URL:** https://api.github.com/users/janedoe/repos
    - **Headers:** Authorization: Bearer <Personal-Access-Token>

2. **API Response (Simplified JSON):**

```
json
[
  {
    "id": 12345,
    "name": "awesome-project",
    "full_name": "janedoe/awesome-project",
    "html_url": "https://github.com/janedoe/awesome-project",
    "description": "This is my first repository",
    "language": "Python"
  },
  {
    "id": 67890,
    "name": "data-analysis",
    "full_name": "janedoe/data-analysis",
    "html_url": "https://github.com/janedoe/data-analysis",
    "description": "Code for my data analysis blog",
    "language": "Jupyter Notebook"
  }
]
```

Data Flow Diagram (REST API)



4. Deep Dive: SOAP API

Key Characteristics:

- **Protocol:** SOAP is a strict protocol with official standards (W3C).
- **XML-Based:** All messages are sent in XML format inside a "SOAP Envelope".
- **Built-in Standards (WS-*):** Has built-in support for security (WS-Security), transactions (WS-Transactions), and reliability (WS-ReliableMessaging). This makes it very secure and ACID-compliant.
- **Stateful:** Can maintain state over multiple requests, which is complex but sometimes necessary.
- **Less Flexible:** More verbose and requires more bandwidth than REST.

Real-Life Example: A Weather Service SOAP API

A application needs to get the temperature for a specific ZIP code.

1. **API Request (SOAP XML Message):**

xml

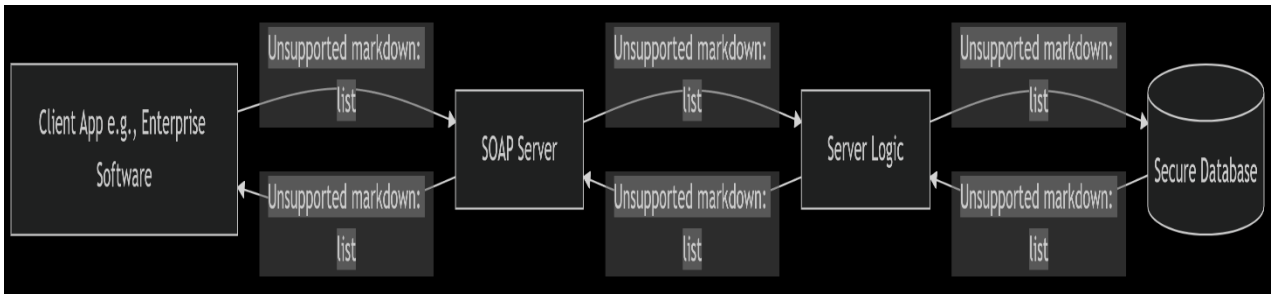
```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <!-- WS-Security credentials could go here -->
  </soap:Header>
  <soap:Body>
    <getWeather xmlns="http://www.example.org/weather/">
      <ZipCode>600001</ZipCode>
    </getWeather>
  </soap:Body>
</soap:Envelope>
```

2. **API Response (SOAP XML Message):**

xml

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <getWeatherResponse xmlns="http://www.example.org/weather/">
      <Temperature>28</Temperature>
      <Unit>Celsius</Unit>
    </getWeatherResponse>
  </soap:Body>
</soap:Envelope>
```

Data Flow Diagram (SOAP API):



5. REST vs. SOAP: A Quick Comparison

Feature	REST API	SOAP API
Architecture	Architectural **style** using HTTP.	Official **protocol** with strict rules.
Data Format	JSON, XML, HTML, plain text (lightweight).	XML only (verbose).
Standards	Relies on HTTP standards (GET, POST, etc.).	Uses WS-* standards (WS-Security, etc.).
Security	Relies on HTTPS (transport layer).	Built-in end-to-end security (message layer).
Performance	Faster, less bandwidth, cacheable.	Slower, more bandwidth, less cacheable.
Use Cases	Web services, mobile apps, public APIs.	Enterprise apps, financial services, high-security transactions.

Conclusion

APIs are the fundamental building blocks of the modern digital world, enabling the seamless integration and interoperability of diverse applications. While **REST** has become the dominant choice for its simplicity, flexibility, and performance in web and mobile scenarios, **SOAP** remains crucial in enterprise environments where stringent security, reliability, and transactional compliance are non-negotiable. Understanding both paradigms is key for any technical professional navigating the interconnected software landscape.