API Notes - Complete Guide

⚙️ What is an API?

API (Application Programming Interface) is a set of rules that allows two software applications to communicate with each other.

With the help of API, the client and server can communicate with each other and send the request and response in a secure way.

Example: When you use a weather app — it calls a Weather API to fetch live temperature data from a server.

💻 Types of APIs Based on Architecture / Communication Style

---

1. REST API (Representational State Transfer)

Most common API today for web applications because they are lightweight, stateless, and easy to implement with Node.js and Express.

REST APIs use standard HTTP methods to handle communication between client and server:

- **GET** - Retrieve data (read)
- **POST** - Create new data
- **PUT** - Update existing data (replace)
- **PATCH** - Partially update data
- **DELETE** - Remove data

Key Points:

- Data usually sent in JSON format
- Easy to understand and use
- Stateless (server doesn't store session info)

---

2. SOAP API (Simple Object Access Protocol)

Older but more secure and structured type of API. They follow strict rules and ensure reliable and guaranteed delivery. Used in Banking systems, financial services, or enterprise apps needing high reliability.

Key Points:

- Uses XML for data exchange
- Very strict standards and error handling

- Supports ACID transactions and enterprise-level security
- SOAP may not be more flexible and lightweight but for secure and reliable delivery they are used

---

3. GraphQL API (Graph Query Language)

What is GraphQL?

GraphQL is an API query language and a runtime developed by Facebook (in 2012).

It's an alternative to REST API, designed to make data fetching more efficient.

Key Features:

- The client can ask exactly what it needs (not over-fetching or under-fetching)
- Uses a single endpoint (e.g., /graphql)
- Developed by Facebook

When to Use GraphQL:

✅ When you have complex data relationships (like social media apps)
✅ When your frontend (web or mobile) needs customized data
✅ When you want to reduce multiple API calls into one
✅ Used by GitHub and Shopify

GraphQL vs REST API

Feature	REST API	GraphQL API
Data Fetching	Multiple endpoints	Single endpoint (/graphql)
Response Data	Fixed data structure	Flexible (client chooses fields)
Over-fetching / Under-fetching	Common problem	Solved – get only needed data
Performance	Multiple network calls	One efficient call
Versioning	New versions (v1, v2)	No versioning needed
Real-time Support	Not built-in	Built-in with subscriptions

Example to Understand

🧩 **In REST API:**

If you want a user's name and their posts, you might need to call:

GET /users/1

GET /users/1/posts

➡️ Two API calls, and maybe you get extra data you don't even need.

⚡ **In GraphQL:**

You can do it in one single request:

```
{
  user(id: 1) {
    name
    posts {
      title
    }
  }
}
```

➡️ You get exactly what you asked for — nothing extra, nothing missing.

🔌 **What is a WebSocket?**

WebSocket is a communication protocol that enables **full-duplex (two-way)** and **real-time communication** between a client (like a browser) and a server over a single TCP connection.

In simpler words:

It allows both the client and server to send and receive data anytime, without having to make new requests each time.

⚙️ **Why Was WebSocket Needed? (The Problem with HTTP)**

Before WebSockets, we only had HTTP (used by REST APIs).
But HTTP has some limitations when you need real-time updates.

🧩 **Limitations of HTTP (REST API):**

1. Unidirectional:

- Only the client can send a request, and the server only responds
- The server cannot push new data to the client by itself

2. Polling:

- To get updates, the client has to repeatedly ask: "Do you have any new data?"
- This wastes bandwidth and increases latency
- So real-time updates are not there

3. Latency Issues:

- Real-time applications (like chat, stock prices, or online games) require instant updates
- HTTP is too slow for that

4. Connection Overhead:

- Each request opens and closes a new connection
- Not efficient for continuous data flow
- Means each time new request needs to establish

How WebSocket Solves This

WebSocket creates a **persistent connection** between the client and server after an initial handshake using HTTP.

Once the connection is established, both sides can send data anytime — instantly and efficiently.

✅ One connection — both directions
✅ Low latency, high performance
✅ Perfect for real-time communication

How It Works (Step by Step)

1. Handshake Phase:

- Starts as a normal HTTP request
- Client requests the server to "upgrade" the connection to WebSocket using this header:
- Upgrade: websocketConnection: Upgrade

2. Connection Established:

- Server agrees, and the connection switches from HTTP to WebSocket

3. Full-Duplex Communication:

- Both client and server can now send messages independently
- No need to wait for requests

4. Connection Closed:

- Either side can close it when done

🧠 **Example Use Cases**

Use Case	Description
💬 Chat Applications	Send and receive messages instantly
🕹️ Online Multiplayer Games	Real-time player moves and updates
📈 Stock/Price Dashboards	Live updates of prices without refresh
🚖 Real-time Tracking	Show location updates instantly (like Uber)
📢 Live Notifications	Push alerts instantly from server to users

---

🧩 WebSocket vs HTTP (REST API)

Feature	HTTP / REST API	WebSocket
Connection Type	One-way (request/response)	Two-way (full-duplex)
Communication	Client → Server only	Client ↔ Server
Connection	New connection for each request	Single persistent connection
Use Case	Normal web apps, CRUD APIs	Real-time apps (chat, live feeds)
Performance	Higher latency	Low latency
Protocol	HTTP	ws:// or wss://

🧰 Advantages of WebSocket

- ✅ Real-time, instant data updates
- ✅ Reduces network traffic (no need for repeated polling)
- ✅ Persistent, efficient connection
- ✅ Ideal for IoT, chat, live dashboards

⚠️ Disadvantages

- ❌ Slightly complex to manage connections and scaling
- ❌ Not needed for simple request/response apps
- ❌ Requires WebSocket-compatible load balancers or servers

End of Notes