API Protocols

1. REST (Representational State Transfer)

- A lightweight architectural style using **HTTP** for communication
- Follows principles like **statelessness (**Each request contains all necessary information, and the server stores **no client session state** between requests**)** and **resource-based URLs**
- Each API call is **independent** (no stored session state)
- Common HTTP methods: **GET, POST, PUT, DELETE**
- **Use Cases:** Web apps, mobile apps, public APIs (Twitter, GitHub)
- **Example:**

  A client hits:

  ```
  GET https://api.github.com/users/arpi
  ```

  Server responds with JSON:

  ```
  { "name": "Arpi", "repos": 42 }
  ```

---

2. SOAP (Simple Object Access Protocol)

- A formal, protocol-driven approach relying on **XML** for messaging
- Designed for **highly structured**, secure, enterprise-level interactions
- Built-in mechanisms for **error handling** and **security** (e.g., WS-Security)
- Typically heavier and more rigid than REST
- **Example:**

  ```
  <soap:Envelope>
    <soap:Body>
      <GetUserInfo>
        <UserID>123</UserID>
      </GetUserInfo>
    </soap:Body>
  </soap:Envelope>
  ```

  The response is also XML wrapped in this envelope format.

---

3. GraphQL

- Lets clients **request exactly the data they need**
- Eliminates over-fetching and under-fetching
- All operations are served from **a single endpoint**
- Response structure is defined by the **client**
- Ideal for complex UIs needing efficient data access
- **Example:**

  Client sends ONE query to the single GraphQL endpoint:

  ```
  POST /graphql
  ```

  Query body:

```
{
  user(id: "123") {
    name
    email
    posts {
      title
    }
  }
}
```

Server returns exactly those fields — nothing more, nothing less.

---

4. gRPC (Google Remote Procedure Call)

- A high-performance RPC framework by Google
- Uses **Protocol Buffers (Protobuf: A binary, schema-based data serialization format by Google that makes data extremely compact and fast to transmit)** for compact and fast message serialization
- Runs over **HTTP/2 (An improved version of HTTP that supports multiplexing, header compression, and faster, parallel communication over a single connection)**, enabling bi-directional streaming
- Supports multiple languages with low latency and small payloads (The payload is the **actual serialized data** (usually Protobuf messages) sent between client and server)
- Excellent for microservices and distributed systems
- **Example:**

  You define a service using Protobuf:

  ```
  service UserService {
    rpc GetUser (UserRequest) returns (UserResponse);
  }
  ```

  The client calls `GetUser()` like a normal function, receiving a compact Protobuf payload like:

  ```
  id: 123
  name: "Arpi"
  ```

  No JSON, no XML — just tiny, binary-encoded data zooming over HTTP/2.

---

5. WebSocket

- Provides **full-duplex (**Both client and server can **send and receive data simultaneously** over the same connection**)**, real-time communication over a single TCP connection
- No need to re-establish connections for continuous data flow
- Very low latency
- **Use Cases:** Chat apps, live dashboards, gaming, real-time updates
- **Example:**

  A WebSocket URL looks like:

  ```
  ws://chat.example.com/socket
  ```

  After connection:

  - Client: `"Hey!"`
  - Server: `"Hello there!"`

- Client: `"Typing..."`
- Server: `"User is offline"`

All real-time, both directions, same connection, no waiting.

Quick Comparison Table

Feature	REST	SOAP	GraphQL	gRPC	WebSocket
Data Format	JSON, XML	XML only	JSON	Protobuf	Custom frames (binary/text)
Flexibility	High	Low	Very High	Medium	High (real-time)
Performance	Fast	Slower	Efficient	Very Fast	Very Fast
Best Use Case	Modern web APIs	Enterprise-grade systems	Dynamic client data needs	Microservices, internal APIs	Real-time apps