[REST API]

Representational state transfer

Key Characteristics of REST:

- **Resource-Oriented:** REST treats everything as a resource, such as users, products, or orders. Each resource is identified by a **unique URL**.
- **Stateless:** RESTful services are stateless, meaning each request from a client to a server must contain all the necessary information for the server to understand and process the request. The server does not store any session state about the client between requests.
- **Standard HTTP Methods:** REST leverages the existing HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources. For example:
  - **GET:** Retrieves a resource.
  - **POST:** Creates a new resource.
  - **PUT:** Updates an existing resource.
  - **DELETE:** Removes a resource.
- **Client-Server Architecture:** REST strictly follows the client-server model, where the client handles the user interface and the server manages the data.
- **Cacheable:** REST responses can be cached to improve performance and scalability.

Advantages of REST:

- **Scalability:** RESTful APIs are scalable due to their stateless nature and the use of standard HTTP methods.
- **Flexibility:** REST is flexible and allows different data formats, such as JSON, XML, or plain text.
- **Standardization:** RESTful APIs follow a standardized approach that makes them easy to understand and implement.
- **Wide Adoption:** REST is widely supported and used across the web, making it easy to find tools, libraries, and community support.

Disadvantages of REST:

- **Verbosity:** REST can be verbose, especially when dealing with complex objects or relationships.
- **Over-fetching/Under-fetching:** Clients may receive more data than needed (over-fetching) or less data than needed (under-fetching), leading to inefficiencies.
- **Limited Actions:** REST is resource-centric, which can make it challenging to model complex actions that do not fit neatly into CRUD operations.

→ This is solved by GraphQL

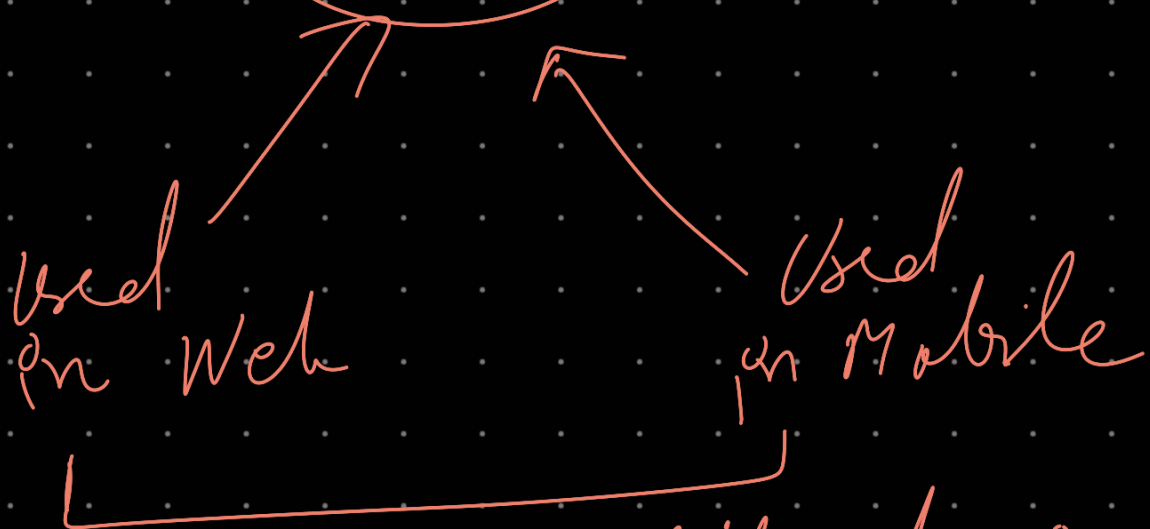
When to use graphQL over REST?

Use case	GraphQL	REST
Public ad-hoc API that you can't predict how it will be used	✅	❌
Specific and well-design use cases API	❌	✅
Simple API that serves one client (webpage)	❌	✅
Enterprise API (new york times)	✅	❌
Well defined schema	✅	❌

What is graphQL ?

L) Need ?

E.g. → /post endpoint



Both needs different json response

L) It can be done using if else condition but code is hard to manage

L) can be solved by graphQL

(Deep dive in later section)

[RPC]
↳ Remote Procedure call
⇓
Seems like function call?

RPC, or Remote Procedure Call, is a protocol that allows a program to execute a procedure (subroutine) on another address space (commonly on another physical machine) as if it were local.

RPC abstracts the complexity of the communication process, allowing developers to focus on the logic of the procedure.

Key Characteristics of RPC:

- **Action-Oriented:** Unlike REST, which is resource-oriented, RPC is action-oriented. It focuses on invoking specific methods or procedures.
- **Function Calls:** RPC allows you to call functions or methods directly on a remote server. The client sends a request to execute a specific method, and the server returns the result.
- **Variety of Protocols:** RPC can be implemented using different protocols, such as JSON-RPC, XML-RPC, or gRPC (Google Remote Procedure Call).
- **Less Overhead:** RPC often has less overhead than REST, which can make it faster and more efficient in certain scenarios.
- **Synchronous Communication**: Traditional RPC is synchronous, meaning the client waits for the server to complete the called procedure and return the results.
- **Interface Definition Language (IDL)**: RPC systems often use an IDL to define the interface between the client and server.

Advantages of RPC:

- **Simplicity:** RPC can be simpler to implement when you need to perform specific actions, as it directly maps to method calls.
- **Performance:** RPC can offer better performance due to its lower overhead compared to REST.
- **Flexibility in Actions:** RPC is more flexible in handling complex operations that do not fit the CRUD model.

Disadvantages of RPC:

- **Tight Coupling:** RPC can lead to tight coupling between the client and server, making it harder to change or update the API without affecting clients.
- **Less Standardization:** Unlike REST, which uses standard HTTP methods, RPC does not follow a standardized approach, which can lead to inconsistencies.
- **Limited Tooling:** While there are tools for RPC, they are not as widely adopted as REST tools, which can make development and debugging more challenging.

When to Choose REST or RPC

Choose REST when:

1. Building public-facing APIs
2. You need maximum scalability and flexibility
3. Your API consumers are diverse and unknown
4. You want to leverage HTTP features (caching, proxies)
5. Your operations map well to CRUD actions on resources

Choose RPC when:

1. Building internal microservices
2. You need maximum performance
3. You have complex operations that don't map well to CRUD
4. Your client and server are tightly coupled
5. You're working in a homogeneous environment