CLIENT

- A Client is: Any device or application that sends a request to a server to get some information or service.
- The client always asks, and the server answers.

"If you ask for something, you become the client."

Examples:

- You ask a shopkeeper → You are the client
- You ask a teacher a question → You are the client
- You ask Swiggy to bring food → Your phone is the client
- The one who requests is always the client.

In computers:

A client can be:

- A browser (Chrome, Firefox)
- A mobile app (Instagram, WhatsApp)
- A laptop or phone
- A desktop application (Zoom, Teams)
- A smart device (Smart TV)

When a client needs something from a server, it sends a request.

Examples:

- When Chrome opens Google → Chrome is the client
- When Instagram loads feed → Instagram app is the client
- When WhatsApp sends a message → WhatsApp is the client

The server receives these requests and responds with data.

Types of Clients (Simple Categories)

1. Web Clients
   - Chrome
   - Firefox
   - Edge

2. Mobile Clients
    - Instagram app
    - Amazon app
    - YouTube app

3. API Clients
    - Postman
    - Hoppscotch

4. IoT Clients
    - Smart bulbs
    - Alexa
    - Smart TV

All of these send requests to servers.

A client = The one who STARTS the conversation.

Servers NEVER start the talk.

Example:

- Instagram server will never contact you first.
- You must open the app → client starts the communication.

SERVER

- Whenever two people communicate, one asks and the other answers.
- The same happens in technology.
- When you ask your friend a doubt → You are the client, your friend is the server.
- When you ask your teacher a question → You become the client, teacher becomes the server.

- A server is simply the “one who answers.”
- **Definition** - A Server is a powerful computer or software that receives requests and sends responses.

SERVER IN STUDENTS POINT OF VIEW

Ask them:

“What do you do when you forget a subject definition during study?”

- You open Google → You send a request
- Google sends back the answer → Google acts as server

“How do you check your exam results?”

- You enter register number → Request
- Server returns marks → Response

“How does Instagram know your follower count?”

- Your app asks server →
- Server checks database →
- Sends follower number →
- Your app shows it.

The server is the source of truth.

HOW A SERVER REALLY WORKS

Step 1: Server waits

- It doesn’t talk unless someone asks.

Step 2: Client sends request

- Example: “Give me my Instagram feed.”

Step 3: Server processes

It checks:

- Is the user logged in?
- What posts should be shown?

- What data is relevant?

Step 4: Server sends response

Usually JSON, like:

```
{
  "posts": [
    "image1.jpg",
    "image2.jpg"
  ]
}
```

Step 5: Client displays it

- Your phone shows the images beautifully.

TYPES OF SERVERS

Application Server

- Runs your backend code (Spring Boot).

Database Server

Stores data like:

- Users
- Orders
- Photos
- Messages

Authentication Server

- Manages logins and tokens.

Game Server

- Manages multiplayer gaming.

Cloud Servers

- AWS, Google Cloud, Azure — where companies run everything.

HOW SERVER LOOKS IN REAL LIFE

Servers are:

- Not normal computers
- Not laptops
- They look like big black boxes in racks
- But your laptop can become a server while developing.

Protocol

A protocol is a set of rules that decides how two systems talk to each other.

Like:

- Rules in a classroom
- Rules in a game
- Rules in traffic

Similarly, computers also need rules to communicate.

Those rules = HTTP.

HTTP

HTTP is the language or rule that clients and servers use to communicate over the internet.

- Your phone → speaks HTTP
- Instagram server → understands HTTP
- Chrome browser → speaks HTTP
- Google server → understands HTTP

Without HTTP, they cannot talk.

Why Do We Need HTTP?

Because without rules:

- Your phone won't know how to request data
- Server won't know what you want
- No website would load
- No Instagram feed
- No login
- No YouTube

HTTP gives structure to communication.

HTTP Request Contain

A request has 3 important parts:

- HTTP Method
- URL
- Headers + Body

HTTP METHOD

HTTP methods tell the server WHAT ACTION the client wants to perform.

Think of methods as verbs in a sentence:

- GET → fetch
- POST → create
- PUT → update
- DELETE → remove

URL

A URL (Uniform Resource Locator) is the address of the resource you want to access on the server.

Just like:

- Home has an address
- College has an address
- A shop has an address

- A website or API resource also needs an address.

Without a URL, the server won't know what you want.

The URL Has Two Main Parts

1. Base URL (Domain — where the server lives)
2. Path (Which exact resource you want)

1. Base URL (Domain)

This is the location of the server on the internet.

Examples:

- https://google.com
- https://instagram.com
- https://college.edu
- https://api.amazon.com

This only tells the browser where the server is, not what to fetch.

2. Path (The exact resource you want)

Examples that students understand:

- Profile page: - /profile
- List of students: - /students
- One specific student: - /students/101
- Posts feed: - /feed
- Login API: - /api/login

This tells the server exactly what data or service the client needs.

Combine Both: Full URL - https://instagram.com/my-feed

Meaning:

- Connect to the Instagram server
- Get the resource called /my-feed (posts)

URL Can Also Contain Query Parameters

Used when you want to send extra information inside the URL.

Examples:

- **Searching on YouTube: -** https://youtube.com/search?q=java
- q=java → search keyword
- **Filtering Amazon products: -** /products?brand=Nike&price=under5000

HEADERS

Headers are additional details sent along with the request to help the server understand the request better.

Think of headers like labels on a parcel:

- Who sent it?
- What type of content is inside?
- What is the priority?
- What language?
- Is this user logged in?

Without headers, the server will not know how to process the request correctly.

IMPORTANT HEADERS

1. Content-Type

Tells the server what format your request body is in.

Examples:

- application/json → Sending JSON
- application/xml → Sending XML
- multipart/form-data → Sending files (images, videos)
- text/plain → Sending plain text

Student-friendly example:

Instagram upload: - Content-Type: multipart/form-data

2. User-Agent

Tells the server which device/browser is sending the request.

Examples:

- Chrome on Windows
- Safari on iPhone
- Instagram app on Android

Real example:

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)

Why needed?

Servers optimize responses based on device.

BODY

The request body is the main data you send to the server.

Think of body as:

- The content of the message
- The main information
- The actual data the server needs

When Do We Send a Body?

Body is used only for methods that send data:

- POST
- PUT
- PATCH
- GET does NOT send a body

Because GET only asks for data — it does not send any.

WHAT DOES THE BODY CONTAIN?

The body depends on what the user is doing.

Login → Body contains credentials

```
{
  "email": "student@gmail.com",
  "password": "123456"
}
```

Registration → Body contains user data

```
{
  "name": "Arun",
  "email": "arun@gmail.com",
  "password": "abcd123"
}
```

Uploading a photo → Body contains file data

Format:

multipart/form-data

Updating profile → Body contains new values

```
{
  "bio": "Future software engineer",
  "location": "Chennai"
}
```

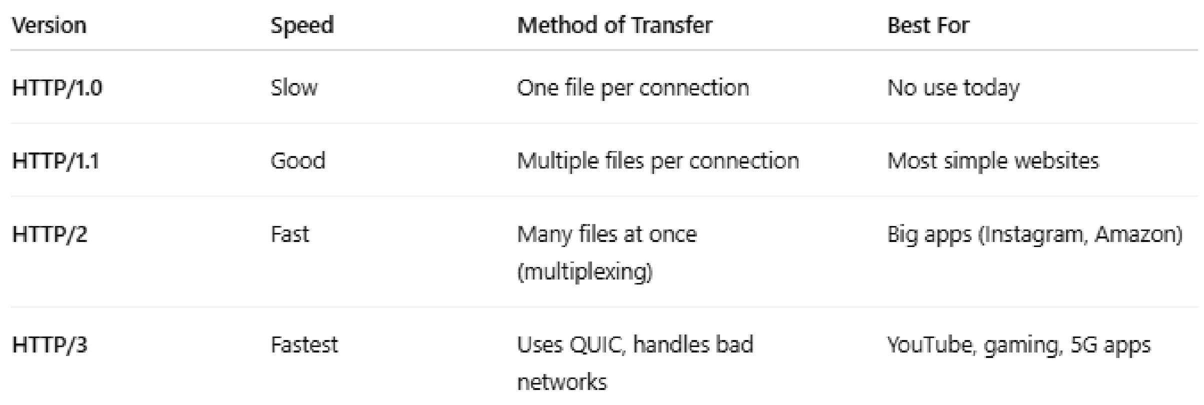
HTTP RULES

- RULE 1 — Client Must Always Start the Communication
- RULE 2 — Every HTTP Communication Has 2 Parts
- RULE 3 — HTTP Follows a Fixed Request Structure – Request line, header, body
- RULE 4 — The Server Must Send a Status Code in Every Response
- RULE 5 — HTTP Uses Standard Methods
- RULE 6 — URLs Must Identify What You Want
- RULE 7 — HTTP Is Stateless
- RULE 8 — Everything Must Be in Text Format
- RULE 9 — The Client Must Specify Data Type
- RULE 10 — HTTP Response Must Follow Structure
- RULE 11 — Use of HTTP Versions
- RULE 12 — Communication Must Follow ASCII Text Encoding
- RULE 13 — HTTP Can Work Over Any Media Layer
- RULE 14 — HTTP Does Not Care About the Device

HTTP Versions

There are 4 major versions you should teach:

- HTTP/1.0
- HTTP/1.1
- HTTP/2
- HTTP/3



Version	Speed	Method of Transfer	Best For
HTTP/1.0	Slow	One file per connection	No use today
HTTP/1.1	Good	Multiple files per connection	Most simple websites
HTTP/2	Fast	Many files at once (multiplexing)	Big apps (Instagram, Amazon)
HTTP/3	Fastest	Uses QUIC, handles bad networks	YouTube, gaming, 5G apps

JSON - JavaScript Object Notation

- JSON is a lightweight, easy-to-read text format used for sending data between client and server.
- It is the language of data in the web world.

Why JSON?

- **Easy to read -** Looks like a simple note.
- **Easy to write -** Simple key–value pairs.
- **Lightweight** - Doesn't take much space.
- **Works in ALL programming languages -** Java, Python, C#, JavaScript, Go, Kotlin — all support JSON.
- **Perfect for API communication**
    - Spring Boot → React
    - Node → Android
    - Server → Browser

JSON STRUCTURE

```
{
  "name": "Selva",
  "age": 22
}
```

JSON Format Rules

- Must start with { } → JSON Object
- Keys must be in "double quotes"
- Values can be:
    - String → "Arun"
    - Number → 20
    - Boolean → true / false
    - Array → [ ]
    - Object → { }
    - Null → null

- Data separated by commas
- Last item should NOT have a trailing comma

EXAMPLE

```
{
  "username": "selvaraj_001",
  "followers": 1200,
  "isVerified": false,
  "posts": [
    "photo1.jpg",
    "photo2.jpg"
  ],
  "bio": {
    "text": "Coder | Guitarist",
    "location": "Chennai"
  }
}
```

XML = Extensible Markup Language

- XML is a structured way of storing data using tags.
- Just like HTML uses tags like <p>, <h1>,
- XML also uses tags — but you create your own tags.

Example:

```
<name>Selva</name>
```

Before JSON existed, XML was used because:

- It is structured
- It is readable
- It supports nested data
- It works in all programming languages
- It was perfect for data exchange between systems

Even today:

- Banks
- Insurance systems
- Railways
- Government portals
- SOAP APIs
- still rely on XML.

Example

```
<student>

  <name>Arun</name>

  <age>20</age>

  <department>CSE</department>

</student>
```

XML Must Have a Root Element

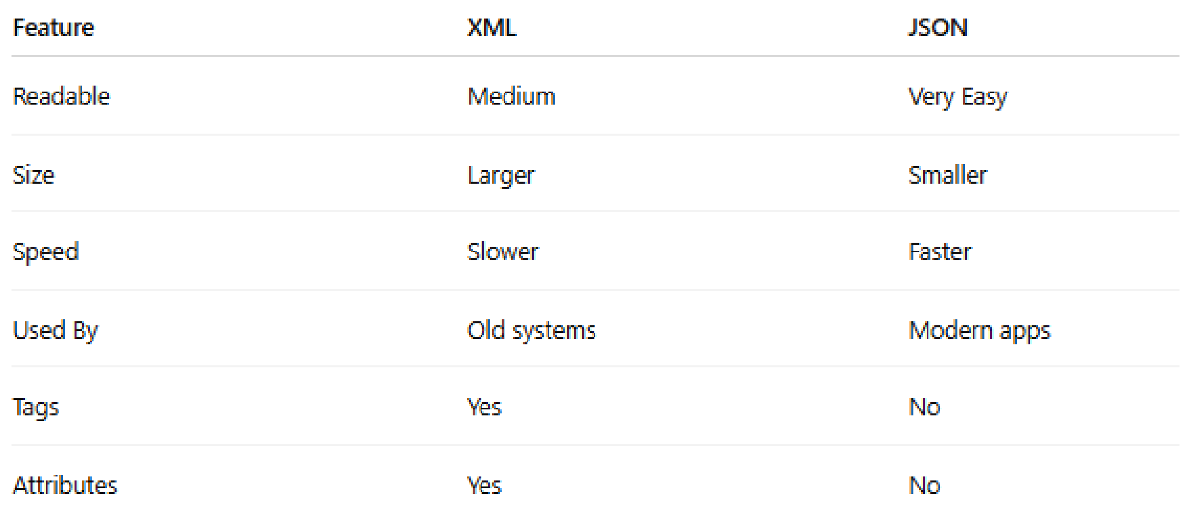
Every XML file MUST start with a single root element that wraps all data.

Example:

```
<students>

  <student>...</student>

  <student>...</student>

</students>
```

Here:

- <students> = root element
- <student> = child elements
- This rule is very important.



Feature	XML	JSON
Readable	Medium	Very Easy
Size	Larger	Smaller
Speed	Slower	Faster
Used By	Old systems	Modern apps
Tags	Yes	No
Attributes	Yes	No