Session 1 :- Introduction to Web

🌐 HTTP – Hypertext Transfer Protocol

📌 What is HTTP?

- A **protocol** used to **transfer hypertext** over the internet.
- It follows a **set of rules** for communication between web clients and servers.
- Used for accessing **web documents (web pages)**.
- Example: When you open a webpage, you're using HTTP or HTTPS (secure version).

🔍 Human Readable

- HTTP is **human-readable**, meaning you can see its activity in:
  - **Network tab** of Developer Tools (in browsers)
  - **Inspect element**
  - **Page source**

🧠 Stateless Protocol

- **Each request is independent** of the previous ones.
- Whenever you visit a page, **you are treated as a new user**.
- There is **no memory of previous requests** by default.
- Example: YouTube home page always resets unless you're logged in.

⚠️ Problem with Statelessness:

- Web apps **need to remember users** for many tasks like:
  - Keeping you **logged in**
  - Storing items in your **shopping cart**
  - Tracking your **preferences** or activity

✅ Solution: Session Management

To solve this, we **create a session** on top of HTTP:

🔷 How it works:

1. You **log in** on a website.
2. The server **creates a session** for you and gives your browser a **Session ID** (usually stored in a **cookie**).
3. On every future request, your browser sends the **Session ID** with the request.
4. The server uses this ID to **identify you** and **remember your state**.

🧾 Session (Maintaining State)

- Sessions help to **store state** between the **frontend (browser)** and **backend (server)**.
- **Cookies** are used to manage session data.
  - Stored in the **browser** and sent to the **server** with every request.

📦 HTTP Headers

When sending info from **server to browser**, HTTP uses **headers** to pass:

- 👤 Client info
- 🖥️ Browser info
- ⏰ Date & time
- 🍪 Cookie data

🔄 Request-Response Model

- The core model of HTTP is based on **Request ↔ Response**.
- Browser (Client) ↔ Server communication follows this pattern:
  1. Browser **sends a request**.
  2. Server **processes it** and **sends a response** back.

🔽 Request:

- Includes:
  - HTTP Method (e.g., **GET, POST, DELETE**)
  - URL (e.g., `http://api.website.com/auth`)
  - Headers
  - Body (optional – mainly in POST, PUT)

🔼 Response:

- Contains:
  - Status Code: **200** (OK), **404** (Not Found), **500** (Server Error), etc.
  - Headers
  - Body (e.g., HTML, JSON, etc.)

🧠 Tip: HTTP is like a conversation

> Think of it like sending and receiving messages or commands – the browser asks, and the server replies.

🌐 HTTP/2 — Next Version of HTTP

✅ What is HTTP/2?

- **HTTP/2** is an updated version of the HTTP protocol.
- **HTTP/1.1** is still used as a **fallback** if HTTP/2 is not supported.

🔧 Key Features of HTTP/2:

1. **Compression** – Reduces data size to speed up loading.
2. **Multiplexing** – Allows **multiple files** (e.g., images, CSS, JS) to be sent at the same time over a single connection.
3. **Encryption** – Uses **HTTPS** (secure) to protect data.

> 🛑 Note: In some private/internal networks (e.g., inside AWS), HTTPS might not be used for internal communication.

🔒 TLS – Transport Layer Security

✅ What is TLS?

- **TLS is a security protocol** that keeps your data **safe** when sent over the internet.
- It's used in **HTTPS** (the "S" means secure).
- It **encrypts** your data so that no one can read or change it while it's traveling between your browser and the server.

💡 Why is TLS Important?

Without TLS:

- Anyone (hackers, attackers) can **see your data** (like passwords, card numbers).
- Your identity and privacy are **not protected**.

With TLS:

- Your connection is **safe** and **private**.
- Even if someone intercepts it, they **can't read the data** (because it's encrypted).

🔐 TLS in Action – Step by Step

1. Browser connects to server

- You enter a URL like `https://example.com`.

2. Server sends TLS Certificate

- This certificate proves the website is **genuine** and **trusted**.
- Issued by a **Certificate Authority (CA)**.

3. Handshake happens

- Browser and server **agree on how to encrypt data**.
- They exchange **keys** securely.

4. Encrypted Communication starts

- Your data (login info, forms, messages) is sent in **encrypted form**.
- Only the server can **decrypt** and read it.

🔍 TLS vs SSL

- **SSL** (Secure Sockets Layer) was the **older version** of TLS.
- Used in HTTP

Inside AWS, HTTPS is often skipped for internal communication

✅ Why?

- **Performance**: HTTPS adds extra overhead (encryption/decryption). For high-performance apps with many internal calls, this can slow things down.

- **Already secure network**: AWS's **internal network is private, isolated, and secure by design**.
  - Traffic within the same **VPC (Virtual Private Cloud)** doesn't go over the public internet.
- **Trust between services**: Services like EC2, Lambda, RDS within the same VPC **trust each other**, and encryption is not always required.

🛡️ But is it safe?

Yes, **it's still safe** because:

- AWS VPC is a **private and protected network**.
- You can still choose to **enable HTTPS or TLS** internally if you want **end-to-end encryption** (like for extra sensitive data or compliance needs).

🧠 Key Terms You Should Know:

👤 User Agent:

- The browser (like Chrome, Firefox) you're using.

📡 Protocols:

- **TCP (Transmission Control Protocol)** – Ensures reliable communication.
- **FTP (File Transfer Protocol)** – Used for transferring files.
- **IP (Internet Protocol)** – Handles addressing and routing.
- **URL (Uniform Resource Locator)** – The web address you enter.
- **DNS (Domain Name System)** – Converts a **URL** (like google.com) into an **IP address** so your computer can find the server.

📩 HTTP Components:

- **Headers** – Extra info like browser type, language, cookies.
- **Payload** – The real data (e.g., your email, password).
- **Cache** – Stores data to speed up future visits.

⚙️ How a Web Request Works (Step-by-Step)

📌 Between your **browser** and the **server**:

1. **Setup a TCP connection** (a reliable link).
2. **Exchange TLS Certificate** if using **HTTPS** (for secure connection).
3. **Send the request**:
   - Method (GET, POST, etc.)
   - URL (web address)
   - Data (form info, etc.)
4. **Server sends response**:
   - Status code (like 200 OK, 404 Not Found)
   - Data (like image, text, JSON, etc.)
5. **Connection is closed**:
   - Because HTTP is **stateless**, the server forgets the client after responding.

💡 Easy Analogy:

> Think of HTTP like a customer (browser) ordering food (request) from a restaurant (server). After the food is delivered (response), the server forgets who you are unless you give a token (session/cookie).