HTTP Interview Questions

Basic Questions

1. What is HTTP, and how does it work?

- **Definition:**
    - HTTP (HyperText Transfer Protocol) is the foundation of communication on the web.
    - It enables the transfer of resources such as web pages, images, and API responses.
- **How It Works:**
    - A **client (browser, mobile app, API client)** sends an **HTTP request** to a web server.
    - The **server processes the request** and returns an **HTTP response** containing data or an error message.
    - The **client receives the response** and renders the requested resource (e.g., a webpage).
- **Example:**
    - When you type `https://www.example.com` in a browser:
        - The browser sends a **GET request** to the web server.
        - The server responds with the HTML of the webpage.
        - The browser displays the webpage to the user.

---

2. Why is HTTP considered a stateless protocol?

- **Statelessness in HTTP:**

    - HTTP **does not retain memory of previous requests** between the client and server.
    - Each request is treated as **independent**, meaning the server does not store session information.
- **Challenges Due to Statelessness:**
    - Maintaining **user sessions** (e.g., staying logged in).
    - Every request must include necessary information (e.g., authentication tokens).
- **Solutions to Maintain State:**
    - **Cookies** – Stored in the browser and sent with requests.
    - **Sessions** – Server-side storage of user data.
    - **Tokens (JWT, OAuth)** – Used for authentication and API security.

---

3. What are the key differences between HTTP and HTTPS?

Feature	HTTP	HTTPS
Security	Data is sent in plain text	Data is encrypted using SSL/TLS
Encryption	No encryption	Uses SSL/TLS encryption
Port	Uses port 80	Uses port 443
Data Integrity	Data can be intercepted and modified	Data is protected from tampering
Use Case	Suitable for non-sensitive data	Essential for secure websites (e.g., banking, login pages)

Intermediate Questions

4. Explain the HTTP request-response cycle with an example.

- **Step 1: The Client Sends a Request**

    - The **browser, mobile app, or API client** sends a request to the server.

Example:

vbnet
CopyEdit

```
GET /index.html HTTP/1.1
Host: www.example.com
```

    -
- **Step 2: The Server Processes the Request**
    - The web server receives the request, processes it, and retrieves the required resource.
- **Step 3: The Server Sends a Response**
    - The server responds with a status code, headers, and body.

Example:

pgsql
CopyEdit

```
HTTP/1.1 200 OK
Content-Type: text/html
```

    -
- **Step 4: The Client Renders the Response**
    - The browser processes the received content and displays the webpage.

---

5. What are HTTP methods? When would you use PUT vs. PATCH?

- **Common HTTP Methods & Use Cases:**
    - **GET** – Retrieve a resource.
    - **POST** – Create a new resource.
    - **PUT** – Update an entire resource.
    - **PATCH** – Partially update a resource.

    - **DELETE** – Remove a resource.

- **PUT vs. PATCH:**

Method	Use Case	Example
PUT	Replaces an **entire** resource	Updating a user profile (name, email, password, etc.)
PATCH	Updates **part** of a resource	Changing only the email of a user without modifying the name

-

---

6. What are HTTP status codes? Give examples of 2xx, 3xx, 4xx, and 5xx status codes.

- **1xx – Informational:** Request received and processing continues.
- **2xx – Success:** Request was successfully processed.
    - **200 OK** – Successful request.
    - **201 Created** – New resource successfully created.
- **3xx – Redirection:** Further action required.
    - **301 Moved Permanently** – The resource has a new URL.
    - **304 Not Modified** – Cached version should be used.
- **4xx – Client Errors:** The client made a mistake.
    - **400 Bad Request** – Invalid request syntax.
    - **401 Unauthorized** – Authentication required.
    - **404 Not Found** – Resource does not exist.
- **5xx – Server Errors:** The server encountered an issue.
    - **500 Internal Server Error** – Generic server failure.
    - **503 Service Unavailable** – Server is temporarily overloaded.

Advanced Questions

7. How do cookies, sessions, and tokens help maintain state in HTTP?

- **Cookies:**
    - Small pieces of data stored in the client's browser.
    - Automatically sent with every request to the server.
    - Used for tracking **user sessions, preferences, and authentication**.
- **Sessions:**
    - Server-side storage of user data.
    - The client is assigned a **session ID**, which is stored in a cookie.
    - Common in login-based applications.
- **Tokens (JWT, OAuth):**
    - **JSON Web Tokens (JWT)** – Used in stateless authentication.
    - **OAuth Tokens** – Used for secure API access.
    - Tokens are stored in **local storage or cookies** and sent in headers.

---

8. What is the difference between 301 (Moved Permanently) and 302 (Found) redirections?

- **301 Moved Permanently:**
    - Used when a resource's URL has changed permanently.
    - Example: Redirecting `http://old-site.com` to `https://new-site.com`.
    - Search engines **update** the indexed URL.
- **302 Found (Temporary Redirect):**

- Used for temporary URL changes.
- Search engines **do not update** the indexed URL.

---

9. How does caching work in HTTP, and which headers control it?

- **Purpose of Caching:**
  - Reduces server load and improves performance.
  - Allows browsers to store copies of resources (e.g., images, CSS).
- **Important HTTP Headers for Caching:**
  - **Cache-Control:**
    - `Cache-Control: max-age=3600` (Cache for 1 hour)
    - `Cache-Control: no-cache` (Always fetch fresh data)
  - **ETag:** Unique identifier for a resource version.
  - **Expires:** Defines the expiration date of cached content.

---

10. What security risks are associated with HTTP, and how can they be mitigated?

- **Security Risks:**
  - **Man-in-the-middle attacks** – Data interception.
  - **Phishing attacks** – Fake websites mimicking real ones.
  - **Data tampering** – Modifying HTTP messages.
  - **Session hijacking** – Stealing session cookies.
- **Mitigation Strategies:**
  - **Use HTTPS** – Encrypts data with SSL/TLS.

- **Secure cookies** – Use `HttpOnly` and `Secure` flags.
- **Implement Content Security Policy (CSP)** – Prevents XSS attacks.
- **Use authentication tokens** – JWT or OAuth for secure API access.
- **Validate user input** – Prevents injection attacks.