

Verinite Technologies Pvt Ltd

HTTP, WWW, REST APIs

1)DNS, HTTP / HTTPS, SSL

DNS (Domain Name System):

Think of DNS like a phonebook for the internet.

It translates human-friendly domain names (like example.com) into IP addresses that computers use to identify each other on the internet.

It helps you find websites and services using names instead of long strings of numbers.

HTTP / HTTPS (Hypertext Transfer Protocol / Secure):

HTTP is the protocol used for transferring web pages and other data on the World Wide Web.

HTTPS is a secure version of HTTP that encrypts data sent between your browser and the website, making it harder for others to intercept or tamper with.

HTTPS is important for protecting sensitive information like passwords and credit card numbers.

SSL (Secure Sockets Layer):

SSL is a technology used to establish a secure encrypted connection between a web server and a browser.

It ensures that data transmitted between the server and the browser remains private and integral.

SSL is now largely replaced by its successor, TLS (Transport Layer Security), but the term SSL is still commonly used to refer to both technologies.

2) HTTP Methods, Headers, Body, Status Codes, Cookie

HTTP Methods:

HTTP methods are like verbs that tell the server what action to perform with a resource.

Common methods include:

GET: Retrieve data from a server (like loading a web page).

POST: Send data to a server (like submitting a form).

PUT: Update a resource on the server.

DELETE: Remove a resource from the server.

PATCH: It's like using a targeted update to change specific parts of something without affecting the whole thing. For example, you might update just the "firstName" of a user profile without changing other details like their age or email address.

HTTP Headers:

Headers provide additional information about the request or response.

Examples include:

Accept: Specifies the media types that the client can understand and handle.

Example: Accept: application/json (Indicates the client accepts JSON responses).

Content-Type: Indicates the media type of the request payload or response body.

Example: Content-Type: application/json (Indicates the payload sent or received is in JSON format).

Cookie: Stores stateful information on the client side that is sent with every request to the same domain.

Example: Cookie: sessionId=abc123 (Stores a session ID that persists across requests).

Authorization: Provides credentials (such as tokens or keys) to access protected resources.

Example: Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c (Provides a bearer token for API authentication).

HTTP Body: The body contains the data being sent with the request (for methods like POST or PUT) or the data returned in the response.

It can be text, JSON, XML, or other formats depending on the application's needs.

HTTP Status Codes:

Status codes indicate the outcome of a HTTP request.

Examples include:

- **200 OK:** Successful request.
- **201 Created:** Resource created successfully.
- **202 Accepted:** Request accepted for processing.
- **204 No Content:** Request processed successfully with no content returned.
- **207 Multi-Status:** Multiple operations each have their own status code.
- **301 Moved Permanently:** Resource has been permanently moved.
- **302 Found (Moved Temporarily):** Resource temporarily moved.
- **307 Temporary Redirect:** Temporary redirect similar to 302.
- **400 Bad Request:** Invalid request syntax or parameters.
- **401 Unauthorized:** Authentication is required and missing or incorrect.
- **403 Forbidden:** Request is understood but refused due to lack of authorization.
- **404 Not Found:** Resource not found.
- **409 Conflict:** Request conflicts with current state of the resource.
- **500 Internal Server Error:** Server encountered an unexpected condition preventing request fulfillment.
- **501 Not Implemented:** Server does not support the functionality required to fulfill the request.
- **502 Bad Gateway:** Server received an invalid response from an upstream server.
- **503 Service Unavailable:** Server is temporarily unable to handle the request due to overload or maintenance.

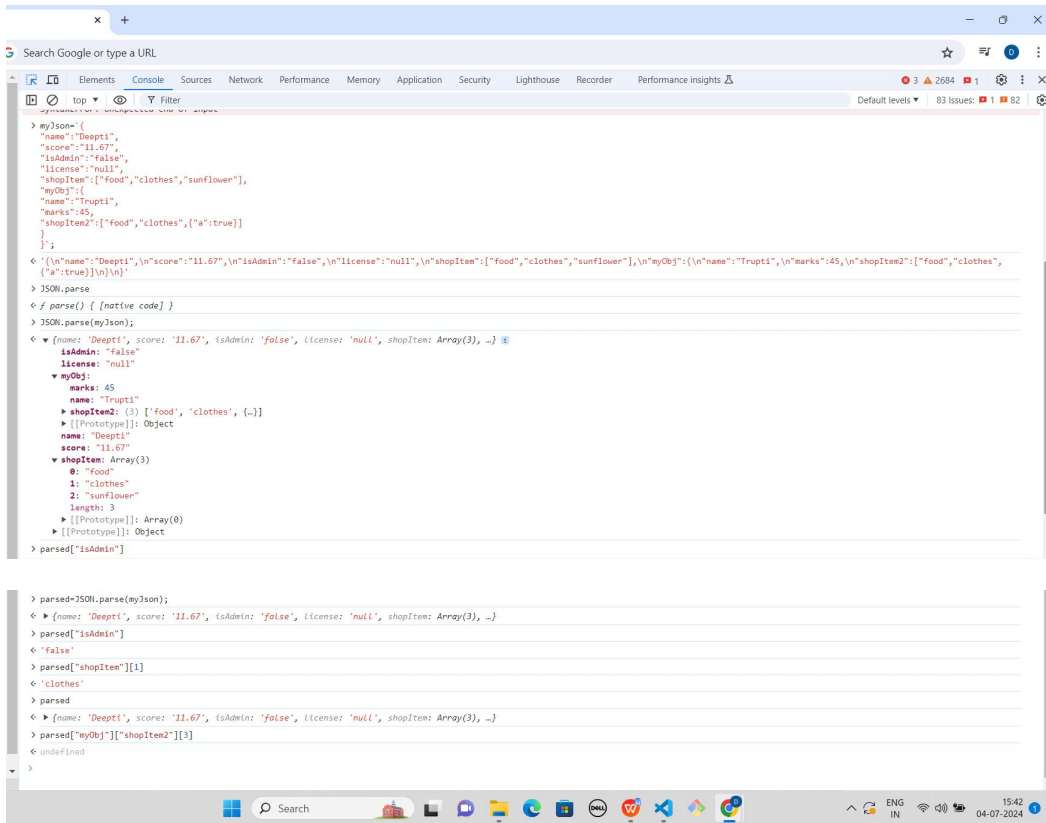
Cookie:

Cookies are small pieces of data stored on the client's side by the web browser.

They are used to remember stateful information (like login sessions or user preferences) across different pages or visits to the website.

Cookies can store information like usernames, shopping cart contents, or language preferences.

JSON



```
> myJson={
  "name":"Deepti",
  "score":"11.67",
  "isAdmin":"false",
  "license":"null",
  "shopItem":["food","clothes","sunflower"],
  "myObj":{
    "name":"Trupti",
    "marks":45,
    "shopItem2":["food","clothes",{"a":true}]
  }
};

< {
  "name": "Deepti",
  "score": "11.67",
  "isAdmin": "false",
  "license": "null",
  "shopItem": ["food", "clothes", "sunflower"],
  "myObj": {
    "name": "Trupti",
    "marks": 45,
    "shopItem2": ["food", "clothes", {"a": true}]
  }
}

> JSON.parse
< f parse() { [native code] }
> JSON.parse(myJson);
< {name: 'Deepti', score: '11.67', isAdmin: 'false', license: 'null', shopItem: Array(3), ...}
  isAdmin: "false"
  license: "null"
  myObj: {
    marks: 45
    name: "Trupti"
    shopItem2: (3) ["food", "clothes", {...}]
  }
  name: "Deepti"
  score: "11.67"
  shopItem: Array(3)
    0: "food"
    1: "clothes"
    2: "sunflower"
    length: 3
  }
  }
  parsed["isAdmin"]

> parsed=JSON.parse(myJson);
< {name: 'Deepti', score: '11.67', isAdmin: 'false', license: 'null', shopItem: Array(3), ...}
> parsed["isAdmin"]
< 'false'
> parsed["shopItem"][1]
< 'clothes'
> parsed
< {name: 'Deepti', score: '11.67', isAdmin: 'false', license: 'null', shopItem: Array(3), ...}
> parsed["myObj"]["shopItem2"][3]
< undefined
```

Authentication:

- **Purpose:** Confirms a user's identity.
- **Process:** User logs in with their username and password on a service (like Google, Facebook).
- **Result:** Service provides an access token, proving the user's identity.

Authorization:

- **Purpose:** Determines what a user can do after they're authenticated.
- **Process:** Uses the access token from authentication to specify allowed actions.
- **Example:** Accessing profile info, posting on behalf of the user, etc.

Web Server:

- **Purpose:** Handles HTTP requests from clients (like web browsers).
- **Function:** Responds with static content (HTML, CSS, images) or forwards dynamic requests to application servers.
- **Examples:** Apache HTTP Server, Nginx.

Application Server:

- **Purpose:** Executes application logic and business processes.
- **Function:** Generates dynamic content, interacts with databases, performs calculations, and more complex tasks.
- **Examples:** Java EE servers (e.g., Apache Tomcat, WildFly), Node.js, Ruby on Rails.

Key Difference: Web servers handle basic HTTP requests and serve static content, while application servers run the backend logic, handle complex tasks, and generate dynamic content for web applications.

Nouns in URLs:

- **Purpose:** Used to identify what the API is about—things or entities.
- **Example:**
 - /users (for accessing user information)
 - /products (for managing product data)
 - /orders (for handling orders)
- **Usage:** Nouns represent the main entities or collections of data your API deals with. They help organize and access specific types of information.

Verbs in URLs:

- **Purpose:** Used to show actions or operations you want to perform on those entities.
- **Example:**
 - /create-user (to add a new user)
 - /update-product/{id} (to modify a specific product)
 - /delete-order/{id} (to remove a particular order)
- **Usage:** Verbs indicate what you want to do with the nouns (entities). They specify actions like creating, updating, or deleting data.

Example Comparison:

- **Nouns:** /users (to list all users), /users/{id} (to get a specific user)
- **Verbs:** /create-user (to add a new user), /update-user/{id} (to modify a user)

CORS (Cross-Origin Resource Sharing)

CORS (Cross-Origin Resource Sharing) headers allow a server to specify who can access its resources from a different origin (domain) than the one that served the initial request.

CORS Headers Explanation:

- **Purpose:** Prevents web pages from making unauthorized cross-origin requests.
- **Headers:**
 - **Access-Control-Allow-Origin:** Specifies which origins can access the resource.
 - **Access-Control-Allow-Methods:** Defines the HTTP methods (like GET, POST) allowed when accessing the resource.
 - **Access-Control-Allow-Headers:** Lists the headers allowed when making the actual request.
 - **Access-Control-Allow-Credentials:** Indicates whether the request can include credentials (like cookies, authorization headers).
 - **Access-Control-Expose-Headers:** Specifies headers exposed to the browser in the response.
 - **Access-Control-Max-Age:** Specifies how long the results of a preflight request (OPTIONS request) can be cached.