### Learn about what is a microservice?

Microservices are an architectural approach to developing software applications as a collection of small, independent services that communicate with each other over a network. Instead of building a monolithic application where all the functionality is tightly integrated into a single codebase, microservices break down the application into smaller, loosely coupled services.

Microservice is a small, loosely coupled service that is designed to perform a specific business function and each microservice can be developed, deployed, and scaled independently.

## About what is a web service?

A **web service** is a standardized software system designed to enable interoperable machine-to-machine interaction over a network, often the Internet. It allows applications, which may be written in different programming languages and run on various platforms, to communicate and exchange data using open protocols and formats like HTTP, XML, or JSON. Essentially, a web service provides specific functionality or data to a client application by accepting requests and returning responses via standardized web protocols.

## What is Cookies?

A **cookie** (also called an HTTP cookie, web cookie, or browser cookie) is a small piece of data that a web server sends to a user's web browser. The browser stores this data locally and sends it back to the server with each subsequent request to that website. Cookies are primarily used to **maintain state and remember information about the user across browsing sessions**, since the HTTP protocol itself is stateless and does not retain information between requests.

Cookies typically contain small text files with name-value pairs that can store session identifiers, user preferences, login status, or other data needed by the website to provide a personalized experience or keep users logged in. For example, cookies enable a website to remember the contents of your shopping cart, or keep you logged in as you navigate different pages

## What is Server?

A **server** is a computer system or software that provides services, resources, or data to other computers, known as clients, over a network. Specifically, in the context of the web, a **web server** is a type of server designed to store, process, and deliver web content to users. It accepts requests from web browsers (clients) via the HTTP or HTTPS protocols and responds by sending the requested web pages, images, videos, or data so the user can view them in their browser.

In simple terms, a server acts as a centralized machine or program that waits for requests from clients and then responds with the requested information or service. For example, when you enter a website URL, your browser sends a request to a web server, which then processes that request and sends back the webpage to display.

## What is LocalHost ?

**Localhost** is a hostname that refers to the local computer or machine you are currently using. It is essentially a way for your computer to communicate with itself. When you use "localhost," you are directing network requests to your own machine rather than reaching out over an external network or the internet.

Technically, localhost corresponds to the IP address **127.0.0.1**, which is a special reserved address used for loopback networking. This means any network traffic sent to 127.0.0.1 stays within your computer and does not go out to any external network.

## What is Domain?

A **domain** (or domain name) is a human-readable, unique address used to identify and access websites and other services on the internet. Instead of having to remember complex numerical IP addresses (which computers use to identify each other), users can type a domain name like "google.com" or "example.org" in their browser to reach a website directly.

The **domain name system (DNS)** translates these domain names into the corresponding IP addresses, enabling proper routing of internet traffic. Domains represent administrative namespaces in the internet addressing system and are organized hierarchically, with top-level domains (TLDs) such as .com, .org, or country codes like .uk, followed by subdomains and specific host names.

## what is Endpoint in the URL?

An endpoint in a URL is the specific digital location or path on a server where an API or web service receives requests to access a particular resource or perform a specific function. It represents the exact URL or part of the URL after the domain, which directs the client to the exact service or data on the server.

For example, in the URL:

https://api.example.com/v1/users

the endpoint is /v1/users. This path tells the server to interact with the "users" resource of version 1 of the API.

## What is the Difference between HTTP and HTTPS?

The primary difference between HTTP (Hypertext Transfer Protocol) and HTTPS (Hypertext Transfer Protocol Secure) is that HTTPS includes encryption and security features that HTTP lacks.

Key differences include:

Security:

HTTP transmits data in plain text, making it vulnerable to interception or attacks such as eavesdropping or data tampering.

HTTPS encrypts data sent between the client (browser) and server using TLS (Transport Layer Security) or SSL (Secure Sockets Layer), ensuring that the information remains private and integral during transmission. It also provides server authentication using digital certificates.

Port number:

HTTP uses port 80 by default.

HTTPS uses port 443 by default

## What is CRUD? and which methods are used for this and give some details about those methods.

CRUD is an acronym that stands for Create, Read, Update, and Delete—the four fundamental operations used for managing persistent data in databases, applications, or APIs. CRUD operations are at the core of how software interacts with data, enabling users and systems to add, retrieve, modify, and remove information as needed.

Overview of CRUD Operations:

Create:

Adds new entries or data into a system. For example, registering a new user or adding a product.

In REST APIs: Typically implemented with the HTTP POST method.

Read:

Retrieves or views existing data from the system, such as displaying user information or listing products.

In REST APIs: Uses the HTTP GET method.

Update:

Modifies existing information. An example is editing a user’s email or updating an item’s price.

In REST APIs: Handled by HTTP PUT (to replace/update an entire resource) or HTTP PATCH (to partially update a resource).

Delete:

Removes data from the system, such as deleting a user account or removing an item from inventory.

In REST APIs: Uses the HTTP DELETE method.

|  |  |  |
| --- | --- | --- |
| **CRUD Operation** | **HTTP Method** | **Details / Common Usage Example** |
| Create | POST | Submits data to be processed; e.g., add a new resource/item |
| Read | GET | Retrieves data or specific records; e.g., fetch user details |
| Update | PUT/PATCH | PUT replaces a record; PATCH updates part of a record |
| Delete | DELETE | Removes a resource by identifier; e.g., delete user by ID |

## What is Payload, Header, Status code?

Payload, Header, and Status Code are fundamental components of HTTP communication between clients and servers, especially in web services and APIs.

**Payload**:

The payload is the actual data being sent or received in an HTTP request or response. It is often located in the body of the message. Unlike headers or metadata, the payload carries the meaningful content that the client wants to transmit to the server or the server returns back to the client. For example, in a POST request to create a user, the payload might contain the user's details in JSON format. Payloads can come in various formats like JSON, XML, or plain text and represent the core information being communicated. Without a payload, the communication might be functional but generally lacks meaningful data exchange.

**Header**:

Headers are key-value pairs sent at the beginning of HTTP requests and responses. They provide metadata about the request or response but do not contain the main payload data itself. Headers include information such as content type (Content-Type), authentication tokens (Authorization), caching directives, accepted data formats (Accept), and more. They help both client and server understand how to process the payload and manage the communication (e.g., signaling what type of data is being sent or expected).

**Status Code:**

The status code is part of the HTTP response sent from the server back to the client, indicating the result of the request. It is a three-digit number where each range has a specific meaning:

1xx: Informational responses

2xx: Successful responses (e.g., 200 OK)

3xx: Redirection messages

4xx: Client errors (e.g., 404 Not Found, 401 Unauthorized)

5xx: Server errors (e.g., 500 Internal Server Error)

## What is Load Balancer? (Research about this)

A **load balancer** is a device or software that distributes incoming network traffic or workloads across multiple servers or resources. Its main purpose is to optimize resource use, maximize throughput, reduce response time, and ensure high availability and reliability of applications or services by preventing any single server from becoming overwhelmed.

## What is Client and Server?

A **client** is a computer, device, or software application that initiates requests for services or resources from another computer or program. It is typically user-facing and interacts with the end user by sending requests to a server.

A **server** is a computer or software that provides services, resources, or data to clients by accepting and processing their requests. Servers host resources such as websites, databases, or applications and deliver responses to the clients over a network.

Together, in the **client-server architecture**, the client sends a request to the server, which processes that request and sends back the requested information or service. This architecture centralizes resource management and allows multiple clients to access shared resources efficiently.