Client-Server Architecture

Basic Concept

The client-server model divides work between two types of systems:

1. clients (which request services) : The client initiates communication by sending requests
2. servers (which provide services) : The server responds with the requested data or performs the requested action.

Key Components

Client - The frontend that users interact with. This could be a web browser, mobile app, desktop application, or even another server. Clients are typically responsible for presentation logic and user interface.

Server - The backend that processes requests and manages resources like databases, files, or business logic. Servers listen for incoming requests, process them, and send back responses.

Network - The communication channel (usually the internet or local network) that connects clients and servers, typically using protocols like HTTP/HTTPS, TCP, or WebSockets.

How the Browser Interacts With the Servers?

The process of interacting with servers through a browser involves several steps:

1. User Enters the URL (Uniform Resource Locator):
The user types a website address (e.g., www.example.com) into the browser's address bar.

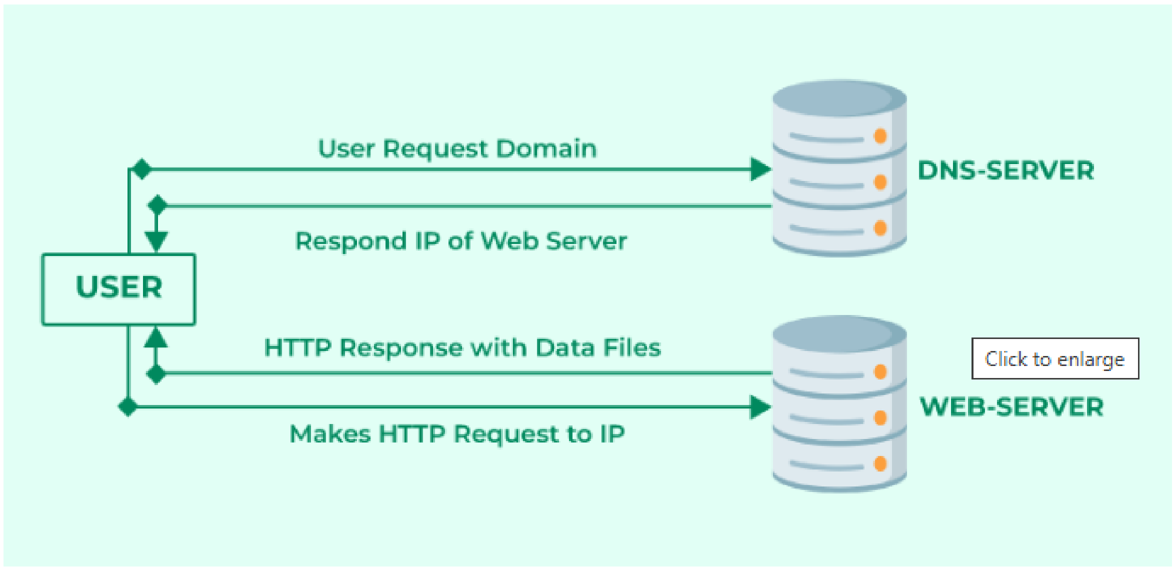
2. DNS (Domain Name System) Lookup:
The browser contacts a DNS server to convert the domain into an IP address.

3. Establishing a Connection:
The browser sends an HTTP/HTTPS request to the server using the resolved IP address.

4. Server Responds:
The server sends back website files (HTML, CSS, JavaScript, images).

5. Browser Renders the Webpage

- **DOM interpreter:** Processes HTML to structure the page.
- **CSS interpreter:** Applies styles
- **JavaScript Engine:** Adds interactivity (using JIT compilation for performance).



Advantages

- Centralized data management : Easy to maintain and back up data
- Better security (sensitive logic stays on the server)
- Cost Efficiency : Clients require less processing power
- Scalability : Servers and clients can scale independently
- Clients can be lightweight since heavy processing happens server-side

Disadvantages

- Server becomes a single point of failure
- Client vulnerability : Risk of malware if servers distribute unsafe files
- Can create bottlenecks if the server gets overwhelmed; Susceptible to DDoS attacks
- Network dependency (clients can't work if connection is lost)
- Scaling challenges as client numbers grow
- Data Spoofing : Unprotected data can be tampered within transit
- MITM Attacks : Unsecured connections can be intercepted by attackers

Variations

Two-tier - Client communicates directly with database server (common in older applications)

Three-tier - Client → Application Server → Database Server (most modern web apps)

N-tier - Multiple layers of servers handling different responsibilities (presentation, business logic, data access, etc.)