🌐 Client–Server Model in System Design

1. What is the Client–Server Model?

The **Client–Server Model** is a **distributed application structure** where:

- **Client** = A program or device that **requests** services or resources.
- **Server** = A program or machine that **provides** services or resources.

👉 In simple words:

- The **client asks** ("Can I have this data?").
- The **server responds** ("Here's the data!").

This model is the foundation of **web applications, databases, email systems, and most modern networks**.

2. Characteristics

- **Two roles**: Clients request, servers respond.
- **Request/Response cycle**: Clients initiate communication, servers reply.
- **Networked**: Usually over TCP/IP protocols (e.g., HTTP, gRPC).
- **Centralized server**: Multiple clients connect to one server (or a cluster).

---

3. Components

🔹 Client

- Runs on user devices (PC, phone, browser, app).
- Sends requests to server (via HTTP, WebSockets, etc.).
- Displays data to the user.
- Examples: Chrome browser, Slack app, mobile banking app.

🔹 Server

- Runs on powerful machines or cloud infrastructure.
- Listens for incoming requests.
- Processes data (business logic).
- Responds with results.
- Examples: Web servers (Nginx, Apache), API servers, database servers.

---

4. How it Works (Step-by-Step)

Example: Opening a website in your browser.

1. **Client sends request**
    - Browser (client) → “GET /index.html” via HTTP.

2. **Server processes request**
    - Web server looks for index.html.
3. **Server sends response**
    - Sends back HTML, CSS, JS to browser.
4. **Client renders data**
    - Browser displays the webpage to user.

👉 Communication follows the **request–response** cycle.

---

5. Types of Client–Server Systems

1. **One-tier (standalone)**
    - Client & server on the same machine.
    - Example: MS Excel, desktop games.
2. **Two-tier**
    - Client talks directly to server.
    - Example: Web browser ↔ Web server.
3. **Three-tier**
    - Client ↔ Application Server ↔ Database Server.
    - Example: E-commerce site (UI → Business logic → Database).

4. **N-tier / Multi-tier**
    - Multiple layers (load balancer, API gateway, microservices, DB).
    - Example: Netflix, Amazon, Google systems.

---

6. Advantages

✅ Centralized management → Server updates affect all clients.
✅ Scalability → Multiple clients can connect to one server.
✅ Security → Access control at the server.
✅ Easier maintenance → Server upgrades apply system-wide.

---

7. Disadvantages

⚠️ **Single point of failure** → If server crashes, all clients are affected.
⚠️ **Network dependency** → Clients need connectivity to access server.
⚠️ **Server bottleneck** → Too many clients → server overload.

---

8. Real-World Examples

- **Web Browsing**:
  Browser (client) → Google server (server).
- **Email**:
  Email client (Outlook, Gmail app) ↔ Mail server (SMTP, IMAP, POP3).
- **Banking Apps**:
  Mobile app (client) ↔ Banking backend (server).

- **Streaming Services**:
  Netflix app (client) ↔ Netflix servers (server).

---

9. Diagram (Conceptual Flow)

```
[ Client Devices ]      <-- Laptop, Mobile, Browser
        |
        | Request (HTTP, gRPC, etc.)
        v
   [ Server ]           <-- Web Server, API Server, DB
        |
        | Response (HTML, JSON, Video, etc.)
        v
[ Client Application ]  <-- User sees the result
```