1. How does client server architecture works?

Client-server architecture is a way of designing network based-applications where two main components clients and servers communicate with each other.

The client-server architecture work as given below:

- The client, such as a workstation or smartphone, connects to the network using a physical or wireless LAN or internet connection.
- The client sends various requests to the server in order to submit, retrieve, or modify the data located on the server.
- The server processes each client request.

  For example: - Web Browsing when you type a website URL (e.g., www.google.com), your browser (client) requests the page from Google's server, which then sends back the webpage data, Email Service your email app (client) connects to an email server to send or receive messages.

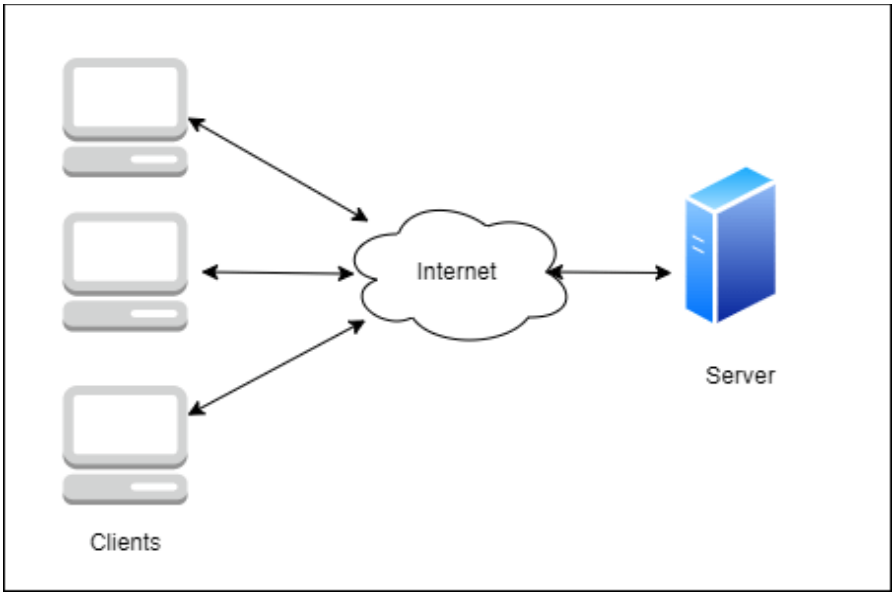

fig of: Client-server architecture

2. What are the factors to be considered when designing a software?

When designing a software the factors to be considered are given below:

- **Requirements Analysis**

  Understand the business needs and user expectations. Define functional (what the

software does) and non-functional (performance, security, etc.) requirements.

- **Scalability**

  Ensure the software can handle increased workload (more users, data, etc.). Use modular architecture to allow easy expansion.

- **Performance**

  Optimize code for speed and efficiency. Reduce latency and optimize response time.

- **Security**

  Implement authentication and authorization (e.g., passwords, tokens). Protect against cyber threats (e.g., SQL injection, XSS, CSRF). Encrypt sensitive data.

- **Maintainability**

  Write a clean, well-designed code for easy updates. Use version control for tracking changes.

- **User Experience (UX) and User Interface (UI)**

  Ensure a simple, intuitive, and attractive design. Follow usability principles for smooth debugging.

- **Modularity and Reusability**

  Break the system into independent modules for easier debugging. Reuse components to reduce redundancy.

- **Compatibility and Integration**

  Ensure the software works across different devices and operating systems. Support integration with third-party services and APIs.

- **Testing and Quality Assurance**

  Perform unit testing, integration testing, and system testing. Automate testing where possible to improve efficiency.

- **Deployment and Continuous Integration**

  Plan for smooth deployment. Use CI/CD pipelines for automated updates and bug fixes.

3. Why do we really need network programming tools and platform? Explain some of them.

Network programming tools and platforms are essential because they simplify the process of developing, testing, and managing applications that communicate over a network. Without

these tools, developers would have to handle low-level networking tasks manually, making development more error-prone and complex.

Reasons for using network programming tools and platforms:

- **Simplifies Development** – Abstracts low-level networking details, allowing developers to focus on logic instead of protocols.
- **Enhances Security** – Provides built-in security features like encryption, authentication, and access control.
- **Improves Performance** – Optimizes data transfer, reduces latency, and manages network congestion.
- **Enables Scalability** – Supports cloud integration and distributed computing for handling large-scale applications.
- **Facilitates Debugging and Monitoring** – Offers tools to analyze network traffic, detect errors, and optimize communication.

Some popular network programming tools and platform are:

- **Sockets API**

  Used for low-level network communication in C, C++, Python, Java etc. Supports TCP and UDP connections.
- **Wireshark**

  A network protocol analyzer that captures and inspects packets. Help debug networking issues and analyze communication between applications.
- **Postman**

  A GUI–based tool for API developing and testing. Allows sending HTTP requests, viewing responses, and debugging APIs.