UNIT II

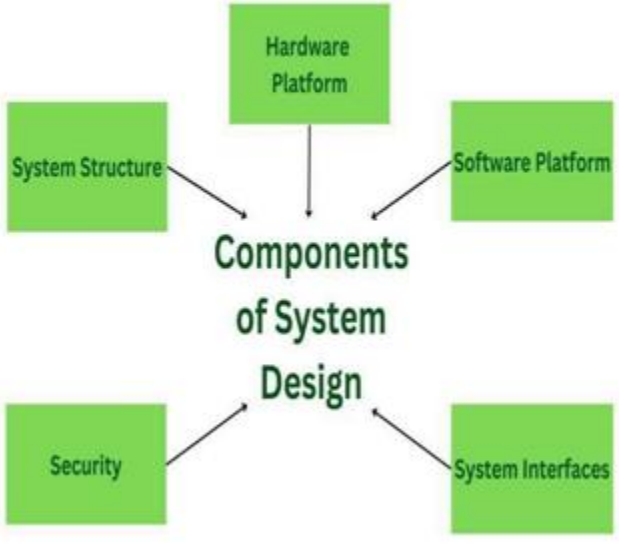
SOFTWARE DESIGN AND UML DIAGRAMS

Design Principles (Modularity, Reusability, Abstraction),UML Diagrams: Use Case, Class, Activity, Sequence, Introduction to Design Patterns (Singleton, Factory, MVC),Building Simple System Architecture (Layered & Client-Server).

BUILDING SIMPLE SYSTEM ARCHITECTURE

Architecture is a critical aspect of designing a system, as it sets the foundation for how the system will function and be built.

- It is the process of making high-level decisions about the system, including the selection of hardware and software components, design of interfaces, and the overall system structure.
- In order to design a good system architecture, it is important to make decisions based on the specific requirements and constraints of the system.
- It is also important to consider the long-term maintainability of the system and to make sure that the architecture is flexible and scalable enough to accommodate future changes and growth.



Hardware Platform: Servers, storage devices, and network infrastructure.

- Software Platform: Operating System, application servers, and other software components that run on the hardware.
- System interfaces: System interfaces include the APIs and user interfaces used to interact with the system.
- System Structure: Overall organization of the system, including the relationship between different components and how they interact with each other.
- Security: To protect the system and its users from malicious attacks and unauthorized access.

Types of Architecture in system design

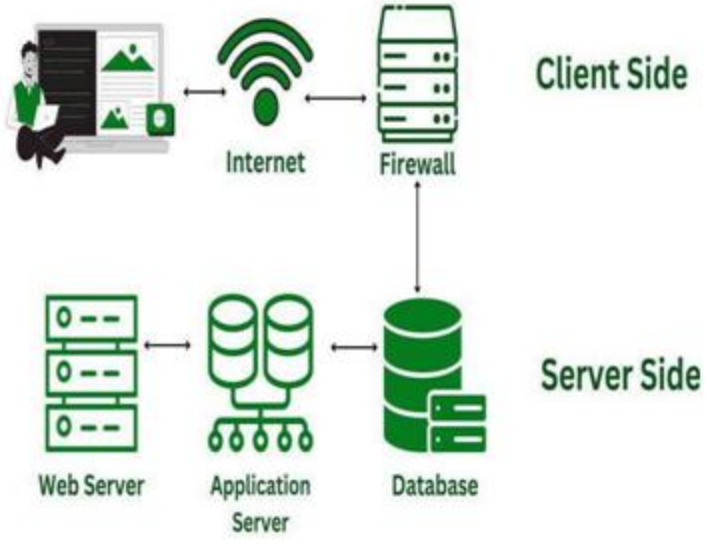
There are several different architectural styles that can be used when designing a system, such as:

- ❖ Monolithic architecture: This is a traditional approach where all components of the system are tightly coupled and run on a single server. The components are often tightly integrated and share a common codebase.
- ❖ Microservices architecture: In this approach, the system is broken down into a set of small, independent services that communicate with each other over a network. Each service is responsible for a specific task and can be developed, deployed, and scaled independently. This allows for greater flexibility and scalability, but also requires more complexity in managing the interactions between services.
- ❖ Event-driven architecture: This approach is based on the idea of sending and receiving events between different components of the system. Events are generated by one component, and are consumed by other components that are interested in that particular event. This allows for a more asynchronous and decoupled system.
- ❖ Serverless architecture: this approach eliminates the need for provisioning and managing servers, by allowing to run code without thinking about servers. In this way, the cloud provider is responsible for scaling and maintaining the infrastructure, allowing the developer to focus on writing code.

When designing a system, it is important to choose an architecture that aligns with the requirements and constraints of the project, such as scalability, performance, security, and maintainability.

Example: website for an online retail store.

One example of a system design architecture is the design of a website for an online retail store. The architecture includes the following components:

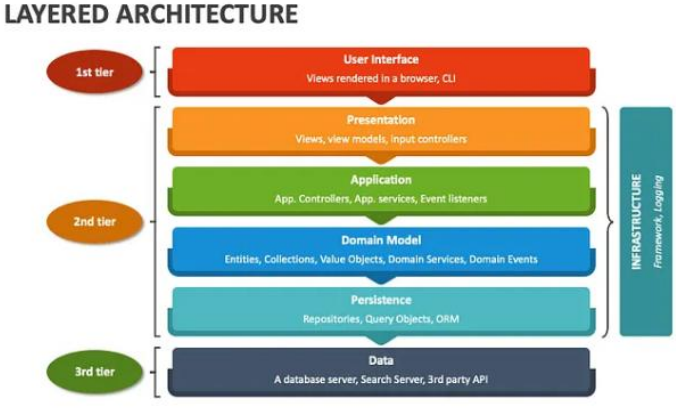


- ❖ Front-end: The user interface of the website, including the layout, design, and navigation. This component is responsible for displaying the products, categories, and other information to the user.
- ❖ Back-end: The server-side of the website, including the database, application logic, and APIs. This component is responsible for processing and storing the data, and handling the user's requests.
- ❖ Database: The component that stores and manages the data for the website, such as customer information, product information, and order information.

- ❖ APIs: The component that allows the website to communicate with other systems, such as payment systems, shipping systems, and inventory systems.
- ❖ Security: The component that ensures the website is secure and protected from unauthorized access. This includes measures such as SSL encryption, firewalls, and user authentication.
- ❖ Monitoring and analytics: The component that monitors the website's performance, tracks user behavior, and provides data for analytics and reporting

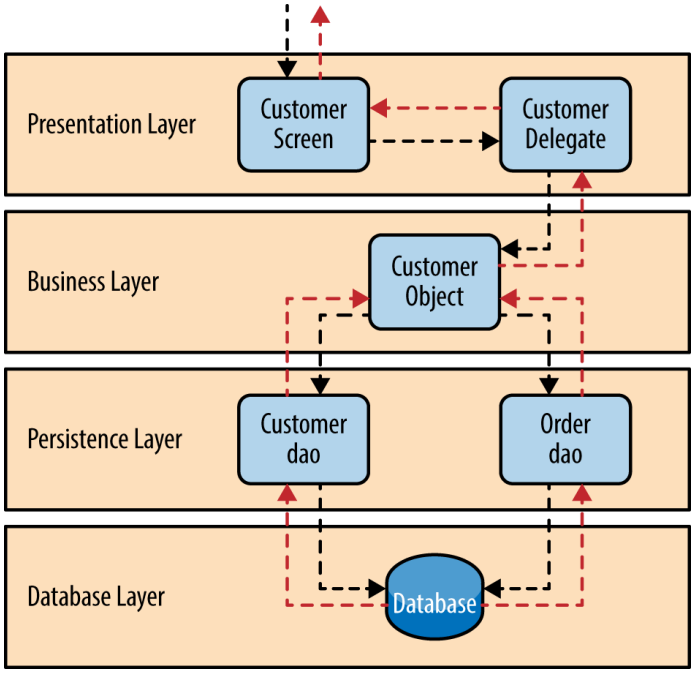
Layered architecture:

- ❖ A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- ❖ At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- ❖ Intermediate layers to utility services and application software functions.



- ❖ One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organization for Standardization) communication system.
- ❖ The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. This pattern is the de facto standard for most Java EE applications and therefore is widely known by most architects, designers, and developers.
- ❖ The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.
- ❖ **Layered architectures** are widely used in software development and are considered to be the most prevalent and commonly **utilized architectural framework**.

- ❖ It is an **architectural pattern** made up of numerous **independent horizontal layers** that work together to form a **single software unit**. This pattern is also known as n-tier architecture.
- ❖ A layer is a logical division of parts or programme code. This architecture typically places related or comparable components on the same tiers. Every layer is unique and affects a different aspect of the entire system.

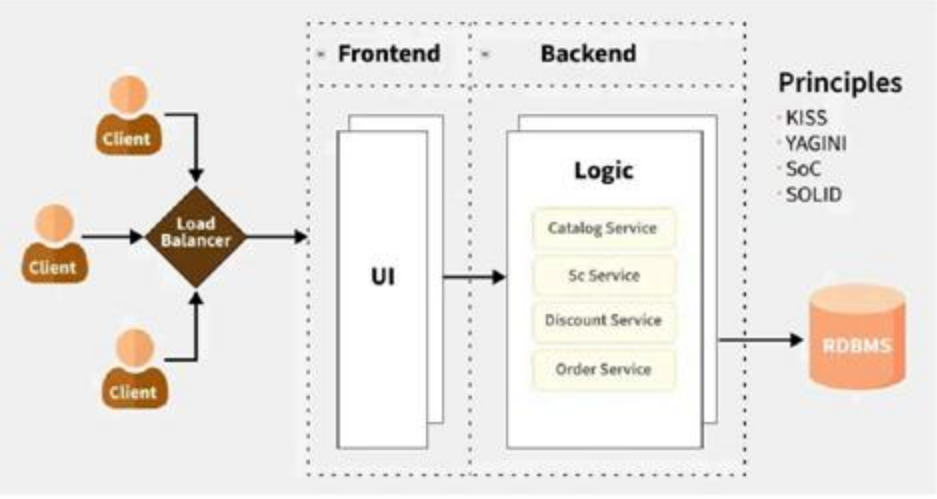


- **Presentation Layer (UI)** : This is where the user interacts with the application. It handles user input and displays information.
- **Application Layer (Business Logic)** : This layer contains the core logic of the application, such as processing user requests and managing business rules.
- **Data Access Layer (Database)** :Data is retrieved and stored. It interacts with the database, ensuring separation of data concerns.

Pattern Description

- Within the application, each layer of the **layered architecture pattern** is responsible for a particular task.
- For instance, the user interface and browser communication logic would be handled by the presentation layer, whilst the **business layer** would be in charge of carrying out the specific business rules related to the request.
- Each layer of the architecture creates an **abstraction** around the work required to complete a specific business requirement.
- For instance, the presentation layer just needs to display customer data on a screen in a specific way; it is not required to understand or worry about how to obtain **customer data**.
- The business layer only needs to obtain the data from the **persistence layer, apply business logic to the data** (e.g., calculate values or aggregate data), and then pass that information up to the presentation layer. In a similar manner, the business layer need not worry about how to format customer data for **display on a screen** or even where the customer data is coming from.
- **Separating** concerns among components is one of the layered architecture pattern's strong points. Components inside that layer deal with only logic specific to a certain layer.
- For instance, the presentation layer's components only deal with **presentation logic**, whereas the business layer's components only deal with business logic.

- Due to the clearly defined component interfaces and constrained component scope, this type of component classification makes it simple to **incorporate efficient roles** and responsibility models into your architecture. It also makes it simple to develop, test, govern, and maintain applications using this **architecture pattern**.



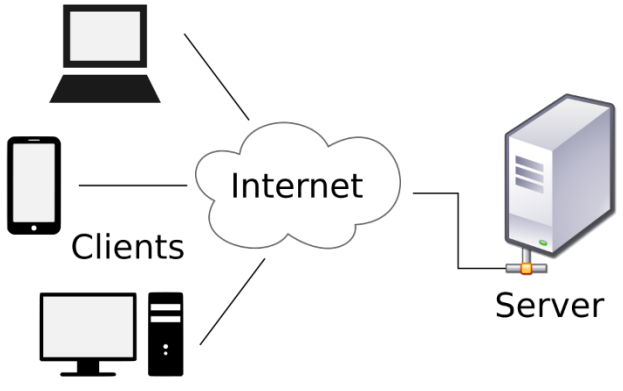
CLIENT-SERVER MODEL

- The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.

How the Client-Server Model works ?

Client:

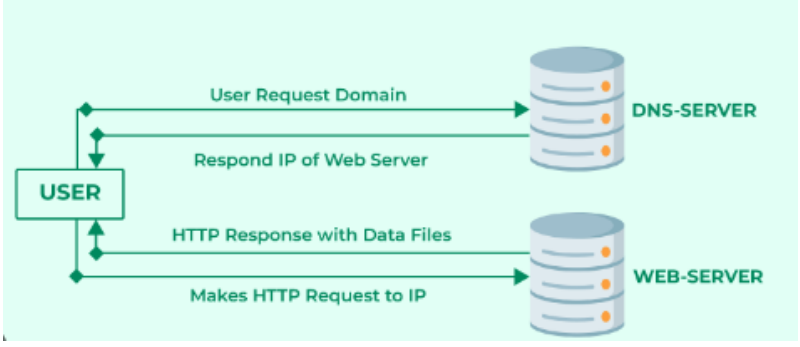
When we talk the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).



Servers:

Similarly, when we talk the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to

particular services. So, its basically the **Client** requesting something and the **Server** serving it as long as its present in the database.



Client-Server Architecture - System Design

Client—server architecture is a widely used system design where multiple clients request services or resources from a central server. The server handles processing, data storage, and resource management, while clients focus on user interaction, enabling efficient, scalable, and organized distributed systems.

- Clients send requests and receive responses using protocols like HTTP/HTTPS or SQL.
- The server centralizes data management, complex processing, and storage.
- The architecture is scalable by upgrading servers or adding multiple servers.
- Commonly used in web applications, databases, and email systems.

Design Principles for Effective Client-Server Architecture

- Designing an effective client-server architecture involves several key principles that ensure the system is robust, scalable, secure, and efficient.

Here are the main design principles:

- Modularity: Modularity is a design principle that structures a system into independent, well-defined components, each handling a specific responsibility.
- Separation of Concerns: Divide the system into distinct modules, such as the client, server, and database, each responsible for specific tasks. This separation simplifies development, testing, and maintenance.
- Reusability: Design components that can be reused across different parts of the system or in different projects.

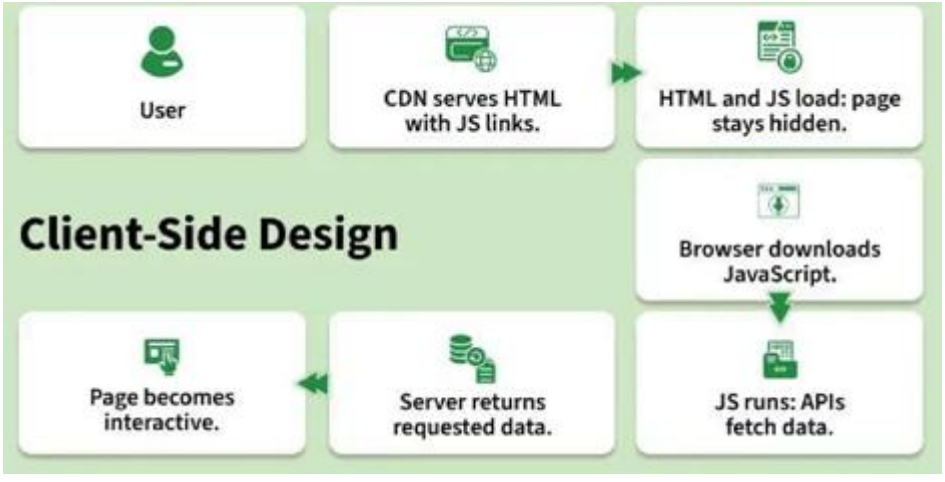
1. Scalability:
   - Scalability is the ability of a system to handle increasing workloads efficiently by adapting its resources.
   - The system achieves horizontal scalability by handling increased load through adding more servers or instances.
   - The system supports vertical scalability by upgrading server hardware such as CPU, memory, and storage.
2. Reliability and Availability:
   - Reliability ensures that a system continues to operate correctly and efficiently even in the presence of failures or high load.
   - The system improves reliability through redundancy by using multiple servers to remain operational during failures.
   - The system enhances performance using load balancing by distributing client requests across servers to avoid bottlenecks.
3. Performance Optimization:
   - Efficient Communication: Optimize the communication protocols and data exchange formats to reduce latency and bandwidth usage.

- Caching: Use caching mechanisms to store frequently accessed data closer to the client to improve response times.

4. Security:
   - Authentication and Authorization: Ensure that only authorized clients and users can access the server and its resources.
   - Encryption: Use encryption protocols (e.g., SSL/TLS) to secure data transmission between clients and servers.
   - Regular Audits: Conduct regular security audits and updates to identify and address vulnerabilities.
5. Maintainability:
   - Clean Code: Write clear, well-documented, and maintainable code to simplify debugging and future enhancements.
   - Version Control: Use version control systems to manage changes in the codebase and coordinate among development teams.
6. Interoperability:
   - Standard Protocols: Use standard communication protocols (e.g., HTTP/HTTPS, REST, SOAP) to ensure compatibility between different clients and servers.
   - Platform Independence: Design the system to support multiple platforms and devices, allowing various clients to interact with the server.

Client Side Design:

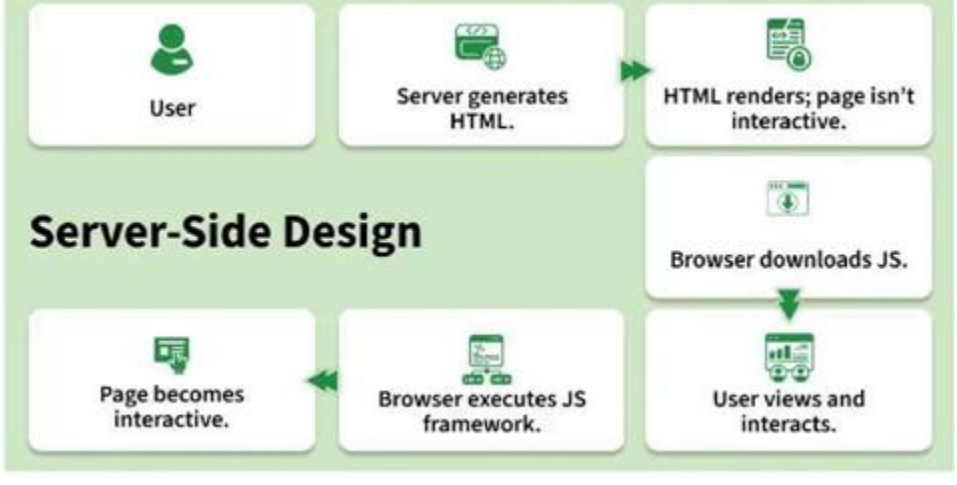
- User Requests a Website: The process begins when a user requests a website by entering a URL in the browser or clicking a link.
- CDN Serves HTML Files: A Content Delivery Network (CDN) quickly serves the HTML files, which contain links to the JavaScript files needed for the website.
- Browser Downloads HTML and JavaScript: The browser first downloads the HTML file and then proceeds to download the linked JavaScript files. During this phase, the site may not be fully visible to the user, as it waits for the JavaScript to render the content.
- Browser Downloads the JavaScripts: The browser continues to download the JavaScript files specified in the HTML.
- JavaScript Execution and API Calls: Once downloaded, the JavaScript is executed. During execution, the JavaScript makes API calls to the server to fetch the necessary data. At this stage, placeholders may be visible to the user while the actual data is being fetched.
- Server Responds with Data: The server responds to the API requests with the required data. This data is sent back to the client-side application.
- Data Fills Placeholders: The data fetched from the APIs fills the placeholders in the client interface. The page becomes fully interactive and visible to the user.



Server Side:

- User Requests a Website: The process starts when a user requests a website.
- Server Creates "Ready to Render" HTML Files: The server processes the request and generates HTML files that are ready to be rendered.
- The Browser Can Quickly Render the HTML but the Site Isn't Interactive: The browser receives and renders the HTML quickly, allowing the user to see the content immediately, but the site is not yet interactive.
- The Browser Downloads the JavaScript: The browser then downloads the necessary JavaScript files.
- The User Can View Content and the Interactions Can Be Recorded: The user can view the content while the JavaScript is being downloaded. User interactions can be recorded even though the site is not fully interactive yet.
- The Browser Executes the JS Framework: Once the JavaScript is downloaded, the browser executes the JavaScript framework (e.g., React, Angular).

- The Recorded Interactions Can Be Executed and the Page Is Now Interactive: The recorded interactions are executed, and the page becomes fully interactive.



How the browser interacts with the servers ?

There are few steps to follow to interacts with the servers a client.

1. User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.

2. **DNS Server** lookup for the address of the **WEB Server**.
3. **DNS Server** responds with the **IP address** of the **WEB Server**.
4. Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
5. Server sends over the necessary files of the website.
6. Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.

Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.