Unit 2

Issue of Web Technology

Unit 2 **Issue of Web Technology**	**3 Hrs.**
Architectural Issues of Web Layer, Tier Technology: 2-Tier, 3-Tier and n-Tier.	

Architectural Issue of Web Layer:

a. Three layers of Web:

The web layer in a software architecture refers to the part of the system that handles the presentation logic and user interface of a web application. It is responsible for serving web pages, handling user requests, and managing the interaction between the user and the underlying application logic.

The three layers within the web layer are:

a. **Content Layer:** The content layer, also known as the structure layer or markup layer, is responsible for defining the semantic structure and content of a web page. It primarily consists of HTML (Hypertext Markup Language) and focuses on organizing and representing the information on a webpage. The content layer defines the basic elements of the page, such as headings, paragraphs, lists, tables, and links. Its purpose is to provide a clear and meaningful structure to the content, ensuring accessibility and proper indexing by search engines.

b. **Style Layer:** The style layer, also referred to as the presentation layer or the design layer, handles the visual aspects and styling of a web page. It is primarily implemented using CSS (Cascading Style Sheets). The style layer defines the layout, colors, typography, and other visual properties of the content defined in the content layer. By separating the styling from the content, the style layer allows for consistent and reusable visual presentation across multiple web pages. It enables developers to easily modify the visual appearance without affecting the underlying content or behavior.

c. **Behavior Layer:** The behavior layer, also known as the interaction layer or the scripting layer, deals with the dynamic and interactive functionality of a web page. It is typically implemented using JavaScript. The behavior layer is responsible for handling user actions and events, such as button clicks, form submissions, and mouse movements. It enables interactive features like form validation, dynamic content updates, animations, and client-side interactions. The behavior layer enhances the user experience by making web pages more responsive and interactive.

By separating these concerns into distinct layers, web developers can achieve better maintainability, reusability, and flexibility in web development. Each layer focuses on a specific aspect of the presentation, allowing developers to work independently on different layers and apply changes more efficiently. Additionally, this separation promotes the use of standard technologies, enhances code readability, and facilitates collaboration among designers and developers.

Separating content, style, and behavior into distinct layers in web development offers several benefits, including:

i. Modularity and Maintainability: By separating content, style, and behavior, you create modular components that are easier to maintain and update. Changes to one layer do not require modifications in other layers, making the codebase more manageable and reducing the risk of introducing unintended side effects.

ii. Reusability: Separating these concerns promotes reusability. Content can be reused across multiple pages, and styles can be applied consistently throughout the website. Separating behavior allows for reusable scripts or components that can be applied to different parts of the site, enhancing efficiency and reducing code duplication.

iii. Collaboration: When different aspects of the web development process are separated into layers, it becomes easier for designers and developers to collaborate. Designers can focus on creating visually appealing styles without having to worry about the underlying structure or functionality. Developers can work on implementing behavior without being constrained by design decisions. This separation fosters better teamwork and smoother workflows.

iv. Performance Optimization: Separating content, style, and behavior can improve performance. By keeping styles separate, it becomes easier to implement caching mechanisms, allowing the browser to cache stylesheets and load them faster. Separating behavior allows for asynchronous loading of scripts, reducing the initial page load time.

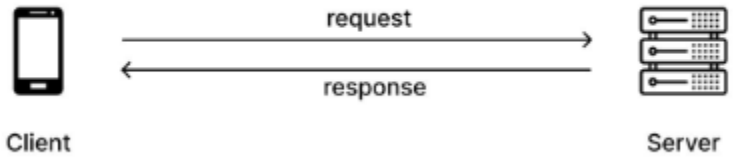
v. Accessibility and SEO: Separating content from style allows developers to focus on semantic HTML, which enhances accessibility and search engine optimization (SEO). By using appropriate HTML tags and structuring content correctly, search engines can better understand the content and improve the website's visibility in search results. Semantically structured HTML also improves accessibility for users with disabilities by enabling screen readers and assistive technologies to interpret the content accurately.

b. Hypertext Transfer Protocol:

HTTP (Hypertext Transfer Protocol) is a protocol that governs the communication between a client and a server over the internet. It is the foundation of data exchange on the World Wide Web and is widely used

for retrieving web pages, sending form data, and interacting with web applications. Port 80 is used for HTTP and 443 is used for HTTPS

(Note that: Ports allow multiple network services to run simultaneously on a single device, enabling communication between different applications or services. They play a crucial role in establishing connections and routing data between clients and servers in the web communication process.)



Here are some key aspects of HTTP:

a. **Request-Response Model**: HTTP operates based on a request-response model. The client, typically a web browser, sends an HTTP request to a server, and the server responds with an HTTP response. Each request consists of a method (such as GET, POST, PUT, DELETE) indicating the desired action, a URL specifying the target resource, and optional headers providing additional information.

b. **Stateless and Connectionless Protocol:** HTTP is stateless, meaning that each request is independent and unrelated to previous or future requests. The server does not retain any information about past requests from the same client. To maintain stateful interactions, additional techniques like cookies or session management are used.

   The term "connectionless" in the context of HTTP refers to the fact that a separate connection is established between the client and the server for each individual request-response transaction. After the response is sent back to the client, the connection is closed. This means that there is no ongoing or persistent connection between the client and server. The client initiates a new connection for each request it makes, and the server processes the request and sends the response over that connection.

c. URLs/URIs: Uniform Resource Locators (URLs) are used in HTTP to identify and locate resources on the web. A URL consists of a scheme (e.g., "http://" or "https://"), followed by the hostname, optional port number, and the path to the specific resource on the server. Uniform Resource Identifiers (URIs) encompass a broader concept that includes URLs and other identifiers.
   Example: https://www.abc.com/xyz/pqr.pdf

d. Methods: HTTP defines several request methods or verbs that indicate the desired action to be performed on a resource. The most common methods include:
   GET: Retrieves a representation of a resource.
   POST: Sends data to the server to create a new resource or trigger a specific action
   PUT: Updates or replaces a resource at a specific URL.
   DELETE: Removes the specified resource.

PATCH: Partially updates a resource.

e. **Status Codes:** HTTP responses include status codes that indicate the outcome of a request. These codes provide information about whether the request was successful, encountered an error, or requires further action. Common status codes include 200 (OK), 404 (Not Found), 500 (Internal Server Error), and many more.

f. **Headers:** HTTP requests and responses include headers, which contain additional information about the request or response. Headers can convey information such as content type, caching directives, authentication credentials, and more. They provide flexibility, control, and customization options for the communication process.

g. **Caching**: HTTP supports caching mechanisms that allow clients and intermediaries (such as web proxies) to store and reuse previously retrieved responses. Caching improves performance and reduces the need to fetch the same resource repeatedly.

h. **Security**: HTTP can be used with TLS (Transport Layer Security) or SSL (Secure Sockets Layer) to establish secure connections. This is known as HTTPS, where the communication is encrypted, ensuring confidentiality and integrity of the data exchanged between the client and server.

HTTP has evolved over time, and different versions, such as HTTP/1.0, HTTP/1.1, and HTTP/2, have been released to address performance, efficiency, and security concerns. These versions introduce new features and improvements, enhancing the capabilities and performance of the protocol.

Overall, HTTP is a fundamental protocol that enables communication between clients and servers on the web, allowing for the retrieval and exchange of resources and data.

c. File Transfer Protocol (FTP):

FTP (File Transfer Protocol) is a standard network protocol used for transferring files between a client and a server on a computer network. It provides a reliable and efficient method for file sharing and management. Operating on a client-server architecture, FTP allows clients to connect to a server and perform operations such as uploading, downloading, deleting, and renaming files, as well as navigating directories. It uses separate control and data connections, with the control connection handling commands and responses, and the data connection transferring the actual file data. FTP supports various authentication methods and offers features like directory listing formats, modes of data transfer, and anonymous access. However, due to security concerns related to the plain text transfer of credentials, secure alternatives like FTPS and SFTP are commonly recommended. While FTP has been widely used in the past, it is gradually being replaced by more secure and modern file transfer protocols.

Here are some key features of FTP:

i. Client-Server Architecture: FTP follows a client-server model, where the client initiates the connection and sends commands to the server, and the server responds accordingly. The client can browse, retrieve, upload, and manage files on the server.

j. Port-based Communication: FTP uses separate control and data connections. The control connection is established on Port 21, and it is used for sending commands and receiving responses. The data connection, established on Port 20 or other dynamically assigned ports, is used for transferring the actual file data.

k. Authentication and Access Control: FTP supports various authentication methods to ensure secure access to the server. Common methods include username and password authentication, anonymous access (where a user can connect without providing credentials), and more advanced authentication mechanisms such as SSH key-based authentication.

l. Commands and Responses: FTP utilizes a set of commands and responses to control the file transfer process. Clients can send commands to navigate directories, list files, download files, upload files, delete files, rename files, and perform other file operations. The server responds with status codes and messages indicating the success or failure of the requested operation.

m. Modes of Operation: FTP supports two modes of operation for transferring files: ASCII mode and Binary mode. ASCII mode is used for transferring text-based files, ensuring proper handling of newline characters and character encoding. Binary mode is used for transferring non-text files, such as images, executables, or compressed files, preserving their binary data without any interpretation.

n. Directory and File Operations: FTP provides commands to navigate and manipulate directories and files on the server. Clients can list the contents of a directory, create new directories, delete directories, rename files or directories, and change the current working directory.

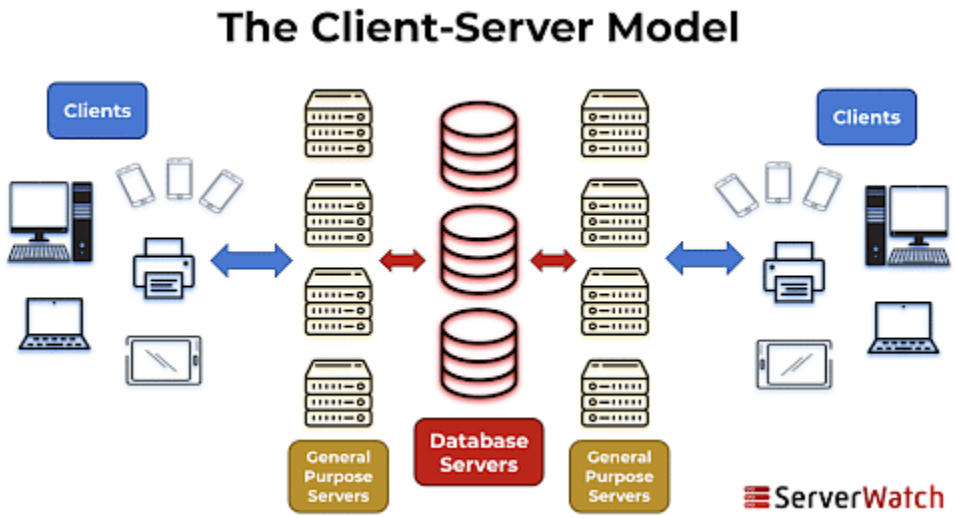
o. Passive and Active Modes: FTP supports two modes for establishing data connections: passive mode (PASV) and active mode (PORT). In passive mode, the server opens a data port and waits for the client to connect. In active mode, the client opens a data port and informs the server to connect to that port. Passive mode is more commonly used, especially in situations where clients are behind firewalls or Network Address Translation (NAT) devices.

p. Secure FTP: To enhance security, there are two popular secure variants of FTP: FTPS (FTP over SSL/TLS) and SFTP (SSH File Transfer Protocol). These secure FTP protocols provide authentication, encryption, and data integrity, protecting data during transit.

d. 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. Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc



Client-server architecture, alternatively called a client-server model, is a network application that breaks down tasks and workloads between clients and servers that reside on the same system or are linked by a computer network.

In the client-server model, clients and servers interact with each other to perform specific functions or exchange data. Here's how the model works:

a. Clients: Clients are the user-facing entities or software applications that initiate requests for services or resources from servers. Clients can be desktop computers, laptops, mobile devices, or software applications running on those devices. They rely on servers to fulfill their requests and provide the desired functionality.

b. Servers: Servers are the provider entities that offer services, resources, or data to clients. They have specialized software and hardware configurations to host and manage the requested services. Servers are typically more powerful and have more resources compared to clients to handle multiple client requests concurrently.

The client-server model operates based on the following characteristics:

- Communication: Clients and servers communicate with each other over a network using predefined protocols, such as HTTP, FTP, or SMTP. Clients send requests to servers, and servers respond with the requested data or perform the necessary actions.

- Request-Response Paradigm: Clients make requests to servers for specific services or data, and servers respond with the requested information. This request-response paradigm forms the basis of client-server communication.

- Scalability: The client-server model allows for scalability by enabling multiple clients to connect to a single server or multiple servers working together to handle client requests. This distributed nature allows for load balancing and efficient resource utilization.

- Resource Sharing: Servers host and manage shared resources or services that can be accessed by multiple clients. This includes databases, files, computational resources, or specialized functionalities offered by the server.

- Centralized Management: Servers often have centralized control and management over shared resources, user access, and security policies. They enforce rules and handle concurrent client requests to ensure the smooth functioning of the system.

- Platform Independence: With the client-server model, clients and servers can run on different platforms or operating systems. Clients can be developed for specific platforms while servers can be deployed on different hardware or software configurations. This flexibility allows clients to access server resources regardless of their underlying technology.

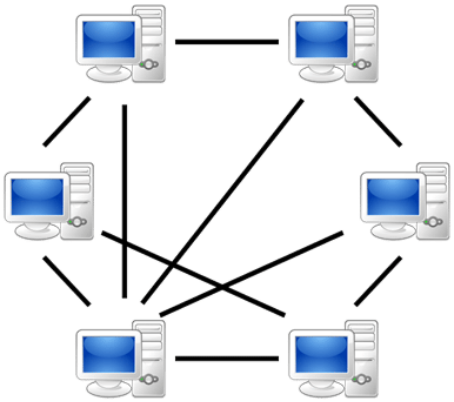
- Reliability and Fault Tolerance: The client-server model can provide fault tolerance and improved reliability. Servers can be designed with redundancy and failover mechanisms, ensuring high availability even if one server fails. Clients can seamlessly connect to an alternate server, reducing the impact of server failures on the overall system.

- Security and Access Control: The client-server model facilitates centralized security management. Servers can implement authentication mechanisms, access controls, and encryption to ensure secure communication between clients and servers. By centralizing security measures on the server-side, it becomes easier to enforce security policies, protect sensitive data, and mitigate security threats.

The client-server model is prevalent in various applications and systems, including web servers, email servers, database systems, cloud computing infrastructures, and more. It allows for efficient utilization of resources, centralized management, and the separation of concerns between clients and servers, leading to more scalable and robust systems.

e. Peer-to-Peer Model:

In a peer-to-peer (P2P) network model, computers, or "peers," participate in a decentralized network where each peer can act as both a client and a server. In other words, peers can request and provide resources or services directly with each other without the need for a central server.



Here are some key characteristics and features of the peer-to-peer model in networking:

- Decentralization: Unlike traditional client-server models, where a central server controls the resources and services, a P2P network operates in a decentralized manner. Each peer has equal privileges and can communicate and interact directly with other peers.

- Resource Sharing: One of the primary benefits of a P2P network is resource sharing. Peers can share various resources, such as files, processing power, storage capacity, or network bandwidth, with other peers. This allows for efficient utilization of resources across the network.

- Autonomous Peers: Each peer in a P2P network is autonomous and has its own resources and capabilities. Peers can independently decide which resources to share and which services to provide. This autonomy leads to a more distributed and self-organizing network.

- Dynamic Network Topology: P2P networks can have dynamic and changing network topologies. Peers can join or leave the network at any time without disrupting the overall network operation. This flexibility enables scalability and fault tolerance.

- Search and Discovery: P2P networks often include mechanisms for peer discovery and resource/service discovery. Peers can search for specific resources or services within the network by querying other peers or using distributed indexing systems.

- Load Balancing: Since each peer in a P2P network can contribute resources, the network can achieve load balancing by distributing the workload across multiple peers. This can help improve performance and handle increased demand efficiently.

- Security and Trust: P2P networks face challenges related to security and trust, as peers interact directly with one another. Implementing authentication, encryption, and trust mechanisms becomes crucial to ensure secure and reliable communication between peers.

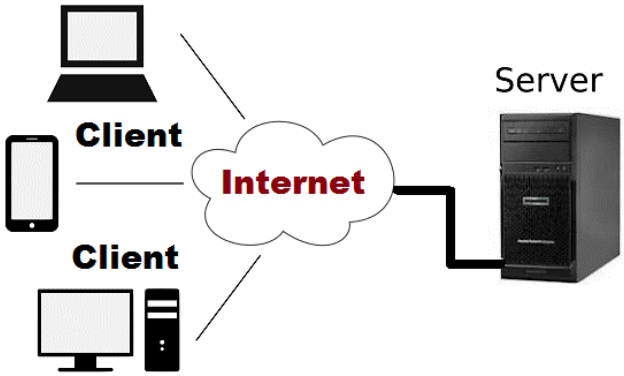
Tier Technology:

In web technology, the term "tier" refers to the different layers or components that make up a web application's architecture. Each tier represents a distinct layer of functionality and plays a specific role in the overall system.

Based on the number of layers used in the architecture, the tier technology is divided into three main categories.

2-tier (client-server) Architecture:

Two-tier web architecture, also known as client-server architecture, is an older model where the application logic and data storage are divided into two main tiers or layers: the client tier and the server tier.



Client Tier: The client tier represents the front-end or presentation layer that runs on the user's device, typically a web browser. It is responsible for rendering the user interface (UI) and handling user interactions. The client tier consists of HTML, CSS, JavaScript, and any client-side frameworks or libraries used to create the UI and enhance the user experience. The client tier interacts directly with the user, presenting information, collecting input, and sending requests to the server for data processing or retrieval.

Server Tier: The server tier, also known as the back-end or application layer, handles the core logic and data processing of the web application. It resides on a remote server or a cluster of servers. The server tier receives requests from the client tier, performs the necessary computations, and returns the requested data or results to the client. It manages the business logic, handles database operations, and performs any other server-side processing required by the application. Common server-side technologies include programming languages like Java, Python, PHP, or Node.js, along with frameworks and libraries for building robust back-end systems.

In a two-tier architecture, the client tier and server tier often communicate using a request-response model. The client sends HTTP requests to the server, which processes the requests and sends back the response. This can involve fetching data from a database, executing business logic, or generating dynamic content to be displayed by the client. The server tier typically handles the validation and processing of data submitted by the client, as well as any necessary data storage and retrieval.

Two-tier architecture is suitable for smaller, simpler applications where the client-side logic is relatively lightweight and most of the heavy lifting happens on the server. However, it can suffer from scalability limitations and lack of flexibility compared to multi-tier architectures. Modern web applications often adopt a three-tier or n-tier architecture, where additional layers are introduced to improve scalability, separation of concerns, and overall application design.

Three Tier (Client/Business/Data) Architecture:

The three-tier client-server web architecture is a widely adopted model that separates the components of a web application into three distinct tiers. Each tier performs specific functions and communicates with the others to create a scalable, modular, and efficient system. In this article, we'll delve into the three tiers of this architecture and their roles in web application development.

Client/Presentation Tier: The presentation tier, also known as the client-side or front-end, focuses on user interface (UI) rendering and user interactions. It runs on the user's device, typically a web browser, and is responsible for presenting data and collecting user input. Technologies like HTML, CSS, JavaScript, and various frameworks (e.g., React, Angular, Vue.js) are commonly used to build responsive and interactive UIs. The presentation tier communicates with the other tiers, requesting and displaying data while sending user actions or data inputs to the application tier.

Application/Middle Tier: The application tier, sometimes referred to as the middle-tier or business logic layer, contains the core application logic and processes data. It receives requests from the presentation tier, performs data processing, applies business rules, and prepares data for storage or retrieval. This tier typically utilizes server-side technologies such as Java, Python, PHP, or Node.js, along with frameworks or libraries (e.g., Express, Django, Spring). It interacts with databases or external services to handle data operations, perform calculations, and execute the application-specific business logic.

Date Tier (Server side): The data tier, also called the server-side or back-end, manages the storage and retrieval of data required by the application. It primarily deals with databases, file systems, or external data sources. The data tier receives requests from the application tier, retrieves or modifies data, and sends it back as a response. Technologies like relational databases (e.g., MySQL, PostgreSQL), NoSQL databases (e.g., MongoDB, Redis), or cloud-based storage systems are commonly employed in this tier. The data tier ensures data integrity, implements data access logic, and manages interactions with persistent data sources.

The three-tier client-server web architecture offers a structured and scalable approach to building web applications. The presentation tier handles UI rendering and user interactions, the application tier manages the business logic and data processing, and the data tier ensures efficient data storage and retrieval. By dividing responsibilities and facilitating communication between tiers, this architecture promotes flexibility, maintainability, and scalability in web application development.

Benefits of Three-Tier Architecture:

The three-tier client-server web architecture offers several advantages:

- Scalability: Each tier can be scaled independently, allowing efficient resource allocation based on demand. This scalability enhances performance and supports the growth of the application.

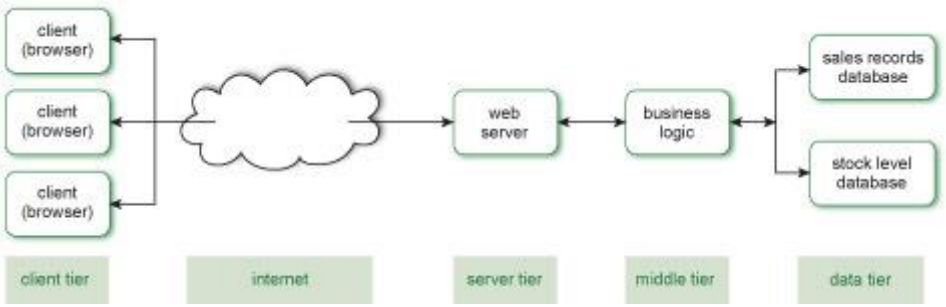
- Modularity: The separation of concerns into distinct tiers promotes modularity and maintainability. Each tier can be developed, tested, and deployed separately, making code management and updates more manageable.

- Reusability: By decoupling the presentation, application, and data layers, components within each tier can be reused in different contexts or across multiple applications.

- Security: The three-tier architecture improves security by enforcing clear boundaries between the presentation, application, and data tiers. Sensitive data can be stored securely in the data tier, and access can be controlled via the application tier.

N-Tier Architecture:

The N-tier or multi-tier web architecture is an extension of the three-tier architecture that allows for further scalability, flexibility, and modularity in web application development. This architecture divides the components of a web application into multiple tiers or layers, each with distinct functionalities and responsibilities. In this article, we'll explore the N-tier architecture and its benefits in building robust and scalable web applications.



The N-tier architecture expands on the three-tier architecture by introducing additional tiers or layers. While the exact number of tiers can vary depending on the complexity of the application, the N-tier architecture commonly includes the following layers:

Presentation Tier (Client-side):

Similar to the three-tier architecture, the presentation tier focuses on the user interface (UI) and user interactions. It handles the rendering of the UI on the client-side, using technologies like HTML, CSS, JavaScript, and client-side frameworks. The presentation tier remains responsible for delivering a seamless user experience.

Application Tier (Business Logic):

Same as in three tier architecture. The application tier, often referred to as the business logic or application logic layer, contains the core logic and processing of the web application. It manages business rules, performs calculations, and coordinates interactions between the different layers. This tier handles application-specific functionality, data validation, and workflow orchestration.

Data Tier (Data Management):

Same as in three tier architecture. The data tier is responsible for managing data storage and retrieval. It encompasses the storage systems, such as relational databases, NoSQL databases, or external APIs. This tier ensures efficient data access, implements data integrity constraints, and handles data-related operations like querying, indexing, and caching.

Additional Tiers (Optional):

In more complex applications, additional tiers may be introduced to enhance scalability, performance, or security. These tiers might include a caching tier to store frequently accessed data, a messaging tier for asynchronous communication, or a service-oriented architecture (SOA) layer to enable modularity and interoperability.

Here are a few examples of additional tiers that can be included in an N-tier web architecture:

a. **Caching Tier:**

   The caching tier is responsible for storing frequently accessed data to improve performance and reduce the load on the data tier. It can utilize technologies like Redis, Memcached, or in-memory caching systems. By caching data closer to the application tier, it reduces the need for repeated expensive database queries and enhances overall responsiveness.

b. **Messaging Tier:**

The messaging tier allows for asynchronous communication between different components or services within the application. It uses message queues, event-driven architectures, or message brokers like RabbitMQ or Apache Kafka. This tier enables decoupling of processes, improves scalability, and supports reliable communication between different parts of the system.

c. Service-oriented Architecture (SOA) Layer:

The SOA layer focuses on service composition, modularity, and interoperability. It promotes the development of independent, reusable services that can be combined to build complex applications. Each service encapsulates specific functionality and communicates through standardized protocols such as RESTful APIs or SOAP. The SOA layer enhances flexibility, enables system integration, and allows for easy updates and modifications.

d. Security Tier:

The security tier handles various security-related concerns in the application. It includes components like authentication servers, access control mechanisms, encryption/decryption modules, and security auditing tools. This tier ensures secure data transmission, protects against unauthorized access, and implements security best practices to safeguard the application and its data.

e. Analytics and Reporting Tier:

The analytics and reporting tier collects, processes, and analyzes data generated by the application. It involves tools and technologies for data analytics, business intelligence, and reporting. This tier enables the extraction of valuable insights from the application's data, facilitates decision-making, and supports performance monitoring and optimization.

f. Scalability and Load Balancing Tier:

The scalability and load balancing tier focuses on distributing application load across multiple servers or instances to handle increased traffic or demand. It employs technologies like load balancers, auto-scaling groups, or container orchestration platforms (e.g., Kubernetes) to dynamically scale resources and ensure high availability and performance.

These are just a few examples of additional tiers that can be included in an N-tier web architecture. The specific additional tiers and their functionalities depend on the complexity, scalability needs, and requirements of the web application. The modular nature of N-tier architecture allows for flexibility in incorporating these tiers based on the specific needs of the application.

The N-tier or multi-tier web architecture expands upon the three-tier architecture, providing scalability, modularity, and flexibility for web applications. The separation of responsibilities into distinct tiers allows for independent development, easy maintenance, and horizontal scalability of specific layers. By utilizing

this architecture, developers can create robust, scalable, and maintainable web applications that can adapt to changing requirements and handle increasing traffic with efficiency.