DAY 21 **DATE:21/05/2025**

NAME: ANNIE JOHN

USER ID:27739

Batch: 25VID0885_DC_Batch4

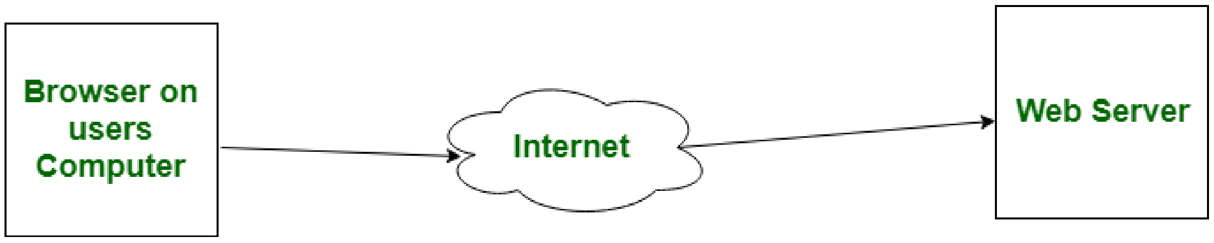
TITLE: WEB SERVER AND APPLICATION SERVER,2-TIER AND 3-TIER ARCHITECTURE

➢ **What is a Web Server?**

A **web server** is a computer program or device that serves web content, primarily handling HTTP requests from clients (usually web browsers). It primarily serves **static web content**, such as HTML, CSS, JavaScript, and image files, by delivering them directly to the client. Web servers can also handle **dynamic web content** by interacting with server-side scripts, which generate content on the fly based on user requests.

Example of Web Servers:

- Apache HTTP Server
- Nginx
- Microsoft IIS
- Resin



Web Server

- **Key features**
  1. Serves Static Content: Delivers HTML, CSS, images, and JavaScript.
  2. Handles HTTP Requests: Manages requests from browsers and sends the requested files.

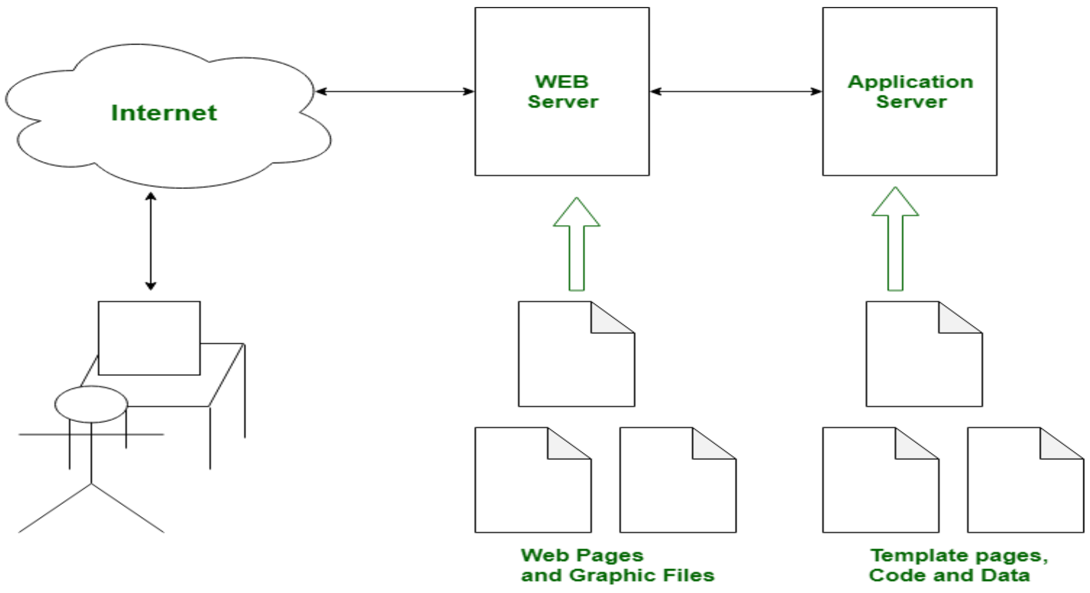
3. Secure Connections: Supports HTTPS for secure data transfer.
4. Proxy and Load Balancing: Can forward requests and distribute traffic across servers.
5. Caching: Stores static files temporarily to speed up delivery.
6. Error Handling: Shows error messages like 404 if content isn't found.
7. Access Control: Restricts access to certain content using basic authentication.
8. Logging: Records requests and errors for monitoring.

➢ What is an Application Server?

An application server is a software platform that hosts and runs **dynamic applications,** handling business logic, database connections, and communication between clients and data sources. It processes requests from users, executes the underlying business rules, and returns dynamic content or data.

Examples of Application Server:

- WebLogic
- JBoss
- WebSphere
- Glassfish



- **Key features**

    1. Handles Dynamic Content: Generates content based on user requests (e.g., fetching data from databases).
    2. Executes Business Logic: Runs the core application processes and rules.
    3. Database Connectivity: Connects to databases and handles data operations.
    4. Session Management: Manages user sessions across multiple requests.
    5. Middleware Support: Handles tasks like authentication and authorization.
    6. API Support: Exposes APIs for communication with other systems.
    7. Scalability: Can scale to handle more users by adding more servers.
    8. Security: Ensures secure access to business logic and data.
    9. Integration with Web Servers: Works with web servers to serve both static and dynamic content.

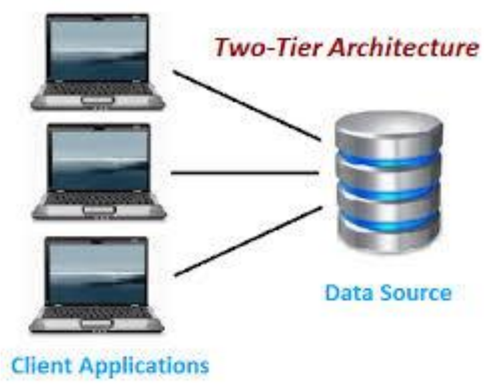
➢ **Integration Endpoints in an Application Server**

Integration endpoints in an application server are the access points (usually APIs or services) that external systems, applications, or components use to interact with the business logic, data, or functionality of the server.
Here are some **real-world examples** of integration endpoints in an application server:

- A customer places an order on an e-commerce website. The application server receives the order details, processes business logic (e.g., inventory check), saves it to the database, and calls a payment service to process the transaction.
- A user checks their account balance through the mobile app. The app calls this endpoint, and the application server fetches real-time data from the banking core.

➢ **2-Tier Architecture**

A **2-tier architecture** is a client-server model where the presentation layer (client) directly communicates with the data layer (server/database), usually through a network.



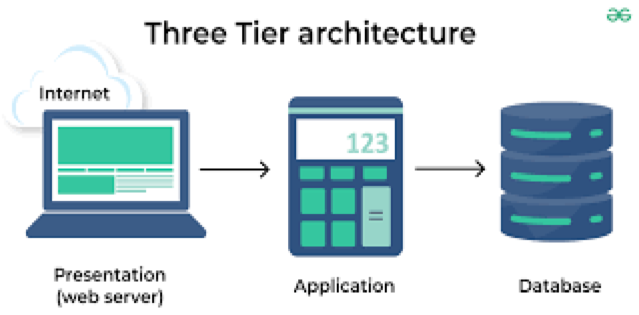
- **Layers in 2-Tier Architecture**
    1. **Client Layer (Presentation Layer)**
       The user interface where users interact with the system.
       Example: Desktop application, web frontend.
    2. **Server Layer (Data Layer)**
       A centralized server containing the database and sometimes the business logic. Processes requests from the client and responds with data.
- **Working of 2-Tier Architecture**
    - i. A user opens an application (e.g., a desktop inventory management system).
    - ii. The application (client) sends a request directly to the database server.
    - iii. The server processes the request (e.g., fetches product info) and sends the response back to the client.
    - iv. The client displays the data to the user.
- **Example of 2-Tier Architecture**
    1. **Desktop Payroll System**:
       A payroll application installed on employee computers (client layer).The application connects directly to a central database server where employee records and salary data are stored (server layer).
- **Advantages of 2-Tier Architecture**
    - **Simple Design**: Easy to set up and understand.
    - **Fast Communication**: Direct connection between client and server makes response times quicker in small systems.
    - **Cost-Effective**: Fewer components mean lower initial infrastructure and maintenance costs.
    - **Good for Small Systems**: Ideal for small applications with limited user access.

- **Disadvantages of 2-Tier Architecture**
  - **Scalability Issues**: Hard to scale with many clients; performance degrades.
  - **Tight Coupling**: Any changes in the database or logic may require updates to all clients.
  - **Security Risks**: Clients connect directly to the database, increasing exposure to potential attacks.
  - **Limited Business Logic Handling**: Pushing business rules to the client can cause redundancy and inconsistency.
- **Real-Life Use Cases**

Use Case	Description
Small Office Inventory System	A desktop app used by a few staff members to track inventory, connected to a local database server.
Library Management Software	A local library's software installed on librarian computers directly accessing the central database.
Bank Teller Systems (older ones)	Teller terminals connect directly to the bank's main database without a middle application layer.
College Examination System	A simple client app on staff computers directly accessing the exam and results database.

➢ **3-Tier Architecture**

A **3-tier architecture** separates an application into three distinct layers: **Presentation**, **Application (Business Logic)**, and **Data** layers. Each layer is independent and communicates with the next layer through interfaces, typically over a network.



- **Layers in 3-Tier Architecture**
  1. **Presentation Layer (Client Layer)**

     The user interface (UI) where users interact with the application.

     Example: Web browser, mobile app, or desktop app.

2. **Application Layer (Business Logic Layer)**
   Processes business rules, logic, validations, and operations.
   Example: Backend server or middleware (Java Spring Boot, .NET Core, Node.js).
3. **Data Layer (Database Layer)**
   Manages data storage and retrieval.
   Example: SQL Server, MySQL, PostgreSQL, Oracle.

- **Working of 3-Tier Architecture**

i. The **client** sends a request (e.g., view user profile).
ii. The **application server** processes the request by applying business rules and logic.
iii. It communicates with the **database** to retrieve or update data.
iv. The **application server** sends the processed response back to the **client**.

- **Example of 3-Tier Architecture**
  1. **Online Banking System**:
     Client Layer: User logs in through a web/mobile app.
     Application Layer: Server validates login and processes transactions.
     Data Layer: Stores user details, account balances, and transaction history in a secure database.
- **Advantages of 3-Tier Architecture**
  - **Separation of Concerns**: Each layer is independent and easier to manage.
  - **Scalability**: Each layer can be scaled separately (e.g., add more app servers).
  - **Better Security**: Database is not directly exposed to the client.
  - **Easier Maintenance**: Changes in one layer don’t affect others directly.
  - **Reusable Logic**: Business logic in the application layer can be reused across multiple clients (web, mobile, etc.).
- **Disadvantages of 3-Tier Architecture**
  - **Increased Complexity**: More layers mean more components and configuration.
  - **Higher Cost**: Requires more resources (servers, network, management).
  - **Latency**: More network hops can introduce slight delays.
  - **Initial Setup Time**: More effort is needed to design and deploy the system.

- **Real-Life Use Cases**

Use Case	**Description**
E-commerce Website	Web frontend (Presentation) → Backend server (Application) → Product/order DB (Data).
Banking Applications	Mobile banking app → Core banking logic → Secure account database.
Online Learning Platforms (LMS)	Student portal → Business logic (courses, quizzes) → Learning content DB.
Healthcare Systems	Doctor’s dashboard → Application logic (scheduling, records) → Patient DB.
Enterprise CRM Systems	Web/mobile UI → Middleware (business rules, workflows) → Customer data DB.