**Web Technologies**

**Lecture 1**

**Web platforms**

Web platforms are comprehensive software environments that enable the creation, hosting, and management of websites and web applications on the internet.

They provide a set of tools, services, and infrastructure for developers and users to interact with digital content and services online.

Web platforms make it simple to build, publish, and use web content by supporting different web technologies.

**Components of Web Platforms**

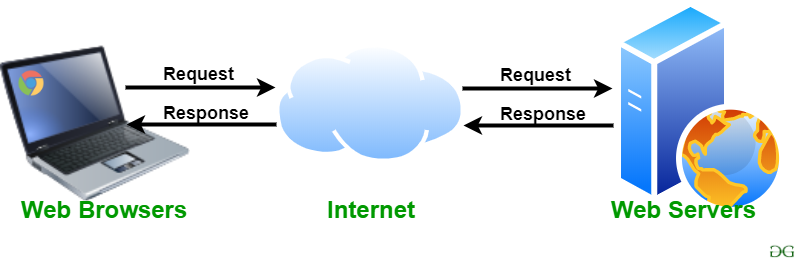
**1. Web Servers**

These are specialized software or hardware systems that store and deliver web content in response to client requests.

It uses rules like HTTP to answer user requests on the internet.

The main job of a web server is to store, process, and show website pages to users.

**Examples** include Apache, Nginx, and Microsoft Internet Information Services (IIS).



**2. Web Browsers**

When a [user](https://en.wikipedia.org/wiki/User_(computing)) requests a [web page](https://en.wikipedia.org/wiki/Web_page) from a particular website, the browser retrieves its [files](https://en.wikipedia.org/wiki/Computer_file) from a [web server](https://en.wikipedia.org/wiki/Web_server) and then displays the page on the user's screen.

Browsers are used on a range of devices, including [desktops](https://en.wikipedia.org/wiki/Desktop_computer), [laptops](https://en.wikipedia.org/wiki/Laptop), [tablets](https://en.wikipedia.org/wiki/Tablet_computer), and [smartphones](https://en.wikipedia.org/wiki/Smartphone)

Web platforms interact with web browsers (e.g., Chrome, Firefox, Safari) that users utilize to access and view web content.

****

**3. Databases**

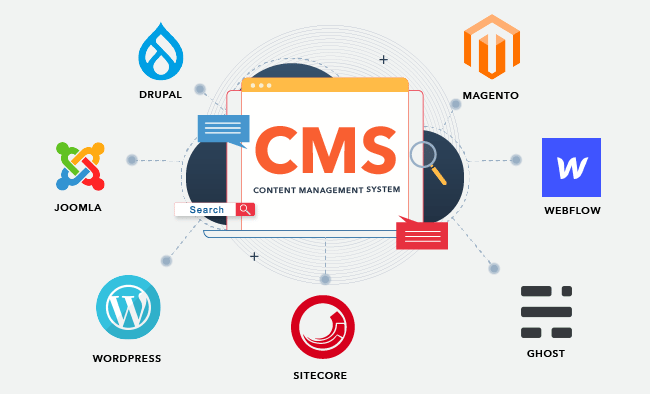
Many web platforms integrate databases to store and manage data used by web applications. **Examples** include MySQL, PostgreSQL, and MongoDB.

**4. Programming Languages and Frameworks**

Web platforms support various programming languages (e.g., PHP, Python, Ruby) and frameworks (e.g., Django, Ruby on Rails, Laravel) to build dynamic web applications.

**5. Content Management Systems (CMS)**

These platforms allow users to create and manage digital content easily. Examples include WordPress, Drupal, and Joomla.

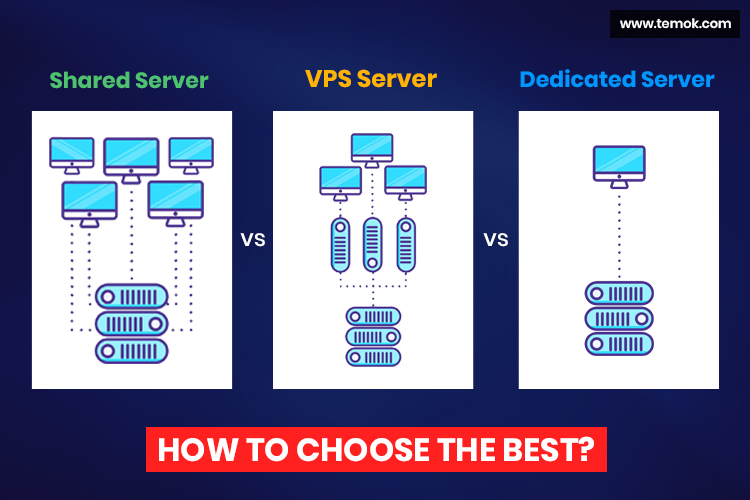


**6. Web Hosting Services**

These are services that let people put their websites on the internet. It gives space, server power, and internet connection so that websites can be seen online. **Examples:** shared hosting, cloud hosting, dedicated hosting.

**Types of Web Hosting:**

1. **Shared Hosting:** A low-cost option where many websites are stored on one server. They all share the server’s resources (CPU, RAM, bandwidth, etc.).
2. **VPS Hosting (Virtual Private Server):** A single physical server is divided into separate virtual servers. Each website gets its own dedicated resources and works independently.
3. **Dedicated Hosting:** One full physical server is used by a single user or company. All resources belong only to them, giving maximum control, speed, and security.
4. **Cloud Hosting:** Uses many connected servers instead of just one, so websites can easily grow and stay online even if one server fails.





**Examples of Web Platforms**

**WordPress**

* **Overview:** WordPress is a widely used content management system (CMS) that makes it easy to create and manage websites. It is free, open-source, and can be customized with themes and plugins.
* **Example Use Case:** A blogger builds a personal website on WordPress to write and share articles.

**Shopify**

* **Overview:** Shopify is an e-commerce platform made for online stores. It provides tools to add products, manage payments, and track inventory.
* **Example Use Case:** A small business owner uses Shopify to create an online shop and sell their products.

**Amazon Web Services (AWS)**

* **Overview:** AWS is a cloud platform by Amazon that offers services like web hosting, databases, storage, and AI tools.
* **Example Use Case:** A startup uses AWS to run its web app with scalable and reliable infrastructure

**Google Sites**

* **Overview:** Google Sites is an easy tool for making simple and shared websites. It is part of Google Workspace.
* **Example Use Case:** A team creates a wiki on Google Sites to store project notes and share information.

**Wix**

* **Overview:** Wix is a beginner-friendly website builder that helps people make attractive websites without coding.
* **Example Use Case:** An artist uses Wix to build an online portfolio to show their artwork.

**Heroku**

* **Overview:** Heroku is a cloud platform (PaaS) that makes it easy to host and scale web apps.
* **Example Use Case:** A developer uploads a Node.js app on Heroku to host it and handle growth easily.

**URL (Uniform Resource Locator)**

A URL is a web address used to identify and locate resources on the internet.

It consists of several components, including the protocol (e.g., http:// or https://), domain name (e.g., [www.example.com](http://www.example.com/)), and path (e.g., /page1).

**Example:** <https://www.example.com/page1>

**Protocols**

**1. Hyper-Text Transfer Protocol.**

It is the set of rules that tells computers how to send and receive web pages over the internet. It is the foundation of data communication on the World Wide Web, enabling the exchange of text, images, videos, links, and other resources between a user's device and web servers.

HTTP operates on a request-response model, where a client sends a request for a resource, and the server responds with the requested data (usually in the form of an HTML web page). It is a stateless protocol, meaning each request from the client to the server is independent, and the server doesn't retain any information about previous requests.

When you open a website, your browser uses HTTP to request the page from a web server, and then the server sends it back to you.

**Example:**When you type **http://example.com** in the browser, HTTP is the protocol that helps your browser and the server talk to each other.

**2. HTTPS (HTTP Secure):**

HTTPS is a secure version of HTTP. It encrypts data exchanged between a web server and a client to protect against eavesdropping and data tampering.

**Example:** https://example.com

**Domain Name**

* A domain name is the easy-to-read address of a website that connects to its IP address on the internet.
* It helps people find websites without remembering long numbers (IP addresses).
* **Example:** www.example.com is a domain name.
* In simple words, it is a text label that points to the numeric IP address so browsers can open the website.

**HTML (Hypertext Markup Language)**

* A **markup language** is a computer language that organizes and formats data in a document or webpage.
* In simple words, markup tells the browser **how to display text, images, and other content** on the page.
* HTML is the main language used to build the **structure and content** of web pages.
* It uses **tags** (like <h1>, <p>, <a>, <img>) to define things such as headings, paragraphs, links, and images.
* HTML is usually combined with **CSS** (for design and styling) and **JavaScript** (for interactivity).

****

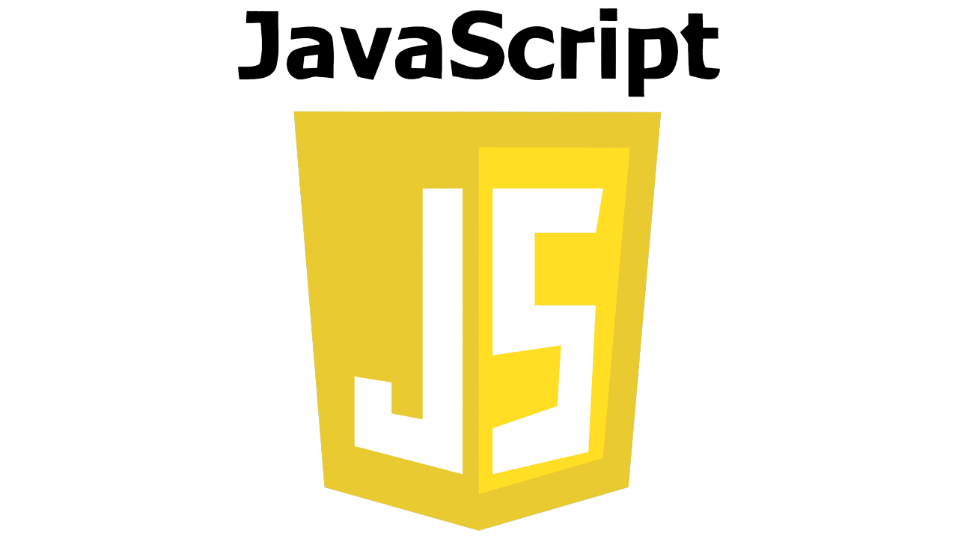
**Cascading Style Sheets (CSS)**

* CSS is a style language used to decide how a webpage looks.
* It works with markup languages like HTML or XML.
* CSS controls the **design** of a page, such as layout, colors, fonts, and spacing.
* In simple words, HTML builds the structure, and CSS makes it **look nice and attractive**.

****

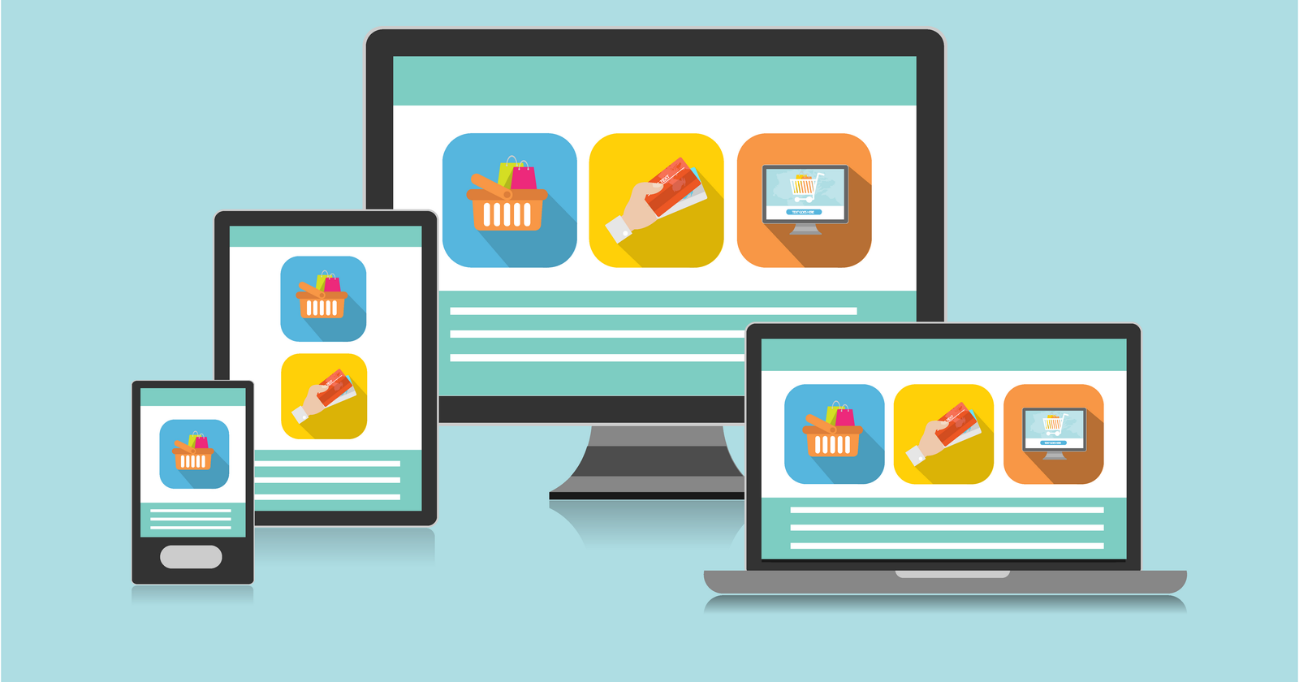
**JavaScript**

JavaScript is a scripting language that makes websites do things.  
It lets developers make web pages respond to clicks, typing, and other actions, and change what is shown on the page.



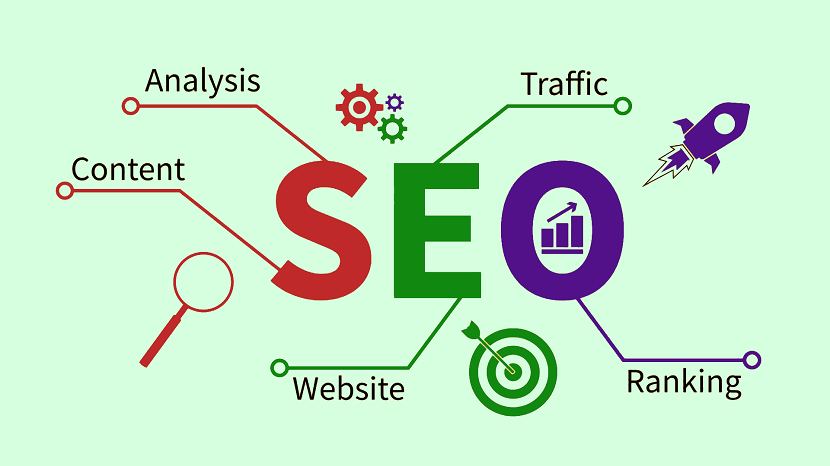
**Responsive Web Design**

Responsive web design is making a website that fits and looks good on any screen, whether it’s a computer, tablet, or phone.

****

**SEO (Search Engine Optimization)**

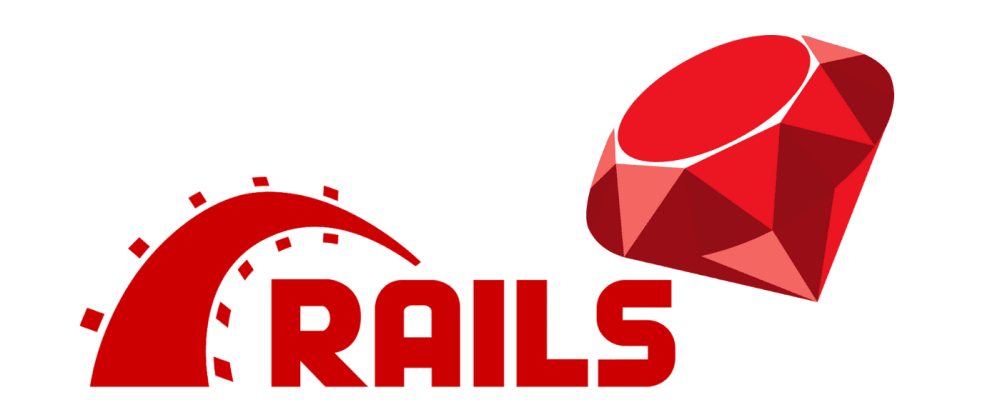
SEO means making a website better, so it shows up higher in search results on Google or other search engines. This helps more people find and visit the website.

****

**Web development framework**

A web development framework is a ready-made set of tools and code that helps you build websites and web apps faster and easier. **Examples** are Ruby on Rails, Django, and Angular.





**API (Application Programming Interface)**

An API (Application Programming Interface) is a **set of rules and tools that let different software programs communicate with each other**. It defines **how one program can request data or services from another** and how the other program will respond.

**For example,** a weather app can use a weather API to get the latest forecast from a weather service without building all the data systems itself.

**Common Use Cases**

* **Social Media Integration:** Add features from Facebook, Twitter, or Instagram to your app.
* **Payment Processing:** Use PayPal or Stripe to handle online payments.
* **Data Retrieval:** Get information like weather updates, stock prices, or news for your app.

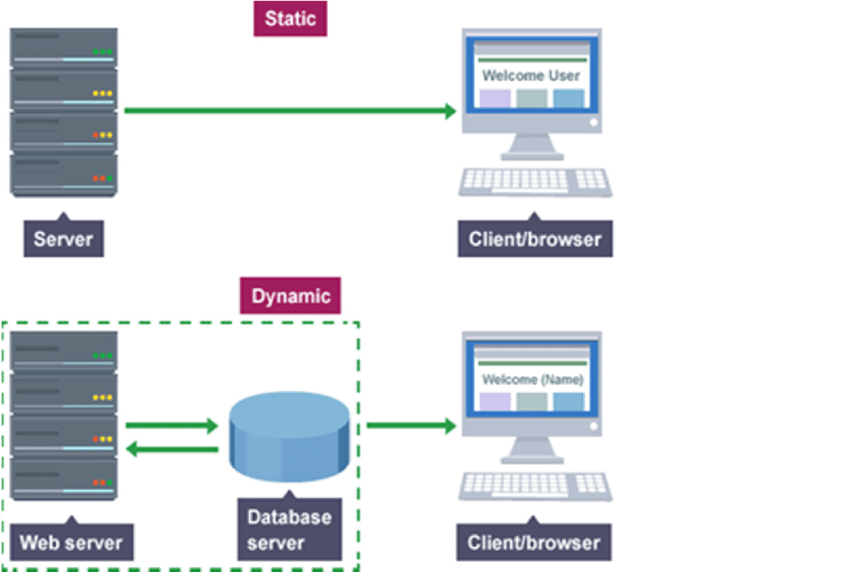
**Web Application Categories**

1. **Static Web Applications:**

* Simple websites that **don’t change much** and **aren’t interactive** unless manually updated.
* Made using **HTML, CSS, and JavaScript**.

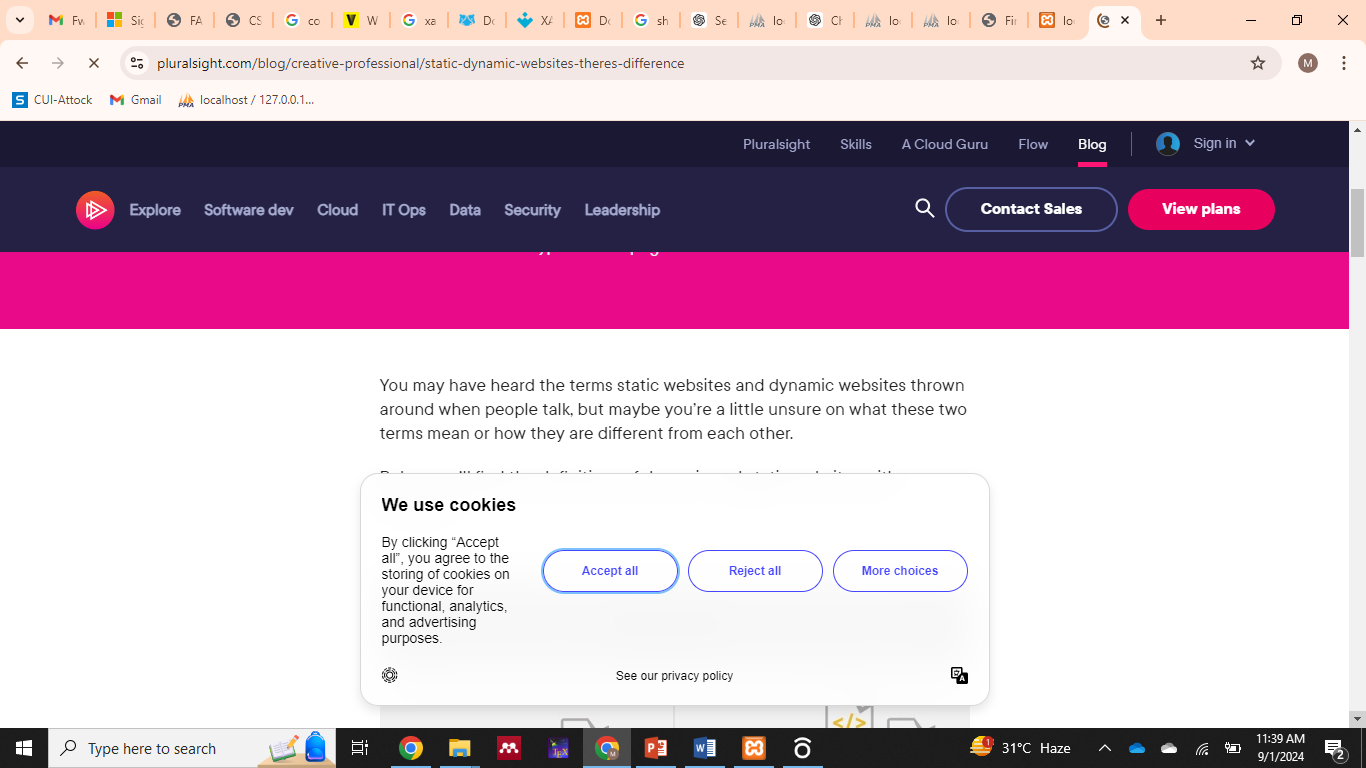
1. **Dynamic Web Applications**

* Websites that **change and respond** to user actions.
* Can **create, read, update, and delete (CRUD) data**.
* Made with **HTML, CSS, JavaScript, PHP, or CMS** like WordPress.



**Cookies**  
Cookies are **small pieces of data that a website stores on a user’s device**. They help the website remember information about the user between visits. **For example,** cookies can keep a user logged in, save site preferences like language or theme, and track user behavior for analytics or ads. Cookies are stored on the **user’s browser** and can last for a short time or longer depending on their purpose.

**Sessions**  
Sessions are a way to **store user-specific information on the server while the user is browsing a website**. Unlike cookies, the data is kept on the server, and the user gets a unique session ID to identify their session. Sessions are used to keep users logged in across pages, remember shopping cart items, and store temporary preferences during a visit. Sessions usually **expire when the user closes the browser or after a set time**.



**Web 1.0**

* **Timeframe:** Late 1980s to early 2000s (Read-only web)
* **Characteristics:** The first version of the internet where websites were **mostly static**. Users could **only read information**, not interact or contribute. Businesses used it to **share information**, and people could **search and view content** but not change it.

**Web 2.0**

**Timeframe:** Early 2000s to mid-2000s

**Characteristics:**

* **Interactive and User-Centric:** Websites became **dynamic**, letting users **add content, comment, and interact**.
* **Social Media Emergence:** Platforms like **Facebook, YouTube, and Twitter** allow users to **share content and connect**.
* **Rich Internet Applications (RIAs):** Technologies like **AJAX** made websites faster and smoother by updating data without reloading pages.
* **Collaboration and Sharing:** Focused on sharing information and working together online.
* **Rise of Blogs and Wikis:** Made it easy for individuals and communities to create and edit content.

**Web 3.0 (Semantic Web)**

**Timeframe:** Mid-2000s onwards (still developing)

**Characteristics:**

* **Smart Web:** Websites and data can be **understood by computers** to give smarter results.
* **Connected Data:** Information on the web is **linked**, so computers can find useful connections.
* **Personalized Experience:** Uses **AI** to show content and suggestions **just for you**.
* **Smart Devices:** Works with **Internet of Things (IoT)** like smart home devices, connecting the real world with the internet.

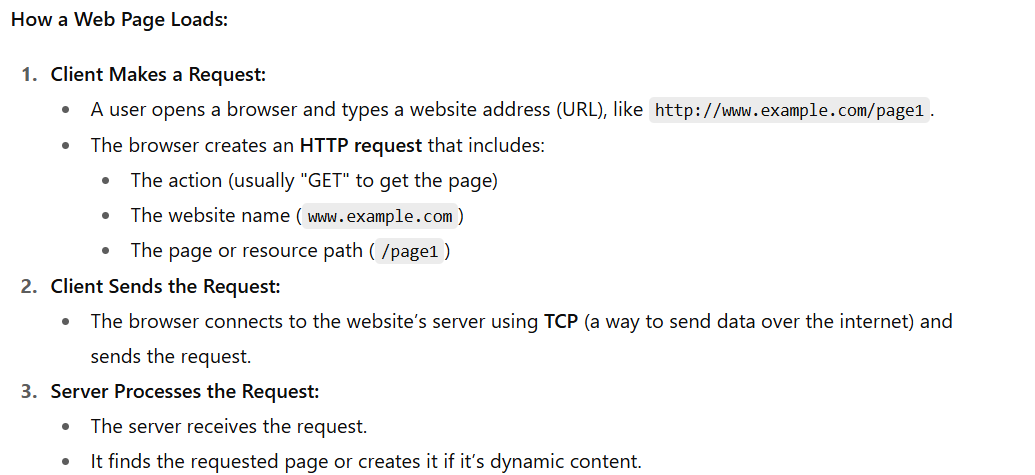
**Web 4.0 (Decentralized Web)**

**Timeframe:** Emerging concept (not fully developed yet)

**Characteristics:**

* **Decentralization:** Web will rely less on central servers and allow **peer-to-peer interactions**.
* **Blockchain and Web 3.0 Integration:** **Blockchain** will help secure data, identities, and transactions.
* **Better Privacy and Security:** Users will have **more control over their data** and better protection from hacks.
* **Extended Reality (XR):** Uses **AR and VR** to create immersive web experiences.

**HTTP Example**



A screenshot of a computer screen

AI-generated content may be incorrect.

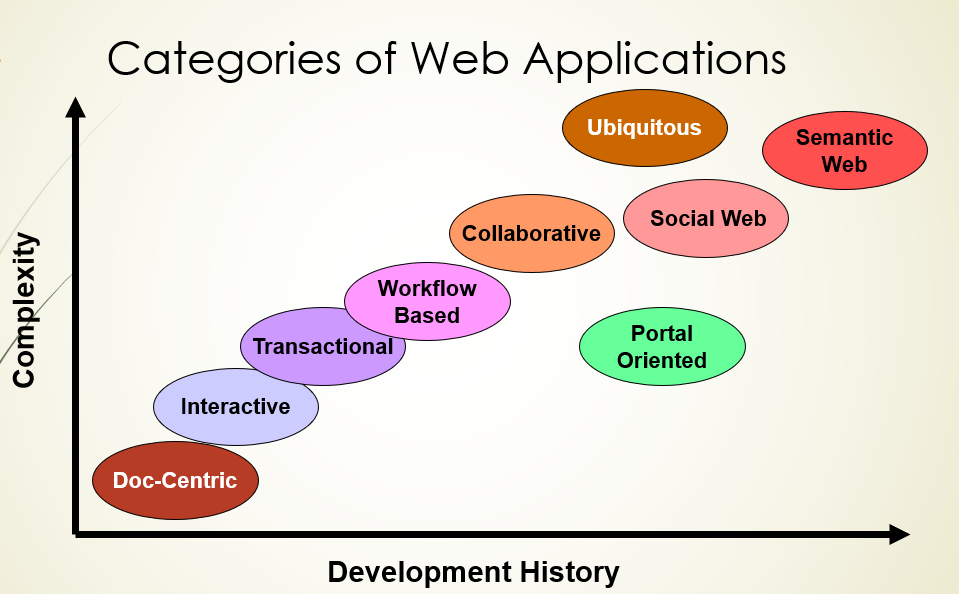
**Transmission Control Protocol (TCP)**

**Transmission Control Protocol (TCP)** is a set of rules that lets computers **send messages to each other over a network**.

TCP makes sure there is a **reliable connection** between two devices so that data is sent **safely and in order**.

When you use the web, **HTTP uses this TCP connection** to transfer web pages and other data between the server and your browser.

**Lecture 2**

****

**Document Centric**

**Early Web Pages (Before Web Applications)**

* They were just simple static HTML documents.
* To make changes, people had to do manual updates.

**Pros (Good Things)**

* Very simple and easy.
* Stable (didn’t crash much).
* Fast response time (loaded quickly).

**Cons (Bad Things)**

* High cost of management if there are many updates or a big website.
* **Inconsistent/Redundant info** → Same content on different pages may not match after updates.

**Example**

* Static home pages (unchanging front pages).

**Collaborative Social Web**

* People **work together on the internet**.
* They can **make, edit, and share information** with others.
* Talking and **communicating with each other is very important**.
* **Examples:** Wikis, GitHub, Stack Overflow

**The Social Web**

* People **join online groups or communities** with the same interests.
* They **share ideas, write blogs, and save useful links**.
* **Examples:** Blogs, social bookmarking (like Delicious)

**Integration with Web Applications**

* Social features are **added to other websites and apps**.
* **Examples:** Blogs, Facebook, Twitter, Netflix

**Semantic Web**

* **Idea by Tim Berners-Lee:** The web should be readable not only by **people** but also by **machines**.
* Uses **metadata** (extra data about data) and **ontologies** (rules/structures for knowledge) to help computers **understand and manage information** better.
* **Content syndication** tools like **RSS and Atom** let information be **shared and reused easily** across websites.

**Characteristics of Web Applications**

How do Web applications differ from traditional applications?

3 dimensions of the ISO/IEC 9126-1 standard

1. product-Usability
2. Usage-functionality
3. Development-Reliability

**Product Reusability**

* This dimension checks if the software can **do the job it was made for** and meet what users need.
* It looks at whether the software **works correctly and does its tasks well**.
* This dimension is about the **quality of the product itself**.

**Usage – Functionality**

* This dimension is about how easy and effective it is for people to use the software.
* It includes things like how simple it is to learn, how clear the design is, and the overall user experience.
* It shows how the product is used in real life, which matches the idea of “usage.”

**Development – Reliability**

* This dimension checks how **trustworthy and strong** the software is, both whiles being built and after it is released.
* It looks at things like **handling errors** and **recovering from problems**, which are important during making the software and later when maintaining it.
* It shows the quality of the **development process** and how well the software can **work without failing**.

**Web Architectures**

The following layers are used in web application development:

1. Presentation layer/user interface layer.
2. Application layer/business logic layer.
3. Data layer.

**Presentation Layer**

This is the part of the software that the user can see and use.  
Through this layer, the user gives inputs and instructions and also sees the output.

**Example**

In an online shopping website, the pages where users browse products, see details, add to cart, and checkout are the presentation layer. Things like buttons, forms, images, and menus belong here.

**Business Logic Layer (Application Layer)**

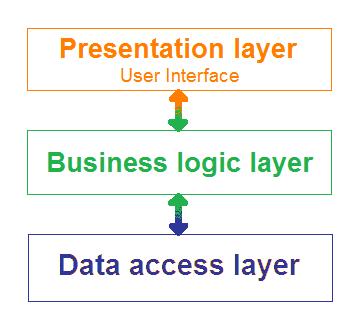
This layer has all the rules, models, and logic that make the application work.  
It acts as a middle layer between the user interface (what the user sees) and the database (where data is stored).  
In simple words, it runs the operations of the application.

**Data Layer (Database Layer)**

The Data Layer is where all the **information of the application is stored and managed**.  
It is also called the **database layer**, **data access layer**, or **back-end**.  
This layer usually uses databases like **MySQL, PostgreSQL, Oracle, or MariaDB**.

**Example**

In an online shopping site, the **list of products, user accounts, orders, and payments** are all stored in the data layer.



**1-Tier Architecture (Single-Tier / Stand-Alone)**

* In this type of architecture, **everything is in one place** — the user interface, business logic, and data are all part of the **same package**.
* The **client and server are on the same computer**.
* It is the **simplest type of architecture**.
* **Limitation:** It is **not good for web applications**, because it can only use data from **one computer** and cannot connect to many users online.

**Example of 1-Tier Architecture**

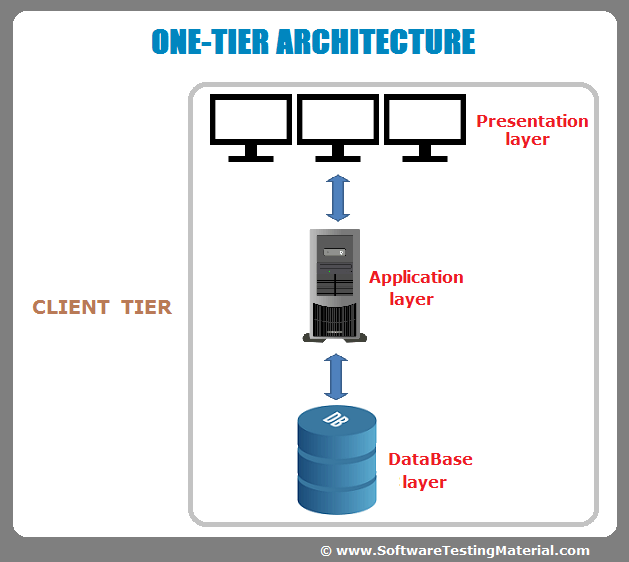
* **MS Office** (like Word, Excel, PowerPoint) is a common example of 1-tier architecture.
* It is **cheap to build,** and such applications are usually **easy to develop**.

**Advantages**

* **Cost-efficient** → Cheaper to make and use.
* **Easy to maintain** → Simple to update since everything is on one machine.

**Disadvantages**

* **Cannot share information** → Data stays only on one computer.
* **Not flexible** → If something changes in the system, the app may stop working.



**2-Tier Architecture (Client–Server)**

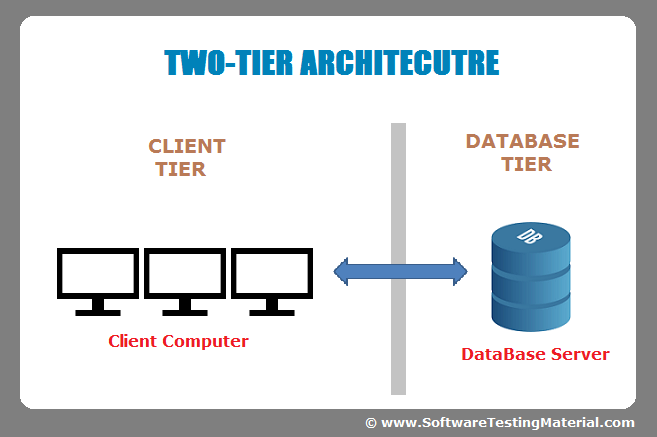
* The **client system** handles the **Presentation Layer** (user interface) and the **Application Layer** (logic/operations).
* The **server system** handles the **Database Layer** (data storage).
* It is also called a **Client–Server Application**.

**How it Works**

1. The **client** sends a request to the server (for example: “show my account balance”).
2. The **server** processes the request and sends the **data back** to the client.

**Example**

**Online Banking System** → The client (your computer or app) requests balance details, and the bank’s server sends the data back.



**3-Tier Architecture**

In this type, the system is divided into **three layers**:

1. **Presentation Layer** → What the user sees (UI).
2. **Business Logic Layer** → The rules and operations of the application.
3. **Data Layer** → Where data is stored (database).

Each layer has its **own job,** and they work together step by step.

**How it Works**

1. The **user** interacts with the **Presentation Layer** (clicks, inputs).
2. The **Business Logic Layer** checks rules, runs operations, and decides what to do.
3. The **Data Layer** stores or provides the required data and sends it back through the logic layer to the user.

**Example**

**E-commerce Website**:

* Presentation → Web pages with products, cart, checkout.
* Business Logic → Checks stock, payment, and order rules.
* Data Layer → Stores user details, product list, and orders.

**Advantages**

* **More secure** → Data and logic are separated.
* **Easier to maintain** → Changes in one layer don’t break the others.
* **Scalable** → Can handle many users.

**Disadvantages**

* **More complex** than 1-tier and 2-tier.
* **Costly to develop** because it needs more planning and resources.

