● What is Emmet?

Emmet is the essential toolkit for web-developers. It allows you to type shortcuts that are then expanded into full-fledged boiler plate code for writing HTML and CSS

● Difference between a Library and Framework?

A key difference between the two is Inversion of control. When using a library, the control remains with the developer who tells the application when to call library functions. When using a framework, the control is reversed, which means that the framework tells the developer where code needs to be provided and calls it as it requires.

● What is CDN? Why do we use it?

A Content Delivery Network (CDN) is a network of servers strategically distributed across different geographical locations. The primary purpose of a CDN is to deliver content (such as images, videos, HTML, JavaScript, CSS files, etc.) to users more quickly and efficiently.

How a CDN Works:

Distributed Servers: A CDN has multiple servers located in different regions. These servers, known as edge servers or PoPs (Points of Presence), cache copies of the content from the origin server.

User Requests: When a user requests a piece of content (like a webpage or a video), the request is routed to the nearest edge server in the CDN. This reduces the distance the data has to travel, leading to faster load times.

Caching: CDNs cache the content on their servers. When content is requested, the CDN serves the cached version. If the content isn't already cached, the CDN fetches it from the origin server, caches it, and then delivers it to the user.

Why Do We Use CDNs?

Improved Performance and Reduced Latency: By serving content from a server closer to the user, CDNs reduce the time it takes for the data to travel, resulting in faster load times and a better user experience.

Scalability: CDNs can handle large volumes of traffic, making them ideal for handling spikes in traffic (e.g., during a product launch or viral event) without overwhelming the origin server.

Reliability: If one server in the CDN fails, others can step in to serve the content, enhancing the reliability and availability of the service.

Security: CDNs can provide additional security features, such as DDoS (Distributed Denial of Service) protection, to help safeguard websites and services.

Bandwidth Savings: By caching content, CDNs reduce the amount of data the origin server needs to serve, saving bandwidth costs.

Overall, CDNs are essential for delivering fast, reliable, and secure content to users, particularly for websites and services with a global audience.

● Why is React known as React?

React is a JavaScript Library. The name ‘React’ was chosen because the library was designed to allow developers to react to changes in state and data within an application, and to update the user interface in a declarative and efficient manner.

● What is crossorigin in script tag?

The crossorigin attribute in the <script> tag is used to specify how the browser should handle CORS (Cross-Origin Resource Sharing) requests for the script. This is particularly relevant when the script is being fetched from a different origin (domain) than the current page, which can involve security considerations and affect the way browsers handle cookies, authentication, and other data.

The crossorigin attribute can take the following values:

anonymous: This value means that the request for the script is made without credentials (such as cookies or HTTP authentication). If the server does not include the appropriate CORS headers, the resource will not be loaded.

use-credentials: This value means that the request for the script will include credentials, such as cookies or HTTP authentication. The server must respond with appropriate CORS headers that allow the credentials to be sent. If the server does not include these headers, the script will not be loaded.

If the crossorigin attribute is omitted, the browser may default to anonymous behavior, meaning the request will not include credentials.

● What is diference between React and ReactDOM

React and ReactDOM are two different packages in the React ecosystem, each serving distinct purposes:

React:

Core Library: React is the core library responsible for defining and managing components. It provides the tools and APIs to create and manage the component lifecycle, state, props, and other fundamental concepts.

Rendering Logic: React itself doesn't directly manipulate the DOM. It focuses on the component logic and building component trees.

Platform-Agnostic: React is platform-agnostic, meaning it can be used to build user interfaces for various platforms, not just web browsers. For example, React Native uses React to build mobile apps.

ReactDOM:

DOM-Specific: ReactDOM is the package that interacts with the DOM. It provides methods to render React components to the DOM and manage them.

Rendering to the Web: When you want to render a React component to the web, you use ReactDOM's ReactDOM.render() method. This method takes a React component and an HTML element as arguments and renders the component into the specified element.

Specialized for Web: ReactDOM is specific to web applications. For other platforms like mobile (using React Native), different packages handle the rendering.

In summary, React provides the framework and tools to build component-based UIs, while ReactDOM takes those components and renders them into the web browser's DOM.

● What is difference between react.development.js and react.production.js files via CDN?

The react.development.js and react.production.js files are different versions of the React library, tailored for specific use cases and environments:

1. react.development.js: Development Build

Purpose: This file is optimized for development and debugging.

Features:

Verbose Error Messages: It provides detailed error messages and warnings, helping developers identify and fix issues more easily. This includes helpful messages for common mistakes and potential problems.

Unminified Code: The code is not minified, making it easier to read and understand. This is useful when debugging, as stack traces and source maps are more readable.

Performance Considerations: The development build includes additional checks and validations that are useful during development but can impact performance. For example, it might perform extra checks to ensure props and state are correctly managed.

2. react.production.js: Production Build

Purpose: This file is optimized for performance and used in production environments where the application is deployed to end users.

Features:

Minified and Optimized: The code is minified and optimized, resulting in a smaller file size. This reduces the amount of data that needs to be downloaded by the user, improving load times.

No Extra Debugging Overhead: It does not include detailed error messages or warnings. This helps improve performance and security by not exposing potentially sensitive information about the application.

Performance: The production build omits the development-only checks and optimizations, ensuring that the application runs as efficiently as possible.

Summary:

Development Build (react.development.js): Includes detailed error messages, warnings, and other tools to aid developers during the development process. It is unminified and not optimized for performance.

Production Build (react.production.js): Minified and optimized for performance, with stripped-down error messages and warnings. This version is intended for use in live, production environments where performance and efficiency are crucial.

When using React via a CDN, it's important to use the appropriate build for your environment: react.development.js for development and testing, and react.production.js for production deployment.

● What is async and defer?

async and defer are attributes that can be added to the <script> tag in HTML to control how and when external JavaScript files are loaded and executed. These attributes are particularly useful for improving the performance of web pages by optimizing script loading.

1. async Attribute

Behavior: When a script tag with the async attribute is encountered, the browser will download the script file asynchronously (i.e., without blocking the parsing of the HTML document). Once the script is downloaded, it is executed immediately, potentially before the rest of the page has finished parsing.

Use Case: async is useful for scripts that are independent of each other and do not rely on the DOM or other scripts. It's commonly used for analytics scripts or ads that do not need to wait for other scripts to run.

Example:

html

Copy code

<script src="script.js" async></script>

2. defer Attribute

Behavior: When a script tag with the defer attribute is encountered, the browser downloads the script file asynchronously, just like with async. However, the script will not be executed until the HTML document has been completely parsed. Scripts with the defer attribute are guaranteed to execute in the order they appear in the document.

Use Case: defer is suitable for scripts that need to interact with the DOM or depend on other scripts being executed first. It's often used for large scripts or libraries that need to manipulate the DOM once it is fully loaded.

Example:

html

Copy code

<script src="script.js" defer></script>

Key Differences:

Execution Timing:

async: Scripts are executed as soon as they are downloaded, which can happen before or after the document has been fully parsed.

defer: Scripts are executed in the order they appear in the document, but only after the entire HTML document has been parsed.

Order of Execution:

async: The order of execution is not guaranteed; scripts may execute as soon as they are ready.

defer: Scripts are executed in the order they are encountered in the document.

Using async and defer helps optimize page load performance by allowing scripts to load in parallel with other resources, reducing the blocking time during page rendering.