# **Disclaimer:**

COPYRIGHT NOTICE

©All content, materials, and intellectual property contained within this document, including but not limited to text, images, diagrams, graphics, code, and any other original works, are protected by copyright laws and international treaties. This intellectual property is the sole and exclusive property of SRIKANTH TEKUMUDI

Unauthorized use, reproduction, distribution, modification, or transmission of any portion of this document, in any form or by any means, electronic or mechanical, without the prior written permission of SRIKANTH TEKUMUDI, is strictly prohibited and may result in severe civil and criminal penalties.

By accessing, viewing, or using this document, you acknowledge and agree to abide by all applicable copyright laws and the terms and conditions outlined herein. This copyright notice serves as a legal warning and reminder that the unauthorized use of this intellectual property is strictly prohibited and will be vigorously enforced.

Thank you for respecting the intellectual property rights of SRIKANTH TEKUMUDI

### Web security and performance questions are common in frontend interviews. So here are must-know concepts for interviews

### **Explain web Vitals and how to imporve? Or What are Web Vitals (LCP, FID, CLS)?**

Web Vitals are a set of metrics introduced by Google to measure the user experience (UX) of a web page. They focus on three key aspects:

1. **Load Performance:** Measured by **Largest Contentful Paint (LCP)**, this metric indicates how long it takes for the largest piece of content on your page to load. Ideally, LCP should be less than 2.5 seconds for a good user experience.
2. **Interactivity:** Measured by **First Input Delay (FID)**, this metric shows the time it takes for a page to become interactive after it loads. This means how long a user has to wait before they can click a button or interact with the page in any way. A good FID score is less than 100 milliseconds.
3. **Visual Stability:** Measured by **Cumulative Layout Shift (CLS)**, this metric reflects how much the layout of your page shifts unexpectedly while loading. This can be jarring for users and make it difficult to interact with the page. A good CLS score is less than 0.1.

By focusing on these core web vitals, developers can ensure their web pages are:

* **Fast to load:** Users don't want to wait long periods for content to appear.
* **Responsive:** Users should be able to interact with the page quickly after it loads.
* **Stable:** The layout of the page shouldn't jump around unexpectedly, frustrating users.

Google also considers these web vitals as ranking factors for search engine results pages (SERPs). This means that pages with good web vital scores are more likely to appear higher in search results.

### **LCP vs FCP**

LCP (Largest Contentful Paint) and FCP (First Contentful Paint) are both important web performance metrics used to measure how quickly content is displayed to users, but they focus on different aspects of the loading process.

### **First Contentful Paint (FCP)**

**Definition**: FCP measures the time from when the page starts loading to when any part of the page's content is rendered on the screen. This content can be text, an image, or a canvas render.

**Focus**: FCP is a key milestone in the loading process because it indicates that the browser has rendered something useful to the user, breaking the initial blank screen and giving users a visual cue that the page is loading.

**How it's measured**:

* FCP is typically the first meaningful visual response the user sees.
* It does not necessarily mean that the content is fully loaded or interactive, just that something is visible.

**Example**:

* A webpage starts loading.
* The first piece of content (e.g., a header or a paragraph) appears on the screen.
* The time taken from the initial load to this point is the FCP.

### **Largest Contentful Paint (LCP)**

**Definition**: LCP measures the time from when the page starts loading to when the largest piece of content (e.g., an image, video, or a large block of text) is rendered on the screen. This metric focuses on the main content of the page that is relevant to the user.

**Focus**: LCP is a more meaningful measure of perceived load speed for users, as it indicates when the main content of the page has finished loading and is visible. This is crucial for user experience because it reassures users that the page is useful and nearly ready.

**How it's measured**:

* LCP is measured by identifying the largest content element within the viewport.
* It occurs later in the page load process compared to FCP, often after several render cycles.

**Example**:

* A webpage starts loading.
* Various pieces of content (e.g., header, images, text) start appearing.
* The largest piece of content (e.g., a hero image or a large block of text) finishes rendering.
* The time taken from the initial load to this point is the LCP.

### **Key Differences**

* **Content Rendered**:
  + **FCP**: Measures the time until any content is visible.
  + **LCP**: Measures the time until the largest content element is visible.
* **User Experience Impact**:
  + **FCP**: Indicates that the page is starting to load and provides the first visual feedback.
  + **LCP**: Indicates that the main content has loaded, providing a more complete user experience.
* **Measurement Timing**:
  + **FCP**: Typically happens earlier in the loading process.
  + **LCP**: Happens later, as it waits for the largest content element to load.

### **Visualization**

Imagine a webpage with a large hero image, some introductory text, and a main article:

1. **Initial Load**: The browser starts loading the page.
2. **FCP**: The introductory text appears first on the screen (FCP recorded).
3. **Further Loading**: Images and other elements continue to load.
4. **LCP**: The large hero image, which is the largest content element, finishes loading (LCP recorded).

* **FCP** measures the time until the first piece of content is visible, giving an early indication of page loading progress.
* **LCP** measures the time until the largest piece of content is visible, providing a more complete picture of when the main content has loaded.

Both metrics are important for understanding and improving user-perceived performance, but they highlight different stages of the loading process.

Here's a quick summary of the three core web vitals:

| **Metric** | **Description** | **Ideal Score** |
| --- | --- | --- |
| Largest Contentful Paint (LCP) | Time to load the largest piece of content | < 2.5 seconds |
| First Input Delay (FID) | Time to become interactive after loading | < 100 milliseconds |
| Cumulative Layout Shift (CLS) | Amount of unexpected layout shift during loading | < 0.1 |

**Finding Web Vitals Metrics:**

There are several tools you can use to measure your website's Core Web Vitals:

1. **Google Search Console (GSC):**
   * Go to the "Core Web Vitals" report in GSC.
   * This report provides an overview of your site's performance across different URLs and mobile/desktop devices.
   * It also identifies specific pages that need improvement and offers potential causes for the issues.
2. **PageSpeed Insights (PSI):**
   * Enter your website URL in PageSpeed Insights:<https://developers.google.com/speed/pagespeed/insights/>.
   * PSI analyzes your page and provides a score for LCP, FID, CLS, and other performance metrics.
   * It generates recommendations and suggestions to improve your scores.
3. **Chrome DevTools:**
   * Open Chrome DevTools (F12 key).
   * Go to the "Performance" tab.
   * Click on the "Web Vitals" audits category.
   * This provides detailed insights into LCP, FID, CLS, and other performance factors.

**Improving Web Vitals:**

Once you've identified areas for improvement, you can implement these strategies:

1. **Optimize Images:**
   * Reduce image file sizes using compression tools or alternative formats (e.g., WebP).
   * Consider lazy loading images, which delays loading them until they are about to be displayed.
   * Set appropriate width and height attributes for images to prevent layout shifts.
2. **Minimize Rendering Blocking Resources:**
   * Identify and defer non-critical CSS and JavaScript files that block the initial rendering of your page.
   * Consider code splitting to load only the necessary code for each page.
   * Minify and combine CSS and JavaScript files to reduce their size.
3. **Leverage Browser Caching:**
   * Implement caching mechanisms for static assets (images, CSS, JavaScript) so browsers don't need to download them repeatedly.
   * Use browser caching headers like Cache-Control and Expires to control how long browsers store these resources.
4. **Improve Server Response Time:**
   * Upgrade your web hosting plan if your server is slow.
   * Reduce the number of database queries and optimize backend code.
   * Use a Content Delivery Network (CDN) to deliver content from geographically distributed servers, reducing latency.
5. **Reduce Third-Party Scripts:**
   * Minimize the use of external scripts that can slow down your page load.
   * If necessary, consider asynchronous loading of third-party scripts.
6. **Prioritize Mobile Optimization:**
   * Use responsive design techniques to ensure your website displays well on all devices, especially mobile.
   * Test your website performance on various mobile devices using tools like Google's Mobile-Friendly Test (<https://chrome.google.com/webstore/detail/responsive-tester/ppbjpbekhmnekpphljbmeafemfiolbki>).

### **Optimizing Web app:**

**Optimizing Web App Performance with Network, Code, and Rendering Techniques:**

A well-performing web app delivers a seamless user experience. Here's how to leverage network, code, and rendering optimizations to achieve this:

**Network Optimization:**

* **GZIP Compression:** Compress static assets (JavaScript, CSS, HTML) using GZIP to reduce their size by up to 70%, minimizing network transfer times. Modern browsers automatically decompress GZIPped files.
* **Reduce JavaScript Payload:** Minimize the amount of JavaScript shipped to the browser. This can involve:
  + Code splitting: Load only the code needed for a specific page or interaction, reducing initial payload size.
  + minification and uglification  
      
     minification plugin: [\*\*TerserWebpackPlugin](https://webpack.js.org/plugins/terser-webpack-plugin/) ,[CssMinimizerWebpackPlugin](https://webpack.js.org/plugins/css-minimizer-webpack-plugin/),[UglifyjsWebpackPlugin](https://v4.webpack.js.org/plugins/uglifyjs-webpack-plugin/)\*\*
  + Third-party library evaluation: Carefully evaluate the necessity of third-party libraries and consider alternatives that might be smaller or offer lazy loading.
  + Dead code elimination: Use build tools like Webpack or Rollup to remove unused code from final bundles.
* **Content Delivery Network (CDN):** Host static assets on a CDN to serve them from geographically distributed servers, minimizing latency for users worldwide. This improves perceived performance, especially for users in regions far from your main server.

**Code Optimization:**

* **Compilation/Bundling:** Use tools like Webpack or Rollup to compile and bundle JavaScript and CSS code. This offers various benefits:
  + Minification: Removes unnecessary whitespace, comments, and formatting, reducing file size.
  + Concatenation: Combines multiple files into one, reducing the number of HTTP requests.
  + Polyfilling: Provides compatibility for older browsers or features with modern syntax.
* **Framework-Specific Optimizations:** Leverage optimizations specific to your chosen framework (e.g., React's React.memo for memoizing components).
* **Type Systems:** Consider using TypeScript or Flow for static type checking. This can help catch potential errors early and improve code maintainability, leading to cleaner, more efficient code.

**Rendering Optimization:**

**Critical Rendering Path (CRP) Optimization:**

* The CRP refers to the minimum resources needed for a browser to render the initial view of your webpage. Optimizing this path is crucial for fast first impressions.  
    
   it includes prefetching crititcal resouces  
    
   asynchrous module loading etc
* **Reduce Repaints:** Repaints occur when the entire viewport or a large portion needs to be re-drawn. Minimize them by:  
  + Avoiding unnecessary DOM manipulations (e.g., excessive setState calls in React).
  + Using techniques like virtual DOM libraries (React) to batch updates and minimize browser reflows.
* **Reduce Composite Operations:** Composite operations involve combining multiple paint layers to create the final rendered image. Minimize them by:  
  + Using hardware acceleration with CSS transform: translate3d(0, 0, 0) or GPU-accelerated libraries (e.g., Three.js for 3D graphics).
  + Simplifying complex layouts and using CSS grid or flexbox for better performance.
* **Lazy Loading:** Load components or content only when they are about to be displayed in the viewport. This reduces the initial load time and improves perceived performance. Popular libraries like React Lazy provide convenient ways to implement lazy loading.

**Additional Considerations:**

* **Code Splitting Strategies:** Choose between route-based, on-demand, or code-splitting by feature to optimize loading based on your app's structure and user experience requirements.
* **Caching:** Implement browser caching for static assets using mechanisms like Cache-Control or Expires headers. This reduces server load and network requests for repeat visitors.
* **Server-Side Rendering (SSR):** Consider SSR (React with Next.js, Nuxt.js for Vue) or frameworks like Gatsby for static site generation (SSG) to improve initial load time, especially for SEO-critical pages.

**Prioritization and Tools:**

* **Identify Performance Bottlenecks:** Analyze your web app's performance using tools like Chrome DevTools' Performance tab, Lighthouse audits, or WebPageTest. Focus on optimizing areas with the most significant impact.
* **Gradual Optimization:** Implement optimizations iteratively, measuring the impact on user experience and metrics (e.g., Largest Contentful Paint, First Input Delay) after each change.

By effectively combining network, code, and rendering optimizations, you can create a highly performant web app that delivers a delightful user experience. Remember to find the best balance for your specific app's needs and complexity.

### **Explain pillars of web security ?**

4 pillars of web security:

Cross-site scripting XSS CSRF Clickjacking CSP - Content security policies

**1. Cross-Site Scripting (XSS):**

* **Description:** XSS occurs when an attacker injects malicious scripts (usually JavaScript) into a website. These scripts then run in the victim's browser, potentially stealing data, redirecting users, or performing other harmful actions.
* **Example:** An attacker injects a script into a forum comment that steals the session cookies of other users who view the comment.
* **Prevention:**
  + **Input validation and sanitization:** Validate all user input to ensure it's in the expected format and remove any potentially malicious code. Sanitize any untrusted input before processing it.
  + **Use prepared statements:** When interacting with databases, use prepared statements to prevent attackers from crafting SQL injection attacks.
  + **Output encoding:** Encode any user-generated content that is displayed back to the user to prevent it from being interpreted as code.

**2. Cross-Site Request Forgery (CSRF):**

* **Description:** CSRF attacks exploit a user's existing authenticated session to perform unintended actions on a website. The attack tricks the user's browser into sending a forged request to the targeted website.
* **Example:** An attacker sends an email to a user containing a link that, when clicked, triggers a request to transfer money from the user's bank account.
* **Prevention:**
  + **CSRF tokens:** Generate unique CSRF tokens for each user action and include them as hidden form fields or headers in requests. Verify the token on the server-side before processing the request.
  + **SameSite cookies:** Configure session cookies with the SameSite=Strict attribute to prevent them from being sent for cross-site requests, mitigating CSRF attacks.

**3. Clickjacking:**

* **Description:** Clickjacking involves tricking a user into clicking on a hidden element (e.g., a button) overlaid on legitimate content. When the user clicks, they unknowingly interact with the hidden element, potentially performing an unauthorized action or visiting a malicious website.
* **Example:** An attacker creates a transparent button overlaid on a "Like" button on a social media platform. When the user tries to click the "Like" button, they unknowingly click the hidden button, which could be used to like a malicious post or steal their credentials.
* **Prevention:**
  + **Frame-busting:** Use the X-Frame-Options HTTP header to prevent your web page from being loaded within a frame (iframe) of another website.
  + **Content Security Policy (CSP):** Configure CSP to restrict the sources of resources (e.g., images, scripts) that can be loaded on your website. This can prevent attackers from embedding malicious content in frames.

**4. Content Security Policy (CSP):**

* **Description:** CSP is a security mechanism that allows you to define a whitelist of sources from which your web page can load resources (scripts, images, stylesheets, etc.). This helps mitigate various attacks, including XSS, clickjacking, and data injection attacks.
* **Example:** You can define a CSP policy that only allows scripts to be loaded from your own domain and a trusted CDN provider. This prevents attackers from injecting malicious scripts from other sources.
* **Prevention (through CSP):**
  + Define a whitelist of trusted sources for loading resources.
  + Consider using hash-based or nonce-based CSP directives for even stricter controls over script execution.

By understanding and implementing these security measures, you can significantly improve the security of your web applications. Remember, security is an ongoing process, so it's important to stay informed about new vulnerabilities and update your security practices accordingly.

### **Content Security Policy**

Here's a breakdown of CSP (Content Security Policy) tailored for explaining it in an interview:

**What is CSP?**

* CSP is a security layer implemented in web browsers that defines which resources a web page can load.
* It acts as a whitelist, specifying allowed sources for scripts, stylesheets, images, fonts, and other content.
* By restricting where resources can come from, CSP helps mitigate various web security attacks like Cross-Site Scripting (XSS) and injection attacks.

**Benefits of CSP:**

* **Reduced Attack Surface:** By limiting the allowed sources, CSP makes it harder for attackers to inject malicious code into your web application.
* **Improved Security Posture:** CSP helps prevent common web vulnerabilities, leading to a more secure web application.
* **Content Delivery Network (CDN) Compatibility:** CSP can be configured to work with CDNs, allowing for efficient content delivery while maintaining security.

**Key CSP Directives:**

* script-src: Defines allowed sources for loading scripts.
* style-src: Defines allowed sources for loading stylesheets.
* img-src: Defines allowed sources for loading images.
* font-src: Defines allowed sources for loading fonts.
* object-src: Defines allowed sources for loading objects (e.g., Flash, iframes).
* frame-ancestors: Defines allowed origins for embedding your page within frames.
* base-uri: Defines allowed base URIs for resolving relative URLs within the page.
* form-action: Defines allowed origins for submitting forms.

<meta http-equiv="Content-Security-Policy" content="default-src 'self'; script-src 'self' <https://apis.google.com>; img-src 'self' data:; style-src 'self' 'unsafe-inline';">

**Implementing CSP:**

* CSP can be implemented through:
  + HTTP headers: Setting the Content-Security-Policy header in the server response.
  + <meta> tag: Adding a <meta http-equiv="Content-Security-Policy" content="..."> tag within the <head> of your HTML document.

**Example CSP Header:**

Content-Security-Policy: script-src 'self' '<https://cdn.example.com>'; style-src 'self' '<https://cdn.example.com/styles.css>'; img-src 'self' '<https://images.example.com>';

**Tips for a Successful CSP Interview:**

* **Demonstrate Understanding:** Explain the purpose and benefits of CSP, highlighting its role in web security.
* **Familiarity with Directives:** Be comfortable explaining the common CSP directives and their functions.
* **Implementation Knowledge:** Briefly discuss how CSP can be implemented through HTTP headers and the <meta> tag.
* **Real-World Examples:** If possible, share an example of how you've implemented CSP in a project or the benefits you observed from its use.

**Additional Points:**

* CSP is a powerful tool, but it's crucial to configure it correctly to avoid unintended consequences and ensure proper website functionality.
* CSP can be fine-tuned to allow specific resources while blocking others, providing a balance between security and functionality.

When starting your journey to improve website performance, selecting the right tools is essential. While Lighthouse is commonly used, it mainly offers lab data, which might not reflect real-world conditions.

For a quick overview of how your website actually performs in real user scenarios, consider exploring tools that provide real user data. These tools can give you an immediate understanding of your site's performance, helping you make informed decisions to improve speed and user experience.

Our top 5 (in random order)

1 Chrome User Experience Report (CrUX)

* Features: Real-world user data, historical trends
* Benefits: Reflects actual user experiences

2 WebPageTest by Catchpoint

* Features: Tests from multiple locations, real browsers
* Benefits: Detailed performance insights

3 RUMvision

* Features: Tracks Core Web Vitals over time
* Benefits: Offers historical insights and trend analysis for website performance

4 Request Metrics

* Features: Core Web Vitals testing and monitoring
* Benefits: Provides detailed reports and performance tracking to optimise web vitals

1. Treo

* Features: Site speed analysis and performance monitoring
* Benefits: Delivers geographic performance insights and continuous site speed monitoring

Links to the free tools can be found in the comments:

### **Advantages of Http 2 ove http 1.1**

HTTP/2 offers several advantages over HTTP/1.1, addressing many of the inefficiencies of the older protocol and providing a more efficient way to deliver web content. Here are the key advantages:

### **1. Multiplexing**

**HTTP/1.1**:

* Each request/response pair requires its own connection, or at best, limited pipelining which can lead to head-of-line blocking.

**HTTP/2**:

* Allows multiple requests and responses to be sent simultaneously over a single TCP connection. This eliminates head-of-line blocking and improves resource utilization.

### **2. Header Compression**

**HTTP/1.1**:

* Headers are sent as plain text and are repeated for each request, which can be redundant and inefficient, especially with larger headers.

**HTTP/2**:

* Uses HPACK header compression to reduce the size of headers, decreasing the amount of data that needs to be transmitted and thus speeding up the communication.

### **3. Stream Prioritization**

**HTTP/1.1**:

* Does not support stream prioritization, leading to potential inefficiencies when loading resources.

**HTTP/2**:

* Allows clients to prioritize streams, enabling more important resources (like the main HTML document) to be delivered faster than less important ones (like images or ads).

### **4. Single Connection**

**HTTP/1.1**:

* Multiple connections are often opened to the same server to fetch resources, leading to increased latency and resource contention.

**HTTP/2**:

* Uses a single connection per server to handle multiple requests, reducing the overhead of opening and closing connections and better utilizing network resources.

### **5. Server Push**

**HTTP/1.1**:

* Servers can only respond to client requests, leading to additional round trips to fetch resources that the server knows the client will need.

**HTTP/2**:

* Allows the server to proactively "push" resources to the client that it anticipates will be needed, reducing latency by eliminating the need for additional request/response cycles.

### **6. Improved Security**

**HTTP/1.1**:

* Supports both HTTP and HTTPS, but adoption of HTTPS is slower and not enforced.

**HTTP/2**:

* While not required, it is most commonly used with HTTPS, encouraging better security practices and encryption for data in transit.

### **7. Reduced Latency**

**HTTP/1.1**:

* Experiences higher latency due to the need for multiple connections and inefficient use of network resources.

**HTTP/2**:

* Lower latency is achieved through multiplexing, header compression, and server push, which reduces the overall time to load a web page.

### **8. Better Handling of Network Resources**

**HTTP/1.1**:

* Network resources can be poorly utilized, leading to bottlenecks and inefficiencies.

**HTTP/2**:

* Optimizes the use of network resources with features like multiplexing and stream prioritization, leading to faster and more reliable web page loading.