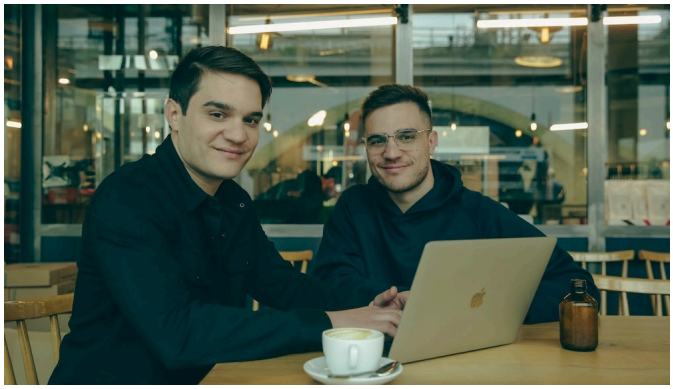
🎯 25 Most Asked Interview Questions for JavaScript Devs in 2025

(For Frontend, Backend & Full-stack Interviews)



Dragos & Bogdan - Senior Engineers and Founders at theSeniorDev.com

🚨 **Note:** All interview questions in this document are REAL questions our mentees faced in REAL technical interviews in the first half of 2025.

🧪👩‍💻 If you want more interview questions, take this **free 10-minutes technical assessment** on our website: **[Free 10 Minutes Technical Assessment.]**

1. Frontend Interview Questions 💅

🔍 **Q.1)** What's the difference between `'async'` and `'defer'` attributes when applied to a `<script/>` tag? Describe a specific use case for each one of them.

💡 **Answer:** "Both `'async'` and `'defer'` mean that the script will be downloaded in parallel to the HTML document being parsed.

The difference is `'async'` scripts will be interpreted as soon as they finish downloading, whereas `'defer'` will be interpreted after the entire HTML document has been parsed. Regarding use cases: `'defer'` is great for scripts that need access to the DOM (like the React scripts that render the App)… And `'async'` is best for scripts that don't need the DOM to be rendered (for instance analytics)."

🔍 **Q.2)** JavaScript is a **pass-by-reference** or **pass-by-value language**?

Explain.

💡**Answer:** JavaScript is a **pass-by-value language**.

That means when arguments are passed into functions, a copy of them is made in memory. If you change the value of an argument in the function code, the original variable will NOT be affected.

The confusion comes when we handle **Objects**.

When an **Object** is passed as a function argument, what we get is a copy of a pointer to that object. And so, a copy of a pointer, is still pointing to the original object. Changing the **Object** inside a function will change the original object.

Remember, **JavaScript is pass-by-value** - even if when dealing with **Objects** it gives the impression of being pass-by-reference.

🔍 **Q.3)** Using **closures** makes **JavaScript** consume more?

- **a)** CPU
- **b)** Disk Space
- **c)** Memory
- **d)** Network bandwidth

Explain why and **how closures work in JavaScript.**

💡**Answer:** Closures basically mean that every function “remembers” the **lexical scope** in which it was created.

This means that the garbage collection algorithm cannot remove any variables from that scope as long as the function is being used or referenced. Which makes **JavaScript a very memory heavy language** (heap memory, which is stored in RAM).

The correct answer is **C) Memory**, which is where pressure will be applied when using closures.

🔍 **Q.4)** How can you improve a bad **CLS (Cumulative Layout Shift)?**

💡**Answer:** Cumulative layout shift happens because either elements are added to the DOM as we render (client-side rendering), or, the height or width of specific DOM elements changes as we load the page (usually because of CSS, fonts, or images loading too late).

To fix a bad CLS score, we can:

1. Extract **Critical CSS** (the CSS needed to render the above the fold part of the website)

2. Specify `'width'` and `'height'` attribute for `<img/>` elements (the browser will render them correctly even if they didn't load yet in the layout and painting phase.

3. Use a **Skeleton Library** to create high fidelity placeholders for elements that will load a bit later (because they depend on data being fetched, eg: **react-loading-skeleton**).

🔍 **Q.5)** Which **web performance metric** is most affected by excessive re-renders in web frameworks such as **React, Vue, or Angular?**

💡**Answer:** Excessive re-renders will decrease your **INP** score (**Interaction to Next Paint**). This is basically the time it takes from a user interaction to re-painting the user interface.

Google establishes a good INP to be **under 200ms**.

Poor INP scores come from unnecessary re-renders (can be fixed with React.Memo, useMemo(), and useCallback()... and by using the Early Return Pattern in Component Design).

🔍 **Q.6)** Give an example of something you can do with **Class Components** that you cannot do with **Functional Components** in React. Explain why.

💡**Answer: Class Components** allow you to hook in the **Component Lifecycle** and add specific logic in **Lifecycle Methods** like `'componentDidCatch()'` and `'getDerivedStateFromError()'`.

Although not widely used, they come in handy if you are building a specific use case - such as a logging library (is most likely what libraries like `react-sentry` use under the hood).

🔍 **Q.7)** Which of the following **Web Performance Metric** will improve the most when using **Server-Side Rendering:**

a) FCP (First Contentful Paint)

b) FID (First Input Delay)

c) TTFB (Time To First Byte)

d) LCP (Largest Contenful Paint)

Explain each metric shortly.

💡**Answer:** The metric that will improve the most by using **SSR (Server Side Rendering)** is option d) LCP (Largest Contenful Paint).

The reason behind this is that the server did the heavy lifting by pre-rendering the React component tree into HTML, which the web browser will be able to pain as soon as it is received.

SSR also improves the CLS (Cumulative Layout Shit) - see previous question - because the elements come pre-rendered, preventing “the white screen of death” (WSoD) 💀

Short Explanation of each performance metric:

a) FCP (First Contentful Paint) -> Time to render first element on screen.

b) FID (First Input Delay) -> Time until the page becomes responsive to input (eg. keyboard stroke/click)

c) TTFB (Time To First Byte) -> Time it takes for the web browser to receive the first byte of response from the server.

d) LCP (Largest Contenful Paint) -> Time it takes to render what the browser identifies as the largest element.

🔍 **Q.8)** What are **TypeScript Generics?** Give me an example of when you should use them?

💡**Answer: Generics in TypeScript** are code constructs that can receive **Type arguments**. We use them to provide type flexibility while maintaining strict type checking.

An example of a generic function is `useState()` in React. It takes a function argument (the initial value of the State) and an optional type argument, which is the type of the State.

`'const [counter, setCounter] = useState<number>(0)'`

If you are a JavaScript Developer, we will get you to Senior, or you don't pay!

-> See how we get you to Senior here!

Want to find your **Technical Gaps to Senior**? Take this **free 10-Minute Technical Assessment** to find your gaps to Senior level:

-> Take the Free Technical Assessment!

2. Full-stack Interview Questions 🛠️

🔍 **Q.1)** What's the difference between `git rebase` and `git merge`?

💡**Answer:** Both commands, `'git rebase'` and `'git merge'`, will merge branches… But `'git rebase'` will re-write the git history to add the commits being merged at the top of the commit history (regardless of when changes were committed). In general, `'git rebase'` will keep a cleaner history.

🔍 **Q.2)** Which **HTTP Status Code** represents **Forbidden Access** to the endpoint/resource?

- **a)** 503
- **b)** 204
- **c)** 403
- **d)** 401
- **e)** 302

Explain.

💡 **Answer:** Client errors are represented by a **4XX** code in **HTTP… 401** meaning the client did not provide credentials, and it is unauthorized, and **403** meaning the client did provide credentials, but they don't have access to the specific resource.

The correct answer is **c) 403.**

🔍 **Q.3)** What is a **preflight request** in the context of **HTTP**?

a) A request by the client to obtain **auth credentials**

b) A request to **check if a cached asset** is still fresh

c) A request to check the **server is down**

d) A request to check **CORS status** with the server

e) A request to update the `max age` property of a static asset

Explain.

💡 **Answer:** A preflight request is an **HTTP request** made by the browser before the actual request, to check the **CORS status**. It is an **OPTIONS request** to which the server needs to respond with the allowed origins, methods and headers. Preflight requests are triggered automatically by the browser **whenever you make a fetch request to a domain other than your own.**

🔍 **Q.4) Content Negotiation** is a mechanism used to…

a) Server the same resources in a different format - for example compressed - based on **HTTP headers** sent by client

b) Return a resource from the browser cache to avoid extra HTTP requests and improve performance

c) A mechanism used to encrypt the request body and headers, as well as the response when using HTTPS

d) A mechanism to negotiate and establish the SSL handshake

Explain.

💡 **Answer:** In **HTTP** a resource can be served in different formats. For example, a CSS file can be served directly as .CSS, or compressed as .CSS.gzip.

The client (web browser) can specify if it prefers any of the formats by using the `'Accept'` header. `'Accept: gzip, br'` tells the server that if it has a compressed version of the asset, either with gzip or brotli it can serve that one. This process is called **Content Negotiation**, and it enables end-to-end compression.

🔍 **Q.5)** Which of the following are **valid ways to version an API** in the case of **breaking changes**?

- **a)** Adding a version number to the **URI (api.com/v2)**
- **b)** Adding a version number as a **subdomain (v2.api.com)**
- **c)** Adding a version number to the query **parameters of the URI (api.com?version=v2)**
- **d)** Adding a version number to the **'Accept' header** ('Accept: v2')
- **e)** Adding a version number to the **body** of the request ('{ version: 2}')

Explain.

💡**Answer:** API versioning can be done with the URI, query parameter, and the `'Accept'` header. The most common is using URI parameters because of better separation of concerns, reduces errors due to caching, and it is easier for frontend developers to switch between versions (you just switch the URL).

🔍 **Q.6)** Which of the following methods are **idempotent in REST**?

- **a)** GET
- **b)** PATCH
- **c)** PUT
- **d)** DELETE
- **e)** POST

Explain.

💡 **Answer: Idempotency** means that doing the same action, over and over again, will not cause more effects.

Example: *Multiplying a number by 1 is and idempotent operation - the results will stay the same not matter how many times you do it - but adding 1, is not - every time you do it the final results will change).*

In REST, GET, PUT and DELETE are idempotent. POST and PATCH are not idempotent. This means for example, making the same GET request will not change the state/data on the server. This is very important in Microservices, where network conditions might cause the client to re-try a request that already went through.

🔍 **Q.7)** What's the difference between **Web Sockets** and **Server Sent Events** for **real time communication** between client and server?

Explain.

💡 **Answer:** Web Sockets are used for when the communication is bidirectional (the client sends events to the server, and the server sends events to the client.

In server-sent events, the communication is unidirectional (the servers sends events to the client, but the client is just listening).

Web Sockets are used in **live chat applications**. **Sever-Sent Events** are used in **LLM powered applications like ChatGPT** or real time applications (like **trading platforms**).

🔍 **Q.8)** Can you define **Cyclomatic Complexity?** And when would you use it?

💡**Answer: Cyclomatic complexity** is a code quality metric that measure the number of linearly independent paths through a code snippet - for example - an `'if else'` statement has a complexity of 2 (there are 2 different paths the code can take).

Code quality tools like **Sonar Cube** can measure the cyclomatic complexity of your code on every commit.

3. Backend Interview Questions ⚙️

🔍 Q.1) A **database index** makes database queries faster by…

a) Caching the **query result** for as long as the underlying data does not change

b) Creating a data structure that allows **binary search** for the index column

c) Ensuring duplicate values are automatically removed from the table and search queries

d) Creating **Read-Only replicas** of the indexed table that can be queried in parallel

Explain.

💡**Answer:** A **database index** will make querying a table much faster, by creating an underlying data structure **(B-tree)** which can be read in logarithmic time - instead of doing a full table scan.

Adding an index will make read queries much faster, going from linear time complexity (O(n)) to logarithmic time (O(log2(n))). For a table containing 1 Mo. requests, an indexed table will find a record in an average time of 19 lookups v.s. a full table scan, which will take 1 Mo.

Keep in mind, adding an index will make ‘write’ and ‘update’ operations to the table less performant (because the B-tree has to also be updated).

🔍 Q.2) Which of the following will prevent a **Distributed Denial-of-Service** (DDoS) attack?

a) Using **parametrized queries** at database level and **validating/sanitizing user input** on the server side

b) Using **HTTP(TLS)** instead of **HTTP(TCP)**

c) **Employing load balancers and traffic filtering**

d) Implementing **rate limiting**

e) Using **JWTs** for authentication

Explain.

💡**Answer:** The most effective way to prevent a **DDoS attack** is traffic filtering and load balancing as it will block **known malicious IP addresses**, and keep the system stable in any case of a traffic spike.

Rate limiting will help control the traffic received from a specific client, but is not effective when the attackers are distributed (rate limiting can help block scrapers if they don't rotate IP).

🔍 **Q.3)** In the context of **Open-Closed Principle,** which of the following best explains what closed means?

a) A class cannot be executed **without modification.**

b) A class can be understood without needing to video it's source code.

c) A class should be **closed** for modification but **open** for extension.

d) A class cannot be understood without viewing its source code.

Explain.

💡**Answer: Open-Closed Principle** is a **SOLID** principle - it states that classes/modules should be closed for modification, but open for extension.

In plain words, you would design your components in a way in which you can modify the behavior without fundamentally changing the component.

To bring this to the **React world, a dropdown Component** is open for extension, but closed for modification if you can pass the **onSelect Callback** as a prop. You can modify how the dropdown will behave without touching its internal code.

🔍 **Q.4)** Which deployment has **zero downtime?**

a) **Blue/Green** Deployment

b) **Rolling** Deployment

c) **Canary** Deployment

d) **In-place** deployment

Explain.

💡 **Answer:** Option **a) Blue/Green Deployment** has zero downtime because it creates a new environment from scratch, and will only switch traffic once the new environment is healthy. The old environment will be kept for a defined period of time in case a fast rollback is needed.

🔍 **Q.5)** To prevent a **Man (person)-In-The-Middle Attack,** we have to?

- **a)** Advise the client to use **strict browser security policies**
- **b)** Use **HTTP authentication** for all requests
- **c)** Use **TLS** to encrypt the communication between two parties
- **d)** Encrypt our cookies before making a request to the server

Explain.

💡 **Answer:** Option **c) Use TLS (HTTPS)** to encrypt traffic between the client and the server end-to-end.

🔍 **Q.6)** Which of the following qualify as **reverse proxy:**

- **a)** Load Balancers
- **b)** CDNs
- **c)** VPN
- **d)** API Gateway
- **e)** Client-Side Caches

Explain.

💡 **Answer:** Option **a) Load Balancers**, **b) CDNs and d) API Gateways,** are all reverse proxies because they proxy traffic on behalf of the server (a normal proxy acts on behalf of a client - for example a VPN).

🔍 **Q.7)** Which of the following is a disadvantage of implementing a **Round Robin** algorithm in a **Load Balancer:**

- **a)** **Complex to implement** and to maintain - hard to change and harder to debug
- **b)** Assumes all the servers are equally performant and all requests take the same resources, leading to **unfair load distribution**

c) **Not performant -** it requires decoding the **HTTP** requests before forwarding them to the next instance

d) **Not scalable** - it requires a dedicated instance for each incoming requests (**not** horizontally scalable)

Explain.

💡**Answer: Option b) Unfair load distribution - Round Robin** can lead to **unfair traffic distribution**, because it distributes the traffic in a circular manner, and it is not aware of the current load of each target instance.

Better algorithms include **Least Connections or APM Metrics** (Memory, CPU).

🔍 **Q.8)** The so-called **Microservices Tax** refers to…

a) The effort invested in migrating from a **Monolith** to **Microservices**

b) The additional complexity of implementing a **Microservices Architecture**

c) The extra time required to deploy **Microservices** individually vs a single **Monolith** deployment

d) The additional cloud costs (more servers) of running a **Microservices Architecture**

Explain.

💡**Answer: Option b) The additional complexity** of implementing a **Microservices Architecture** - including Authentication between Services, Security and Networking.

🔍 **Q.9)** Which statements about **garbage collection** and **memory leaks** are true? (Two answers apply)

a) **Garbage collection** automatically frees all unused memory.

b) **Memory leaks** can still occur even in **garbage collected languages.**

c) Circular references can prevent **garbage collectors** from reclaiming memory.

d) **Garbage collectors** operate in real-time, immediate reclaiming memory when it’s no longer referenced.

💡 **Answer: Option b) and Option c) -** Memory leaks can still happen in garbage collected languages (see closures in JavaScript) - and circular references can prevent garbage collectors from reclaiming memory (also closures).

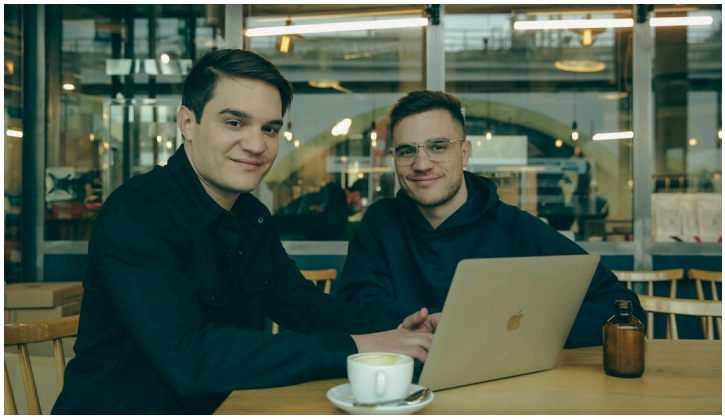
— END —

If you are a JavaScript Developer, we will get you to Senior, or you don't pay!

-> See how we get you to Senior here!

Want to find your Technical Gaps to Senior? Take the Free 10-Minute Technical Assessment to find your gaps to Senior level:

-> Take the Free Technical Assessment!



Best of success ✌️
Dragos & Bogdan