### \*\*React JS Interview Questions\*\*

#### \*\*General React Concepts\*\*

1. What is the virtual DOM?

2. What are hooks? How many hooks have you worked with, and explain them.

3. Explain `useEffect` in detail. How does it behave based on the dependency array?

4. What is `useContext`?

5. Have you created any custom hooks? If yes, for what purpose?

6. How do you optimize or improve performance in a React application?

7. What is HOC (Higher-Order Component)?

8. What is the difference between props and state in React?

9. What is prop drilling, and how can it be avoided?

10. What are React hooks, and how do they improve functional components?

11. Have you created any custom React hooks?

12. What is reconciliation in React?

13. How would you call an API without showing it in the browser's Network tab?

14. What is the difference between `let`, `const`, and `var` in JavaScript?

15. How many API calls have you handled on the frontend for data fetching?

#### \*\*State & Data Management\*\*

1. How do you handle add and remove product from cart?

2. How do you store data from multiple components to cart without using local storage?

3. How do you handle authentication flow in your project?

4. Where do you store your JWT token — cookie or `localStorage`? Why?

5. How do you handle access token vs refresh token?

6. What are HttpOnly cookies and how are they used?

7. How do components check event detection? Explain working also.

8. What is `NgOnChanges`? How can you check \*isSubmit\* Previous-value and Current-value on `NgOnChanges` Hook?

9. How do you implement conditional validation in Angular? \*(Note: Likely meant for React context)\*

10. What is `patchValue()` and `setValue()` in Angular forms, and what is the difference? \*(Note: Angular-specific, but mentioned in React context)\*

#### \*\*Performance & Optimization\*\*

1. How do you optimize the performance of your React application?

2. How do you implement lazy loading?

3. What is the use of `useCallback`?

#### \*\*UI & Styling\*\*

1. Are you able to convert a Figma design into a Tailwind-based UI?

2. Have you worked with Tailwind CSS?

#### \*\*Testing & Tools\*\*

1. Have you written unit test cases?

2. What git commands do you frequently use during development?

---

### \*\*Next.js Interview Questions\*\*

#### \*\*Core Concepts\*\*

1. What is the difference between Next.js and React.js?

2. Which version of Next.js are you using? What are its features?

3. How is Next.js better than React?

4. What are the main features of Next.js?

5. What is `getStaticProps`?

6. What is `getServerSideProps`?

7. What is SSR (Server Side Rendering)?

8. What is SSG (Static Site Generation)?

9. How have you used SSR and SSG in your projects?

10. What is the purpose of the `/api` folder in a Next.js application?

#### \*\*Deployment & Integration\*\*

1. What is the command to create a new Next.js project?

2. How much time would you estimate to build a page with a header, footer, sidebar, and content in Next.js?

#### \*\*Security & API Handling\*\*

1. How can you make an API call without it appearing in the browser's Network tab?

#### \*\*Project Implementation\*\*

1. Implement the solution using TypeScript and Next.js.

2. Utilize routes instead of a single-page application (SPA), with the working demonstrated on the index route.

3. Create separate routes for Create, Read, Update, and Delete operations.

4. Reuse input components across the application.

5. Make the Read component a server component.

---

### \*\*React + Next.js Combined Questions\*\*

1. What is the key difference between Next.js and React.js? As a developer, which one do you prefer—Next.js or React.js—and why?

2. Task: Retrieve data from JSON Placeholder and store it in local storage, then perform CRUD operations.

- Use \*\*Zod\*\* for data validation.

- Implement the solution using \*\*TypeScript and Next.js\*\*.

- Use \*\*routes\*\* instead of SPA.

- Create separate routes for CRUD.

- Reuse input components.

- Make the Read component a \*\*server component\*\*.

- Implement validation for valid email and 10-digit contact number.

- Use React forms for getting input values.

3. Have you developed a page by converting a Figma design into a Next.js layout?

---

### \*\*Angular Interview Questions\*\*

\*(Although not React/Next.js, included as a major frontend framework covered in the file)\*

#### \*\*Core Concepts\*\*

1. What is Angular?

2. What is a component?

3. What is a module?

4. What is a service?

5. What is a decorator?

6. What is a pipe? What is the difference between pure and impure pipes?

7. What is two-way data binding?

8. What is the lifecycle of an Angular component?

9. What is the difference between Template-driven and Reactive forms? Which is better?

10. What is NGRX?

11. What is the use of RxJS?

12. What are directives? How many types of directives?

13. What is the use of interceptors?

14. What is the use of AuthGuards?

15. What is the difference between eager loading and lazy loading?

16. What is the use of `APP\_INITIALIZER` and how will you set it up in your project?

17. What is view encapsulation?

18. Explain Angular's Change Detection Mechanism.

#### \*\*Forms & Validation\*\*

1. How do you implement conditional validation in Angular?

2. What is `patchValue()` and `setValue()` in Angular forms, and what is the difference?

#### \*\*Project Implementation\*\*

1. How to implement cart functionality like Zepto or Zomato?

2. How to handle add and remove product from cart?

3. How do you call multiple API calls? (e.g., 100 API calls)

4. Create two modules:

- Shared Module: Header and Footer components (without lazy loading)

- Core Module: Login (Template-Driven) and Register (Reactive Form) with lazy loading

5. Implement lazy loading for the core module.

#### \*\*JavaScript & TypeScript\*\*

1. What is hoisting?

2. What are closures?

3. What is the difference between `display: none` and `visibility: hidden`?

4. What is the disadvantage of `async` and `await`?

5. What is the use of `strict` mode?

6. What is a higher-order function?

7. What is the event loop?

8. What is the difference between synchronous and asynchronous?

9. What is the difference between `undefined` and `not defined`?

10. What is deep copy and shallow copy?

#### \*\*Debugging & Output\*\*

1. Write code to reverse a string.

2. Console.log questions:

- `console.log('0' === false)`

- `console.log(typeof 0 + undefined)`

- `console.log('1' !== true)`

- `console.log(typeof null + undefined)`

- `1 + '1'`, `1 - '1'`, `1 == '1'`, `1 === '1'`, etc.

---

### \*\*JavaScript / Frontend General Questions\*\*

#### \*\*Core JavaScript\*\*

1. Difference between `let`, `var`, and `const`?

2. What is hoisting?

3. What is a closure?

4. What are promises?

5. What is `async/await`?

6. What is the event loop and task queue?

7. What is function currying?

8. Explain lexical scope.

9. What is the difference between `call`, `apply`, and `bind`?

10. What is the prototype in JavaScript?

11. What is the difference between `==` and `===`?

12. What are first-class functions?

13. What is IIFE?

14. What is the difference between `fetch` and `axios`?

15. What is the difference between `slice`, `splice`, and `split`?

16. What is the difference between `block` and `non-blocking` code?

#### \*\*DOM & Events\*\*

1. What is event bubbling and event capturing?

2. What is the DOM?

3. What is the difference between `preventDefault()` and `stopPropagation()`?

#### \*\*Git & Tools\*\*

1. What git commands do you frequently use during development?

2. What is the command to create a new Next.js project?

3. What does `git merge` and `git pull` do, and when do we use them?

---

### \*\*Summary of Frontend Topics Covered\*\*

- \*\*React JS\*\*: Hooks, State, Performance, Context, HOC, Custom Hooks

- \*\*Next.js\*\*: SSR, SSG, API Routes, `getStaticProps`, `getServerSideProps`

- \*\*React + Next.js\*\*: Full-stack implementation with TypeScript, CRUD, Zod validation, routing

- \*\*Angular\*\*: Components, Modules, Forms, Lifecycle, NGRX, RxJS, Lazy Loading

- \*\*JavaScript\*\*: Closures, Hoisting, Event Loop, Promises, Async/Await, ES6+

- \*\*Styling & Tools\*\*: Tailwind CSS, Figma to Code, Git, Zod

Let me know if you'd like these organized by company or difficulty level.

Absolutely! Below is a \*\*comprehensive, well-structured, and beginner-friendly\*\* explanation of \*\*all the frontend-related questions\*\* extracted from your file, categorized under \*\*React JS\*\*, \*\*Next.js\*\*, and \*\*React + Next.js\*\*, with \*\*simple explanations\*\*, \*\*working examples\*\*, and \*\*code snippets\*\*.

---

# ✅ \*\*Frontend Interview Questions (React JS, Next.js, React + Next.js)\*\*

---

## 🟡 \*\*1. React JS Interview Questions\*\*

### \*\*Q1. What is the Virtual DOM?\*\*

\*\*Explanation\*\*:

The \*\*Virtual DOM (VDOM)\*\* is a lightweight copy of the real DOM. React uses it to track changes and only update the real DOM where necessary, improving performance.

\*\*Why it's used\*\*:

Direct DOM manipulation is slow. React compares the Virtual DOM before and after a change (diffing), then updates only the changed parts (reconciliation).

\*\*Example\*\*:

```jsx

// Without Virtual DOM (Slow)

document.getElementById("myDiv").innerHTML = "New Content";

// With React (Fast - uses Virtual DOM)

function App() {

const [text, setText] = useState("Old Content");

return <div onClick={() => setText("New Content")}>{text}</div>;

}

```

---

### \*\*Q2. What are Hooks? List and explain commonly used hooks.\*\*

\*\*Explanation\*\*:

Hooks are functions that let you use \*\*state and lifecycle features\*\* in functional components.

#### Common Hooks:

| Hook | Purpose |

|------|--------|

| `useState` | Manage state |

| `useEffect` | Handle side effects (API calls, subscriptions) |

| `useContext` | Access context without prop drilling |

| `useRef` | Reference DOM elements or persist values |

| `useReducer` | Manage complex state logic |

| `useCallback` | Memoize functions |

| `useMemo` | Memoize expensive calculations |

\*\*Example\*\*:

```jsx

import { useState, useEffect } from 'react';

function Counter() {

const [count, setCount] = useState(0);

useEffect(() => {

document.title = `Count: ${count}`;

}, [count]);

return <button onClick={() => setCount(count + 1)}>Count: {count}</button>;

}

```

---

### \*\*Q3. Explain `useEffect` in detail. How does it behave based on the dependency array?\*\*

\*\*Explanation\*\*:

`useEffect` runs after render. Behavior depends on the \*\*dependency array\*\*:

| Dependency | Behavior |

|----------|---------|

| `[]` | Runs once (like `componentDidMount`) |

| `[dep]` | Runs when `dep` changes |

| No array | Runs after every render |

| Cleanup | Return a function to clean up (e.g., unsubscribe) |

\*\*Example\*\*:

```jsx

useEffect(() => {

console.log("Runs every render");

});

useEffect(() => {

console.log("Runs once");

}, []);

useEffect(() => {

console.log("Runs when name changes");

}, [name]);

useEffect(() => {

const timer = setInterval(() => console.log("Tick"), 1000);

return () => clearInterval(timer); // Cleanup

}, []);

```

---

### \*\*Q4. What is `useContext`?\*\*

\*\*Explanation\*\*:

`useContext` allows components to access data from a \*\*Context\*\* without passing props through every level.

\*\*Use Case\*\*: Theme, user auth, language.

\*\*Example\*\*:

```jsx

const ThemeContext = createContext();

function App() {

return (

<ThemeContext.Provider value="dark">

<Toolbar />

</ThemeContext.Provider>

);

}

function Toolbar() {

const theme = useContext(ThemeContext);

return <div className={theme}>Toolbar</div>;

}

```

---

### \*\*Q5. Have you created any custom hooks? If yes, for what purpose?\*\*

\*\*Yes!\*\* Custom hooks reuse logic across components.

\*\*Example: `useFetch`\*\*

```jsx

function useFetch(url) {

const [data, setData] = useState(null);

const [loading, setLoading] = useState(true);

useEffect(() => {

fetch(url)

.then(res => res.json())

.then(data => {

setData(data);

setLoading(false);

});

}, [url]);

return { data, loading };

}

// Usage

function UserList() {

const { data, loading } = useFetch('/api/users');

if (loading) return <p>Loading...</p>;

return <ul>{data.map(u => <li key={u.id}>{u.name}</li>)}</ul>;

}

```

---

### \*\*Q6. How do you optimize or improve performance in a React application?\*\*

\*\*Techniques\*\*:

1. `React.memo()` – Prevent re-renders of pure components

2. `useCallback` – Memoize functions

3. `useMemo` – Memoize expensive calculations

4. Code splitting with `React.lazy()`

5. Avoid inline objects/functions in props

\*\*Example\*\*:

```jsx

const ExpensiveComponent = React.memo(({ user }) => {

return <div>{user.name}</div>;

});

```

---

### \*\*Q7. What is HOC (Higher-Order Component)?\*\*

\*\*Explanation\*\*:

A HOC is a function that takes a component and returns a new enhanced component.

\*\*Use Case\*\*: Authentication, logging, loading.

\*\*Example\*\*:

```jsx

function withAuth(WrappedComponent) {

return function (props) {

const isLoggedIn = localStorage.getItem('token');

return isLoggedIn ? <WrappedComponent {...props} /> : <p>Please login</p>;

};

}

const ProtectedProfile = withAuth(Profile);

```

---

### \*\*Q8. What is the difference between props and state in React?\*\*

| Props | State |

|------|-------|

| Passed from parent | Managed within component |

| Read-only | Mutable (`useState`) |

| Used for configuration | Used for dynamic data |

\*\*Example\*\*:

```jsx

function Welcome(props) {

// props.name → read-only

return <h1>Hello {props.name}</h1>;

}

function Counter() {

const [count, setCount] = useState(0); // state → mutable

return <button onClick={() => setCount(count + 1)}>{count}</button>;

}

```

---

### \*\*Q9. What is prop drilling, and how can it be avoided?\*\*

\*\*Explanation\*\*:

Passing props through many intermediate components.

\*\*Solution\*\*: Use \*\*Context API\*\* or \*\*Redux\*\*.

\*\*Example with Context\*\*:

```jsx

// Instead of: Parent → Middle → Child (passing user)

// Use Context to directly access user in Child

const UserContext = createContext();

function Child() {

const user = useContext(UserContext);

return <p>Welcome {user.name}</p>;

}

```

---

### \*\*Q10. How do you handle add and remove product from cart?\*\*

\*\*Example\*\*:

```jsx

function Cart() {

const [cart, setCart] = useState([]);

const addToCart = (product) => {

setCart([...cart, product]);

};

const removeFromCart = (id) => {

setCart(cart.filter(item => item.id !== id));

};

return (

<>

{cart.map(item => (

<div key={item.id}>

{item.name} <button onClick={() => removeFromCart(item.id)}>Remove</button>

</div>

))}

</>

);

}

```

---

### \*\*Q11. Where do you store JWT token — cookie or `localStorage`? Why?\*\*

| localStorage | Cookies |

|-------------|--------|

| Easy to access | Can be HttpOnly |

| Vulnerable to XSS | Secure against XSS if HttpOnly |

| No CSRF risk | CSRF risk if not secured |

✅ \*\*Best Practice\*\*: Store in \*\*HttpOnly Cookie\*\* to prevent XSS.

---

### \*\*Q12. What are HttpOnly cookies and how are they used?\*\*

\*\*Explanation\*\*:

HttpOnly cookies cannot be accessed via JavaScript. They are sent automatically with HTTP requests.

\*\*Use Case\*\*: Secure JWT storage.

\*\*Backend (Node.js)\*\*:

```js

res.cookie('token', jwtToken, { httpOnly: true, secure: true });

```

Frontend cannot read it → safer from XSS.

---

### \*\*Q13. How do you implement conditional validation in Angular?\*\*

> ⚠️ This is \*\*Angular\*\*, not React. But if you meant React:

\*\*React Example (using `react-hook-form`)\*\*:

```jsx

const { register, watch } = useForm();

const isStudent = watch('isStudent');

<TextField

{...register('grade', { required: isStudent })}

label="Grade"

/>

```

---

### \*\*Q14. Are you able to convert a Figma design into a Tailwind-based UI?\*\*

✅ \*\*Yes\*\*. Use \*\*Tailwind CSS\*\* to style components based on Figma specs.

\*\*Tools\*\*:

- Figma → Export design

- Use `@apply` or utility classes

- Match fonts, spacing, colors

\*\*Example\*\*:

```jsx

<div className="bg-gray-100 p-6 rounded-lg shadow-md">

<h1 className="text-2xl font-bold text-blue-600">Welcome</h1>

</div>

```

---

### \*\*Q15. Have you written unit test cases?\*\*

✅ Yes. Use \*\*Jest + React Testing Library\*\*.

\*\*Example\*\*:

```jsx

test('renders counter', () => {

render(<Counter />);

const button = screen.getByText(/Count: 0/i);

expect(button).toBeInTheDocument();

});

```

---

## 🟠 \*\*2. Next.js Interview Questions\*\*

### \*\*Q1. What is the difference between Next.js and React.js?\*\*

| React | Next.js |

|------|--------|

| Client-side only | SSR, SSG, ISR |

| No routing by default | File-based routing |

| Manual setup | Built-in optimization |

| SPA | Full-stack capable |

✅ \*\*Next.js = React + SSR + Routing + API Routes + Optimization\*\*

---

### \*\*Q2. What is `getStaticProps`?\*\*

\*\*Explanation\*\*:

Fetch data at \*\*build time\*\*. Used for \*\*Static Site Generation (SSG)\*\*.

\*\*Use Case\*\*: Blogs, product listings.

\*\*Example\*\*:

```js

export async function getStaticProps() {

const res = await fetch('https://jsonplaceholder.typicode.com/posts');

const posts = await res.json();

return { props: { posts }, revalidate: 60 }; // ISR

}

```

---

### \*\*Q3. What is `getServerSideProps`?\*\*

\*\*Explanation\*\*:

Fetch data on \*\*each request\*\*. Used for \*\*Server-Side Rendering (SSR)\*\*.

\*\*Use Case\*\*: User-specific data.

\*\*Example\*\*:

```js

export async function getServerSideProps(context) {

const { req } = context;

const token = req.cookies.token;

const user = await getUser(token);

return { props: { user } };

}

```

---

### \*\*Q4. What is SSR and SSG?\*\*

| SSR (Server-Side Rendering) | SSG (Static Site Generation) |

|----------------------------|-----------------------------|

| HTML generated on each request | HTML generated at build time |

| Dynamic, personalized | Fast, cached |

| Slower | Faster |

✅ Use \*\*SSG\*\* for public content, \*\*SSR\*\* for private/user-specific.

---

### \*\*Q5. What is the purpose of the `/api` folder in Next.js?\*\*

It contains \*\*API routes\*\*. Each file becomes an API endpoint.

\*\*Example\*\*: `pages/api/hello.js`

```js

export default function handler(req, res) {

res.status(200).json({ message: 'Hello' });

}

```

→ Access at `/api/hello`

---

### \*\*Q6. How can you make an API call without it appearing in the browser's Network tab?\*\*

✅ Call it from \*\*`getServerSideProps`\*\*, \*\*`getStaticProps`\*\*, or \*\*API Route\*\*.

Because:

- These run on \*\*server\*\*, not browser

- No network trace

\*\*Example\*\*:

```js

// pages/index.js

export async function getServerSideProps() {

const res = await fetch('https://internal-api.com/data'); // Hidden

const data = await res.json();

return { props: { data } };

}

```

---

## 🟢 \*\*3. React + Next.js Combined Questions\*\*

### \*\*Task: CRUD App with Next.js, TypeScript, Zod, and Routes\*\*

#### ✅ Requirements:

- Use \*\*TypeScript\*\*

- Use \*\*Zod\*\* for validation

- Separate \*\*Create, Read, Update, Delete\*\* routes

- Reuse input components

- Make \*\*Read\*\* a \*\*Server Component\*\*

- Validate email and 10-digit phone

---

### \*\*Step 1: Project Setup\*\*

```bash

npx create-next-app@latest my-app --typescript

cd my-app

npm install zod react-hook-form

```

---

### \*\*Step 2: Define Schema (Zod)\*\*

```ts

// lib/schema.ts

import { z } from 'zod';

export const userSchema = z.object({

name: z.string().min(2),

email: z.string().email(),

phone: z.string().regex(/^\d{10}$/),

});

```

---

### \*\*Step 3: API Route (pages/api/users.ts)\*\*

```ts

// pages/api/users.ts

let users = [];

export default function handler(req, res) {

if (req.method === 'GET') {

res.status(200).json(users);

} else if (req.method === 'POST') {

const newUser = req.body;

users.push(newUser);

res.status(201).json(newUser);

}

}

```

---

### \*\*Step 4: Reusable Input Component\*\*

```tsx

// components/Input.tsx

export default function Input({ label, ...props }) {

return (

<div>

<label>{label}</label>

<input {...props} className="border p-2" />

</div>

);

}

```

---

### \*\*Step 5: Read Page (Server Component)\*\*

```tsx

// app/read/page.tsx

import { getUserData } from '@/lib/data'; // Server-only fetch

export default async function ReadPage() {

const users = await getUserData(); // Runs on server

return (

<ul>

{users.map(u => <li key={u.email}>{u.name}</li>)}

</ul>

);

}

```

---

### \*\*Step 6: Create Page (Client Component)\*\*

```tsx

// app/create/page.tsx

'use client';

import { useState } from 'react';

import Input from '@/components/Input';

export default function CreatePage() {

const [form, setForm] = useState({ name: '', email: '', phone: '' });

const handleSubmit = async (e) => {

e.preventDefault();

await fetch('/api/users', {

method: 'POST',

body: JSON.stringify(form),

});

alert('User created!');

};

return (

<form onSubmit={handleSubmit}>

<Input label="Name" value={form.name} onChange={e => setForm({...form, name: e.target.value})} />

<Input label="Email" value={form.email} onChange={e => setForm({...form, email: e.target.value})} />

<Input label="Phone" value={form.phone} onChange={e => setForm({...form, phone: e.target.value})} />

<button type="submit">Create</button>

</form>

);

}

```

---

### ✅ Final Notes:

- \*\*Validation\*\*: Use Zod in API route

- \*\*Routing\*\*: `/create`, `/read`, `/update`, `/delete`

- \*\*Security\*\*: Store JWT in HttpOnly cookie

- \*\*Performance\*\*: Use SSG/SSR where possible

---

## 🔚 Summary

| Topic | Key Answer |

|------|-----------|

| \*\*Virtual DOM\*\* | Fast updates via diffing |

| \*\*useEffect\*\* | Side effects with dependency control |

| \*\*JWT Storage\*\* | HttpOnly Cookie (secure) |

| \*\*SSR vs SSG\*\* | SSR: per request, SSG: build time |

| \*\*API Security\*\* | Hide calls in `getServerSideProps` |

| \*\*Custom Hooks\*\* | Reuse logic (e.g., `useFetch`) |

| \*\*CRUD in Next.js\*\* | Use API routes + TypeScript + Zod |

---