## Full Stack Developer Interview Questions & Answers

### 🔹 Frontend (React, Next.js, JavaScript, TypeScript, Redux)

#### ✅ Beginner-Level

**1. What are the main differences between React and Next.js?**

* **Answer:**
  + React is a frontend JavaScript library for building UIs.
  + Next.js is a framework built on top of React that provides SSR (Server-Side Rendering), static site generation (SSG), file-based routing, and API routes.
* **Example:**
  + Use Next.js when you need SEO optimization (e.g., blog, e-commerce site).
  + Use plain React for SPAs with client-side routing (e.g., dashboards).

**2. What is JSX? How does it work under the hood?**

* **Answer:** JSX is a syntax extension that allows writing HTML-like code in JavaScript. It’s transpiled to React.createElement() calls by Babel.
* **Example:**
* <h1>Hello</h1>  
  // becomes  
  React.createElement('h1', null, 'Hello');

**3. Explain the virtual DOM and how React uses it.**

* **Answer:** Virtual DOM is an in-memory representation of the actual DOM. React uses it to batch updates and reduce direct DOM manipulation.
* **Example:** Updating a counter triggers diffing algorithm and only changes affected DOM node.

**4. How does** ``\*\* work in React?\*\*

* **Answer:** useState is a React Hook that declares state in functional components.
* **Example:**
* const [count, setCount] = useState(0);  
  setCount(count + 1);

**5. What is the difference between** \*\* and \*\***?**

* **Answer:**
  + Props are read-only and passed from parent to child.
  + State is mutable and managed within a component.

**6. What is SSR in Next.js?**

* **Answer:** Server-Side Rendering means rendering HTML on the server for each request, improving performance and SEO.
* **Example:**
* export async function getServerSideProps() {  
   return { props: { data: await fetchData() } }  
  }

**7. How do you handle routing in Next.js?**

* **Answer:** File-based routing. Pages are created in the /pages directory.
* **Example:** /pages/about.js maps to /about route.

**8. How is TypeScript different from JavaScript? Why use it?**

* **Answer:** TypeScript is a statically typed superset of JavaScript. It provides type checking at compile time.
* **Example:**
* function greet(name: string): string {  
   return `Hello, ${name}`;  
  }

#### ✅ JavaScript & TypeScript - Additional Questions

**9. What is hoisting in JavaScript?**

* **Answer:** JavaScript hoists declarations (not initializations) of variables and functions to the top of their scope.
* **Example:**
* console.log(a); // undefined  
  var a = 10;

**10. What are closures in JavaScript?**

* **Answer:** A closure is a function that retains access to its lexical scope even when the function is executed outside that scope.
* **Example:**
* function outer() {  
   let count = 0;  
   return function inner() {  
   count++;  
   return count;  
   }  
  }  
  const counter = outer();  
  console.log(counter()); // 1  
  console.log(counter()); // 2

**11. What is the difference between** \*\*, \*\***, and** ``**?**

* **Answer:**
  + var is function-scoped and hoisted.
  + let and const are block-scoped. const cannot be reassigned.

**12. What are interfaces and types in TypeScript?**

* **Answer:**
  + interface defines a contract for object structure.
  + type can define unions, primitives, and other complex types.
* **Example:**
* interface User {  
   name: string;  
   age: number;  
  }  
    
  type Status = 'active' | 'inactive';

**13. How does TypeScript help in reducing bugs?**

* **Answer:** By enforcing static typing, TypeScript helps catch errors during development, improving reliability and maintainability.
* **Example:**
* function add(a: number, b: number): number {  
   return a + b;  
  }  
  // add("1", 2); // Error: Argument of type 'string' is not assignable to parameter of type 'number'.

**14. What is type inference in TypeScript?**

* **Answer:** TypeScript automatically infers the type based on the assigned value.
* **Example:**
* let message = "Hello"; // inferred as string

**15. Explain enums in TypeScript.**

* **Answer:** Enums define named constants for better readability and intent.
* **Example:**
* enum Direction {  
   Up,  
   Down,  
   Left,  
   Right  
  }  
  const move = Direction.Up;

### 🔹 Backend (Node.js, Express.js, Sequelize)

#### ✅ Beginner-Level

**1. Difference between** \*\* and \*\*\*\* in Node.js?\*\*

* **Answer:**
  + require is CommonJS (default in Node).
  + import is ES Module syntax (requires Node >=14 with “type”: “module” or .mjs file).

**2. What are middlewares in Express.js?**

* **Answer:** Functions that execute during the request-response cycle.
* **Example:**
* app.use(express.json());  
  app.use((req, res, next) => { console.log(req.method); next(); });

**3. How do you define routes in Express?**

* **Example:**
* app.get('/api/users', (req, res) => res.send(users));

**4. How do you handle async code in Node.js?**

* **Answer:** Using Promises and async/await.
* const data = await fetchData();

**5. What is Sequelize? Why use it?**

* **Answer:** ORM for SQL DBs. Converts models into DB tables and provides query abstraction.
* **Example:**
* const User = sequelize.define('User', { name: Sequelize.STRING });

### 🔹 Database (Oracle, SQL Server, MongoDB)

#### ✅ General

**1. Difference between SQL and NoSQL?**

* SQL: Structured, table-based (Oracle, SQL Server)
* NoSQL: Document/key-value (MongoDB)

**2. How to model a social media post in SQL vs MongoDB?**

* **SQL:** Tables for Posts, Comments, Likes with foreign keys.
* **MongoDB:** Embed comments inside post document.

**3. Write a stored procedure in SQL Server:**

CREATE PROCEDURE GetUsers  
AS  
BEGIN  
 SELECT \* FROM Users;  
END

**4. MongoDB aggregation example:**

Post.aggregate([  
 { $match: { status: "published" } },  
 { $group: { \_id: "$author", count: { $sum: 1 } } }  
]);

### 🔹 Others (Redux, REST API)

**1. What is the Redux flow?**

* Action → Reducer → Store → UI
* **Example:**
* dispatch({ type: 'INCREMENT' });

**2. How does Redux Toolkit improve Redux?**

* Less boilerplate
* Built-in createSlice, createAsyncThunk

**3. RESTful API methods:**

* GET: Read data
* POST: Create
* PUT: Replace
* PATCH: Update partial
* DELETE: Remove

**4. Secure a REST API using JWT:**

* Issue token at login, verify on protected routes
* const token = jwt.sign(user, secret);

Would you like to continue with:

* Mid-Level and Advanced questions?
* Real-world project examples?
* Coding challenges?
* PDF/Excel export of this content?