Full Stack Developer Interview Questions & Answers

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

✅Beginner-Level

... (existing content remains unchanged) ...

✅Mid-Level and Advanced JavaScript & TypeScript Questions

16. What is the difference between `** and **` in JavaScript?

- **Answer:**
- `==` performs type coercion before comparison.
- `===` checks both value and type strictly.
- **Example:**

```
'5' == 5     // true
'5' === 5    // false
```

17. Explain event delegation in JavaScript.

- **Answer:**
- Event delegation uses a single event listener on a parent element to manage events from its children using event bubbling.
- **Example:**

```
document.getElementById('list').addEventListener('click', function (e) {
  if (e.target.tagName === 'LI') {
    console.log('Item clicked:', e.target.textContent);
  }
});
```

18. What are generics in TypeScript?

- **Answer:**
- Generics allow you to create reusable components that work with any data type.
- **Example:**

```
function identity<T>(arg: T): T {
  return arg;
}
const num = identity<number>(10);
```

19. What is the difference between `** and **` in TypeScript?

- **Answer:**
- `interface` can be extended and merged.
- `type` is more flexible with unions and intersections.
- **Use Case Example:**

```
type A = { a: string };
type B = A & { b: number }; // intersection

interface A2 { a: string; }
interface B2 extends A2 { b: number; }
```

20. What is the `` type in TypeScript?

- **Answer:**
- It represents values that never occur, often used in exhaustive checks.
- **Example:**

```
function fail(msg: string): never {
  throw new Error(msg);
}
```

21. Explain destructuring in JavaScript.

- **Answer:**
- It allows unpacking values from arrays or objects into variables.
- **Example:**

```
const [a, b] = [1, 2];
const {name, age} = {name: 'John', age: 25};
```

22. What are optional chaining and nullish coalescing in JavaScript?

- **Answer:**
- Optional chaining `?.` avoids errors when accessing nested properties.
- Nullish coalescing `??` returns the right-hand value only if the left is `null` or `undefined`.
- **Example:**

```
const user = { profile: null };
console.log(user.profile?.name); // undefined
const name = user.profile?.name ?? 'Guest';
```

23. What is the difference between `**, **`, and `` in JavaScript?

- **Answer:**
- `map()` transforms each element.
- `filter()` returns elements that satisfy a condition.
- `reduce()` accumulates a result from array elements.
- **Example:**

```
const arr = [1, 2, 3];
arr.map(x => x * 2); // [2, 4, 6]
arr.filter(x => x > 1); // [2, 3]
arr.reduce((a, b) => a + b, 0); // 6
```

24. Explain the concept of currying in JavaScript.

- **Answer:**
- Currying is the process of transforming a function with multiple arguments into a sequence of functions each taking a single argument.
- **Example:**

```
function add(a) {
  return function (b) {
    return a + b;
  };
}
const add5 = add(5);
console.log(add5(3)); // 8
```

25. What are utility types in TypeScript?

- **Answer:**
- Utility types like `Partial`, `Pick`, `Omit`, `Readonly` simplify type transformations.
- **Example:**

```
interface User {
  name: string;
  age: number;
}
const user: Partial<User> = { name: 'Alice' };
```

26. What are Type Guards in TypeScript?

- **Answer:**
- Type guards help TypeScript infer a more specific type within a conditional block.
- **Example:**

```ts
function isString(x: any): x is string {
  return typeof x === 'string';
}
function printLength(x: string | number) {
  if (isString(x)) {
    console.log(x.length);
  }
}
```

27. What is hoisting in JavaScript?

- **Answer:**
- Hoisting moves variable and function declarations to the top of their scope.
- **Example:**

```ts
console.log(a); // undefined
var a = 10;
```

28. What is a closure in JavaScript?

- **Answer:**
- A closure is a function that retains access to variables in its lexical scope even when called outside of that scope.
- **Example:**

```ts
function outer() {
  let count = 0;
  return function inner() {
    count++;
    return count;
  };
}
const counter = outer();
console.log(counter()); // 1
console.log(counter()); // 2
```

29. How does the `` keyword behave differently in arrow functions vs regular functions?

- **Answer:**
- Arrow functions do not have their own `this`; they inherit from their lexical context.
- Regular functions bind `this` to the caller.
- **Example:**

```
const obj = {
  name: 'Test',
  arrow: () => console.log(this.name),
  regular() { console.log(this.name); }
};
obj.arrow();   // undefined
obj.regular(); // 'Test'
```

30. What is type assertion in TypeScript?

- **Answer:**
- Type assertion tells the compiler to treat a value as a specific type.
- **Example:**

```
let someValue: any = "hello";
let strLength: number = (someValue as string).length;
```

(continued...)