Final MERN Internship Revision Questions with Answers

JavaScript

1. What is hoisting? - JavaScript moves **declarations** to the top of their scope during compilation. Only declarations are hoisted, not initializations.

2. What is the Temporal Dead Zone (TDZ)? - The period before a `let` or `const` variable is declared where accessing it causes a **ReferenceError**.

3. Explain the difference between `let` and `const`. - `let` is block-scoped and reassignable. - `const` is block-scoped and **not reassignable**.

4. Explain the difference between function scope, block scope, and global scope. - Function scope: variables accessible only inside the function. - Block scope: variables declared with `let` or `const` inside `{ }`. - Global scope: variables accessible everywhere.

5. What is `this` in JavaScript? How does it behave in normal vs arrow functions? - `this` refers to the execution context. - Normal function: has its own `this`. - Arrow function: inherits `this` from parent scope.

6. What is the difference between `==` and `===`? - `==` checks **value** only. - `===` checks **value and type**.

7. Explain closures with an example. - A closure is a function that remembers variables from its outer scope.

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

8. Explain the JavaScript event loop in simple terms. - Allows JS to handle async tasks without blocking the main thread by moving callbacks from the queue to the call stack.

9. What are callbacks, promises, and async/await? How do they differ? - Callback: function passed to another function to run later. - Promise: represents a future value, uses `.then()` and `.catch()`. - Async/Await: syntactic sugar over promises, uses `try/catch`.

React

10. What is React and why do we use it? - React is a JS library for building UI with **components**; allows reusable and efficient UI.

11. What is a component? Why do we use components? - A small reusable UI piece like a button or card. Helps in **reusability and maintainability**.

12. What is the difference between props and state? - Props: read-only, passed from parent. - State: mutable, managed inside the component.

13. What is the purpose of `useState`? - Hook to add **state to functional components**, allows re-render on update.

```
const [count, setCount] = useState(0);
setCount(count + 1);
```

14. What is the difference between `useState` and `useEffect`? - `useState`: manages state. - `useEffect`: performs side effects after render (like fetching data).

15. Explain controlled vs uncontrolled components. - Controlled: React manages the input value via state. - Uncontrolled: DOM manages the input value.

16. What is the purpose of the `key` prop in lists? - Provides **unique identity** to list elements for efficient re-rendering.

17. Difference between function components and class components. - Function: simple functions, use hooks. - Class: ES6 classes, use `this.state` and lifecycle methods.

18. How does conditional rendering work in React? - Use ternary operators or logical `&&` to render elements based on conditions.

19. How do you handle forms in React? - Manage inputs via `useState`, submit via event handlers, and optionally validate inputs.

Node.js & Express

20. What is Node.js and why do we use it? - JS runtime for server-side. Non-blocking, fast, same language for frontend/backend.

21. What is Express.js and why do we use it? - Node.js framework to simplify routing, middleware, and APIs.

22. How do you create routes in Express? - Use `express.Router()` and define `GET` / `POST` / `PUT` / `DELETE` routes.

23. What are controllers in Express? - Functions that handle request logic and interact with database.

24. How do you handle POST requests in Express? - Define `router.post('/route', controller)` and use `req.body` to access data.

25. How do you connect Express to MongoDB? - Use `mongoose.connect('mongodb://url/db')` and define models.

MongoDB

26. What is MongoDB and why use it? - NoSQL database, stores JSON-like documents, flexible schema, easy integration with JS.

27. What are collections and documents? - Collections: groups of documents. - Documents: individual records stored as JSON.

28. How do you perform CRUD operations in MongoDB? - Create: `Model.create()` - Read: `Model.find()` - Update: `Model.updateOne()` - Delete: `Model.deleteOne()`

29. What is the difference between SQL and NoSQL? - SQL: table-based, structured, schema required. - NoSQL: document-based, flexible schema.

MERN Stack Practical Flow

30. Explain the full MERN data flow (React → Express → MongoDB → React). - React sends request → Express route/controller → MongoDB fetch/save → Response back to React → Update UI.

31. How do you fetch data from backend and display it in React? - Use `axios` or `fetch`, store response in state, render with `map()`.

32. How do you submit form data from React to Express/MongoDB? - Capture inputs with `useState`, send POST request via `axios` / `fetch` to Express, save in MongoDB.

33. How do you update or delete data in MERN stack? - Update: `axios.put('/api/route', data)` → Express → MongoDB update. - Delete: `axios.delete('/api/route/id')` → Express → MongoDB delete.

34. How do you handle errors in MERN stack? - Backend: try/catch and proper HTTP status codes. - Frontend: catch promises or try/catch with async/await.

35. What is the difference between server-side and client-side rendering? - SSR: rendered on server, faster initial load. - CSR: rendered on client, requires JS to run.

Async / API & Advanced JS

36. How does fetch or axios work in React? - Makes HTTP requests to backend APIs asynchronously, returns a promise.

37. Explain promise chaining. - Using `.then()` consecutively to handle multiple async tasks in order.

38. How do you handle errors in async/await? - Use `try/catch` blocks around `await` calls.

39. What is the difference between synchronous and asynchronous code? - Synchronous: runs sequentially, blocking. - Asynchronous: runs tasks without blocking main thread.

40. How do you prevent callback hell? - Use **promises** or **async/await** instead of deeply nested callbacks.