Here are the concise answers to help you ace your Frontend (React) and Backend (Node.js) interviews:

**𝚐𝚛𝚛𝚝𝚊𝚎𝚕 𝚗𝚒𝚜𝚎𝚛 – 𝚒𝚑𝚜𝚑 (React)**

**✨ Core Concepts**

1. **Class vs Functional Components**: Class components use lifecycle methods; functional components use hooks. Functional are simpler, easier to test.
2. **State vs Props**: State is local and mutable; props are external and read-only.
3. **Virtual DOM**: A lightweight DOM copy that React updates efficiently before syncing with real DOM.
4. **Reconciliation**: Process of diffing old and new virtual DOM trees to apply minimal DOM updates.
5. **React Hook**: Functions like useState/useEffect in functional components to manage state/lifecycle.
6. **useEffect**: Runs side-effects. Example:

useEffect(() => { fetchData(); }, [deps]);

1. **Managing Side Effects**: useEffect, cleanup functions, custom hooks.
2. **useContext**: Allows sharing data (like theme, auth) across components without prop drilling.

**🤝 Advanced Topics**

1. **useReducer vs useState**: useReducer is preferred for complex state logic; useState for simple values.
2. **React Context**: Provides global state without passing props manually.
3. **SSR**: Renders React on the server. Benefits: SEO, performance.
4. **Suspense**: Lazy loading components or data. Improves loading strategies.
5. **React.memo**: Prevents unnecessary re-renders by memoizing components.
6. **Performance Optimization**: Code-splitting, memoization, useCallback, useMemo, SSR, lazy loading.

**🌐 React Ecosystem**

1. **Redux**: State container. Connects React via react-redux. Centralized state management.
2. **Controlled vs Uncontrolled**: Controlled has state managed by React. Uncontrolled uses refs.
3. **React Router**: Handles routing with , , , etc.
4. **TypeScript in React**: Adds static typing. Improves safety and dev experience. Setup via Create React App with TS template.

**✅ Testing and Debugging**

1. **Testing Tools**: Jest, React Testing Library, Enzyme.
2. **Testing Example**:

test('renders', () => {

render(<MyComponent />);

expect(screen.getByText('Hello')).toBeInTheDocument();

});

1. **Debugging Strategies**: React DevTools, console logs, breakpoints, Profiler.

**📃 Practical Application**

1. **Forms & Validation**: useState/useForm + libraries like Formik or React Hook Form.
2. **Error Boundaries**: Catch render errors. Implement using componentDidCatch() in class components.
3. **Keys in Lists**: Help React identify changed elements. Use unique, stable keys for performance.

**𝚐𝚢𝚒𝚑𝚠 – 𝚑𝚎𝚝𝚊𝚜 (Node.js)**

**✨ Core Concepts**

1. **Node.js**: JS runtime for server-side apps. Used in web servers, APIs, CLI tools.
2. **Why Node.js**: Fast, non-blocking I/O, JS on backend.
3. **How it works**: V8 engine + event loop + single-threaded async execution.
4. **Single-threaded Reason**: Easier concurrency model with event-driven async architecture.
5. **Concurrency Handling**: Event loop, async callbacks, promises, worker threads.
6. **Callback**: A function passed as an argument for async handling.
7. **Promises over Callbacks**: Cleaner, avoids callback hell, easier chaining.
8. **I/O**: Input/Output - e.g., file system, DB operations.
9. **Use Cases**: REST APIs, real-time apps, microservices.
10. **Frontend vs Backend**: Frontend = UI; Backend = logic, DB, APIs.

**⚖️ Node & Modules**

1. **NPM**: Node Package Manager - install/manage dependencies.
2. **Modules**: Reusable code units - built-in, custom, third-party.
3. **module.exports**: Exposes variables/functions for import.
4. **Why Node.js > PHP/Java**: Lightweight, fast, JS full stack.
5. **Angular vs Node.js**: Angular = frontend framework; Node = backend runtime.
6. **Popular DB with Node.js**: MongoDB.
7. **Popular Libraries**: Express.js, Mongoose, Lodash, JWT, Dotenv.
8. **Pros & Cons**:

* Pros: Fast, scalable, full-stack JS.
* Cons: CPU-intensive tasks not ideal.

**📖 Essential APIs & Features**

1. **Import Command**: require('module') or ES6 import.
2. **Event-Driven**: Code responds to emitted events.
3. **Event Loop**: Handles async tasks by queueing them.
4. **process.nextTick vs setImmediate**: nextTick runs before IO events; setImmediate runs after.
5. **EventEmitter**: Emits/handles custom events.
6. **API Function Types**: Asynchronous & synchronous.
7. **package.json**: Metadata file managing scripts, dependencies.
8. **URL Module**: Parses, formats URLs (require('url')).
9. **Express.js**: Minimalist web framework.
10. **Simple Express App**:

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send('Hello World'));

app.listen(3000);

1. **Streams**: Handle large data efficiently (e.g., file reading).
2. **Dependency Commands**: npm install/update/uninstall package-name
3. **Simple Node Server**:

require('http').createServer((req, res) => {

res.end('Hello World');

}).listen(3000);

1. **Async & Non-blocking**: Async APIs allow parallel tasks without blocking the thread.
2. **Async Implementation**: async/await, promises.
3. **Callback Function**: Handles async results.
4. **REPL**: Interactive Node shell.
5. **Control Flow Function**: Manages async flow (e.g., waterfall, parallel).
6. **Control Flow Management**: Ensures sequence of async calls.
7. **fork() vs spawn()**: fork = new Node process; spawn = any command.
8. **Buffer Class**: Handles binary data.
9. **Piping**: Connect streams for chained operations.
10. **File Read/Write Flags**: 'r', 'w', 'a', etc.
11. **Open File**: fs.open(path, flags, callback).
12. **Callback Hell**: Nested callbacks causing unreadable code.
13. **Reactor Pattern**: Manages I/O operations using event loop.
14. **Test Pyramid**: Emphasizes unit tests, less integration/UI tests.
15. **Why V8**: Google’s fast JS engine.
16. **Exit Codes**: 0 = success; others = errors.
17. **Middleware**: Functions in Express to handle requests/responses.
18. **HTTP Request Types**: GET, POST, PUT, DELETE, etc.
19. **MongoDB Connection**:

mongoose.connect(DB\_URI);

1. **NODE\_ENV**: Sets app environment (e.g., development/production).
2. **Timing Features**: setTimeout, setInterval, process.hrtime, etc.
3. **WASI**: WebAssembly System Interface for running WebAssembly outside browsers securely.

Let me know if you'd like flashcards, code snippets, or mock interview answers prepared from this!