**React Core Concepts**

**1. What are the main differences between React and Angular? What are the advantages and disadvantages of each?**

**React:**

* Library for building UI components; focuses on view layer.
* Uses JSX for templating.
* Unidirectional data flow (one-way binding).
* Requires additional libraries for state management (e.g., Redux, Context API).
* Lightweight and flexible.

**Advantages:**

* High flexibility.
* Large community support.
* Easy to learn for those familiar with JavaScript.

**Disadvantages:**

* More boilerplate for complete application setup.
* Requires knowledge of additional libraries for complex projects.

**Angular:**

* Full-fledged framework with built-in tools.
* Uses TypeScript and two-way data binding.
* Component-based architecture.
* Opinionated structure with strict coding patterns.

**Advantages:**

* Complete solution for large-scale applications.
* Built-in dependency injection and routing.
* Strong support for unit testing.

**Disadvantages:**

* Steeper learning curve.
* Heavier framework compared to React.

**2. In a Create React App project, which component renders first and how is it configured?**

* The first component to render is App. It is configured in the index.js file, where ReactDOM.render(<App />, document.getElementById('root')); initializes the application by rendering the App component inside the root DOM element.

**3. Can you explain the purpose and contents of package.json? What configurations and dependencies does it typically contain?**

**Purpose:**

* Manages project dependencies, scripts, and metadata.

**Contents:**

* name and version: Project name and version.
* scripts: Commands to build, test, and run the project.
* dependencies: Packages required for production.
* devDependencies: Packages for development (e.g., testing tools).
* browserslist: Targeted browser environments.

**React Router DOM Implementation**

**4. Explain React Router DOM implementation:**

**How do you set up routing in a React application?**

* Install react-router-dom.
* Use BrowserRouter to wrap the application.
* Define routes using Route components inside Routes.

**How do you configure parent and child routes?**

* Parent routes are defined using Route components.
* Child routes are nested within parent Route components.

**What is BrowserRouter and how is it implemented?**

* BrowserRouter is the router implementation that uses the HTML5 history API to manage routing. It should be wrapped around the main component in index.js.

**Hooks in React**

**5. What are React hooks and why were they introduced?**

* Hooks allow state and lifecycle management in functional components. Introduced to replace class components, simplifying the code and avoiding issues with this.

**Examples:**

* useState: Manages state.
* useEffect: Handles side effects.

**6. State Management:**

**How do you define state using useState hook?**

const [state, setState] = useState(initialValue);

**Why can't we update state directly by assignment?**

* Directly mutating state does not trigger a re-render, which can lead to stale UI.

**What is immutability and why is it important in React?**

* Immutability ensures state changes create new objects, triggering React to update the DOM efficiently.

**JSX**

**7. What is JSX and how does it work?**

* JSX is a syntax extension that allows mixing HTML with JavaScript. It transpiles to React.createElement calls.

**Rules and limitations:**

* Must have a single parent element.
* JavaScript expressions inside {}.
* Components must start with an uppercase letter.

**API and Data Handling**

**8. Which packages are commonly used for making API calls in React?**

* axios
* fetch

**9. Explain lazy loading:**

**What is lazy loading and how is it implemented?**

* Lazy loading defers loading non-critical resources. In React, it is implemented using React.lazy and Suspense.

**Example:**

const LazyComponent = React.lazy(() => import('./LazyComponent'));

**File Upload Implementation**

**10. Explain the complete flow of file upload functionality:**

**Frontend Implementation:**

* Use an HTML <input type="file"> element.
* Handle file selection with onChange.
* Send file data via FormData to the backend using axios or fetch.

**Backend Handling:**

* Receive file with a middleware like multer in Node.js.
* Save the file to the server or cloud storage.

**Considerations for large file uploads:**

* Use chunked uploads.
* Provide progress feedback to the user.

**11. AWS S3 Related:**

**How do you integrate S3 for file storage?**

* Use the AWS SDK to upload files programmatically.

**What is a pre-signed URL and when would you use it?**

* A pre-signed URL allows users to upload/download files securely without exposing credentials. Useful for direct client-to-S3 interactions.

**12. Database Design for File Storage:**

**What information would you store in the database for uploaded files?**

* File name, path, size, type, and metadata (e.g., upload timestamp).

**How would you handle file metadata?**

* Store metadata in a separate table or as part of the file record.

**How would you implement file retrieval?**

* Provide an API endpoint to fetch file details and a link to download.

**13. Performance Considerations:**

**How would you handle large file uploads (like video files)?**

* Implement chunked uploads and resumable uploads.

**Strategies for optimal file transfer:**

* Use compression and content delivery networks (CDNs).

**How would you implement progress indicators?**

* Use onUploadProgress in axios or XHR events in fetch.

**System Design**

**14. Design questions:**

**How would you design a system to handle multiple file uploads?**

* Use batch processing and parallel uploads.

**How would you implement file type validation?**

* Validate file types on both client and server side.

**How would you handle error cases in file uploads?**

* Return meaningful error messages and implement retry mechanisms.

**15. Security Considerations:**

**What security measures would you implement for file uploads?**

* Validate file types and size.
* Scan for malware.

**How would you handle file access permissions?**

* Use signed URLs or token-based authentication.

**How would you prevent malicious file uploads?**

* Restrict file extensions and implement virus scanning.

**16. Advanced Concepts:**

**How would you implement chunked file upload?**

* Split files into smaller parts and upload sequentially.

**What's the difference between multipart/form-data and base64 encoding?**

* multipart/form-data: Used for uploading files.
* base64: Encodes binary data as text (less efficient).

**How would you handle file compression before upload?**

* Use libraries like pako or zlib to compress files.

**17. Scaling Considerations:**

**How would your implementation change with 1000+ concurrent users?**

* Use load balancers and horizontal scaling.
* Optimize server resources with autoscaling.

**What caching strategies would you implement?**

* Use CDNs and in-memory caches (e.g., Redis).

**How would you handle rate limiting?**

* Implement rate limiting at API gateways or using middleware (e.g., express-rate-limit).

**React Core Concepts**

**1. What are the main differences between React and Angular? What are the advantages and disadvantages of each?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **React** | **Angular** |
| Type | Library (for UI) | Full-fledged framework |
| Language | JSX | TypeScript |
| Data Binding | One-way | Two-way |
| DOM | Virtual DOM | Real DOM + Incremental DOM |
| Learning Curve | Moderate | Steep |
| Advantages | Flexible, lightweight, fast updates | Comprehensive, strong tooling |
| Disadvantages | Requires additional libraries for routing, state management | Heavy, slower for small projects |

**2. In a Create React App project, which component renders first and how is it configured?**

* The App component renders first.
* Configured in the src/index.js file using ReactDOM.render(<App />, document.getElementById('root')).

**3. Can you explain the purpose and contents of package.json? What configurations and dependencies does it typically contain?**

* Purpose:
  + Metadata about the project.
  + Lists dependencies, scripts, and configurations.
* Typical Contents:
  + **Dependencies:** React, ReactDOM, etc.
  + **DevDependencies:** Webpack, Babel.
  + **Scripts:** start, build, test.
  + **Versioning:** version, license.
  + **Custom Configurations:** Proxy settings, environment variables.

**4. Explain React Router DOM implementation:**

**o How do you set up routing in a React application?**

* Install React Router DOM: npm install react-router-dom.
* Import components: BrowserRouter, Routes, Route.
* Define routes in App.js:
* <BrowserRouter>
* <Routes>
* <Route path="/" element={<Home />} />
* <Route path="/about" element={<About />} />
* </Routes>

</BrowserRouter>

**o How do you configure parent and child routes?**

<Routes>

<Route path="dashboard" element={<Dashboard />}>

<Route path="profile" element={<Profile />} />

<Route path="settings" element={<Settings />} />

</Route>

</Routes>

**o What is BrowserRouter and how is it implemented?**

* BrowserRouter uses the HTML5 history API to manage URL paths.
* Wraps the application to enable routing:
* <BrowserRouter>
* <App />

</BrowserRouter>

**5. Hooks in React:**

**o What are React hooks and why were they introduced?**

* Hooks allow functional components to manage state and lifecycle methods.
* Introduced to simplify complex class components and share logic.

**o Explain useState hook with an example:**

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

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

**o What is the purpose of useEffect hook?**

* Handles side effects like API calls, subscriptions, or DOM updates.

**o What are the different dependency scenarios in useEffect?**

1. Empty array []: Runs once on mount.
2. No dependencies: Runs after every render.
3. Specific dependencies: Runs when dependencies change.

**6. State Management:**

**o How do you define state using useState hook?**

const [state, setState] = useState(initialValue);

**o Why can't we update state directly by assignment?**

* Direct updates don’t trigger re-renders.
* Use setState to ensure the UI updates.

**o What is immutability and why is it important in React?**

* Ensures predictable state updates.
* Avoids unintended side effects and simplifies debugging.

**7. JSX:**

**o What is JSX and how does it work?**

* JSX: Syntax extension for JavaScript.
* Transpiles to React.createElement calls.

**o What are the rules and limitations of JSX?**

1. Return a single parent element.
2. Use className instead of class.
3. Close all tags, even self-closing ones.
4. JavaScript expressions inside {}.

**API and Data Handling**

**8. Which packages are commonly used for making API calls in React?**

* Axios
* Fetch API (native)
* React Query

**9. Explain lazy loading:**

**o What is lazy loading and how is it implemented?**

* Defers loading components until needed.
* Use React.lazy and Suspense:
* const LazyComponent = React.lazy(() => import('./LazyComponent'));
* <Suspense fallback={<div>Loading...</div>}>
* <LazyComponent />

</Suspense>

**o What's the difference between lazy loading and eager loading?**

* Lazy: Loads on demand.
* Eager: Loads upfront.

**o When should you use lazy loading?**

* Large applications with infrequently used components.

**10. Explain the complete flow of file upload functionality:**

**o How would you implement file upload from frontend to backend?**

1. Frontend:
   * Use input type="file".
   * Send file using FormData via Axios or Fetch.
2. Backend:
   * Use middleware like multer to handle file uploads.

**o What considerations should be made for large file uploads?**

* Chunked uploads.
* Increase server timeouts.

**o How would you handle different file types?**

* Validate MIME types.
* Use file type libraries.

**11. AWS S3 Related:**

**o How do you integrate S3 for file storage?**

* Use AWS SDK:
* const s3 = new AWS.S3();

s3.upload({ Bucket, Key, Body }, callback);

**o What is a pre-signed URL and when would you use it?**

* Temporarily grants secure upload/download access.

**o What is an unsigned URL in S3 and how does it differ from pre-signed URLs?**

* Unsigned: Public access to files.
* Pre-signed: Temporary and secure.

**12. Database Design for File Storage:**

**o What information would you store in the database for uploaded files?**

* Filename, path, size, MIME type, upload date.

**o How would you handle file metadata?**

* Store as JSON or separate fields.

**o How would you implement file retrieval?**

* Serve files via a CDN or S3 URL.

**13. Performance Considerations:**

**o How would you handle large file uploads (like video files)?**

* Chunked uploads.
* Use CDNs for delivery.

**o What strategies would you use for optimal file transfer?**

* Compression, caching, and parallel uploads.

**o How would you implement progress indicators for file uploads?**

* Use Axios progress event listeners or HTML5 APIs.

**System Design**

**14. Design questions:**

**o How would you design a system to handle multiple file uploads?**

* Use parallel uploads with retry logic.

**o How would you implement file type validation?**

* Validate on both frontend and backend.

**o How would you handle error cases in file uploads?**

* Provide detailed error messages and retry options.

**15. Security Considerations:**

**o What security measures would you implement for file uploads?**

* Validate files, sanitize filenames, and scan for malware.

**o How would you handle file access permissions?**

* Use signed URLs and IAM roles.

**o How would you prevent malicious file uploads?**

* Restrict file types, scan for viruses, and limit upload size.

**16. Advanced Concepts:**

**o How would you implement chunked file upload?**

* Split files into chunks, upload, and reassemble on the server.

**o What's the difference between multipart/form-data and base64 encoding?**

* Multipart: Efficient for binary files.
* Base64: Increases size, not efficient for large files.

**o How would you handle file compression before upload?**

* Use client-side compression libraries like pako.

**17. Scaling Considerations:**

**o How would your implementation change with 1000+ concurrent users?**

* Load balancing, horizontal scaling.

**o What caching strategies would you implement?**

* Use Redis or CDN for caching.

**o How would you handle rate limiting?**

* Implement throttling with tools like express-rate-limit or AWS API Gateway.