Topic 1: React Fundamentals (Components, JSX, State, Props)

1. **What is the primary way to create a functional component in React?**
   - A) Using class syntax
   - B) Using the function keyword or arrow functions
   - C) Using `React.createComponent`
   - D) Using `useComponent` hook

   Answer: B
   Explanation: Functional components are defined using the `function` keyword or arrow functions, and are the preferred approach in modern React (React Docs).

2. **How do you pass data from a parent component to a child component in React?**
   - A) Using state
   - B) Using props
   - C) Using context
   - D) Using refs

   Answer: B
   Explanation: Props are the primary mechanism for passing data from parent to child components (React Docs).

3. **What is JSX?**
   - A) A JavaScript extension that allows writing HTML-like code in JavaScript
   - B) A CSS preprocessor
   - C) A state management library
   - D) A routing library

   Answer: A
   Explanation: JSX is a syntax extension for JavaScript that allows writing HTML-like code, which is transpiled to `React.createElement` calls (React Docs).

4. **How do you conditionally render a component in JSX?**
   - A) Using `if` statements inside JSX
   - B) Using ternary operators or logical AND
   - C) Using `switch` cases
   - D) Using `for` loops

   Answer: B
   Explanation: JSX supports JavaScript expressions, so ternary operators (`?:`) and logical AND (`&&`) are used for conditional rendering (React Docs).

5. **What is the purpose of the `key` prop in lists?**
   - A) To identify unique items for efficient updates
   - B) To style list items
   - C) To handle events on list items
   - D) To manage state in list components

   Answer: A

Explanation: The `key` prop helps React identify which items have changed, are added, or removed, enabling efficient DOM updates (React Docs).

6. **In React 19, what is the new way to handle form actions?**
   - A) Using `onSubmit` event handlers
   - B) Using the `action` prop on `<form>`
   - C) Using `useForm` hook
   - D) Using `ReactDOM.createForm`

   Answer: B
   Explanation: React 19 introduces support for the `action` prop on `<form>`, allowing functions to handle form submissions (React Docs).

7. **In a standard Create React App project, where should you place a new component?**
   - A) In the `public` folder
   - B) In the `src` folder
   - C) In the `node_modules` folder
   - D) In the `build` folder

   Answer: B
   Explanation: Components should be placed in the `src` folder, as that's where application code resides (Create React App Docs).

8. **What happens if you call `setState` inside a `useEffect` hook without dependencies?**
   - A) It updates state only once
   - B) It causes an infinite render loop
   - C) It throws an error
   - D) It updates state conditionally

   Answer: B
   Explanation: Without a dependency array, `useEffect` runs on every render, and calling `setState` inside it triggers another render, leading to an infinite loop (React Docs).

9. **Which of the following correctly renders a list of numbers in JSX?**
   - A) `<div>{numbers.map(num => <p>{num}</p>)}</div>`
   - B) `<div>{for (let num of numbers) <p>{num}</p>}</div>`
   - C) `<div>{numbers.forEach(num => <p>{num}</p>)}</div>`
   - D) `<div>{numbers.map(<p>{num}</p>)}</div>`

   Answer: A
   Explanation: `map` returns an array suitable for rendering in JSX, while `for` and `forEach` are not expressions that return values (React Docs).

10. **Can a child component directly modify the state of its parent component?**
    - A) Yes, by using `this.props.state`
    - B) Yes, by calling `this.props.setState`
    - C) No, child components cannot directly modify parent state
    - D) Only if the parent passes a callback function

    Answer: D

Explanation: Child components cannot directly modify parent state; the parent must pass a callback function (e.g., a setter from `useState`) for the child to trigger state changes (React Docs).

Topic 2: React Hooks

11. **Which hook is used to manage state in functional components?**
    - A) `useEffect`
    - B) `useState`
    - C) `useContext`
    - D) `useReducer`

    Answer: B
    Explanation: `useState` is the primary hook for adding state to functional components (React Docs).

12. **What is the purpose of the `useEffect` hook?**
    - A) To manage state
    - B) To handle side effects like data fetching
    - C) To provide context
    - D) To optimize rendering

    Answer: B
    Explanation: `useEffect` is used for side effects such as data fetching, subscriptions, or DOM manipulation (React Docs).

13. **How do you create a custom hook in React?**
    - A) By defining a function that uses other hooks
    - B) By extending `React.Hook` class
    - C) By using `createHook` method
    - D) Custom hooks are not possible

    Answer: A
    Explanation: Custom hooks are functions that use built-in hooks and can be shared across components (React Docs).

14. **What does the `useMemo` hook do?**
    - A) Memoizes a component
    - B) Memoizes a value to avoid expensive calculations
    - C) Manages state
    - D) Handles side effects

    Answer: B
    Explanation: `useMemo` memoizes a computed value, recalculating only when dependencies change (React Docs).

15. **When should you use `useCallback`?**
    - A) To memoize values
    - B) To memoize functions to prevent re-creation

- C) To handle asynchronous operations
- D) To manage context
  Answer: B
  Explanation: `useCallback` memoizes functions, useful for passing stable references to child components (React Docs).

16. **What is the purpose of `useRef`?**

- A) To access DOM elements
- B) To store mutable values that don't trigger re-renders
- C) Both A and B
- D) To manage state
  Answer: C
  Explanation: `useRef` holds references to DOM elements and stores mutable values without causing re-renders (React Docs).

17. **In React 19, what does the `useActionState` hook provide?**

- A) State management for forms
- B) Handling of async actions with pending, error, and success states
- C) Optimistic updates
- D) Context management
  Answer: B
  Explanation: `useActionState` simplifies handling async actions by providing state for pending, error, and success (React Docs).

18. **How does `useTransition` help with performance?**

- A) By deferring state updates
- B) By batching updates
- C) By allowing non-urgent updates to be interrupted
- D) By memoizing components
  Answer: C
  Explanation: `useTransition` marks updates as non-urgent, allowing React to interrupt them for higher-priority updates (React Docs).

19. **What is the difference between `useEffect` and `useLayoutEffect`?**

- A) `useEffect` runs after paint, `useLayoutEffect` before paint
- B) `useEffect` is for side effects, `useLayoutEffect` for state management
- C) `useEffect` is asynchronous, `useLayoutEffect` is synchronous
- D) Both A and C
  Answer: D
  Explanation: `useLayoutEffect` runs synchronously after DOM mutations but before painting, while `useEffect` runs asynchronously after painting (React Docs).

20. **Can you call hooks inside loops or conditionals?**

- A) Yes, always
- B) No, hooks must be called at the top level

- C) Only in functional components
- D) Only in class components
  Answer: B
  Explanation: Hooks must be called at the top level of components or custom hooks to ensure consistent order (React Docs).

Topic 3: Advanced React Patterns

21. **What is the Context API used for?**

- A) Managing local state
- B) Sharing data across components without prop drilling
- C) Handling side effects
- D) Optimizing performance
  Answer: B
  Explanation: Context API allows sharing data like themes or user data across the component tree without passing props manually (React Docs).

22. **How do you create a context in React?**

- A) Using `React.createContext()`
- B) Using `useContext()`
- C) Using `Context.Provider`
- D) Using `useState()`
  Answer: A
  Explanation: `React.createContext()` creates a context object for use with providers and consumers (React Docs).

23. **What is a Higher-Order Component (HOC)?**

- A) A component that renders other components
- B) A function that takes a component and returns a new component with additional props or behavior
- C) A component that uses hooks
- D) A component that manages state
  Answer: B
  Explanation: HOCs are functions that enhance components by adding functionality, like `withRouter` or `connect` in Redux (React Docs).

24. **What is the render prop pattern?**

- A) Passing a function as a prop to control what a component renders
- B) Rendering components conditionally
- C) Using props to manage state
- D) Rendering components on the server
  Answer: A
  Explanation: Render props allow components to share code by passing a function that returns what to render (React Docs).

25. **In React 19, what are Server Components?**

- A) Components that run only on the client
- B) Components that can be rendered on the server with zero client-side JavaScript
- C) Components that manage server-side state
- D) Components that handle server requests
  Answer: B
  Explanation: Server Components enable server-side rendering with minimal client-side JavaScript (React Docs).

26. **How do you define a Server Action in React 19?**

- A) Using the `use server` directive
- B) Using the `use client` directive
- C) Using the `useEffect` hook
- D) Using the `useState` hook
  Answer: A
  Explanation: Server Actions are async functions marked with `use server` to run on the server (React Docs).

27. **What is the purpose of `useOptimistic`?**

- A) To manage state
- B) To show optimistic UI updates during async requests
- C) To memoize values
- D) To handle errors
  Answer: B
  Explanation: `useOptimistic` displays temporary UI updates during async operations, reverting on error (React Docs).

28. **How does React 19 improve error reporting for hydration mismatches?**

- A) By throwing multiple errors
- B) By logging a single error with a diff
- C) By ignoring mismatches
- D) By stopping the render process
  Answer: B
  Explanation: React 19 logs a single error with a diff to help identify hydration mismatches (React Docs).

29. **What is the benefit of using `<Context>` as a provider in React 19?**

- A) It reduces bundle size
- B) It simplifies the syntax for context providers
- C) It improves performance
- D) It handles asynchronous operations
  Answer: B
  Explanation: In React 19, `<Context>` can be used directly as a provider, simplifying syntax (React Docs).

30. **How can you handle cleanup for refs in React 19?**

- A) By returning a cleanup function from the ref callback
- B) By using `useEffect`
- C) By calling `ref.current = null`
- D) By using `useRef` with a dependency array

  Answer: A

  Explanation: Ref callbacks can return cleanup functions, called when the ref is detached (React Docs).

Topic 4: React Routing with React Router

31. **What is the primary purpose of React Router?**

- A) To manage state in React applications
- B) To handle navigation and routing in single-page applications
- C) To optimize performance
- D) To handle forms and inputs

  Answer: B

  Explanation: React Router manages navigation and routing in React applications, enabling different views based on URLs (React Router Docs).

32. **In React Router v7, how do you define a route?**

- A) `<Route path="/path" component={Component} />`
- B) `<Route path="/path" element={<Component />} />`
- C) `<Route to="/path" element={<Component />} />`
- D) `<Link path="/path" element={<Component />} />`

  Answer: B

  Explanation: In React Router v6 and v7, routes use the `element` prop (React Router Docs).

33. **What is the difference between `<Link>` and `<NavLink>` in React Router?**

- A) `<Link>` is for navigation, `<NavLink>` is for active class styling
- B) `<NavLink>` is for navigation, `<Link>` is for active class styling
- C) Both are the same
- D) `<Link>` is deprecated

  Answer: A

  Explanation: `<Link>` is for navigation, while `<NavLink>` adds styling for active links (React Router Docs).

34. **How do you access route parameters in a functional component?**

- A) `useParams` hook
- B) `this.props.match.params`
- C) `useRouteMatch` hook
- D) `useLocation` hook

  Answer: A

  Explanation: `useParams` is used in functional components to access route parameters (React Router Docs).

35. **What is the purpose of the `useNavigate` hook?**

- A) To get the current location
- B) To programmatically navigate to another route
- C) To access route parameters
- D) To handle form submissions

  Answer: B

  Explanation: `useNavigate` enables programmatic navigation, replacing `history.push` (React Router Docs).

36. **How can you protect routes in React Router?**

- A) Using `<PrivateRoute>` component
- B) Using conditional rendering based on authentication state
- C) Both A and B
- D) Routes cannot be protected

  Answer: C

  Explanation: Protected routes can be implemented with a `<PrivateRoute>` component or conditional rendering (React Router Docs).

37. **What is a nested route in React Router?**

- A) A route defined inside another route
- B) A route that shares the same path
- C) A route that uses the same component
- D) A route with multiple paths

  Answer: A

  Explanation: Nested routes define sub-routes within a parent route, useful for layouts (React Router Docs).

38. **How do you handle 404 pages in React Router?**

- A) Using `<Route path="*" element={<NotFound />} />`
- B) Using `<Route path="/404" element={<NotFound />} />`
- C) Using `useHistory` to redirect
- D) Using `window.location`

  Answer: A

  Explanation: The `*` path matches any undefined route, rendering a 404 component (React Router Docs).

39. **What is the difference between `BrowserRouter` and `HashRouter`?**

- A) `BrowserRouter` uses HTML5 history API, `HashRouter` uses URL hash
- B) `HashRouter` is for server-side rendering, `BrowserRouter` for client-side
- C) Both are the same
- D) `HashRouter` is deprecated

  Answer: A

  Explanation: `BrowserRouter` uses the HTML5 history API for clean URLs, while `HashRouter` uses URL hashes for compatibility (React Router Docs).

40. **In React Router v7, how do you define a route with a loader?**

- A) `<Route path="/path" loader={loaderFunction} element={<Component />} />`
- B) Using `useLoader` hook
- C) Using `data` prop in `<Route>`
- D) Loaders are not supported
  Answer: A
  Explanation: Routes can include a `loader` function to fetch data before rendering (React Router Docs).

Topic 5: React Optimization Techniques

41. **What is code splitting in React?**

- A) Dividing code into smaller functions
- B) Loading parts of the application on demand
- C) Using multiple components
- D) Optimizing state management
  Answer: B
  Explanation: Code splitting enables lazy loading of application parts, improving initial load time (React Docs).

42. **How do you implement lazy loading in React?**

- A) Using `React.lazy` and `Suspense`
- B) Using `useEffect`
- C) Using `useMemo`
- D) Using `useCallback`
  Answer: A
  Explanation: `React.lazy` and `Suspense` enable lazy loading of components (React Docs).

43. **What does `React.memo` do?**

- A) Memoizes a component to prevent re-renders if props are the same
- B) Memoizes a value
- C) Handles side effects
- D) Manages state
  Answer: A
  Explanation: `React.memo` prevents unnecessary re-renders by comparing props (React Docs).

44. **When should you use `useMemo`?**

- A) For expensive calculations that don't need to run on every render
- B) For managing state
- C) For handling events
- D) For accessing DOM elements
  Answer: A
  Explanation: `useMemo` caches results of expensive computations (React Docs).

45. **What is the purpose of `useCallback`?**

- A) To memoize functions

- B) To memoize components
- C) To handle asynchronous operations
- D) To manage context

  Answer: A

  Explanation: `useCallback` memoizes functions to prevent re-creation on every render (React Docs).

46. **How can you optimize a list of components in React?**

- A) Using `key` prop
- B) Using `React.memo` on list items
- C) Both A and B
- D) Using `useEffect`

  Answer: C

  Explanation: The `key` prop aids efficient list updates, and `React.memo` prevents unnecessary re-renders of list items (React Docs).

47. **What is the benefit of using `useTransition`?**

- A) It allows smoother UI updates by marking updates as non-urgent
- B) It manages state transitions
- C) It handles form submissions
- D) It optimizes image loading

  Answer: A

  Explanation: `useTransition` keeps the UI responsive by allowing non-urgent updates to be interrupted (React Docs).

48. **In React 19, what is the `useDeferredValue` hook used for?**

- A) To defer state updates
- B) To provide an initial value for state
- C) To handle asynchronous data
- D) To memoize values

  Answer: A

  Explanation: `useDeferredValue` defers updates to a value, aiding UI responsiveness during heavy computations (React Docs).

49. **How does React's concurrent rendering improve performance?**

- A) By allowing multiple renders at the same time
- B) By prioritizing important updates
- C) By batching state updates
- D) By reducing the number of components

  Answer: B

  Explanation: Concurrent rendering prioritizes updates, enhancing UI responsiveness (React Docs).

50. **What is the purpose of the `startTransition` function?**

- A) To start a transition effect

- B) To mark a state update as non-urgent
- C) To handle form submissions
- D) To optimize rendering
  Answer: B
  Explanation: `startTransition` marks state updates as non-urgent, allowing React to batch and prioritize updates (React Docs).

Topic 6: React Native Fundamentals

1. **What is the primary difference between React and React Native components?**

   - A) React uses HTML elements, React Native uses native components.
   - B) React is for web, React Native is for mobile.
   - C) Both A and B.
   - D) There is no difference.
     Answer: C
     Explanation: React uses HTML-like elements via JSX for web development, while React Native uses native components like `View` and `Text` for mobile platforms. Additionally, React targets web browsers, whereas React Native targets iOS and Android (React Native Docs).

2. **What is the equivalent of `<div>` in React Native?**

   - A) `View`
   - B) `Text`
   - C) `Image`
   - D) There is no equivalent
     Answer: A
     Explanation: The `View` component is the primary layout component in React Native, analogous to `<div>` in HTML, used for structuring UI elements (React Native Docs).

3. **How do you handle text input in React Native?**

   - A) Using `<input>` component
   - B) Using `TextInput` component
   - C) Using `Text` component
   - D) There is no built-in component for text input
     Answer: B
     Explanation: The `TextInput` component is used for capturing user input, similar to `<input type="text">` in HTML (React Native Docs).

4. **What is the role of the `AppRegistry` in React Native?**

   - A) To register the root component of the app
   - B) To handle navigation
   - C) To manage state
   - D) To style components
     Answer: A
     Explanation: `AppRegistry` registers the root component, serving as the entry point for the React Native app (React Native Docs).

5. **How do you handle images in React Native?**

   - A) Using `<img>` tag
   - B) Using `Image` component
   - C) Using `BackgroundImage` component
   - D) Images are not supported

   Answer: B
   Explanation: The `Image` component displays images from local or remote sources in React Native (React Native Docs).

6. **What is the difference between `require` and `import` in React Native for images?**

   - A) `require` is for static assets, `import` is for modules
   - B) There is no difference
   - C) `import` is for images, `require` is for modules
   - D) Both can be used interchangeably

   Answer: A
   Explanation: `require` is used for static assets like images (e.g., `require('./image.png')`), while `import` is used for JavaScript modules (React Native Docs).

7. **How do you handle permissions in React Native?**

   - A) Using the `PermissionsAndroid` module for Android and `request` for iOS
   - B) Permissions are handled automatically
   - C) Using third-party libraries
   - D) Permissions are not needed in React Native

   Answer: A
   Explanation: `PermissionsAndroid` handles permissions on Android, while iOS permissions often require native modules or libraries like `react-native-permissions` (React Native Docs).

8. **What is the role of the `Linking` module in React Native?**

   - A) To handle deep linking and URL opening
   - B) To manage navigation between screens
   - C) To link components together
   - D) To handle API requests

   Answer: A
   Explanation: The `Linking` module manages deep linking, opening URLs, and interacting with the app's linking system (React Native Docs).

9. **How do you handle platform-specific code in React Native?**

   - A) Using conditional statements based on `Platform.OS`
   - B) Using separate components for iOS and Android
   - C) Both A and B
   - D) There is no need for platform-specific code

   Answer: C
   Explanation: You can use `Platform.OS` for conditional logic or create separate files with `.ios.js` and `.android.js` extensions for platform-specific code (React Native Docs).

10. **How do you play audio in React Native?**

    - A) Using `Audio` component from `react-native`
    - B) Using third-party libraries like `react-native-sound`
    - C) Audio is not supported
    - D) Using HTML5 audio

      Answer: B

      Explanation: React Native does not have a built-in audio component; libraries like `react-native-sound` or `expo-av` are used for audio playback (React Native Docs).

Topic 7: Styling and Layout in React Native

11. **What is the default layout direction in React Native?**

    - A) Row
    - B) Column
    - C) Flex
    - D) Grid

      Answer: B

      Explanation: Flex containers in React Native default to a column layout, aligning children vertically (React Native Docs).

12. **How do you make a component take up the full screen in React Native?**

    - A) Set `width: '100%', height: '100%'`
    - B) Use `flex: 1`
    - C) Both A and B
    - D) It's not possible

      Answer: C

      Explanation: Using `flex: 1` makes a `View` fill available space, while explicit `width` and `height` can also work, though `flex` is more common (React Native Docs).

13. **What is the purpose of the `justifyContent` property?**

    - A) To align items horizontally
    - B) To align items vertically
    - C) To distribute space between items
    - D) To set the background color

      Answer: C

      Explanation: `justifyContent` distributes space along the main axis of a flex container, controlling spacing between items (React Native Docs).

14. **How do you center a child component within a parent in React Native?**

    - A) Set `alignItems: 'center', justifyContent: 'center'` on the parent
    - B) Set `margin: 'auto'` on the child
    - C) Use `position: 'absolute', top: 0, left: 0, right: 0, bottom: 0` on the child
    - D) All of the above can work depending on the context

      Answer: D

Explanation: Flexbox properties (`alignItems`, `justifyContent`) or absolute positioning can center a child, depending on the use case (React Native Docs).

15. **What is the difference between `style` and `styles` in React Native?**
    - A) `style` is for inline styles, `styles` is for `StyleSheet.create`
    - B) There is no difference
    - C) `styles` is for global styles, `style` is for component styles
    - D) `style` is deprecated

    Answer: A
    Explanation: The `style` prop accepts inline style objects or arrays, while `styles` typically refers to the object created by `StyleSheet.create` (React Native Docs).

16. **What is the purpose of the `transform` property in React Native styles?**
    - A) To apply transformations like rotate, scale, translate
    - B) To change the component's type
    - C) To handle touch events
    - D) To set the opacity

    Answer: A
    Explanation: The `transform` property applies transformations like rotation, scaling, or translation to components (React Native Docs).

17. **How do you make a component scrollable in React Native?**
    - A) Wrap it in a `ScrollView` component
    - B) Set `overflow: 'scroll'`
    - C) Use `FlatList` for lists
    - D) Both A and C

    Answer: D
    Explanation: `ScrollView` is used for general scrollable content, while `FlatList` is optimized for large lists (React Native Docs).

18. **What is the purpose of the `elevation` property in React Native styles for Android?**
    - A) To add a drop shadow effect
    - B) To raise the component above others
    - C) Both A and B
    - D) Elevation is not supported

    Answer: C
    Explanation: `elevation` adds a drop shadow and raises the component, a feature of Android's Material Design (React Native Docs).

19. **How do you make a component rounded in React Native?**
    - A) Set `borderRadius` in styles
    - B) Use `cornerRadius`
    - C) Rounded corners are not supported
    - D) Use `borderWidth`

    Answer: A

Explanation: `borderRadius` sets the corner radius for components like `View` or `Image` (React Native Docs).

20. **How do you define a style in React Native?**

    - A) Using CSS files
    - B) Using inline styles with JavaScript objects
    - C) Using `StyleSheet.create`
    - D) Both B and C

    Answer: D
    Explanation: Styles can be defined inline using JavaScript objects or with `StyleSheet.create` for better performance and readability (React Native Docs).

Topic 8: Navigation in React Native

21. **What is the basic unit of navigation in React Navigation?**

    - A) Screen
    - B) Navigator
    - C) Route
    - D) Stack

    Answer: A
    Explanation: Screens are the basic units in React Navigation, representing individual views, with navigators managing transitions between them (React Navigation Docs).

22. **What is the purpose of the `NavigationContainer` component?**

    - A) To wrap the entire app and provide navigation context
    - B) To define a single screen
    - C) To handle deep linking
    - D) To style the navigation bar

    Answer: A
    Explanation: `NavigationContainer` is the root component that provides navigation state and context to its children (React Navigation Docs).

23. **How do you define a stack navigator in React Navigation?**

    - A) Using `createStackNavigator`
    - B) Using `Stack.Navigator`
    - C) Using `NavigationContainer`
    - D) Stack navigators are not supported

    Answer: B
    Explanation: In React Navigation 5 and above, `Stack.Navigator` is used to create a stack navigator, after importing `createStackNavigator` (React Navigation Docs).

24. **How do you pass parameters between screens in React Navigation?**

    - A) Using the `params` option in the navigation prop
    - B) Using global state
    - C) Using context API

- D) Parameters cannot be passed
  Answer: A
  Explanation: Parameters are passed using `navigation.navigate('Screen', { param: value })` (React Navigation Docs).

25. **What is a tab navigator used for?**

- A) To switch between different stacks or screens horizontally
- B) To stack screens vertically
- C) To handle drawer navigation
- D) Tab navigators are not part of React Navigation
  Answer: A
  Explanation: Tab navigators enable switching between screens or stacks using tabs, typically at the bottom of the screen (React Navigation Docs).

26. **In React Navigation, what does the `navigate` action do in a stack navigator?**

- A) Pushes a new screen onto the stack
- B) Replaces the current screen
- C) Goes back to the previous screen
- D) Resets the navigation state
  Answer: A
  Explanation: In a stack navigator, `navigate` pushes a new screen onto the stack, allowing users to return to the previous screen (React Navigation Docs).

27. **What is the difference between `navigate` and `push` in React Navigation?**

- A) `navigate` checks for existing screens, `push` always adds a new screen
- B) `push` is for tab navigation, `navigate` for stack navigation
- C) There is no difference
- D) `push` is deprecated
  Answer: A
  Explanation: `navigate` focuses an existing screen if present, updating params, while `push` always adds a new screen to the stack (React Navigation Docs).

28. **How do you handle deep linking in React Navigation?**

- A) Using the `linking` prop in `NavigationContainer`
- B) Using the `Linking` module alone
- C) Deep linking is not supported
- D) Using `useDeepLink` hook
  Answer: A
  Explanation: The `linking` prop in `NavigationContainer` configures deep linking for specific routes (React Navigation Docs).

29. **What is the purpose of the `initialRouteName` prop in a navigator?**

- A) To set the default screen when the navigator loads
- B) To define the navigation type
- C) To style the navigator

- D) To reset the navigation state
  Answer: A
  Explanation: `initialRouteName` specifies the screen to display when the navigator is first rendered (React Navigation Docs).

30. **How do you customize the header in a stack navigator?**

- A) Using the `options` prop on `Stack.Screen`
- B) Using `headerStyle` in `StyleSheet`
- C) Using `NavigationContainer` props
- D) Headers cannot be customized
  Answer: A
  Explanation: The `options` prop on `Stack.Screen` allows customization of the header, such as title or style (React Navigation Docs).

Topic 9: State Management and Data Flow in React Native

31. **What is the simplest way to manage state in a React Native component?**

- A) Using `useState` hook
- B) Using Redux
- C) Using Context API
- D) Using AsyncStorage
  Answer: A
  Explanation: `useState` is the simplest way to manage local state in functional components (React Docs).

32. **When should you use the Context API for state management?**

- A) When state needs to be shared across many components
- B) For local component state
- C) For persisting state
- D) Context API is not for state management
  Answer: A
  Explanation: Context API is ideal for sharing state across components without prop drilling (React Docs).

33. **What is Redux used for in React Native?**

- A) Global state management
- B) Local state management
- C) Styling
- D) Navigation
  Answer: A
  Explanation: Redux manages global state, useful for complex applications (Redux Docs).

34. **How do you persist state in React Native?**

- A) Using AsyncStorage
- B) Using localStorage

- C) Using sessionStorage
- D) State cannot be persisted
  Answer: A
  Explanation: AsyncStorage provides persistent key-value storage for React Native apps (React Native Docs).

35. **What is the purpose of the `useReducer` hook?**

- A) To manage complex state logic
- B) To handle side effects
- C) To memoize values
- D) To manage navigation
  Answer: A
  Explanation: `useReducer` is used for complex state logic, similar to Redux reducers (React Docs).

36. **What is the benefit of using Redux in React Native?**

- A) Centralized state management
- B) Easier testing
- C) Better performance for large apps
- D) All of the above
  Answer: D
  Explanation: Redux centralizes state, simplifies testing, and can improve performance by reducing prop drilling (Redux Docs).

救助

37. **How do you connect a component to the Redux store?**

- A) Using `connect` from `react-redux`
- B) Using `useSelector` and `useDispatch` hooks
- C) Both A and B
- D) Components cannot be connected directly
  Answer: C
  Explanation: Class components use `connect`, while functional components use `useSelector` and `useDispatch` (Redux Docs).

38. **What is the `useEffect` hook used for in React Native?**

- A) To handle side effects like data fetching
- B) To manage state
- C) To render components
- D) To handle navigation
  Answer: A
  Explanation: `useEffect` handles side effects like data fetching or subscriptions (React Docs).

39. **How can you share state between multiple components without prop drilling?**

- A) Using Context API
- B) Using Redux

- C) Both A and B
- D) It's not possible
  Answer: C
  Explanation: Context API and Redux allow state sharing across components without prop drilling (React Docs).

40. **What is a common use case for AsyncStorage in React Native?**

- A) Storing temporary session data
- B) Persisting small amounts of data like user preferences
- C) Managing large datasets
- D) Handling navigation state
  Answer: B
  Explanation: AsyncStorage is ideal for persisting small amounts of data, such as user settings or tokens (React Native Docs).

Topic 10: Project Structure and Best Practices for React Native

41. **Where should you place navigation-related code in a React Native project?**

- A) In a `navigation` folder
- B) In the main `App.js`
- C) In each screen component
- D) There is no standard place
  Answer: A
  Explanation: A `navigation` folder is commonly used to organize navigation configurations for clarity (React Navigation Docs).

42. **What is the purpose of the `assets` folder in a React Native project?**

- A) To store images, fonts, and other static assets
- B) To store source code
- C) To store configuration files
- D) Assets folder is not needed
  Answer: A
  Explanation: The `assets` folder stores static assets like images and fonts (React Native Docs).

43. **How should you structure your components in a React Native project?**

- A) Group related components in folders, e.g., `components/Auth`, `components/Home`
- B) Put all components in a single `components` folder
- C) Place components inside screen folders
- D) There is no standard way
  Answer: A
  Explanation: Grouping components by feature or functionality improves organization and scalability (React Native Best Practices).

44. **What is a good practice for handling API calls in React Native?**

- A) Create a separate `services` or `api` folder with functions for each API endpoint

- B) Make API calls directly in components
- C) Use global variables for API responses
- D) API calls should be handled on the server-side
  Answer: A
  Explanation: Separating API logic into a `services` or `api` folder enhances maintainability and reusability (React Native Best Practices).

45. **How can you improve the performance of a React Native app?**

- A) Use `FlatList` for large lists
- B) Memoize components with `React.memo`
- C) Use lazy loading for images
- D) All of the above
  Answer: D
  Explanation: `FlatList`, `React.memo`, and lazy loading images are effective performance optimization techniques (React Native Docs).

46. **What is a common practice for organizing screens in a React Native project?**

- A) Place all screens in a `screens` folder
- B) Place each screen in its own folder with related components
- C) Both A and B are acceptable
- D) Screens should be in the root directory
  Answer: C
  Explanation: Both a `screens` folder and feature-based subfolders are common practices for organizing screens (React Native Best Practices).

47. **Where should you place utility functions in a React Native project?**

- A) In a `utils` folder
- B) In the component files
- C) In the main `App.js`
- D) Utility functions are not needed
  Answer: A
  Explanation: Utility functions are typically placed in a `utils` folder for reusability and organization (React Native Best Practices).

48. **How can you handle different environments (dev, prod) in React Native?**

- A) Using environment variables with `.env` files
- B) Using conditional logic based on `process.env.NODE_ENV`
- C) Both A and B
- D) Environments are handled automatically
  Answer: C
  Explanation: `.env` files with libraries like `react-native-dotenv` and `process.env.NODE_ENV` checks manage environment-specific configurations ([React Native Best Practices](https://medium.com/@beobo/struct Asc)).

49. **What is the purpose of the `index.js` file in a React Native project?**

- A) It's the entry point of the app
- B) It contains the main component
- C) It registers the app with AppRegistry
- D) All of the above
  Answer: D
  Explanation: `index.js` imports the main app component and registers it with `AppRegistry`, serving as the entry point (React Native Docs).

50. **What is a good practice for versioning your React Native app?**

- A) Update the `version` in `package.json`
- B) Use semantic versioning
- C) Include the version in the app's UI
- D) All of the above
  Answer: D
  Explanation: Updating `package.json`, using semantic versioning, and displaying the version in the UI are all good practices for versioning (Semantic Versioning).

Topic 11: Express.js Fundamentals

51. **What is Express.js?**

- A) A full-stack JavaScript framework
- B) A minimal and flexible Node.js web application framework
- C) A database ORM for Node.js
- D) A front-end JavaScript library
  Answer: B
  Explanation: Express.js is a minimal and flexible framework for building web applications and APIs on Node.js (Express Docs).

52. **What is the entry point file for an Express application?**

- A) `app.js` or `index.js`
- B) `server.js`
- C) `main.js`
- D) Any file can be the entry point
  Answer: A
  Explanation: By convention, `app.js` or `index.js` is used as the entry point for Express applications, though any file can be specified (Express Docs).

53. **How do you start an Express server?**

- A) `app.start(port)`
- B) `app.listen(port)`
- C) `server.run(port)`
- D) `app.run(port)`
  Answer: B
  Explanation: The `app.listen(port)` method starts the Express server on the specified port (Express Docs).

54. **What does `app.use()` do in Express?**

    - A) Defines a new route
    - B) Registers middleware to handle requests
    - C) Starts the server
    - D) Configures the database

    Answer: B
    Explanation: `app.use()` mounts middleware functions to process requests at the specified path (Express Docs).

55. **What is the default HTTP method for `app.get()` in Express?**

    - A) POST
    - B) GET
    - C) PUT
    - D) DELETE

    Answer: B
    Explanation: `app.get()` defines a route handler for HTTP GET requests (Express Docs).

56. **Where should you place environment variables in an Express project?**

    - A) In a `.env` file
    - B) In `package.json`
    - C) In `app.js`
    - D) Environment variables are not supported

    Answer: A
    Explanation: Environment variables are typically stored in a `.env` file and accessed using libraries like `dotenv` (Express Best Practices).

57. **What is the purpose of the `express.json()` middleware?**

    - A) To parse JSON request bodies
    - B) To serve static files
    - C) To handle CORS
    - D) To compress responses

    Answer: A
    Explanation: `express.json()` parses incoming requests with JSON payloads, making them available in `req.body` (Express Docs).

58. **How do you handle errors in an Express application?**

    - A) Using `try-catch` blocks in route handlers
    - B) Using error-handling middleware with four parameters
    - C) Both A and B
    - D) Errors are handled automatically

    Answer: C
    Explanation: `try-catch` can be used in route handlers, but Express requires error-handling middleware with `(err, req, res, next)` to catch errors globally (Express Docs).

59. **What is the default status code for a successful response in Express?**

- A) 200
- B) 201
- C) 204
- D) 404

  Answer: A

  Explanation: Express defaults to a 200 OK status code for successful responses unless specified otherwise (Express Docs).

60. **How do you set up a basic Express app?**

- A) Import `express`, create an `app`, define routes, and call `app.listen`
- B) Use `createExpressApp()`
- C) Use `express.createServer()`
- D) Express apps are configured automatically

  Answer: A

  Explanation: A basic Express app involves importing `express`, creating an instance with `express()`, defining routes, and starting the server with `app.listen` (Express Docs).

Topic 12: Express Middleware

11. **What is middleware in Express?**

- A) A function that processes requests and responses
- B) A database connection layer
- C) A routing mechanism
- D) A template engine

  Answer: A

  Explanation: Middleware functions have access to the request (`req`), response (`res`), and `next` function, allowing them to process or modify requests (Express Docs).

12. **In what order are middleware functions executed in Express?**

- A) In the order they are defined
- B) Randomly
- C) Based on HTTP method
- D) Based on route specificity

  Answer: A

  Explanation: Middleware is executed in the order it is mounted using `app.use()` or route-specific methods (Express Docs).

13. **What does the `next()` function do in Express middleware?**

- A) Sends the response to the client
- B) Calls the next middleware or route handler
- C) Terminates the request
- D) Parses the request body

  Answer: B

  Explanation: `next()` passes control to the next middleware or route handler in the stack (Express Docs).

14. **How do you create a custom middleware in Express?**

    - A) Define a function with `(req, res, next)` parameters
    - B) Use `express.createMiddleware()`
    - C) Extend the `Middleware` class
    - D) Custom middleware is not supported

      Answer: A

      Explanation: Custom middleware is a function that takes `req`, `res`, and `next` as arguments and calls `next()` to continue processing (Express Docs).

15. **What is the purpose of the `cors` middleware in Express?**

    - A) To compress responses
    - B) To enable Cross-Origin Resource Sharing
    - C) To parse JSON bodies
    - D) To serve static files

      Answer: B

      Explanation: The `cors` middleware allows cross-origin requests by setting appropriate headers (CORS Docs).

16. **How do you apply middleware to a specific route in Express?**

    - A) Using `app.use('/path', middleware)`
    - B) Using `app.get('/path', middleware, handler)`
    - C) Both A and B
    - D) Middleware cannot be route-specific

      Answer: C

      Explanation: Middleware can be applied globally with `app.use()` or to specific routes using route methods like `app.get()` (Express Docs).

17. **What happens if you forget to call `next()` in middleware?**

    - A) The request hangs and times out
    - B) The server crashes
    - C) The response is sent automatically
    - D) The next route is called

      Answer: A

      Explanation: Without `next()`, the request pipeline stalls, causing the client to wait until a timeout occurs (Express Docs).

18. **How do you handle errors in middleware?**

    - A) Call `next(err)` with an error object
    - B) Throw an error
    - C) Return an error response directly
    - D) All of the above

      Answer: D

      Explanation: Errors can be passed to error-handling middleware with `next(err)`, thrown, or handled by sending a response directly (Express Docs).

19. **What is the role of `express.static` middleware?**

    - A) To serve static files like images and CSS
    - B) To parse static JSON data
    - C) To handle static routes
    - D) To compress static assets

      Answer: A

      Explanation: `express.static` serves static files from a specified directory, such as `public` (Express Docs).

20. **How can you limit middleware execution to specific HTTP methods?**

    - A) Use `app.get()`, `app.post()`, etc., with middleware
    - B) Check `req.method` inside middleware
    - C) Both A and B
    - D) Middleware applies to all methods

      Answer: C

      Explanation: Middleware can be applied to specific methods using route methods or by checking `req.method` within the middleware (Express Docs).

Topic 13: Express Routing and APIs

21. **What is routing in Express?**

    - A) Mapping HTTP methods and URLs to handler functions
    - B) Managing database queries
    - C) Handling static assets
    - D) Configuring middleware

      Answer: A

      Explanation: Routing defines how an application responds to client requests based on HTTP methods and URLs (Express Docs).

22. **How do you define a route parameter in Express?**

    - A) Using `:param` in the route path
    - B) Using `{param}` in the route path
    - C) Using `?param` in the route path
    - D) Route parameters are not supported

      Answer: A

      Explanation: Route parameters are defined with a colon, e.g., `/users/:id`, and accessed via `req.params` (Express Docs).

23. **How do you access query parameters in an Express route?**

    - A) `req.query`
    - B) `req.params`
    - C) `req.body`
    - D) `req.headers`

      Answer: A

      Explanation: Query parameters (e.g., `?name=value`) are available in `req.query` (Express Docs).

24. **What is the purpose of `Router` in Express?**

    - A) To create modular, mountable route handlers
    - B) To handle database connections
    - C) To parse request bodies
    - D) To serve static files

      Answer: A

      Explanation: `express.Router()` creates modular route handlers that can be mounted on specific paths (Express Docs).

25. **How do you define a RESTful GET endpoint for retrieving all users?**

    - A) `app.get('/users', (req, res) => { ... })`
    - B) `app.post('/users', (req, res) => { ... })`
    - C) `app.get('/users/:id', (req, res) => { ... })`
    - D) `app.all('/users', (req, res) => { ... })`

      Answer: A

      Explanation: A GET request to `/users` is the standard RESTful endpoint for retrieving all users (REST API Best Practices).

26. **Where should you place route definitions in an Express project?**

    - A) In a `routes` folder
    - B) In `app.js`
    - C) In `middleware.js`
    - D) Routes are defined automatically

      Answer: A

      Explanation: Routes are typically organized in a `routes` folder for modularity, with each file handling specific resources (Express Docs).

27. **How do you send a JSON response in Express?**

    - A) `res.json(data)`
    - B) `res.send(data)`
    - C) Both A and B
    - D) `res.write(data)`

      Answer: C

      Explanation: `res.json()` sends a JSON response with the correct content-type, while `res.send()` can also send JSON if passed an object (Express Docs).

28. **What is the purpose of `app.all()` in Express?**

    - A) To handle all HTTP methods for a specific path
    - B) To define all routes in the app
    - C) To register all middleware
    - D) To handle all errors

      Answer: A

      Explanation: `app.all()` defines a handler for all HTTP methods (GET, POST, etc.) on a specific path (Express Docs).

29. **How do you handle a 404 error in Express?**

    - A) Use middleware at the end of the route stack
    - B) Define a route with `app.get('*')`
    - C) Both A and B
    - D) Express handles 404 errors automatically

      Answer: C

      Explanation: A 404 handler can be middleware at the end of the stack or a catch-all route like `app.get('*')` (Express Docs).

30. **How do you validate request data in an Express API?**

    - A) Using libraries like `express-validator`
    - B) Manually checking `req.body`
    - C) Both A and B
    - D) Validation is not needed

      Answer: C

      Explanation: Data can be validated manually or using libraries like `express-validator` for robust validation (Express Validator Docs).

Topic 14: JWT Authentication

31. **What is a JSON Web Token (JWT)?**

    - A) A secure way to transmit information between parties as a JSON object
    - B) A database authentication method
    - C) A session management library
    - D) A front-end authentication library

      Answer: A

      Explanation: JWT is a standard for securely transmitting information, typically used for authentication (JWT Docs).

32. **What are the three parts of a JWT?**

    - A) Header, Payload, Signature
    - B) Token, Secret, Claims
    - C) Header, Body, Footer
    - D) Key, Value, Hash

      Answer: A

      Explanation: A JWT consists of a Header (metadata), Payload (claims), and Signature, separated by dots (JWT Docs).

33. **How do you create a JWT in Node.js?**

    - A) Using the `jsonwebtoken` library
    - B) Using `crypto` module
    - C) Using `express-jwt`
    - D) JWTs are created automatically

      Answer: A

Explanation: The `jsonwebtoken` library is commonly used to sign and verify JWTs in Node.js (jsonwebtoken Docs).

34. **Where should you store a JWT in a React application?**

- A) In `localStorage` or `sessionStorage`
- B) In a cookie with `httpOnly` flag
- C) In Redux state
- D) Both A and B

Answer: D
Explanation: `localStorage` is common but less secure; `httpOnly` cookies are safer to prevent XSS attacks (JWT Best Practices).

35. **How do you verify a JWT in an Express middleware?**

- A) Using `jwt.verify(token, secret)`
- B) Using `express-jwt` middleware
- C) Both A and B
- D) JWTs are verified automatically

Answer: C
Explanation: You can manually verify with `jwt.verify` or use `express-jwt` middleware for automatic verification (jsonwebtoken Docs).

36. **What is the purpose of the `exp` claim in a JWT?**

- A) To set the expiration time
- B) To encrypt the payload
- C) To define the issuer
- D) To store user roles

Answer: A
Explanation: The `exp` claim specifies when the token expires, in Unix timestamp format (JWT Docs).

37. **How do you secure a JWT?**

- A) Use a strong secret and HTTPS
- B) Store it in plain text
- C) Avoid using expiration
- D) Use a short secret key

Answer: A
Explanation: A strong secret, short expiration, and HTTPS ensure JWT security (JWT Best Practices).

38. **What happens if a JWT is intercepted?**

- A) It can be used until it expires
- B) It cannot be used without the secret
- C) It is automatically invalidated
- D) It triggers an error

Answer: A

Explanation: An intercepted JWT can be used until it expires unless invalidated server-side, as it doesn't require the secret for use (JWT Docs).

39. **How do you invalidate a JWT?**

    - A) Maintain a server-side blacklist of tokens
    - B) Change the secret key
    - C) Both A and B
    - D) JWTs cannot be invalidated

    Answer: C
    Explanation: Invalidating JWTs requires blacklisting tokens or rotating the secret key, as JWTs are stateless (JWT Best Practices).

40. **Where should you store the JWT secret in an Express project?**

    - A) In a `.env` file
    - B) In `app.js`
    - C) In `package.json`
    - D) In the client-side code

    Answer: A
    Explanation: The JWT secret should be stored securely in a `.env` file and accessed via `process.env` (Express Best Practices).

Topic 15: Integration of Express and JWT with React/React Native

41. **How do you send a JWT with an API request from React?**

    - A) Include it in the `Authorization` header as `Bearer <token>`
    - B) Send it as a query parameter
    - C) Include it in the request body
    - D) JWTs are sent automatically

    Answer: A
    Explanation: The standard is to send JWTs in the `Authorization` header with the `Bearer` scheme (JWT Best Practices).

42. **How do you make an authenticated API call in React Native?**

    - A) Use `fetch` or `axios` with the `Authorization` header
    - B) Use `Linking` module
    - C) Use `AsyncStorage` to send the token
    - D) Authenticated calls are not supported

    Answer: A
    Explanation: Use `fetch` or `axios` to send requests with the JWT in the `Authorization` header, retrieving the token from storage (React Native Docs).

43. **Where should you store a JWT in a React Native app?**

    - A) In AsyncStorage
    - B) In a secure storage library like `react-native-keychain`
    - C) Both A and B

- D) In the component state
  Answer: C
  Explanation: AsyncStorage is common, but secure storage libraries like `react-native-keychain` are preferred for sensitive data (React Native Docs).

44. **How do you handle JWT expiration in a React app?**

- A) Use refresh tokens to obtain new JWTs
- B) Redirect to login on expiration
- C) Both A and B
- D) JWTs do not expire
  Answer: C
  Explanation: Refresh tokens can extend sessions, but redirecting to login is common when tokens expire (JWT Best Practices).

45. **How do you protect an Express route with JWT authentication?**

- A) Use middleware to verify the JWT
- B) Check `req.body.token` in the route handler
- C) Use `app.protect()`
- D) Routes cannot be protected
  Answer: A
  Explanation: Middleware verifies the JWT in the `Authorization` header before allowing access to the route (jsonwebtoken Docs).

46. **How do you refresh a JWT in an Express API?**

- A) Create an endpoint to issue a new JWT using a refresh token
- B) Automatically refresh JWTs in middleware
- C) JWTs cannot be refreshed
- D) Use `jwt.refresh()`
  Answer: A
  Explanation: A dedicated endpoint validates a refresh token and issues a new JWT (JWT Best Practices).

47. **What is a common folder structure for an Express API integrated with React?**

- A) Separate `client` (React) and `server` (Express) folders
- B) Place all code in a single `src` folder
- C) Use `public` for Express routes
- D) No specific structure is needed
  Answer: A
  Explanation: Separating `client` and `server` folders organizes the frontend and backend code clearly (Express Best Practices).

48. **How do you handle CORS issues when connecting React Native to an Express API?**

- A) Use the `cors` middleware in Express
- B) Set headers manually in each route
- C) Both A and B

- D) CORS is not an issue in React Native
  Answer: C
  Explanation: The `cors` middleware simplifies CORS handling, but manual header settings can also work (CORS Docs).

49. **How do you test an Express API with JWT authentication?**

- A) Use tools like Postman to send requests with JWTs
- B) Write unit tests with `supertest` and mock JWTs
- C) Both A and B
- D) Testing is not needed
  Answer: C
  Explanation: Postman is useful for manual testing, while `supertest` supports automated testing with mocked JWTs (Supertest Docs).

50. **What is a best practice for securing an Express API used with React Native?**

- A) Use HTTPS, validate inputs, and limit JWT expiration
- B) Store JWTs in plain text
- C) Allow unlimited JWT validity
- D) Disable CORS
  Answer: A
  Explanation: Using HTTPS, input validation, and short-lived JWTs enhances API security (Express Best Practices).