ReactJS Practice Test (Sample)

Study Guide

Everything you need from our exam experts!

Table of Contents

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- **Practice answering questions under realistic conditions,**
- **Improve accuracy and speed,**
- **Review explanations to strengthen weak areas, and**
- **Approach the exam with greater confidence.**

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here’s how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don’t just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There’s no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

Questions

1. Which statement about ReactDOM.render is accurate?

- **A. It only updates the virtual DOM and does not touch the real DOM.**
- **B. It compiles JSX to JavaScript at runtime.**
- **C. It renders a React element into the DOM.**
- **D. It is used to define routes in a React app.**

2. What is the purpose of an error boundary in React?

- **A. Catch rendering errors in a component subtree and render fallback UI.**
- **B. Catch only network errors from fetch outside render.**
- **C. Prevent any errors from ever occurring.**
- **D. Replace a component with a generic error component on any error.**

3. What is the difference between props drilling and using contextual data in React?

- **A. Props drilling passes props through intermediate components; Context provides data to nested components without explicit prop passing.**
- **B. Context provides data to nested components only if you pass a prop.**
- **C. Props drilling uses context to avoid passing props.**
- **D. Context eliminates the need for props entirely.**

4. Passing information: pass props from a stateful parent component to a stateless child component.

- **A. Update the child's internal state directly**
- **B. Pass props from the parent to the child component in its render**
- **C. Use the context API to pass information**
- **D. Use a global store to share data**

5. Which directory contains the JavaScript that is processed by webpack and is the heart of the React app?

- **A. src**
- **B. public**
- **C. dist**
- **D. node_modules**

6. Which statement about higher-order components is accurate?

A. A higher-order component is a function that takes a component and returns a new component with enhanced behavior.

B. A higher-order component is a component that renders another component as a child.

C. A higher-order component is a state management library.

D. A higher-order component is a React hook for memoization.

7. What does diffing refer to in React's virtual DOM?

A. Rendering updates directly to the actual DOM.

B. Comparing the new virtual DOM with a pre-update version to identify changes.

C. Serializing JSX to HTML strings.

D. Removing unused DOM nodes from memory.

8. Explain optimistic UI updates and how to implement them with rollback on error.

A. Update the UI immediately and rollback on error if the request fails.

B. Always wait for server response before updating UI.

C. Never rollback changes after a failed request.

D. Optimistic UI is only applicable to read operations.

9. Which statement best describes a JSX element?

A. A basic unit of JSX found in a JavaScript file which is treated as, and has the functionality of, a JavaScript expression.

B. A React component that must be defined using a function or class.

C. A direct DOM node inserted into the page.

D. A CSS class name used in JSX.

10. Which statement about event handlers in class components is true?

A. They are defined as methods on the component class

B. They must be bound in the constructor only

C. They cannot be used in class components

D. They are defined as separate functions outside the class

Answers

1. C
2. A
3. A
4. B
5. A
6. C
7. B
8. A
9. B
10. A

Explanations

1. Which statement about ReactDOM.render is accurate?

A. It only updates the virtual DOM and does not touch the real DOM.

B. It compiles JSX to JavaScript at runtime.

C. It renders a React element into the DOM.

D. It is used to define routes in a React app.

Rendering a React element into the DOM is what ReactDOM.render does. It takes a React element (for example, <App />) and a DOM node (like document.getElementById('root')) and renders the UI into that node. This is the bridge between React's virtual element tree and what you actually see in the browser, so React can update the DOM efficiently when things change. JSX isn't compiled by this function at runtime; JSX is transformed to JavaScript by a tool like Babel during build or at runtime, while ReactDOM.render's role is to mount the prepared element tree into the DOM. It isn't used to define routes; routing is handled by libraries like React Router, which render their own components into the app. For example: ReactDOM.render(<App />, document.getElementById('root')).

2. What is the purpose of an error boundary in React?

A. Catch rendering errors in a component subtree and render fallback UI.

B. Catch only network errors from fetch outside render.

C. Prevent any errors from ever occurring.

D. Replace a component with a generic error component on any error.

An error boundary catches rendering errors within a part of the UI and shows a fallback UI instead of crashing the whole app. When a child component throws during render (or in its lifecycle methods or constructors), the boundary intercepts that error and React renders the boundary's fallback UI, letting the rest of the interface stay interactive. This provides a safer, more resilient user experience by isolating failures to a specific subtree. To implement, you typically create a class component that tracks an error state and uses componentDidCatch (and optionally getDerivedStateFromError) to set the fallback state and perform logging. In render, you show the fallback UI if an error was caught; otherwise you render the normal children. Keep in mind that error boundaries don't catch errors inside event handlers or in asynchronous code outside the render lifecycle, and they don't prevent errors from occurring in the first place. They're a targeted, graceful recovery mechanism for render-time failures in a defined portion of the UI.

3. What is the difference between props drilling and using contextual data in React?

A. Props drilling passes props through intermediate components; Context provides data to nested components without explicit prop passing.

B. Context provides data to nested components only if you pass a prop.

C. Props drilling uses context to avoid passing props.

D. Context eliminates the need for props entirely.

It's about how data moves through a React component tree. Prop drilling happens when you pass a value down through intermediate components that don't use it themselves, just to reach a deeper component that needs it. This can clutter those intermediate components and make changes harder because you have to touch every level of the tree even if most components aren't using the data. Context offers a different approach: you provide a value once with a Provider at some level, and any descendant can access that value directly, without it being passed as a prop through every intermediate component. Components read the value with useContext (or a Consumer) and can react to changes in that context. This pattern is ideal for things many parts of the UI rely on, like theme, locale, or the current user. It's not a universal replacement for props; use it when many components need the data or when prop drilling would become too cumbersome. Just be mindful that changing context can trigger re-renders in all consuming components, so use it judiciously to avoid unnecessary updates.

4. Passing information: pass props from a stateful parent component to a stateless child component.

A. Update the child's internal state directly

B. Pass props from the parent to the child component in its render

C. Use the context API to pass information

D. Use a global store to share data

In React, data flows from a stateful parent to a (stateless) child through props. The parent keeps the data in its state and passes it to the child during render, for example by writing <Child data={this.state.data} />. The child receives that data as props and renders based on it. Props are read-only for the child, so the child shouldn't try to mutate them directly; if the child needs to signal a change, it can call a function passed from the parent (like onSomething) and let the parent update its state, which will re-render the child with the new props. While the context API or a global store can be useful for sharing data across many components or deeper trees, they're not needed for the straightforward case of passing information from a single stateful parent to a stateless child. This direct prop passing is the simplest, idiomatic approach for that relationship.

5. Which directory contains the JavaScript that is processed by webpack and is the heart of the React app?

A. src

B. public

C. dist

D. node_modules

The directory that contains the JavaScript webpack processes and where you write the app's React components is the src directory. This is where the entry point (often src/index.js) and all the component code live, so webpack can bundle it into the final build. The public folder typically holds static assets like index.html and icons and isn't where the app logic resides. The dist folder usually contains the bundled output after webpack runs, not the source code you edit. The node_modules folder stores installed dependencies, not your application code. So, src is the place that best fits the description.

6. Which statement about higher-order components is accurate?

A. A higher-order component is a function that takes a component and returns a new component with enhanced behavior.

B. A higher-order component is a component that renders another component as a child.

C. A higher-order component is a state management library.

D. A higher-order component is a React hook for memoization.

Higher-order components are functions that take a component and return a new component with enhanced behavior. This pattern lets you reuse logic by wrapping the original component and injecting extra props, state, or behavior without modifying the component itself. For example, a withLoading HOC can wrap any component to show a spinner while data loads, then render the wrapped component with the data once ready. This concept is different from a component that simply renders another component as a child, and it's not a state management library or a memoization hook. Those other ideas describe separate patterns or tools, whereas a higher-order component specifically refers to a function that produces a new component by wrapping another one.

7. What does diffing refer to in React's virtual DOM?

A. Rendering updates directly to the actual DOM.

B. Comparing the new virtual DOM with a pre-update version to identify changes.

C. Serializing JSX to HTML strings.

D. Removing unused DOM nodes from memory.

Diffing is the process React uses to reconcile the UI by comparing the updated virtual DOM tree with the previous one to figure out exactly what changed. When state or props update, React builds a new virtual DOM and then walks both trees to identify the minimal set of changes required to bring the real DOM up to date. This selective update is what makes React efficient, preventing unnecessary re-renders and DOM manipulations. The correct description fits this idea: it describes comparing the new virtual DOM with a prior version to identify changes. The other options describe different ideas—updating the real DOM directly would bypass diffing, serializing JSX to HTML strings isn't about computing changes, and removing unused nodes is more about memory cleanup than calculating updates.

8. Explain optimistic UI updates and how to implement them with rollback on error.

A. Update the UI immediately and rollback on error if the request fails.

B. Always wait for server response before updating UI.

C. Never rollback changes after a failed request.

D. Optimistic UI is only applicable to read operations.

Optimistic UI updates mean updating the interface right away to reflect the user's action, assuming the server will succeed, so the app feels fast. The rollback-on-error part comes in as a safety net: you keep a snapshot of what UI looked like before the change, apply the optimistic update, and then perform the server request. If the server says the action failed, you revert to the previous state and show an error, so the UI stays consistent with reality. To implement this, track the previous state before the change. Apply the UI update immediately using the new value, then send the request to the server. If the server responds with success, you can optionally reconcile with any data the server returns (in case there are small adjustments). If the request fails, restore the previous state and notify the user. A practical pattern is: take a copy of the current data, apply the change in the UI, fire the network request, and on error restore the old data. In React terms, you might store the current list, save a backup, set the new list optimistically, and in the catch path revert to the backup. Libraries like React Query or Apollo support this approach with dedicated hooks or options to perform optimistic updates and automatically rollback on error. This approach is especially valuable for write actions like liking a post, adding an item to a cart, or editing a field, where immediate feedback improves perceived performance. It's not about reads, and you should plan for rollback, error messaging, and possible retries to handle failures gracefully.

9. Which statement best describes a JSX element?

A. A basic unit of JSX found in a JavaScript file which is treated as, and has the functionality of, a JavaScript expression.

B. A React component that must be defined using a function or class.

C. A direct DOM node inserted into the page.

D. A CSS class name used in JSX.

JSX elements are the building blocks you use to describe what the UI should look like in React. They aren't actual DOM nodes yet; they're descriptions that React will turn into UI. When you write JSX, it's transformed into JavaScript expressions (usually calls to React.createElement) that produce React elements—plain objects describing what to render. A JSX element can represent either a standard DOM tag like or a custom component like <MyComponent />, so it isn't limited to components alone. The idea is a JSX element is a basic unit in your JSX code that, after compilation, becomes a JavaScript expression that tells React what to render. That means the statement that a JSX element is a React component defined by a function or class isn't accurate, since you can write JSX that renders plain DOM elements as well.

10. Which statement about event handlers in class components is true?

A. They are defined as methods on the component class

B. They must be bound in the constructor only

C. They cannot be used in class components

D. They are defined as separate functions outside the class

Event handlers in class components are defined as methods on the component class. This setup lets the handler access the component's instance data, like this.state and this.props, and call methods to update state in response to user interactions. You typically declare a method inside the class, such as handleClick() { ... }, and attach it in render with onClick={this.handleClick}. Since the function needs the correct this binding, you bind it to the component instance—either in the constructor (this.handleClick = this.handleClick.bind(this);) or by using the class field syntax (handleClick = (e) => { ... }), which auto-binds. Defining handlers outside the class or binding only in the constructor aren't the sole or required approaches, and event handlers can and do work inside class components.

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://reactjs.examzify.com

We wish you the very best on your exam journey. You've got this!