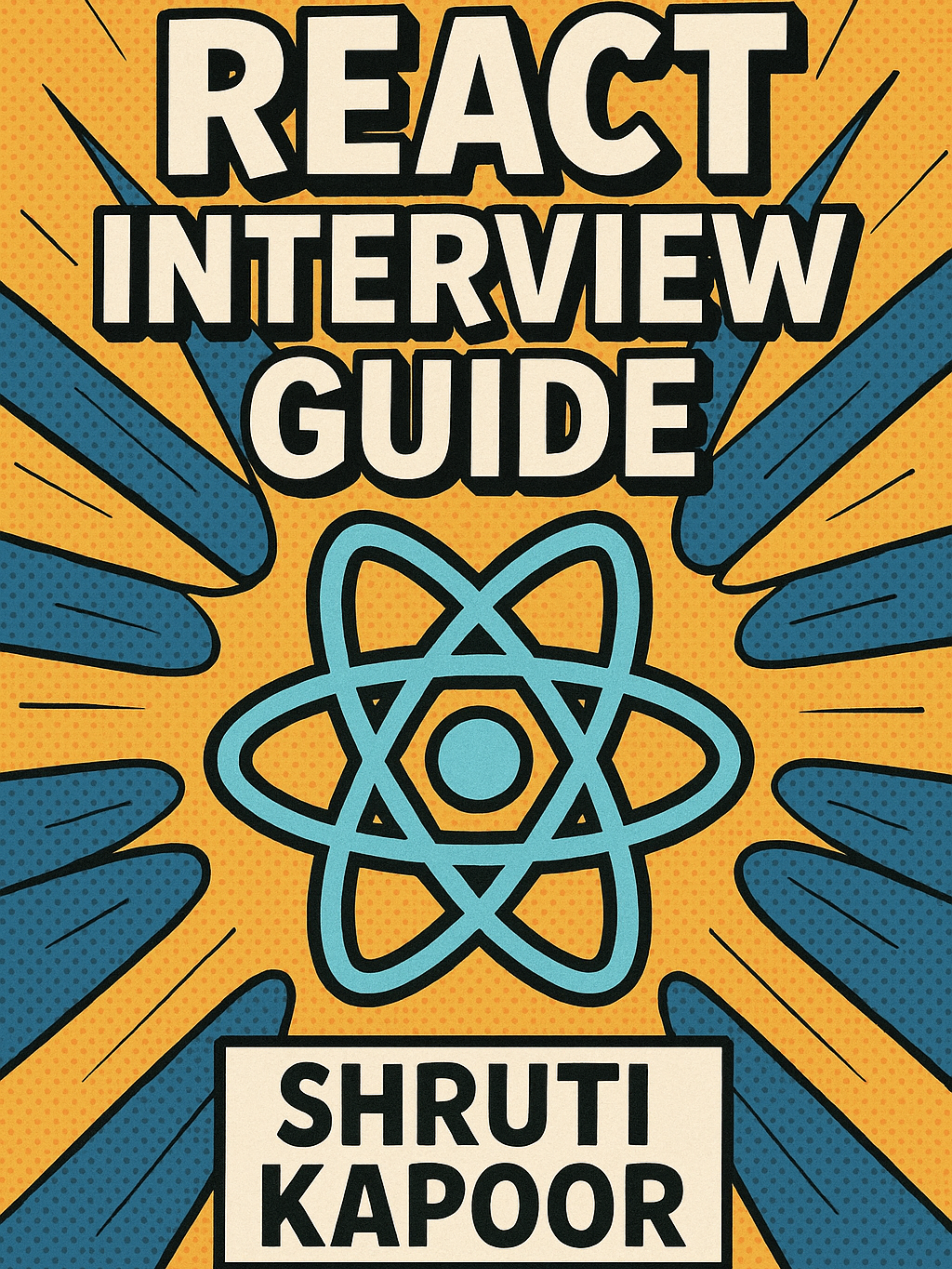
REACT INTERVIEW GUIDE



SHRUTI KAPOOR

TABLE OF CONTENTS

CHAPTER 1: GETTING THE INTERVIEW

POLISH UP RESUME

Your resume is your marketing document and first impression with hiring managers. Make every word count!

1. Create a compelling structure:
    - Start with a brief professional summary highlighting your React expertise
    - List your technical skills with emphasis on React ecosystem technologies (Redux, Next.js, React Native, React Query, React Router, etc.)
    - Organize experience in reverse chronological order.
    - For each role, include 3-5 bullet points highlighting achievements and your specific contributions, not just responsibilities
2. Highlight React-specific achievements and quantify your impact, such as -
    - "Reduced development time by 30% across 5 product teams by developing a React Design System."
    - "Decreased bundle size by 45% through code splitting and lazy loading React components"
    - "Optimized performance by 20% by migrating class components to functional components with React hooks"
    - "Built custom React hooks that standardized data fetching across the platform and reduced technical debt by 70%"

3. Quantify your impact:
   - Use metrics whenever possible: load time improvements, conversion rate increases, bug reduction. People love numbers!
   - Include scale indicators: team size, user base, transaction volume
   - Document performance optimizations with before/after comparisons
4. Customize for each application:
   - Research each company and tailor your resume to match their job description to ensure you are highlighting the tasks that you accomplished.
   - Mirror keywords and tech stack from the job description if they mention specific React versions or tools if you have used them.
   - Highlight relevant projects that showcase skills mentioned in the job listing.
5. Include links to find your work:
   - Include links to your GitHub profile, portfolio, or significant projects
   - List contributions to open-source React projects or libraries
   - Mention specific React patterns you've implemented (compound components, render props, etc.)
6. Get your resume reviewed:
   - Have experienced React developers review for technical accuracy
   - Use peer review services within developer communities
   - Book a time with me to review your resume.

WHEN TO APPLY FOR A JOB?

NOW!

Don’t wait for a “perfect time”. Start applying as soon as possible. Recruiter calls and hiring manager screens can take time. You won’t know what skills you need to improve until you start interviewing. No developer knows it all, even those who have been working for decades in the ecosystem. So go ahead and hit that “Apply” button!

You won’t know what skills you are missing until you start giving your first few interviews. I typically give myself one month to prepare before taking my first coding interview to give myself enough time to prepare. Keep in mind, I code in React every day, and your own timing may differ based on your experience with React.

You wont pass them all, and that’s okay. Interviewing is a numbers game. The more you do, the better you get at it.

Here’s a quick assessment to understand if you are ready to give your interview -

1. You should be comfortable building React applications from scratch

2. You understand core React concepts (components, props, state, lifecycle/hooks)

3. You can describe React's rendering process and optimization techniques

4. You've built at least 2-3 substantial projects with React

HOW TO GET AN INTERVIEW

Networking

Building a professional network is often more effective than applying through job boards. Cultivate quality connections rather than quantity.

LinkedIn Strategy

1. Optimize your LinkedIn profile:
    - Professional photo and React-focused headline
    - Detailed experience section highlighting React projects
    - Skills section with React ecosystem technologies
    - Featured section with links to React projects or articles you've written
    - Request recommendations from colleagues who can speak to your React skills
2. Strategic connection targeting:
    - **Hiring managers**: Look for "Hiring" badges or mentions of open React positions
    - **Recruiters**: Target recruiters who specialize in front-end or React roles.
    - **Engineers**: Reach out to engineers who are working at companies you're interested in, particularly those on Front End teams

3. Ensure your outreach is effective and polite:

- Do not send network connections without a note, they are more likely to be ignored.
- Keep initial messages shorts and be specific with your request, such as ”I'm interested in the Senior React Developer role at [Company] and wondering if

you'd be willing to refer me" instead of asking for a "coffee chat". People like helping but nobody likes wasting their time.

- If someone has mentioned a specific role they are hiring for, reference the posts or job link.
- Include a brief highlight of your relevant experience

4. Follow-up protocol:
    - Wait 5-7 business days before following up once
    - Keep follow-ups brief and respectful of their time
    - Accept non-responses gracefully and move on to other connections

APPLYING TO JOBS

A strategic application process increases your chances of landing interviews for React positions.

1. Job search platforms:
    - **LinkedIn Jobs**: Set up alerts for React positions
    - **Indeed**: Filter for remote or local React opportunities
    - **AngelList/Wellfound**: Startup-focused React positions
    - **Stack Overflow Jobs**: Developer-focused listings
    - **GitHub Jobs**: Often includes React-heavy roles
    - **Company career pages**
    - **Discord Groups:** Communities often have jobs channel where people post open job positions in their company.'

2. Application strategy:
    - Apply to 5-10 jobs per week consistently
    - Focus 60% on roles that match your experience, 30% on stretch roles, and 10% on "dream" positions
    - Track all applications in a spreadsheet with status updates and follow-up dates
    - Set a reminder to check back on older applications after 2 weeks
    - Don't limit yourself by qualifications - apply if you meet 60%+ of requirements

CHAPTER 2: INTERVIEW PREPARATION

RECOMMENDED RESOURCES

1. React fundamentals:
   - Official React documentation: https://react.dev/
   - Epic React by Kent C. Dodds: https://www.epicreact.dev/
   - React O'Reilly Course by Shruti Kapoor (me! 🙋🏻‍♀️) - https://www.oreilly.com/library/view/react-fundamentals-building/0636920981428/
   - React Udemy Course by Maximilian Schwarzmüller - https://www.udemy.com/course/react-the-complete-guide-incl-redux/
   - React for Beginners by Wes Bos: https://reactforbeginners.com/

2. Practice coding questions on interview platforms:
   - Great Front End: React-specific coding challenges that are commonly asked in recent interviews
   - GitHub repository for React questions:
   - LeetCode: For JavaScript algorithm questions (filter by JavaScript)
   - pramp.com - for mock interviews and front end practice

- Glassdoor: Search past interviews of the companies by finding out what companies are asking for.
- System Design by Gaurav Sen on YouTube
- Roadside Coder’s React Coding Playlist

TECHNICAL CONCEPTS TO MASTER

1. React fundamentals:
    - Component lifecycle (class and functional with hooks)
    - State management approaches (useState, useReducer, Context, Redux)
    - Props and prop drilling alternatives
    - Virtual DOM and reconciliation process
    - React's rendering process and optimizations
    - JSX syntax and compilation

2. React hooks:
    - Core hooks:
        - useState
        - useEffect
        - useContext
        - useRef
        - useReducer
        - useMemo
        - useCallback

    - Rules of hooks and common gotchas
    - Custom hook creation and implementation
    - Performance considerations with hooks

3. State management:
    - Local component state vs. global state
    - Context API implementation and limitations
    - Avoiding prop-drilling
    - Managing data flow
4. React performance optimization:
    - Memoization with React.memo, useMemo, and useCallback
    - Code splitting and lazy loading
    - Bundle size optimization
    - Virtualization for long lists
    - Identifying and resolving render bottlenecks
    - Caching
    - Server side rendering & static side generation

If you need help, you can setup time to chat 1:1 - https://bit.ly/shruti-mentorship

CHAPTER 3: MUST-DO CODING QUESTIONS

EVALUATION CRITERIA

Each question is typically evaluated on:

1. **Functionality**: Does it meet all requirements?
2. **Code quality**: Is the code clean, well-organized, and maintainable?
3. **Performance**: How efficiently does it handle large datasets and frequent updates?
4. **Error handling**: How gracefully does it handle edge cases and errors?
5. **Accessibility**: Is the component usable for all users?
6. **State management**: How effectively are the different states managed?

TIME ALLOCATION (FOR INTERVIEW SETTINGS)

- 45-50 minutes for implementation
- 5-10 minutes for discussion and questions

TIPS

1. Talk through your thought process.
2. Listen to hints from your interviewer and lways make sure your interviewer is on the same page while you code.
3. Keep a watch on time.

QUESTION 1: SEARCH INPUT WITH AUTOCOMPLETE

To get the solution, watch this video: https://bit.ly/react-interview-1

You are tasked with building a high-performance search input component with autocomplete functionality for an e-commerce product search. Think Search Bar in Google Search.

Requirements

Core Functionality

1. Create a search input field that displays autocomplete suggestions as the user types
2. Fetch suggestion data from the provided API endpoint
3. Allow users to select suggestions via mouse click or keyboard navigation
4. Handle the submission of the selected item or custom search text

Technical Requirements

1. API Integration with Debounce

- Implement a debounce mechanism to prevent excessive API calls while typing
- Make API calls to a given API such as : https://api.example.com/products/search?query={searchTerm}
- The API returns data in this format: { "results": [{ "id": "1", "name": "Product Name", "category": "Category", "thumbnail": "url/to/image" }, { "id": "2", "name": "Another Product", "category": "Different Category", "thumbnail": "url/to/image" }], "totalResults": 243}

2. Keyboard Navigation

- Implement complete keyboard navigation for accessibility:

    - Up/Down arrows to navigate through suggestions
    - Enter to select the currently highlighted suggestion
    - Escape to close the suggestions dropdown
    - Tab to move focus away from the component
- Visual feedback must indicate the currently selected suggestion
- Maintain scroll position within the dropdown when navigating through many results

3. State Management

- Implement comprehensive state handling for:
    - Loading state while fetching suggestions (display a loading indicator)
    - Error states if the API request fails (show appropriate error message)
    - Empty states when no results are found (display "No results found")
    - Initial state before any searches (can be empty or show popular searches)
- All state transitions should be smooth

4. Performance Optimization

- Implement request cancellation for outdated searches when a new search is initiated
- Implement virtualization for rendering large result sets (100+ items)
- Use proper React memoization techniques to prevent unnecessary re-renders
- Optimize expensive operations to maintain 60fps performance even on mobile devices

QUESTION 2: FORM WITH VALIDATION

You are tasked with creating a multi-step registration form for an application that requires client-side validation of data as the user types and server-side validation once the user has submitted the data.

Requirements

Core Functionality

1. Create a multi-step form with at least three sections:
    - Personal information (name, email, phone)
    - Account security (password, security questions)
    - Financial information (income range, employment status)
2. Implement form progression that prevents users from advancing to the next step until all validations pass
3. Allow users to return to previous steps to edit information
4. Display a summary of all information before final submission

Technical Requirements

Client-Side Validation

- Implement real-time validation as users type with appropriate delays
- Include these validation rules:
    - Email: Valid format with proper domain
    - Password: At least 8 characters, containing uppercase, lowercase, number, and special character

    - Phone: Valid format with country code handling
    - Required field validation for all necessary fields
- Show validation feedback (success/error states) with clear error messages
- Implement field-level, step-level, and form-level validation

Server-Side Validation

- Simulate an API submission with a mock function that randomly succeeds or fails
- Handle server validation errors by highlighting the specific fields that failed validation
- Implement rate limiting to prevent validation endpoint abuse

Form Submission and Error Handling

- Simulate an API submission with a mock function that randomly succeeds or fails
- Implement proper loading states during submission
- Handle network errors gracefully with retry options
- Provide contextual error messages for different types of submission failures
- Implement form persistence to prevent data loss on page refresh
- Create a reconciliation strategy when server-side validation conflicts with client-side validation

QUESTION 3: API INTEGRATION AND DATA FETCHING

You are tasked with building a React application that fetches and displays data from an external API. The application should efficiently handle loading, success, and error states, support data pagination, and implement reusable and optimized data-fetching patterns, including caching strategies, and follow a design spec that is shared by the interviewer.

Requirements

Core Functionality

- Fetch and display a list of items (e.g., products, users, posts) from a REST API.
- Implement loading, success, and error states with UI feedback.
- Support pagination to efficiently handle large datasets.

Technical Requirements

State Management & API Handling

- Parse data and render in the UI according to design spec.
- Manage API requests efficiently.
- Implement a useFetch custom hook for reusable API fetching logic.
- Ensure that API requests are debounced to avoid unnecessary calls (e.g., when searching/filtering).
- Handle authentication headers if needed for API requests.

Loading, Success, and Error States

- Display a loading spinner or skeleton UI while fetching data.

- Show a success state once data is successfully loaded.
- Handle error states by displaying appropriate messages and retry options.

Data Pagination

- Implement infinite scrolling or paginated fetching based on API support.
- Use cursor-based or offset-based pagination depending on the API response format.
- Allow users to navigate between pages with a "Next" and "Previous" button.
- Optimize re-fetching of paginated data to avoid unnecessary API calls.

Caching Strategies

- Cache API responses to reduce redundant network requests.
- Implement stale-while-revalidate logic for keeping data fresh.
- Store cached responses in local storage, IndexedDB, or session storage where appropriate.
- Use background re-fetching to keep data up to date without blocking UI interactions.

QUESTION 4: CAROUSEL/SLIDESHOW COMPONENT

You are tasked with building a responsive and accessible carousel/slideshow component in React. The component should support looping functionality, smooth CSS transitions, and an autoplay feature that allows slides to advance automatically.

Requirements

Core Functionality

- Display a list of images or content slides in a horizontal layout.
- Allow users to navigate slides using previous/next buttons and dot indicators.
- Implement looping functionality so the carousel seamlessly transitions from the last slide back to the first and vice versa.
- Support autoplay mode, where slides advance automatically after a set interval.
- Pause autoplay when the user interacts with the carousel (hover, focus, or manual navigation).
- Ensure the carousel is keyboard-accessible and supports ARIA roles and attributes for accessibility.

Technical Requirements

Slide Transitions & Animation

- Use **CSS transitions** for smooth slide animations.
- Implement fade, slide, or zoom effects as transition options.

Looping Functionality

- When the user reaches the last slide, seamlessly transition back to the first slide.

- Handle edge cases where users navigate quickly to prevent animation glitches.

Autoplay & Controls

- Implement an autoplay feature that advances slides at a configurable interval (e.g., every 3 seconds).
- Allow users to pause autoplay when hovering over or interacting with the carousel.
- Include navigation buttons (previous/next) for manual slide control.
- Add dot indicators to show the current slide and allow users to jump to a specific slide.

Responsiveness & Accessibility

- Make the carousel fully responsive with flexible image scaling.
- Ensure keyboard navigation (arrow keys for previous/next and focus handling).

Error Handling & Edge Cases

- Handle scenarios where the image fails to load (display a fallback UI).

QUESTION 5: IMPLEMENT TIC/TAC/TOE GAME

Technical Requirements

State Management

- Utilize React's hooks for managing game state.
- Maintain state variables for:
  - **Game Board**: Representing the current status of each cell.
  - **Player Turns**: Tracking whose turn it is.
  - **Game Status**: Indicating ongoing, won, lost, or draw states.
  - **Timer and Move Counter**: (Optional) Tracking game duration and number of moves.

Game Logic Implementation

- Implement turn-based logic to alternate player moves.
- Check for winning combinations after each move.
- Detect draw conditions when all cells are filled without a winner.
- Provide a reset option to start a new game.

Responsive UI Design

- Design the game board to be responsive, adjusting to various screen sizes and orientations.
- Use CSS Grid or Flexbox for flexible and adaptive layouts.
- Provide visual feedback for different cell states (e.g., revealed, hidden, flagged).

CONCEPTUAL QUESTIONS TO PRACTICE

1. Explain the virtual DOM and its benefits
2. Describe the component lifecycle in both class and functional components
3. What are the key differences between Redux and Context API?
4. How would you optimize a React application that's rendering slowly?
5. Explain how you would structure a large-scale React application
6. What are controlled vs. uncontrolled components?
7. How does React handle events differently from standard DOM events?
8. Explain the concept of lifting state up
9. What are React portals and when would you use them?
10. Describe how you would test a React component
11. How does React's reconciliation algorithm work?
12. What is the difference between useMemo and useCallback?
13. How would you implement code splitting in a React application?
14. Explain the difference between client-side and server-side rendering in React
15. What are higher-order components (HOCs) and how do they compare to render props?
16. How does React's concurrent rendering improve performance?
17. What are some common pitfalls when using useEffect?
18. How would you manage forms in a React application efficiently?
19. What is suspense in React, and how does it work with data fetching?
20. How does React handle hydration in server-rendered applications?