## TL;DR Quick Start

| Step | Description | Prompt / Script |
|---|---|---|
| 1 | Install Cursor (AI code editor) | Download and install the Cursor app for macOS, Windows, or Linux from the official site. |
| 2 | Open your React project in Cursor | Launch Cursor and open your existing React project folder. |
| 3 | Generate a new component using a prompt | You are my React assistant. Create a <Button> component using TypeScript with props onClick and label. Use CSS Modules. Generate the TSX file and a matching Button.module.css. |
| 4 | Detect potential errors and improvements | Scan this React project for common issues like missing keys, incorrect dependency arrays, unused props, and basic performance problems. List issues with file paths and suggested fixes. |

---

# How to Set Up the Cursor for a React Project

## Step 1 – Installation

Download and install the Cursor IDE for your operating system. Optionally, install the Cursor CLI/Agent if you want to trigger prompts from the terminal.

Verify CLI installation:

| Step | Prompt / Script |
|---|---|
| Verify CLI | cursor --version |

**Mini-checklist:** - Install the Cursor desktop app
- (Optional) Install Cursor CLI / Agent
- Verify CLI availability

---

## Step 2 – Open Your React Project

Open your React project folder in Cursor. The editor reads your project structure for context-aware suggestions.

**Mini-checklist:** - Launch Cursor
- Open the React project folder
- Confirm files load correctly

---

## Step 3 – Initial Configuration

| Task | Details |
|---|---|
| Component structure | Functional/class-based |
| File naming | Custom conventions |
| Styling | CSS Modules, Styled Components, Tailwind CSS |
| Linting/formatting | Enable auto-suggestions |
| Review | Confirm configuration before running commands |

---

# Generate React Components with Cursor

## 1. Component Scaffolding

| Step | Prompt / Script |
|---|---|
| Scaffold new component | Create a NavBar React component in TypeScript with props links and activeIndex. Use CSS Modules for styling. Generate TSX file and matching CSS module. Ensure accessibility features like keyboard navigation. |

**Mini-checklist:** - Decide on component name and type
- Define props and default states
- Choose styling method
- Review auto-generated code

---

## 2. Hook Generation

| Step | Prompt / Script |
|---|---|
| Generate custom hook | Generate a custom React hook called useFormValidation for validating email and password. Include state management, validation logic, and error handling. Return both values and errors. |

**Checklist:** - Identify hook purpose
- Define input parameters and return values

- Generate hook using Cursor prompt
- Integrate and test

---

## 3. Template Management

| Step | Prompt / Script |
|---|---|
| Save reusable template | Treat this button component as a reusable template. Apply the same structure and styling when generating new button components. |
| Update template | Update this template to include dark mode support. |

**Mini-checklist:** - Identify reusable components/hooks
- Save templates with descriptive names
- Customize templates
- Use prompts for modifications

---

# Debugging and Refactoring with Cursor

## 1. Error Detection

| Step | Prompt / Script |
|---|---|
| Detect common issues | Scan src/components and src/hooks for common React issues like missing keys, incorrect dependency arrays, unused props, and prop drilling. List issues with file paths and suggested fixes. |
| Suggest prop fixes | Suggest prop type fixes for all functional components. |

**Workflow:** - Select folders/files
- Run prompt
- Review issues and apply fixes
- Re-run prompt to confirm

---

## 2. Performance Optimization

| Step | Prompt / Script |
|---|---|
| Analyze component | Review ComponentName.tsx and identify potential causes of unnecessary re-renders. Suggest optimizations such as memoization, component splitting, or caching. |

**Checklist:** - Detect slow-rendering components
- Apply React.memo where appropriate
- Split large components
- Re-test performance

---

### 3. Code Refactoring Prompts

| Step | Prompt / Script |
|---|---|
| Refactor component | Refactor this component to improve readability and reduce complexity without changing behavior. |
| Refactor hook | Refactor this hook to reduce unnecessary state updates and improve clarity. |

**Workflow:** - Select target component/hook
- Prompt Cursor with goal
- Compare before/after
- Test functionality

---

## Productivity Impact Table

| Metrics | Example | Time Saved | Errors Reduced |
|---|---|---|---|
| Component scaffolding | NavBar, Button | 40% | 0–1 per file |
| Hook generation | useFormValidation | 35% | 0 |
| Debugging suggestions | Error detection prompts | 25% | 2–3 per file |

---

## Workflow Tips & Best Practices

### Mini-Checklist

- Define templates for commonly used components
- Keep Cursor prompts specific
- Review auto-generated code for edge cases
- Document recurring patterns

### Version Control

- Commit small increments
- Tag reusable templates

• Separate experimental changes in feature branches
• Include prompts in commit messages

## Team Collaboration Prompts

| Prompt Examples |
|---|
| Generate accessible modal component with keyboard navigation. |
| Create reusable form hook with validation for email and password. |
| Refactor button components to follow new CSS module conventions. |

**Checklist:** - Save team-specific prompt templates
- Share runbooks and guides
- Review and approve generated code collectively