Coding & Development / JavaScript

Create React functional components with props, state, hooks, and JSX based on a described UI element.
Difficulty: Intermediate → Advanced
Model: GPT-4 / Claude / Gemini
Use Case: React Development, UI Components, Frontend Development
Updated: May 2026
Why This Prompt Exists
Most React components are written from scratch — repetitive boilerplate for every UI element.

You get:

- components missing PropTypes or TypeScript types
- no default props (undefined errors)
- poor state management (unnecessary re-renders)
- missing useEffect cleanup (memory leaks)
- inconsistent styling patterns

But a React component is not just JSX.

It is a reusable unit with props, state, and lifecycle management.

- Props: inputs to the component with types and defaults
- State: internal data with proper setters
- Hooks: useEffect, useCallback, useMemo for optimization
- Event handlers: functions for user interaction
- Conditional rendering: loading, error, empty states

Without structure, components become buggy and hard to maintain.

This framework forces AI to generate production-ready React components.

The Prompt

```
Assume the role of a senior React developer who writes clean,
reusable, performant components.

Your task is to generate a React functional component.

Generate:

1. IMPORTS
   - React, useState, useEffect, etc.
   - CSS module or styled-components (if needed)

2. PROPS (with PropTypes or TypeScript)
   - Props interface/types
   - Default props

3. COMPONENT BODY
   - State declarations (useState)
   - Side effects (useEffect with cleanup)
   - Event handlers
   - Derived state (useMemo, useCallback)

4. JSX RETURN
   - Conditional rendering
   - Loading/error/empty states
```

   - Main UI structure

5. STYLES
   - CSS module or inline styles

6. EXPORT DEFAULT

INPUTS:

Component Name:
[INSERT]

Component Description (what it does):
[DESCRIBE]

Props (name, type, required?, default):
[LIST]

State (name, type, initial value):
[LIST]

API Calls or Side Effects (if any):
[DESCRIBE]

Styling Approach:
[CSS MODULES / STYLED-COMPONENTS / TAILWIND / INLINE]

RULES:
- Use functional components with hooks (not classes)

```
- Destructure props in function signature
- Add PropTypes or TypeScript interfaces
- Include cleanup in useEffect (return function)
- Use useCallback for functions passed to child components
- Use useMemo for expensive calculations
- Handle loading and error states for async operations
- Make components reusable (don't hardcode business logic)
```

How To Use It

- Describe the component’s purpose clearly — what does it show or do?
- List all props with types and defaults (required vs optional).
- If the component fetches data, describe the API call.
- Test the generated component with different prop combinations.
- Add accessibility attributes (aria) after generation.

Example Input

Component Name: UserProfileCard

Component Description: Displays user information (name, avatar, bio, follower count) with an edit button that opens a modal.

Props: userId (string, required), isEditable (boolean, default: false), onEdit (function, optional)

State: user (object, null), loading (boolean, true), error (string, null), modalOpen (boolean, false)

API Calls or Side Effects: Fetch user data from /api/users/{userId} when component mounts or userId changes

Styling Approach: CSS MODULES

Why It Works

Most React components are written from scratch.

This framework improves outcomes by forcing:

- props with types (type safety)
- state management (correct hooks)
- effect cleanup (memory leak prevention)
- conditional rendering (UX)
- loading/error states (robustness)

Great React components don’t just render — they handle loading, errors, and cleanup gracefully.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, JavaScript frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The Node.js Endpoint Generator