1. Identified legacy function/class

import React, { Component } from 'react';

import PropTypes from 'prop-types';

class LegacyCounter extends Component {

constructor(props) {

super(props);

this.state = {

count: props.initialCount

};

this.increment = this.increment.bind(this);

}

componentDidMount() {

// Legacy: side effects here

console.log('Component mounted');

}

componentWillReceiveProps(nextProps) {

// Legacy: update state based on new props

if (nextProps.initialCount !== this.props.initialCount) {

this.setState({ count: nextProps.initialCount });

}

}

increment() {

this.setState({ count: this.state.count + 1 });

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

LegacyCounter.propTypes = {

initialCount: PropTypes.number

};

LegacyCounter.defaultProps = {

initialCount: 0

};

export default LegacyCounter;

2. Use AI to analyze and generate test plan

Prompt:

Here in this project I'm having legacy code. Also can you generate test plan for this legacy code project?

For each section, analyze and provide:

1. Risk Assessment (High/Medium/Low risk if this breaks)

2. Test Complexity (Easy/Medium/Hard to test)

3. Business Impact (Critical/Important/Nice-to-have)

4. Recommended Testing Strategy

5. Priority Score (1-10)

Output:

| File | Risk | Test Complexity | Business Impact | Testing Strategy | Priority |
| --- | --- | --- | --- | --- | --- |
| LegacyCounter.jsx | Medium | Medium | Important | Unit, snapshot, lifecycle tests | 7 |
| index.js | Low | Easy | Critical | Smoke/integration test | 8 |

1. Legacy Code Example

* You have a React project using class components and old lifecycle methods (e.g., componentWillReceiveProps), which are considered legacy patterns.

2. Risk & Test Analysis

1. LegacyCounter.jsx:

* Risk: Medium
* Test Complexity: Medium
* Business Impact: Important
* Priority: 7/10
* Recommended: Unit, snapshot, and lifecycle tests.

1. index.js:

* Risk: Low
* Test Complexity: Easy
* Business Impact: Critical
* Priority: 8/10
* Recommended: Smoke/integration test.

3. Test Plan

* Objectives: Ensure legacy components and entry points work as expected.
* Scope: All React components and entry points.
* Test Types: Unit, integration, smoke, and snapshot tests.
* Key Test Cases: Rendering, state management, lifecycle methods, prop updates, and app bootstrapping.
* Tools: Jest, React Testing Library, or Enzyme.
* Environment: Node.js, npm/yarn, browser for manual checks.
* Risk Areas: Deprecated lifecycle methods and app entry point.
* Regression: Run all tests after changes or updates.

3. Create wrapper or facade if needed

Prompt:

Take this legacy project and create a testable wrapper version that:

- Accepts dependencies as parameters

- Maintains backward compatibility

- Allows easy mocking

- Preserves existing functionality

Output:

import React, { Component } from 'react';

import logo from './logo.svg';

import './App.css';

import LegacyCounter from './component/LegacyCounter';

class App extends Component {

render() {

const { Counter = LegacyCounter } = this.props;

return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<p>

Edit <code>src/App.js</code> and save to reload.

</p>

<a

className="App-link"

href="https://reactjs.org"

target="\_blank"

rel="noopener noreferrer"

>

Learn React

</a>

</header>

<Counter initialCount={0} onIncrement={(count) => console.log('Count:', count)} />

</div>

);

}

}

export default App;

4. Generate comprehensive test suite

Output:

import React from 'react';

import { render, screen } from '@testing-library/react';

import App from './App';

// Mock the LegacyCounter component

jest.mock('./component/LegacyCounter', () => {

return function MockLegacyCounter({ initialCount, onIncrement }) {

return (

<div data-testid="mock-counter">

Mock Counter: {initialCount}

<button onClick={() => onIncrement(initialCount + 1)}>Mock Increment</button>

</div>

);

};

});

describe('App Component', () => {

test('renders without crashing', () => {

render(<App />);

expect(screen.getByText(/Edit/i)).toBeInTheDocument();

});

test('renders LegacyCounter with default props', () => {

render(<App />);

expect(screen.getByTestId('mock-counter')).toBeInTheDocument();

expect(screen.getByText(/Mock Counter: 0/i)).toBeInTheDocument();

});

test('passes correct props to LegacyCounter', () => {

const consoleSpy = jest.spyOn(console, 'log').mockImplementation();

render(<App />);

screen.getByText('Mock Increment').click();

expect(consoleSpy).toHaveBeenCalledWith('Count:', 1);

consoleSpy.mockRestore();

});

test('renders with custom Counter component', () => {

const CustomCounter = () => <div data-testid="custom-counter">Custom Counter</div>;

render(<App Counter={CustomCounter} />);

expect(screen.getByTestId('custom-counter')).toBeInTheDocument();

});

});

5. Refactor original code safely

Output:

import React, { useState, useEffect } from 'react';

import logo from './logo.svg';

import './App.css';

import LegacyCounter from './component/LegacyCounter';

function App({ Counter = LegacyCounter }) {

const [count, setCount] = useState(0);

useEffect(() => {

console.log('App mounted');

}, []);

const handleIncrement = (newCount) => {

console.log('Count:', newCount);

setCount(newCount);

};

return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<p>

Edit <code>src/App.js</code> and save to reload.

</p>

<a

className="App-link"

href="https://reactjs.org"

target="\_blank"

rel="noopener noreferrer"

>

Learn React

</a>

</header>

<Counter initialCount={count} onIncrement={handleIncrement} />

</div>

);

}

export default App;