React Getting Started – Intern Onboarding

Welcome to **Safe Security** 👋

This document is a practical guide to help new interns get started with **React**. It covers the fundamentals, best practices we follow as a team, and how to write clean, maintainable React code.

---

1. What is React?

React is a **JavaScript library** for building **user interfaces**, especially single-page applications. It allows you to create reusable UI components and manage application state efficiently.

Why we use React at Safe Security: - Component-based architecture - Predictable UI updates - Strong ecosystem and tooling - Easy collaboration across teams

---

2. Prerequisites

Before diving into React, you should be comfortable with:

- **JavaScript (ES6+)**
- `let` / `const`
- Arrow functions
- Destructuring
- Spread operator
- Array methods (`map`, `filter`, `reduce`)
- **HTML & CSS basics**
- **Git & GitHub** (basic commands and PR workflow)
- **Node.js & npm/yarn** (package management)

---

3. Setting Up a React Project

We typically use modern tooling that is already configured in our repositories. However, you should know the basics:

- React projects are usually bootstrapped using tools like **Vite** or **Create React App**
- Dependencies are managed via `package.json`
- Development server runs locally and supports hot reload

Important folders you'll commonly see: - `src/` → Application code - `components/` → Reusable UI components - `pages/` or `views/` → Page-level components - `utils/` → Helper functions - `hooks/` → Custom React hooks

---

4. Core Concepts You Must Know

4.1 Components

React applications are built using **components**.

- Components are reusable pieces of UI
- Written as **functions** (we prefer functional components)
- Must return JSX

Rules: - One component = one responsibility - Keep components small and focused

---

4.2 JSX

JSX allows you to write HTML-like syntax inside JavaScript.

Key things to remember: - Use `className` instead of `class` - JavaScript expressions go inside `{}` - Components must return a **single parent element**

4.3 Props

Props are how data is passed **from parent to child components**.

Best practices: - Treat props as **read-only** - Use meaningful prop names - Prefer passing primitives or objects instead of multiple flags

---

4.4 State

State represents **data that can change over time**.

- Managed using React hooks
- Updating state causes re-render

Guidelines: - Keep state as minimal as possible - Don’t duplicate derived data in state - Lift state up when multiple components need it

---

4.5 Events & Handlers

React uses camelCase event handlers: - `onClick` - `onChange` - `onSubmit`

Best practices: - Avoid inline complex logic - Prefer named handler functions

---

5. React Hooks (Very Important)

Hooks let you use React features without writing classes.

5.1 `useState`

- Used for local component state
- State updates are asynchronous

5.2 `useEffect`

- Used for side effects (API calls, subscriptions)
- Runs after render

Common use cases: - Fetching data - Listening to events - Cleaning up resources

Important: Always understand the dependency array.

5.3 Custom Hooks

- Reusable logic extracted into functions
- Must start with `use`
- Helps keep components clean

---

6. Conditional Rendering

Render UI based on conditions: - Ternary operators - Logical AND (`&&`)

Use cases: - Loading states - Empty states - Permission-based UI

---

7. Lists & Keys

When rendering lists: - Always use a **unique and stable** `key` - Avoid using array index as key unless necessary

---

8. Folder & Code Organization

We strongly value **readable and scalable code**.

Guidelines: - One component per file - Name files same as component - Keep logic separate from UI where possible - Avoid deeply nested components

9. Styling Guidelines

Depending on the project, we may use: - CSS / SCSS - CSS Modules - Styled components - Design system components

Best practices: - Avoid inline styles unless necessary - Follow design system conventions - Reuse shared styles

---

10. Performance Basics

Things to keep in mind: - Avoid unnecessary re-renders - Memoize expensive computations when needed - Keep component tree shallow

Don’t optimize prematurely—write clean code first.

---

11. Common Mistakes to Avoid

- Mutating state directly
- Overusing `useEffect`
- Large components doing too many things
- Ignoring lint warnings
- Writing business logic inside JSX

---

12. Code Quality Expectations at Safe Security

We expect: - Clean, readable, and well-structured code - Meaningful variable and function names - Proper error handling - Thoughtful PR descriptions

Before raising a PR: - Test your changes - Remove console logs - Ensure no regressions

---

13. How to Learn Faster

Recommended approach: - Read existing codebase - Ask questions early - Break tasks into small steps - Review PR feedback carefully

---

14. Helpful Resources

- React Official Docs

- MDN JavaScript Docs
- Internal Safe Security components & design system

---

15. Final Notes

React is best learned by **building and reading code**. Don’t worry about knowing everything on day one. Focus on fundamentals, write clean code, and keep improving.

Welcome aboard, and happy coding 😥