assignment # check-off # front-end # react

Check-Off 6 — React Tutorial

School of Engineering and Technology, University of Washington Tacoma
TCSS 460 — Client/Server Programming, Spring 2026

Due Date

Sunday, May 10, 2026, 11:59 PM

Description

This is your introduction to React — the component-based front-end framework you will use for the rest of the quarter. You will work through three pages of the official React documentation: the **Quick Start**, the **Tic-Tac-Toe Tutorial**, and **Thinking in React**. Along the way you will build a working tic-tac-toe game with move history (time travel), then ship three of the five "Try Out Some Ideas" challenges from the end of the tutorial. The check-off is less about typing code than about being able to point at a piece of UI and explain — in React's vocabulary — what is a component, what is a prop, what is state, where state lives, and why.

Learning Objectives

By completing this check-off, you will:

- Identify components, props, state, and event handlers in a small React app
- Explain why state is **lifted up** to the closest common parent and how children update parent state via callback props
- Apply the **immutability** rule when updating arrays/objects in state (`setSquares(nextSquares)`, not `squares[i] = ...`)
- Use **time travel** as a concrete demonstration that a UI is a function of state
- Read a UI mockup, break it into a component tree, and identify the minimal set of state needed to render it (the *Thinking in React* process)
- Extend an existing React app by adding a feature without breaking the rest

Course Learning Objectives

This check-off contributes to the following course learning objectives:

- **LO 4:** Build interactive front-end applications using a component-based framework (e.g., React/Next.js)
- **LO 6:** Transfer object-oriented programming skills to a new language and ecosystem (e.g., Java to TypeScript)

It also supports these course outcomes:

- **Inquiry and Critical Thinking** — reading a UI and reasoning backward to its component tree and state
- **Communication/Self-Expression** — explaining your component design to your checker in React's vocabulary, not just "this part of the page"

Before You Begin

Ensure you have:

- A modern browser (Chrome, Firefox, Edge, or Safari)
- A free CodeSandbox account — recommended; the React docs tutorial is built around the CodeSandbox embed
- *(Optional)* Node.js 22+ if you want to run the tutorial locally instead of in CodeSandbox

Guides for This Check-Off

- React Fundamentals — components, JSX, props, state, hooks
- HTML, CSS, and the DOM — what React is rendering *to* and why
- The official React docs are the primary reference for this check-off

---

Project Setup

Requirement 1: Pick a Workspace

Choose **one** of the following two options. The work and the checklist are identical either way — the checker only cares that they can see your running game and the source code.

Option A — CodeSandbox (recommended)

The React tutorial is written around CodeSandbox. Every code block has a "Fork" button that drops you into a working sandbox with the right starter files. This is the path of least resistance — no installs, no toolchain.

1. Open the Tic-Tac-Toe tutorial in your browser.
2. Scroll to the "Setup for the tutorial" section and click **Fork** on the starter sandbox.
3. Sign in to CodeSandbox if prompted — this saves the sandbox to your account so you can come back to it.
4. Bookmark the sandbox URL. You will share this URL with your checker.

Option B — Local React project

If you prefer to work in your editor, follow react.dev's own Installation guide. Pick the "Try React locally" path — it walks you through creating a minimal React project with a build tool. **Do not use Next.js for this check-off** — Next is a full framework with its own routing and rendering rules that we will introduce in a later week. For now, plain React is the right scope.

When your project is running, open the dev server URL in your browser and confirm you see the default page. You will copy the tutorial's starter code into `App.js` (or `App.tsx`) to begin.

Requirements

Requirement 2: Read the Quick Start

> **Reference**
>
> react.dev/learn — the Quick Start page

Read the entire **Quick Start** page in the React docs (it is a single page, ~10–15 minutes). You do not need to type any code — this is a reading pass to build vocabulary before the tutorial.

By the end you should be able to point at a JSX snippet and identify:

- A **component** (a function that returns JSX)
- **Props** being passed in
- **State** declared with `useState`
- An **event handler** (`onClick`, `onChange`, …)
- Conditional rendering (`{condition && <X />}` or ternary) and list rendering (`.map(...)`)

Requirement 3: Complete the Tic-Tac-Toe Tutorial

Reference

react.dev/learn/tutorial-tic-tac-toe

Work through the tutorial **end-to-end**, including the "**Adding Time Travel**" section at the end. Do not skip ahead — the tutorial is built so each step earns the next concept (lifting state up, immutability, key-based list rendering, derived state from history).

When you finish you should have a working game that:

- Lets two players (X and O) take turns clicking squares
- Declares a winner when one is found, and prevents further moves
- Shows a numbered list of past moves you can click to **jump back in time** to any earlier board state

Stuck on a step?

Every code block in the tutorial has a sandbox you can fork to see the working state at that point. If your code diverges from the tutorial's, fork the next block's starting sandbox and continue from there — you will not lose credit for resetting.

Requirement 4: Read *Thinking in React*

Reference

react.dev/learn/thinking-in-react

Read **Thinking in React** in full. This page does not introduce new syntax — it teaches the **process** of going from a static mockup to a working React app, in five steps:

1. Break the UI into a component hierarchy
2. Build a static version in React
3. Find the minimal but complete representation of UI state
4. Identify where your state should live
5. Add inverse data flow

You do not have to type the example. You **do** need to be able to summarize the five steps in your own words during the check-off.

Requirement 5: Ship Three of the Five Challenges

At the end of the tic-tac-toe tutorial there is a section titled "**Try out some ideas**" with five challenges, listed in order of increasing difficulty. Pick **any three** and ship them in the same sandbox/project as your finished tutorial:

1. For the current move only, show "**You are at move #…**" instead of a button.
2. Rewrite `Board` to use **two loops** to render the squares instead of hardcoding them.
3. Add a **toggle button** that lets the user sort the move list in ascending or descending order.
4. When someone wins, **highlight the three winning squares**. When no one wins and the board is full, display a **draw** message.
5. Display the **location of each move** in the move history in the format `(row, col)`.

For each challenge you ship, be ready to:

- Demo it working in the running game.
- Point at the lines you changed and explain *why* the change goes there. (Which component owns the new state? Which prop did you have to add? Did you have to lift any state up?)

> **Pick a mix of difficulties**
>
> The challenges roughly grow harder going down the list. Picking #1, #3, and #4 (or similar) gives you a richer story to tell your checker than three easy ones.

Requirement 6: Be Ready to Explain the Core Concepts

The point of the check-off is the conversation, not the keystrokes. Before you find a checker, make sure you can answer each of the following in your own words while pointing at your code:

- **Component:** Which functions in your file are React components? How can you tell?
- **Props:** What does the `Square` component receive from its parent, and how is that different from how `Board` was hardcoded at the start of the tutorial?

- **State:** Where is the `squares` array declared? Why there, and not inside `Square`? (Your answer is the *lifting state up* answer.)
- **Immutability:** When a square is clicked, why do you write `const nextSquares = squares.slice(); nextSquares[i] = …; setSquares(nextSquares)` instead of `squares[i] = …; setSquares(squares)`?
- **Derived UI from state:** How does the move list re-render when you click "Go to move #3"? Where does the displayed board come from when you do that?

Peer Check-Off

Find a classmate to verify your work.

> **For the Checker**
>
> The checker has 10 points to award. There is no rubric — use your judgment. Did they do the work? Can they explain it? Award points based on completeness and understanding — not perfection.

Check-Off Checklist

#	Verify	✓
1	Student opens their project (CodeSandbox URL or local dev server) and the tic-tac-toe game renders in a browser	☐
2	Two players can take turns; the status line correctly shows whose turn it is and announces the winner when there is one	☐
3	The move history list is present, and clicking an earlier move jumps the board back to that state (time travel works)	☐
4	Student can point at a component in the source and identify it as a component (function returning JSX)	☐
5	Student can show where `squares` state is declared, and explain in their own words why it lives in `Board` (or `Game`) and not in `Square`	☐

#	Verify	✓
6	Student can explain the **immutability** rule: why they `.slice()` the array before mutating, and what would go wrong if they didn't	☐
7	Student can summarize the five steps of *Thinking in React* in their own words (component hierarchy → static version → minimal state → where state lives → inverse data flow)	☐
8	**Challenge #1 of 3** is implemented and demoed; student can point at the lines they changed and explain why	☐
9	**Challenge #2 of 3** is implemented and demoed; student can point at the lines they changed and explain why	☐
10	**Challenge #3 of 3** is implemented and demoed; student can point at the lines they changed and explain why	☐
11	Student can name at least one piece of state that they had to add or move in order to ship one of their challenges (or, if no new state was needed, can explain why the existing state was enough)	☐

How to Submit

1. Complete the check-off requirements above.
2. Find a classmate to be your checker.
3. Demo your work — walk them through each item on the checklist.
4. The checker has 10 points to award. There is no rubric — use your judgment. Did they do the work? Can they explain it? Award points based on completeness and understanding — not perfection.
5. Both of you fill in the shared spreadsheet linked from the Canvas assignment: the student enters their name in Column A, the checker enters their name in Column B, and the checker enters the score in Column C.

Canvas — Check-Off 6: React Tutorial

Guide Reference

Guide / Resource	What It Covers
React Quick Start	Components, JSX, props, state, events, conditional and list rendering
Tic-Tac-Toe Tutorial	The hands-on build — lifting state up, immutability, time travel
Thinking in React	The five-step process from mockup to working React app
React Fundamentals (course guide)	Course-side summary of the same concepts
HTML, CSS, and the DOM	What React renders to, and why a virtual DOM exists

---

Gen AI & Learning: Using AI for This Check-Off

You are welcome to use AI coding assistants to help you debug or to ask "why did the tutorial do it this way?" — that is a great use of an AI tutor. You should **not** paste the challenge prompts into an AI and submit its answer. Your checker will ask you to point at the lines you changed and explain *why* — if you cannot, the check-off is not complete. The point of this check-off is the conversation, and the conversation only works if you wrote (or fully understand) the code.

This assignment is part of TCSS 460 — Client/Server Programming, School of Engineering and Technology, University of Washington Tacoma.