Overview

What are you building?

Setup for the tutorial

Overview

Inspecting the starter code

Building the board

Passing data through props

Making an interactive component

React Developer Tools

Completing the game

Lifting state up

Why immutability is important

Taking turns

Declaring a winner

Adding time travel

Storing a history of moves

Lifting state up, again

Showing the past moves

Picking a key

Implementing time travel

Final cleanup

Wrapping up

Tutorial: Tic-Tac-Toe

You will build a small tic-tac-toe game during this tutorial. This tutorial does not assume any existing React knowledge. The techniques you'll learn in the tutorial are fundamental to building any React app, and fully understanding it will give you a deep understanding of React.

Note

This tutorial is designed for people who prefer to **learn by doing** and want to quickly try making something tangible. If you prefer learning each concept step by step, start with Describing the UI.

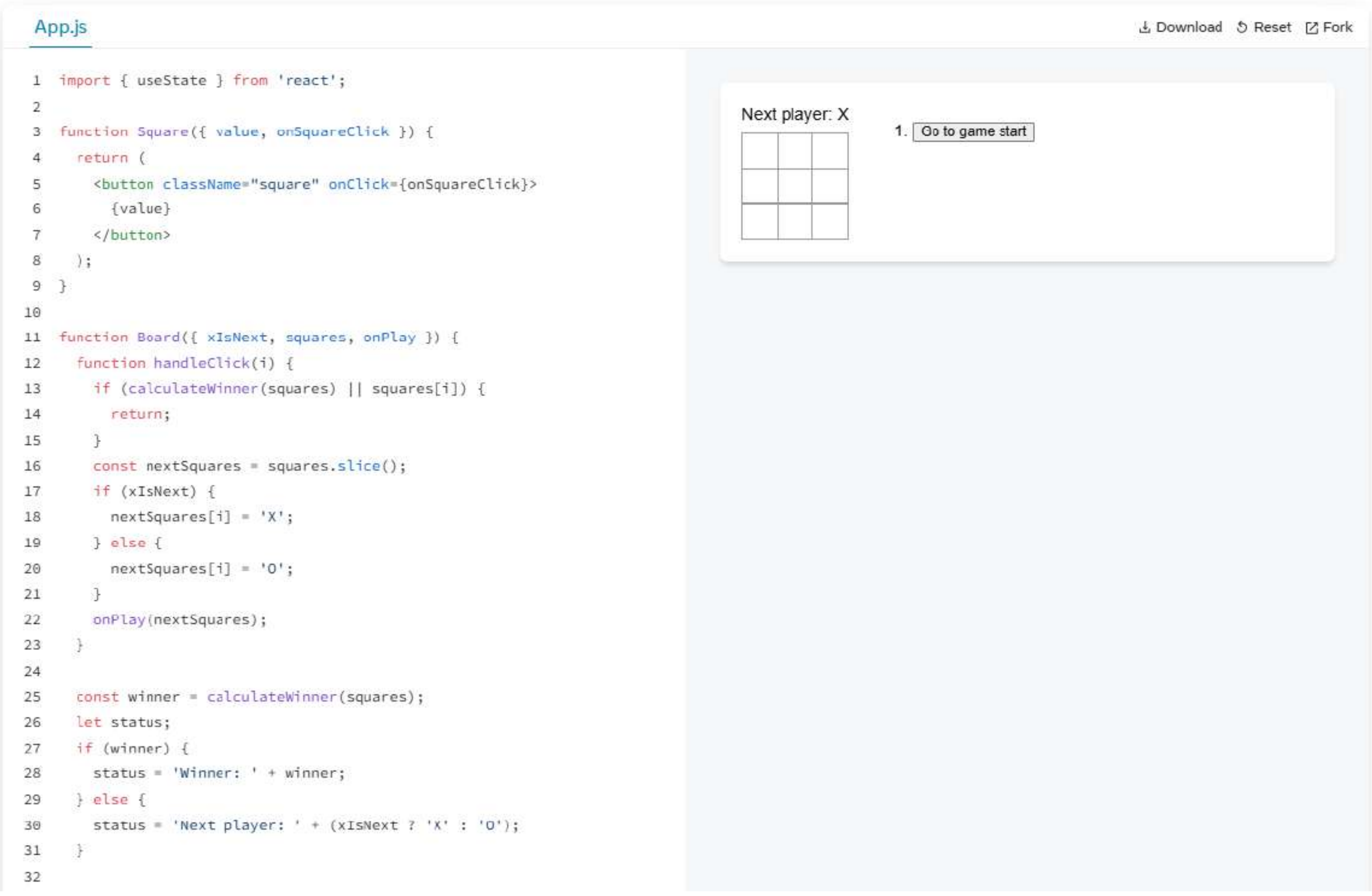
The tutorial is divided into several sections:

- Setup for the tutorial will give you **a starting point** to follow the tutorial.
- Overview will teach you **the fundamentals** of React: components, props, and state.
- Completing the game will teach you **the most common techniques** in React development.
- Adding time travel will give you **a deeper insight** into the unique strengths of React.

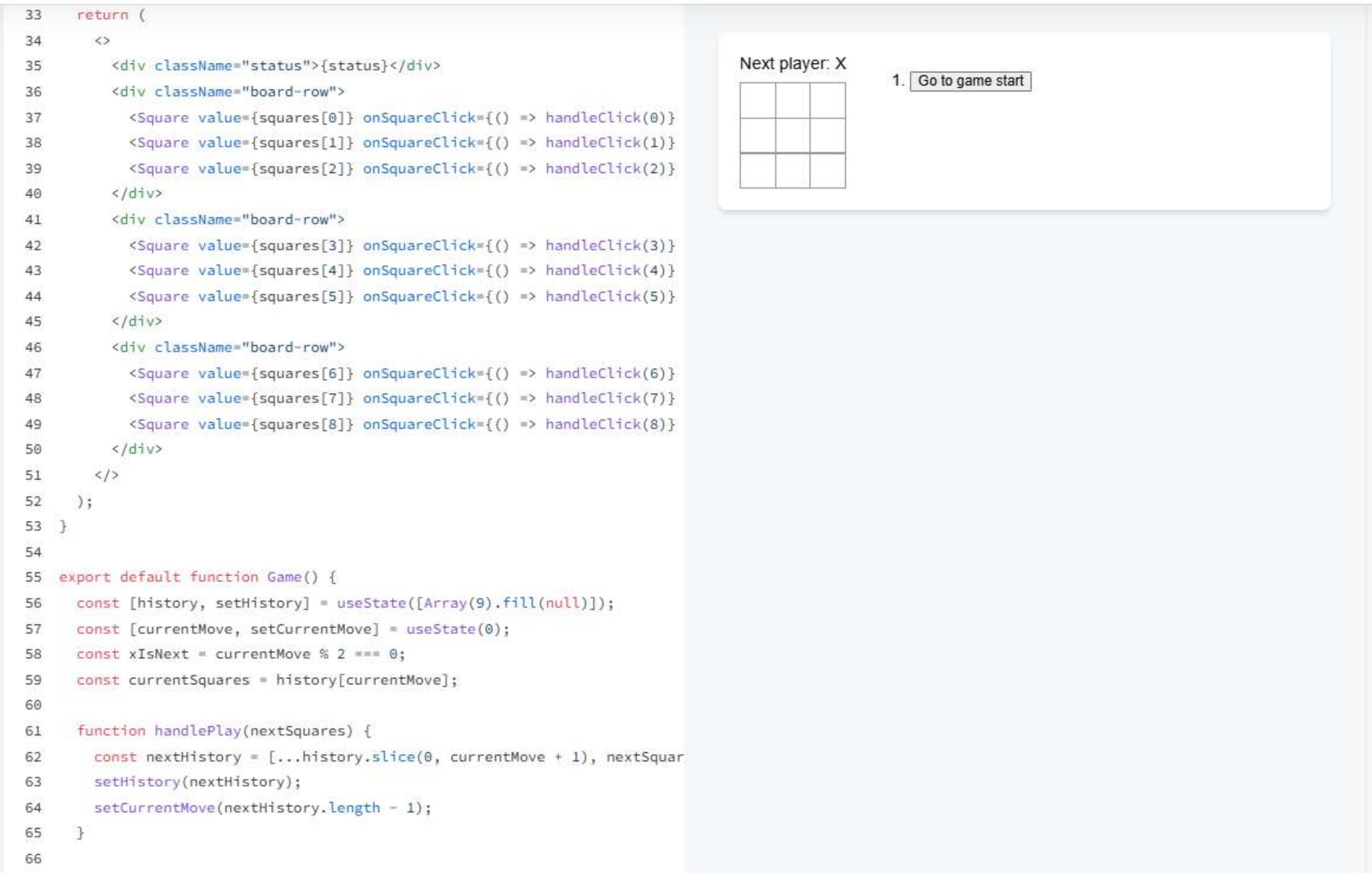
What are you building?

In this tutorial, you'll build an interactive tic-tac-toe game with React.

You can see what it will look like when you're finished here:



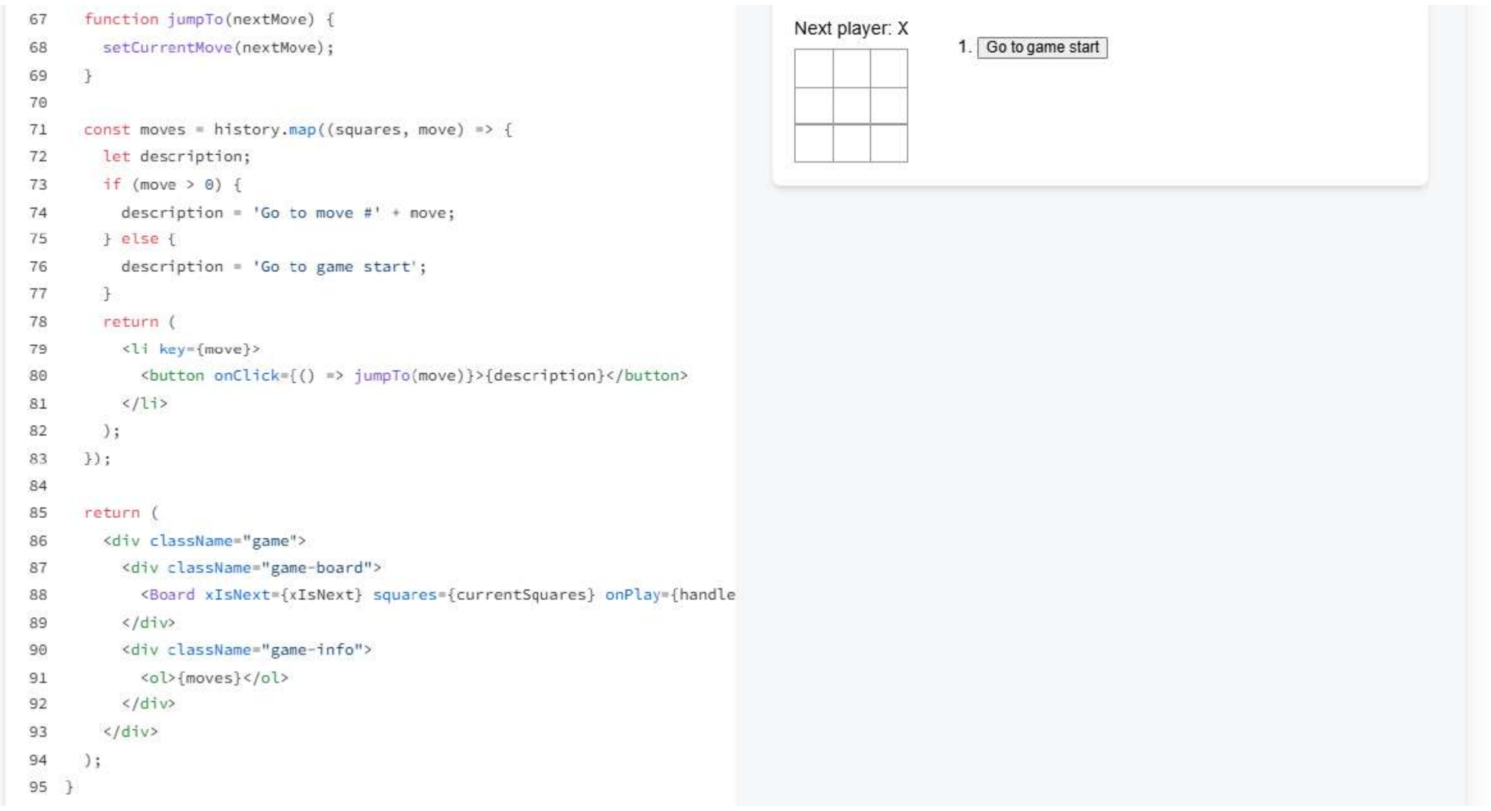
```
import { useState } from 'react';

function Square({ value, onSquareClick }) {
  return (
    <button className="square" onClick={onSquareClick}>
      {value}
    </button>
  );
}

function Board({ xIsNext, squares, onPlay }) {
  function handleClick(i) {
    if (calculateWinner(squares) || squares[i]) {
      return;
    }
    const nextSquares = squares.slice();
    if (xIsNext) {
      nextSquares[i] = 'X';
    } else {
      nextSquares[i] = 'O';
    }
    onPlay(nextSquares);
  }

  const winner = calculateWinner(squares);
  let status;
  if (winner) {
    status = 'Winner: ' + winner;
  } else {
    status = 'Next player: ' + (xIsNext ? 'X' : 'O');
  }

```



```
  return (
    <>
      <div className="status">{status}</div>
      <div className="board-row">
        <Square value={squares[0]} onSquareClick={() => handleClick(0)}
        <Square value={squares[1]} onSquareClick={() => handleClick(1)}
        <Square value={squares[2]} onSquareClick={() => handleClick(2)}
      </div>
      <div className="board-row">
        <Square value={squares[3]} onSquareClick={() => handleClick(3)}
        <Square value={squares[4]} onSquareClick={() => handleClick(4)}
        <Square value={squares[5]} onSquareClick={() => handleClick(5)}
      </div>
      <div className="board-row">
        <Square value={squares[6]} onSquareClick={() => handleClick(6)}
        <Square value={squares[7]} onSquareClick={() => handleClick(7)}
        <Square value={squares[8]} onSquareClick={() => handleClick(8)}
      </div>
    </>
  );
}

export default function Game() {
  const [history, setHistory] = useState([Array(9).fill(null)]);
  const [currentMove, setCurrentMove] = useState(0);
  const xIsNext = currentMove % 2 === 0;
  const currentSquares = history[currentMove];

  function handlePlay(nextSquares) {
    const nextHistory = [...history.slice(0, currentMove + 1), nextSquar
    setHistory(nextHistory);
    setCurrentMove(nextHistory.length - 1);
  }

```

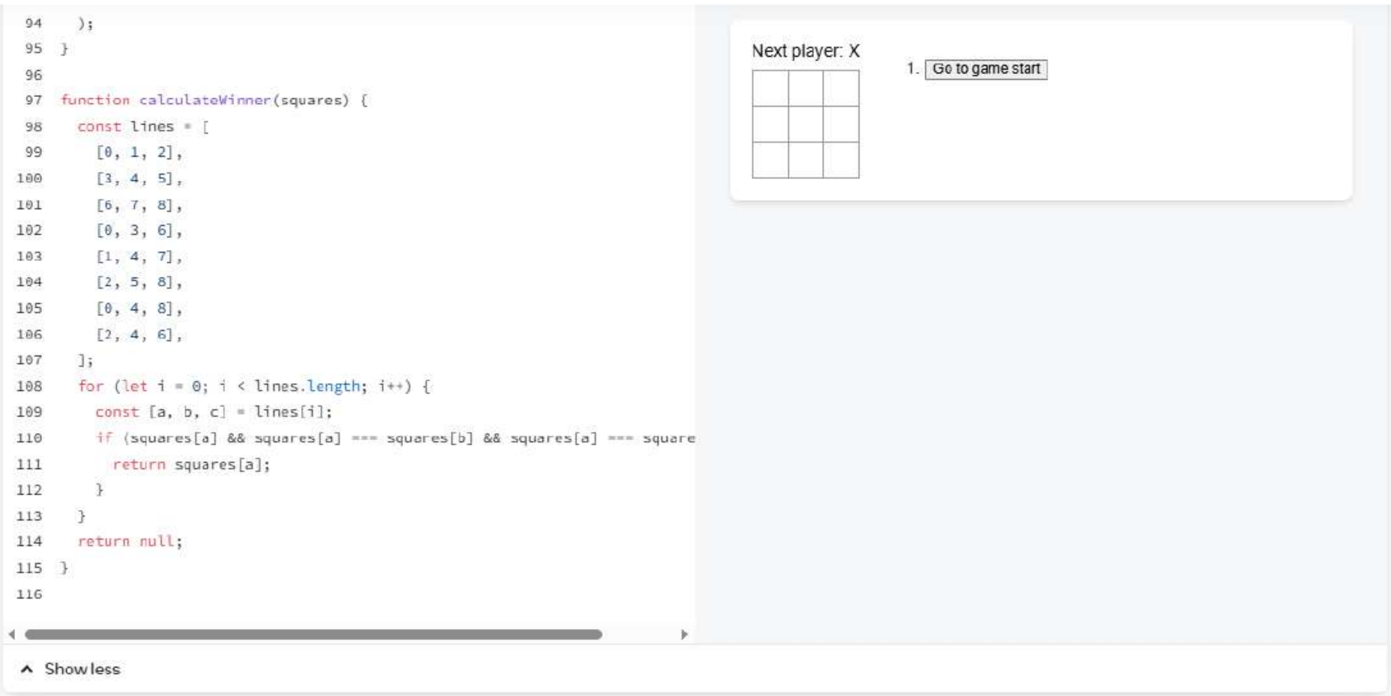
Next player: X

1. Go to game start



```
  function jumpTo(nextMove) {
    setCurrentMove(nextMove);
  }

  const moves = history.map((squares, move) => {
    let description;
    if (move > 0) {
      description = 'Go to move #' + move;
    } else {
      description = 'Go to game start';
    }
    return (
      <li key={move}>
        <button onClick={() => jumpTo(move)}>{description}</button>
      </li>
    );
  });

  return (
    <div className="game">
      <div className="game-board">
        <Board xIsNext={xIsNext} squares={currentSquares} onPlay={handle
      </div>
      <div className="game-info">
        <ol>{moves}</ol>
      </div>
    </div>
  );
}
```

```
Next player: X

1. Go to game start
```



```
  );
}

function calculateWinner(squares) {
  const lines = [
    [0, 1, 2],
    [3, 4, 5],
    [6, 7, 8],
    [0, 3, 6],
    [1, 4, 7],
    [2, 5, 8],
    [0, 4, 8],
    [2, 4, 6],
  ];
  for (let i = 0; i < lines.length; i++) {
    const [a, b, c] = lines[i];
    if (squares[a] && squares[a] === squares[b] && squares[a] === square
      return squares[a];
    }
  }
  return null;
}

```

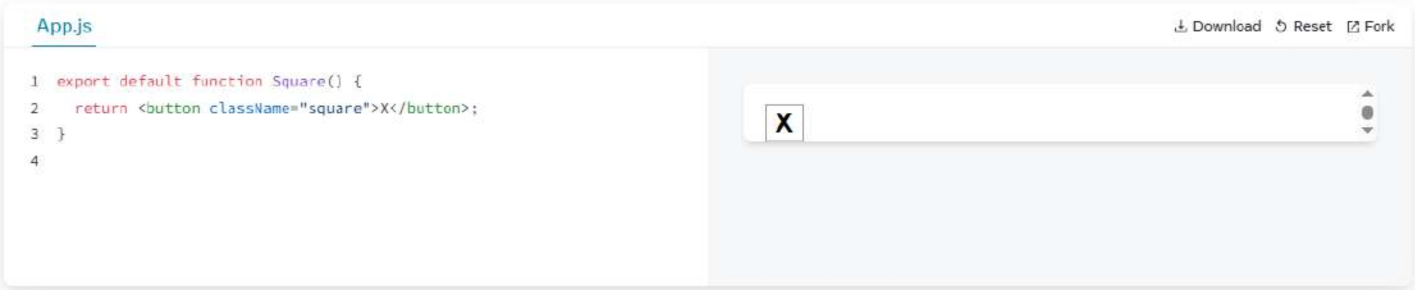
If the code doesn't make sense to you yet, or if you are unfamiliar with the code's syntax, don't worry! The goal of this tutorial is to help you understand React and its syntax.

We recommend that you check out the tic-tac-toe game above before continuing with the tutorial. One of the features that you'll notice is that there is a numbered list to the right of the game's board. This list gives you a history of all of the moves that have occurred in the game, and it is updated as the game progresses.

Once you've played around with the finished tic-tac-toe game, keep scrolling. You'll start with a simpler template in this tutorial. Our next step is to set you up so that you can start building the game.

Setup for the tutorial

In the live code editor below, click **Fork** in the top-right corner to open the editor in a new tab using the website CodeSandbox. CodeSandbox lets you write code in your browser and preview how your users will see the app you've created. The new tab should display an empty square and the starter code for this tutorial.



```
export default function Square() {
  return <button className="square">X</button>;
}

```

Note

You can also follow this tutorial using your local development environment. To do this, you need to:

1. Install Node.js
2. In the CodeSandbox tab you opened earlier, press the top-left corner button to open the menu, and then choose **Download Sandbox** in that menu to download an archive of the files locally
3. Unzip the archive, then open a terminal and `cd` to the directory you unzipped
4. Install the dependencies with `npm install`
5. Run `npm start` to start a local server and follow the prompts to view the code running in a browser

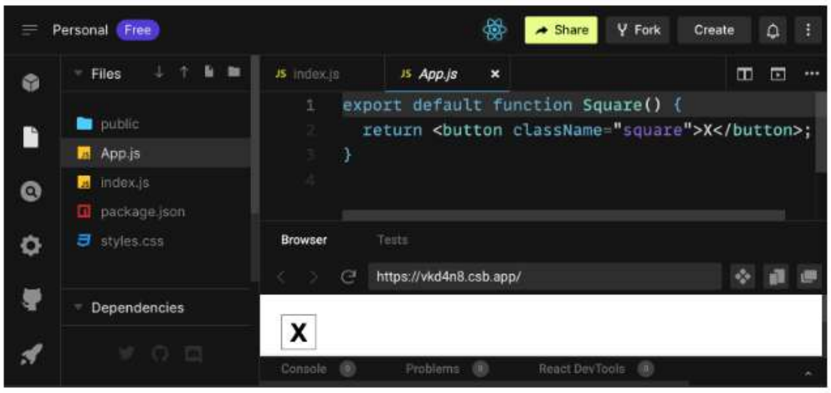
If you get stuck, don't let this stop you! Follow along online instead and try a local setup again later.

Overview

Now that you're set up, let's get an overview of React!

Inspecting the starter code

In CodeSandbox you'll see three main sections:



```
export default function Square() {
  return <button className="square">X</button>;
}
```

1. The *Files* section with a list of files like `App.js`, `index.js`, `styles.css` and a folder called `public`
2. The *code editor* where you'll see the source code of your selected file
3. The *browser* section where you'll see how the code you've written will be displayed

The `App.js` file should be selected in the *Files* section. The contents of that file in the *code editor* should be:

```
export default function Square() {
  return <button className="square">X</button>;
}
```

The *browser* section should be displaying a square with a X in it like this:

Now let's have a look at the files in the starter code.

`App.js`

The code in `App.js` creates a *component*. In React, a component is a piece of reusable code that represents a part of a user interface. Components are used to render, manage, and update the UI elements in your application. Let's look at the component line by line to see what's going on:

```
export default function Square() {
  return <button className="square">X</button>;
}
```

The first line defines a function called `Square`. The `export` JavaScript keyword makes this function accessible outside of this file. The `default` keyword tells other files using your code that it's the main function in your file.

```
export default function Square() {
  return <button className="square">X</button>;
}
```

The second line returns a button. The `return` JavaScript keyword means whatever comes after is returned as a value to the caller of the function. `<button>` is a *JSX element*. A JSX element is a combination of JavaScript code and HTML tags that describes what you'd like to display. `className="square"` is a button property or *prop* that tells CSS how to style the button. `X` is the text displayed inside of the button and `</button>` closes the JSX element to indicate that any following content shouldn't be placed inside the button.

`styles.css`

Click on the file labeled `styles.css` in the *Files* section of CodeSandbox. This file defines the styles for your React app. The first two *CSS selectors* (`*` and `body`) define the style of large parts of your app while the `.square` selector defines the style of any component where the `className` property is set to `square`. In your code, that would match the button from your Square component in the `App.js` file.

`index.js`

Click on the file labeled `index.js` in the *Files* section of CodeSandbox. You won't be editing this file during the tutorial but it is the bridge between the component you created in the `App.js` file and the web browser.

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import './styles.css';

import App from './App';
```

Lines 1-5 bring all the necessary pieces together:

- React
- React's library to talk to web browsers (React DOM)
- the styles for your components
- the component you created in `App.js`.

The remainder of the file brings all the pieces together and injects the final product into `index.html` in the `public` folder.

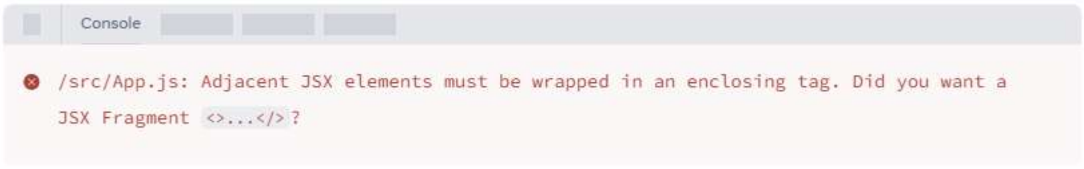
Building the board

Let's get back to `App.js`. This is where you'll spend the rest of the tutorial.

Currently the board is only a single square, but you need nine! If you just try and copy paste your square to make two squares like this:

```
export default function Square() {
  return <button className="square">X</button><button className="square">X</button>;
}
```

You'll get this error:

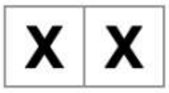


```
/src/App.js: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a
JSX Fragment <>...</>?
```

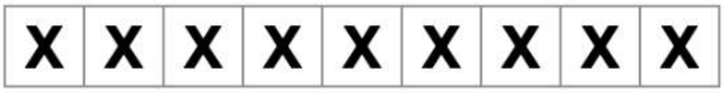
React components need to return a single JSX element and not multiple adjacent JSX elements like two buttons. To fix this you can use *Fragments* (`<>` and `</>`) to wrap multiple adjacent JSX elements like this:

```
export default function Square() {
  return (
    <>
      <button className="square">X</button>
      <button className="square">X</button>
    </>
  );
}
```

Now you should see:



Great! Now you just need to copy-paste a few times to add nine squares and…

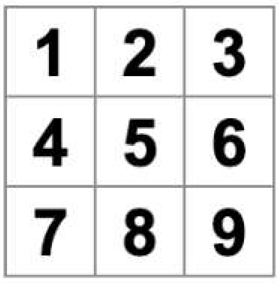


Oh no! The squares are all in a single line, not in a grid like you need for our board. To fix this you'll need to group your squares into rows with `div`s and add some CSS classes. While you're at it, you'll give each square a number to make sure you know where each square is displayed.

In the `App.js` file, update the `Square` component to look like this:

```
export default function Square() {
  return (
    <>
      <div className="board-row">
        <button className="square">1</button>
        <button className="square">2</button>
        <button className="square">3</button>
      </div>
      <div className="board-row">
        <button className="square">4</button>
        <button className="square">5</button>
        <button className="square">6</button>
      </div>
      <div className="board-row">
        <button className="square">7</button>
        <button className="square">8</button>
        <button className="square">9</button>
      </div>
    </>
  );
}
```

The CSS defined in `styles.css` styles the divs with the `className` of `board-row`. Now that you've grouped your components into rows with the styled `div`s you have your tic-tac-toe board:



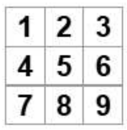
But you now have a problem. Your component named `Square`, really isn't a square anymore. Let's fix that by changing the name to `Board`:

```
export default function Board() {
  //...
}
```

At this point your code should look something like this:

App.js   Download Reset Fork

```
export default function Board() {
  return (
    <>
      <div className="board-row">
        <button className="square">1</button>
        <button className="square">2</button>
        <button className="square">3</button>
      </div>
      <div className="board-row">
        <button className="square">4</button>
        <button className="square">5</button>
        <button className="square">6</button>
      </div>
      <div className="board-row">
        <button className="square">7</button>
        <button className="square">8</button>
        <button className="square">9</button>
      </div>
    </>
  );
}
```



Show less

Note

Psssst… That’s a lot to type! It’s okay to copy and paste code from this page. However, if you’re up for a little challenge, we recommend only copying code that you’ve manually typed at least once yourself.

Passing data through props

Next, you’ll want to change the value of a square from empty to “X” when the user clicks on the square. With how you’ve built the board so far you would need to copy-paste the code that updates the square nine times (once for each square you have)! Instead of copy-pasting, React’s component architecture allows you to create a reusable component to avoid messy, duplicated code.

First, you are going to copy the line defining your first square (`<button className="square">1</button>`) from your `Board` component into a new `Square` component:

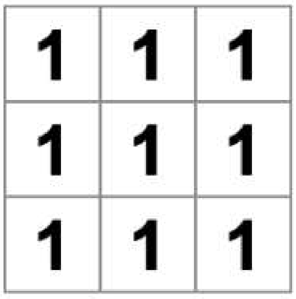
```
function Square() {
  return <button className="square">1</button>;
}

export default function Board() {
  // ...
}
```

Then you’ll update the Board component to render that `Square` component using JSX syntax:

```
// ...
export default function Board() {
  return (
    <>
      <div className="board-row">
        <Square />
        <Square />
        <Square />
      </div>
      <div className="board-row">
        <Square />
        <Square />
        <Square />
      </div>
      <div className="board-row">
        <Square />
        <Square />
        <Square />
      </div>
    </>
  );
}
```

Note how unlike the browser `div`s, your own components `Board` and `Square` must start with a capital letter.

Let's take a look:



Oh no! You lost the numbered squares you had before. Now each square says "1". To fix this, you will use *props*

Oh no! You lost the numbered squares you had before. Now each square says “1”. To fix this, you will use *props* to pass the value each square should have from the parent component (`Board`) to its child (`Square`).

Update the `Square` component to read the `value` prop that you'll pass from the `Board`:

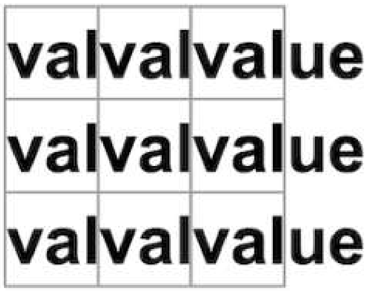
```
function Square({ value }) {
  return <button className="square">1</button>;
}
```

`function Square({ value })` indicates the Square component can be passed a prop called `value`.

Now you want to display that `value` instead of `1` inside every square. Try doing it like this:

```
function Square({ value }) {
  return <button className="square">value</button>;
}
```

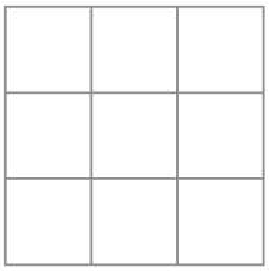
Oops, this is not what you wanted:



You wanted to render the JavaScript variable called `value` from your component, not the word “value”. To “escape into JavaScript” from JSX, you need curly braces. Add curly braces around `value` in JSX like so:

```
function Square({ value }) {
  return <button className="square">{value}</button>;
}
```

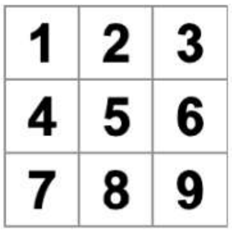
For now, you should see an empty board:



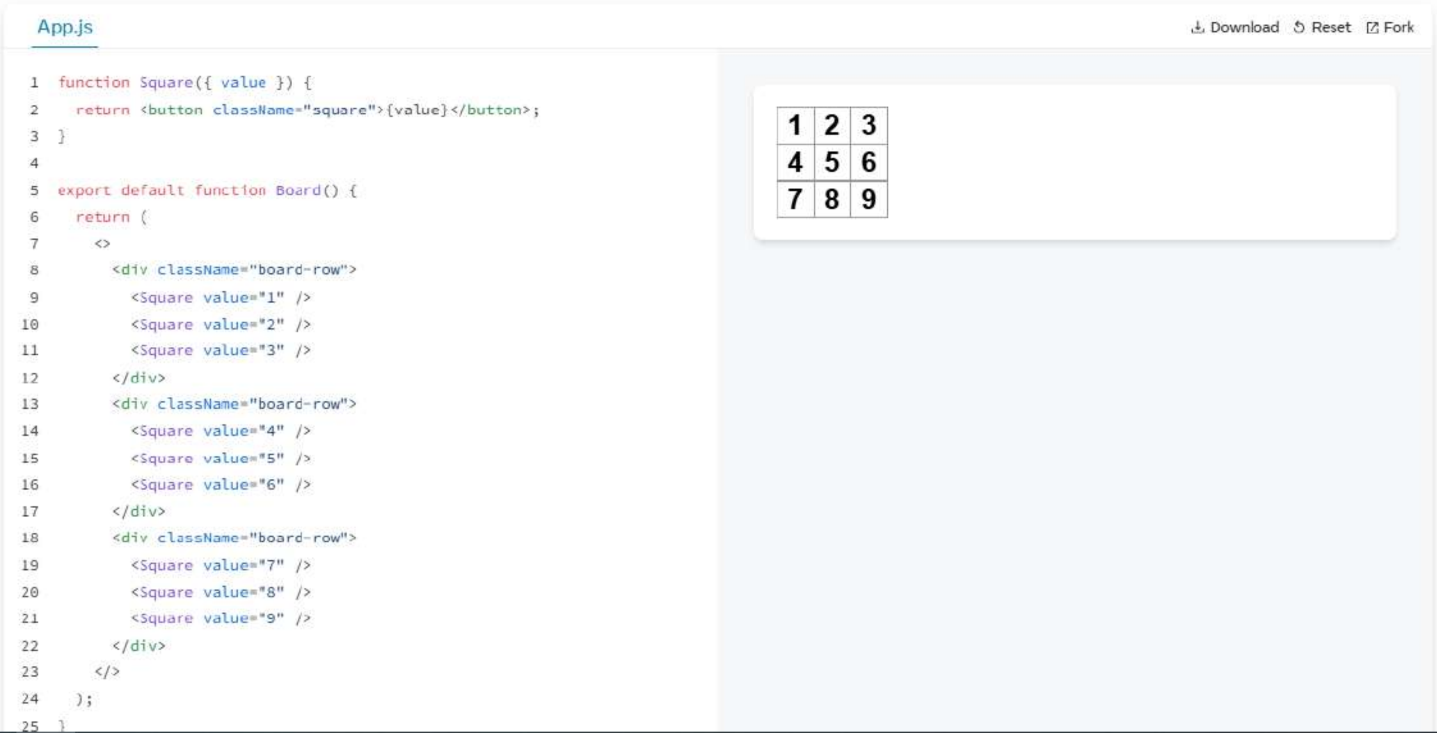
This is because the `Board` component hasn't passed the `value` prop to each `Square` component it renders yet. To fix it you'll add the `value` prop to each `Square` component rendered by the `Board` component:

```
export default function Board() {
  return (
    <>
      <div className="board-row">
        <Square value="1" />
        <Square value="2" />
        <Square value="3" />
      </div>
      <div className="board-row">
        <Square value="4" />
        <Square value="5" />
        <Square value="6" />
      </div>
      <div className="board-row">
        <Square value="7" />
        <Square value="8" />
        <Square value="9" />
      </div>
```

Now you should see a grid of numbers again:



Your updated code should look like this:



```
function Square({ value }) {
  return <button className="square">{value}</button>;
}

export default function Board() {
  return (
    <>
      <div className="board-row">
        <Square value="1" />
        <Square value="2" />
        <Square value="3" />
      </div>
      <div className="board-row">
        <Square value="4" />
        <Square value="5" />
        <Square value="6" />
      </div>
      <div className="board-row">
        <Square value="7" />
        <Square value="8" />
        <Square value="9" />
      </div>
    </>
  );
}
```

Making an interactive component

Let's fill the `Square` component with an `X` when you click it. Declare a function called `handleClick` inside of the `Square`. Then, add `onClick` to the props of the button JSX element returned from the `Square`:

```
function Square({ value }) {
  function handleClick() {
    console.log('clicked!');
  }

  return (
    <button
      className="square"
      onClick={handleClick}
    >
      {value}
    </button>
  );
}
```

If you click on a square now, you should see a log saying `"clicked!"` in the *Console* tab at the bottom of the *Browser* section in CodeSandbox. Clicking the square more than once will log `"clicked!"` again. Repeated console logs with the same message will not create more lines in the console. Instead, you will see an incrementing counter next to your first `"clicked!"` log.

Note

If you are following this tutorial using your local development environment, you need to open your browser's Console. For example, if you use the Chrome browser, you can view the Console with the keyboard shortcut **Shift + Ctrl + J** (on Windows/Linux) or **Option + ⌘ + J** (on macOS).

As a next step, you want the Square component to “remember” that it got clicked, and fill it with an “X” mark. To “remember” things, components use *state*.

React provides a special function called `useState` that you can call from your component to let it “remember” things. Let’s store the current value of the `Square` in state, and change it when the `Square` is clicked.

Import `useState` at the top of the file. Remove the `value` prop from the `Square` component. Instead, add a new line at the start of the `Square` that calls `useState`. Have it return a state variable called `value`:

```
import { useState } from 'react';

function Square() {
  const [value, setValue] = useState(null);

  function handleClick() {
    //...
```

`value` stores the value and `setValue` is a function that can be used to change the value. The `null` passed to `useState` is used as the initial value for this state variable, so `value` here starts off equal to `null`.

Since the `Square` component no longer accepts props anymore, you’ll remove the `value` prop from all nine of the Square components created by the Board component:

```
// ...
export default function Board() {
  return (
    <>
      <div className="board-row">
        <Square />
        <Square />
        <Square />
      </div>
      <div className="board-row">
        <Square />
        <Square />
        <Square />
      </div>
```