

Extracted from:

Modern Front-End Development for Rails  
Hotwire, Stimulus, Turbo, and React

This PDF file contains pages extracted from *Modern Front-End Development for Rails*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2021 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Raleigh, North Carolina

The  
Pragmatic  
Programmers

# Modern Front-End Development for Rails

Hotwire, Stimulus, Turbo, and React



Noel Rappin  
*edited by Katharine Dvorak*



# Modern Front-End Development for Rails

Hotwire, Stimulus, Turbo, and React

Noel Rappin

The Pragmatic Bookshelf

Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit <https://pragprog.com>.

The team that produced this book includes:

CEO: Dave Rankin

COO: Janet Furlow

Managing Editor: Tammy Coron

Development Editor: Katharine Dvorak

Copy Editor: Adaobi Obi Tulton

Indexing: Potomac Indexing, LLC

Layout: Gilson Graphics

Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact [support@pragprog.com](mailto:support@pragprog.com).

For international rights, please contact [rights@pragprog.com](mailto:rights@pragprog.com).

Copyright © 2021 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN-13: 978-1-68050-721-8

Encoded using the finest acid-free high-entropy binary digits.

Book version: P1.0—June 2021

## Interactivity, State, and Hooks

At this point, React has taken over part of our page and is drawing the seats, which is nice enough, but we'd like it to, you know, *react* to something. We'd like to have a little interactivity.

In React, you can use JSX to specify event handlers on React elements in much the same way you would when writing old-school JavaScript embedded in HTML. The problem is how to make changes to our components as a result of those events. As mentioned, the props we pass into each component are immutable, which means if we want to change something about a component, we can't use props. React uses the term *state* to refer to the parts of a component that change and trigger an update to how the component is displayed when they are changed.

To be clear, although a component can't change its own props, changing the state of a component can cause that component to rerender child components with new props.

Because state changes are used by React to trigger a redrawing of the page, React requires you to register them with the system; you can't just change the value of a variable and be done with it. React allows you to designate a value as being part of the state and gives you a special setter for that value using a mechanism called *hooks*.

Hooks are new in React as of version 16.8. Before then, components defined as functions could not manage changing state (components defined as classes always could manage state using a different mechanism). As mentioned earlier, the React core team has said that hooks and functional components are the way of the future, which is why we will be focusing on using hooks to manage state in this book.

Here's the code for the Seat component that changes status when clicked:

```
chapter_04/03/app/packs/components/seat.tsx
Line 1  import * as React from "react"
-
-  interface SeatProps {
-    seatNumber: number
5     initialStatus: string
-  }
-
-  const Seat = ({
-    seatNumber,
10   initialStatus,
-  }: SeatProps): React.ReactElement => {
-    const [status, setStatus] = React.useState(initialStatus)
```

```

-
-   function changeState(): void {
15   if (status === "held") {
-       setStatus("unsold")
-   } else {
-       setStatus("held")
-   }
20 }

-
-   function stateDisplayClass(): string {
-       if (status === "held") {
-           return "bg-green-500"
25   } else {
-       return "bg-white hover:bg-blue-300"
-   }
-   }
-
30   const cssClass = "p-4 m-2 border-black border-4 text-lg"
-
-   return (
-       <td>
-       <span
35         className={` ${cssClass} ${stateDisplayClass()} `}
-         onClick={changeState}>
-           {seatNumber + 1}
-         </span>
-       </td>
40   )
-   }
-
-   export default Seat

```

The first new React-specific line here is line 12, `const [status, setStatus] = React.useState(props.initialStatus)`. We are calling the React method `useState`, which is a React *hook* method. It's called a hook because it allows our component to “hook into” the React rendering life cycle to allow the component to change the larger system. React defines several different default hooks, plus you can create your own.

Right here, right now, we're calling `useState`. What `useState` does is register a given value as being a part of React state such that changing that value triggers a rerender. The argument to `useState` is the initial value of the new state object in question—in our case, we're taking the value from an `initialState` passed in as a prop. (We'll need to change the `row.tsx` component so that its call looks like this: `<Seat key={seatNumber} seatNumber={seatNumber} initialState="unsold" />`.)

The `useState` method has kind of a weird return value; it returns a two-element array, which you typically capture into two different variables using JavaScript's destructuring syntax. Here we are capturing the values into variables

named `status` and `setStatus`. The first return value, in our case, `status`, is a property that has the current value of our state. The second return value, `setStatus`, is our state setter—a function that we can call later in our component to change the value of the state and trigger a redraw.

---

#### The `useState` Hook Initial Value

---



One important gotcha to keep in mind here is that the argument passed to `useState` is only used the first time the component is rendered. On subsequent rerenders, the component keeps track of the existing state and does not need or use the initial value.

---

This is great—we now have a mechanism for both getting and setting the value of the changing state of our component, which we can then use through the rest of our component.

Let's jump to the JSX return value of the component. Two things about this value have changed:



- the `className` now includes a call to a `stateDisplayClass()` function, and
- we’ve added another prop to the `span`, namely `onClick={changeState}`.

The `onClick` prop is how React does event handling: you create a prop whose name is `on` followed by the event; the value of that prop is a function that is called when the event happens. In our case, we’re using `{changeState}`. (For a complete list of event names supported by React, check out the official docs on the React website.)<sup>2</sup>

When the button is clicked, the `onClick` event fires, which causes us to go to the `changeState` function inside our component. Within that function we do a check on the value of `status`—the same `status` variable that was defined by the call to `useState`. We then change the value of `status` based on the current value of `status` using the `setState` function, also the one defined by `useState`, to officially register the change with React.

Using `setState` triggers a redraw of the element, which takes us back to the return value and the call to `stateDisplayClass()`, which is used to change the background color of the item based on the current status. Clicking once changes the status to `held`, which then causes the display class to be `bg-green-500`—Tailwind-speak for “make the background green.” Clicking again calls `setStatus("unsold")`, and the rerender changes the display class to `bg-white hover:bg-blue-300`, or “make the background white but change it to light blue when we hover the mouse pointer over it.” There are a couple of logistical issues with React hooks to keep in mind:

- Hooks can only be used in components that are defined as functions and can only be declared at the top level of the function—not inside a nested function, loop, or if statement.
- If you want to manage more than one value in state, you can make multiple calls to `useState` to get setters for each of them, or you can have the initial value be an array or object. If the value gets more complicated, there may be other hooks that will be easier to use, which we’ll talk more about in [Chapter 12, Managing State in React, on page ?](#).
- If it bothers you that the status takes strings as values but only has a limited number of valid string values, never fear, TypeScript has a mechanism for that, and we’ll take a look at it in [Chapter 14, Validating Code with Advanced TypeScript, on page ?](#).

---

2. <https://reactjs.org/docs/events.html>