Future UI

Chris Biscardi

Contents

Chapter 1

What is this book?

This book will help you explore the state of cutting edge UI/UX development. We will focus on React, Webpack, both Flux and "Flux-inspired" architectures, Routing and Universal Applications.

1.1 Tools

The TC39 categorises proposals into 4 stages:

- Stage 0 - Strawman
- Stage 1 - Proposal
- Stage 2 - Draft
- Stage 3 - Candidate
- Stage 4 - Finished

All of the code in this book is written using a stage 0 babel transpiler. This means we have access to future ECMAScript features all the way back to the Strawman stage (currently considered "ES7").

1.2 Code examples

The simple code examples often use @gaearon's react-hot-boilerplate which provides a simple base to jump in and explore example code.

Chapter 2

Rendering

In this book, we will be focusing on React for controlling the rendering of our applications. According to the React site, React is:

> A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

Notice that there is no mention of the DOM because React is not tied to a specific render target. In practice, this means we can render to DOM, SVG, Canvas and even native iOS components. Most importantly, React is a component model for abstractions.

2.1 Virtual DOM

The abstraction used which allows us to render to multiple targets is referred to as the Virtual DOM. Simply speaking, the Virtual DOM is a tree structure which allows us to diff against previous versions of our rendered application.

In practice one of the benefits is that we have to worry less about DOM insertion optimizations such as batching updates. A spec-compliant synthetic event system means that bubbling and capturing works the same across browsers.

The React docs explain the implications well:

> Event delegation: React doesn't actually attach event handlers to the nodes themselves. When React starts up, it starts listening for all events at the top level using a single event listener. When a component is mounted or unmounted, the event handlers are simply added or removed from an internal mapping. When an event occurs, React knows how to dispatch it using this mapping. When there are no event handlers left in the mapping, React's event handlers are simple no-ops. To learn more about why this is fast, see David Walsh's excellent blog post.

2.2 Basics

2.2.1 A Simple Component

The following component renders a heading. We can then use `React.render` to attach to the body of the document.

```
import React, { Component } from 'react';

export default class Simple extends Component {
  render() {
    return <p>Hello World</p>;
  }
}
```

```
React.render(<Simple/>, document.body);
```

To see this component in action, run the following:

```
git clone git@github.com:future-ui/basics.git
cd basics && npm install
npm start
```

then go to `localhost:3000`

The example is hot-reloadable, which means that you can edit the `src/App.js` file and see the changes in the browser immediately. You should play around a bit and see what happens.

2.2.2 A More Complex Example

We want to display a counter which we can adjust using various buttons.

```
import React, { Component } from 'react';

export default class Counter extends Component {

  state = {
    count: 0
  }

  increment = (e) => {
    const { count } = this.state;
    this.setState({
      count: count + 1
```

```
    });
  }

  decrement = (e) => {
    const { count } = this.state;
    this.setState({
      count: count - 1
    });
  }

  render() {
    return (
      <div>
        {this.renderControls()}
        <section>
          <h1>My Awesome Counter!</h1>
          <p>Counters are an integral part of counting things!</p>
        </section>
      </div>
    )
  }

  renderControls = () => {
    const { count } = this.state;
    return (
      <section>
        <button onClick={this.decrement}>-</button>
        <span>{count}</span>
        <button onClick={this.increment}>+</button>
      </section>
    );
  }
}
```

If you take the above example and paste it in to the App.js file from before, you will see it render roughly as:

```
<div data-reactid=".0">
  <section data-reactid=".0.0">
    <button data-reactid=".0.0.0">-</button><span data-reactid=
    ".0.0.1">7</span><button data-reactid=".0.0.2">+</button>
  </section>

  <section data-reactid=".0.1">
    <h1 data-reactid=".0.1.0">My Awesome Counter!</h1>
```

```
    <p data-reactid=".0.1.1">Counters are an integral part of counting
    things!</p>
  </section>
</div>
```

Each of the `data-reactids` is a node in the tree.