Building Web Applications with React

Chapter 1: Getting Started with React

Chapter Objectives

In this chapter, we will:

- Answer the question: what is React?
- Create a 'Hello World' React application
- Compose a React application from components

Chapter Concepts

Introducing React

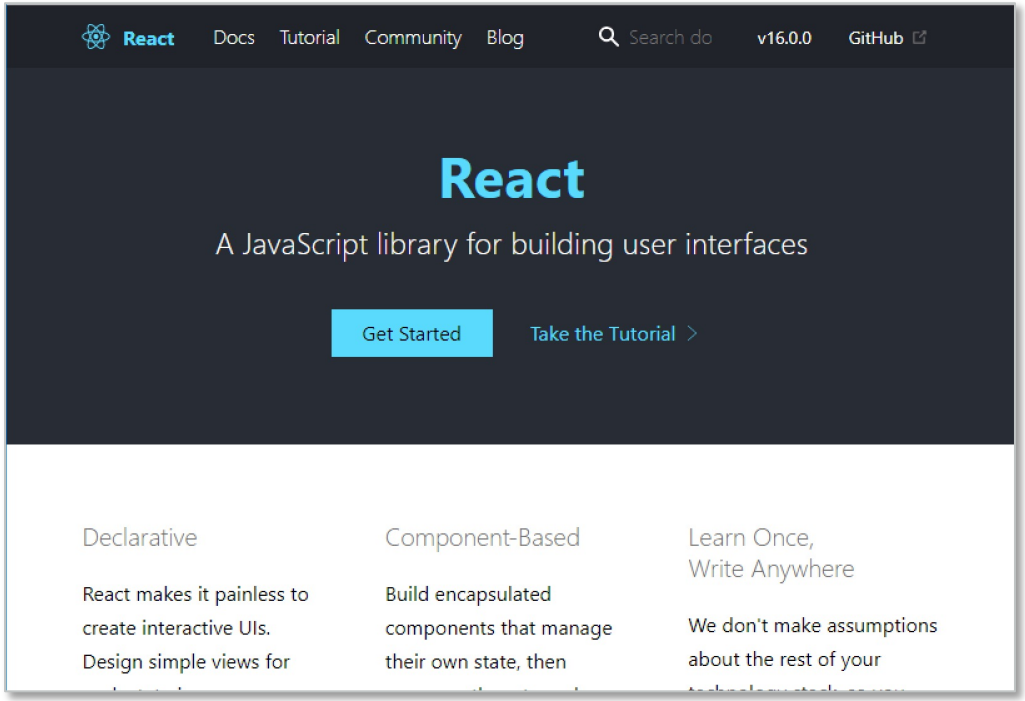
Introducing JSX

React Components

Chapter Summary

What Is React?

- React is a JavaScript library for building User Interfaces
  - Created and maintained by Facebook
    - Used for Instagram and WhatsApp, and extensively on Facebook
    - Growing popularity outside of Facebook
  - Available under an Open-Source MIT license



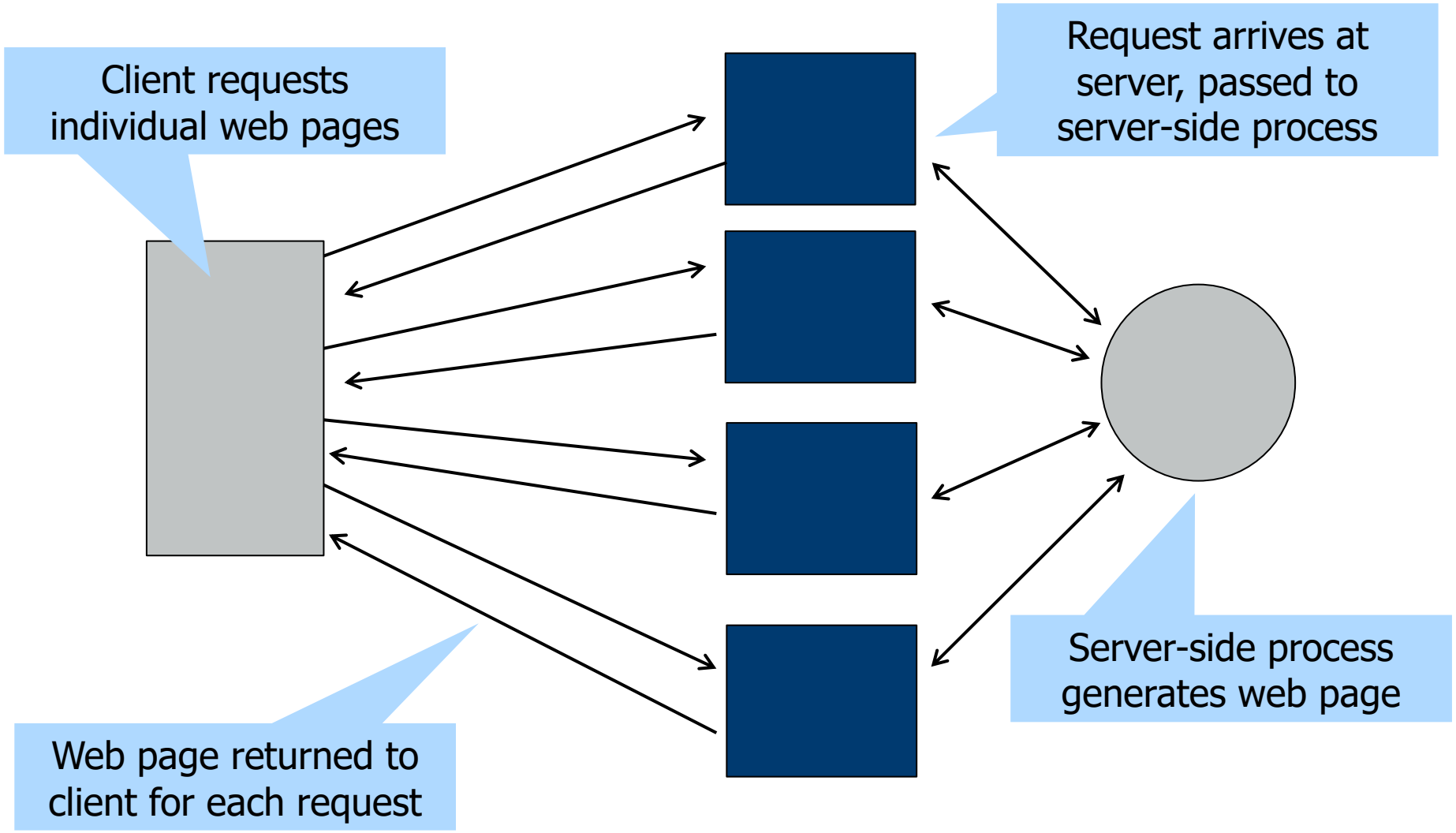
Forms of React

- React is used to create Single Page Applications (SPAs)
  - Primarily HTML/web-based applications
  - Also, native iOS and Android apps with React Native
- Different approaches to using React
  - Integration with existing web pages
    - React added via `<script>` element
  - Conventional React SPA
    - React provides the entire page/site
    - All JavaScript runs on the client
  - Server-rendered React
    - React runs on the server to generate static resources
    - Resources are returned to the client
  - Others
    - E.g., Progressive Web App Generators
    - Ever growing list of toolchains leveraging React

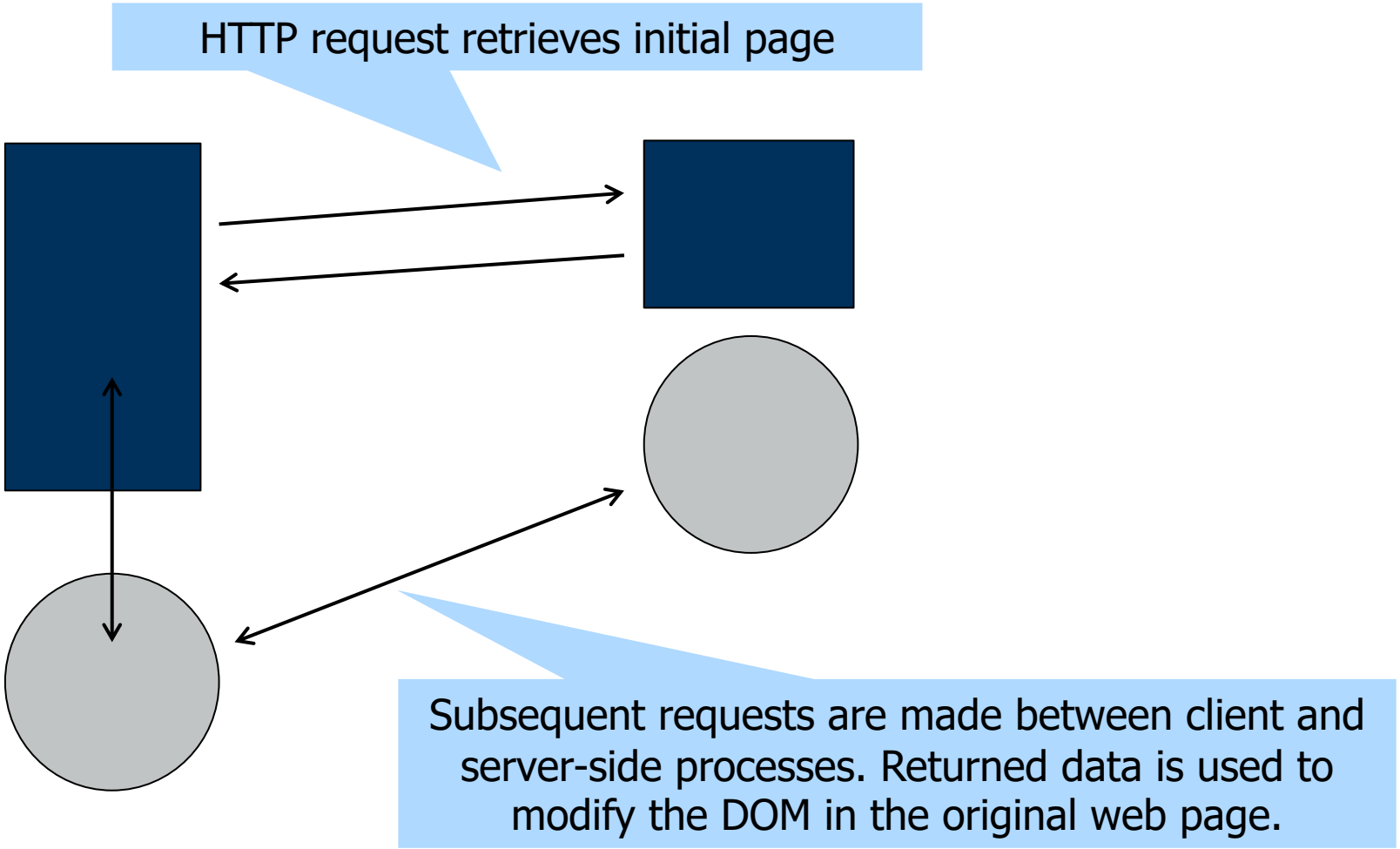
Single Page Applications

- Traditional web applications are comprised of multiple pages
  - Hyper Text Transfer Protocol (HTTP) is used to request web pages
  - Each request involves a complete page refresh
- SPAs use a single shell page
  - HTTP requests use Ajax to fetch JavaScript Object Notation data
  - JSON is used to update the page's Document Object Model (DOM)
- Many SPA Frameworks are available
  - `Angular.js`
  - `Ember.js`
  - `Aurelia.js`
  - `React.js`
  - Etc.

Traditional Web Applications



Single Page Applications



Why Choose React?

- React does not provide many of the features of other SPAs
  - No built-in Ajax functionality
    - Uses existing browser functionality
  - No two-way data binding
- React is purely a tool for building the UI
  - *Very* fast
  - Easy to develop and maintain
- Declarative model, simpler than conventional client-side development
  - No complicated DOM interaction
  - No developer DOM updates at all
- React components simply render output based on the current data
  - Data changes are the *logical* equivalent of a page refresh
    - Actual DOM updates are restricted to changes only
    - Underlying React engine manages all interaction with the DOM

The Virtual DOM

- React re-renders the entire application on every update
  - This sounds very slow and resource-intensive
  - It would be, were it not for the virtual DOM
- The browser has a DOM (Document Object Model)
  - An abstract representation of the web page
  - DOM interaction can be slow and inconsistent between browsers
- React has its own virtual DOM
  - An abstract representation of the browser's abstract representation
  - The virtual DOM is lightweight and very fast
- React components render to the virtual DOM
  - React compares virtual DOM with existing in-memory virtual DOM
  - Diff algorithm returns the minimum changes required
    - React updates actual DOM, using browser-specific optimizations

React Component Modules

- React components are normally developed as *modules*
  - Small units of reusable code
  - Functionality is *exported* by the module, *imported* by the client
  - Modules are typically created in separate individual files
    - Can be packaged for reuse
- Various module specifications have existed for some time
  - AMD, CommonJS, Node, etc.
- An official part of JavaScript since the release of ES2015
- Modules are not yet supported directly by all browsers
  - Need to be compiled into browser-compatible syntax
    - Typically, also bundled into single `.js` file
    - Still developed in separate files

Module Code Sample Using Classes

Instructs the environment that `Book` variable is to be exported for reuse

```
export default class Book extends React.Component {
    render() {
       return (<tr>
                 <td>{this.props.title}</td>
                 <td>{this.props.author}</td>
               </tr>);
        }
}
```

Binds imported variables from other modules into the current scope

```
import React from 'react';
import Book from './Book';

export default class Books extends React.Component {
```

`Book` is a module, and so is React itself!

Module Code Sample Using Functions

Alternative syntax for creating React components as functions

```
const About = () => (
        <div className="row">
            <h1>About Book Reactions</h1>
        </div>
)

export default About;
```

Could also have exported the class like this

Functional component imports are the same as class components

```
import About from './components/about/About';
import './App.css';

function App() {
```

Browser Support for Modules

- Not all browsers directly support modules
  - Need to transpile it into a syntax they do support
  - Just as JSX is transpiled into browser-friendly JavaScript
  - Babel is capable of transpiling both JSX and modules for browsers
- Modules are developed in separate files
  - Promotes maintainability and reuse
- Need to be converted into correct syntax and bundled into single file
- Recall that modules are often distributed as packages
  - Bundle will include packages on which the application depends
    - Such as React itself

Bundling React Modules

- Creating the final React bundle involves multiple tools working together
  - Many different choices and combinations
- Node.js
  - Widely used to download packages and specify dependencies
    - React modules are available as node packages
      - Can be downloaded using npm
- Build Managers
  - WebPack
  - Gulp
- Tools to transpile and build for the browser
  - Browserify
  - Babel
- Can be complex to manage
  - Greatly simplified by `create-react-app`

create-react-app

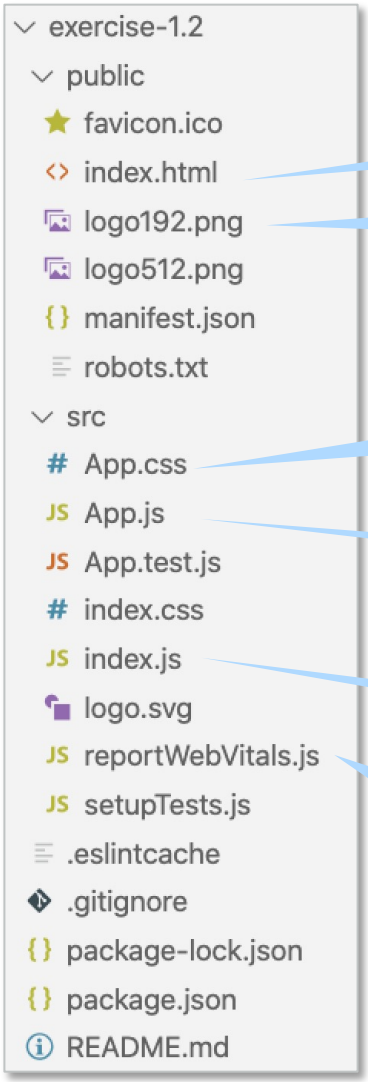
- Provides a complete environment for building React apps
- Massively simplifies configuration
  - Creates a fully configured and working start point
  - Can be "ejected" if greater control is required
    - Making it equally useful for newcomers and power users
- Provides pre-configured commands for different builds
  - Development
  - Production
  - Test
- Includes a development server with hot-reload
  - No need to stop and restart to see changes
- Installed via `npx`
  - Ensures always using the latest version

```
npx create-react-app app-name
```

Exercise 1.1: Getting Started with React

- In this exercise, you will create a simple 'Hello World' web page using `create-react-app`
- Please refer to the Exercise Manual

File Structure



Compiled JS files will be referenced in `index.html`

Resources such as images can go here

CSS to include with the compiled application

The root component

Connects React to the DOM

Can be used for performance metrics

index.js

- Connects React to the DOM
  - Contains boilerplate code

Imports `React` and web-specific `ReactDOM`

Any CSS in `index.css` will be added to a generated CSS file

Import the root component

The `render()` method connects React to the DOM

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a
 function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/C
RA-vitals
reportWebVitals();
```

App.js

- The root component
  - React is hierarchical
  - Every other component is nested beneath the root

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      {/*code omitted */}
    </div>
  );
}

export default App;
```

The root component, called from `index.js`

This is JSX, not HTML. Compiles to JavaScript.

Chapter Concepts

Introducing React

Introducing JSX

React Components

Chapter Summary

Writing React

- Two ways to write React
  - Pure JavaScript
  - JavaScript with JSX
    - XML-like syntax that compiles to JavaScript
- Pure JavaScript approach uses the React API methods to create HTML
  - `React.createElement()`
  - `React.isValidElement()`

Element to create

Element content

```
React.createElement("h1", null, "Hello World")
```

Optional HTML attributes object literal

JSX

- Most React applications are written using JSX
  - Simpler than the React API
    - Especially when building nested HTML elements
- JSX is an optional HTML-like syntax
  - Generates JavaScript objects using HTML syntax
  - Easy learning curve for web developers
- JSX is *not* understood by browsers
  - Must be transpiled into JavaScript
  - The Babel JavaScript library converts JSX into JavaScript
    - Part of build process run by a code packager
      - Webpack
      - Gulp
      - Etc.
    - `create-react-app` uses Webpack

The JSX Advantage

- Although it looks and feels like HTML, JSX is compiled into JavaScript
  - Any unclosed tags will lead to a compilation error at design time
- Compilation provides a big advantage
  - Over HTML itself
  - Over SPA frameworks like AngularJS which use string templates for HTML
    - Not compiled
    - Unclosed tags fail at runtime, not design-time

JSX Advantage Illustrated

C:\WINDOWS\system32\cmd.exe

```
[21:32:00] Starting 'build'...
[21:32:00] Finishe
[21:32:00] Startin
[21:32:00] Finishe
events.js:141
      throw er; // Unhandled 'error' event
      ^

SyntaxError: C:/2015 Programming/React Course/Course/Load/Demos/Chapter 2 contro
lled inputs/src/components/books/AddBookForm.jsx: Unterminated JSX contents (28:
11)
  26 |                                <label>3. Title (controlled):<input type
="text" className="form-control" value={this.state.title} onChange={this.setTitl
e.bind(this)} /></label>
  27 |                        </div>
> 28 |                        </form>);
     |                               ^
  29 |  }
  30 |
  31 | }
    at Parser.pp.raise (C:\2015 Programming\React Course\Course\Load\Demos\Chapt
er 2 controlled inputs\node_modules\babelify\node_modules\babel-core\node_module
s\babylon\index.js:1378:13)
    at Parser.pp.jsxReadToken (C:\2015 Programming\React Course\Course\Load\Demo
s\Chapter 2 controlled inputs\node_modules\babelify\node_modules\babel-core\node
```

Babel transpiler identifies error and provides file and line number to facilitate correction

JSX Syntax

- Almost identical to HTML
- Elements are lowercase
  - `<div>` not `<DIV>`
- Conform to XML syntax for empty elements
  - `<img />` not `<img>`
- Like XML, must have single root element
  - In each `render()` method
  - Root element can be `React.Fragment`: `<></>`
    - Does not render any HTML into the DOM
- Minor differences from HTML where keywords clash
  - CSS `class` selector and JavaScript `class`
  - HTML `for` attribute and JavaScript `for`
    - Use `className` and `htmlFor` instead
- JSX `textarea` uses `value` attribute, not element content

JSX Syntax Example

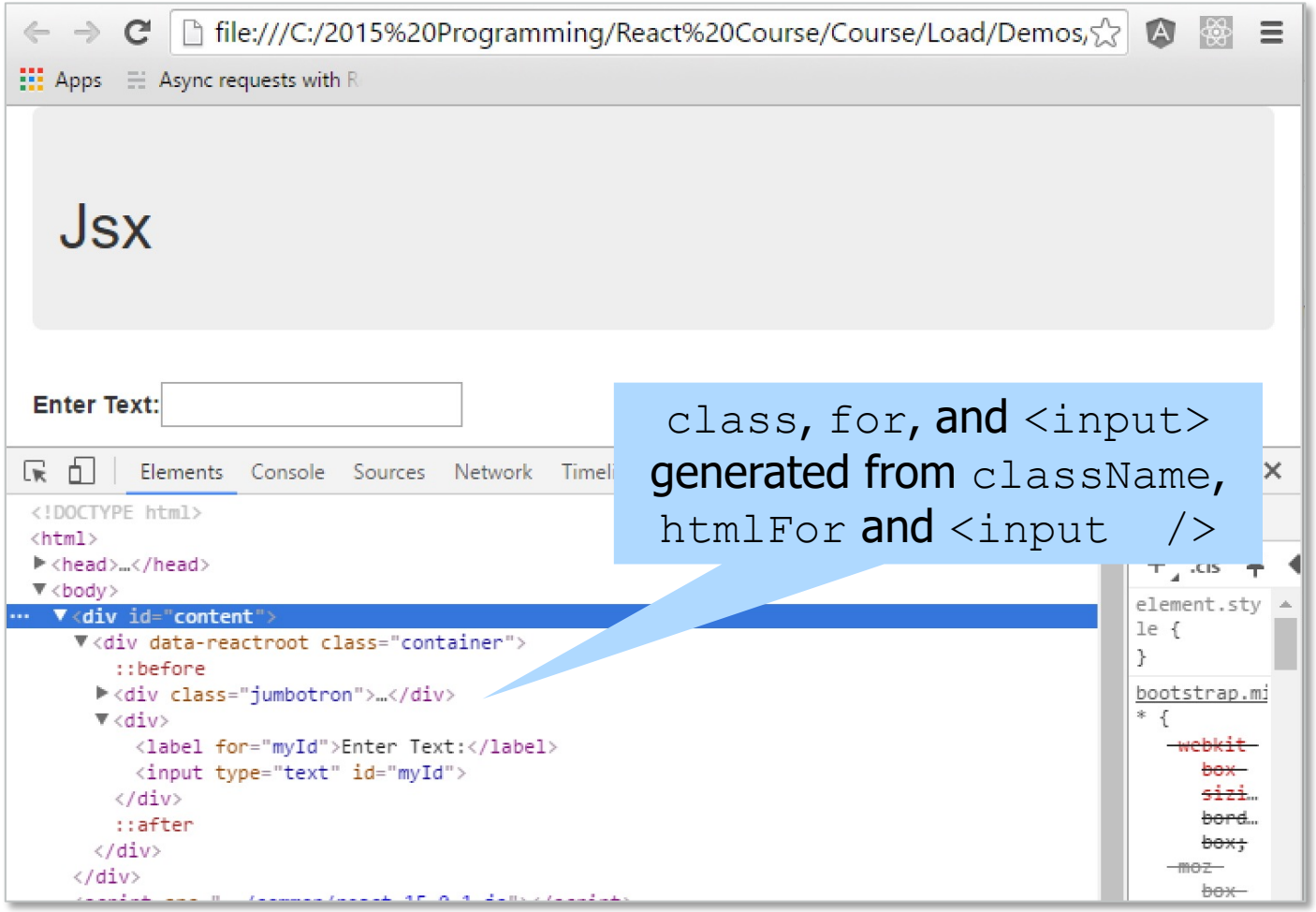
```
<div className="container">
  <div className="jumbotron">
    <h1>Jsx</h1>
  </div>
  <div>
    <label htmlFor="myId">Enter Text:</label>
    <input id="myId" type="text" />
  </div>
</div>
```

`className` instead of `class`

`htmlFor` instead of `for`

XML style empty element

JSX Example Output



Chapter Concepts

Introducing React

Introducing JSX

React Components

Chapter Summary

React Components

- React components are composable, reusable objects
- Two approaches to writing them:
  - JavaScript Classes inherited from `React.Component`
  - Functional components
- Class components
  - Use lifecycle methods
  - Have a single compulsory method `render()` that returns either:
    - A single `ReactElement`, with or without children
    - Another `React.Component`
- Functional components are JavaScript functions
  - Some are simple arrow functions
    - Body of function becomes the `render()` method
  - Others more complex
    - Explicit `return` from the function become the `render()` method
    - Integrate with component lifecycle via *hooks*
      - So called because they *hook into* lifecycle events

ES6 Classes

- Historically, JavaScript did not have classes
  - Many frameworks created their own, including React
- ES6 contains classes
  - Really just syntactic sugar over prototypical inheritance

```
class Thing {
  constructor(hello)
  {
     this.property = hello;
  }
}

class ChildThing extends Thing {
  constructor() {
      super("hello");
      this.newProperty = "world";
  }
  doSomething() {
      alert("I'm a function!");
  }
}
```

Inherit using `extends`

Call parent constructor

`function` keyword not required

React Component Classes

- React ES6 classes extend React.Component
  - One compulsory method: `render()`

```
class App extends React.Component {

  render() {
     return (<p>Hello class world</p>);

  }

}
```

Return output in the `render()` method

- Not all browsers support ES6 classes yet
  - Must be converted into browser-compatible syntax
  - Done automatically as part of Webpack build process
    - Part of `create-react-app` configuration

Functional Components

- Can be written using function keyword or ES6 arrow syntax
- Come in two forms
  - Simple functions that return JSX and do no other work
    - Originally the only way of using functional components
  - Complex functions with additional functionality
- Are now functionally equivalent to class components thanks to Hooks
- Are generally preferred to class approach for new development
- Class components still make up the majority of existing apps
  - Are not deprecated and do not need to be rewritten
  - Need to be understood by any React developer
    - The majority of existing code uses classes
- This course will use both approaches

Functional Component Syntax

```
function App() {

/*can do other work here */

  return (
    <div className="App">
      {/*code omitted */}
    </div>
  );
}

export default App;
```

Could also have used =>

The return statement is the equivalent of `render()` in a class component

Simple Functional Components

```
const About = () => (
  <div className="row">
    <h1>Book Reactions</h1>
    <h2>Where you react to books</h2>
  </div>
)

export default About;
```

Could also have used `function` keyword

For classes that only return JSX. The `return` keyword is omitted, and body of the function is the JSX.

Exercise 1.2: Creating a React Component

- In this exercise, you will create a React component and use it from an existing React component
- Please refer to the Exercise Manual

Combining JSX and JavaScript

- JavaScript can be embedded inside JSX
  - Use `{}` to mark a JavaScript section
- Some restrictions on JavaScript constructs
  - No `for` loops or `if` statements
    - Use `.map()` to iterate through arrays
    - Use ternaries inside JSX instead of `if…else` statements
- Multiline `if…else` statements must be resolved *outside* of JSX
  - JSX can be created and assigned to a variable inside an `if` block
    - Variable can then be passed into JSX
- As with JavaScript code, comments must be placed inside `{}`

```
{ /* <Reviews />: removed for now */ }
```

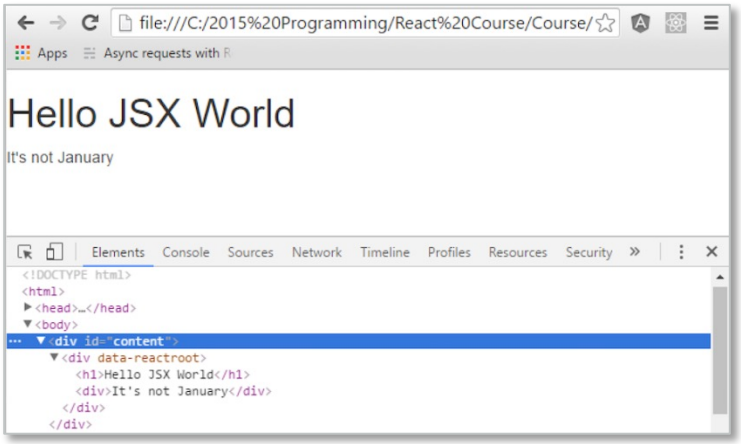
Embedding JavaScript Example

```
var d = new Date();
var myJSX = <h1>Hello JSX World</h1>;
if (d.getMonth() == 0) {
   myJSX = <h1>Happy New Year!</h1>
}
ReactDOM.render(<div>
    {myJSX}
    <p>  d.getMonth() == 0 ?
         "It's January" :
             "It's not January" }</p>
             </div>,
         document.getElementById('content'));
```

`if` statement used to *generate* JSX, not *inside* JSX

Ternary expressions are allowed in JSX

Passing the result of the `if` statement to JSX

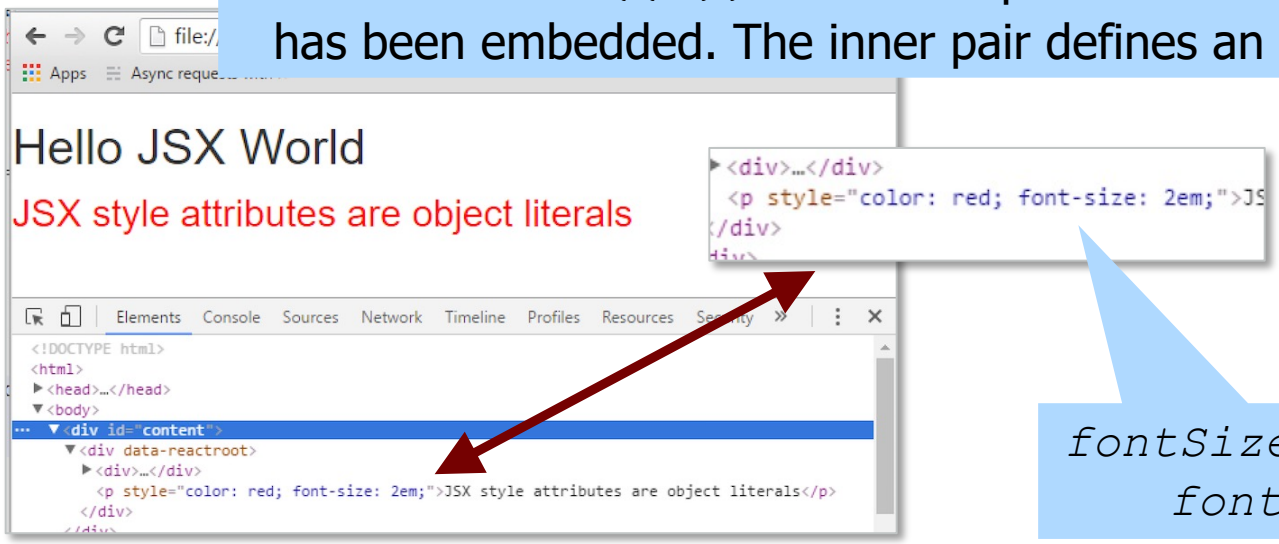


The Style Attribute

- The JSX style attribute expects an object literal
  - Name/value pairs are converted into CSS properties/values
    - Values are strings
  - Hyphenated CSS properties are specified using camel-case

```
<p style={{ color: 'red', fontSize: '2em' }}>
```

Note the double `{{ }}`. The outer pair tell JSX that JavaScript has been embedded. The inner pair defines an object literal.



fontSize generates *font-size*

Attribute Values Reconsidered

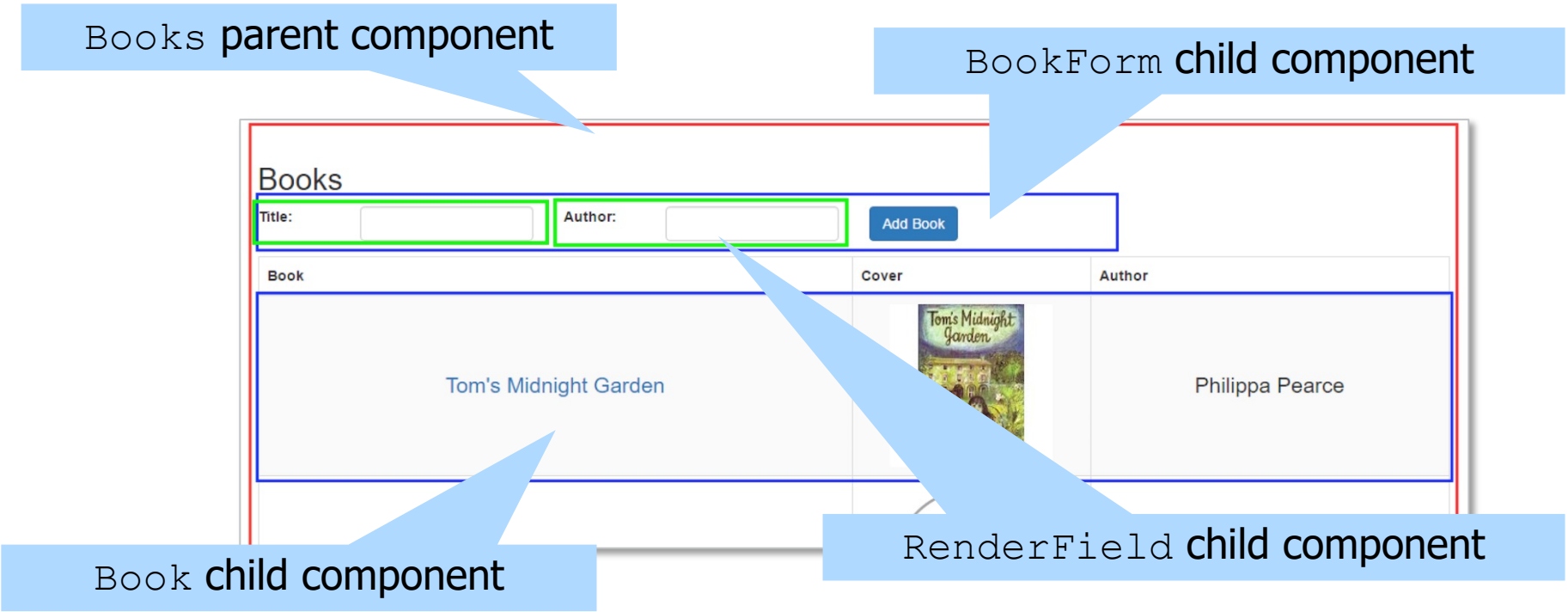
- Passing string values to attributes is straightforward in JSX
  - You did this in the exercise
    - className = "string"
- Booleans, objects, and numeric arguments are handled differently
  - Passed inside curly braces
    - Not as strings

```
<button type="submit" disabled={true}>send</button>
```

Designing for React

- React components should do only one job
  - The `Books` component should contain multiple child *Book* components
    - Not individual book details
  - Simplifies composition
  - Maximizes component reuse

Designing for React Illustrated



Composing Components

```
class Head extends React.Component {
  render() {
    return (<div>
             <h1>Book Reactions</h1>
             <p>Where you React to books</p>
            </div> );
   }
}
```

`Head` component

```
class App extends React.Component {
  render() {
    return (<div className="container">
      <Head />
      <Books />
    </div>);
  }
}
```

`App` component composed from both JSX and child components

Exercise 1.3: Working with JSX

- In this exercise, you will create components using ES6 classes and combine JSX with JavaScript to create an HTML table
- Please refer to the Exercise Manual

Chapter Concepts

Introducing React

Introducing JSX

React Components

Chapter Summary

Chapter Summary

In this chapter, we have:

- Answered the question: what is React?
- Created a 'Hello World' React application
- Composed a React application from components