React JS

July 26, 2018

Class Goals

Basic Components

- Props
- Component hierarchy
- Building the components
- Rendering components

DOM Abstraction

- How React views the Document Object Model
- Event listeners
- JSX vs. HTML
- Rendering dynamic HTML
- Using plain JavaScript
- Element and custom factories
- Form components
- Keys and refs

Architecture with Components

- Prop validation
- Built-in and custom validators
- Component composition strategies
- Lifecycle
- Components that retain state
- Nesting components and passing state
- Immutability

Advanced User Interaction

- Animations
- Drag and drop

Routing

- The React router
- The index router
- Parameters
- Active links
- Properties

Routing (cont'd)

- The user interface and the URL
- Changing routes programmatically

Flux

- Flux methodology
- Stores
- Actions
- Dispatchers
- Using Flux asynchronously

Performance Tuning

- Reconciliation
- Batching and sub-tree rendering
- ReactPerf and the performance test application

Testing Components

- Popular Testing Frameworks: Jest and Jasmine
- What to test
- Rendering components for testing
- Traversing components
- Shallow rendering
- Simulating Events

Isomorphic Applications in Node

- Node.js and Express web services
- Rendering React components on the server
- Mounting on the client
- Internal Routes
- Dynamic Data Fetching
- Rendering Routes

Deployment

- Deploying Babel in production
- Build tools and webpack

Introduction to React JS

https://reactjs.org/blog/2013/06/05/why-react.html

"React is a library for building composable user interfaces. It encourages the creation of reusable UI components which present data that changes over time."

- React JS is a front-end framework, and is oftentimes referred to as simply a library.
- React isn't an MVC framework.
- React doesn't use templates.
- React was created and is still heavily maintained by Facebook.
- Reactive updates are simple.
- React runs on the desktop (React JS), mobile device (React Native), and on the server (Node.js)
- React helps you build front end systems that perform client-side rendering based on data that is received in JSON format from a back end.

Development Software

Node

https://nodejs.org/en/download/

NPM - node package manager

```
npm install -g npm@latest
```

Yarn

https://yarnpkg.com/en/docs/install

Facebook's React extension for Chrome

https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi

Microsoft's Visual Studio Code

https://code.visualstudio.com

Microsoft's Chrome Debugger for VS Code

https://marketplace.visualstudio.com/items?itemName=msjsdiag.debugger-for-chrome

PDF viewer vscode-pdf

https://marketplace.visualstudio.com/items?itemName=tomoki1207.pdf

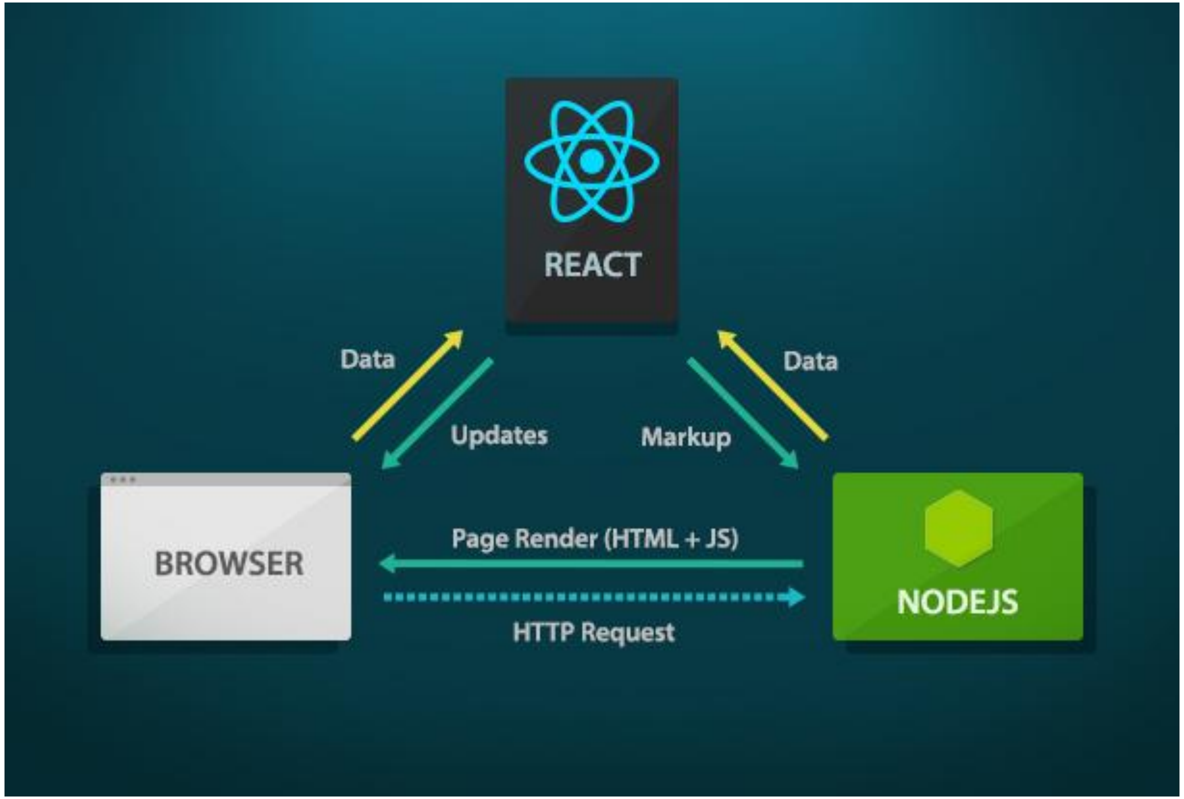
JSX styler vscode-styled-jsx

https://marketplace.visualstudio.com/items?itemName=blanu.vscode-styled-jsx

Live Server

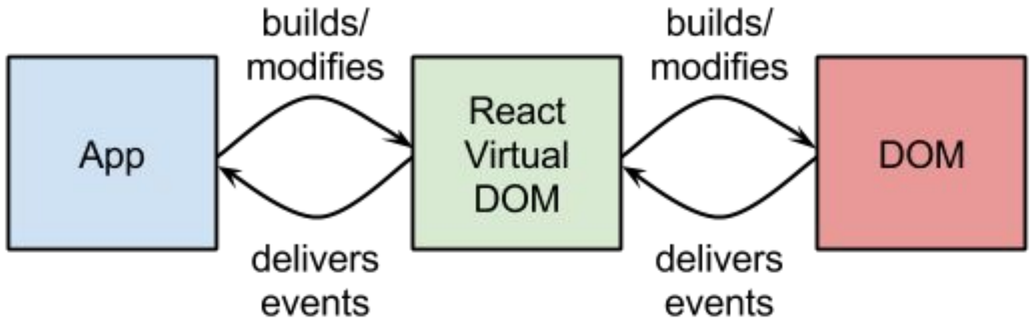
```
npm install -g live-server
```

How React Works



https://github.com/arun-curriculum/React-Two-Day/tree/master/day_1

React’s Virtual DOM



- React receives data from backend system that is then integrated into page view as a part of a "diff".
- React creates a JavaScript representation of the DOM and is then responsible for creating and updating the "real" DOM itself.
- This increases React's performance dramatically. Re-render is fast (around 1ms for TodoMVC), without the need to explicitly specify data bindings.

https://github.com/arun-curriculum/React-Two-Day/tree/master/day_1

Add React to a Website

https://reactjs.org/docs/add-react-to-a-website.html

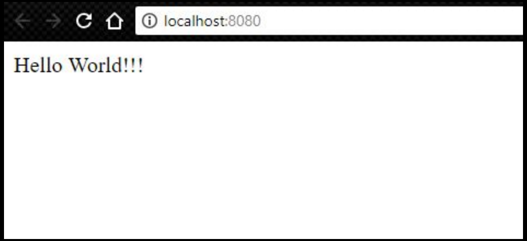
Use as little or as much React as you need.

“React has been designed from the start for gradual adoption, and you can use as little or as much React as you need. Perhaps you only want to add some “sprinkles of interactivity” to an existing page. React components are a great way to do that.

“The majority of websites aren’t, and don’t need to be, single-page apps. With a few lines of code and no build tooling, try React in a small part of your website. You can then either gradually expand its presence, or keep it contained to a few dynamic widgets.”

Exercise 1. Single Page “Hello World”

```
<!DOCTYPE html>
<html lang="en">

<head>
   <title>Hello World Example</title>
</head>

<body>
   <div id="hello-world"></div>
   <script src="https://fb.me/react-15.0.0.js"></script>
   <script src="https://fb.me/react-dom-15.0.0.js"></script>
   <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.34/browser.min.js"></script>

   <script type="text/babel">
      var HelloWorld = React.createClass({
        render: function() {
          return ( <p>Hello World!!!</p> );
        }
      });

      ReactDOM.render( <HelloWorld/>, document.getElementById('hello-world'));

     </script>
</body>
</html>
```



Hello World!!!

Exercise 2. Add React in One Minute

https://gist.github.com/gaearon/6668a1f6986742109c00a581ce704605

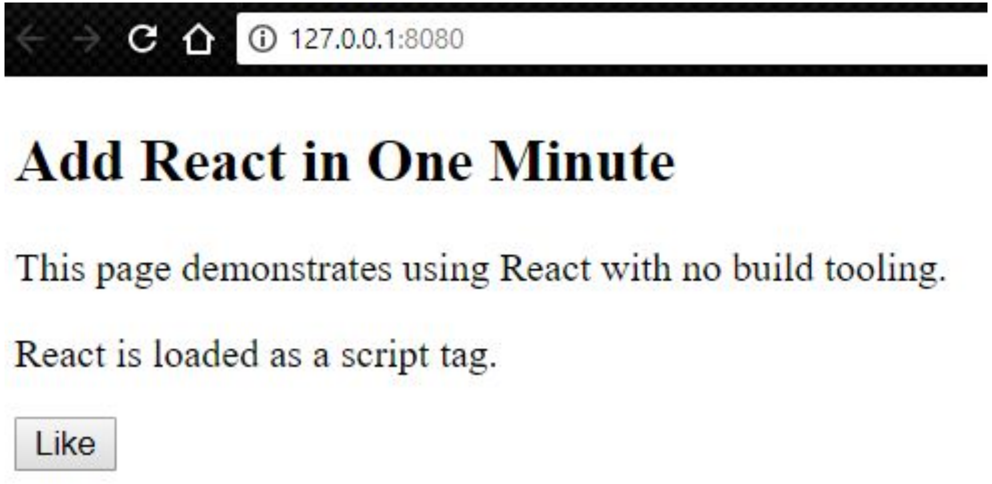
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Add React in One Minute</title>
  </head>
  <body>

    <h2>Add React in One Minute</h2>
    <p>This page demonstrates using React with no build tooling.</p>
    <p>React is loaded as a script tag.</p>

    <!-- We will put our React component inside this div. -->
    <div id="like_button_container"></div>

    <!-- Load React. -->
    <!-- Note: when deploying, replace "development.js" with "production.min.js". -->
    <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>

    <!-- Load our React component. -->
    <script src="like_button.js"></script>

  </body>
</html>
```

Exercise 2. Add React in One Minute, contd.

https://gist.github.com/gaearon/6668a1f6986742109c00a581ce704605

```
'use strict';

const e = React.createElement;

class LikeButton extends React.Component {
  constructor(props) {
    super(props);
    this.state = { liked: false };
  }

  render() {
    if (this.state.liked) {
      return 'You liked this.';
    }

    return e(
      'button',
      { onClick: () => this.setState({ liked: true }) },
      'Like'
    );
  }
}

const domContainer = document.querySelector('#like_button_container');
ReactDOM.render(e(LikeButton), domContainer);
```

A React Website



127.0.0.1:8080

Add React in One Minute

This page demonstrates using React with no build tooling.

React is loaded as a script tag.

Components in React

Functional

- Also known as ‘dumb’ component
- Usually for display purposes only

```
import React from "react";

const HelloWorld = (props) => {
    return (
        <div>Hello World!</div>
    );
}

export default HelloWorld;
```

Class-based

- Also known as ‘smart’ component
- Capable of affecting other components, pull data from APIs, etc.

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

class HelloWorld extends Component {
    render() {
        return (
            <div>Hello World!</div>
        );
    }
}
export default HelloWorld;
```

The key difference is that the smart components can take advantage of React's "lifecycle" methods to create more advanced functionality (more on this later).

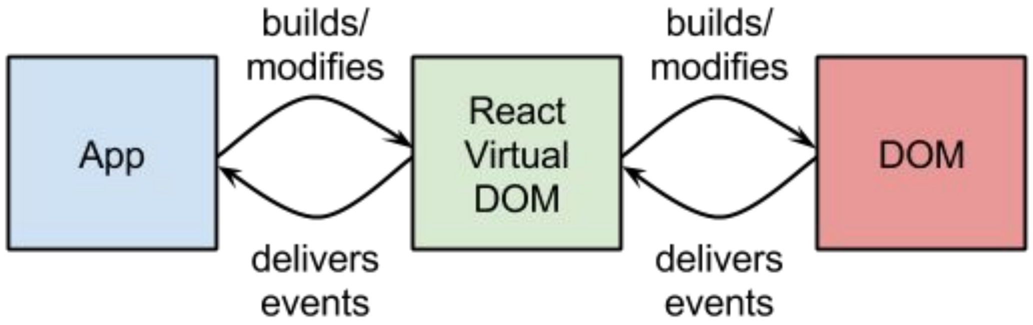
Mounting Components to the "Real" DOM

- All changes to the "DOM" happen to the virtual DOM, so what about what the client actually sees?
- React has a package called "ReactDOM" that handles placing injecting virtual DOM changes into the real DOM.
- This should generally happen in the entry point to your application:

```
import React from "react";
import ReactDOM from "react-dom";

ReactDOM.render(<App />, document.getElementById("root"));
```

- This is the only time in a React application that you will see standard DOM manipulation.

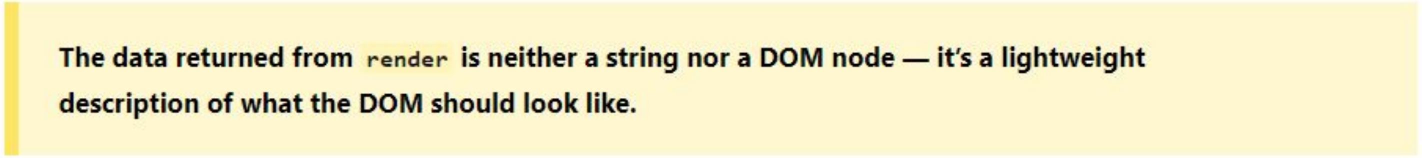
React’s Virtual DOM



- React receives data from backend system that is then integrated into page view as a part of a "diff".
- React creates a JavaScript representation of the DOM and is then responsible for creating and updating the "real" DOM itself.
- This increases React's performance dramatically. Re-render is fast (around 1ms for TodoMVC), without the need to explicitly specify data bindings.

https://github.com/arun-curriculum/React-Two-Day/tree/master/day_1

render method in React JS


The data returned from `render` is neither a string nor a DOM node — it's a lightweight description of what the DOM should look like.

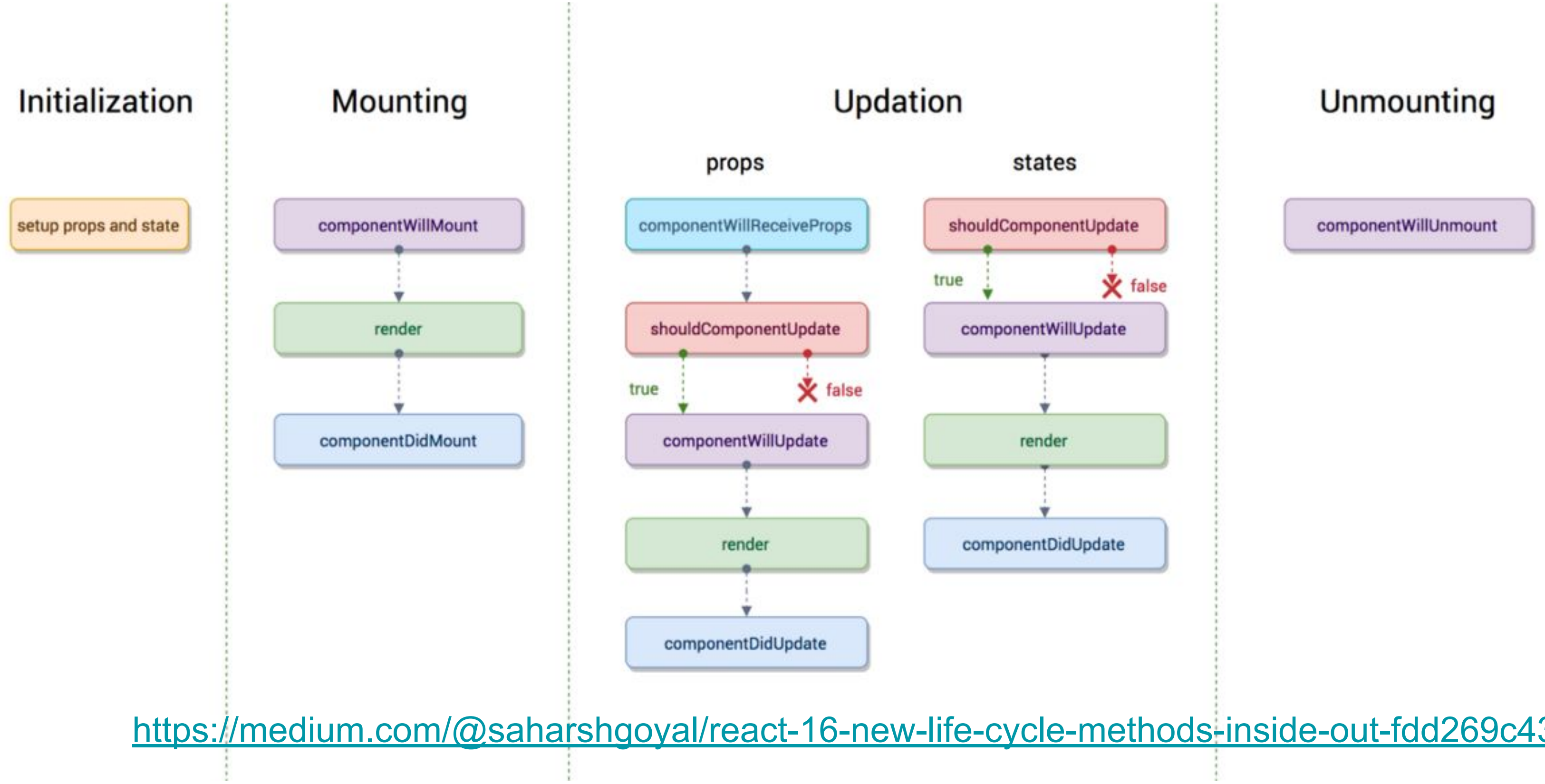
https://reactjs.org/blog/2013/06/05/why-react.html

“When your component is first initialized, the render method is called, generating a lightweight representation of your view. From that representation, a string of markup is produced, and injected into the document. When your data changes, the render method is called again. In order to perform updates as efficiently as possible, we diff the return value from the previous call to render with the new one, and generate a minimal set of changes to be applied to the DOM.
The data returned from render is neither a string nor a DOM node — it's a lightweight description of what the DOM should look like. We call this process reconciliation. Because this re-render is so fast (around 1ms for TodoMVC), the developer doesn't need to explicitly specify data bindings. We've found this approach makes it easier to build apps.”

Component Lifecycle

- Each component goes through a lifecycle from initialization to rendering and beyond.
- Here is some additional information: https://reactjs.org/docs/react-component.html.
- The most commonly used methods are:
    a. componentWillMount
    b. render
    c. componentDidMount
- Lifecycle methods can only be used in class-based components.
- Lifecycle methods are commonly used to perform setup and teardown operations within components, e.g. a popular example is fetching data via AJAX on componentDidMount.

Component Lifecycle



https://medium.com/@saharshgoyal/react-16-new-life-cycle-methods-inside-out-fdd269c43ccd

Create-React-App

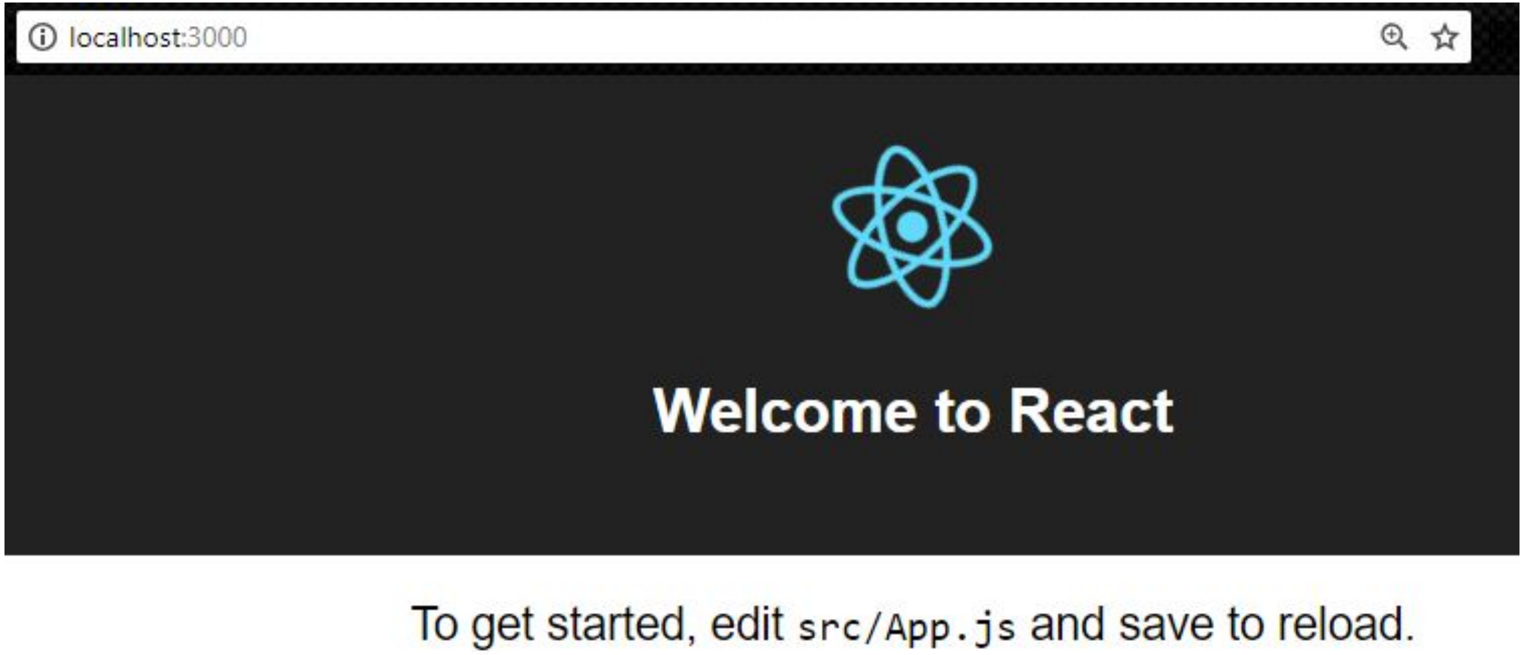
https://github.com/facebook/create-react-app

- React, JSX, ES6, and Flow syntax support.
- Language extras beyond ES6 like the object spread operator.
- Autoprefixed CSS, so you don't need `-webkit-` or other prefixes.
- A fast interactive unit test runner with built-in support for coverage reporting.
- A live development server that warns about common mistakes.
- A build script to bundle JS, CSS, and images for production, with hashes and sourcemaps.
- An offline-first service worker and a web app manifest, meeting all the Progressive Web App criteria.
- Hassle-free updates for the above tools with a single dependency.

Exercise 3. Creating a React App

```
c:/users/files:~$ node -v
v9.11.2
c:/users/files:~$ npm -v
5.6.0
c:/users/files:~$ yarn -v
1.7.0
c:/users/files:~$ npx create-react-app react_app
Creating a new React app in c:/users/files/react_app.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts...
c:/users/files:~$ cd react_app
c:/users/files:~$ npm run eject
c:/users/files:~$ npm install
c:/users/files:~$ npm start
```

Running React App (development mode)


localhost:3000

Welcome to React

To get started, edit `src/App.js` and save to reload.

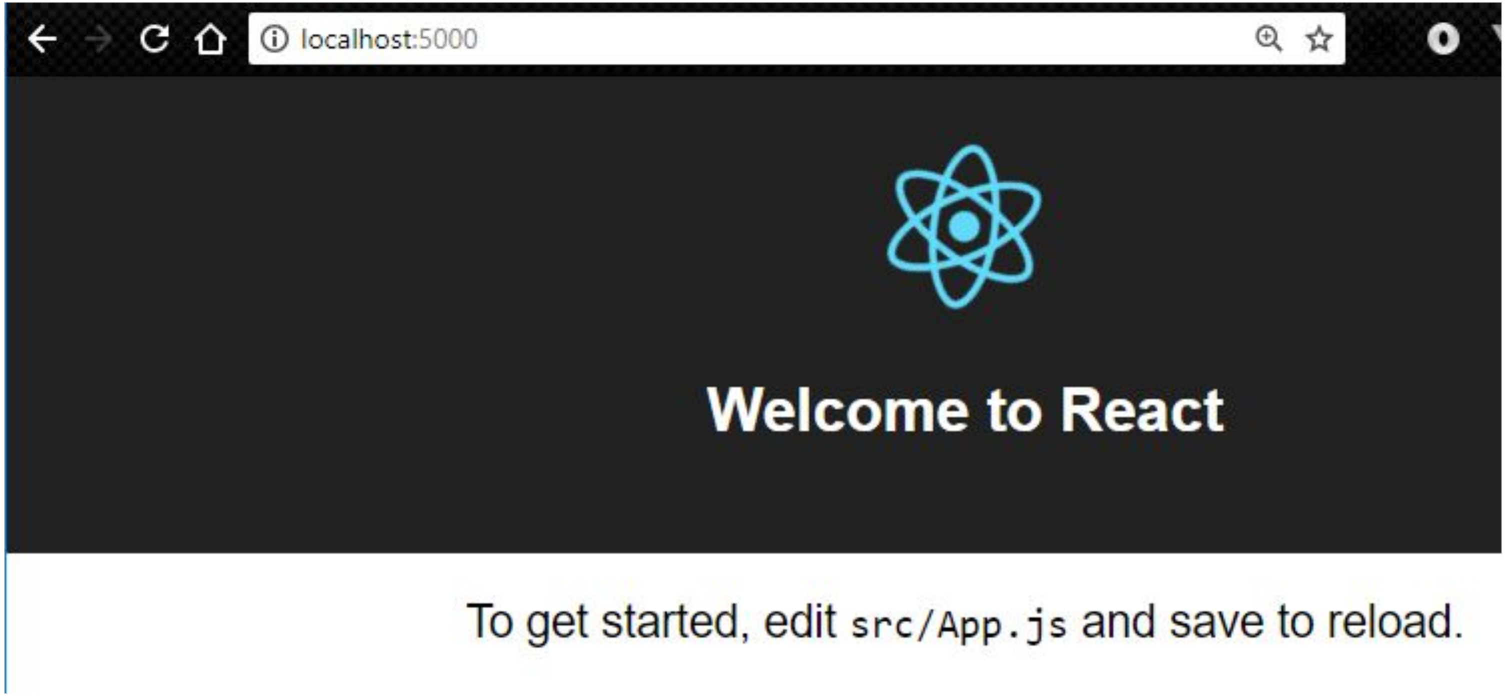
Create React App folder structure

- config - Webpack configuration files and environment data.
- public - Any public assets that are to be served as-is.
- scripts - Script files that are run by "scripts" in package.json to build, test, and start your application.
- src - This is where the main source code of the project will go.
- node_modules - Packages stored here. These are the app’s dependencies
- package.json - Lists the packages that your project depends on
- README.md
- Yarn.lock - Yarn needs to store exactly which versions of each dependency were installed.

Exercise 3. Creating the React App contd.

```
c:/users/files/react_app:~$ ls
config/ node_modules/  package.json  public/  README.md  scripts/ src/   yarn.lock
c:/users/files/react_app:~$ ls src/
App.css  App.js  App.test.js  index.css  index.js  logo.svg  registerServiceWorker.js
c:/users/files/react_app:~$ yarn build
Compiled successfully.
c:/users/files/react_app:~$ ls
build  node_modules/  package.json  public/  README.md  src/  yarn.lock
c:/users/files/react_app:~$ ls build/
asset-manifest.json  favicon.ico  index.html  manifest.json  service-worker.js  static
c:/users/files/react_app:~$ yarn global add serve
success Installed "serve@9.3.0" with binaries
c:/users/files/react_app:~$ serve -s build
Serving!
- Local:              http://localhost:5000
- On Your Network:  http://192.168.0.13:5000
```

Running React App (production build)


localhost:5000

Welcome to React

To get started, edit `src/App.js` and save to reload.

Props and Event Handling

- Each component can have "props" associated with it, which essentially look like HTML attributes.
- Props can pass data to components from data such as strings and objects, to functions.
- Let's see an example with a simple click event:

```
import React, { Component } from "react";
class ClickExample extends Component {
    handleClick() {
        alert("Button Clicked!");
    }
    render() {
        return (
            <button onClick={this.handleClick.bind(this)}>Click Me</button>
        );
    }
}
export default ClickExample;
```

- Here the .bind(this) is used to maintain the class context when the event handler fires. This is very important in React.

Component State

- Every class-based component maintains a "state", which controls how the virtual DOM, and thus the real DOM get updated.

- Every class-based component has a state property that is accessible throughout the entire class, including all lifecycle methods.

- Essentially, changes in state trigger a re-render of the component.

- State can also be initialized in the constructor of the component class.

- State is never mutated directly. Instead, a setter method called .setState() is used.

- The next page has an example.

Example of Component State

Clicking the button will trigger a component refresh.

```
import React, { Component } from "react";
class HelloWorld extends Component {
    constructor() {
        this.state = {greeting: "Hello World!"}
    }
    handleClick() {
        this.setState({greeting: "Goodbye World!"});
    }
    render() {
        return (
            <div>{this.state.greeting}</div>
            <div>
                <button onClick={this.handleClick.bind(this)}>Greet</button>
            </div>
        );
    }
}
export default HelloWorld;
```

Forms in React

- Forms take two forms in React - "controlled" and "non controlled".
- Non controlled forms are those that are blank and accept new data to be submitted.
- Controlled forms rely on existing data to populate the form fields, and the user generally has the option to edit these data. An example is an edit form that is pre-populated with data from an existing object.
- Both types of forms require the component's state to be made aware of changes in the fields.
- This process mimics a two-way data binding system found in other popular frameworks.
- The next page has an example.

Example of Forms in React

```
import React, { Component } from "react";
import update from "immutability-helper";

class SampleForm extends Component {
    constructor() {
        this.state = {
            firstname: "",
            lastname: ""
        };
    }

    handleChange(event) {
        this.setState(update(this.state, {
            $merge: {
                [event.target.name]: event.target.value
            }
        }));
    }
```

Example of Forms in React (cont’d)

```
    handleSubmit(event) {
        event.preventDefault();
        // Handle form submission here
    }

    render() {
        return (
            <form onSubmit={this.handleSubmit.bind(this)}>
                <div>
                    <input onChange={this.handleChange.bind(this)} name="firstname" type="text" />
                </div>
                <div>
                    <input onChange={this.handleChange.bind(this)} name="lastname" type="text" />
                </div>
                <div> <button type="submit">Submit</button> </div>
            </form>
        );
    }
}
export default SampleForm;
```

To JSX or not to JSX?

HTTP Requests with Axios

- Axios is a library that makes working with AJAX requests simple.
- It is promise-based, so we can use the traditional .then() .catch() model of native JavaScript promises.
- The documentation can be found here: https://github.com/axios/axios.
- To use the library we will be first importing it:

```
import axios from "axios";


axios.request({
    url: "endpoint url here",
    method: "GET"
})
.then((response) => {
    console.log(response.data);
})
.catch((err) => {
    console.log(err);
});
```

React Router V4

- Axios is a library that makes working with AJAX requests simple.
- It is promise-based, so we can use the traditional .then() .catch() model of native JavaScript promises.
- The documentation can be found here: https://github.com/axios/axios.
- To use the library we will be first importing it:

```
import { Router, Route, Switch } from "react-router";
import createBrowserHistory from "history/createBrowserHistory";
const history = createBrowserHistory();
<Router history={history}>
    <Switch>
        <Route exact path="/" component={Home} />
    </Switch>
</Router>
```

- Since these routes are just components themselves, they are generally included in a top-level component that is injected into the real DOM by ReactDOM.
- path defines a path to match against for mounting a specific component. You can also use parameters like so:

```
<Route exact path="/wines/:id" component={EditWine} />
```

- These parameters can be accessed by `this.props.match.params` in the appropriate class.
- Note that parameters are only accessible in the component classes that are mounted via the router itself; not any child components.

JavaScript Toolchain

https://reactjs.org/docs/create-a-new-react-app.html

Use an integrated toolchain for the best user and developer experience.

- Scaling to many files and components.
- Using third-party libraries from npm.
- Detecting common mistakes early.
- Live-editing CSS and JS in development.
- Optimizing the output for production.

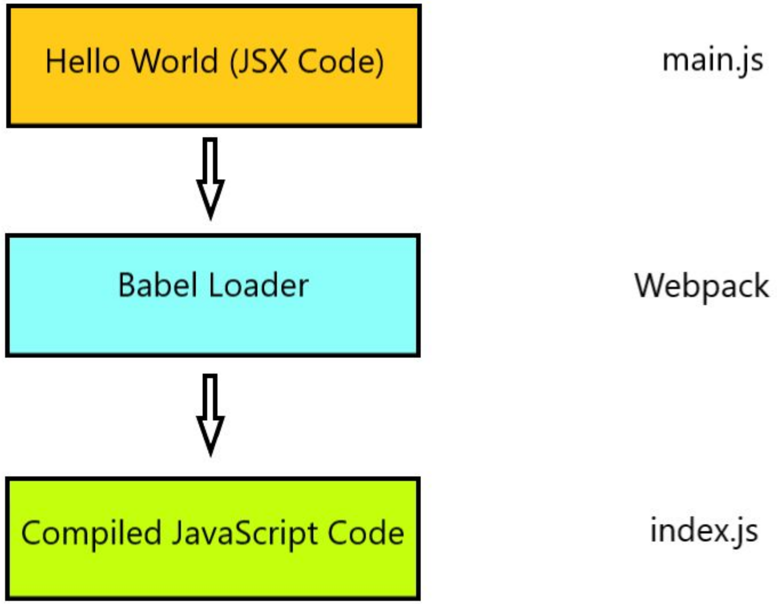
JavaScript Toolchain

https://reactjs.org/docs/create-a-new-react-app.html

A JavaScript build toolchain typically consists of:

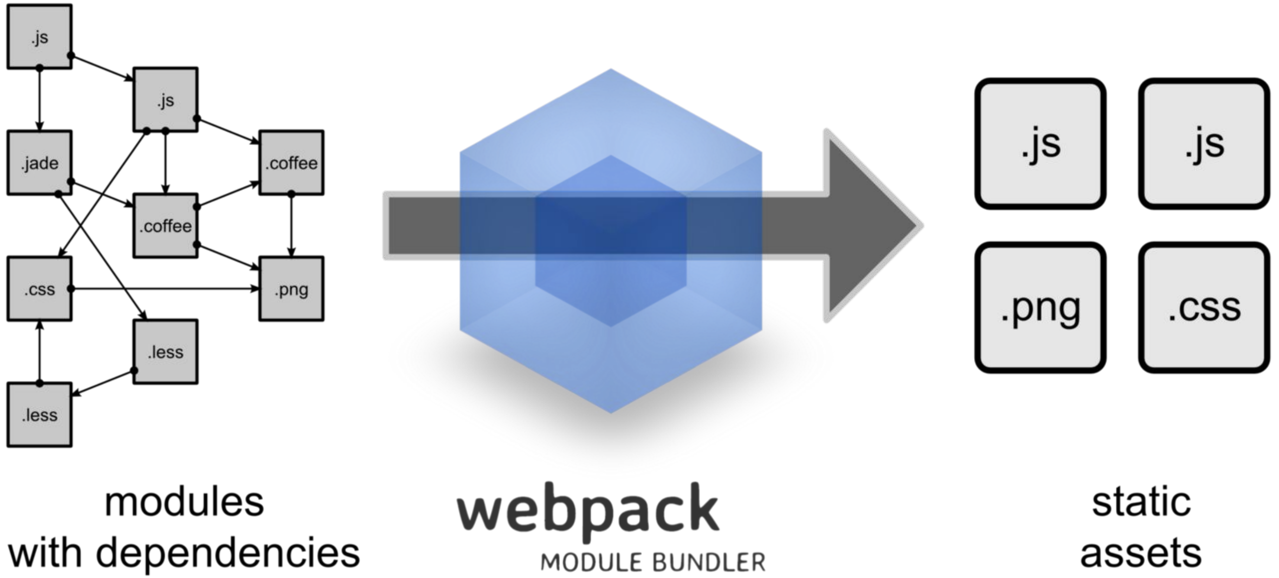
- A package manager, such as **Yarn or npm**. It lets you take advantage of a vast ecosystem of third-party packages, and easily install or update them.
- A bundler, such as **webpack** or Parcel. It lets you write modular code and bundle it together into small packages to optimize load time.
- A compiler such as **Babel**. It lets you write modern JavaScript code that still works in older browsers.

npm start


Hello World (JSX Code)
main.js
Babel Loader
Webpack
Compiled JavaScript Code
index.js

Webpack module bundling

https://medium.com/@phoebe.greig/headache-free-isomorphic-app-tutorial-react-js-react-router-node-js-ssr-with-state-and-es6-797a8d8e493a

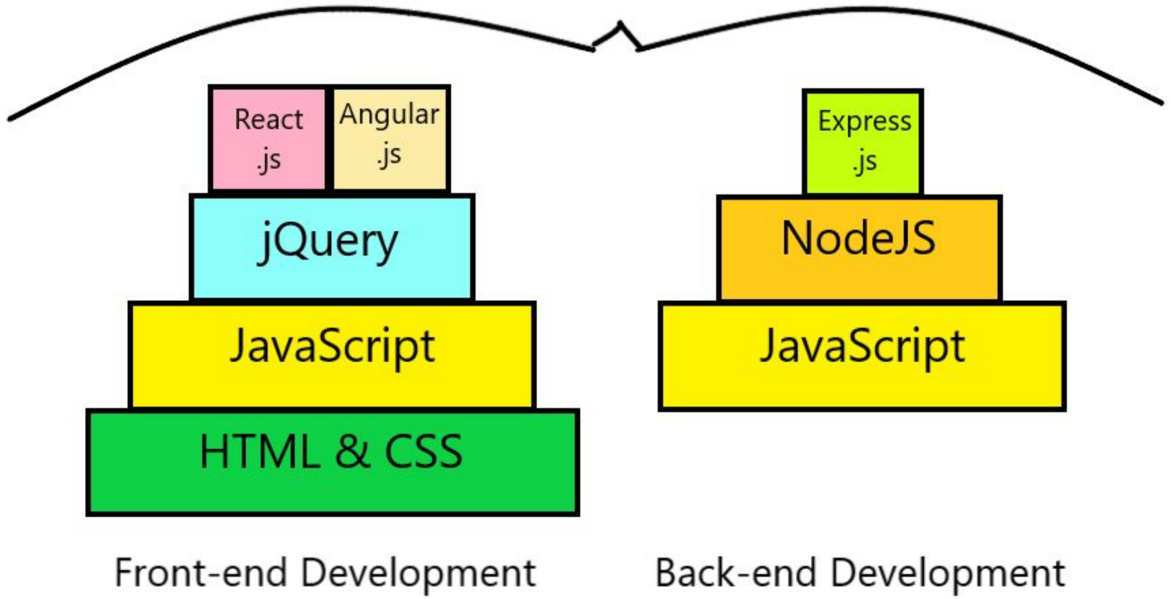


React Browser Tools

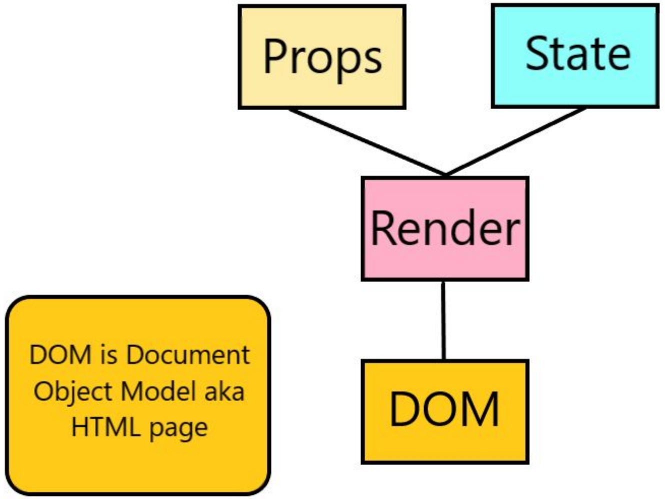
https://reactjs.org/blog/2015/09/02/new-react-developer-tools.html

- Google Chrome
- Mozille Firefox

Full Stack Development



Architecture



Model + component = DOM