INTRODUCING REACT

Agenda

- Creating our first react app
- With/without CLI tools
- Understand the basic flow of the application

Bit Of History

FaxJS

Early prototype of React Created by Jordan Walke

"Our biggest mistake was betting to much on HTML5". React is created internally and decoupled from Facebook code by Pete Hunt

React is open sourced

React developer tools become an extension of Chrome

react-canvas

GraphQL

Relay

React Native

Redux

Mobx

React Storybook

Blueprint

React Fiber

Relicensing

Preact

Getting Started

- Different approaches exist
  - Add React as a plain <script> tag
  - Seed project
  - CLI tools
    - create-react-app
    - Next.js
- Lets start with the first option …

Plain script tag

- Probably the easiest way to integrate React into existing app
- yarn add react react-dom
- Add the UMD modules to the index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <script src="node_modules/react/umd/react.development.js"></script>
    <script src="node_modules/react-dom/umd/react-dom.development.js"></script>
</body>
</html>
```

Container for root Component

- Create an HTML element that serves as the container of the root component

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <div id="app"></div>

  <script src="node_modules/react/umd/react.development.js"></script>
  <script src="node_modules/react-dom/umd/react-dom.development.js"></script>
  <script src="App.js"></script>
  <script src="main.js"></script>
</body>
</html>
```

A Component

- A plain JavaScript class
- Usually extends React.Component
- Implements the render method

```
class App extends React.Component {
   constructor(props) {
      super(props);
   }

   render() {
      return React.createElement("h1", null, "Hello React");
   }
}
```

Will be discussed later

The content of the element

Bootstrapping

- This is where the magic begins
- Tell React to render the root component into the container element

```
ReactDOM.render(React.createElement(App), document.querySelector("#app"));
```

- Take a look at the live DOM

```
<body>
    <div id="app">
        <h1>Hello React</h1>
    </div>
</body>
```

Child Component


```
const e = React.createElement;

class Counter extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      counter: 0,
    };
  }

  render() {
    return e("div", null,
      e("button", {onClick: ()=>this.dec()}, "Dec"),
      e("span", null, this.state.counter),
      e("button", {onClick: ()=>this.inc()}, "Inc")
    );
  }

  dec() {
    this.setState({counter: this.state.counter - 1});
  }

  inc() {
    this.setState({counter: this.state.counter + 1});
  }
}
```

Use the Component


```
class App extends React.Component {
   render() {
      return e("div", null,
         e("h1", null, "Hello React"),
         e(Counter)
      );
   }
}
```

Child component is being used the same as plain HTML element ☺

Data Binding

- A child component may bind to “props” specified by the parent

```
class ContactList extends React.Component {
  render() {
    const contacts = [
      {id:1, name: "Ori"},
      {id:2, name: "Roni"},
    ];

    return e("ul", null, contacts.map(c => e(Contact, {
        contact: c,
        key: c.id,
      }))
    );
  }
}
```

```
class Contact extends React.Component {
  render() {
    const {contact} = this.props;

    return e("li", null,
      e("span", null, contact.name),
    );
  }
}
```

Contact receives this plain object through this.prop

JSX

- Using React.createElement is tedious
- JSX provides syntactic sugar

```
<ul>
  {contacts.map(c => <Contact contact={c} key={c.id} />) }
</ul>
```

- Compiles to

```
e("ul", null, contacts.map(c => e(Contact, {
      contact: c,
      key: c.id,
   }))
);
```

- The “catch” is that we need a compiler …

Babel/Typescript

- Originally Facebook offered their own JSX compiler
  - Now is considered deprecated
- Most of the React world uses
  - Babel
  - Typescript
  - Can even mix them together

Babel

- ES6+ to ES5 compiler
- Supports JSX compilation through a plugins
- yarn add
  - @babel/core
  - @babel/cli
  - @babel/preset-react
- Define babel.config.js file

```
const presets = [
  [
    "@babel/preset-react"
  ]
];

module.exports = { presets };
```

Babel

- Move your code into src folder so you can pass the whole folder to Babel cli

```
node_modules\.bin\babel src --out-dir dist
```

- Since the code is not strictly JavaScript it is common to rename the files to .jsx extension
- Can watch for changes

```
node_modules\.bin\babel src --out-dir dist -w
```

Complete Sample


```
class App extends React.Component {
   render() {
      const contacts = [
         {id:1, name: "Ori"},
         {id:2, name: "Roni"},
      ];

      return <div>
         <h1>Hello React</h1>
         <Counter />
      </div>;
   }
}
```

```
ReactDOM.render(<App />,
document.querySelector("#app"));
```

```
class Counter extends React.Component {
   constructor(props) {
      super(props);

      this.state = {
         counter: 0,
      };
   }

   render() {
      return <div>
         <button onClick={()=>this.dec()}>Dec</button>
         <span>{this.state.counter}</span>
         <button onClick={()=>this.inc()}>Inc</button>
      </div>;
   }

   dec() {
      this.setState({counter: this.state.counter - 1});
   }

   inc() {
      this.setState({counter: this.state.counter + 1});
   }
}
```

Modern Web Development

- ES6 Modules
- Bundling & Minification
- CSS Modules
- SASS
- Typescript
- HMR

- Webpack to rule them all …

Webpack

- Officially its just a module bunlder
- Practically it is a build tool
- A very configuration oriented tool
- Most developers find it unintuitive
- May abstraction tries to hide it
  - create-react-app
  - @angular/cli
- Now you need to master two abstractions …

create-react-app

- Zero configuration abstraction over Webpack
- No config allows for future toolchain upgrade
- Usually is installed as global command

```
yarn global add create-react-app
```

- Create new project

```
create-react-app my-first-app
```

- Run it

```
cd my-first-app
yarn start
```

Production Build

- yarn run build
- Outputs files to the build folder
- Code is bunded and minified
- CSS files are extracted to another bundle
  - Thus allow for parallel download
- Include file hashes (cache buster)
- Include service worker

react-scripts@next

- Next version supports new capabilities
  - SASS
  - CSS Modules
  - Lerna
  - ES6 compatible uglification
  - More …

```
npx create-react-app@next --scripts-version=2.0.1 app-created-with-latest-scripts
```

Adding Typescript support

- We use the same create-react-app but with different plugin

```
yarn create react-app my-app --scripts-version=react-scripts-ts
```

- Use yarn start as usual

yarn eject

- create-react-app hides webpack configuration
- However, deep customization requires playing with Webpack configuration directly
- Think carefully … there is no way back

Custom Build

- This is the probably the best way to go
  - Thinking long term
- However it requires deep understanding of multiple technologies
- Start with Webpack documentation
  - Evolve gradually
  - See my own seed project at https://github.com/oricalvo/react-seed

Summary

- Using create-react-app it very easy to start a new application
- React application consists of many components
- A component is just a class
- JSX is compiled to plain JavaScript