React

Main sources for this training

Code examples:

https://github.com/bhovhannes/trainings/tree/master/react-overview

Links to other used sources will be provided during the training

0.

What is React?

Meet React

What is React ?

React is a JavaScript **library** for creating user interfaces, which offers:

Declarative views

Component-based architecture

Server-side rendering

Native rendering on mobile devices (React Native)

Easy integration with your technology stack

Declarative views

React allows you to design simple views for each state of your application.

And when your data changes, React will efficiently update and render your view.

Declarative views makes your code more predictable and easier to debug.

Component-based architecture

React allows you to create encapsulated components that manage their own state and the compose them to make complex UIs.

"Encapsulated" means that everything related to the component is contained inside it. That includes components template, css styles, js code, etc.

Server-side rendering

React components can be rendered server-side to HTML string and then sent back to the browser.

After that, your application will continue just from that point.

That means end-user will not see any loading indicators for loading UI. They will see the UI as soon as HTML loads and UI will become interactive as soon as app loads.

Native rendering on mobile devices

React Native project provides a set of components which can be used to create UIs for mobile devices.

With React Native, you don't build a "mobile web app", an "HTML5 app", or a "hybrid app". You build a real mobile app that's indistinguishable from an app built using Objective-C or Java.

React Native uses the same fundamental UI building blocks as regular iOS and Android apps.

You just put those building blocks together using JavaScript and React.

Easy integration with your technology stack

React does only one thing - rendering.

That makes easy to integrate React with your existing technology stack.

React plays nicely with all other frameworks - Angular, etc.

1.

About React

Some facts

History and Stats

React was created in Facebook by Jordan Walke in 2011.

React was open-sourced at JSConf US in May 2013.

Because of its simple API and performance, community quickly adopted React, making it the most popular GitHub project of all times.

⊙ Watch ▾	4,619	★ Star	69,465	⑂ Fork	12,939

Angular stats for comparison (https://github.com/angular/angular)

⊙ Watch ▾	2,527	★ Star	25,165	⑂ Fork	6,423

Battle-tested in production

First React was used in desktop version of Instagram.

After success it has made its way to facebook.com.

Currently React Native powers facebook and instagram mobile apps.

Many other web sites and companies also use React, including AirBnb, BBC, IMDb, Walmart, Netflix, etc.

https://github.com/facebook/react/wiki/sites-using-react

2.

JSX

Allows to write
HTML directly in JS

Popularized by React

```
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}

ReactDOM.render(<HelloMessage name="Jane" />, mountNode);
```
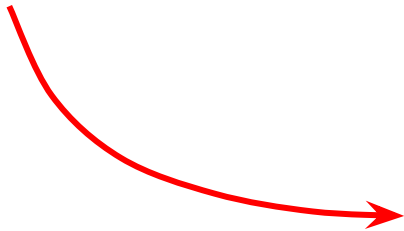
JSX adds XML syntax to JavaScript.

JSX

JSX is optional and not required to use React. It just adds a preprocessor step.

```
class HelloMessage extends React.Component {
  render() {
    return <div>Hello {this.props.name}</div>;
  }
}
```

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
}
```

JSX pragma

JSX is a separate standard.

It has its own spec and is not tied to React.

Different implementations provide different JSX pragmas.

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
}
```

JSX pragma

Converting JSX

Both Babel and TypeScript can transpile JSX to JS.

If webpack is used, babel-loader and ts-loader will do the trick.

You may specify JSX pragma using loader options.

JSX allows you to use JS to write XML

```
class HelloMessage extends React.Component {
  render() {
    return (
        <div>
            {[1,2,3].map((i) => {
                return <div>Hello {i}</div>
            })}
        </div>
    );
  }
}
```

Renders

Hello 1
Hello 2
Hello 3

3.

Virtual DOM

… or why React is fast?

Reasons to have "another" DOM

Real DOM is slow.

Every browser operation with DOM is slow. Because:

1) each DOM node is a heavy data structure holding a lot of attributes (to match spec).

2) altering DOM node may cause browser repaint or even worse - relayout. Both are time consuming operations.

What is virtual DOM?

A virtual DOM tree is a Javascript data structure that describes a DOM tree.

It consists of nested virtual DOM nodes, also known as *vnodes*.

Technically, altering virtual DOM node is as fast as assigning a new value to variable.

Thus, virtual DOM is very fast.

How virtual DOM works?

Let's say you want to make some updates to real DOM:

1) setAttribute("disabled", "disabled")

2) … . appendChild(…)

3) removeAttribute("data-test-id")

You already have a snapshot of real DOM as a virtual DOM structure.

a) you make all (1-3) changes in virtual DOM data structure

b) after that you diff new virtual DOM with the previous one to get a patch with all changes

c) you apply patch to the real DOM

How virtual DOM works?

So, we have the following cycle:

    a) do changes   ->   b) compute diff   ->   c) apply patch
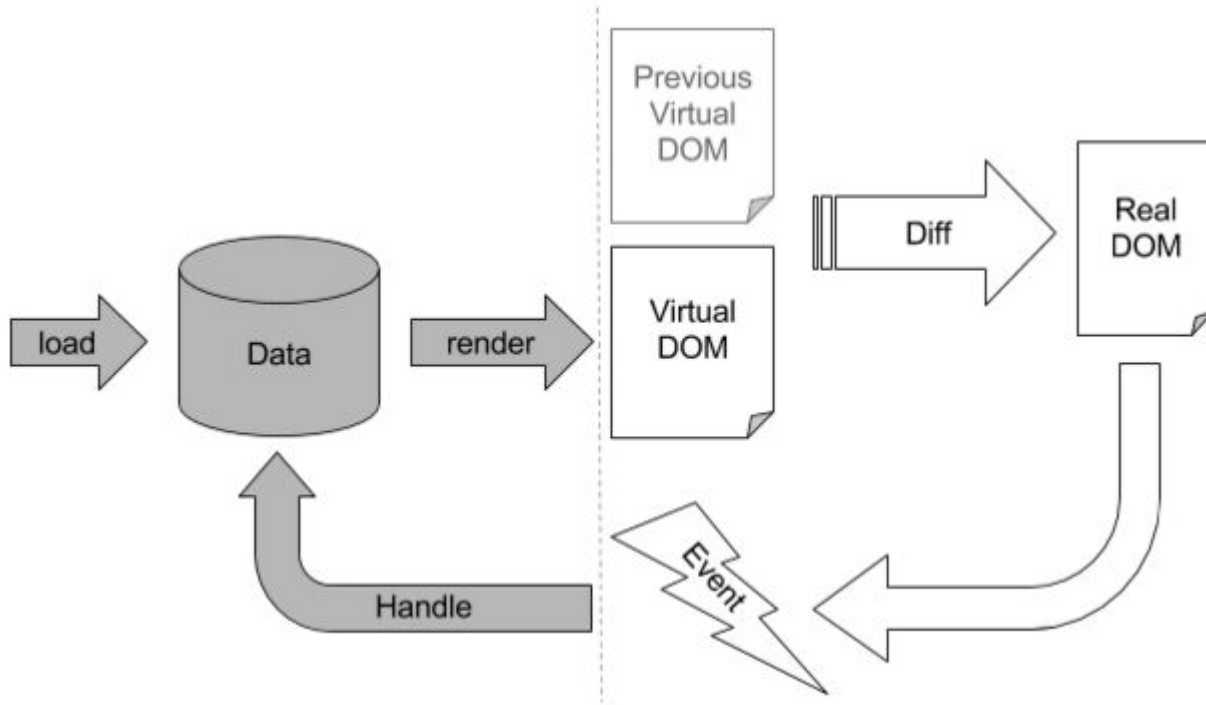
(a) is very fast

(b) also is fast because it works on 2 data structures fully in memory without accessing real DOM

(c) is the only point where we touch to real DOM

Summarizing, we did 1 DOM operation instead of 3.

When you frequently change DOM you'll get a large speed boost.

How virtual DOM works?

Who uses Virtual DOM approach ?

React

virtual-dom (https://github.com/Matt-Esch/virtual-dom)

Vue.js (https://vuejs.org)

Mithril.js (https://mithril.js.org/)

and more …

4.

React components

Building blocks

Components and Props

A component is a building piece in React.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Conceptually, components are like JavaScript functions.

They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.

Functional components

This function is a valid React component because it accepts a single "props" object argument with data and returns a React element.

Such components are called "functional" because they are literally JavaScript functions.

All React components must act like pure functions with respect to their props. That is, props are read-only.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Composing components

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

```
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

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

Class components

Components can also be defined as ES6 class.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

The component written above, defined as ES6 class.

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Class components

Class components allow to:

    1) Define internal **component state**

    2) Make use of **component lifecycle methods**

These 2 things are impossible with functional components.

Component state

"State" is a data internal to the given component.

Component may return different output based on its props and state.

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

Component lifecycle methods

Let's say we want to set up a timer whenever our Clock component is rendered to the DOM for the first time ("mounting").

We also want to clear that timer whenever the DOM produced by the Clock is removed ("unmounting").

These special methods on the component class allow to run code when a component mounts and unmounts:

      componentDidMount()

      componentWillUnmount()

4.

Styling in React

Applying CSS

Inline styles out-of-the-box

```
const divStyle = {
  color: 'blue',
  backgroundImage: 'url(' + imgUrl + ')',
};

function HelloWorldComponent() {
  return <div style={divStyle}>Hello World!</div>;
}
```

Aphrodite

https://github.com/Khan/aphrodite

```
1   import React, { Component } from 'react';
2   import { StyleSheet, css } from 'aphrodite';
3
4   class App extends Component {
5       render() {
6           return <div>
7               <span className={css(styles.red)}>
8                   This is red.
9               </span>
10              <span className={css(styles.hover)}>
11                  This turns red on hover.
12              </span>
13          </div>;
14      }
15  }
16
17  const styles = StyleSheet.create({
18      red: {
19          backgroundColor: 'red'
20      },
21      hover: {
22          ':hover': {
23              backgroundColor: 'red'
24          }
25      }
26  });
```

Aphrodite

https://github.com/Khan/aphrodite

Aphrodite makes your styles unique so CSS styles of your component just cannot modify look of other components.

It works by applying CSS styles as inline styles.

It also supports server-side rendering.

styled-components

GitHub   Documentation

```
const Button = styled.a`
  display: inline-block;
  border-radius: 3px;
  padding: 0.5rem 0;
  margin: 0.5rem 1rem;
  width: 11rem;
  color: white;
  border: 2px solid white;

  /* The GitHub button is a primary button */
  ${props => props.primary && css`
    background: white;
    color: palevioletred;
  `}
`
```

styled-components

styled-components works by generating unique CSS class names and passing them to your component via props.

Uniqueness of class names guarantees that styles do not leak outside of your component thus providing full encapsulation of styles.

CREDITS

Special thanks to all the people who made and released these awesome resources for free:

- ▸ Simple line icons by Mirko Monti
- ▸ E-commerce icons by Virgil Pana
- ▸ Streamline iconset by Webalys
- ▸ Presentation template by SlidesCarnival