REACT - DAY TWO

INTRODUCTION

Bartosz Cytrowski — infoShare Academy

WHAT IS REACT?

REACT IS...

a component-based library focused on making the user interface in a declarative way.

You can learn more about it on https://reactjs.org/

WHAT IS THE DIFFERENCE BETWEEN A LIBRARY AND A FRAMEWORK?

People have different definitions for **library** and **framework**

One definition I know is:

- **framework** is a software where you plug your code into
- **library** is a software that you plug into your code

In terms of this definition, React is a framework.

But some people, especially in the front-end world, say a framework has to bring stuff like routers and/or widgets etc.

So **Angular, Ember.js and ExtJS are frameworks**, but **React isn't**, because it only gives you the means to build components and render them on the screen.

And to be clear - **Facebook itself defines React as a library**.

WHY DO WE NEED REACT?
ISN'T VANILLA JS GOOD ENOUGH?

REACT IS STRONGLY FOCUSED ON USING VANILLA JS BUT IT GIVES US A SIMPLE ABSTRACTION TO SOLVE COMMON UI RENDERING PROBLEMS.

Just keep in mind the code you need to build some UI in the browser in an efficient way using the browser DOM API.

```
const divNode = document.createElement('div');
divNode.classList.add('container');
```

Now with **React**, using **JSX**:

```
const divNode = <div className="container" />
```

Simple, isn't it?

WHAT IS JSX?

It means *JavaScript and XML* and it's a special kind of syntax which extends the basic expressiveness of JS, eg.

```
const divNode = <div className="container" />
```

It is being transpiled to pure JS by **babel** so the browser gets something like this:

```
const divNode = React.createElement(
  'div',
  { className: 'container' }
)
```

We just do not have to write it manually - but we can if we want.

If you want to learn more about **babel** go to the docs: https://babeljs.io/docs/en/

One can say that we can achieve similar brevity in vanilla JS:

```
const divNode = `<div className="container" />`
```

However the resulting value is not even a DOM Node - it's simply a `string`. You need to make some tricks to convert it into the Node:

```
const tmpNode = document.createElement('div');
tmpNode.innerHTML = `<div className="container" />`
const divNode = tmpNode.firstElementChild;
```

VIRTUAL DOM

React does its tricks too - the `React.createElement` function does not make a true `DOM` node. It produces intermediate value named `Virtual DOM Node`.

You should read about it here: https://reactjs.org/docs/faq-internals.html#what-is-the-virtual-dom

To make it short:

Virtual DOM is an object describing how the real browser DOM should look like.

In the past DOM modifications in the browser were extremely slow comparing to the JS performance so the React team decided to calculate all potential modifications within Virtual DOM and let React apply those to the DOM in an efficient way.

Right now the browser DOM performance is improved but the pattern React team used remains.

WHEN I WANT TO USE REACT?

When you are building an application that modifies its UI intensively then you can get into some unpredictable states very easily - just imagine having a bunch of functions adding DOM nodes, some event listeners removing them and an interval in between pending AJAX requests.

It can be written in a good , maintainable way - but it is **HARD**.

React *brings structure and conventions to help us out and make our code easier to reason about.*

React focuses on small composable elements named `components`. You can think of one as of function:

```
function PageHeader() {
  return <h1>Welcome in our App</h1>
}
```

Components can be used together, eg.

```
function PageHeader() {
  return <h1>Welcome in our App</h1>
}

function App() {
  return (
    <div>
      <PageHeader />
      <p>We are building something awesome...</p>
    </div>
  )
}
```

Notice the UpperCamelCase notation - it is require for React to know that you want to use a component instead of making an HTML tag named `PageHeader`. Imagine having a component named `Div` - there has to be a convention for that.

IF A COMPONENT IS A FUNCTION THEN HOW CAN I CALL IT WITH SOME ARGUMENTS?

The thing is **we do not call those function by ourselves**. React uses our definitions and calls them when it needs.

Let's see how it all starts:

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

This is the main entry point to the React application. We use a special `react-dom` library here which is dedicated for working with DOM in the browser.

First argument of `ReactDOM.render` is a React Element created by `React.createElement`. However we use JSX syntax here so we do not have to write the whole call by hand.

It tells React what is the definition of the main component we want to render as the whole application.

Usually **we need to have only one** `ReactDOM.render` call.

What about passing arguments to our components?

In React we can pass arguments as an object - the docs name this object **props**. Take a look at the example

```
function PageHeader(props) {
  return <h1>Hello {props.username}</h1>
}

function App() {
  return (
    <div>
      <PageHeader username="John" />
      <p>We are building something awesome...</p>
    </div>
  )
}
```

The result of the code you see above will look like...

...this:

```
<div>
  <h1>Hello John</h1>
  <p>We are building something awesome...</p>
</div>
```

What if we want to pass multiple arguments? We need to define our component in a way it handlers multiple arguments:

```
function PageHeader(props) {
  return (
    <h1>
      Hello {props.username} <span>{props.email}</span>
    </h1>
  )
}
```

And then use its definition within JSX like that:

```
<PageHeader username="John" email="john@doe.com" />
```

Let's summarize:

1. React helps us build UI in a declarative way (using expressions).
2. In the browser we use `react-dom` and `react` libraries. First one handles DOM modifications, second has a core React functions.
3. We use JSX do describe how the DOM should look like when React renders it.
4. We use **props** object to pass arguments into components. With JSX we can pass those as attributes on JSX tags.