REACT WORKSHOP

QUICK INTRO, WHO WE ARE

AND EXPECTATIONS

Sebastian Sîrb

Robert Pop

Grațian Muntean

AGENDA

- We'll take a quick walk across some of the main concepts in React. We'll have exercises to practice some of these concepts, starting from really simple ones and moving onto some slightly more advanced towards the end.
- We won't spend too much time on the theory, we'll be a bit more focused on the coding exercises

- Great if you don't know React or you just tried it a bit, but do know JavaScript (and ES6)
- Not that great if you already use React on a daily basis and have experience - you'll probably get bored :) - or if you don't really know JavaScript / ES6 - you might get a bit lost :) -

- Regardless, you're more than welcome to stay, just an FYI about the previous skills suggested

Rough Schedule:

- 9.30-14.00, with a 20-30 min break around 12

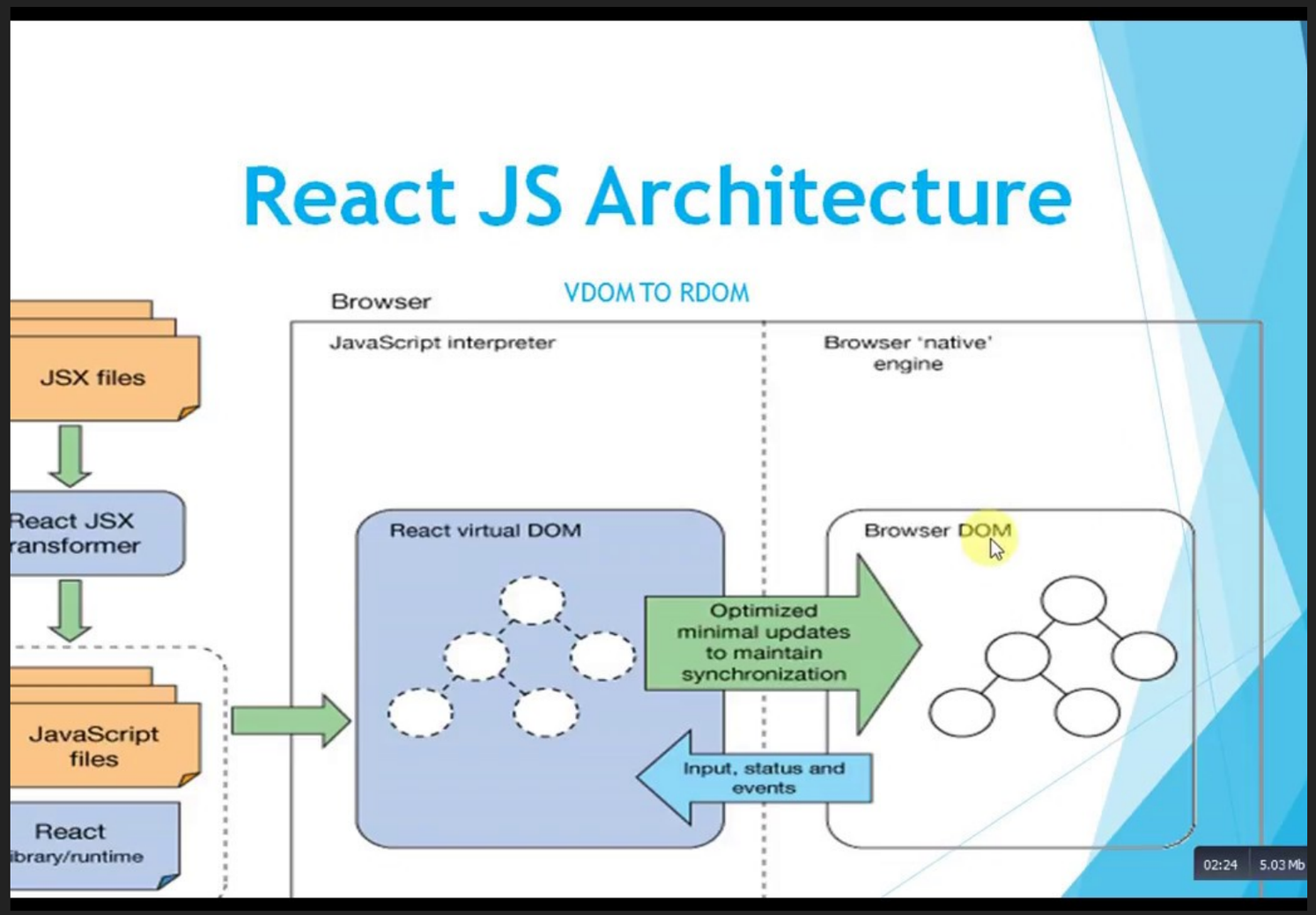
WHAT IS REACT?

- the most popular JavaScript library for building user-interfaces
- created & maintained by Facebook
- open-sourced in 2013
- was a major paradigm-shift in the front-end community a few years ago
- fairly un-opinionated, meaning you have more freedom over how to do things, which library to use for what etc, as opposed to some popular frameworks (i.e. Angular)


REACTJS
REACTJS EVERYWHERE

SOME THINGS ABOUT REACT

- Component based architecture (everything is a component)
- One-way data-flow
- Virtual DOM



JSX

- JSX may look like html (or maybe xml), but it's really not.
- Babel compiles JSX down to React.createElement() calls.
- The two variables below are identical:

```
const jsxElement = (
    <h1 className="greeting">
        Hello, world!
    </h1>
);
const transpiled = React.createElement(
    'h1',
    {className: 'greeting'},
    'Hello, world!'
);
```

TEXT

A QUICK WORD ABOUT STYLING

There are multiple ways to style ui components in React:

- inline-styles
- external css stylesheet
- css modules
- styled components

Today we'll see examples of the first two approaches, but mostly we'll focus on the JS/React part, not that much on styling.

- Head on to https://github.com/react-cluj/workshop-react , and fork the repo. Clone it so you have it on your machine as well.
- Once you've done that, make sure to run yarn at the root level of the project, to install the dependencies
- Then run yarn start

WORKSHOP STRUCTURE

The repo has a few branches:

- master /* beginning, has ex01 in it */
- ex02 /* has the solution for ex01 and the assignment for ex02 */
- ex03 /* has the solution for ex02 and the assignment for ex03 */
- ex04 /* has the solution for ex03 and the assignment for ex04 */
- ex05 /* has the solution for ex04 and the assignment for ex05 */
- ex06 /* has the solution for ex05 and the assignment for ex06 */
- final /* has the solution for ex06 */

- For each exercise, we'll have the following setup: talk a few minutes about the main things in that exercise (ex: what is state and how to use it), give the task and wait a few minutes -no more than 10 -for you to try and solve it. You may google certain stuff or ask us about the concept presented, however to get the most out of this you should try and solve it yourself first, not just wait for the solution
- The solution may not always be the only one, there's multiple ways to do it, it's just what I thought of as being a good solution - you might find a better approach

EX 01

PROPS

- Props are React's way of passing data/actions between components.
- A core principle of React is one-way data-flow. The data flows down from parent components to child components via props.
- Parent components also pass down callback functions via props.
- Then, when an event occurs in a child-component, the child component calls the function it receives.
- Notice, the logic is NOT being dictated by the child components (which happens in 2-way data-binding), child components merely announce the parent component that a certain event happened. The logic is dictated by the parent.

PROPS – EXAMPLE

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

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

EX 02

*you can checkout branch to ex02

FYI – CLASS VS FUNCTIONAL COMPONENTS WITH HOOKS

- Hooks were introduced in late 2018, and are increasingly becoming more and more the “de-facto” way to create a Component.
- Not to be confused with lifecycle hooks in class components
- You may see many examples on the internet using class components - they’re still relevant, though in time, might become less and less so. Thus, we’ll do the exercises in this workshop using hooks and functional components.
- https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108

STATE

- while **props** are React's way of passing data between components, **state** is the way of managing a component's own data.

- **props are immutable**, meaning that a child cannot change its parent data just because it received it as prop, whereas state is meant to be mutated inside of the component

- however, we NEVER mutate state directly (i.e. this.state = something), rather we use use the built-in **useState** hook - or **setState** in class components -

STATE

- State should typically be used for data that will be altered inside of that component (for instance, a form with input fields; you'll probably want to keep the state of the inputs, and update the state as they change).

- If it's not data you expect to change from that component, rely just on props -remember the button component we did earlier-

- suggested read on props vs state: https://github.com/uberVU/react-guide/blob/master/props-vs-state.md

STATE - EXAMPLE

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

QUICK BREAK (~20 MIN)

EX 03

*you can checkout branch to ex03

MAPPING OVER AN ARRAY AND RETURNING SOME JSX FOR EACH ELEMENT

- Very often in our front-end endeavors we need to iterate over an array of some data.. and display something for each element; In React the most common way to do this is to map over the given array and simply return some jsx with the relevant data, for each element

- Example of mapping over an array of users (notice the key as well)

```
renderUsersList = (users) => (
    <ul>
        {
            users.map((user, index) => (
                <li key={index}>
                    {user.name}
                </li>
            ))
        }
    </ul>
);
```

EX 04

*you can checkout branch to ex04

useEffect

- you might have heard about pure and impure functions before; ideally, we would want to have as few side effects as possible in most of our components, meaning they simply receive some props and display data based on these props; obviously that's not always possible, hence we have useEffect.

- useEffect comes in handy whenever we want to create side effects such as changing a global variable or local storage, making a network call and so on

- example: https://reactjs.org/docs/hooks-effect.html

QUICK BREAK

EX 05

*you can checkout branch to ex05

CONTEXT

- Context is very useful when dealing with that can be considered “global” for a React application, or part of the application. Common examples would be themes (style), switching language or having an authenticated user. What’s great about it is not having to pass certain props several times through multiple components just to use them once somewhere down the line.

- useContext is a hook that allows consuming context that was given by a provider

- Example: https://reactjs.org/docs/hooks-reference.html#usecontext

EX 06

*you can checkout branch to ex06

CUSTOM HOOKS

- React also allows building custom hooks to reuse logic. They are simply wrapper functions over the existing hooks. The convention is that for a custom hook component, the name should also start with *use* (similar to the built-in hooks)

- Example: https://usehooks.com/useLocalStorage/

Q&A

THANK YOU!