React.js

Alen Murtić

Sofascore

TABLE OF CONTENTS

Sofascore

01

React
background

React background
What is React?

- JavaScript library for building user interfaces (UI)

- Open-source project by Facebook (after few years of internal usage)

- Declarative

    - Write how the component should look and how it should behave

    - Complex UIs are easy to create

- Component-Based

    - Encapsulate logic and presentation in reusable components

- Learn Once, Write Anywhere - runs wherever JS runs

    - Web Apps (client- and server-side), React Native on mobile devices

- Almost always used with JSX

Sofascore

React background
JSX

- Syntax extension to JavaScript

    - allows JS to understand HTML elements (implemented as objects)

- Template language with full JS power

- Usage with React is optional, i.e. React can be used without it? But why would you do that?

    - Preact.js - not really React, but it's without JSX - because of performance edges

- JavaScript expressions can be embedded inside JSX with {} brackets

- JSX Elements are basic building blocks

    - Elements are used to build components

- .jsx file extension

- Describes how UI should look

Sofascore

React background

Basic JSX example

```
const greet = (
  <h1>
    <span className={isFancy ? 'fancy' : 'regular'}>
      Hello
    </span>
    {name}
  </h1>
)
```

- Note the ( ) brackets

- Attributes are camelCase in JSX.

  - className instead of class, tabIndex instead of tab-index, …

  - class is a reserved keyword in JS so we use className to add CSS classes to the element.

- Any JS expression can be embedded into JSX. Eg. {name}, {2+4}, {person.name}, {getScore()}

- JSX is JS expression -> can be stored in variables, used in JSX, …

Sofascore

React background
Vite.js

- A way to setup React application with predefined configuration

- https://vitejs.dev/

- yarn create vite [app_name] --template react-ts

    - Creates a new React project with types

    - tsconfig.json file in which you define compiler rules

    - Your TS code is transpiled into desired version of JS code

Sofascore

React background

Create React App - deprecated, but still working

- A way to setup React application with predefined configuration

- https://create-react-app.dev/

- yarn create react-app [app-name] --template typescript

    - Creates a new React project with types

    - tsconfig.json file in which you define compiler rules

    - Your TS code is transpiled into desired version of JS code

Sofascore

02

Components

Components
React components

- Encapsulate behavior and presentation

- Basic block of component composition

- Always capitalised (e.g. Counter not counter)

- Translate state and props into JSX (markup)

    - Props - properties a component receives

    - State - internal data that can be passed to children

- Should return only one parent element - i.e. SINGLE TOP ELEMENT

    - Multiple elements should be wrapped in fragments <> and </>, i.e. one parent element

    - Can also return null, false, string, array

Sofascore

Components

Encapsulation and composition

- Two key principles of React and writing good code

- Encapsulation

    - "hide" as much logic as possible in separate components

    - Achieve separation of concerns

    - Code is easier to understand

- Component composition

    - Separate logic and presentation into components

    - Use components as building blocks of more complex features -> code reuse

Sofascore

Components

Functional components

- Initial idea: Function: (props) => Markup

    - Like in math, e.g. f(x) = x + 2

```
function Text() {
  return <p>This is our first React component</p>
}
```

Sofascore

Components
Props

- Used to pass informations into components

    - Can be values, Functions, Objects

- Read-only - **<u>Do not modify props</u>**

- When props change, component updates

- Passed as JSX attributes

    - Try to have generic props, i.e. avoid *isRed*, *isBlue*, use *color* prop

- In Typescript, type of props is also specified

Sofascore

Components
Props

Components
Children

- Component can have child components

    - Pass to it via props

- To specify that component has children, we use PropsWithChildren*<OtherPropsType>* as component's props type

- The most important way of component composition

Sofascore

Components
Styling JSX Elements

- We will be using a CSS in JS library from later lessons and for the final project
    - At Sofascore we currently use *styled-components*, but that might change in the near future
- Each element can receive style object with its styles
    - Inline styling - like style attribute on a HTML element
- Elements can also be styled via classes

Sofascore

React example from scratch - demoProject.md

03

Hooks and useState

Hooks and useState
Class vs functional components

- In initial React releases, class components were much more powerful
    - Functional components could not access React lifecycle, just return data based on props
    - Entered: **Hooks**
    - Now they are equally powerful but with less code -> class components become an afterthought
- We will be learning only functional components
- Hooks are always prefixed with *use* keyword

Sofascore

Hooks and useState

Class vs functional components

Mark Dalgleish
@markdalgleish

Following

"With hooks, beginners no longer need to learn about 'this' to avoid shooting themselves in the foot."

Closures:

HELLO THERE

1:23 PM - 11 Feb 2019

93 Retweets **578** Likes

sofascore

useState

- **Internal** component state used inside it
    - Belongs to an instance, not a class or function itself, so every instance will have its own encapsulated state
- <u>When state changes, component updates</u>
- <u>Should not be modified directly</u> (it's a const after all) - use setter method
    - const [counter, setCounter] = useState(0)
- Used to store dynamic data
- Parent component can't access child components state
    - A component can pass its state (or part of it) as a prop to the children

Sofascore

04

useEffect

Rules of hooks and useEffect

React lifecycle

- React lifecycle phases:

    - Mounting -> Rendering element in the DOM for the first time

    - Updating -> Re-rendering component with fresh props & state

    - Unmounting -> Removing element from the DOM

    - Error Handling

Sofascore

Hooking into func. lifecycle with useEffect hook

```
React.useEffect(() => {
  // body of a function which is executed in hook, after first render

  return () => {
    // cleanup method which is executed when the component is unmounted from the DOM
  }
}, [/* array of dependencies */])
```

- Executed when <u>component renders/updates</u> and a <u>value in array of dependencies changes</u>

- Array of dependencies

  - OPTIONAL - no array of dependencies: executed on every re-render

  - Empty - on first render

- The method in effect OPTIONALLY returns a <u>cleanup function</u> - free all taken resources and remove listeners

Sofascore

useEffect example - clock.md

Rules of hooks and useEffect

React lifecycle

- Function inside useEffect can't be async because async method returns a promise, not void or cleanup method
    - Define an async function inside your useEffect function and then run it
- The place to make *effects* happen
    - Adding and cleaning up a listener, for example keypress listeners
- Not the place to do:
    - Transforming and filtering data for rendering - we'll talk about useMemo later
    - Handling user input -> that can be done just via click and state

Sofascore

Rules of hooks (more later...)

- **Only call hooks at the top level**

    - i.e. never call them after any return method

- **Don't call them conditionally**

    - but they can encapsulate conditional logic

- By abiding these two rules, you will hopefully abide a third one

    - **At every re-render, exactly the same amount of hooks should be called**

    - **In the exactly the same order**

- **Only Call Hooks from React Functions**
    - i.e. call them from React function components or custom hooks

Sofascore

05

Lists

Lists
React Lists

- Array is a valid JSX element

    - Can contain any valid JSX element

- key attribute is a must

    - SHOULD be unique for each element in an array

    - Describe an element as closely as possible (ideally unique Id for each element)

    - Do not use the index as key!

- key is important for React to be performant when rendering lists

    - Allows React to reuse old DOM structure

Sofascore

List example - list.md

Thank you for your attention!

Sofascore