## Introduction to FE in JS (React)

Miroslav Holeček, Martin Ledvinka

KBSS

Winter Term 2022



Contents









Miroslav Holeček, Martin Ledvinka (KBSS)

### Motivation

- Increased interactivity
- Seamless (and asynchronous!) re-rendering
- Fetching data on-the-fly
- Fast (client-side) validation
- $\bullet \ \rightarrow \text{Improved UX}$



# History



### HTML + JS

- Normal "static" HTML
- JS to add interactivity, etc.
- Using DOM APIs
- Historically: many implementation inconsistencies (not just IE)



## jQuery

- Fast, small, and feature-rich JavaScript library
- Makes HTML document traversal and manipulation, event handling, animation, Ajax, ... much simpler
- Easy-to-use API that works across a multitude of browsers
- Nowadays: mostly obsolete the DOM has evolved



# Nowadays



### Promising: Web Components

- Suite of different technologies for creating reusable custom elements
- HTML + DOM standardized
- https://github.com/aubi/sample-js-webelements



### Node & ecosystem

- Open-source, cross-platform JavaScript runtime environment
- Not just for server-side JS tooling for FE development (and more)
- NPM<sup>1</sup>  $\sim$  Maven (including a repository)
- Over 1 million packages (including feats like is-odd)
- package.json  $\sim$  pom.xml



<sup>1</sup>https://www.npmjs.com/

Miroslav Holeček, Martin Ledvinka (KBSS)

### Building and bundling

- Non-standard syntax often used (ESNext, TypeScript, JSX,  $\dots^2$ )  $\rightarrow$  *transpilation* step required
- Minification to reduce file size: e.g. renaming local variables
- Bundling to include libraries, reduce the amount of (HTTP) requests
- May seem like magic, sometimes breaks and requires a magic solution

```
const AwesomeButton:
                                                                  .push=t.bind(null,r.push.bind(r))}(),function(){var e=n(791),t=n
 (props: { text: string }) ⇒ ReactElement
                                                                  (164).r=function(e){e&e instanceof Function&n.e(787).then(n.bind
 = (\{text : string \}) \Rightarrow \{
                                                                  (n,787)).then((function(t){var n=t.getCLS,r=t.getFID,l=t.getFCP,a=t
                                                                  .getLCP.o=t.getTTFB:n(e).r(e).l(e).a(e).o(e)}))}.l=n(184).
 return <button style={{color: 'red'}}>{text}</button>
                                                                   a=function(e){var t=e.text;return(0,l.jsx)("button".
                                                                  {style:{color:"red"}.children:t})};
ReactDOM.render(
  <React.StrictMode>
                                                                  -t.render((0,l.jsx)(e.StrictMode.{children:(0,l.jsx)(a.{text:"hi!"})
                                                                  }).document.getElementBvId("root"))
    <AwesomeButton text="hi!" />
 </React.StrictMode>.
 document.getElementById( elementid: 'root')
                                                                           Figure: main.cf6a94b4.js
):
```



<sup>2</sup>Not just JS: templates, styles (Less, Sass), even images.

Miroslav Holeček, Martin Ledvinka (KBSS)

Introduction to FE in JS (React)

### TypeScript

- Strongly typed programming language that builds on JavaScript
- Compile-time only, compiles (through tsc) to normal JS
- Type inference, duck typing
- Supports nullable types
- Type-related syntax similar to e.g. Kotlin
- Not always 100% correct, even introduces some issues
- Current verdict: usually worth it

### React basics

- Declarative using JSX
- Component-based
- Composition, composition, composition
- $\bullet\,$  Very easy to make a massive mess of  $\to$  we will use TypeScript

```
30 function Container(props) {
31 ● return <div style={{ background: "red" }}>
32 - {props.children}
33 - </div;
34 }
35 const Greeting = (props) ⇒
36 - <marquee>{props.name}</marquee>;
```





### Creating and running a project

### Install Node, NPM.<sup>3</sup>

Inpx create-react-app my-example-app --template typescript

- 3 cd my-example-app
- npm start
- Your browser will open and display your app.
- Try making a change (e.g. in text) and see hot-reloading in action.

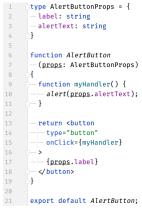
<sup>&</sup>lt;sup>3</sup>Using a version manager, such as **nvm** or **n**, is a good idea. On Linux, preferably don't rely on default repositories, they are often out-of-date.



### Custom components

Let's create a new component called AlertButton: a button with a label that creates an alert when clicked.

- Create a new file called AlertButton.tsx.
- Define the *props* (component API) using 2 TypeScript.
- Implement the function component itself using JSX.
- Export the new component from the file and import it in App.tsx.
- Use the component (e.g. after the link in header) and see the results!





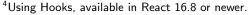
## Local state management<sup>4</sup>

React includes the usestate hook, which returns a 2-tuple (array) of state and its setter: const [myState, setMyState] = useState("default");

### Simple example

```
const Greeter: FC = () => {
 const [name, setName] = useState("");
 return <>
   <input
    tvpe="text"
    value={name}
    onChange={ (ev) => setName (ev.target.value) }
   />
   {name && Hello, {name}!}
 </>;
};
```

For global (app-level) state management, you need a different approach (Context/Redux/...). This data is also transient, i.e. lost on **remount**.



## AJAX using fetch API<sup>5</sup>

```
import { useEffect, useState } from "react";
export default function IpDisplay() {
 const [ip, setIp] = useState<string>("loading");
 useEffect(() => \{
   fetch("https://icanhazip.com")
    .then((resp) => \{
      return resp.ok ? resp.text() : "unknown";
    })
    .then((text) => setIp(text.trim()))
    .catch((e) => {
     console.error(e);
     setIp("unknown");
    });
 }, []);
 return Your IP is {ip}.;
```



<sup>5</sup>You will probably almost always use a library instead of this "raw" API.

# Using a (component) library

For rapid development, you might want to use a component library. These often include a plethora of controls, from styled text to interactive tables.

- Pick a library. There's lots of them, let's try Evergreen here.
- Examine the documentation (including possible extra install steps).
- Install the package with NPM: npm install everyreen-ui 3
- In your code, import the component from the package. For example: import { Button } from "evergreen-ui";
- Use the component in your code, as if you were importing locally.

For other libraries, including more technical ones like react-router, the process is the same (but the specifics may of course differ).



# Consultations



### Resources

- https://jquery.com/
- https://developer.mozilla.org/en-US/docs/Web/ Web\_Components
- https://www.typescriptlang.org/
- https://reactjs.org/

