Introduction to FE in JS (React)

Miroslav Holeček, Martin Ledvinka

KBSS

Winter Term 2023

Contents

Motivation

- Increased interactivity
- Seamless (and asynchronous!) re-rendering
- Fetching data on-the-fly
- Fast (client-side) validation
- $\rightarrow$ 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[1] $\sim$ Maven (including a repository)
- Over 1 million packages (including feats like `is-odd`)
- `package.json` $\sim$ `pom.xml`

[1] `https://www.npmjs.com/`

Building and bundling

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

```
6   const AwesomeButton:
7     (props: { text: string }) => ReactElement
8     = ({text : string }) => {
9     return <button style={{color: 'red'}}>{text}</button>
10  }
11
12  ReactDOM.render(
13    <React.StrictMode>
14      <AwesomeButton text="hi!" />
15    </React.StrictMode>,
16    document.getElementById( elementId: 'root')
17  );
```

```
.push=t.bind(null,r.push.bind(r))}(),function(){var e=n(791),t=n
(164),r=function(e){e&&e instanceof Function&&n.e(787).then(n.bind
(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),

a=function(e){var t=e.text;return(0,l.jsx)("button",
{style:{color:"red"},children:t})};

t.render((0,l.jsx)(e.StrictMode,{children:(0,l.jsx)(a,{text:"hi!"})
}),document.getElementById("root"))
```

Figure: `main.cf6a94b4.js`

[2] Not just JS: templates, styles (Less, Sass), even images.

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

```
7   type Dog = {
8     name: string
9     age: number
10    city?: string
11  }
```

```
13  const cricket: Dog = {
14    name: 'Cricket',
15    age: 7
16  }
```

```
18  function greetDog(dog: Dog): string {
19    return `Hi ${dog.name}`
20  }
```

React basics

- Declarative using JSX
- Component-based
- Composition, composition, composition

```
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>;
```

```
22  function GreetingPage() {
23      return (
24          <Container>
25              <Greeting name="EAR" />
26          </Container>
27      );
28  }
```

Creating and running a project

1. Install Node, NPM.[3]
2. `npx create-react-app my-example-app --template typescript`
3. `cd my-example-app`
4. `npm start`
5. Your browser will open and display your app.
6. Try making a change (e.g. in text) and see hot-reloading in action.

[3] Using a version manager, such as **nvm** or **n**, is a good idea. Do not rely on default repositories on Linux, 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.

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

```tsx
type AlertButtonProps = {
  label: string
  alertText: string
}

function AlertButton
  (props: AlertButtonProps)
{
  function myHandler() {
    alert(props.alertText);
  }

  return <button
    type="button"
    onClick={myHandler}
  >
    {props.label}
  </button>
}

export default AlertButton;
```

Local state management[4]

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
      type="text"
      value={name}
      onChange={(ev) => setName(ev.target.value)}
    />
    {name && <p>Hello, {name}!</p>}
  </>;
};
```

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

[4] Using Hooks, available in React 16.8 or newer.

AJAX using fetch API[5]

```
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 <p>Your IP is {ip}.</p>;
}
```

[5] 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.

1. Pick a library. There's lots of them, let's try Evergreen here.
2. Examine the documentation (including possible extra install steps).
3. Install the package with NPM: `npm install evergreen-ui`
4. In your code, import the component from the package.
   For example: `import { Button } from "evergreen-ui";`
5. 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 (specifics may differ).

Consultations

Resources

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