Week 9: Introduction to React

Agenda

- Setting up a React app
- Components
- Event Callbacks
- Hooks
- CSS in React
- Storybook
- Demo

Setting up a React app

Vite

- `npm create vite@latest my-app -- --template react-ts` (Preferred way)
- Fast, flexible and very responsive hot module reloading

CRA (Create React app)

- `npx create-react-app my-app --template typescript` (Not really used anymore)

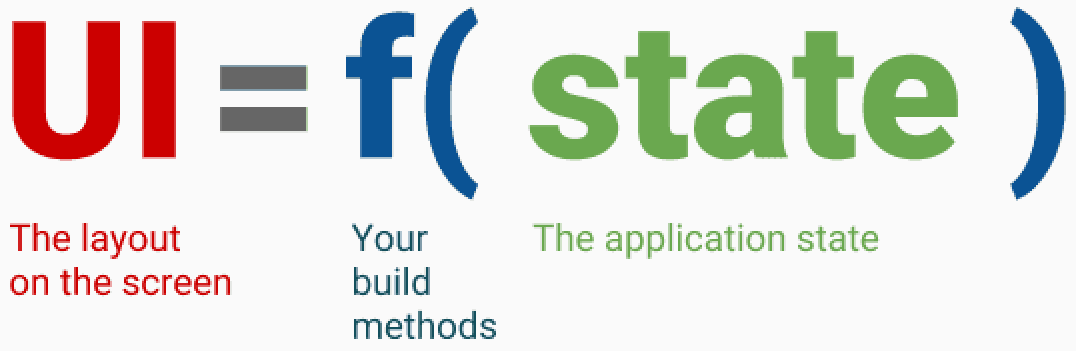
Bundling

- Vite or CRA takes care of compiling and bundling your application

Project build includes:

- Minified and optimized JavaScript code (This is typically done using a tool like Webpack)
- HTML file is generated that includes a reference to the minified JavaScript code
- CSS files (CSS files are also bundled and included in the production build)
- Statis assets (images, fonts...)
- More...

UI



- UI is pretty much only a function of the state of the application.
- React automatically updates the UI to reflect application changes
- This is why React is often called a "declarative" framework, because you declare what your UI should look like based on the current state of your application

Components

- A way of splitting the UI into independent, reusable pieces
- Use functional way to create components and define an interface for its props

```
export const Component = (props: ComponentProps) => {
    <p> {"Hello there"} </p>
}
```

- First letter of the component name must always be capitalized

Components

- The React component lifecycle is a series of phases that a component goes through, from creation to destruction.
- During the lifecycle, a component can be mounted, updated, and unmounted.
- The lifecycle methods can be used to perform actions at specific points in the component's lifecycle.

Properties

- `props` (short for properties) are a way to pass data from a parent component to a child component

```
function Greeting({ name }) {
  return <h1>Hello, {name}!</h1>;
}

function App() {
  return <Greeting name="John" />;
}
```

Rendering lists

- Rendering a list of elements in React typically involves mapping over an array of data and generating a new element for each item in the array
- To ensure that each element in the list has a unique key, React requires that you provide a key prop for each element.
- This key prop ensures that should the data for one element change, we do not have to re-render the entire list.

Rendering lists

```
function MyList({ items }) {
  return (
    <ul>
      {items.map((item) => (
        <li key={item.id}>{item.text}</li>
      ))}
    </ul>
  );
}

const myListItems = [
  { id: 1, text: "Item 1" },
  { id: 2, text: "Item 2" },
  { id: 3, text: "Item 3" },
];
```

Event callbacks

- Functions that are executed when a specific event occurs in a React component, such as a button click or a form submission
- Typically defined as methods within the component class and are passed down to child components as props
- Can be used to update component state, trigger side effects, or interact with APIs and other external services

Examples

- `onClick`, `onMouseEnter`, `onMouseLeave`, `onChange`, `onSubmit`

Event callbacks (OnClick)

```
function MyButton({ onClick }) {
  return <button onClick={onClick}>Click me!</button>;
}

function MyComponent() {
  function handleClick() {
    console.log("Button clicked!");
  }

  return <MyButton onClick={handleClick} />;
}
```

Event callbacks (OnChange)

```
function MyInput({ value, onChange }) {
  return <input type="text" value={value} onChange={onChange} />;
}

function MyComponent() {
  const [inputValue, setInputValue] = useState("");

  function handleInputChange(event) {
    setInputValue(event.target.value);
  }

  return <MyInput value={inputValue} onChange={handleInputChange} />;
}
```

Hooks

- Hooks are special functions that are aware of a component's life cycle
- React provides built-in hooks, but can create our own more complex hooks

When using hooks there are a few rules that you need to follow:

- Only call hooks at the top level of the component
- Only call hooks from React functions

Example hooks:

- `useState` , `useEffect` , `useRef` , `useMemo` , `useCallback`

State hook

- Essentially a variable which needs to be dynamically rendered

```
const [ state, setState ] = useState<type>(initialValue)
```

- Change the value only by using setState, as it also tells each component using the state to rerender

```
const [ state, setState ] = useState<number>(5)
setState(6);
setState(previousValue => previousValue + 1);
```

- We can access previous value or set the new value directly

Effect hook

- Lets you perform side effect when rendering
- `useEffect(func)` is called at every render
- `useEffect(func, [])` is called on mount
- `useEffect(func, [state])` is called whenever one of the listed states changes
- It's a good place to make API calls, and you will need it when fetching data from REST API

Effect hook

```
useEffect(() => {
  // Effect callback
  // Only called when 'foo' or 'goo' changes
  return () => {
    // Cleanup callback
    // Called on unmount or before the effect callback is called because dependencies changed
  };
}, [foo, goo]);
```

- Effect hook can also specify what happens when component is going to be unmounted

Ref hook

- Built-in hook in React that allows you to create a mutable reference to a DOM element or to a value
- The main use case for useRef is to access a DOM element without triggering a re-render when its value changes

```
function MyComponent() {
  const inputRef = useRef(null);

  function handleClick() {
    inputRef.current.focus();
  }

  return (
    <div>
      <input type="text" ref={inputRef} />
      <button onClick={handleClick}>Focus input</button>
    </div>
  );
}
```

Other hooks

UseMemo

- It's quite similar to `useEffect`, but returns a value
- Used to speed up and optimize your app by storing the results of expensive function calls by returning the cached result when its dependencies don't change between renders

```
const winner = useMemo(() => getWinner(board), [board]);
```

Use Callback

- Specialized version of the `useMemo` hook, used for memoizing functions

```
const onBoardRestart = useCallback(() => {setBoard({});}, []);
```

CSS in React

- Inline as in HTML
- Inline with a style object
- Import stylesheet as in HTML
- CSS modules, similar to stylesheets, but styles are local
- Libraries: MUI, Bootstrap, Tailwind

CSS in React (MUI)

```
import { Button, TextField } from "@material-ui/core";

function MyForm() {
  return (
    <form onSubmit={handleSubmit}>
      <TextField label="Username" />
      <TextField label="Password" type="password" />
      <Button variant="contained" color="primary" type="submit">
        Submit
      </Button>
    </form>
  );
}
```

CSS in React (Bootstrap)

```
import 'bootstrap/dist/css/bootstrap.min.css';
import { Button, Form, FormGroup, Label, Input } from 'reactstrap';

function MyForm() {
  return (
    <Form onSubmit={handleSubmit}>
      <FormGroup>
        <Label for="username">Username</Label>
        <Input type="text" name="username" id="username" placeholder="Enter your username" />
      </FormGroup>
      <FormGroup>
        <Label for="password">Password</Label>
        <Input type="password" name="password" id="password" placeholder="Enter your password"
      </FormGroup>
      <Button color="primary" type="submit">
        Submit
      </Button>
    </Form>
  );
}
```

CSS in React (Tailwind)

```
import 'tailwindcss/tailwind.css';

function App() {
  return (
    <nav className="bg-white shadow">
        <div className="max-w-7xl mx-auto px-2 sm:px-6 lg:px-8">
          <div className="flex items-center justify-between h-16">
            <div className="flex-shrink-0">
              <img className="h-8 w-8" src="/logo.svg" alt="Logo" />
            </div>
          </div>
        </div>
    </nav>);
```

Storybook

- In some seminars we will use a tool called storybook
- It is used for building isolated components, and it is perfect for learning to build components
- You can install it by running `npx sb init` inside an existing project, but both demo and iteration will have it installed beforehand
- Use `npm run storybook` to start the storybook

Hands on: Demo

- Download demo from Interactive syllabus
- Read `readme.md`