Front-end <Web /> Development

Lesson 11: React (Part 2)

University of Tehran

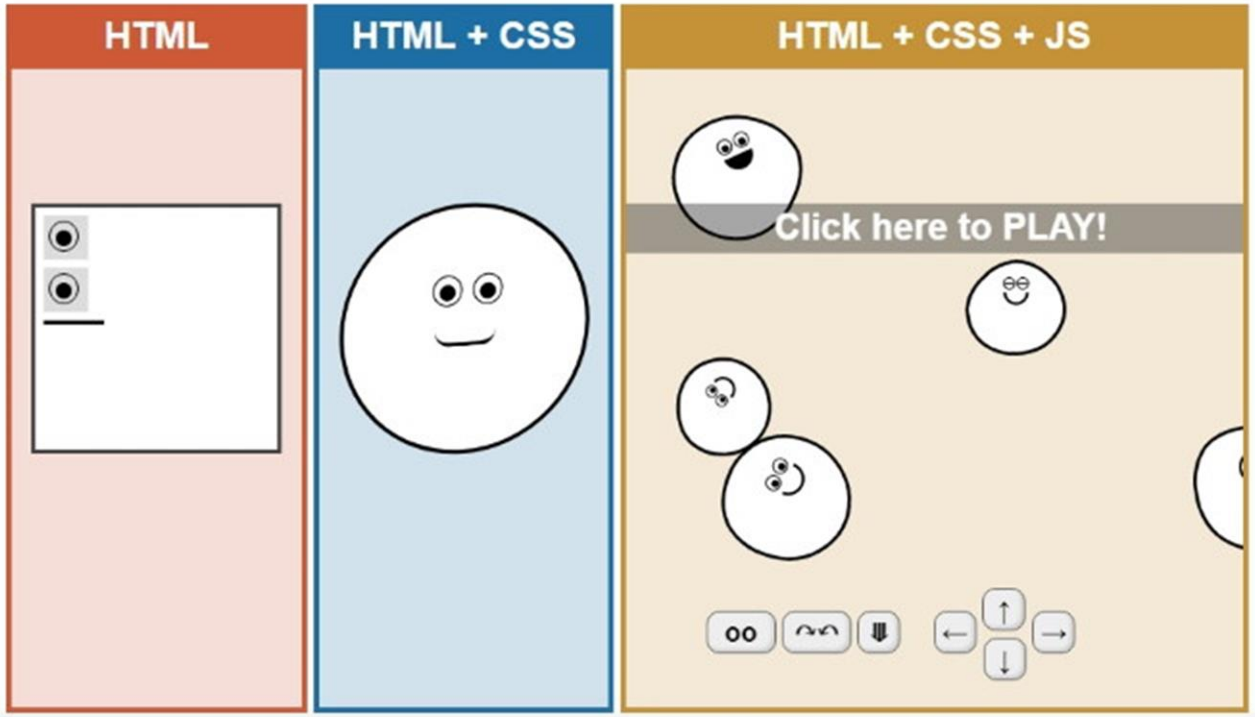
ACM
Summer of Code 2024

Lecturer:
Misagh Mohaghegh

A Short Review…

- Modules (export, import, default)
- Bundling (and minification, uglification, transpilation)
- Imperative vs Declarative
- Virtual DOM
- JSX
- Function and class components
- Props
- Rendering lists and repeats
- Conditionals
- Different HTML attributes
- Installation (CRA) and project structure

Continue…



+

React

State

State

State is used for a component to remember some information.

In React, we don't write commands such as "disable the button" or "show the message".
Instead, we describe our UI for different states such as "initial state" and "success state".

Hooks

React functions starting with "use" are called **Hooks**.
For example, **useState** is a Hook.

Hooks should only appear on the top-level of a component.

useState

The syntax for useState is as follows:

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

function MyComponent() {
    const [myVar, setMyVar] = useState(0);
    return <p>{myVar}</p>;
}
```

useState takes the initial value (0 here, but it could be anything, even an object) and returns 2 values. We should never directly set myVar and should instead use the setMyVar function.

Example

Lets say we want to have a button that counts how many times it has been clicked:

```
function MyButton() {
    const [count, setCount] = useState(0);

    function handleClick() {
        setCount(count + 1);
    }

    return <button onClick={handleClick}>{count}</button>;
}
```

Lifting State Up

What if we want 2 buttons that share the same counter?

We should move the state to their parent and pass the values as props.

```
function MyButton({ count, onClick }) {
    return <button onClick={onClick}>{count}</button>;
}
function ParentComponent() {
    const [count, setCount] = useState(0);
    function handleClick() {
        setCount(count + 1);
    }

    return (<>
        <MyButton count={count} onClick={setCount} />;
        <MyButton count={count} onClick={setCount} />;
    </>)
}
```

Updating Objects

What if our state is an object and we want to update only one field?

```
function Component() {
    const [test, setTest] = useState({
        first: '',
        second: 0,
        third: false
    });

    function handleChange(newFirst) {
        setTest({ ...test, first: newFirst });
    }

    return ...
}
```

Updating Arrays

What if our state is an array and we want to update only one element?

```
function Component() {
    const [test, setTest] = useState([
        { id: 0, name: 'first' },
        { id: 1, name: 'second' }
    ]);

    function handleChange(id, newName) {
        setTest(test.map(t => {
            if (t.id === id) return { ...t, name: newName };
            else return t;
        });
    }
}
```

Renders

The useState's set function does not change the value until the next re-render is triggered:

```
function MyButton() {
    const [count, setCount] = useState(0);

    function handleClick() {
        setCount(count + 1);
        console.log(count); // still the previous value
    }

    return <button onClick={handleClick}>{count}</button>;
}
```

Context

Prop Drilling

Prop drilling is the process of passing props down to children to reach a certain child on the bottom.

For example, we want to pass the current user's name to a form element that is 6 components deep from where the username is stored.
We have to pass the username as props to all components in their way so they can pass it downwards in the component tree.

Sometimes the components in the middle of the way do not even need the prop and just directly pass it to the children.
This can make our code complex.

Context

useContext is another Hook which allows us to pass implicit props to all children of an element in any depth without actually passing the prop.
This means that all child components and their children (the whole component subtree) can access a value without actually having it passed as a prop argument.

To access a value in such way, a context should be created first:

```
// ThemeContext.js

import { createContext } from 'react';

export const ThemeContext = createContext('light');
```

Provider

Now whichever part of the code that all children of it should access a value will be wrapped in the provider:

```
import { ThemeContext } from './ThemeContext';

function App() {
    const [theme, setTheme] = useState('light');
    return (
        <ThemeContext.Provider value={theme}>
            <SomeComponent />
        </ThemeContext.Provider>
    );
}
```

useContext

Now all children of <App /> can access the value of the nearest provider:

```
import { useContext } from 'react';
import { ThemeContext } from './ThemeContext';

function SomeComponent() {
    const theme = useContext(ThemeContext);
    ...
}
```

Default Value

The value passed to the provider is accessed by useContext.
If no provider is found in the upper tree of where useContext was used, the default context value (which was passed to createContext) will be returned.

The context provider value will usually be a state variable which can trigger a re-render on the children that access it after a setState.

Effect

useEffect

The useEffect Hook is used mostly for fetching data from external systems.

We can also use it to run a code the first time a component is mounted, or on every update.

```
import { useEffect } from 'react';

function MyComponent() {
    useEffect(() => {
        ...
    }), []);

    return ...;
}
```

useEffect

useEffect takes two parameters: a function and a dependency list.
Whenever a variable in the dependency list changes, the function is run.

If the list is empty, the function is only run once on component mount.
If we omit the list, the function will be run on every component update.

The function may return another function which will be run on component unmount.

React Router

React Router

React does not have any routing built-in.

React Router is a library that provides routing for our pages.

```
npm install react-router-dom
```

- RouterProvider
- createBrowserRouter
- errorElement
- useParams
- useSearchParams
- useNavigate
- <Link />

Thank you for your attention.