React Hooks, Style, Routing, and Libraries

CIS 1962 (Winter 2026)
February 26th, 2026

Penn Engineering

Lesson Plan

5	React Hooks & States
33	Styling in React
44	React Routing
53	React Libraries

Logistics

- Homework 4 Due Today!
- Homework 5 Released!
  - Mainly a front-end homework (like HW3) using React
  - Build a Pokedex with a provided API to fetch Pokemon!
  - Learn about pagination, modals, and implement a search feature!

Review Activity

https://edstem.org/us/courses/91614/lessons/160635/slides/943661

Let's review some content from the previous lecture before we start!

Hooks & States

What are React Hooks and what powerful features do they provide?

Early React: Class Components

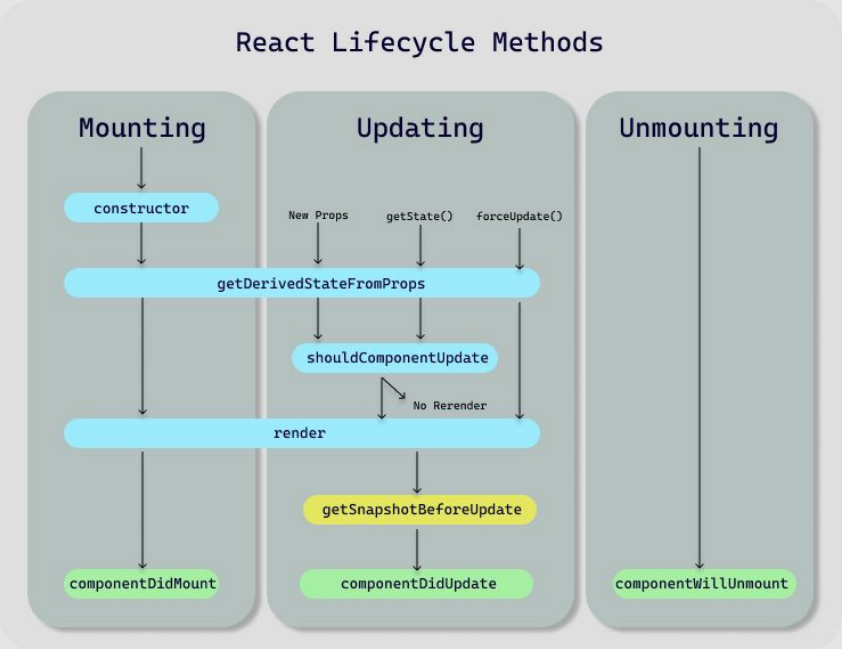
Before Hooks, **class components** were used over functional components due to them allowing state management.

However, it was quite complex, required specific syntax (`this` binding), and some component lifecycle management.

```
import React from "react";

class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  handleIncrement = () => {
    this.setState((prevState) => ({
      count: prevState.count + 1
    }));
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.handleIncrement}>Increment</button>
      </div>
    );
  }
}

export default Counter;
```

Component Lifecycle

There used to have some annoying lifecycle methods to tell React **when** to do things, like `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.



React Hooks

In 2019, **Hooks** were introduced to React, allowing functional components to have states.

This made managing states simpler with functional component syntax without needing to manage lifecycle methods!

After Hooks, functional components became standard while class components were phased out.

Component Lifecycle, Simplified

While the component lifecycle still exists, Hooks make it easier to interface with it.

- The body of functional components runs every re-render, but any stateful values are kept between renders
- `useEffect` covers most of the lifecycle methods in terms of mounting, updating, and cleanup with the use of a dependency array.

Using Hooks

Hooks need to be imported from the React library to be used.

They are named exports within the library.

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

const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

Rules of Hooks

Hooks must be:

- Called from top level, **NOT** inside conditionals, event handlers, or callback functions
- Called from a React functional component, **NOT** a regular JS function

useState: Initialize

`useState` accepts an initial state and returns 2 values:

- The current state
- An update function

You'll often use destructuring to get these 2 values.

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

const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

useState: Read and Update

You can use the state variable anywhere in the scope of the component.

To update, you can use the state updater function only, you cannot **directly** update the state variable value.

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

const Counter = () => {

  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

useState: Updating Objects

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

const UserProfileForm = () => {
  const [user, setUser] = useState({
    name: "Jane Doe",
    preferences: {
      newsletter: true
    }
  });

  const handleNewsletter = () => {
    setUser((prevUser) => ({
      ...prevUser,
      preferences: {
        ...prevUser.preferences,
        newsletter:
!prevUser.preferences.newsletter
      }
    }));
  };

...
```

```
 ...

  return (
    <div>
      <h2>User Profile</h2>
      <p>Name: {user.name}</p>
      <label>
        Subscribe to newsletter:
        <input
          type="checkbox"
          checked={user.preferences.newsletter}
          onChange={handleNewsletter}
        />
      </label>
      <pre>{JSON.stringify(user, null, 2)}</pre>
    </div>
  );
}

export default UserProfileForm;
```

Updates are Batched and Async

For performance sake, multiple updates to states in a short period are **batched** together to re-render at the same time.

State updates are also **asynchronous**, being scheduled for the most optimization rather than occurring synchronously the moment their update functions are called.

Class Activity

https://edstem.org/us/courses/91614/lessons/160635/slides/943660

React "escape hatches"

React is declarative with states, props, and rendering.

However, sometimes you need a way to step outside of React, with an **"escape hatch"** to do things like:

- Interact with Browser APIs
- Timers/ subscriptions
- Sockets/Servers
- Use non-React libraries

React achieves this through `useEffect` and `useRef`.

useEffect: Side Effects

Side effects are tasks unrelated to rendering UI, such as data fetching, timers, and logging.

`useEffect` is often used to unify lifecycle methods to run tasks on component mount, update, and unmount.

This allows you to control **when** side effects run.

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

useEffect(() => {
    ...
});
```

useEffect: Dependency Array

The timing of when side effects run is controlled by the 2nd argument to `useEffect`: the dependency array.

Side effects can:

```
useEffect(() => {
    ...
});
```

Run every render

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

Run on first render

```
useEffect(() => {
    ...
}, [x, y]);
```

Run on first render AND When dependency (x or y) changes

useEffect: Timer

This component counts up continuously, since count is always changing and re-rendering.

```
const Timer = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  });

  return <h1> {count} </h1>;
}
```

useEffect: CountOnce

This component counts up only once, on the first render.

```
const CountOnce = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
  }, [])

  return <h1> {count} </h1>;
}
```

Sidenote: Strict Mode

`<StrictMode>` helps detect potential issues in a React app, by detecting unsafe lifecycle methods, unexpected side effects, and more. This happens **during development only.**

It also causes certain functions to run twice, (Like effects) so that the rendering result is idempotent (same result each time)

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.tsx'

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

Strict Mode: Twice is Nice

The code below shows 'Effect is running' twice! What's happening?

1. Component is mounted
2. Effect runs
3. Unmounts the component (and runs cleanup)
4. StrictMode mounts the component again
5. Effect runs

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

export default function App() {
  useEffect(() => {
    console.log("Effect is running");

    return () => {
      console.log("Cleanup is running");
    };
  }, []);

  return <h1>Check the console</h1>;
}
```

useEffect: Counter

This component initially sets the calculation variable, then automatically runs again when count is updated.

```
const Counter = () => {
  const [count, setCount] = useState(0);
  const [calculation, setCalculation] = useState(0);

  useEffect(() => {
    setCalculation(() => count * 2);
  }, [count]);

  return (
    <>
      <p>Count: {count}</p>
      <button onClick={() => setCount((c) => c + 1)}>+</button>
      <p>Calculation: {calculation}</p>
    </>
  );
}
```

useEffect: Cleanup

If you want to perform actions when a component is unmounted or if the effect is rerun, you can return a function to perform the task.

```
const Timer = () => {
  const [count, setCount] = useState(0);

  useEffect(() => {
    let timer = setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);

    return () => clearTimeout(timer);
  });

  return <h1> {count} </h1>;
}
```

Class Activity

https://edstem.org/us/courses/91614/lessons/160635/slides/943663

useRef

`useRef` allows values to persist between renders.

Any change to this value will not trigger a re-render.

```
const Toggle = () => {
  const [toggle, setToggle] = useState(true);
  const count = useRef(0);

  useEffect(() => {
    count.current = count.current + 1;
  });

  return (
    <>
      <button onClick={() =>
setToggle(!toggle)}>Press Me!</button>
      <p>Renders: {count.current}</p>
    </>
  );
}
```

useContext

`useContext` allows a way to manage state globally.

This solves the problem of prop drilling by managing a Context instead.

```
const Grandparent = () => {
  return <Parent user="voravich" />;
}

const Parent = (props) => {
  return <Child user={props.user} />;
}

const Child = (props) => {
  return <div>User: {props.user}</div>;
}
```

Penn Engineering

```
const UserContext = createContext();

const Grandparent = () => {
  return (
     <>
          <UserContext.Provider
     value="voravich">
               <Parent />
          </UserContext.Provider>
          <Brother/>
     </>
  );
};

const Parent = () => {
  return <Child />;
};

const Child = () => {
  const user = useContext(UserContext);
  return <div>User: {user}</div>;
};
```

useMemo

useMemo can be used to memorize values, essentially caching them.

This can be used to keep expensive operations from running every render, and only updating when the dependency updates.

```
function slowFunction(num) {
  console.log("Calculating...");
  let result = 0;
  for (let i = 0; i < 1e8; i++) {
    result += num;
  }
  return result;
}

const ExpensiveCalculator = ({ number }) => {
  const result = useMemo(() =>
slowFunction(number), [number]);

  return <div>Result: {result}</div>;
}
```

Extra Activity

https://edstem.org/us/courses/91614/lessons/160635/slides/943664

Custom Hooks

Sometimes you wish you had a hook in React for a specific purpose for reuse. You can write a **Custom Hook** for it!

```jsx
import { useState, useEffect } from "react";

function Users() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")
      .then(res => res.json())
      .then(setData);
  }, []);

  return (
    <ul>
      {data.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

w/o custom hook

```jsx
import { useState, useEffect } from "react";

export function useFetch(url) {
   const [data, setData] = useState([]);

   useEffect(() => {
      fetch(url)
         .then(res => res.json())
         .then(setData);
   }, [url]);

   return data;
}

function Users() {
   const users =
useFetch("https://jsonplaceholder.typicode.com/users");
   ...
}
```

Custom Hook

Class Activity

https://edstem.org/us/courses/91614/lessons/160635/slides/943865

Styling in React

How do we write style code in React?

React Styles

We can write CSS the classic way with:

- **CSS Modules**
- **CSS-in-JS** with styled-components
- **Inline Styles and Utility Classes**

We can also make use of popular CSS and React UI Libraries:

- **CSS Libraries** like Bootstrap & Bulma
- **React UI** like MUI, React Bootstrap, & Chakra

CSS Modules

If you still want to write plain CSS, CSS Modules will allow you to classes scoped to components. This reduces collisions, and allows you to manage one CSS file per component easily.

```
import styles from './style.module.css';
// or import { myClass } from
"./style.module.css";

// function MyComponent(...) {
return <div
className={styles.myClass}>Hi!</div>;
```

```
/* style.css */

.myClass {
  font-size: 20rem;
}
```

CSS-In-JS

Alternatively, you can use a technique called CSS-In-JS where you write CSS directly in your JS code.

This keeps all your code in one place rather than in multiple files.

Additionally, since we're working with components, we extract repeated items into components for better readability!

CSS-In-JS: styled-components

`styled-components` is a popular library that allows you to insert styles right into JS.

We can define these style components with specific CSS styles and can insert them into the file immediately.

```
const Wrapper = styled.section`
  padding: 4em;
  background: papayawhip;
`;

const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: ${(props) => props.color ??
'#bf4f74'};
`;

// function MyComponent(...) {
return (
  <Wrapper>
    <Title color="#ffccff">How Stylish</Title>
  </Wrapper>
);
```

Inline styles with Utility Classes

Writing styles inline is also a popular method, through libraries like Tailwind.

It's faster than writing full CSS properties, and many libraries will include utility classes to easily build UIs, including pseudo-classes and mobile-responsive support.

```
<div
     className={`h-full flex ${active ?"justify-center items-end lg:items-center
      lg:justify-end" : ""}`}
>
     ...
</div>
```

Tailwind

Our class website uses Tailwind! Take a look at how Tailwind's pre-written classes get applied to a section of the web page:

```
<nav
    className="fixed top-0 left-0 w-full bg-white dark:bg-black shadow z-50 border-b border-slate-200"
    style={{ fontFamily: "var(--font-source-sans)" }}
>
    <div className="mx-auto flex items-center gap-6 px-4 sm:px-8 md:px-12 lg:px-24 xl:px-32 py-4 max-w-7xl">
        <Link href="/#" aria-label="Home">
            <Image
                src={upenn}
                alt="University of Pennsylvania Shield"
                width={1760}
                height={2000}
                className="w-11 md:w-20 h-12.5 md:h-20 min-w-20 min-h-20"
            />
        </Link>
        <div className="flex flex-col flex-1 text-black dark:text-white">
            <h1 className="text-center md:text-left font-extrabold text-5xl leading-tight overflow">
                CIS 1962<span className="hidden md:inline">: JavaScript</span>
            </h1>
```

CSS Libraries

In contrast to writing traditional CSS or Tailwind utility classes, CSS libraries will include a lot more opinionated styles and components for different themes with little customization.

```
// Bulma
const BulmaNotification = ({ message,
type = "is-primary" }) => (
  <div className={`notification
${type}`}>
    {message}
  </div>
);
```

```
<BulmaNotification type="is-danger"
message="Something went wrong!" />
```

React UI Libraries

You can go further by installing a React UI library that provides ready-made components.

You provide props to these components that under the hood that turned into style classes.

```
// MUI
<Grid xs={12} sm={12} md={6} lg={3} xl={3}>
  <Box
    width="100%"
    backgroundColor={colors.primary[400]}
    display="flex"
    alignItems="center"
    justifyContent="center"
  >
    <StatBox
      title="32,441"
      subtitle="New Clients"
      progress="0.30"
      increase="+5%"
      icon={
        <PersonAddIcon
          sx={{ color: colors.greenAccent[600], fontSize: '26px' }}
        />
      }
    />
  </Box>
</Grid>
```

Popular UI Libraries

Utility-First CSS Libraries

- Tailwind ❤️
- Emotion
- UnoCSS
- WindiCSS

CSS-In-JS

- styled-components
- vanilla-extract
- Stiches

CSS Libraries

- Bootstrap
- Bulma
- Foundation
- Semantic UI
- Materialize
- Skeleton

React UI Libraries

- Material UI (MUI)
- React Bootstrap
- Chakra UI
- Ant Design
- … and many more!

Class Activity

https://edstem.org/us/courses/91614/lessons/160635/slides/943666

React Routing

How do we make an app feel like a multi-page app within React?

Routing on the Web

Routing refers to mapping URLs to code that handle those URLs.

This can take the form of:

- **Front-end routing**: navigating to new pages without reloading the webpage
- **Back-end routing**: executing HTTP methods such as GET and POST to retrieve and submit data

Routing with React

While normal React doesn't include routing, there are some popular libraries that provide routing to React:

- **React Router**: Client-side routing through the browser, used with SPAs
- **Express.js**: Server-side routing through HTTP requests to a website server, used for SSR

React Router: Install & Use

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

```jsx
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import Home from './Home';
import About from './About';

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/about">About</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home/>} />
        <Route path="/about" element={<About/>} />
      </Routes>
    </BrowserRouter>
  );
}
```

React Router: <Link>

Unlike <a>, <Link> prevents the reload of the whole page, allowing faster navigation.

This also preserves React states due to not needing to reload/re-render the whole page.

```
import { Link } from "react-router-dom";

function Navbar() {
      return (
            <nav>
                  <Link to="/">Home</Link>
                  <Link to="/about">About</Link>
            </nav>
      );
}
```

React Router: <Route>

<Route> maps some URL path to a component.

```
// {baseurl}/home goes to <Home>
<Route path="/home" element={<Home />} />
```

Dynamic routes within the URL, denoted with :, can be specified and accessed with `useParams()`

```
// App.tsx
<Route path="/users/:id" element={<User />} />

// User.tsx
import { useParams } from "react-router-dom";

const User = () => {
      const { id } = useParams();
      return <h1>User {id}</h1>;
}
```

React Router: Nested Routes

Nested Routes let you share layouts, letting you render child components inside parent components, but with different URLs.

```
<Route path="/dashboard" element={<Dashboard />}>
  <Route path="settings" element={<Settings />} />
</Route>
```

This requires an <Outlet> Component in the parent.

```
import { Outlet } from "react-router-dom";

function Dashboard() {
  return (
    <div>
      <h1>Dashboard</h1>
      <Outlet /> {/* Will now display <Settings> from above*/}
    </div>
  );
}
```

React Router: Still an SPA!

Technically, React Router doesn't make your React app a multi-page app, due to not reloading the whole page.

The React app is still technically still a single page (still rendering on top of the single index.html!)

Class Activity

https://edstem.org/us/courses/91614/lessons/160635/slides/943867

React Libraries

What are some important React libraries to learn?

The React Ecosystem

React, while powerful for front-end, is small by itself, mostly handling UI and managing states.

It relies on libraries to better manage state, forms, animation, data fetching, style, and more.

We'll talk about 3 selected libraries:

- **React/TanStack Query**
- **React Hook Form** (Form States and Validation)
- **Motion** ❤️ (Animations)

React/TanStack Query

```
npm i @tanstack/react-query
```

React, or TanStack Query, helps manage server state, caching, and background updates efficiently.

A **server state** refers to more of an external state, your app connecting to external data rather than internal data.

Delegating the handling of this state to an external library like TanStack Query improves performance and takes a lot of headache out of managing server updates and data fetching.

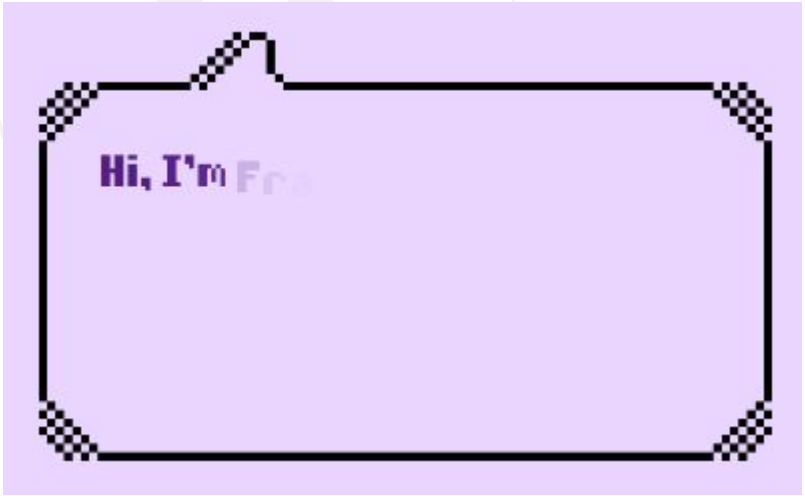
react-hook-form

```
npm i
react-hook-form
```

A library that optimizes forms by removing re-renders and makes them easier to write and verify their inputs.

```tsx
import { useForm } from "react-hook-form";
import type { SubmitHandler } from "react-hook-form";

type FormData = {
  name: string;
  email: string;
};

const MyForm = () => {
  const { register, handleSubmit, formState: { errors } } =
useForm<FormData>();

  const onSubmit: SubmitHandler<FormData> = (data) => {
    console.log(data);
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register('name', { required: true, minLength: 8 })} />
      {errors.name && <span>Name required</span>}
      <input type="email" {...register('email')} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

Motion




```
npm install motion
```

This library helps you make beautiful page animations in React.

It includes support for things like mobile gestures (tapping, dragging), composite animations, orchestration/propagation animations, and much more!

https://motion.dev/docs/react
https://motion.dev/examples

Class Activity: Motion

https://edstem.org/us/courses/91614/lessons/160635/slides/944126