

# **React 3**

**CS571: Building User Interfaces**

**Cole Nelson & Yuhang Zhao**

# Before Lecture

- Clone today's code to your machine.
  - Run the command `npm install` inside of the `starter` , `mid-solution` , and `solution` folders.
- Complete your daily Wordle.
  - We'll be going through it in class.

# Quick Notes...

1. Uncontrolled components moved to React 4.
2. Midterm exam is on Thursday, Oct 26th @ 5:45 pm.
  - a. No class during the afternoon.
  - b. Old exams are on Canvas.
  - c. You will have 1 hour for 30 MC problems.
  - d. You may bring a **single-sided** notesheet.
  - e. Conflict? [Let us know ASAP](#).
  - f. Exam location is TBD.

# Quick Notes...

3. We take academic integrity seriously!
  - a. You may **not** copy another student's code.
  - b. Code from sources such as StackOverflow, ChatGPT, etc. must be cited as such.
4. Get started on projects **early!**
  - a. Done > Started > Empty
  - b. Perfect === 

# Recap: React 1 & 2

A common workflow is...

1. Get some data from an API using `useEffect`
2. Save it to a state variable created using `useState`
3. Create many components, passing down `props`

# Ticket Board

## todo tickets

### Fix Server Outage

Investigate and resolve the server outage issue affecting the production environment.

todo

in progress

done

### Implement Backup Solution

Set up an automated backup solution for critical data and configurations.

todo

in progress

done

### Optimize Codebase

Review and optimize the existing codebase to improve application performance.

todo

in progress

done

### Migrate Email Servers

Migrate email servers to a new infrastructure provider for improved reliability.

todo

in progress

done

### Resolve Network Latency Issue

Investigate and resolve network latency issues affecting remote office connectivity.

todo

in progress

done

### Create Documentation

Create comprehensive documentation for the IT systems and procedures.

todo

in progress

done

### Patch Software Vulnerabilities

Identify and patch software vulnerabilities to mitigate security risks.

todo

in progress

done

### Create Backup Strategy

Develop a backup strategy to ensure data recovery in case of failures.

todo

in progress

done

### Implement Data Encryption

Implement data encryption to enhance data security during transmission and storage.

todo

in progress

done

## inprogress tickets

### Upgrade Database

Upgrade the database system to the latest version to improve performance and security.

todo

in progress

done

### Develop User Authentication System

Create a user authentication system for the new web application.

todo

in progress

done

### Deploy New API Endpoint

Deploy a new API endpoint to support a mobile app feature.

todo

in progress

done

### Implement Two-Factor Authentication

Enhance security by implementing two-factor authentication for user accounts.

todo

in progress

done

### Set Up Monitoring Alerts

Configure monitoring alerts for critical system components to proactively detect issues.

todo

in progress

done

### Deploy Load Balancer

Deploy a load balancer to distribute traffic evenly across multiple servers.

todo

in progress

done

# Your turn!

Create a ticket tracking app! A `TicketBoard` should have three `TicketLane` which can hold many `Ticket`.

Each `Ticket` should display their name and description as well as buttons to move in to "TODO", "In Progress", and "Done" lanes.

Use <https://cs571.org/api/f23/weekly/week06>

[Clone from here.](#)

# Uh-oh!

We need to talk back to our parent?

# What will we learn today?

- What are React Fragments? <></>
- How can we share state in React?
  - Using callbacks
  - Using `useContext`
  - Using `sessionStorage` and `localStorage`
  - Using cookies and an API (next time!)
  - Other libraries (on your own!)...
- How can we handle routing in React?

# Quick Note: React Fragments

A JSX component can only return *one thing*.

Sometimes we use `<></>` instead of `<div></div>`.

`<></>` is a React Fragment, a virtual separator not represented in the real DOM.

[Learn More](#)

# State Management in React

Unrelated, but interesting... [stateofjs](#)

# State Management

How do we talk back to our parent? How do siblings talk to each other?

- Using callbacks
- Using `useContext`
- Using `sessionStorage` and `localStorage`
- Using cookies and an API (next time!)
- Using third-party libraries like (on your own!)...
  - [Redux](#), [Recoil](#), [MobX](#), [XState](#)

# Passing Callbacks

The original way to do child-to-parent communication.

```
const TodoList = (props) => {
  const [items, setItems] = useState();

  const removeItem = (itemId) => {
    // Do Remove!
  }

  return <div>
  {
    items.map(it => <TodoItem key={it.id} {...it} remove={removeItem}/>)
  }
</div>
}
```

# Passing Callbacks

This callback function is then used in the *child* to mutate the *parent*.

```
const TodoItem = (props) => {  
  
  const handleRemove = () => {  
    alert("Removing TODO item!");  
    props.remove(props.id);  
  }  
  
  return <Card>  
    <h2>{props.name}</h2>  
    <Button onClick={handleRemove}>Remove Task</Button>  
  </Card>  
}
```

# Ticket Management

Move tickets from lane to lane via passing callbacks.

# **useContext Hook**

A useful hook for managing state across web apps with large component hierarchies.

# useContext Hook

**Motivation:** How can we effectively manage state for web apps with large component hierarchies?

- Spotify
  - SpotifyLandingPage
  - RecentSearches
    - AuthorCard
    - AuthorImage
    - AuthorName

Passing props down and down and down is known as "props-drilling" and is considered bad practice.

# `useContext` Hook

Three steps to using context.

1. Create and export a context.
2. Provide the context with some value.
3. Use the context in a child component.

Often used in combination with `useState`.

# useContext Hook

A context must be exported.

```
const MyDataContext = createContext([]);  
export default MyDataContext;
```

# useContext Hook

A context must be provided to child component(s).

```
function ParentComponent() {
  const [data, setData] = useState(["Apples", "Oranges", "Bananas", "Grapes"]);
  return (
    <MyDataContext.Provider value={data}>
      <SomeChildComponent />
      <SomeOtherChildComponent />
    </MyDataContext.Provider>
  );
}
```

# useContext Hook

The context can be used by any of child, grandchild, great-grandchild, etc. component(s).

```
function SomeChildComponent() {
  const data = useContext(MyDataContext);
  return (
    { /* Do something interesting with data here! */ }
  );
}
```

[See StackBlitz](#)

# Local Storage



# v/s

# Session Storage

# v/s

# Cookies

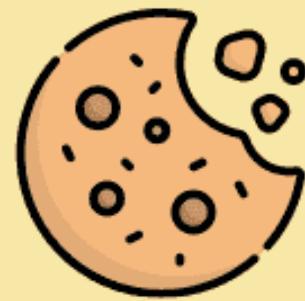


[Image Source](#)  
CS571 Building User Interfaces | Cole Nelson & Yuhang Zhao | Lecture 10: React 3

# Cookies vs. Session vs. Local

These exist in your browser! They will persist even after a page refresh.

- [Wordle](#) uses `sessionStorage` and `localStorage` to store game state.
- [Facebook](#) uses `cookies` to track you.
- [Vanguard](#) uses `cookies` to store temporary session/login credentials.



Criteria	Local Storage	Session Storage	Cookies
Storage Capacity	5-10 mb	5-10 mb	4 kb
Auto Expiry	No	Yes	Yes
Server Side Accessibility	No	No	Yes
Data Transfer HTTP Request	No	No	Yes
Data Persistence	Till manually deleted	Till browser tab is closed	As per expiry TTL set
Image Source	CS571 Building User Interfaces   Cole Nelson & Yuhang Zhao   Lecture 10: React 3		

# Cookies vs. Session vs. Local

Type	Notes
Cookies	Can be set programmatically, but typically set by server through a <code>Set-Cookie</code> header
Session	Set programmatically via <code>sessionStorage</code> , typically used with form data.
Local	Set programmatically via <code>localStorage</code> , typically used with long-lasting data.

# Cookies vs. Session vs. Local

These are all just key-value pairs of *strings*!

Type	Example
Cookies	<code>document.cookie = 'lang=en'</code>
Session	<code>sessionStorage.setItem('name', 'Cole')</code>
Local	<code>localStorage.getItem('lastLogin')</code>

# Let's Persist Some Data!

Using `sessionStorage` or `localStorage`.

[StackBlitz Solution](#) | [Inspitation from WDS](#)

# sessionStorage and localStorage

**Reminder:** sessionStorage and localStorage only store *strings*! Use JSON.parse and JSON.stringify !

## INCORRECT Way

```
sessionStorage.setItem('nums', [2, 6, 19])
const storedNums = sessionStorage.getItem('nums')
```

## Correct Way

```
sessionStorage.setItem('nums', JSON.stringify([2, 6, 19]))
const storedNums = JSON.parse(sessionStorage.getItem('nums'))
```

# Third Party Libraries

Each have their own unique way of managing state.

Examples include...

- [Redux](#)
- [Recoil](#)
- [MobX](#)
- [XState](#)

**Note!** These come and go. See [flux](#).

# **Client Vs. Server-Side Storage**

These are all examples of doing client-side storage.

What if we want to persist data long-term?

Server-side storage with `PUT` , `POST` , and `DELETE` !

# Multi-Page Apps

Knowing how to do state management, how do we manage apps with many pages?

# **React is a *library*, not a *framework*!**

This means that *batteries are not included*. You'll be choosing many of your own tools and libraries!

- **Layout & Design:** [Bootstrap](#) [React-Bootstrap](#),  
[Reactstrap](#), [Material](#), [Elemental](#), [Semantic](#)
- **Routing & Navigation:** [React Router](#), [React Navigation](#), [React Location](#)
- **State Management:** [Redux](#), [Recoil](#), [MobX](#), [XState](#)

# Navigation w/ React Router

See [StackBlitz](#)

# Types of Routers

- `BrowserRouter` : What you typically think of!
- `MemoryRouter` : Same as `BrowserRouter` , but the path is hidden from the browser in memory! 🤯
- `HashRouter` : Support for older browsers.
- `StaticRouter` : Used for server-side rendering.
- `NativeRouter` : We'll use `react-navigation` instead!

# Routing

Using a Router , Routes , and Route !

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Layout />}>
      <Route index element={<Home />} />
      <Route path="about-us" element={<AboutUs />} />
      <Route path="other-info" element={<OtherInfo />} />
      <Route path="*" element={<Home />} />
    </Route>
  </Routes>
</BrowserRouter>
```

# Browser outlet

<Outlet/> shows the component returned by the child route! e.g. in Layout we may see...

```
function Layout() {
  return (
    <>
      <Navbar bg="dark" variant="dark">
        { /* Some navigation links... */ }
      </Navbar>
      <Outlet />
    </>
  );
}
```

# Navigable Components

Notice how each route maps to a component.

```
function Home() {  
  return <h2>Home</h2>  
}  
function AboutUs() {  
  return <h2>About Us :)</h2>  
}  
function OtherInfo() {  
  return <h2>Other Info!</h2>  
}
```

# useNavigate Hook

Useful for programmatic navigation!

```
export default function OtherInfo() {  
  
  const navigate = useNavigate();  
  
  const handleClick = () => {  
    navigate('/home');  
  }  
  
  return <div>  
    <h2>Other Info!</h2>  
    <Button onClick={handleClick}>Back to Home</Button>  
  </div>  
}
```

# Navigation

Navigation for a `BrowserRouter` is done via URLs.

```
<>
  <Navbar bg="dark" variant="dark">
    <Nav className="me-auto">
      <Nav.Link as={Link} to="/">Home</Nav.Link>
      <Nav.Link as={Link} to="/about-us">About Us</Nav.Link>
      <Nav.Link as={Link} to="/other-info">Other Info</Nav.Link>
    </Nav>
  </Navbar>
  <Outlet />
</>
```

# HW5 Demo

Badger Buddies!

The Madison Cat Project



# What did we learn today?

- What are React Fragments? <></>
- How can we share state in React?
  - Using callbacks
  - Using `useContext`
  - Using `sessionStorage` and `localStorage`
  - Using cookies and an API (next time!)
  - Other libraries (on your own!)...
- How can we handle routing in React?

# Questions?