UQCS React Workshop

with Paul Clarke, Kenton Lam, and Olivia Mackenzie-Ross

If you haven't installed React yet:

1. *Install Node.js (nodejs.org)*
2. Create a new folder to store your react projects
3. Use the terminal/console to path find your way to your new folder: cd {directory}
4. Create your react app (This process will take some time) (npx is installed with npm): npx create-react-app {name}
5. Enter your react project's folder: cd {name}
6. Start your development server: npm start
   a. If you already have a browser open, a new tab should open up by the name of localhost:3000

Workshop Repository

link.uqcs.org/react

What is JSX?

- JSX is a form of markup used in React

- JSX allows you to write basic HTML in Javascript and have it render on the page

- Create React App handles the Babel, so we don't have to worry about it

- JSX also changes how CSS/Element attributes work on a page

Before Babel:

```
1  <div>
2    <label className='label' htmlFor='name'>Enter name:</label>
3    <input id='name' type='text' />
4    <button style={{background: 'blue', color: 'white'}}> {button
5  </div>
```

After Babel:

```
1  "use strict";
2
3  React.createElement("div", null, React.createElement("label", {
4    className: "label",
5    htmlFor: "name"
6  }, "Enter name:"), React.createElement("input", {
7    id: "name",
8    type: "text"
9  }), React.createElement("button", {
10   style: {
11     background: 'blue',
12     color: 'white'
13   }
14 }, " ", buttonText));
```

What are **Components?**

- Components are *custom HTML elements*, written in React

- They let you split your UI into reusable pieces
    - Each piece works in isolation

- There exist two types, class and functional components
    - For the purposes of this workshop however, we will be using functional components

- Finally, each component must have a render function, telling the page how to render itself

What's in a Tweet?

Components in the large
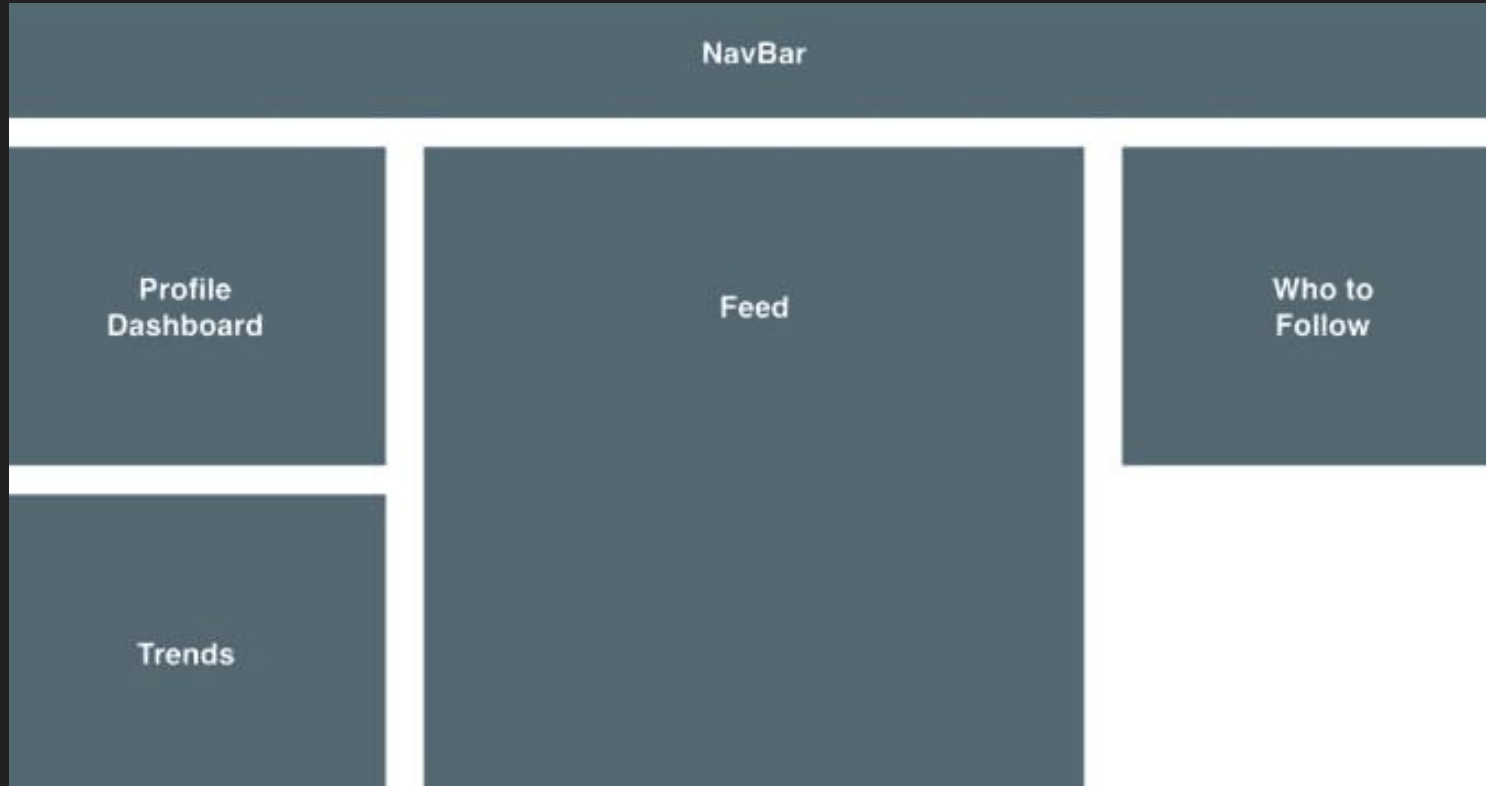
(Image courtesy of programmingwithmosh.com)

How to code a functional component

```
function App() {
  return (
    <div>
      <h1>Hello!</h1>
    </div>
  )
}
```

```
function Tweet() {
    return (
        <div>
            <h3>Tweet Title</h3>
            <h4>Handle</h4>
            <p>Your tweet content message!</p>
        </div>
    )
}
```

What are **Props/Properties?**

- Props are used to pass data through components
    - (Like read-only function parameters in other languages)

- For components to use props, you need to pass it as an argument into the function

- A component with props:

```
< MyComponent my_prop={value}/>
```

Coding Time !!!!!

someecards
user card

What is the **State?**

- The state controls how a component renders, almost like a set of rules it follows when generating HTML

- Each component has a state, and that state belongs *only* to that component

- Whenever the state changes, the component will automatically re-render to make the necessary HTML changes

Defining state variables

- Defining and updating the state is quite easy. The example to right shows you would do it in a functional component

- Whenever the state changes, the component will automatically re-render to make the necessary HTML changes

```
const [variable,setVar] = useState(initialValue)
```

- Always use the function (setVar in this case) to change the state variable, **never** change it directly

```
setVar(newValue)
```

Stateful Component example

```
function MyComponent() {
    const [presses,setPresses] = useState(0)


    return(
        <div>
            <h1>Likes: {presses}</h1>
            <button onClick={() => setPresses(presses+1)}></button>
        </div>
    )
}
```

Updating parent state

- We can pass a function through a component prop to allow child components extra permissions

- Below is pictured a previous example of ours without any child components

```
function MyComponent() {
    const [presses,setPresses] = useState(0)

    return(
        <div>
            <h1>Button Presses: {presses}</h1>
            <button onClick={() => setPresses(presses+1)}></button>
        </div>
    )
}
```

Updating parent state

- If we wanted to make the button it's own component…

- Here's how our original component would look like

```
function MyComponent() {
    const [presses,setPresses] = useState(0)

    return(
        <div>
            < ButtonPresses numpresses={presses} setNewPress={setPresses}/>
        </div>
    )
}
```

Updating parent state

- Our new button component:

```
function ButtonPresses() {
    return(
        <>
            <h1>Button Presses: {props.numpresses}</h1>
            <button onClick={() => props.setNewPress(props.numpresses+1)}></button
        </>
    )
}
```

IT'S TIME TO GET CODING
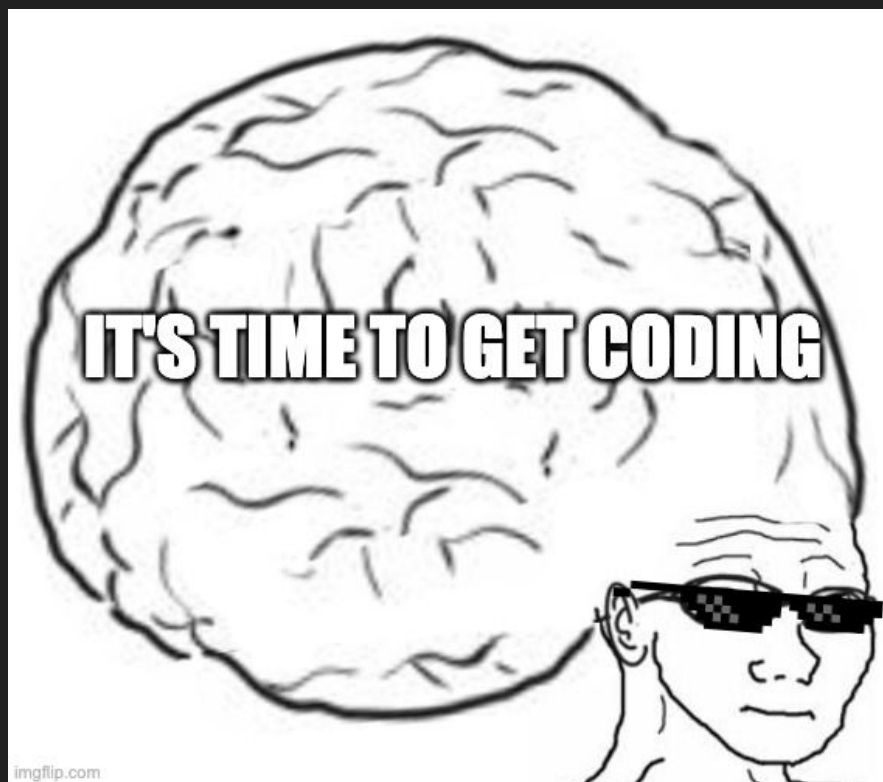
Now it's your turn!

Here are some ideas: (Page 1)

- An online shop
    - Items listed with their price, quantity in stock, and more information
    - You have a cart you can place items into

- A Todo list application
    - You have 'tasks' which you have to complete at which point they disappear
   - Add/Remove tasks, maybe even categorise them

Now it's your turn!

Here are some ideas: (Page 2)

- A Blog
    - You can see a users post, together with their date of posting
    - You can like/dislike the post and maybe even comment

- A Tic Tac Toe Game
    - Using buttons to denote places on a grid
    - Can you design an end game?

When it's time to export your app:

1. Save your application and close down the development server (Ctrl+C on console)

2. Go to console and run: npm run build
   a. This will minimise the number of HTTP requests being made when your application goes live on a server
   b. This will take a couple of seconds/minutes depending on the size of the project

3. Once this process is completed, you'll have an application built in the 'build' folder

4. This folder can be uploaded straight to Github Pages / wherever you want. No server-side code needed!
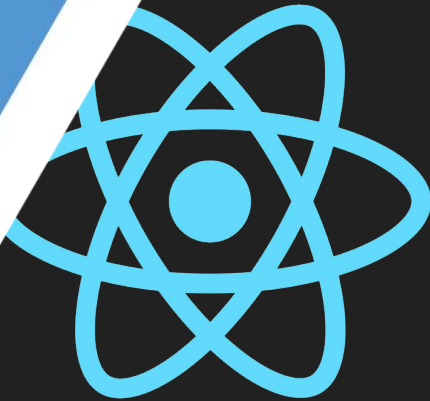
Thank you

What are **Hooks?**

- We use hooks to modify the state and any other side effects of components

- Side **effects** include component lifecycle methods, which are similar to Javascript onload events

- Custom Hooks (not covered)

- We use *useState* to change state variables
    - To start, include it like so:

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

- useState accepts an initial argument and returns a state variable with a function to change it

```
const [variable,setVar] = useState(initialValue)
```