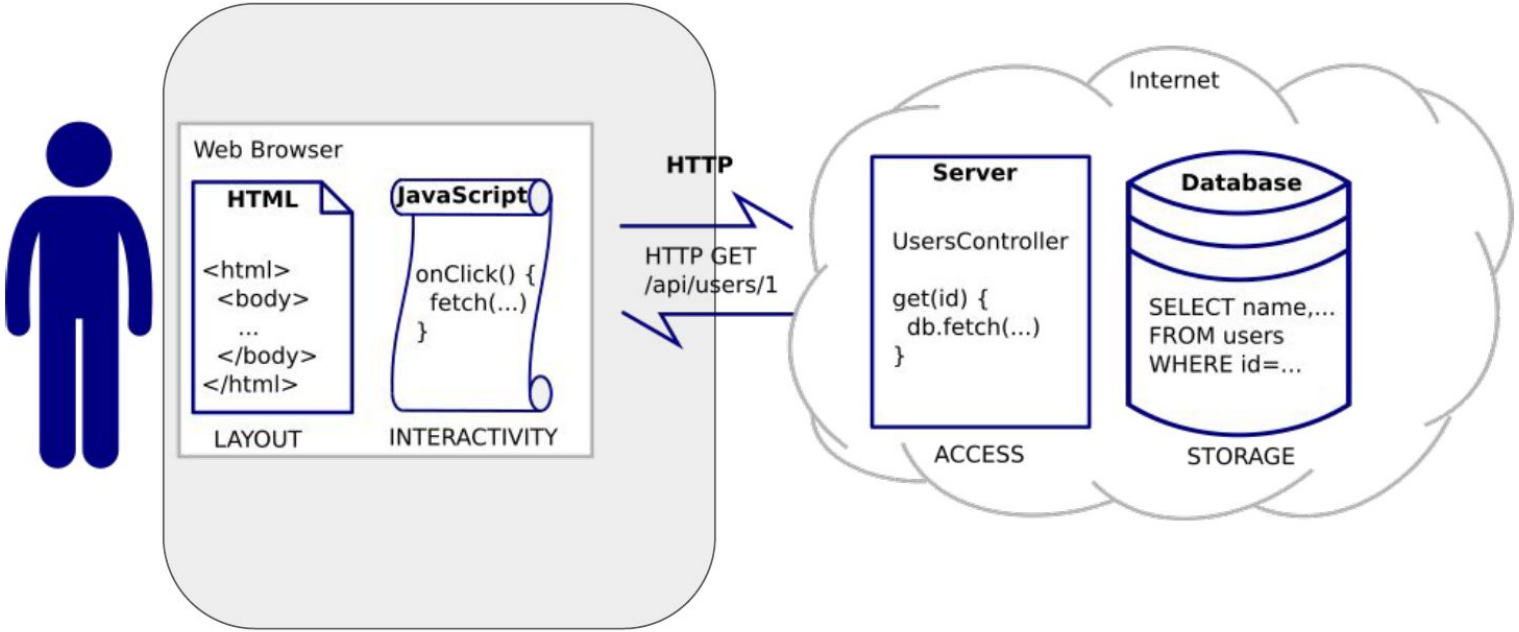
React Context

Focus Area



INCEPTIONU

The repo for this series of slides

Get them from GIT

We will be working with this code as examples. You can fetch the resources we will be using in class from here:

https://github.com/EvolveU-FSD/c7-superheroes

Agenda

- Introduce “wrapper” components
- Introduce global state (React context)
- Hide/Show parts of the app based on state

“Wrapper” Components

We create wrapper components when we want to “enhance” their children or selectively process them in some way

React Router uses this concept, Each Route is a child of the Routes component. The Routes component conditionally renders one of its children depending on the path

Every component has a property passed to it, “children”, we can either use it by itself to simply render the children, or iterate over it to make some changes to each of its children or selectively render them

Example Wrapper Component

Declaration:

```
const BlueBackground= ({children}) => {
   return (
      <div style = {{backgroundColor: “blue”}}>
         {children}
      </div>
   )
}
```

Usage:

```
<BlueBackground>
   <div>Hello, I am a child of BlueBackground
</BlueBackground>
```

INCEPTIONU

Managing Global State

Say hello to Context components (and the hook)

In react “Context Components” facilitate the housing of data at high levels in an applications component tree. Lower level components can depend upon and fetch the latest value of the context at any time.

It is sort of like useState, but at a higher level of abstraction.

When do we use them?

- When we have something that belongs “at the top” of a react application which affects rendering all over the component tree (think dark theme or light theme)
- When we have a piece of information that greatly affects which components should be visible (think authentication)
- We have some client side information that can be updated from pretty much anywhere in the application (think of a shopping cart)

Managing Global State

Context components have three parts

1. A “Context” that defines the data available in the context and the functions or callbacks that operate on that data.

2. A “Context Provider” is a wrapper component that manages the values of a context with “useState” and implements the callbacks for the context.

3. A “Context Consumer” is any other component in the system. It can use a hook to get the latest value of the context from the “nearest” provider. It can call the callbacks in the context to change the state.

Let's try it out

Create the context

- AuthenticationContext => React.createContext()
- Add a "username",
- Add a stub function for the "logIn" and "logOut" method

Create the provider

- Write a component that manages the "username" using useState
- Also create the callbacks for "logIn" and "logOut" to modify this state
- Use the "children" prop inside AuthenticationContext.Provider component

Wrap the App inside the provider

useContext

Now that we have the provider and the context, we'll use the useContext hook to make one of our components a "consumer"

useContext is called and we pass in the Context we created earlier, we can save the result of this in a variable, authContext.

This will allow us to access the values inside our global state and change them using the functions inside the Context