Evolve Full Stack Developer

Intro To ReactJS

INCEPTIONU

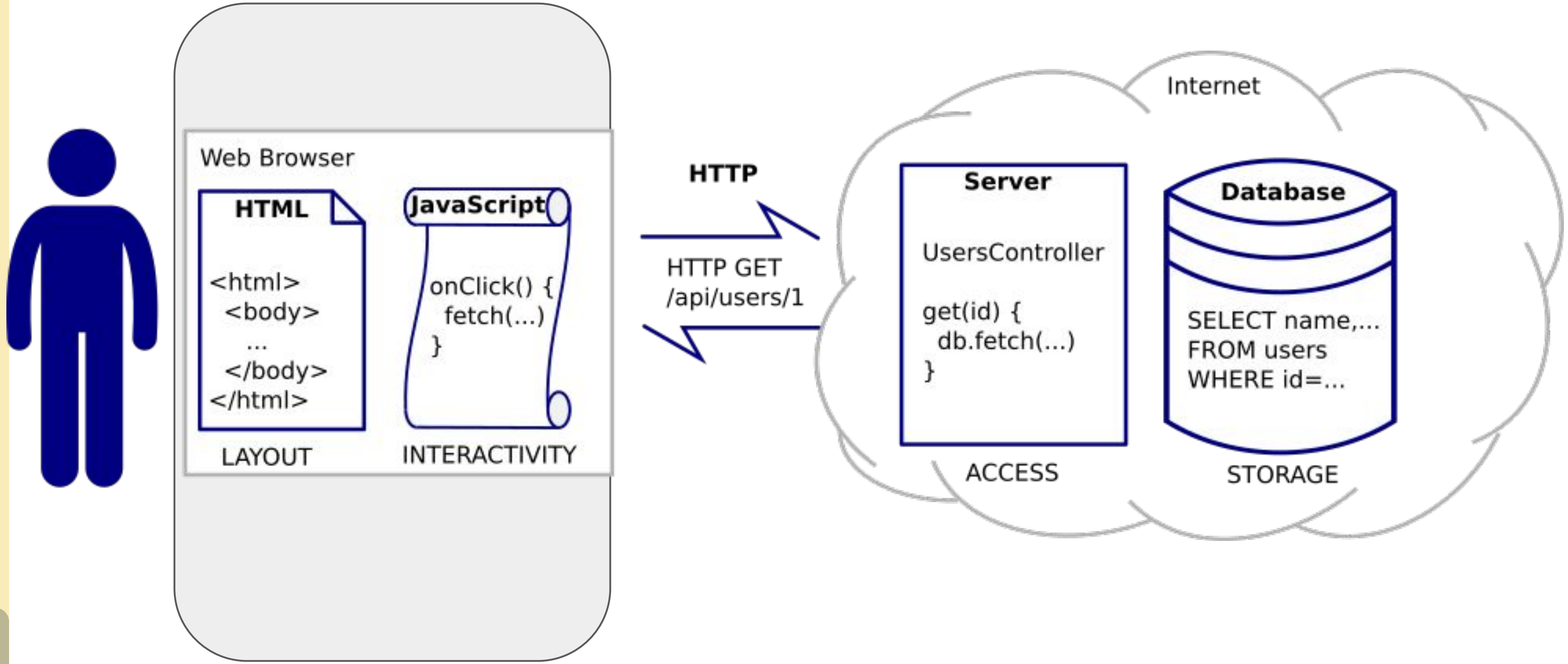
Big Picture Idea - What is React?

- React is a JavaScript **library** created by Facebook (Meta)
- React is a **User Interface** (UI) library
- React is a tool for building **UI Components (frontend)**
- React provides **consistent behaviour** across browsers and versions
- React "***cleans up***" the APIs for DOM elements and events

Encapsulates a set of **Best Practices** *for building performant, consistent web apps*

INCEPTIONU

Focus Area


Web Browser
HTML
<html>
<body>
...
</body>
</html>
LAYOUT
JavaScript
onClick() {
fetch(...)
}
INTERACTIVITY
HTTP
HTTP GET
/api/users/1
Internet
Server
UsersController
get(id) {
db.fetch(...)
}
ACCESS
Database
SELECT name,...
FROM users
WHERE id=...
STORAGE

Exercise #1 - Create React App

- Run the command: `npm create vite@latest highgear-ex1`
- Take a quick walkthrough of the file structure to understand the parts:
  - `main.jsx` – the start of it all
  - `App.jsx` – your app!
  - `package.json` – the “node project” configuration
- Install the dependencies: `npm install`
- Run the command: `npm run dev`

JSX

- What is it?
  - A syntax extension to JavaScript
  - Looks like HTML
  - Helps us to create “components” using the tags we know from HTML
- How is it different from HTML?
  - We use it to lay out our custom “components”, not just the base HTML tags
  - Because we are writing it in a JavaScript file, we can pass properties to the “tags” that result from JavaScript expressions!
  - There are a few other differences, including the way we have to write certain properties (re: “class” and “for”)

What is a Component?

- Components are **independent** and **reusable** bits of code (re: Functions)
- Components have 4 characteristics:

❏ the component’s name - must be **Capitalized**

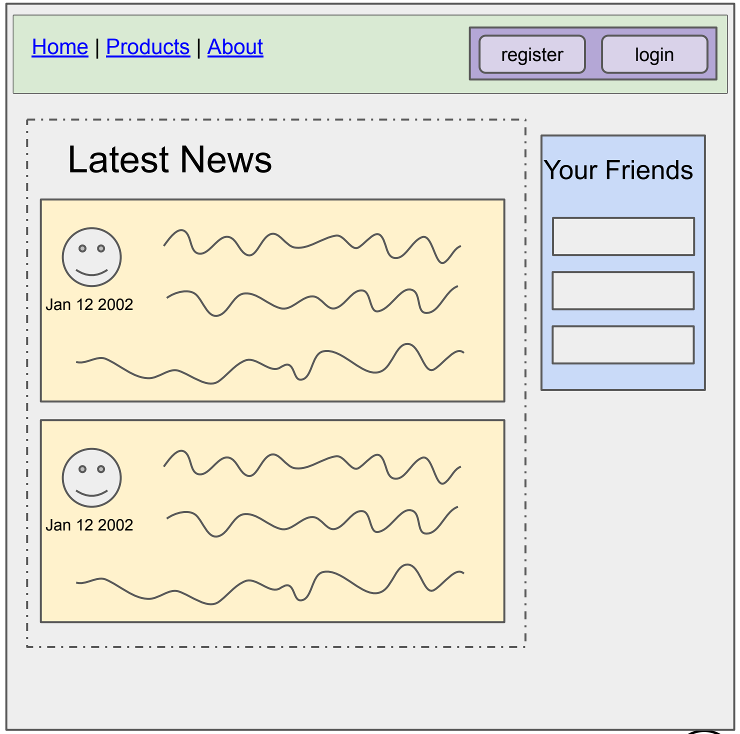
❏ the return value - must be **JSX**

❏ the parameter values (properties) passed in - lovingly called **props**

❏ the data that, when it changes, the component re-renders - called **state**

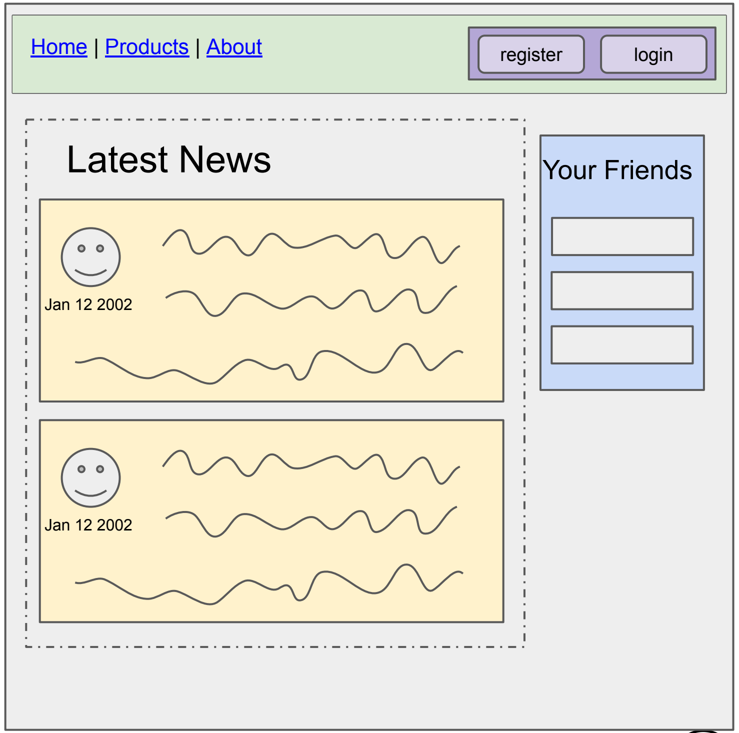
Where Do I Find Components

- Everywhere!



Where Do I Find Components

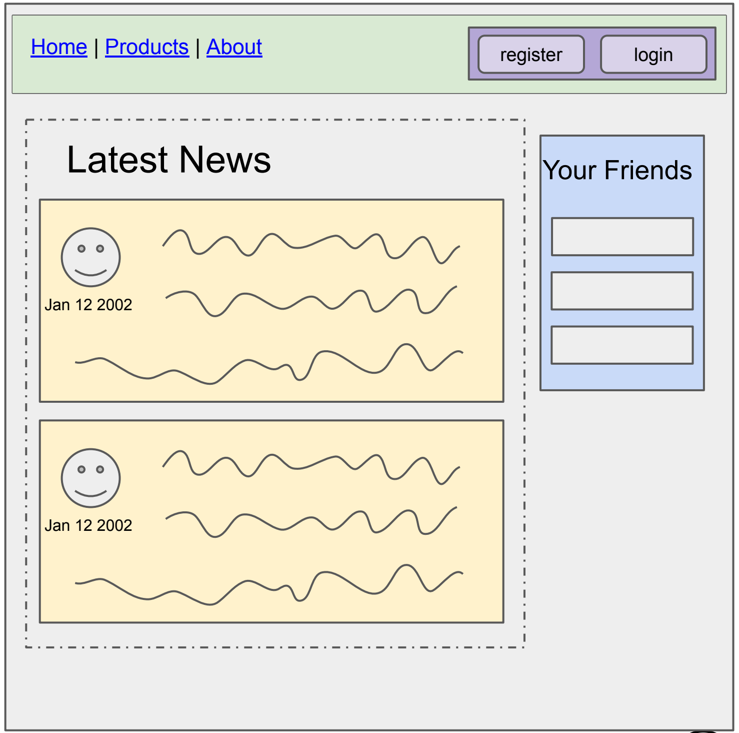
- Everywhere!
- **App.js** -- the WHOLE app -- is a component!!



INCEPTIONU

Where Do I Find Components

- Everywhere!
- **App.js** -- the WHOLE app -- is a component!!
- Pages
- “Zones”
  - Cards, Forms, Items
- Groups of “Controls”
  - navigation & side bars, menus, radio buttons
- Individual “Controls”
  - Buttons, Headers, Images, Icons



It is Components all the way down!

App.js:

```
<NavBar />
<NewsFeed />
<FriendsList />
```

NavBar.js:

```
<Link "home" />
<Link "products" />
<Link "about" />
<LoginButtons />
```

LoginButtons.js:

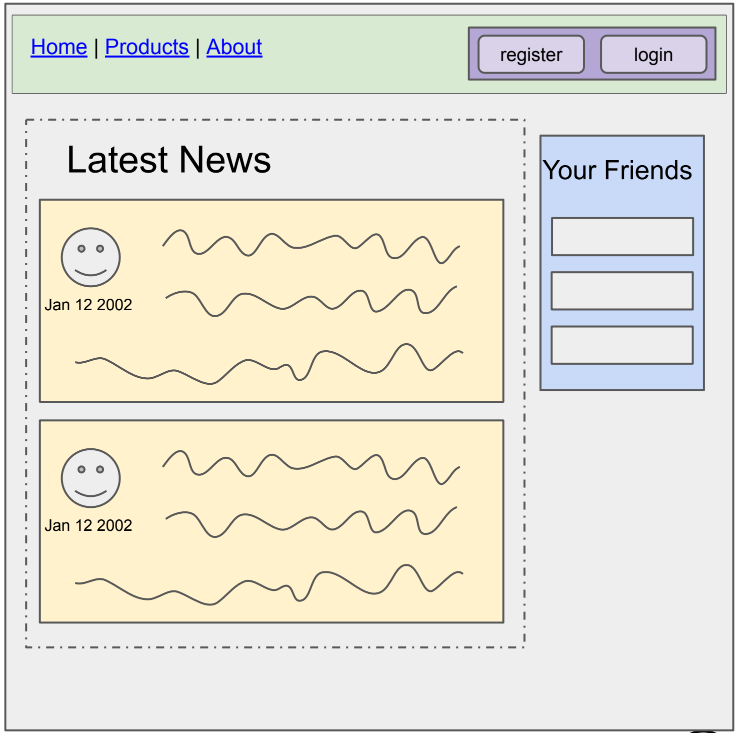
```
<RegisterButton />
<LoginButton />
```

NewsFeed.js:

```
<h1>Latest News</h1>
newsItems.map(() => <NewsItem />)
```

FriendsList.js:

```
<h3>Your Friends</h3>
myFriends.map(() => <FriendItem />)
```



INCEPTIONU

It is Components all the way down!

App.js:

```
<NavBar />
<NewsFeed />
<FriendsList />
```

NavBar.js:

```
<Link "home" />
<Link "products" />
<Link "about" />
<LoginButtons />
```

LoginButtons.js:

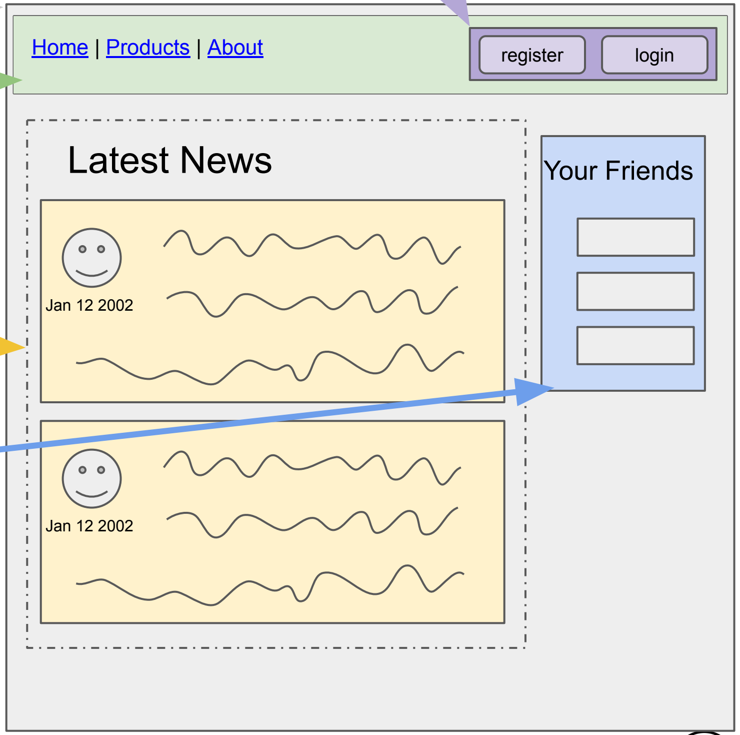
```
<RegisterButton />
<LoginButton />
```

NewsFeed.js:

```
<h1>Latest News</h1>
newsItems.map(() => <NewsItem />)
```

FriendsList.js:

```
<h3>Your Friends</h3>
myFriends.map(() => <FriendItem />)
```



It is Components all the way down!

App.js:

```
<NavBar />
<NewsFeed />
<FriendsList />
```

NavBar.js:

```
<Link "home" />
<Link "products" />
<Link "about" />
<LoginButtons />
```

LoginButtons.js:

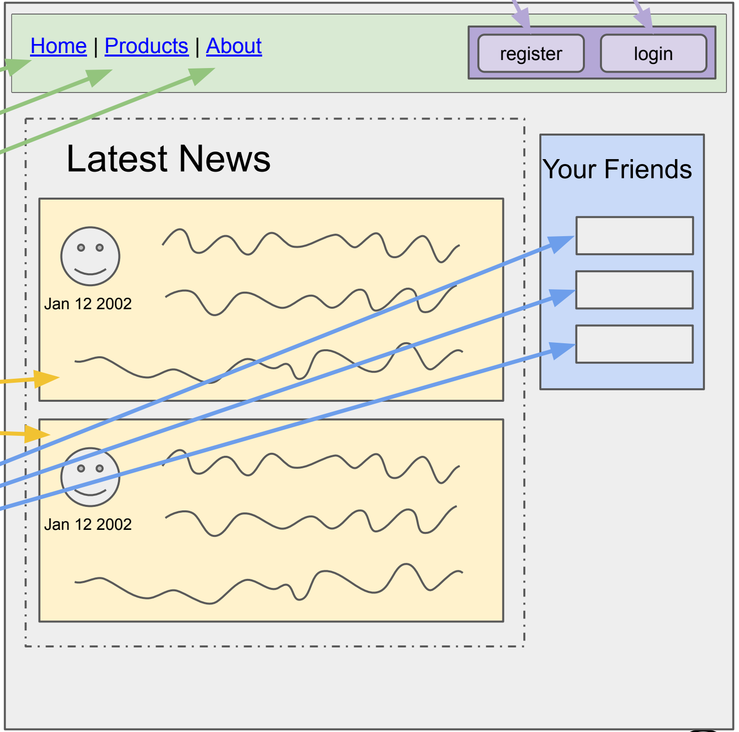
```
<RegisterButton />
<LoginButton />
```

NewsFeed.js:

```
<h1>Latest News</h1>
newsItems.map(() => <NewsItem />)
```

FriendsList.js:

```
<h3>Your Friends</h3>
myFriends.map(() => <FriendItem />)
```



INCEPTIONU

A Big Change in the React Community

- Traditionally:
    - Everyone learned about **Class Components**
    - *Functional Components* were “interesting”
    - *Hooks* were “weird” - “they don’t work like Class events do”
- In 2020, the React world learned that:
    - **WE WERE DOING HOOKS WRONG**
    - Hooks are ***awesome****!* They allow us to focus on *tasks* and you can *separate concerns.*
    - Functional Components make for ***leaner & cleaner code***
- In 2021 and beyond:
    - Class and Functional Components are both valid, will still exist
    - Most **new apps** will use Functional Components primarily and exclusively
    - Class Components will, slowly, become *legacy, maintenance* and *quaint*

INCEPTIONU

Today

- We will only cover **Functional Components** (and some key Hooks!)
- Everything we cover today is valid when working with Class Components
    - (well, except for the Hooks)

```
const LittlestComponent = () => {
    return <div/>;
}
```

Name is Capitalized

```
const LittlestComponent = () => {

    return <div/>;

}
```

- ✓ the component's name - must be **Capitalized**
- ❑ the return value - must be **JSX**
- ❑ the parameter values (properties) passed in - lovingly called **props**
- ❑ the data that, when it changes, the component re-renders - called **state**

INCEPTIONU

```
const LittlestComponent = () => {
    return <div/>;
}
```

Returns JSX

- ✓ the component's name - must be **Capitalized**
- ✓ the return value - must be **JSX**
- ❏ the parameter values (properties) passed in - lovingly called **props**
- ❏ the data that, when it changes, the component re-renders - called **state**

INCEPTIONU

Properties are passed in as arguments to the function

```
const LittlestComponent = (props) => {
    return <div/>;
}
```

- ✓ the component’s name - must be **Capitalized**
- ✓ the return value - must be **JSX**
- ✓ the parameter values (properties) passed in - lovingly called **props**
- ❏ the data that, when it changes, the component re-renders - called **state**

No longer the littlest

Hey look! A Hook!!

```
const SecondLittlestComponent = (props)    {
  const [fred, setFred] = useState();
  return <div/>;
}
```

State variable

State Set function

- ✓ the component’s name - must be **Capitalized**
- ✓ the return value - must be **JSX**
- ✓ the parameter values (properties) passed in - lovingly called **props**
- ✓ the data that, when it changes, the component re-renders - called **state**

INCEPTIONU

Break!

and then Labs!

INCEPTIONU

A Tree

A React app is a ***tree of components***.

App.js renders its JSX -- its ***child components***

Those components render their JSX -- their child components

... and so on

```
const App = () => {
  return <LittlestComponent />;
}
```

```
const LittlestComponent = (props) => {
  return <p>Hello World!</p>;
}
```

A Quick Word About State

```
const SecondLittlestComponent = (props) => {

  const [fred, setFred] = useState();

  return <div/>;

}
```

The `useState()` hook is a React API that:

- returns 2 values: a **state variable** and a **state set function**
- initializes the state variable to the value passed to `useState()` -- or **undefined** if no value is passed in

When you use the state set function to change the value of the state variable, React will re-render the current component (that is, re-draw the UI for this component)

INCEPTIONU

A Quick Word About Props

```
const App = () => {
  return <LittlestComponent message={"Hi C8!"} />;
}
```

App itself doesn't render any text to the browser

Attribute names on the JSX tag...

```
const LittlestComponent = (props) => {
  let text = props.message;
  return <p>{text}</p>;
}
```

...become keys in the component's ***props***

LittlestComponent ends up rendering "Hi C8!" to the browser

INCEPTIONU

Can we play now?

INCEPTIONU

Exercise #2 - Playing With State

You will be editing App.js. You will remove some code, and add new code. No need to worry about "breaking things" - we can always return to the results of Exercise #1 that we have stored in ***source control***!

1. In the `return` statement of App.js, remove all of the code between inside the `<header>` tags (but keep the <header> tags themselves). We no longer want the React icon, the paragraph text, or the hyperlink to reactjs.org.
2. Use the useState hook to add a state variable named `count` & its associated setter function, `setCount`
3. Enhance the `return` function to display the current value of this state variable.
4. Add a `<button>` to the UI that when clicked increases the value of the state variable by `1`.

Commit your results to Git!

Exercise #3 - Components, State and Props

1. In App.js, use the **useState** hook to add a new state variable **counter2** & its associated setter function
2. Create 2 new components
   - **Displayer.js**: expects a single value passed to its ***props*** and renders that value as paragraph text
   - **Adder.js**: expects a single ***state set function*** as its prop and contains a button that when clicked calls the state set function increasing the value by 1
3. Have App.js render these two new components, passing them the new state variable (counter2) and its setter function

You should see 2 buttons on the screen, and two values being output. Each button controls the increase of its one associated value.

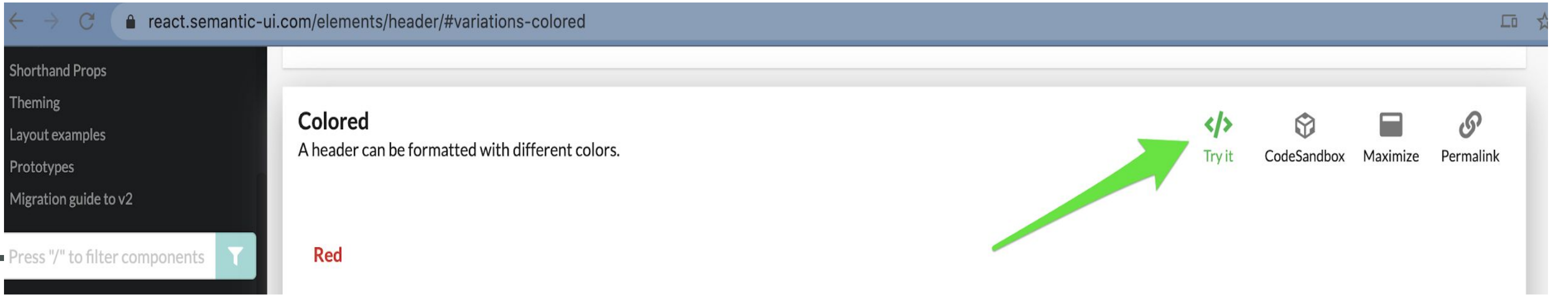
Commit your results to Git!

INCEPTIONU

Bonus Exercise - Add Some Style

There are several approaches to style components in React. We can use native CSS, a CSS framework like Tailwind, or use an existing *styled component library* such as **Semantic UI** or **Material UI** (there are several - just google for "react styled component libraries" to see several "top 10 lists")
For this exercise, add some style using the method of your choice. If you want to try Semantic UI, here's something to get you started:

1. Start with **Option 1** of [Get Started page of Semantic-UI React](https://react.semantic-ui.com/usage) (https://react.semantic-ui.com/usage)
2. In the **Displayer** component, use a colored **<Header>** component as seen:
   https://react.semantic-ui.com >> Elements >> Header >> Variations >> Colored
   to display the `count2` value. You can see an example of how to import the Semantic-UI library and how to use a <Header> component by clicking:



Colored
A header can be formatted with different colors.

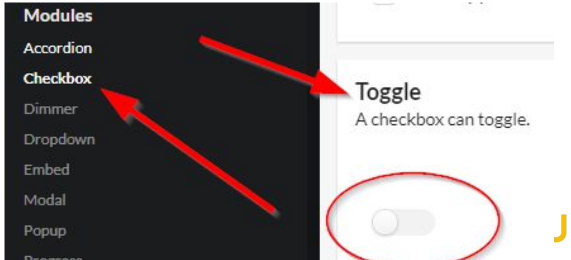
Red

BONUS BONUS Exercise - Dynamic Style!

- Add another state variable to **App.js** called `colour`. Initialize it to the value `"red"`.
- Add a function in **App.js** called `swapColours()` that changes the value of `colour` to `"green"` if the current value is `"red"`, or vice-versa.
- In **Adder.js**, add a <Checkbox> from Semantic-UI such that when it is clicked on, it causes the value of `colour` to switch between `"red"` and `"green"`
- In **Displayer.js**, use the value of `colour` as a parameter to the existing <Header> component.

The result is you have a toggle switch that changes the display of count2 to be either **red** text or **green** text.

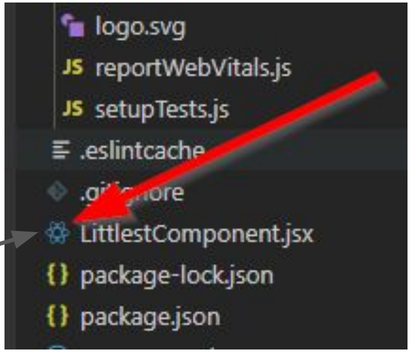
Bonus Bonus Bonus: get the <Checkbox> to render as a slider:



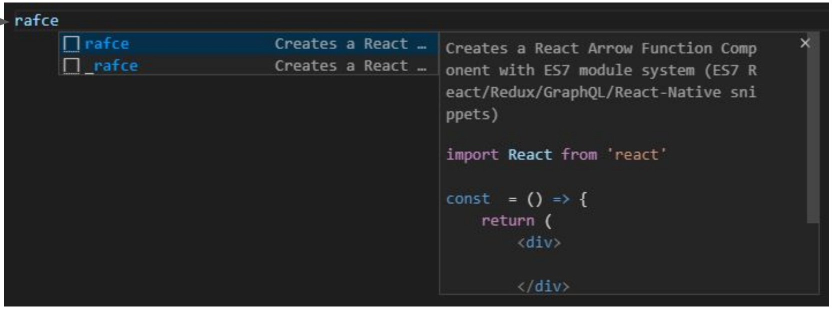
Commit your results to Git!

Take Away Thoughts

- In VS Code, name component files ***.jsx** and they will get a small React icon
- In VS Code, install the ES7 React/Redux/GraphQL extension
  - Gives you dozens of snippets such as **rafce**
- Only use **Functional Components**
  - Converting a typical Class Component to a Functional Component takes ***1-2 minutes***
- In VS Code, install the **GitLens** extension
  - Shows histories of project, branches and file
- Install React Developer Tools to your browser
  - Shows React Component Hierarchy



```
logo.svg
reportWebVitals.js
setupTests.js
.eslintcache
.gitignore
LittlestComponent.jsx
package-lock.json
package.json
```

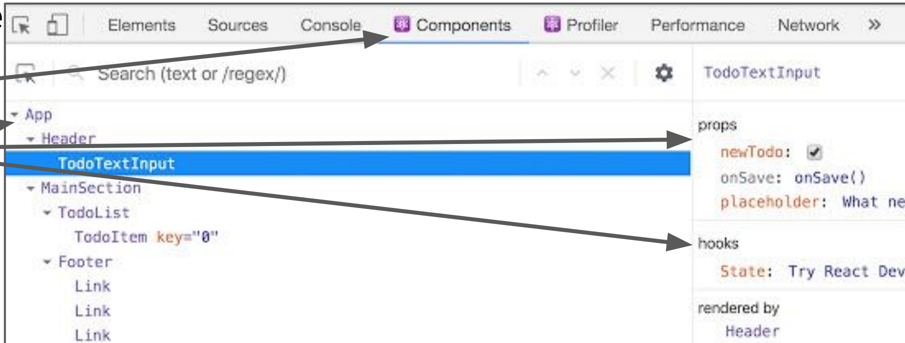


```
rafce
  rafce      Creates a React …    Creates a React Arrow Function Comp
  _rafce     Creates a React …    onent with ES7 module system (ES7 R
                                  eact/Redux/GraphQL/React-Native sni
                                  ppets)

                                  import React from 'react'

                                  const  = () => {
                                      return (
                                          <div>

                                          </div>
```



```
▾ App                          TodoTextInput
  ▾ Header
      TodoTextInput            props
  ▾ MainSection                  newTodo: ✓
    ▾ TodoList                   onSave: onSave()
        TodoItem key="0"         placeholder: What ne
    ▾ Footer
        Link                   hooks
        Link                     State: Try React Dev
        Link
                               rendered by
                                 Header
```

Still have more questions?

Now's the time to ask.

INCEPTIONU