CPSC 455

Applied Industry Skills

May 28, 2022

Workshop 2 - React & Redux

Reminders

- Fill out the form with your group name & members & repo NOW
- Project Proposals will be graded in the 2nd half of this workshop
- Scrum Updates due tomorrow 10pm (we will cover this later)
- Keep a close eye on the #announcements channel!
    - Important info regarding scheduling, homework, surveys
- Please use 1st week labs as office hours (after mini-lecture)

Plan for Today

1. Learn about React with Hooks
2. Learn about Redux

LUNCH

3. My experience working at D2L + career Q&A
4. Go over next assignment + project progress

BREAKOUT

5. Peer/TA review project topic + rough design (+ late grading)

Intro

- We have so much to cover today!
  - It took me over a month at work to *start* feeling comfortable with React
  - We have 2 hours. Yay!
  - So, my goal is to get you familiar with the concepts, and provide some examples
  - But, the majority of the learning will happen as you work on the individual assignment, work on the project, and find some tutorials to follow!

ReactJS

What is it?

- React is a JavaScript **Library**
  - Most people just call it a framework
    - Debate: https://digitalya.co/blog/is-react-a-framework-or-library/
  - It leaves much of the stack up to developers (state management, build tools, testing tools)
  - Just controls the **View** layer

What is it?

- It is **Declarative**
  - GREAT article: https://tylermcginnis.com/imperative-vs-declarative-programming/
  - In vanilla JS, you manipulate the DOM yourself. In React, you tell the DOM what it should contain, and it manipulates itself

Components

- Based on **~*Components*~**
    - Everything is components
    - Part of knowing how to use React is knowing how to break your ideas down into component

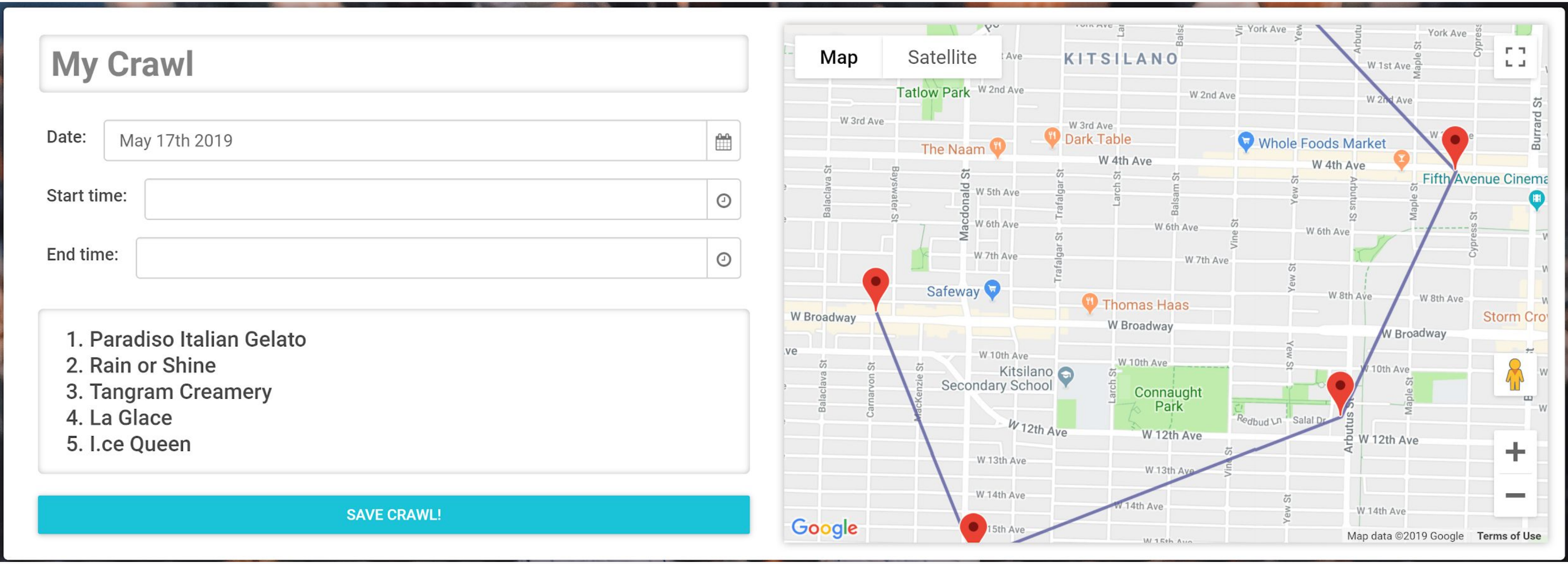
Components

Examples of components:

- Search bar
- Blog Post
- Recipe Card (hint hint)
- Button (fancier than just HTML `<button>`)
  - OR button that has specific functionality tied to it that you would like to reuse
- Profile Photo
- Seriously, anything

Components

http://vancrawler.herokuapp.com/



My Crawl

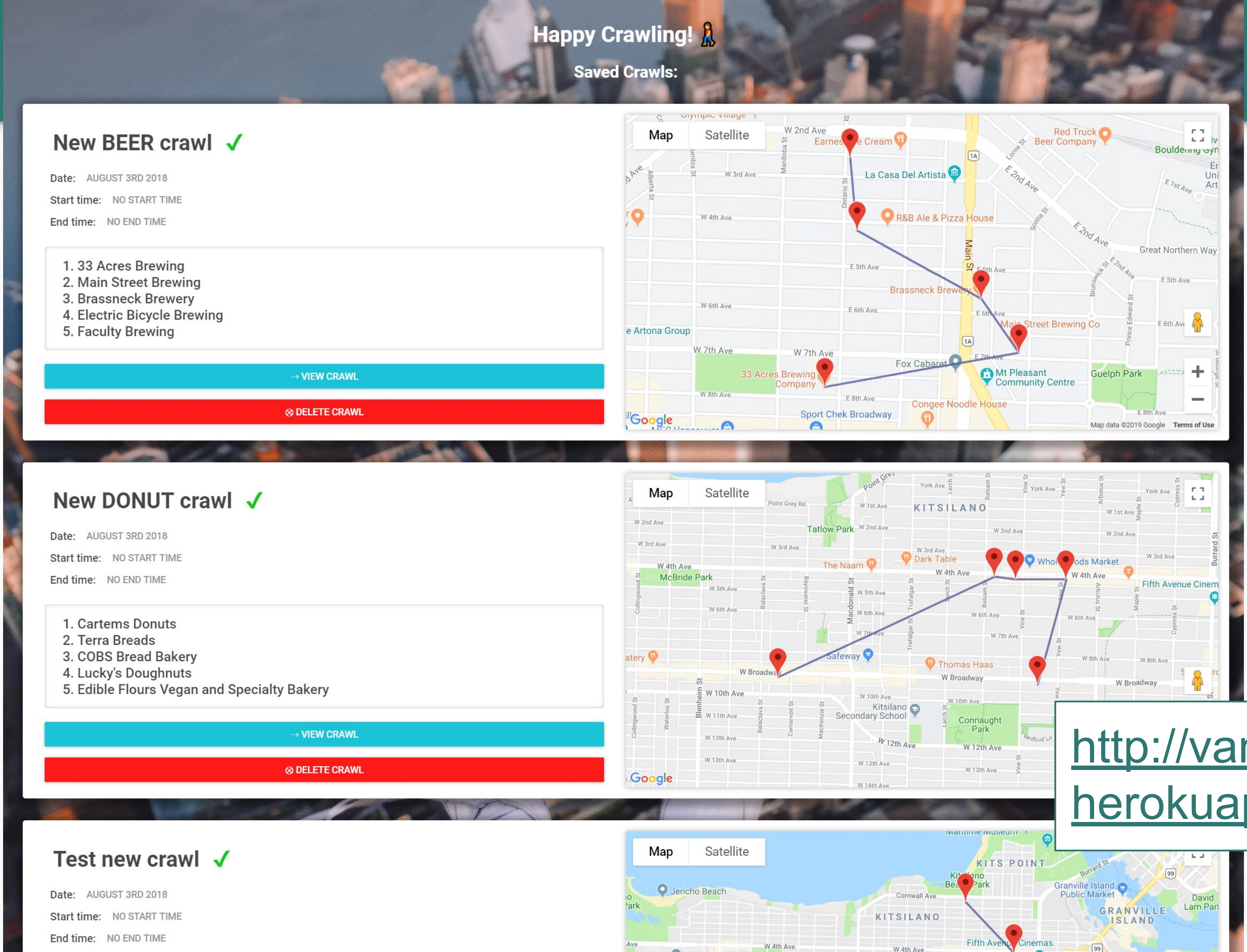
Date: May 17th 2019

Start time:

End time:

1. Paradiso Italian Gelato
2. Rain or Shine
3. Tangram Creamery
4. La Glace
5. I.ce Queen

React



Happy Crawling! 🚶

Saved Crawls:

New BEER crawl ✓

Date: AUGUST 3RD 2018

Start time: NO START TIME

End time: NO END TIME

1. 33 Acres Brewing
2. Main Street Brewing
3. Brassneck Brewery
4. Electric Bicycle Brewing
5. Faculty Brewing

→ VIEW CRAWL

⊗ DELETE CRAWL

New DONUT crawl ✓

Date: AUGUST 3RD 2018

Start time: NO START TIME

End time: NO END TIME

1. Cartems Donuts
2. Terra Breads
3. COBS Bread Bakery
4. Lucky's Doughnuts
5. Edible Flours Vegan and Specialty Bakery

→ VIEW CRAWL

⊗ DELETE CRAWL

Test new crawl ✓

Date: AUGUST 3RD 2018

Start time: NO START TIME

End time: NO END TIME

http://vancrawler.herokuapp.com

Components

- To review:
  - The buttons were not separate component b/c they just acted like regular HTML buttons (we just had the two styles using css classes - `red-button` and `blue-button`)
  - Map was its own component b/c I wanted to separate out so much complex functionality (readability!!)
  - Each “Crawl Event” was its own card b/c it was a repeated view!

Components

- Components always have a “view” to them
- They usually have a functional aspect (but sometimes not)
- A good rule of thumb is: **If you are rendering part of your view in more than one place (e.g. You have a profile photo that appears in blog postings as well as in the navigation bar), then it should be its own component**

JSX

- Before we start making components, we have to learn about JSX
- JSX is separate from React, but almost always used with React
- Essentially, you can use HTML inside of JS (wow!)
- (It’s just JS “behind the scenes”, but it LOOKS like HTML)
- Read more here: https://www.reactenlightenment.com/react-jsx/5.1.html

```
const element = <h1>Hello, World!</h1>;
```

JSX

- This allows you do more interesting things easier

```
function greet(user) {
    if(user) {
        return <h1>Hello, {user}!</h1>;
    }
    return <h1>Hello, stranger!</h1>;
}
```

Note: Curly braces indicate “process this part as JS”. Good for passing in variables!

React

Go here:

codesandbox.io , then click “React” and paste this into App.js ->

```
export default function App() {
 //add your consts here
 return (
   <div>
     <h1>Hello World</h1>
   </div>
 );
}
```

- Task 1: Instead of directly returning `<h1>Hello World</h1>`, make a const and pass it in to the return statement
- Task 2: Add a name variable, and pass it in to say “Hello, Pam!”

Let’s create the solution together here...

```
export default function App() {
 //add your consts here
 return (
   <div>
     <h1>Hello World</h1>
   </div>
 );
}
```

Components

- So, as you may have noticed, in our last exercise, we were actually modifying a component!
- The component was called “App”

```
export default function App() {
 return (
   <div>
     <h1>Hello World</h1>
   </div>
 );
}
```

Components

- There are **functional** components and **class-based** components
    - Excellent article about the differences, with examples! https://www.twilio.com/blog/react-choose-functional-components
- We are going to learn **functional** components today

```
export default function InputForm() {
 return (
   <div>
     <input type="text" id="firstname">
   </div>
 );
}
```

React

Class components
Functional components
imgflip.com

- Has lifecycles & lifecycle methods
- Lots of boilerplate, especially with redux
- Must have a render()
- Necessary evil

- Has hooks
- Very lean
- Must return a React node
- 95% of components you'll write

Stolen from Stephanie Mah's slides!!

Components

- Always capitalize your component name
- Only needs to return a react node ("HTML" tags)

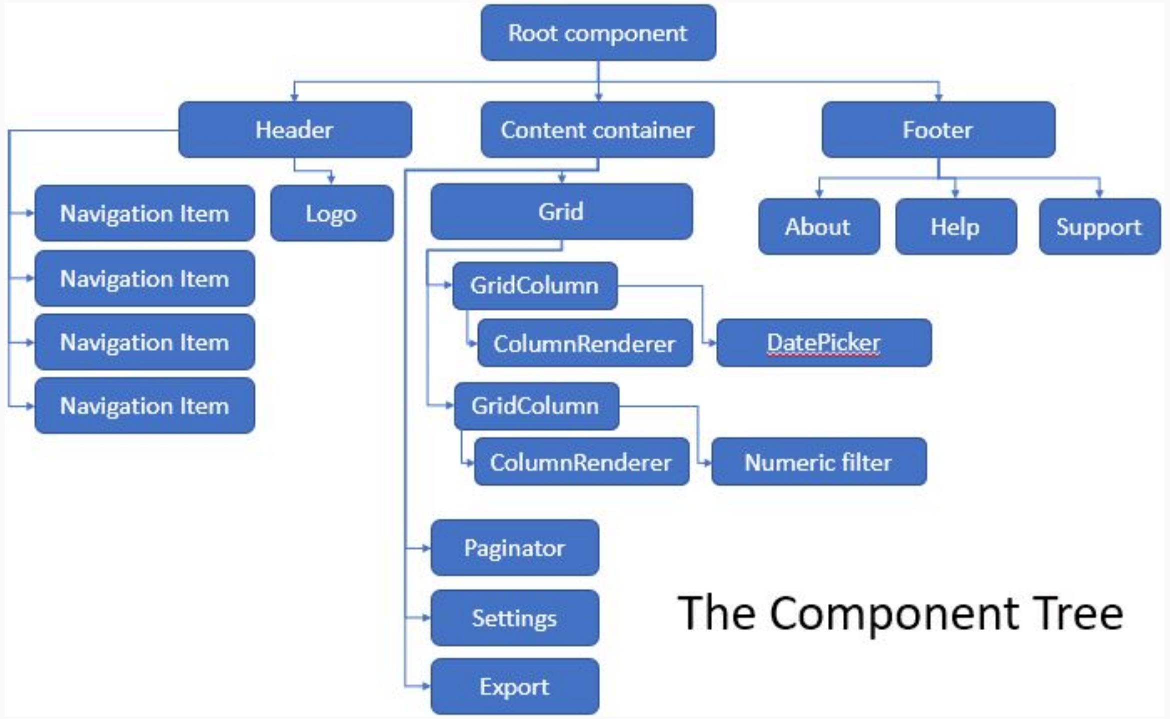
```
export default function InputForm() {
 return (
   <div>
     <input type="text" id="firstname">
   </div>
 );
}
```

Components

```
export default function InputForm() {
 return (
   <div>
     <NameInput/>
     <AgeInput/>
     <AddressInput/>
     <SubmitButton/>
   </div>
 );
}
```

- Once you create a component, you can use it in parts of your app!
- Kind of like a glorified HTML tag
- Reminder - look up how to **export** and import components between files

Components



- Will make more sense with complex components
- Entire app is a component tree

Components

Hierarchy/Nesting activity:

1. Create a new component called “Greetings” that simply renders 3 “Hello” components (from previous example)
2. Make yet another component, called “SoManyGreetings” that renders 3 “Greetings” components

Note: Check **index.js** if your top-level component is not rendering. It’s expecting to render a component called App, so you may need to pass in a different component to index.js

Components

- Some code to get you started
- Check the naming of your files & components
- Use import statements to access components from other files

```
export default function Hello() {
 return (
   <div>
     <h1>Hello World</h1>
   </div>
 );
}
```

```
import Hello from "./Hello.js";

export default function Greetings() {
    return <div>hi</div>;
}
```

Props

- Components accept **props** (properties)!
- Props only flow down the component tree (are passed “down” to nested components)

```
<GroceryStore>
  <Item fruit="apple"/>
  <Item fruit="orange"/>
  <Item fruit="tomato"/>
</GroceryStore>
```

```
export default function Item(props) {
    return <h1>
        {props.fruit}
    </h1>;
}
```

Note: Remember, curly braces means the inner content is interpreted as JS!

Props

- You can name props whatever you'd like, but it's good to be descriptive
- You may pass in as many props as you'd like
- Props are read-only! Cannot do `this.props.fruit = “banana”`

```
<GroceryStore>
  <Item thingy="apple" price="$1" expiry="tomorrow"/>
  <Item thingy="toilet paper" price="$500" />
</GroceryStore>
```

Props

- Next task for this example:
    - Pass down a different name for each “Hello” component!
    - Your “Hello” component will accept the name as props, and display it.
    - If you have extra time, add another property!

Styling ✨

- Since “class” is a reserved word, we need to use a different word - **className**!

```
class Item extends React.Component {
  render() {
    return <div className="alert">
                I am an apple.
        </div>;
  }
}           //this is a class-based component
```

```
<!--CSS file-->

.alert {
    background-color: red;
}
```

REACT HOOKS

Hooks & State

- Until now, all of our data has been static
  - We’re passing around hard-coded values
- Let’s add some functionality!
  - To do that, we’ll need to know about
    - State
    - Hooks

Let’s Start Coding!

Optionally start from a fresh Codesandbox, OR undo the changes in your `index.js` file to import and use the `App` component at the root.

```
/src
    index.js
    App.js
```

(Can leave other files there, just ignore them)

Button.js

Let's start with a simple functional component

- Make a new file `Button.js` in the /src folder

```
export default function Button() {
    return <h1>Is this really a button?</h1>;
}
```

App.js

- We have to import our Button component in App.js
- And then use it
- Using export statement differently for fun :)

```
import "./styles.css";
import Button from "./Button";

function App() {
 return (
   <div className="App">
     <Button />
   </div>
 );
}

export default App;
```

Button.js

Let’s add some basic functionality

```
export default function Button() {
    return (
        <div>
            <h1>I want 5 scoops of ice cream!</h1>
            <button onClick={() => console.log("clicked!")}>
                Click Me
            </button>
        </div>
    );
}
```

State

Let’s talk about the data in our app..

We will have a few different ways of managing data:

1. MongoDB (for long-term storage)
   a. Ex. All of your customers’ data
2. Store/Context (with Redux or Context)
   a. Global app data -> current username, language
3. State (the most fleeting)

State

- State is a **JSON object** that stores some fleeting data about our app
- It stores the “state” of our app at that point in time
- Any component can have its own state, but it doesn’t need to
- Examples:
    - how many times a button has been clicked
    - the text in an input field
    - a boolean on/off toggle

Hooks

- Introduced in React 16.8
- Only used in functional components
- Many different kinds of hooks
  - useState
  - useEffect
  - Custom Hooks
- “Hooks are functions that let you “hook into” React state and lifecycle features from function components”

Hooks

- Two Rules of Hooks
  - Only call Hooks at the top level. Don’t call Hooks inside loops, conditions, or nested functions.
  - Only call Hooks from React function components. Don’t call Hooks from regular JavaScript functions.

- The React Docs are AMAZING https://reactjs.org/docs/hooks-state.html
- Another good resource: https://www.smashingmagazine.com/2020/04/react-hooks-best-practices/#call-hooks-react-components

useState Hook

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

- `useState` allows you to hook into React state
- Here, we create a new piece of state `state.count`
- `count` is initialized to `0`
  - State can be any data type - num, string, array, obj..
- We are also creating the `setCount` function which will allow us to modify `count`

useState Hook

1. At the top of Button.js, we need to import **useState**
2. Inside the function, we'll use the **useState** hook

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

export default function Button() {
    const [count, setCount] = useState(0);
    return (
        ...
```

Using state.count

Now let’s use our newly-created piece of state! So Easy!

```
return (
        <div>
            <h1>I want {count} scoops of ice cream!</h1>
            <button onClick={() => console.log('clicked!')}>
                Click Me
            </button>
        </div>);
```

Modifying state.count

We simply use the setCount function we initialized earlier.

```
return (
        <div>
            <h1>I want {count} scoops of ice cream!</h1>
            <button onClick={() => setCount(count + 1)}>
                More Ice Cream
            </button>
        </div>);
```

Task Time!

1. Start the initial (default) number of scoops at 5
2. Increase the scoop counter by 2 on each click
3. Add a second button to decrement the counter

(~5 mins)

Useful Visibility

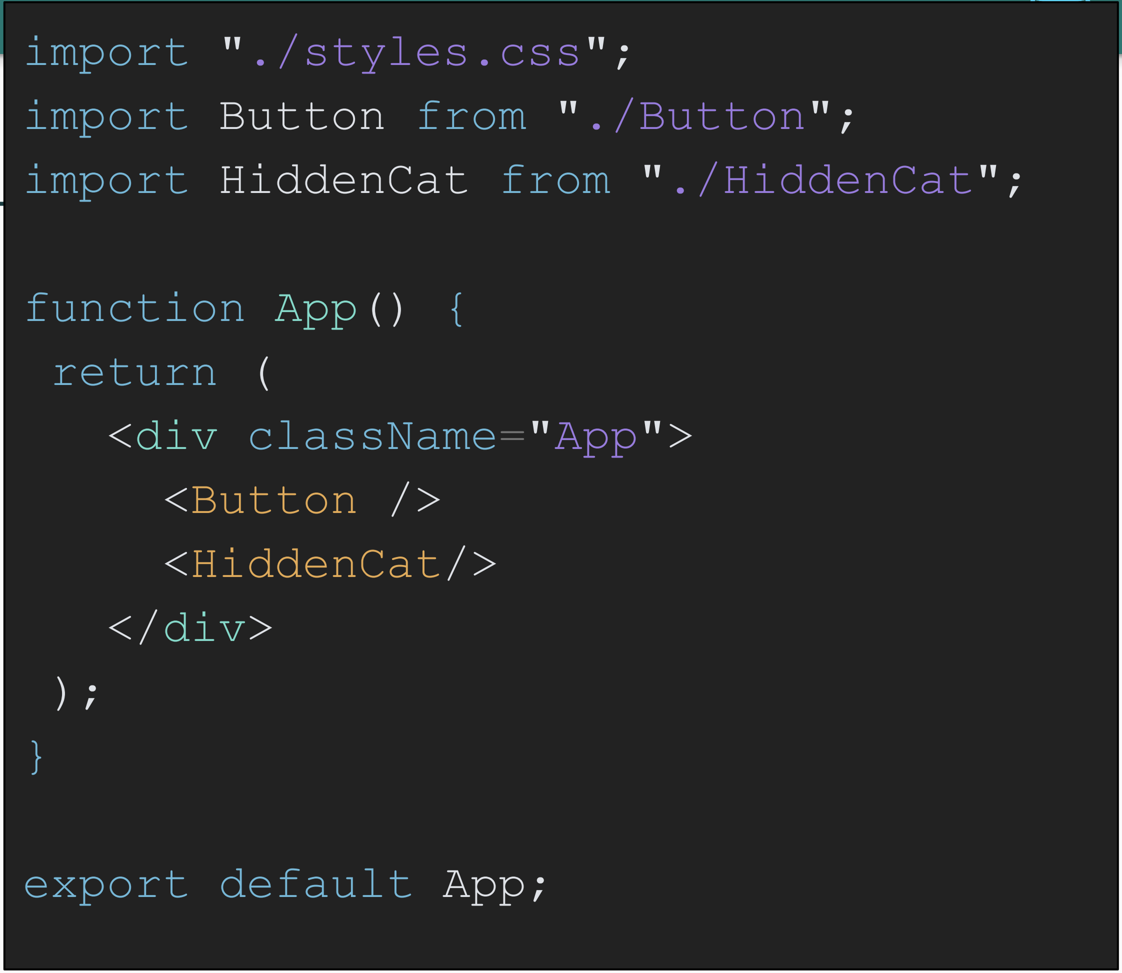
Let’s make a new component in the src folder called HiddenCat.js

```
function HiddenCat() {
    return (
        <div>
            <img src="your_own_cat_url.jpg"/>
        </div>);
}

export default HiddenCat;
```

Useful Visibility

Make sure you import your new HiddenCat component into App.js



```
import "./styles.css";
import Button from "./Button";
import HiddenCat from "./HiddenCat";

function App() {
 return (
   <div className="App">
     <Button />
     <HiddenCat/>
   </div>
 );
}

export default App;
```

Useful Visibility

Add the useState hook to HiddenCat to set isVisible to true or false

- What is the default value here?

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

function HiddenCat() {

    const [isVisible, setIsVisible] = useState(false);
    return (
        ...
```

Incredibly Useful JSX

- Wrap the image element in your `isVisible` boolean
- The element will conditionally appear! (So useful!)

```
return (
        <div>
            {isVisible &&
                ( <img src="your_own_cat_url.jpg"/> )
            }
        </div>);
```

Task Time

1. Create a button element in HiddenCat
2. Have the onClick of the button toggle the visibility of the cat

(~3 mins)

useEffect Hook

- The next-most-important hook!
- Replaces lifecycle functions (for those familiar with React < 16.8)
    - ComponentDidMount
    - ComponentDidUpdate
- Commonly used for:
    - Data fetching
    - setting up a subscription
    - manually changing the DOM

useEffect Hook

- useEffect runs either:
  - When the component first renders ONLY
  - When the component first renders AND everytime the component re-renders
  - When the component first renders AND everytime a specific state updates

Let’s make a new Component

1. Make a component called TimeAndDate.js in the /src folder.

```
export default function TimeAndDate() {
    return (<div>The date is: </div>);
}
```

2. In App.js import your new component, and insert it above <HiddenCat/>

Let’s start with what we know

What have we done here?

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

export default function TimeAndDate() {
    const [dateTime, setDateTime] = useState(new Date());
    return (
        <div>
            The date is: {dateTime.toLocaleTimeString()}
        </div>);
}
```

Adding useEffect

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

export default function TimeAndDate() {
    const [dateTime, setDateTime] = useState(new Date());

    useEffect(() => {
        setDateTime(new Date());
    }, []);

    return <div>The date is: {dateTime.toLocaleTimeString()}</div>;
}
```

When does useEffect run?

- When the component first renders ONLY
- When the component first renders AND everytime the component re-renders
- When the component first renders AND everytime a specific state updates

```
useEffect(() => {
        setDateTime(new Date());
    },[]);
```

```
useEffect(() => {
        setDateTime(new Date());
    });
```

```
useEffect(() => {
        setDateTime(new Date());
    },[count]);
```

When does useEffect run?

- When the component first renders AND everytime the component re-renders

```
useEffect(() => {
        setDateTime(new Date());
    });
```

What will happen if we use this case?
(Pass in no second param)

Final Step!

Let’s have the dateTime refresh every second.

```
useEffect(() => {
        setInterval(() => setDateTime(new Date()), 1000);
    },[]);
```

Another example

useEffect is often used for fetching data, so it will make more sense when you start calling APIs!

```
useEffect(() => {
   let active = true;
    const fetchData = async () => {
    const response = await
fetch(`https://swapi.dev/api/people/${props.id}/`);
    const newData = await response.json();
      if (active) {
       setFetchedId(props.id);
       setData(newData);
    }
  };
  fetchData();
}, [props.id]);
```

Performance

- Components get re-rendered every time their **state** or **props** change
    - We don’t have to update it ourselves.. We just change the data it relies on
- Components ONLY get re-rendered if their state or props change
    - This means entire sections of the app will stay the same if its “data” stays the same
- Read more! https://lucybain.com/blog/2017/react-js-when-to-rerender/

A Note on Custom Hooks

- Custom Hooks are Hooks made up of other hooks
  - Ex. a hook with both useState and useEffect
- By convention, prefix your custom hook with “use”
  - Ex. useStatus, useWindowWidth
- Good resource: https://dev.to/damcosset/how-to-create-custom-hooks-in-react-44nd

Redux

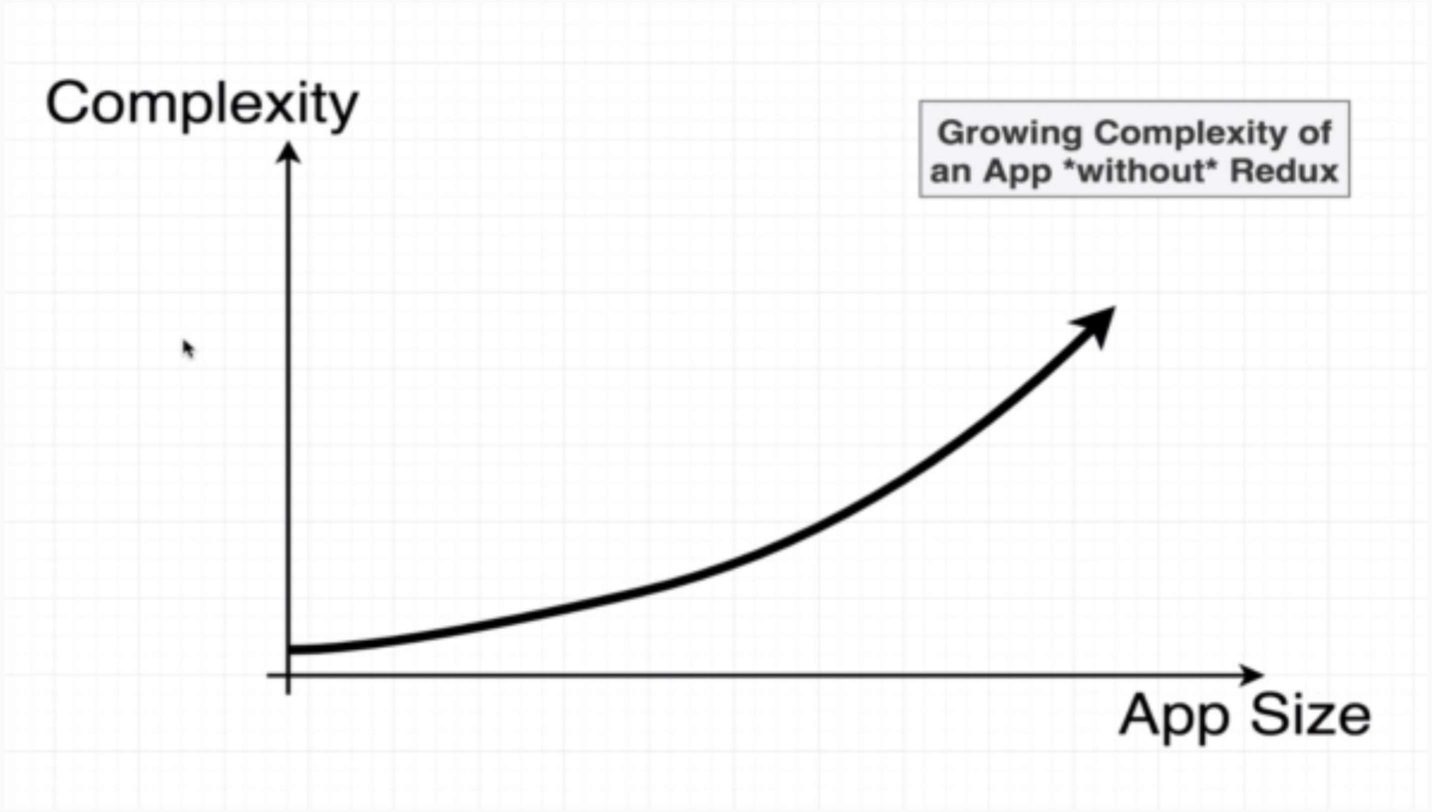
Redux (same slide from state)

Let’s talk about the data in our app..

We have a few different ways of managing data

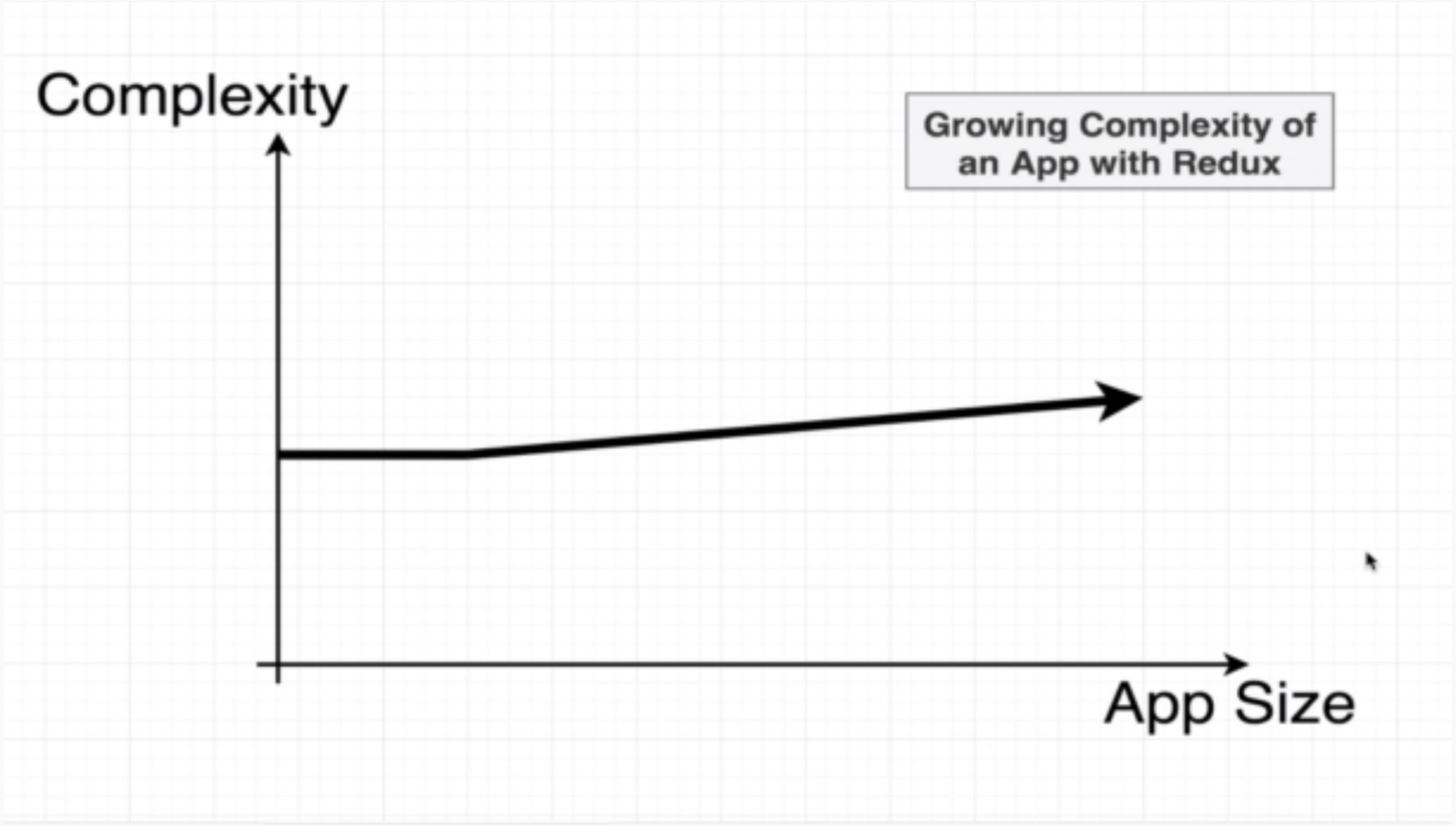
1. MongoDB (for long-term storage)
2. Redux (for tracking impermanent data in your app)
3. State (the most fleeting)

Redux



https://www.udemy.com/react-redux/

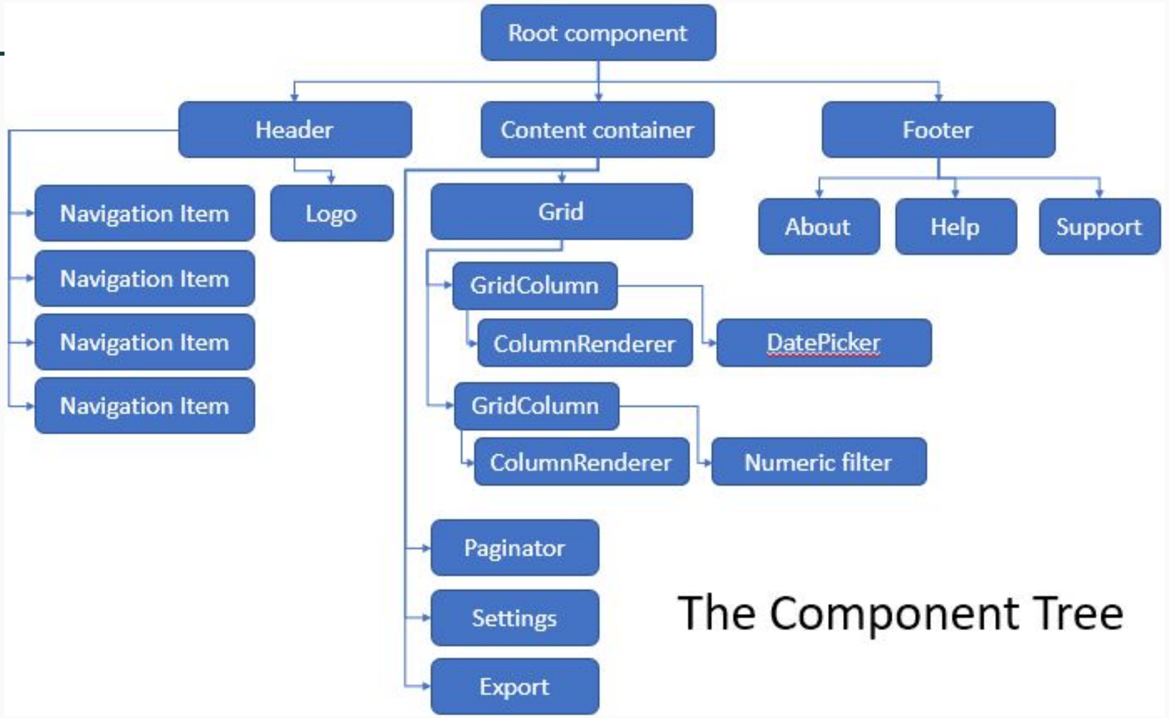
Redux



https://www.udemy.com/react-redux/

What is it?

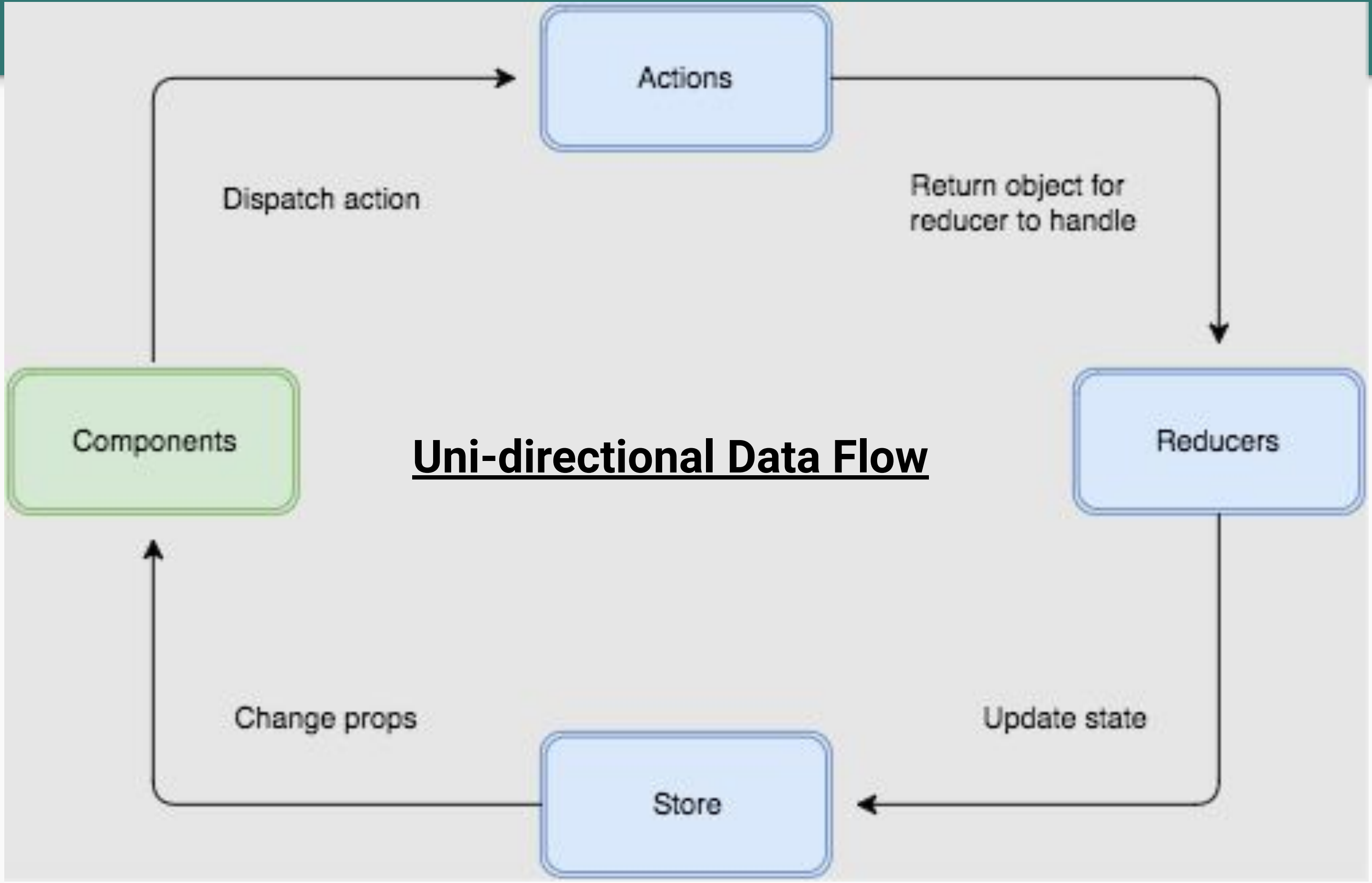
- React state can get really tangled
- Trees of components
- Have to pass props through parent component
- Read about “lifting state up”



What is it?

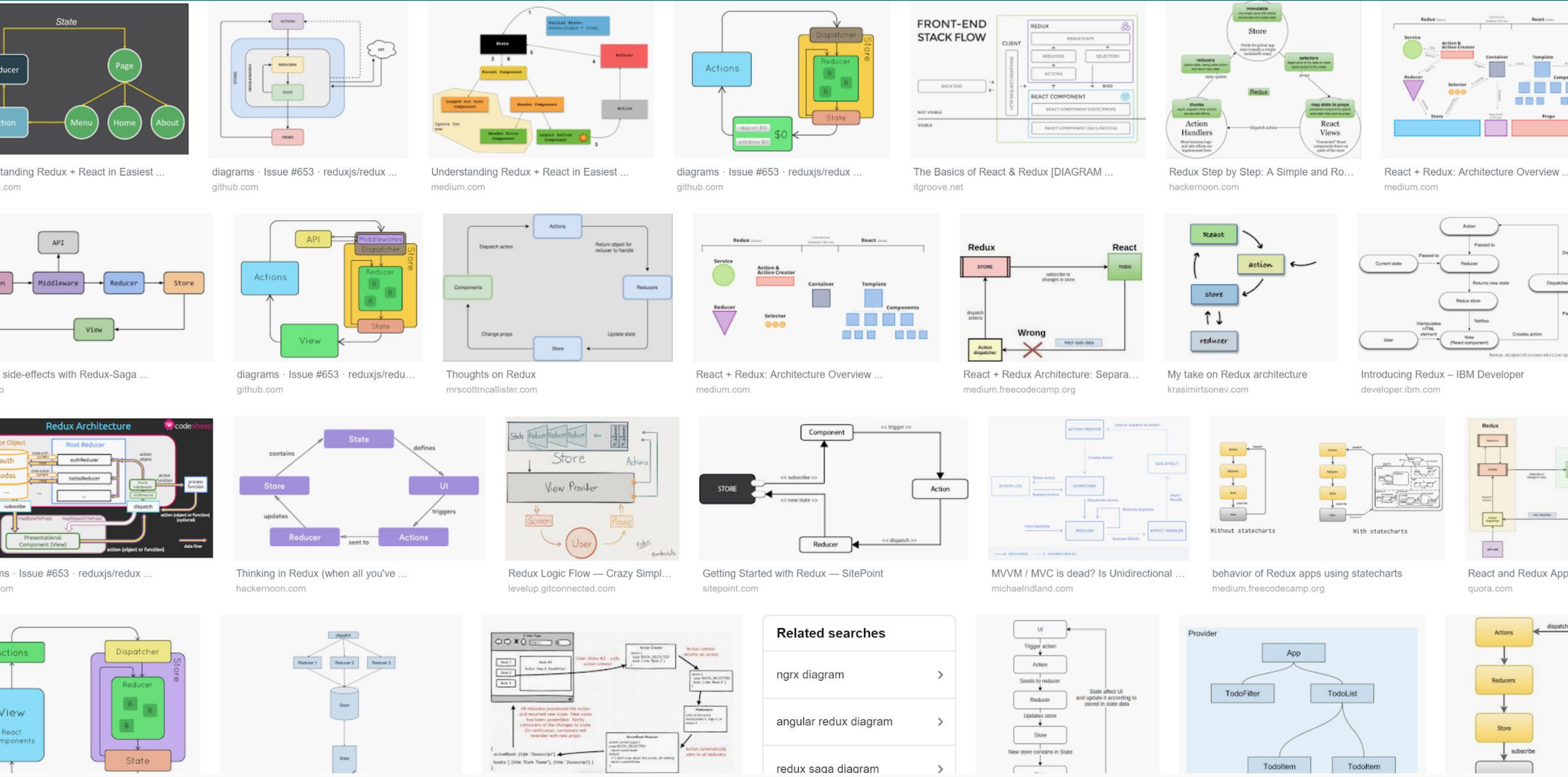
- Manages the **state** of your entire app
- Puts the whole **state** into one central **store**
  - **Single-source of truth!!**
- The only way to modify the **store** is for your **view** (React components) to call an **action**
- A **reducer** uses the **old state** + the **action** to create the **new state**
- State is immutable: https://redux.js.org/faq/immutable-data

Flow



http://www.mrscottmcallister.com/thoughts-on-redux/

Redux


tanding Redux + React in Easiest ...
.com
diagrams · Issue #653 · reduxjs/redux ...
github.com
Understanding Redux + React in Easiest ...
medium.com
diagrams · Issue #653 · reduxjs/redux ...
github.com
The Basics of React & Redux [DIAGRAM ...
itgroove.net
Redux Step by Step: A Simple and Ro...
hackernoon.com
React + Redux: Architecture Overview ...
medium.com
side-effects with Redux-Saga ...
diagrams · Issue #653 · reduxjs/redu...
github.com
Thoughts on Redux
mrscottmcallister.com
React + Redux: Architecture Overview ...
medium.com
React + Redux Architecture: Separa...
medium.freecodecamp.org
My take on Redux architecture
krasimirtsonev.com
Introducing Redux – IBM Developer
developer.ibm.com
ns · Issue #653 · reduxjs/redux ...
om
Thinking in Redux (when all you've ...
hackernoon.com
Redux Logic Flow — Crazy Simpl...
levelup.gitconnected.com
Getting Started with Redux — SitePoint
sitepoint.com
MVVM / MVC is dead? Is Unidirectional ...
michaelridland.com
behavior of Redux apps using statecharts
medium.freecodecamp.org
React and Redux App
quora.com
Related searches
ngrx diagram
angular redux diagram
redux saga diagram

Example

- I used to do a step-by-step example for Redux, but it’s slow and doesn’t add much value
- So instead, I will show you a completed example, and we can talk about the pieces!
- Link to my example: https://github.com/danyakarras/react-redux-button-counter-2022
- You can fork & clone if you wish!

Example

Actions

- Are called to notify reducers
- Can act on more than one piece of the store at a time
- Name your actions clearly
- Can (not always) have a payload -> some data passed into the action

```
export const increment = amount => {
    return {
        type: 'INCREMENT_COUNTER',
        payload: amount
    };
};
```

Example

Initialized to 0

Reducers

- "Listen" for Actions and then carry out the task
- Actually modify the data in the store
- Do different things to the data based on which action was called

```
const buttonCount = (count = 0, action) => {
    switch(action.type) {
        case 'INCREMENT_COUNTER':
            return count + action.payload;
        default:
            return count;
    }
};
```

Example

useSelector

- In your Component, **`useSelector`** allows you to access the data in the Redux store

```
Import it:
    import { useSelector } from 'react-redux';

Select the piece of data you want:
    const count = useSelector(state => state.buttonCount);

Use it in the React Node:
    <h1>I want {count} scoops of ice cream!</h1>
```

Example

useDispatch

- In your Component, call/dispatch your actions!

```
Import it:
     import { useDispatch } from 'react-redux';

Initialize it:
     const dispatch = useDispatch();

Call it from a function:
     <button onClick={() => dispatch(increment(1))}>More!</button>
```

Example (necessary boilerplate)

Don’t forget to:

- Update the index.js file (follow a tutorial, or use my code)
- Use combineReducers if you have many reducers (likely with a more complex app)

Example - Live Coding

Let’s add a button to decrement the counter!

- Will need a new action - DECREMENT_COUNTER
- Make sure the reducer handles this action
- Create the button element, and call the action!

You can see the result in my repo on the branch `decrement-counter`

Conclusion

- Instead of managing state in a lot of separate components, put all your state in a Redux store.
- In order to maintain uni-directional data flow, you can't modify the store directly, but only through Actions and Reducers

Redux probably won't make sense until you start using it!

Where to go next - React

- Lists and Keys
  - https://reactjs.org/docs/lists-and-keys.html
- Controlled Components (good for form, input elements)
  - https://reactjs.org/docs/forms.html#controlled-components
- AMAZING resource to review what we covered in the slides + more:
  - https://www.w3schools.com/react/default.asp

Where to go next - Redux

- More of useSelector and useDispatch with Redux
  - https://medium.com/@mendes.develop/introduction-on-react-redux-using-hooks-useselector-usedispatch-ef843f1c2561
- A similar example to the one we used in class:
  - https://levelup.gitconnected.com/react-redux-hooks-useselector-and-usedispatch-f7d8c7f75cdd
- Another example
  - https://scriptverse.academy/tutorials/reactjs-useselector-usedispatch.html

Setting up your Assignment and Project

- **Follow steps under “Create React App”**
  - https://reactjs.org/docs/create-a-new-react-app.html#create-react-app
- **Install Redux**
  - `npm install redux`
  - `npm install react-redux`

LUNCH

My path to

D2L

(and some advice)

What are we talking about?

- Each alumni speaker will talk about their career path to some extent
- Some will resonate with you more than others
- Hopefully gives you some perspective from ppl who were in your position 2+ years ago!
- Feel free to ask us more questions after! **#career** channel

Tiny bit of background

- I did my BSc in Physics
  - I really like building things (circuits, websites)
  - I'm not interested in research :(
- BCS was the perfect fit!
  - I took a REALLY long time to complete it
  - Did co-op at Semios and D2L

Choosing a big vs small company

Small company (start-up)

- Worked on own project
    - Learned a lot, sense of ownership and pride
- Lonely! Wanted to learn programming skills from other devs

Larger company (D2L)

- Great team -> Lots to learn from, different skill sets
- Slightly less rewarding projects, need to find your place

D2L

- ~1000 employees
- Exposure to a lot of new build processes + had to learn git better!
- Rewarding being involved in the education sector
    - The project you're working on DOES matter
- D2L gives us lots of opportunities to work on neat projects (like this), and they really invest in their employees!! ++
    - Difficult to know if a company will be good at this before you join

Interviews

I’ve now interviewed candidates for both co-op and full-time positions at D2L

- In an interview, they are trying to get to know you. You have to help them out!
- You have to bring your strengths to the forefront.
- BE ABLE to answer "easy" questions like "what's the coolest thing you've ever worked on" or "what's a neat problem that you've tackled?"
  - Again, we WANT you to impress us!
- Turn “weakness” questions into learning/growth examples
- DON’T talk badly about past co-workers/bosses.

Do stuff and document it

- Write stuff on your resume/LinkedIn right away.
  - Even write some additional notes for yourself.
- Have a personal website and/or linkedIn
  - Need to kind of market yourself (more and more)
- PROVE your universal skills (durable skills, soft skills).
  - Talk about how you actually displayed leadership/organizational/teamwork.

Advance yourself!

- Know when to ask for help, but also know what kind of help to ask for!
  - Help yourself before you ask for help from others

- Go find opportunities for yourself. Talk to people, do things that scare you.
  - People will remember you when they are looking for someone to do something.
  - I used to be a mentor at LHL on the side! (Want to connect + find more opportunities)

Team advice

- In a team, start with the assumption that everyone WANTS to contribute, and wants to be a good team member.
- Take advantage of lab time to plan what you will do when you're apart.
- Get used to working remotely
  - Need good communication in order to work together while not in the same place.
- If you're finding it hard to work with your group, talk to a TA sooner rather than later. Try pair programming or mob programming to help other group member(s) get up to speed. You may be at different levels, and that's ok.

Your career is a journey

The hardest things will be the best learning experiences.

(And that is the point of this course…)

Ask Us!

#career channel on Slack

I enjoy answering career-related questions :)

UNIT 2 ASSIGNMENTS

Post-Workshop Survey

- Surveys for Unit 1 + 2
- For participation marks
- Must be done in a few days
- We take them seriously and implement changes quickly!
- It’s short!
- Thanks!

Scrum Updates

1. What did you work on this past iteration (2 weeks)?
2. What were any major issues/challenges you ran into?
3. What do you plan to work on for this coming iteration (2 weeks)?

Total: ~5 sentences max!

See Grading Rubric here:
https://blogs.ubc.ca/cpsc436i2020s/assessment-rubrics/#sec-2

Scrum Updates

- Due tomorrow (Sunday) 10pm
- Should take <10mins to write
- One group member should make an **issue** in your project repo, and then each team member should make a **comment** on that issue.
- INCLUDE YOUR **CSID** IN YOUR POST

Scrum Updates

- We are having this due AFTER this workshop so that you are better able to understand the technology you’re going to be using
- The real effort comes from talking to your group and planning your next steps!
- Keep it short, succinct, and summarize the key points

Assignment 2

- Use React & Redux
- BUILD ALL YOUR OWN COMPONENTS
  - Do not use pre-built components from 3rd party libraries
- Be ready to talk about how your assignment works, and answer questions about React & Redux

Project Progress 2

- Begin setting up your App!
- All PP requirements are VERY minimal
    - If you want an amazing project, you will need to do more
    - Present your code next Workshop in Code Review

Design/Code Reviews

- 2nd half of every Workshop starting today!
- Casual Presentation of your Project so far
- Grading
  - Points for the Project requirements
  - Points for asking at least 1 question of another group
- Purpose of Design/Code reviews is to get feedback & ideas from your classmates, and to get comfortable talking about code!

THE END!