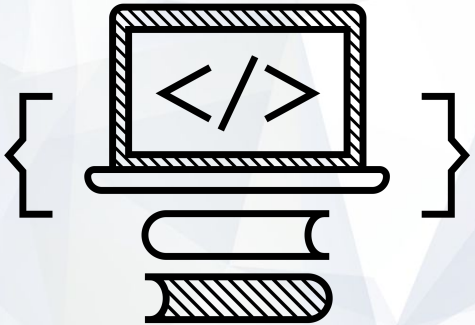
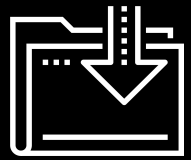


Forms, Conditional Rendering, and React Router

Skills Bootcamp in Front-End Web Development

Lesson 13.3



WELCOME

Learning Objectives

By the end of class, you will:

- Deepen your understanding of managing state with React components.
- Understand conditionally rendering React components.
- Understand the axios library and the concept of the component lifecycle.
- Understand routing with React Router.

Instructor Demonstration

Forms

In this example, we will demonstrate how to handle simple forms with React.

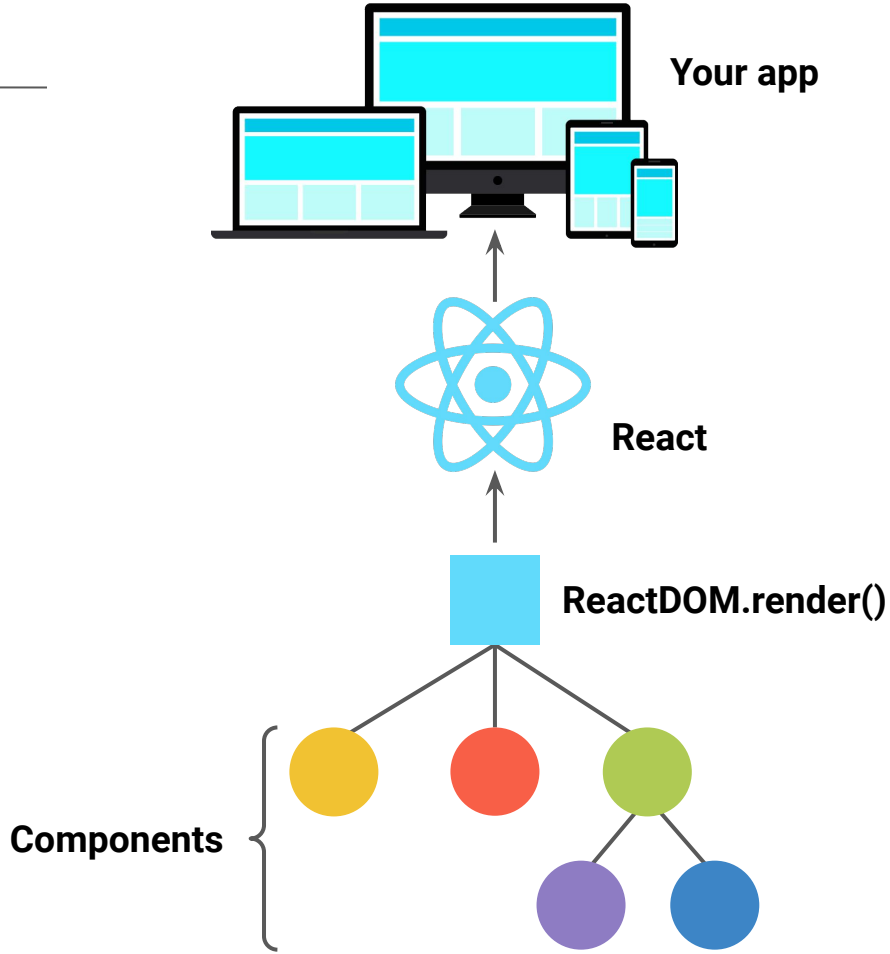
A component's **"state"** is a property defined on a class component instance used for storing values that we want to associate with it.

A Component's "State"

This property is recognized by React and can be used to embed data inside of a component's UI, which we want to update over time.

Whenever a component's state is updated, its render method is fired along with the render methods of all of its children.

This updates the application's UI to display the new data without having to refresh the browser.



Activity: Fun with Forms

In this activity, you will add some new functionality to the previous form example.

Suggested Time:

Time’s Up! Let’s Review.

Since we definitely only have one input field under the control of this `handleInputChange` method, could we decrease the amount of code being used inside of this method?

Review: AJAX

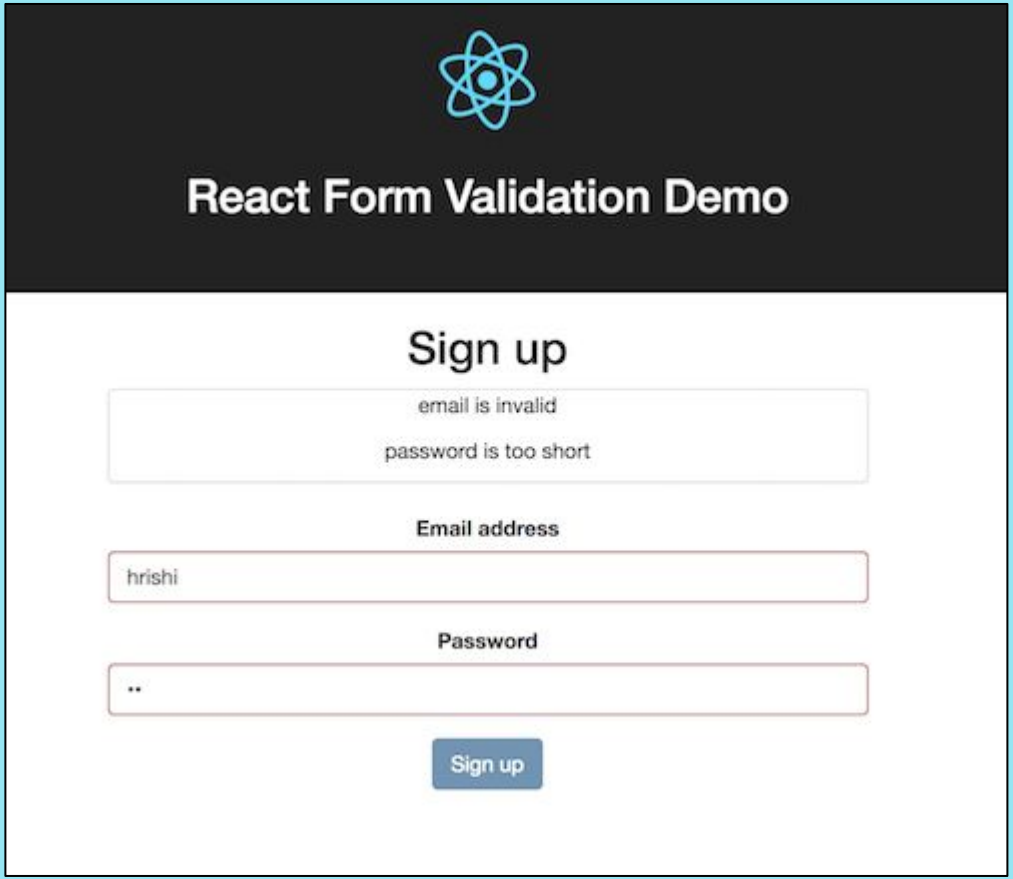
Yes, the current setup accounts for the possibility of adding new input fields.

But if we were positive we'd only have one input field, we could use the following code instead:

```
handleInputChange = event => {
  this.setState({
    search: event.target.value,
  });
};
```

Think back to forms you've created in the past.

Is there anything else we should do with our data after setting the application state?



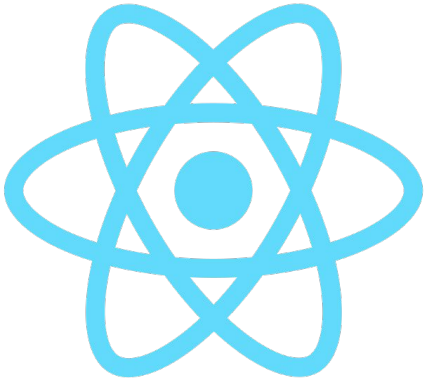
We could add a POST request to `handleFormSubmit` so that the user's input can be sent to a server and saved in a database.

Instructor Demonstration

AJAX Demo

In this example, we will demonstrate AJAX requests with React.

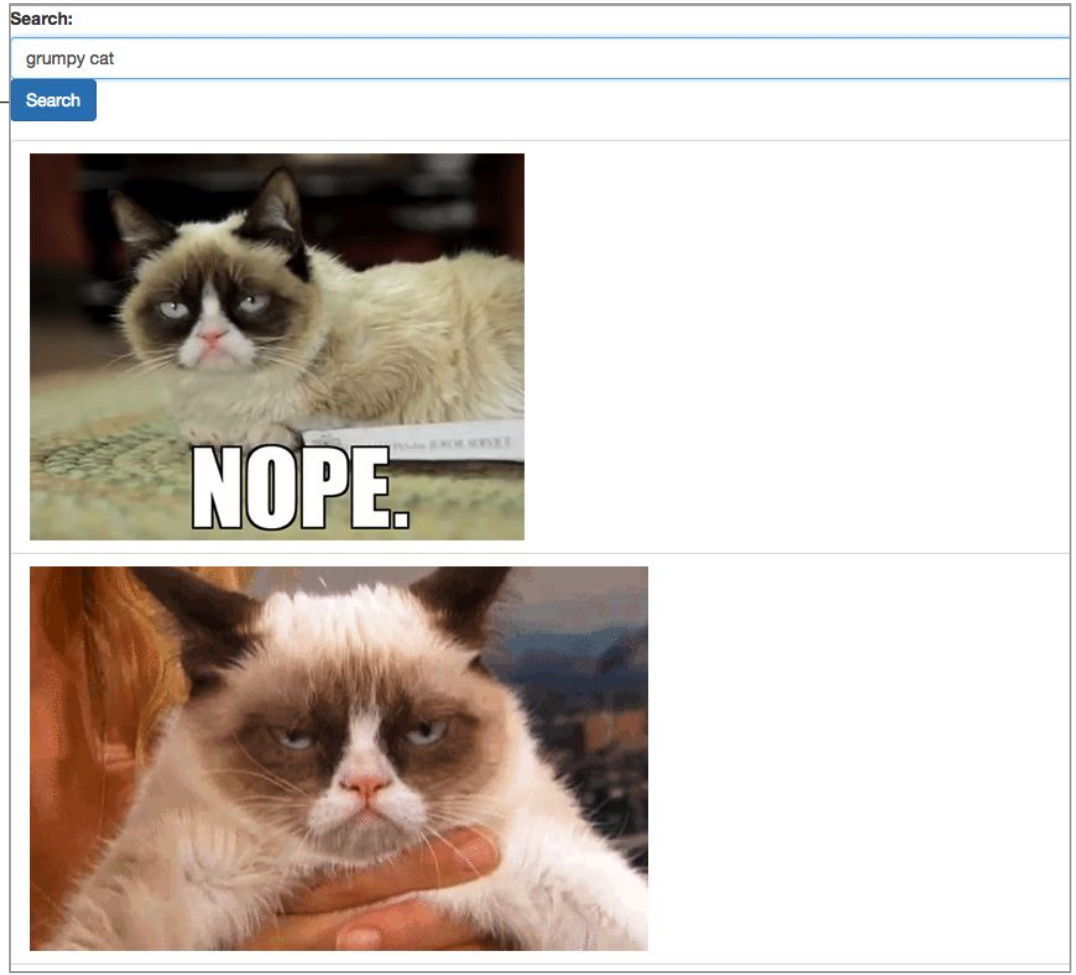
+



AJAX Demo

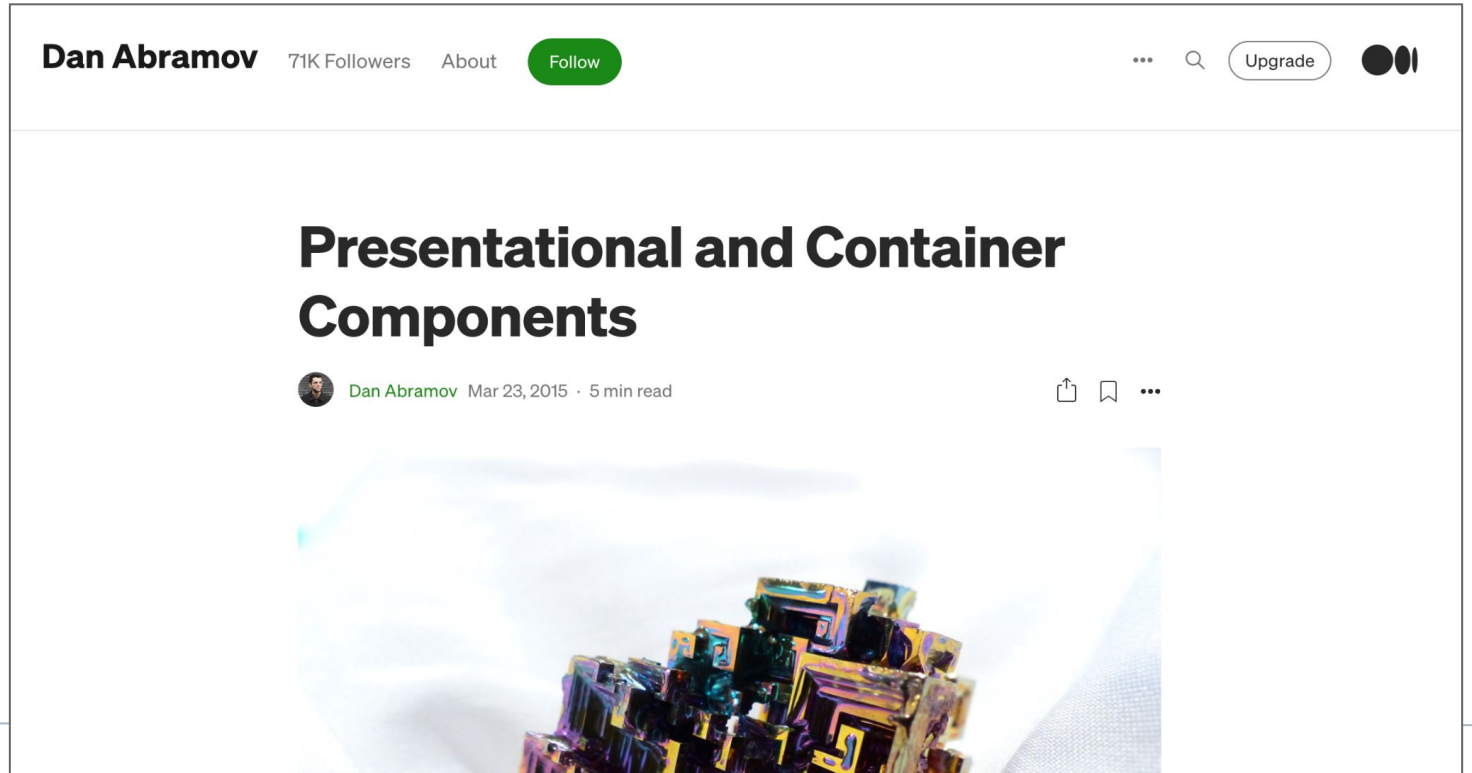
This app searches the Giphy API for whatever is typed into the input field and then displays the results below.

When we first load the app, we should see kitten-related results.



AJAX Demo

On your own time, read this [article written by Dan Abramov]() (Redux Author, React Core Contributor, Create React App Core Contributor).



AJAX Demo

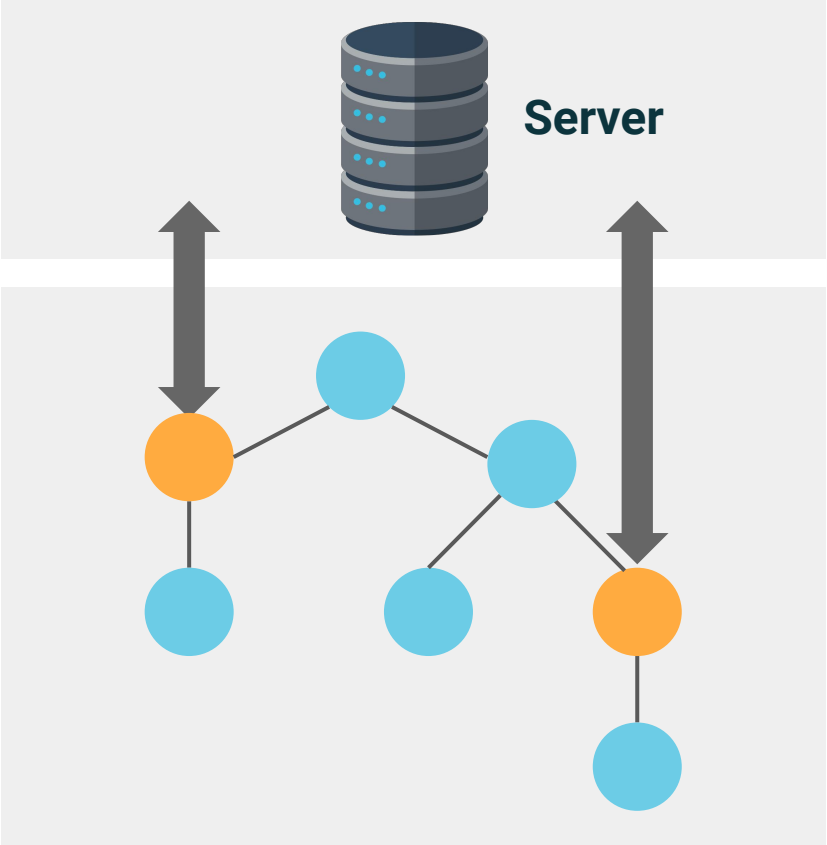
Dan Abramov describes the pattern of separating components into “container” and “presentational” components.

Container components

Are primarily concerned with how things work and render very little, if any, of their own markup. Instead, they mostly render other components and pass down the logic and data they need to work.

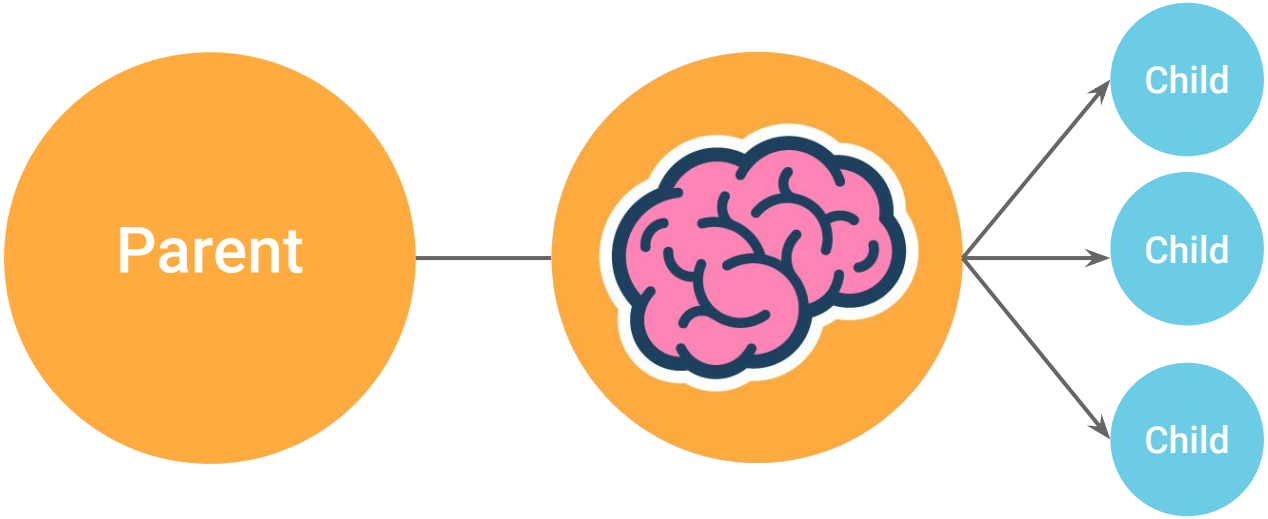
Presentational components

Are concerned with how things look and typically don’t contain any logic that doesn’t have to do with their own individual UI.



Takeaway

There should be a few of these “container” components that act as the “brain” for their children. In our case, this is `SearchResultContainer`.



Component Lifecycle Events

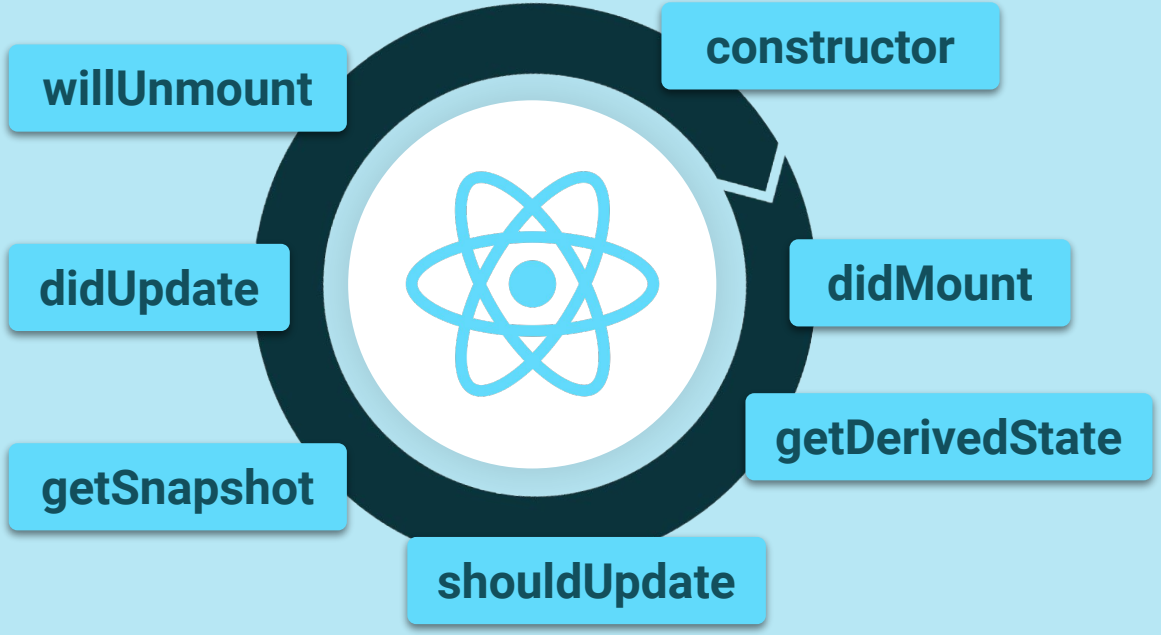
When working with class components in React, we are able to hook into a few different component lifecycle events that allow us to automatically execute logic at certain times.

These component lifecycle events are similar to some of the DOM lifecycle events that we've worked with in vanilla JavaScript or jQuery.

In React, `componentDidMount` runs after a component and all of its children have been initially rendered and mounted to the DOM and have no further expected initialization steps.

Component Lifecycle Events

There are a few different React component lifecycle events that we can hook into, but `componentDidMount` is probably the one we'll use most frequently, as this is the best place to run any async code that we need to populate our component's state after the component mounts.



Why wouldn’t we just run our `searchGiphy` method inside of our component’s constructor and populate our initial state there?

Component Lifecycle Events

Assuming that we went back to explicitly using the constructor method:

- Constructor functions run synchronously, and AJAX requests are asynchronous. So we might not have the async data available by the time the component is instantiated.
- Additionally, running our async code inside of the constructor could cause extra re-renders of our component—making our application feel slow or glitchy on startup—or introduce bugs that are difficult to track down.
- By the time `componentDidMount` is run, there's no more work for our component needs to do. Even if the AJAX request fails or takes a long time to complete, we'd still have our component and its children rendered to some degree.
- `componentDidMount` is called automatically once per component instance.
- Technically, render is another component lifecycle event, but rather than only running once, the render method is called every time our component's state is updated or anytime our component receives new props.

Activity: AJAX

In this activity, you will create a simple React application with which users can query the OMDb API and display information about the movie that is searched for.

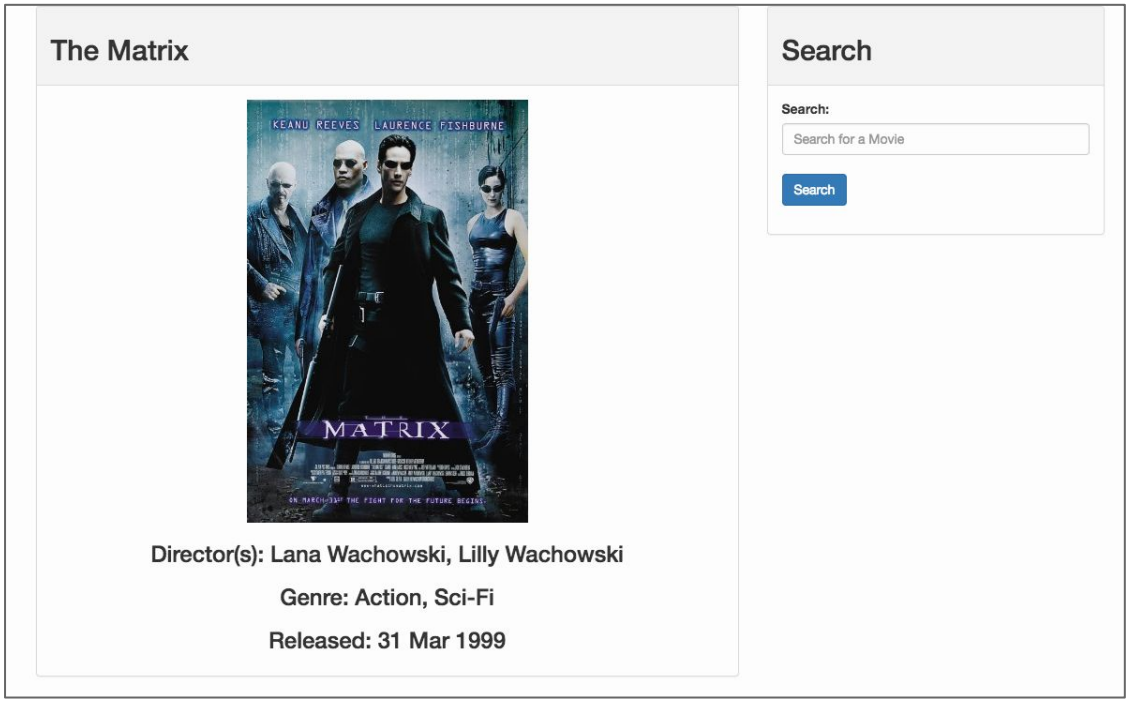
Suggested Time:

Time’s Up! Let’s Review.

Review: AJAX

When we search for a movie using the form on the right side, some information about the movie is displayed in the left card.

When the component first "mounts," some information about the movie *The Matrix* is displayed.



In what part of our application would we be performing this initial AJAX request to the OMDb API?

Review: AJAX

Inside of the componentDidMount lifecycle method of `OmdbContainer`.

This method is where we want to perform any initial async logic for our components.

```
16  componentDidMount() {
17    this.searchMovies("The Matrix");
18  }
19
20  searchMovies = query => {
21    API.search(query)
22      .then(res => this.setState({ result: res.data }))
23      .catch(err => console.log(err));
24  };
```

Since we definitely only have one input field under the control of this `handleInputChange` method, could we decrease the amount of code being used inside of this method?

Activity: Conditional Render

In this activity, you will render one of four different components based on a component's state.

Suggested Time:

Time’s Up! Let’s Review.

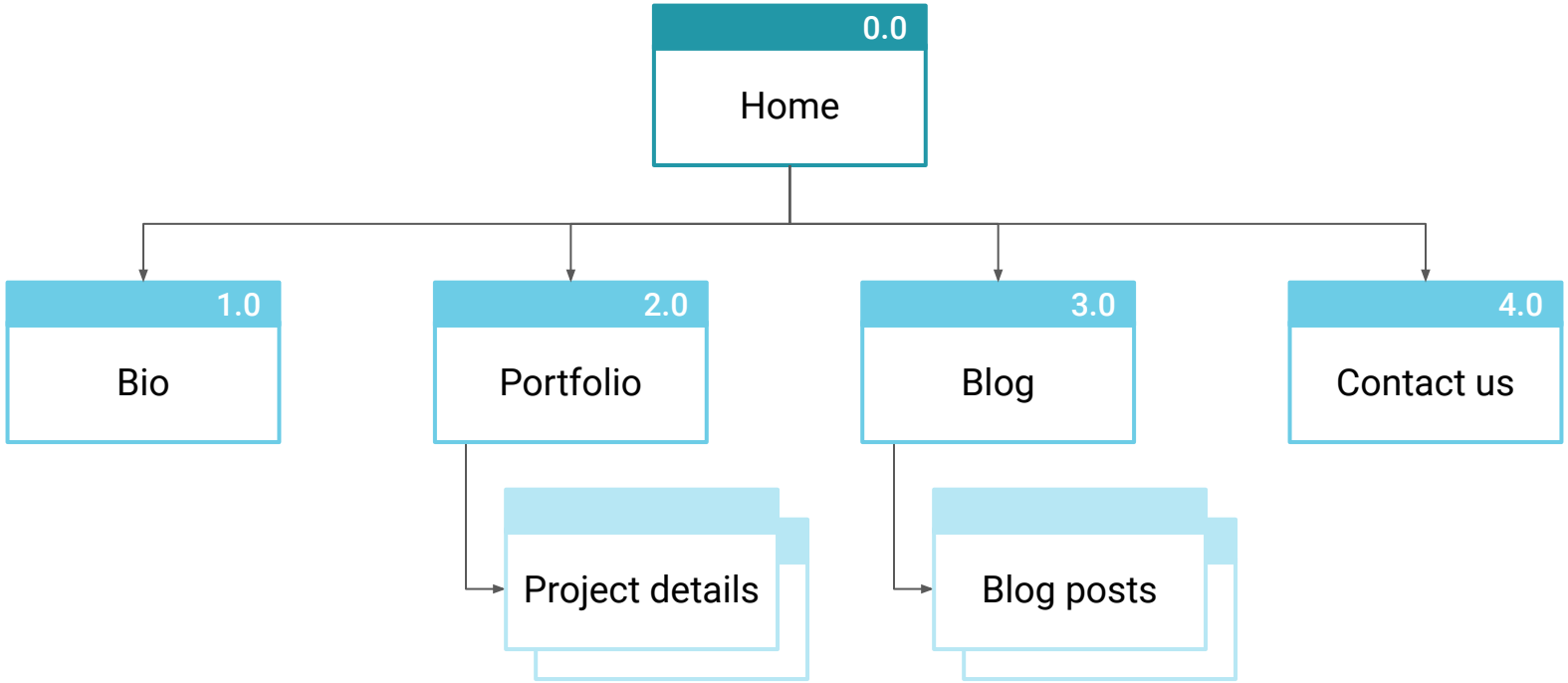
Review: Conditional Render

When we click the different navigation items, a different component is rendered. The styles of the links in the address bar change when we do this, and we are also rendering different content depending on our application state.

React Router

React Router

So far, we've been working with React applications with only one page of content, but in the real world, web applications have multiple—often complex—pages and routes.



What if we deployed the previous activity's portfolio website and we wanted to share a URL with someone that they could use to visit the `About` "page"?

React Router

Currently, we don't have a way to do that. The user would still have to navigate to the `About` "page" on their own from scratch every time since the URL in our address bar doesn't actually change as we click through the tabs.

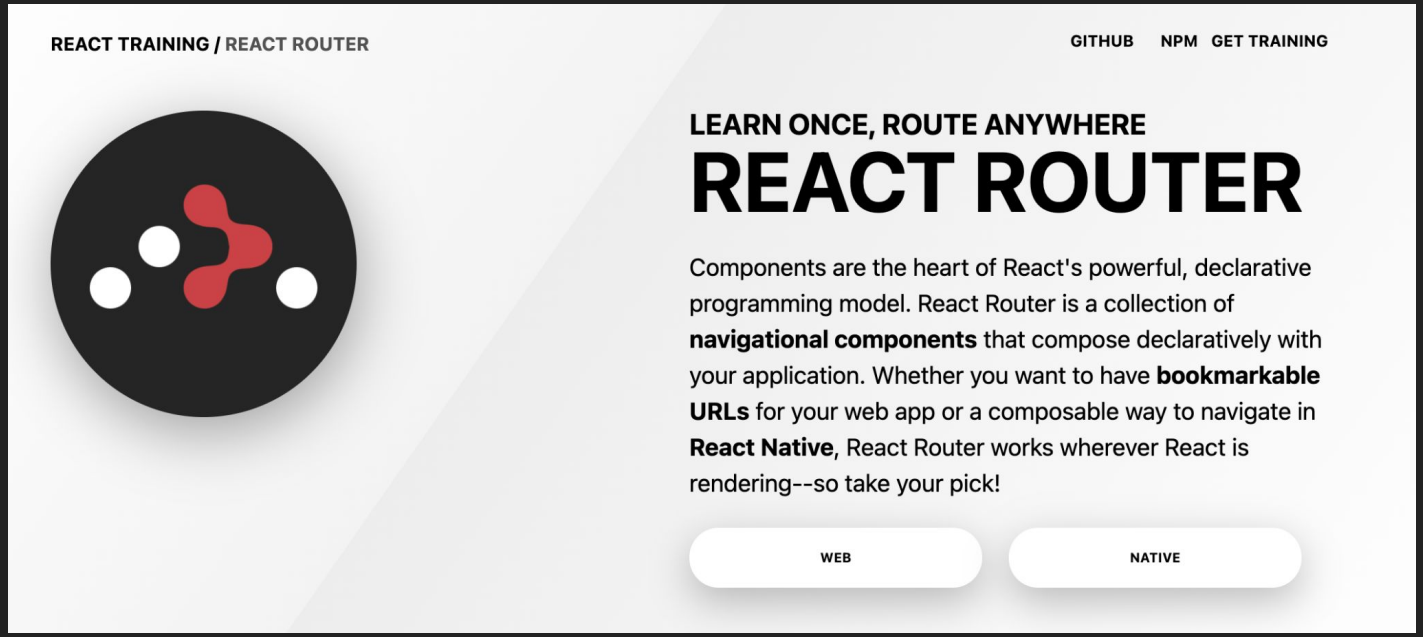
This may seem trivial now, but:

- What if our application were as large as amazon.com?
- What if we wanted to share the URL to a page containing one of millions of different products with someone?
- How would we get users where we intend for them to go?

React Router

Thankfully, we don’t have to code out our own solution to this problem.

One of the most popular React companion libraries out today is React Router.



React Router is a **library** made up of special components for conditionally rendering other components based on the current URL path.

React Router

React Router has modules for routing React applications:

On the web

In native applications

In our case, we're going to be working with React Router on the web.

React Router

While a little intimidating at first, the [React Router documentation]() is some of the best for any library we've covered so far, full of concise and helpful examples.

REACT TRAINING / REACT ROUTER

CORE | WEB | NATIVE

ANNOUNCEMENTS
- The Future of React Router

EXAMPLES
- Basic
- URL Parameters
- Nesting
- Redirects (Auth)
- Custom Link
- Preventing Transitions
- No Match (404)
- Recursive Paths
- Sidebar
- Animated Transitions
- Route Config
- Modal Gallery
- StaticRouter Context
- Query Parameters

Philosophy

This guide's purpose is to explain the mental model to have when using React Router. We call it "Dynamic Routing", which is quite different from the "Static Routing" you're probably more familiar with.

Static Routing

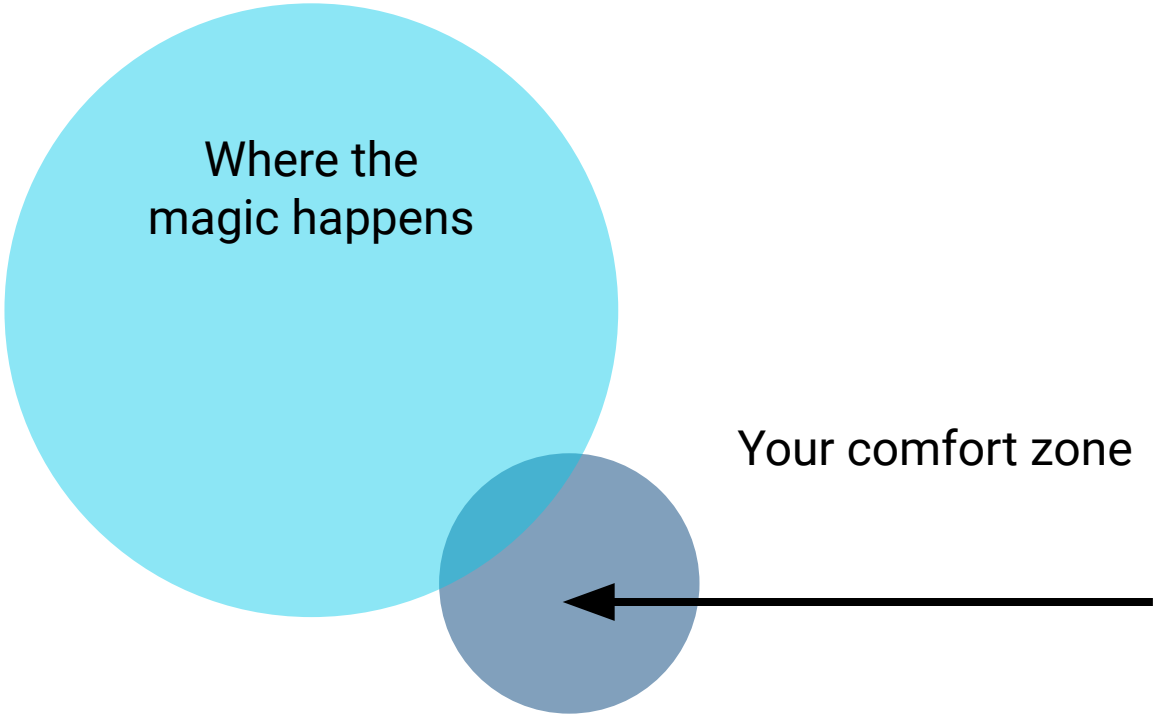
If you've used Rails, Express, Ember, Angular etc. you've used static routing. In these frameworks, you declare your routes as part of your app's initialization before any rendering takes place. React Router pre-v4 was also static (mostly). Let's take a look at how to configure routes in express:

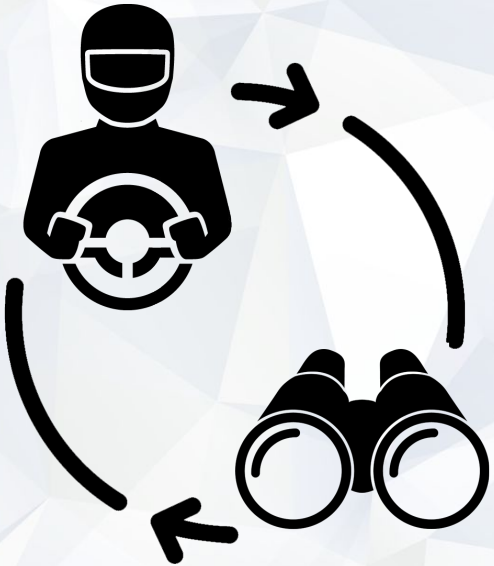
```
// Express Style routing:
app.get("/", handleIndex);
app.get("/invoices", handleInvoices);
app.get("/invoices/:id", handleInvoice);
app.get("/invoices/:id/edit", handleInvoiceEdit);

app.listen();
```

Note how the routes are declared before the app listens. The client side routers we've used are similar. In Angular you declare your routes up front and then import them to the top-level `AppModule` before rendering:

React Router

We'll be going over the fundamental 20% or so of syntax that you're likely going to be using 80% of the time.





Pair Programming Activity:

Pupster App

In this activity, you will work with partners to create a full React application from scratch, complete with routing and AJAX requests to the Dog Ceo API.

Suggested Time:

Time’s Up! Let’s Review.

The End