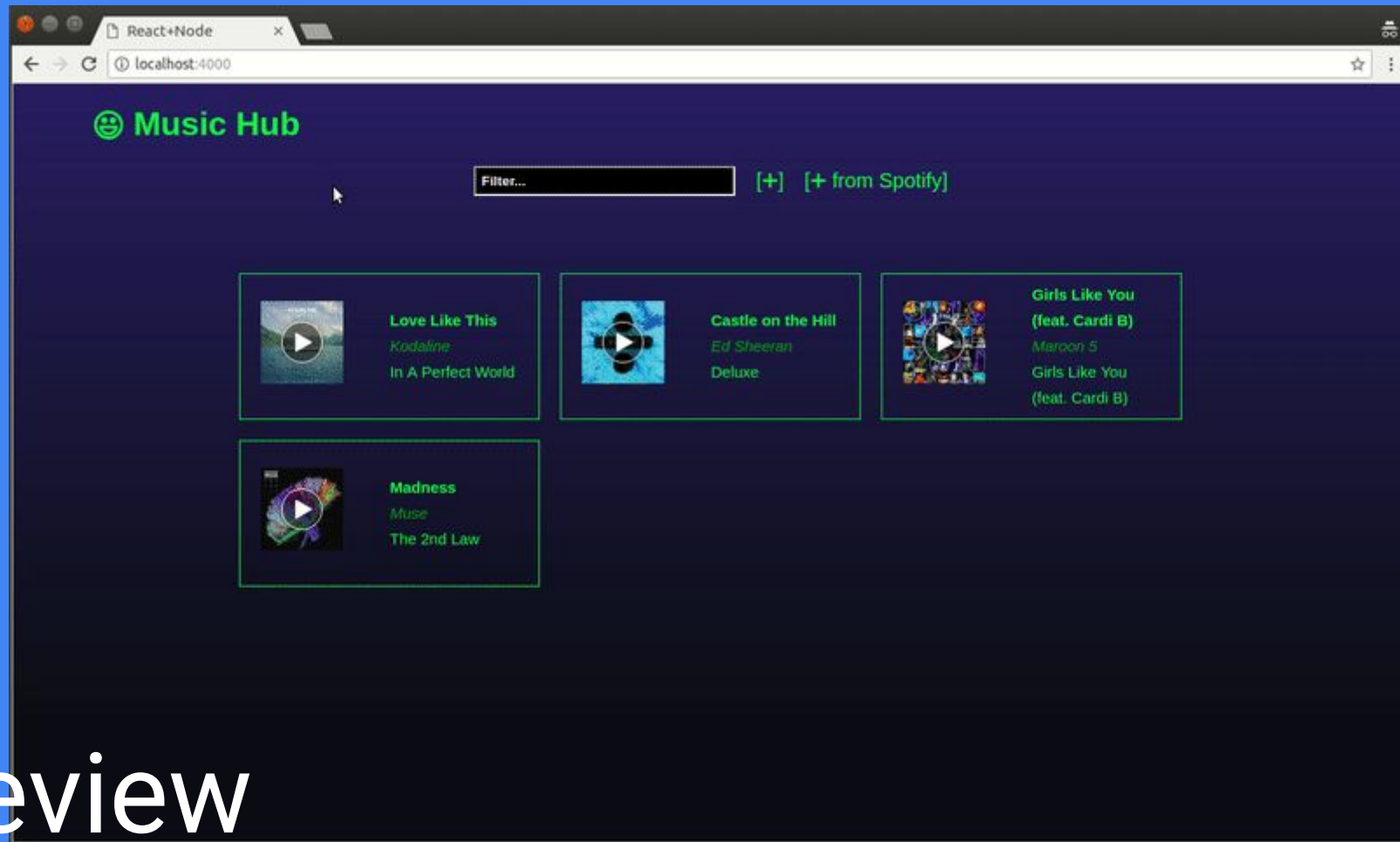




React workshop

Learn front-end web development with React





Preview

We need:

For Windows users:

Check out this doc if having problems with Node,
npm installation

<https://goo.gl/WN3LmA>

1. Internet! 🌐
2. Join the **Slack**: <https://goo.gl/wz8GG2>
3. Install Git and Sign up to Github (if you don't have one yet)
 - You'll need this to fork the Github Repo
4. Install Chrome/Firefox
5. Code editor & terminal - pick your favorite
 - I use **VSCode**, which has multi terminals built-in
6. Install node - install **latest LTS version**
 - Download and install from <https://nodejs.org/en/download/>
 - To test: <https://nodejs.org/en/docs/guides/getting-started-guide/>
7. Open this page in new tab: <https://codesandbox.io/u/lenworld/sandboxes>
8. Check out the cheat sheet <https://goo.gl/m8i2gc> for some quick info



Git setup Also in Slack #resources

1. Log-in to github
2. Go to https://github.com/lenworld/react_workshop
3. In github, fork branch  Fork
 - You'll be asked to login to your github account

4. On local terminal, clone the repo

```
$ git clone https://github.com/<username>/react_workshop.git
```

or use SSH if you'd like

5. Checkout dev branch (which should be an empty slate)

```
$ cd react_workshop
```

```
$ git checkout dev
```

6. Open code editor with **react_workshop** as the root directory

7. **We're ready!** 

Git: exploring workshop code

Also in Slack [#resources](#)

Comparing branches

1. Each chapter/step is numbered, corresponding to a github branch
2. See diff at any step, e.g. to see code in step c1.6

https://github.com/<user_name>/react_workshop/compare/c1.5...c1.6

Or use Github compare tool:



Checkout code at any step

```
$ git stash // (or git clean -fd) (or commit + push in a new branch)
$ git checkout <chapter> // e.g. git checkout c4.1
$ npm install // install deps to make code work
// if needed, restart terminal processes for npm run dev
```

Introduction



Let's make this better!

After the workshop/before you leave

Please answer a short survey about the workshop

<https://goo.gl/M6Bdc3>

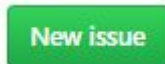
Please go to the code repo, and give it a star

(if it deserves one, of course!)

https://github.com/lenworld/react_workshop



Any mistakes or improvements, please submit an issue
or even better, contribute to open-source! (ping me in Slack for details)



React and webpack setup

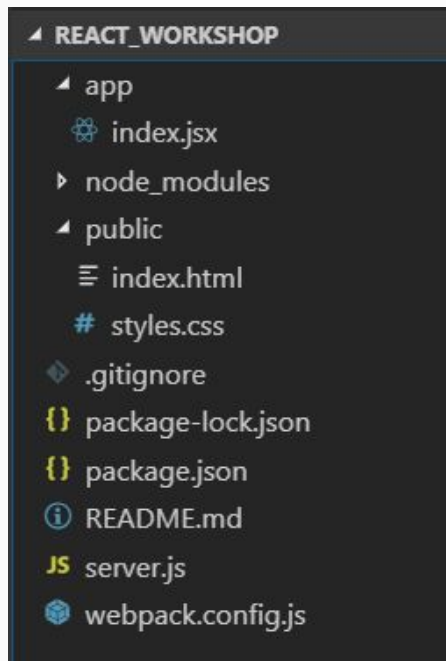
Our React boilerplate

- Checkout **/dev** branch

```
$ git checkout dev  
$ npm install  
$ npm run dev
```

Our React boilerplate

- These files and folders comprise a working React boilerplate running on a Node + webpack server
- Quick walkthrough:
 - webpack.config.js
 - public/
 - index.html
 - package.json
 - **app/index.jsx**
 - **app/data.js**



Our first React component

- What is a **component**?
 - reusable, self-contained code that contains rendering logic, JSX
- What is **JSX**?
 - HTML-like syntax representing UI elements to be rendered in a page
- How do you make React components appear in HTML?
 - **render** the root component to an HTML node

```
// app/index.jsx

import React from 'react';
import ReactDOM from 'react-dom';

class App extends React.Component {
  render() {
    return (
      <div>React: Hello World!</div>
    );
  }
};

ReactDOM.render(<App />, document.getElementById('app'));
```

1 React Basics

Codesandbox: Component Hierarchy

Codesandbox:

<https://codesandbox.io/u/lenworld/sandboxes/>

sandbox name:

2A_components

or use direct link: <https://codesandbox.io/s/p791r850r0>



c1.1 UIManager.jsx

- Create a new file **app/UIManager.jsx**
- Import and render **UIManager** component inside **App** component

```
// app/index.jsx

import React from 'react';
import ReactDOM from 'react-dom';
import UIManager from './UIManager';

...
render() {
  return (
    <UIManager />
  );
}
...
```

```
// app/UIManager.jsx

import React from 'react';
import data from './data';

console.log(data);

class UIManager extends React.Component {
  render() {
    return(
      <div>List goes here...</div>
    );
  }
}

export default UIManager;
```

c1.2 Header.jsx

- Create a new file **app/Header.jsx**
 - Copy from Slack #react_snippets:
Header.jsx
- Import and render **Header** component inside **UIManager** component
 - *** Notice that we have to enclose return JSX in a <div> ***

```
// app/UIManager.jsx
import Header from './Header';

...

return(
  <div>
    <Header />
    <div>List goes here...</div>
  </div>
);

...
```



Different ways to style your components:

- Regular CSS stylesheets (we're using this), inline (used only in Header.jsx), etc

c1.3 List.jsx and Item.jsx

- Create new files
 - **app/**
 - **List.jsx**
 - **Item.jsx**
- Import and render **List** in **UIManager**
- Import and render **Item** in **List**

Component tree

```
App
├── UIManager
│   ├── Header
│   ├── List
│   │   └── Item
│   └── ...
```

```
// app/Item.jsx
import React from "react";
class Item extends React.Component {
  render() {
    return <div>...item...</div>;
  }
}
export default Item;
```

```
// app/List.jsx
import React from "react";
import Item from "../Item";

class List extends React.Component {
  render() {
    return (
      <div><Item /></div>
    );
  }
}
export default List;
```

```
// app/UIManager.jsx
import List from '../List';

...

<Header />
<List />
```



chrome web store



lenmorld@gma



React Developer Tools

Offered by: Facebook



1,093

[Developer Tools](#)



1,446,738 users

Add to Chrome

Props

What?

Props is a data object passed from a parent component to a child component. It uses attribute syntax like in HTML

How?

Inside the JSX of a parent component, declare the child component, passing the data in this format:

```
<Child propname=propvalue>
```

e.g. `<Person name="Lenny">`

as an example



Music Hub

'Hub' your music

Hello : Lenny



Search (text or /regex/)



Props

name: "Lenny"

text: "Hello"

List.jsx

app ▸ List.jsx ▸ List ▸ render

```
1 import React from "react";
2 import Item from "../Item";
3
4 class List extends React.Component {
5   render() {
6     var greeting = "Hello";
7     return (
8       <div>
9         <Item name="Lenny"
10           text={greeting} />
11       </div>
12     );
13   }
14 }
15
16 export default List;
```

Item.jsx

app ▸ Item.jsx ▸ Item ▸ render

```
2
3 class Item extends React.Component {
4   render() {
5     return <div>{this.props.text} : {this.props.name}</div>;
6   }
7 }
8
9 export default Item;
10
```

Codesandbox: Props

Codesandbox:

<https://codesandbox.io/u/lenworld/sandboxes/>

sandbox name:

2B_react_props

or use direct link: <https://codesandbox.io/s/7joom0lp6x>



State

aka Data

state change → re-render UI

What and why?

- State is where data lives.
- Any change of a piece of data in state results to a re-render.
 - React is smart enough to only render what changed (virtual DOM)
- The “data-driven” behavior is the reason why we use frontend frameworks
- i.e. instead of writing logic to manipulate HTML elements (e.g. in vanilla JS/ jQuery), you write reusable data-driven UI components

Implementing State

How?

Inside a component declaration:

1. add a ***constructor()*** { }
2. Call ***super()***
3. define ***this.state***
 - a. State is initialized when an instance of this component is created

To read data, use ***this.state.<obj>***

```
class App extends React.Component {  
  constructor() {  
    super();  
    this.state = {  
      name: "Concordia"  
    };  
  }  
  
  render() {  
    return (  
      <div>  
        { /* get from state */}  
        <School name={this.state.name} />  
      </div>  
    );  
  }  
}
```

Codesandbox: State

Codesandbox:

<https://codesandbox.io/u/lenmorl/sandboxes/>

sandbox name:

2C_react_state

or use direct link: <https://codesandbox.io/s/0x3y5vj61l>



State and Props gotchas! 🤖

Props

- Props can only be passed downwards: parent → child
 - One hierarchy at a time: cannot pass grandparent to child without passing parent
- Functions can also be passed (discussed later) as **function props**

State

- Only the component has access to its own state, but it can pass **props** downwards for child components to render or modify the data
- A component can also be **stateless**

State and Props analogy

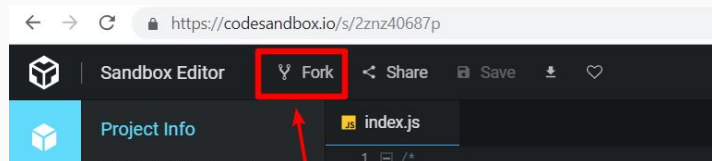
- Boss and Employees



Codesandbox: **4A_state_props_analogy**

<https://codesandbox.io/s/xp7zv30934>

Please fork before editing



Component_tree:

Boss

Employee

Props passing:

Boss → \$100 → Employee

c1.4 Add state to UIManager, pass list object as props to List

```
// app/UIManager.jsx
```

```
...  
class UIManager extends React.Component {
```

```
  constructor() {  
    super();  
    this.state = {  
      listName: "The list",  
      list: data.music  
    }  
  }  
}
```

```
  render() {
```

```
    ...
```

```
    <List name={this.state.listName}  
    list={this.state.list} />
```

```
    ...  
  }
```

```
// app/List.jsx
```

```
...  
class List extends React.Component {
```

```
  render() {
```

```
    return (
```

```
      <div>
```

```
        <h3>{this.props.name}</h3>
```

```
        <div>List: {this.props.list}</div>
```

```
        <hr/>
```

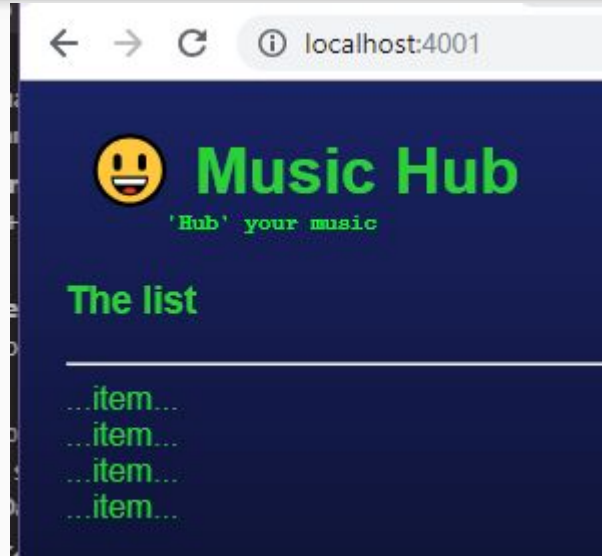
```
        <Item />
```

```
      </div>
```

```
✖ ▶ Uncaught Error: Objects are not valid as a React child (found: object with keys {id, artist, title, album}). If you meant to render a collection of children, use an array instead.  
    in div (created by List)  
    in div (created by List)  
    in List (created by UIManager)  
    in div (created by UIManager)
```

c1.5 Using map() to render list

```
// app/List.jsx
render() {
  var list = this.props.list;
  return (
    <div>
      <h3>{this.props.name}</h3>
      <hr />
      {
        list.map(function (item) {
          return (
            <Item item={item}
              key={item.id} />
          )
        })
      }
    </div>
  );
}
```



📝 NOTE: always include a **key** to help React optimize rendering

Map, filter, reduce

Codesandbox:

<https://codesandbox.io/u/lenworld/sandboxes/>

sandbox name: **js_map_filter_reduce**

or use direct link: <https://codesandbox.io/s/4z684jjzxw>

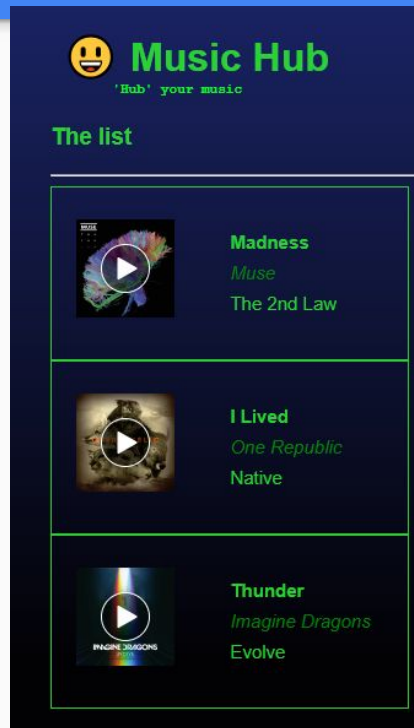


c1.6 rendering Item

Copy from **Slack #react_snippets**: **Item.jsx**

```
// app/Item.jsx
...
class Item extends React.Component {
  render() {
    var item = this.props.item;
    ...

    <div className="right">
      <div className="title">{item.title}</div>
      <div className="artist">{item.artist}</div>
      <div className="album">{item.album}</div>
    </div>
  }
}
```



Presentational components

- **Item** can be called a **dumb / presentation component**
 - in contrast to **UIManager** that has data, **Item** has no logic at all
- Since it's only job is to render the `item` object into JSX / HTML elements
- **Dumb components** can be transformed into a *stateless/functional components*

c1.7 Stateless/functional components

```
// CLASS Component
import React from "react";
class Item extends React.Component {
  render() {
    var item = this.props.item;
    return (
      <div>
        <div className="title">{item.title}</div>
        <div className="artist">{item.artist}</div>
        <div className="album">{item.album}</div>
      </div>
    );
  }
}
export default Item;
```

```
// STATELESS (functional) Component
const Item = (props) => (
  <div>
    <div className="title">{props.item.title}</div>
    <div className="artist">{props.item.artist}</div>
    <div className="album">{props.item.album}</div>
  </div>
);
export default Item;
```

succinct, slight increase in performance

*required when using **state**, `constructor()`, and other lifecycle methods

2 Forms and event handlers

c Hub

Filter...

[+] [+ from Spotify]



Love Like This

Kodakline

In A Perfect World



Castle on the Hill

Ed Sheeran

Deluxe



Girls Like You (feat. Cardi B)

Maroon 5

Girls Like You
(feat. Cardi B)



Madness

Muse

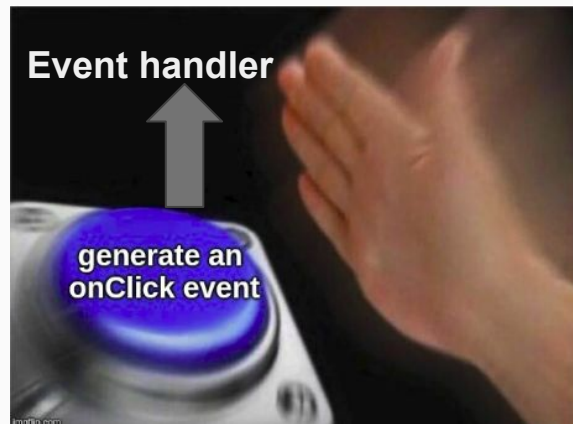
The 2nd Law

Events and event handlers

HTML elements (e.g. inputs, buttons) generate events (onClick, onChange, etc), which is processed by an **event handler** function. This is where we can define what to do when the event happens

Codesandbox: `js_event_handler`

<https://codesandbox.io/s/l2mw8wrj5z>

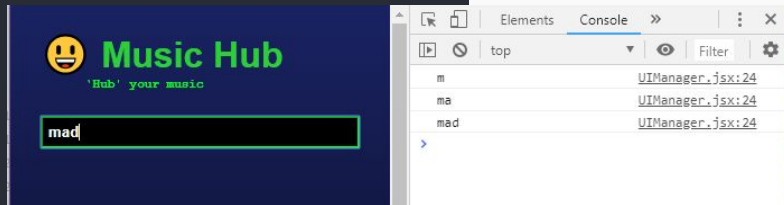


c2.1 event handler

```
// app/UITManager.jsx
...
searchList(event) {
  var search_term = event.target.value;
  console.log(search_term);
}

render() {
  return(
    <div>
      <Header />
      <div className="options">
        <input type="text"
          placeholder="Filter..."
          onChange={this.searchList} />
      </div>
      <List list={this.state.list}/>
    </div>
  );
}
...
```

*event parameter is automatically passed here as the default argument of a HTML element event handler



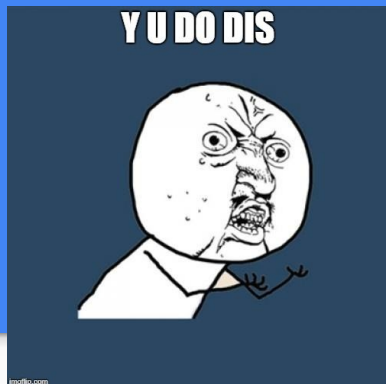
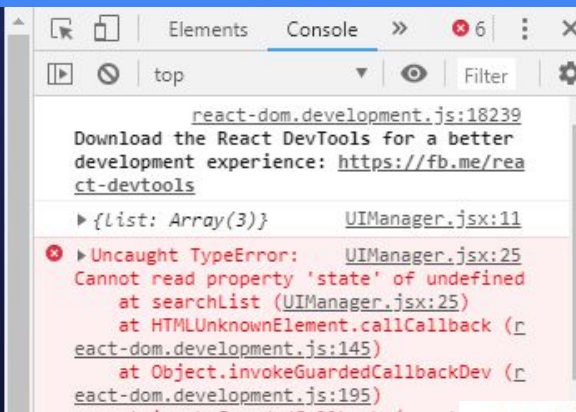
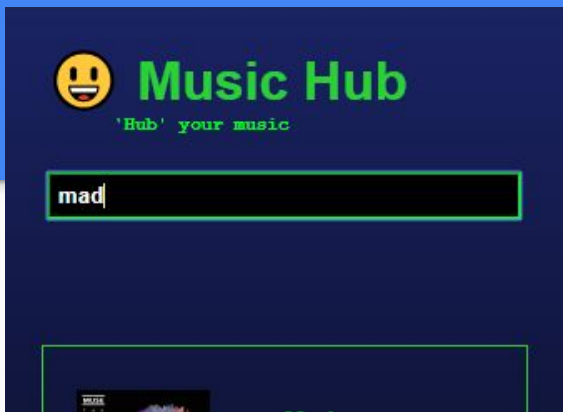
c2.2 setState() to modify state

For us to filter the list based on current input:

- We have to track user's input by putting it in state
 - `this.state.search_term`
- To modify state, we use **setState()**
 - **Never do: `this.state.obj = new_obj`**
 - **Why? Setting state can be async**

```
class UIManager extends React.Component {  
  ...  
  this.state = {  
    listName: "The list",  
    list: data.music,  
    searchTerm: '',  
  }  
}  
  
searchList(event) {  
  var searchTerm = event.target.value;  
  console.log(`${this.state.searchTerm} -> ${searchTerm}`);  
  
  this.setState({  
    searchTerm: searchTerm  
  });  
}
```

But we're getting an error!



- **Cannot read 'state' of undefined! why?**
- **'This' is undefined inside the event handler**
- We don't have access to **this.state**, **this.setState**
- We need to bind the function to the object instance, to make sure **this** is defined inside a nested function (i.e. a function inside a function)

```
searchList(event) {  
  event = SyntheticEvent {dispatchConf  
  var search_term = event.target.value; // console.log(se  
  undefined  
  debugger;  
  console.log("current search term: ", this.state.search_  
  this.setState({
```

c2.3 using an arrow function to bind *this*

```
// app/UISManager.jsx
...
    <div className="options">
      <input type="text"
        placeholder="Filter..."
        onChange={ (event) => this.searchList(event)} />
    </div>
...

```

Alternatives:

- Bind in constructor (better performance)
- Class properties (need ES6 stage 3 features enabled)

Codesandbox: **3A_binding_event_handlers_this**

<https://codesandbox.io/s/3x3z1rm365>

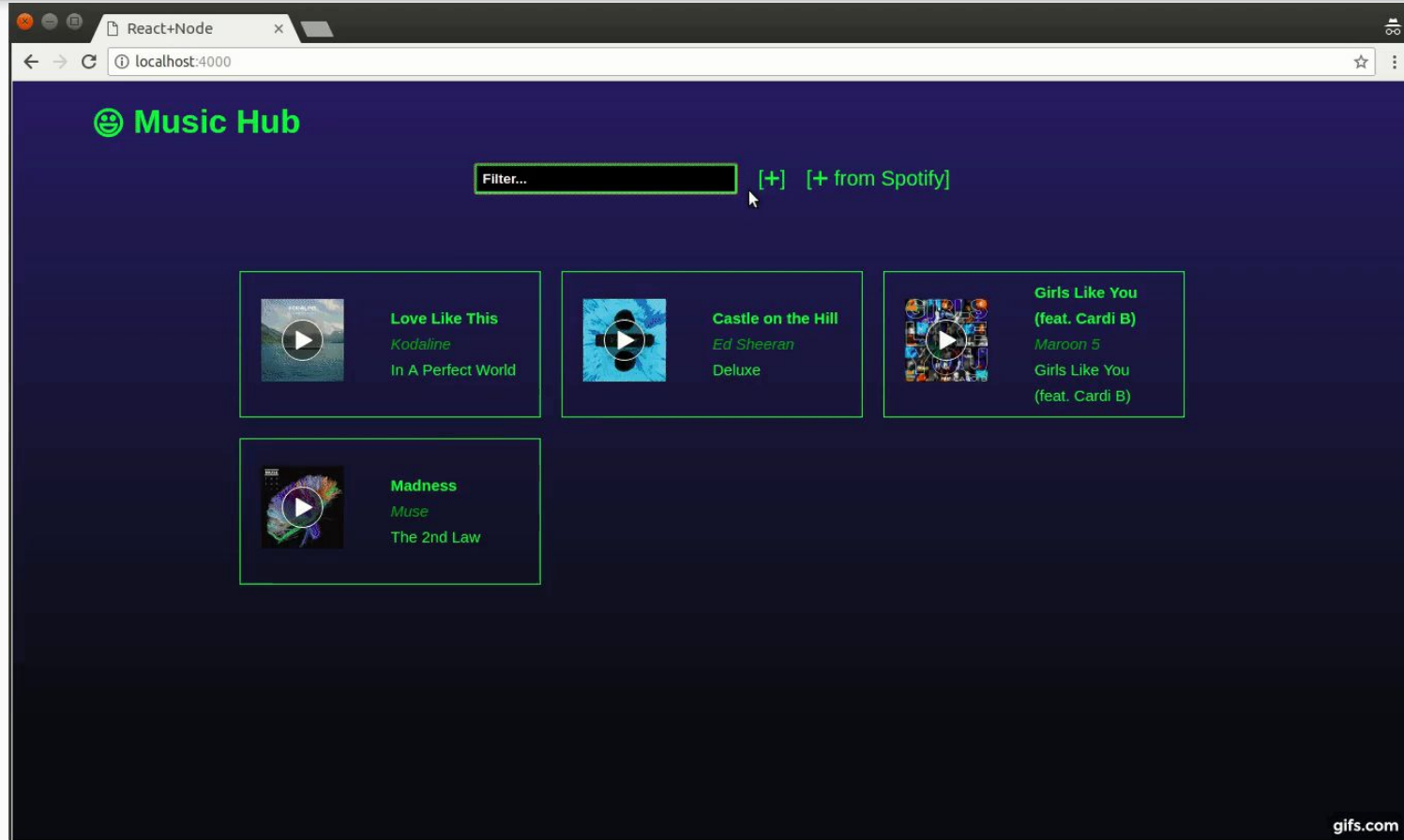
Explore the issue and solutions here



c2.4 filtering list based on *state.search_term*

```
render() {  
  // filter list based on current user input -> searchTerm  
  var list = this.state.list;  
  var searchTerm = this.state.searchTerm;  
  var filteredList;  
  
  if (!searchTerm) {  
    filteredList = list;  
  } else {  
    filteredList = list.filter(function (item) {  
      return item.title.toLowerCase().includes(searchTerm.toLowerCase());  
    });  
  }  
  
  return (  
    ...  
    <List name={this.state.listName} list={filteredList} />  
  )  
}
```

Create / Add new Item



**** from this chapter onwards,
see full code change on Github**

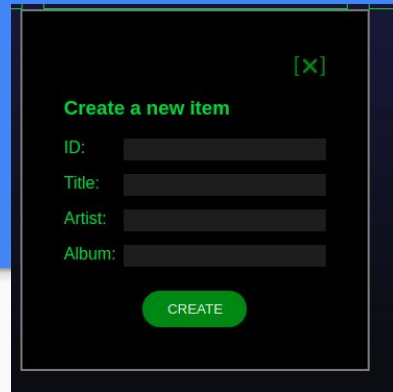
c2.5 ItemForm.jsx

We'll create a component that has an HTML form

1. Create new file **app/ItemForm.jsx**
 - a. Copy from **Slack #react_snippets: ItemForm.jsx**
2. In **UIManager**
 - a. add `this.state.formFields` as an object, initializing item fields to empty string
 - b. Import and render `<ItemForm ... />` same level, and after `<List ... />`
 - c. And pass `this.state.formFields` as props
 - i. `<ItemForm item={this.state.formFields} />`

NOTES: 

- You can have an **object** data type in state



A screenshot of a web application interface. At the top right is a close button labeled '[X]'. Below it is a heading 'Create a new item'. There are four input fields labeled 'ID:', 'Title:', 'Artist:', and 'Album:'. At the bottom is a green button with the text 'CREATE'.

```
searchTerm: '',
formFields: {
  id: '',
  title: '',
  artist: '',
  album: '',
},
```

Function props

- Boss and Employees



Remember that **Employee** (child) component cannot modify **Boss** (parent) state directly.

However, **Boss** can pass a function to **Employee** that it can call whenever it wants to change Boss's state. e.g:

In Boss render():

```
<Employee performWell={ (money) => this.increaseMoney(money) }>
```

In Employee render():

```
<button onClick={() => this.props.performWell(100)}>
```

Codesandbox: 4B_state_props_analogy

<https://codesandbox.io/s/6jormxo65n>



c2.6 Function props

```
<UIManager>  
  <ItemForm>
```

Props passing:

```
<UIManager> ---- onChangeFormInput(event) ----> <ItemForm>
```

UIManager.jsx

UIManager renders ItemForm, passing a **function prop** that allows ItemForm to change data (state) in UIManager

```
class UIManager extends React.Component {  
  ...  
  onChangeFormInput(event) {  
    console.log("input changed");  
  }  
  ...  
  <ItemForm item={this.state.formFields}  
    onChangeFormInput={(event) => this.onChangeFormInput(event)} />
```

c2.6 Function props

```
<UIManager>  
  <ItemForm>
```

Props passing:

```
<UIManager> ---- onChangeFormInput(event) ----> <ItemForm>
```

ItemForm.jsx

ItemForm then calls the function prop given by UIManager, whenever there is a change in input (onChange event).

```
class ItemForm extends React.Component {  
  ...  
  
  <p>  
    <label>ID:</label>  
    <input name="id"  
      onChange={(event) => this.props.onChangeFormInput(event)}  
      value={item.id} />  
  </p>  
}
```

We do this for the 4 inputs (id, item, artist, album)

c2.7 Other event handlers of ItemForm

ItemForm.jsx

1. When hiding the form ([X])
 - a. onClick → this.hideForm
2. When submitting form (CREATE)
 - a. onClick → *this.onSubmitForm(event)*

```
class ItemForm extends React.Component {  
  hideForm() {  
    console.log("hide form");  
  }  
  
  onSubmitForm(event) {  
    event.preventDefault(); // prevent reload of page  
    console.log("form submitted");  
  }  
  ...  
  <div className="close_form">  
    <span onClick={this.hideForm}>[X]</span>  
  </div>  
  ...  
  <div className="create">  
    <button onClick={(event) =>  
this.onSubmitForm(event)}>  
      CREATE  
    </button>  
  </div>  
}
```

c2.8 onChangeFormInput

Once **UIManager** receives the event from **ItemForm** (through function prop)

- The event input can now be processed and reflected in state

```
class UIManager extends React.Component {  
  ...  
  onChangeFormInput(event) {  
    // copy values (not reference) using ES6 spread  
    var currentListFields = { ...this.state.formFields };  
    // e.g. current_list_fields['artist'] = 'Artist1'  
    currentListFields[event.target.name] = event.target.value;  
    // apply new value to state  
    this.setState({  
      formFields: currentListFields  
    });  
  }  
}
```

c2.9 createItem()

```
<UIManager>  
  <ItemForm>
```

Props passing:

```
<UIManager> → createItem(item) → <ItemForm>
```

In UIManager

- define ***createItem()***
 1. Get Item data from state
 2. Copy List values (not reference), using ES6 spread
 3. Add new item to copy
 4. Apply changes to state using *this.setState*
 5. Empty form fields
- Pass as function props to **ItemForm**

```
<ItemForm item={this.state.formFields}  
  onChangeFormInput={(event) => this.onChangeFormInput(event)}  
  createItem={() => this.createItem()} />
```

```
class UIManager extends React.Component {  
  ...  
  // data API - CRUD methods  
  createItem() {  
    // get Item data from state  
    var item = this.state.formFields;  
    // copy list values, not reference, using ES6 spread  
    // operator  
    var currentListItems = [...this.state.list];  
    // add new item  
    currentListItems.push(item);  
    // apply change to state  
    this.setState({  
      list: currentListItems  
    });  
    // empty fields for next round  
    this.setState({  
      formFields: {  
        id: '',  
        title: '',  
        artist: '',  
        album: ''  
      }  
    });  
  }  
}
```

c2.9 createItem()

ItemForm

- Forward request to function props, when form is submitted

```
class ItemForm extends React.Component {  
  ...  
  onSubmitForm(event) {  
    event.preventDefault(); // prevent reload of page  
    // console.log("form submitted");  
    this.props.createItem();  
  }  
}
```

c2.10 show and hide ItemForm

UIManager:

Add [+] button beside search box, add *onClick* and set to *showForm()*

showForm()

Invoke when clicking [+] -> set style to 'block'

ItemForm:

Set CSS class to "modal"

hideForm()

Invoke when clicking [X] on modal -> set style to 'hidden'

Optional: invoke after adding a new item

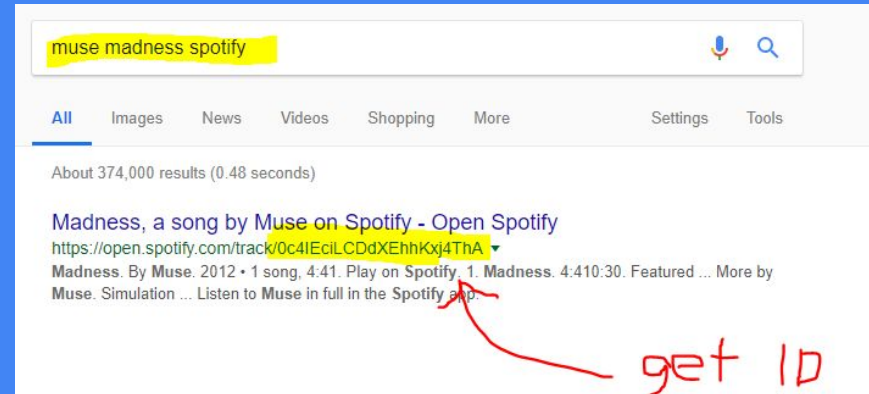
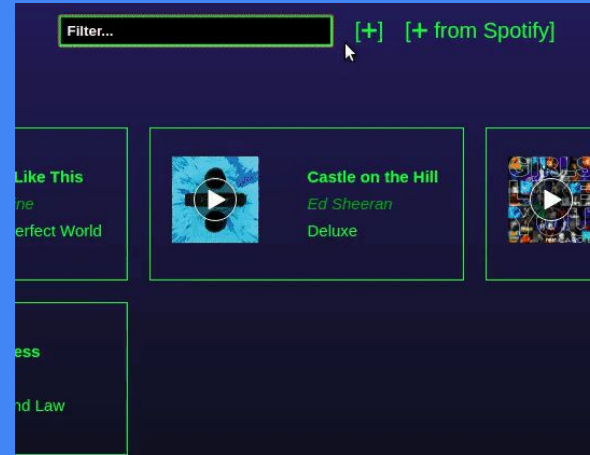
View code changes:

https://github.com/lenmorld/react_workshop/compare/c2.9...c2.10

TEST IT!

Add a song or two to your playlist
(Not a lot since it will be gone on page refresh!)

- Google “<Artist> <song> spotify track”
- Get the ID in `track/ID`
- Use ID to Create new Item



Recap of React forms

we covered:

- React core: components, JSX, state, props
- Forms, event handlers, data binding
- some object and array tricks: map, filter, ES6 spread
- ES6 arrow functions, “this”, setState()
- function props

questions?

A form example

Codesandbox:

<https://codesandbox.io/u/lenworld/sandboxes/>

sandbox name: **6_react_forms**

or use direct link: <https://codesandbox.io/s/k9v60wxp07>



3 Forms CRUD

Delete and Update item



Love Like This

Kodaline

In A Perfect World



Castle on the Hill

Ed Sheeran

Deluxe



Girls Like You (feat. Cardi B)

Maroon 5

Girls Like You
(feat. Cardi B)



Madness

Muse

The 2nd Law



Thunder / Young Dumb & Broke (

Imagine Dragons

Thunder / Young
Dumb & Broke (

c3.1 Delete and Edit

```
<UIManager>
  <List>
    <Item>
  Props passing:
    <UIManager> -- deleteItem() ---> <List> -- deleteItem() ---> <Item>
                                editItem()                editItem()
```

Delete and Update button icons will be inside **Item** as icons that appear on hover

1. **UIManager.jsx** - define *deleteItem* and *editItem* and pass down to List as function props
2. **List.jsx** - pass function props down to Item as a middleman
3. **Item.jsx** - invoke function props with required parameter on onClick event handler

NOTE: the use of **arrow functions to be able to use this**



c3.2 implementing deleteItem()

1. Copy list values, not reference
2. Filter copy using *filter()*, by excluding item to delete
3. *setState()*

NOTE: notice the current pattern in CRUD methods

1) Copy list 2) do operation on copy 3) setState

c3.3 editItem() - form **mode**



When user clicks edit, we have to show ItemForm, but let it know that we want to EDIT, not CREATE. We do this by adding a **mode** in state, and passing the item to be edited so the form fields would be populated.

UIManager.jsx

- add `this.state.form_mode`, init to 'CREATE'
- pass to **ItemForm** as props, with propname **mode**
 - *This is to show props is just a name*

ItemForm.jsx

- use **this.props.mode** to set labels correctly

```
<ItemForm  
  mode={this.state.form_mode} />
```



c3.4 editItem() - show ItemForm on edit mode

UIManager must get correct item to be edited, before passing to **ItemForm**

In **editItem()**

1. Copy list values, not reference
2. Filter copy using filter(), get the one matching item
3. setState() - set mode to 'EDIT', set form_fields to the item
4. Show **ItemForm**

Now, we are getting the item values in the form fields

c3.5 editItem() - saveUpdatedItem



When form is saved, we need a CRUD method that will apply changes to our data UIManager ***saveUpdatedItem()***:

1. Copy list values, not reference
2. Init a new empty array and copy all values here, except the updated item
3. setState()
4. Hide **ItemForm**

Pass ***saveUpdatedItem*** to **ItemForm** as function props

ItemForm:

- Based on `this.props.mode`, invoke either `this.props.create` Or `this.props.saveUpdatedItem`



c3.6 set ItemForm fields on [+] too

🐞 **BUG** 🐞: Notice that after Edit, clicking Create [+] sets the item fields to previous item incorrectly in **ItemForm**

UIManager:

1. Define **onAddItem()** function that will set mode to 'CREATE' and fields to empty
2. Replace event handler on [+] with `onAddItem()`
 - Note the use of arrow function

TEST IT!

Create, Edit, Delete, Filter (Search)

4 Integrating with Giphy API

Fetching data
Lifecycle methods
Refs

```
# axios is a promise-based HTTP library  
$ npm install axios
```

c4.1 Component setup

1. install axios - a fetching library
2. create a file **app/Giphy.jsx**

Giphy.jsx

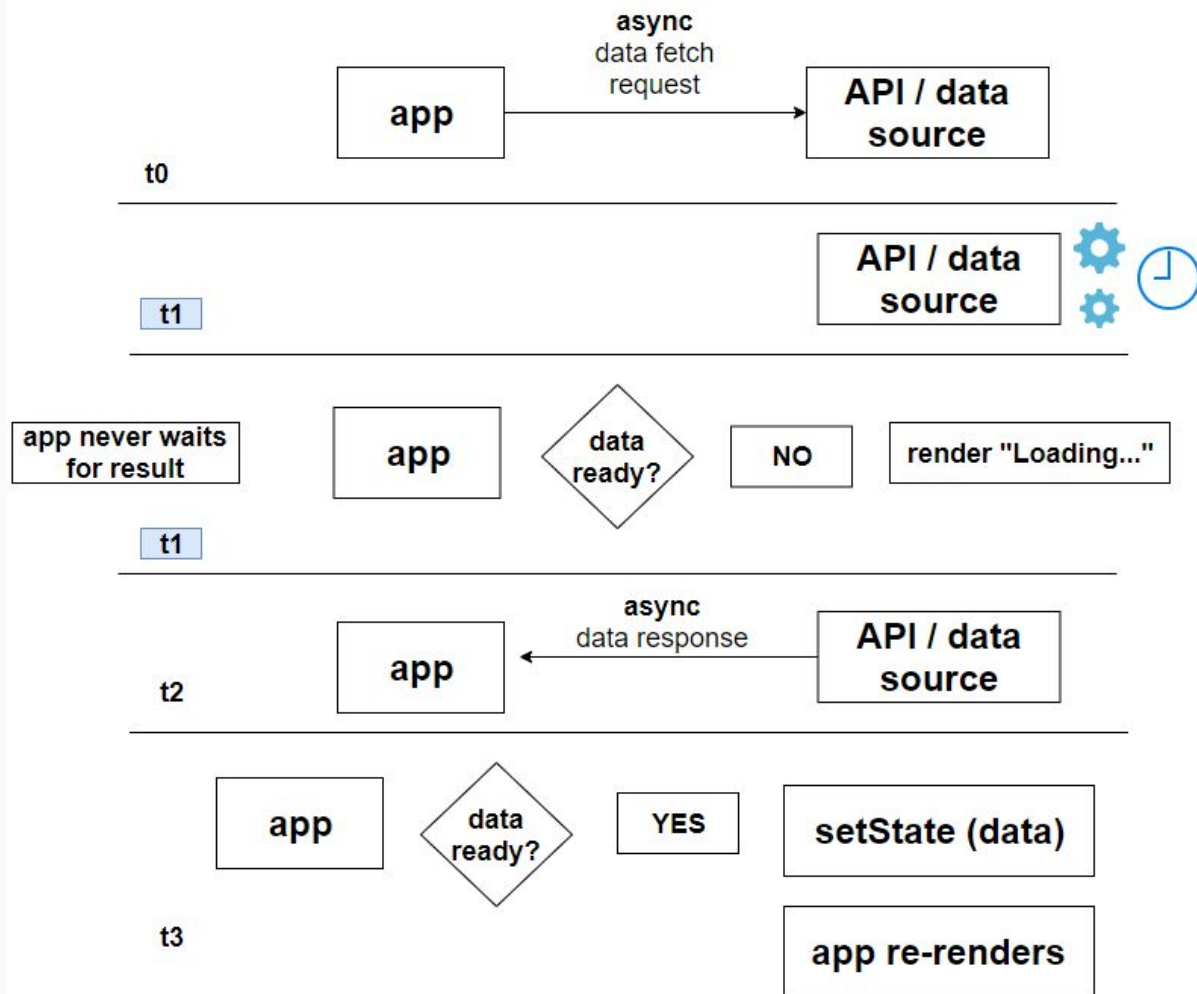
- **constructor()** with state
 - init **this.state.memes** to empty array

index.jsx

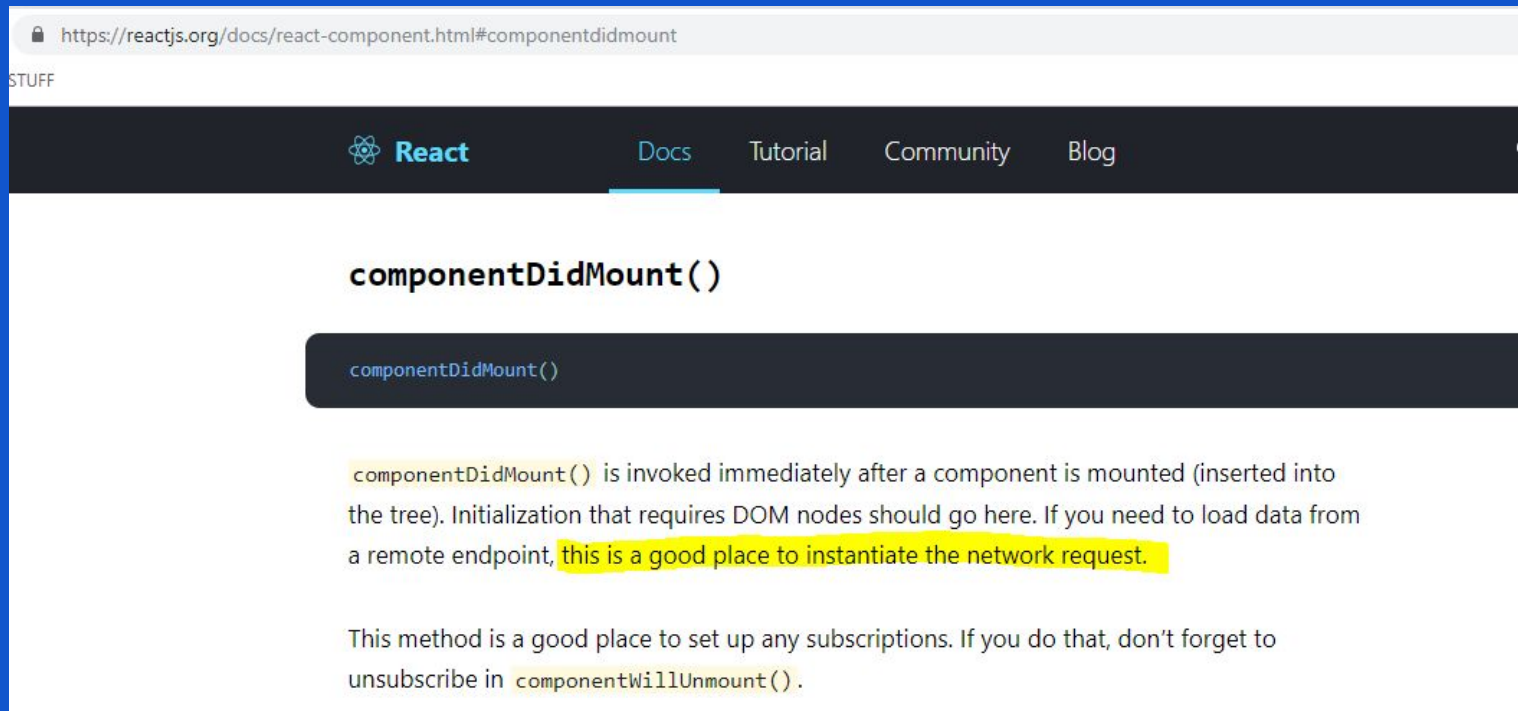
- replace `<UIManager />` with `<Giphy />` temporarily
 - *we'll fix this later in Chapter 5: Routing*

how do we fetch
data in JS?

**common
pattern in
Async data
fetch**



componentDidMount, and other React lifecycle methods



The screenshot shows the React.js documentation page for the `componentDidMount()` lifecycle method. The browser's address bar displays the URL `https://reactjs.org/docs/react-component.html#componentdidmount`. The page features a dark navigation bar with the React logo and links to Docs, Tutorial, Community, and Blog. The `componentDidMount()` title is prominently displayed. Below the title, a code block shows the method signature `componentDidMount()`. The main text explains that `componentDidMount()` is invoked immediately after a component is mounted. It notes that initialization requiring DOM nodes should occur here, and specifically highlights that it is a good place to instantiate a network request. A concluding paragraph states that this method is also a good place to set up subscriptions, provided they are cleaned up in `componentWillUnmount()`.

STUFF

<https://reactjs.org/docs/react-component.html#componentdidmount>

React Docs Tutorial Community Blog

componentDidMount()

```
componentDidMount()
```

`componentDidMount()` is invoked immediately after a component is mounted (inserted into the tree). Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request.

This method is a good place to set up any subscriptions. If you do that, don't forget to unsubscribe in `componentWillUnmount()`.

c4.2 componentDidMount()

Fetch data from backend before component is rendered. How?

`componentDidMount()`

- React lifecycle method that runs after component **mounted**
- good place to put data fetch requests, then load data to the component state

```
class Giphy extends React.Component {  
  componentDidMount() {  
    axios.get(`<REQUEST_URL>`)  
      .then((res) => {  
        console.log(res);  
      });  
  }  
  ...  
}
```



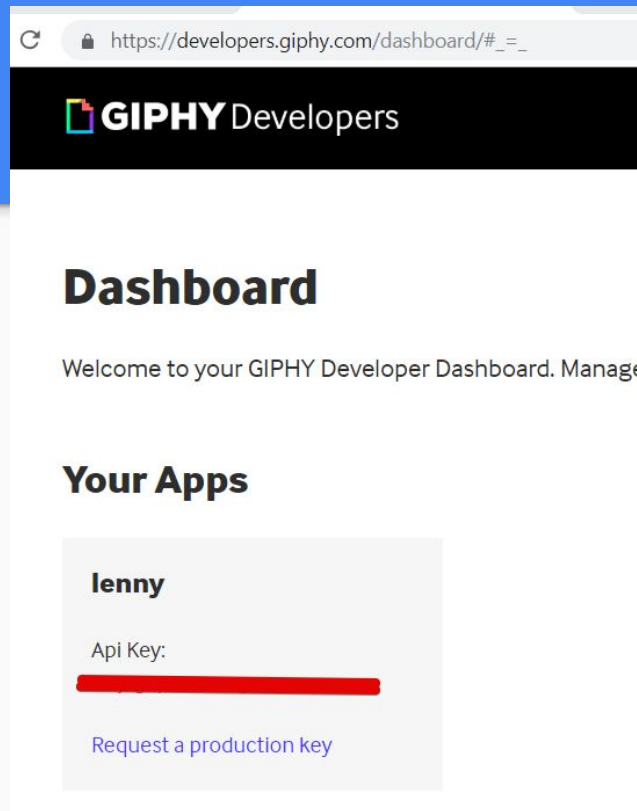
ApiTest.jsx:15

```
{data: {...}, status: 200, statusText: "OK", headers: {...}  
  , config: {...}, ...} 1  
  ▶ config: {adapter: f, transformRequest: {...}, transform...  
  ▼ data:  
    ▶ data: (5) [{...}, {...}, {...}, {...}, {...}]  
    ▶ meta: {status: 200, msg: "OK", response_id: "5c5efd...  
    ▶ pagination: {total_count: 20690, count: 5, offset: ...  
    ▶ ... : Object
```

Using the Giphy API

1. Login to <https://developers.giphy.com/>
2. Create an app
3. Get API key

NOTE: different APIs have varying degree of authentication (no auth, API key, OAuth, etc)
Check out the docs of the API you need to use.



c4.3 setState and rendering

componentDidMount()

When response is received, set current state data to response data

render()

Render array into JSX items using *map()*, following API formats and desired styling, layout, etc

```
class ApiTest extends React.Component {  
  componentDidMount() {  
    ...  
    .then((res) => {  
      const memes = res.data.data;  
      this.setState({  
        memes: memes,  
      })  
    });  
  }  
  
  render() {  
    return (  
      <div>  
        {this.state.memes.map((item) =>  
          <div key={item.id}>  
            <iframe src={item.embed_url} ... />  
          </div>  
        )}  
      </div>  
    );  
  }  
  ...  
}
```

c4.4 Show loading when data not ready

Initialize **state.list** to `[]`, to show a Loading page if results are not back yet

- Why? in cases of **slow network**, we would show Loading instead of a blank page

```
class ApiTest extends React.Component {  
  ...  
  render() {  
    if (this.state.memes.length === 0) {  
      return <div>Loading...</div>;  
    }  
    return (  
      <div>  
        {this.state.memes.map((item) =>  
          ...  
        )}  
      </div>  
    );  
  }  
}
```

Searching meme and adjusting count

Instead of hard-coding `SEARCH_QUERY` and `RESULTS_LIMIT`, we can use our knowledge of forms to include a search!

Similar to Chapters 2 and 3, we can setup *searchQuery* and *resultsLimit* in the state and wire an event handler to them (**DO THIS AS A PRACTICE**).

But since this is a smaller example, we'll explore a simpler way of using inputs in React, called **refs**.

Controlled vs Uncontrolled inputs

Controlled inputs:

- form data sync'd with state
- event handler for each input binding

Uncontrolled input:

- form data is handled by DOM
- no event handler needed to bind input

Codesandbox: **3B_controlled_inputs**

<https://codesandbox.io/s/vvpvr721l7>

Codesandbox: **3C_uncontrolled_inputs**

<https://codesandbox.io/s/0yw2xy86vv>

c4.5 refs

- Setup the refs in *constructor()* using **React.createRef()**
- Connect the refs to the inputs using **ref={this.ref}**
- Access **ref'ed** input element through **ref.current**

```
class ApiTest extends React.Component {
  constructor() {
    ...
    // React refs for searchQuery and resultsLimit
    this.searchQueryInput = React.createRef();
    this.resultsLimitInput = React.createRef();
  }
  ...
  submitSearch(event) {
    const searchQuery = this.searchQueryInput.current.value;
    const resultsLimit = this.resultsLimitInput.current.value;
    // execute request
    axios.get() {...}
    ...
  }
  render() {
    ...
    <form>
      <p>Search: <input type="text" ref={this.searchQueryInput} /></p>
      <p>Num. results: <input type="number" ref={this.resultsLimitInput} /></p>
      <button onClick={(event) => {this.submitSearch(event)}}>Search memes!</button>
    </form>
```

Meme search using Giphy API and React refs



localhost:4001

Search: classic memes

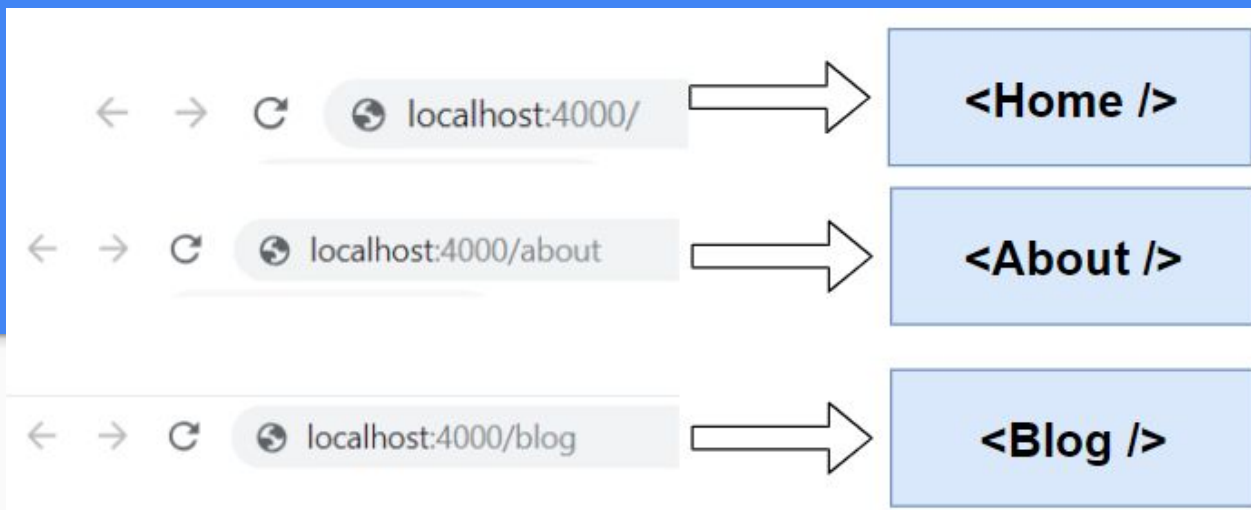
Count: 3

SEARCH!



5 Frontend Routing

Routing



When doing a SPA (Single Page Application) in React, we need a **routing** library to:

- match React components with URLs
- manage browser navigation and history

We use **react-router**

```
# React routing Library  
$ npm install react-router-dom
```

c5.1 setup routes

```
// index.jsx

import { HashRouter, Route, Link } from 'react-router-dom';

class App extends React.Component {
  render() {
    return <BrowserRouter>
      <div>
        { /* nav links to take us to the route */ }
        <nav>
          <ul>
            <li><Link to="/">Songs</Link></li>
            <li><Link to="/memes">Memes</Link></li>
          </ul>
        </nav>
        { /* match route to React component */ }
        <Route exact path="/" component={UIManager} />
        <Route exact path="/memes" component={Giphy} />
      </div>
    </HashRouter>
  }
}
```

...

c5.2 passing URL params to components

```
// index.jsx
...
<li><Link to="/greeting/en">Greeting</Link></li>
...
<Route exact path="/greeting/:lang" render={(props) => <Greeting text="Hello, " {...props} /> }
```

```
// Greeting.jsx
...
import qs from "qs";
...
class Greeting extends React.Component {
  render() {
    // debugger;
    const { text, location, match } = this.props;
    const lang = match.params.lang;
    const name = qs.parse(location.search, { ignoreQueryPrefix: true }).name;
    ...
  }
}
```

← → ↻ ⓘ localhost:4001/#/greeting/en?name=lenny

Greeting page

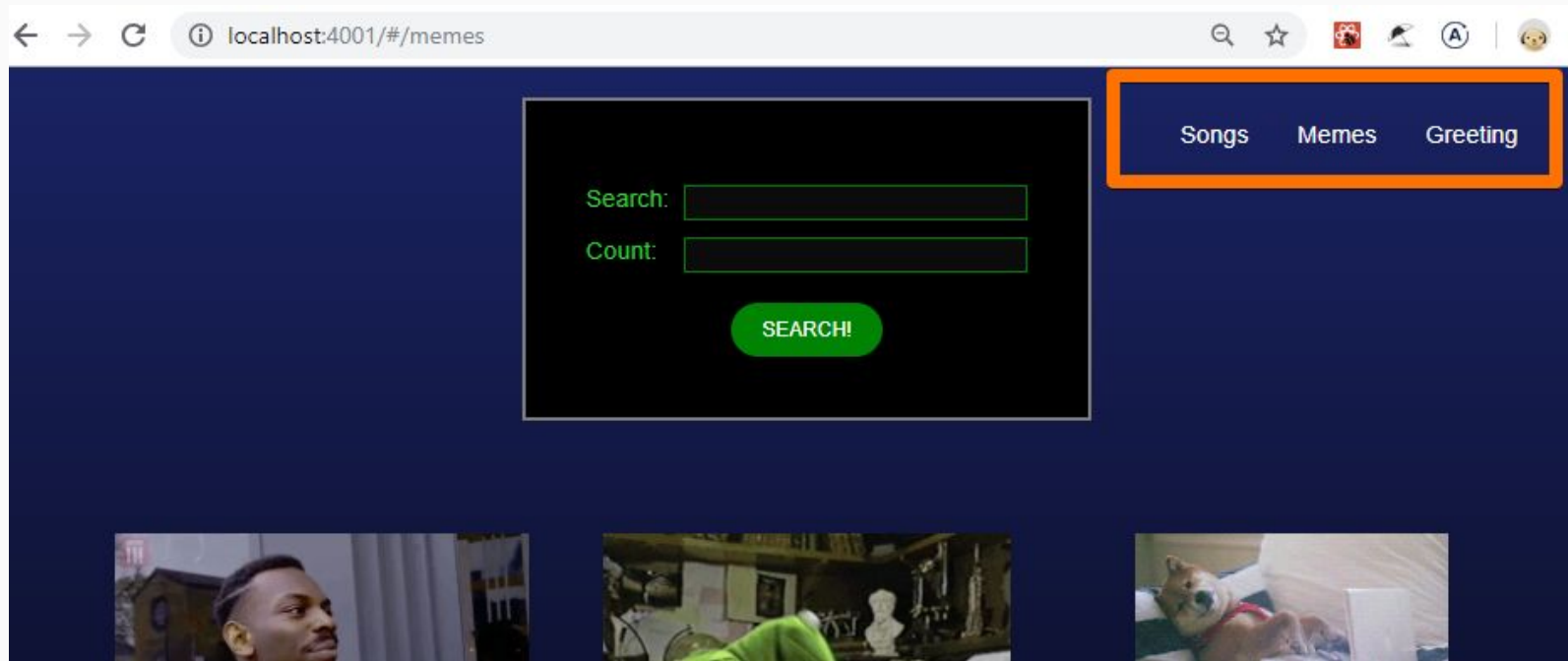
lang: en

Hello, lenny

location.search

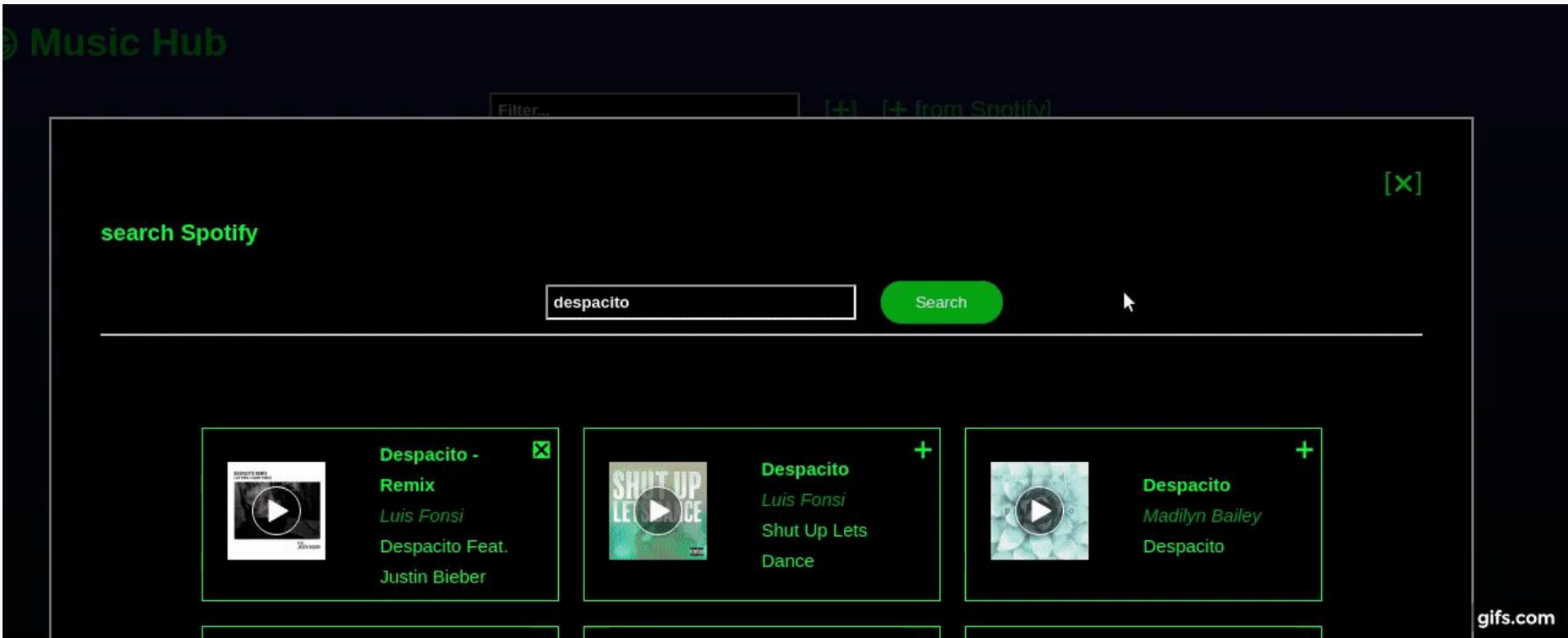
match.params

Simple routing with react-router



6 Integrating with Spotify API

Spotify search API



c6.0 Server setup

```
$ git checkout c6.0
$ npm install qs      # query parser
$ npm run dev
```

This branch contains code for:

- Spotify API connection
- Routes and controllers for Auth and searching songs
- Server routes



c6.1 Spotify.jsx

Create **app/Spotify.jsx**

Copy from Slack [#react_snippets: Spotify.jsx](#)

UIManager.jsx

- Import Spotify.jsx
- Add ***showSpotify()*** and ***hideSpotify()*** event handlers , similar to what we have before for show|hideForm() but use `.spotify_modal` as the selector
- Add button **[+ from Spotify]** and its event handler ***showSpotify()***
- Render **<Spotify />** component and pass ***hideSpotify*** as function props



c6.2 Controlled inputs

Setup **Spotify.jsx** form inputs as controlled inputs, similar to chapters 2 and 3, by defining:

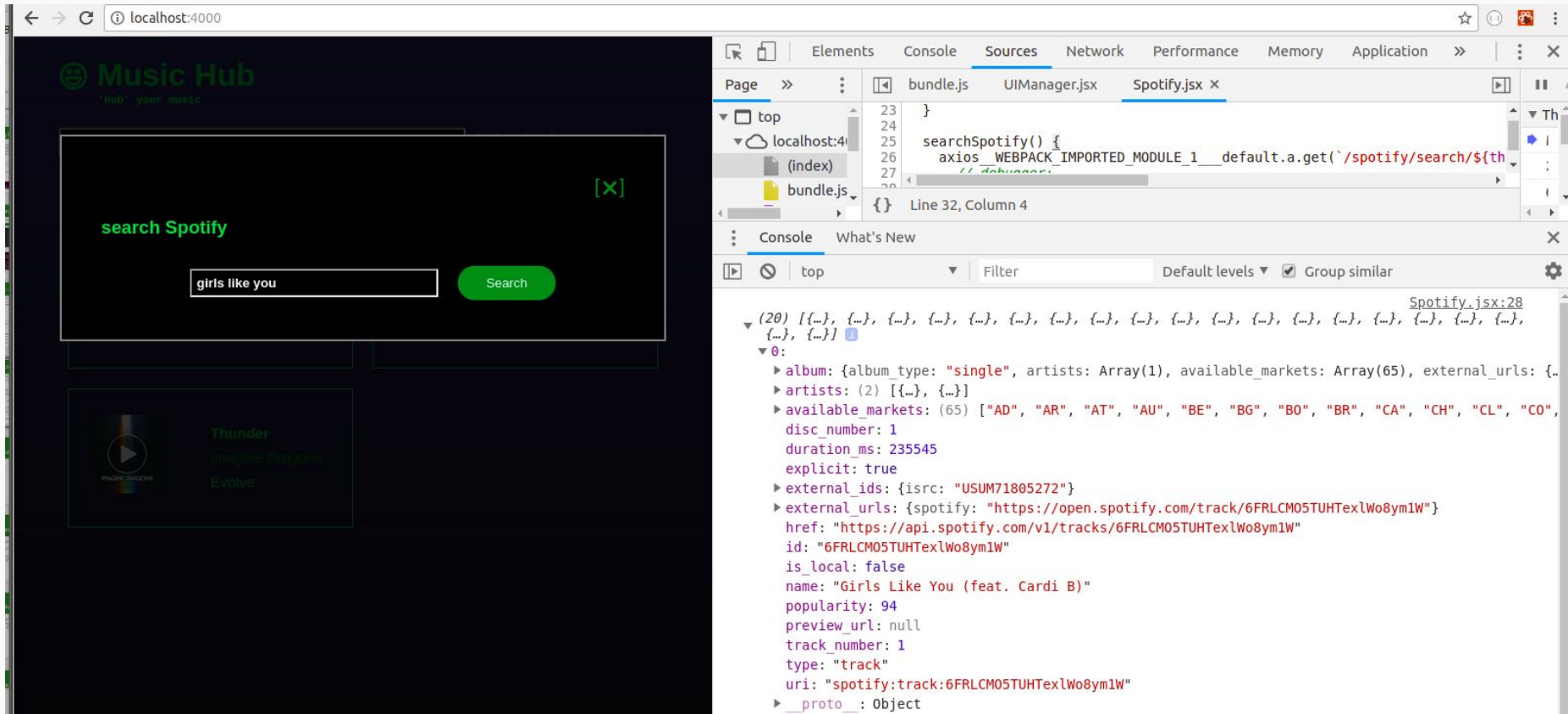
- **constructor()**
- **trackSearchTerm()**
- **searchSpotify()**

also import axios for the fetch request

```
import axios from 'axios';

class Spotify extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      searchTerm: ''
    }
  }
  trackSearchTerm(event) {
    var searchTerm = event.target.value;
    this.setState({searchTerm: searchTerm});
  }
  searchSpotify() {
    axios.get(`/api/songs?search=${this.state.searchTerm}`)
      .then((res) => {console.log(res.data);})
      .catch((err) => {
        console.log(`[Spotify.jsx] search error: ${err}`);
      });
  }
  ...
}
```

Search results in the debugger console



The image shows a web browser window on the left and a debugger console on the right. The browser window displays a web application titled "Music Hub" with the tagline "Hub your music". It features a search bar with the text "girls like you" and a green "Search" button. Below the search bar, there is a list of search results, including a track titled "Thunder" by "Imagine Dragons" with the album "Evolve". The debugger console on the right shows the "Sources" tab with the file "Spotify.jsx" selected. The console displays the search results for the query "girls like you", showing a list of tracks. The first track in the list is "Thunder" by "Imagine Dragons" with the album "Evolve". The console also shows the source code of the "Spotify.jsx" file, with the search function highlighted.

Music Hub
"Hub" your music

search Spotify

girls like you Search

Thunder
Imagine Dragons
Evolve

bundle.js UIManager.jsx Spotify.jsx x

```
23 }  
24  
25 searchSpotify() {  
26   axios__WEBPACK_IMPORTED_MODULE_1___default.a.get(`/spotify/search/${th  
27   // debugger;  
28 }  
Line 32, Column 4
```

Console What's New

top Filter Default levels Group similar

Spotify.jsx:28

```
(20) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...},  
  {...}, {...}]  
0:  
  album: {album_type: "single", artists: Array(1), available_markets: Array(65), external_urls: {...},  
  artists: (2) [{...}, {...}]  
  available_markets: (65) ["AD", "AR", "AT", "AU", "BE", "BG", "BO", "BR", "CA", "CH", "CL", "CO",  
  disc_number: 1  
  duration_ms: 235545  
  explicit: true  
  external_ids: {isrc: "USUM71805272"}  
  external_urls: {spotify: "https://open.spotify.com/track/6FRLCM05TUHTexlWo8ym1W"}  
  href: "https://api.spotify.com/v1/tracks/6FRLCM05TUHTexlWo8ym1W"  
  id: "6FRLCM05TUHTexlWo8ym1W"  
  is_local: false  
  name: "Girls Like You (feat. Cardi B)"  
  popularity: 94  
  preview_url: null  
  track_number: 1  
  type: "track"  
  uri: "spotify:track:6FRLCM05TUHTexlWo8ym1W"  
  __proto__: Object
```

c6.3 transforming API's response data into UI data

So far, this is the item format we have been using:

The object returned by Spotify is too big. We only need: **id**, **artist**, **title**, **album**.

res.data array can be mapped into a new array that contains only the track attributes we need.

```
{
  "id": "0c4IEciLCDdXEhhKxj4ThA"
  "artist": "Muse",
  "title": "Madness",
  "album": "The 2nd Law",
}
```

```
class Spotify extends React.Component {
  ...
  searchSpotify() {
    ...
    .then((res) => {
      const searchResults = res.data;
      const squashedResults =
searchResults.map((track) => {
        return {
          id: track.id,
          artist: track.artists[0].name,
          album: track.album.name,
          title: track.name
        };
      });
      console.log(squashedResults);
    });
  }
}
```

That's more like it! Now we have to put it in our UI

The image shows a web browser window on the left and a code editor on the right. The browser window displays a music hub interface with a search bar and a list of songs. The code editor shows the source code for the Spotify.js file, which is responsible for fetching and displaying the search results.

Browser Window:

- Address bar: localhost:4000
- Page title: Music Hub
- Search bar: search Spotify
- Search input: girls like you
- Search button: Search
- Results list:
 - Thunder
 - Imagine Dragons
 - Evolve

Code Editor:

- File: Spotify.jsx
- Line 32, Column 4
- Code snippet:

```
    {  
      album: "Girls Like You (feat. Cardi B)"  
      artist: "Maroon 5"  
      id: "6FRLCM05TUHTeXlWo8ym1W"  
      title: "Girls Like You (feat. Cardi B)"  
    },  
    {  
      album: "Red Pill Blues (Deluxe)"  
      artist: "Maroon 5"  
      id: "6V1bu6o1Yo5ZXnsCJU80vk"  
      title: "Girls Like You (feat. Cardi B)"  
    },  
    {  
      id: "60FHi11vdKk1nAnAA7dvz" artist: "Maroon 5" album: "Red Pill Blues (Deluxe)" title: "Gi
```

c6.4 We need a List and Item component...

But wait! We already have one! 🐼

Hooray for reusable components! 🎉

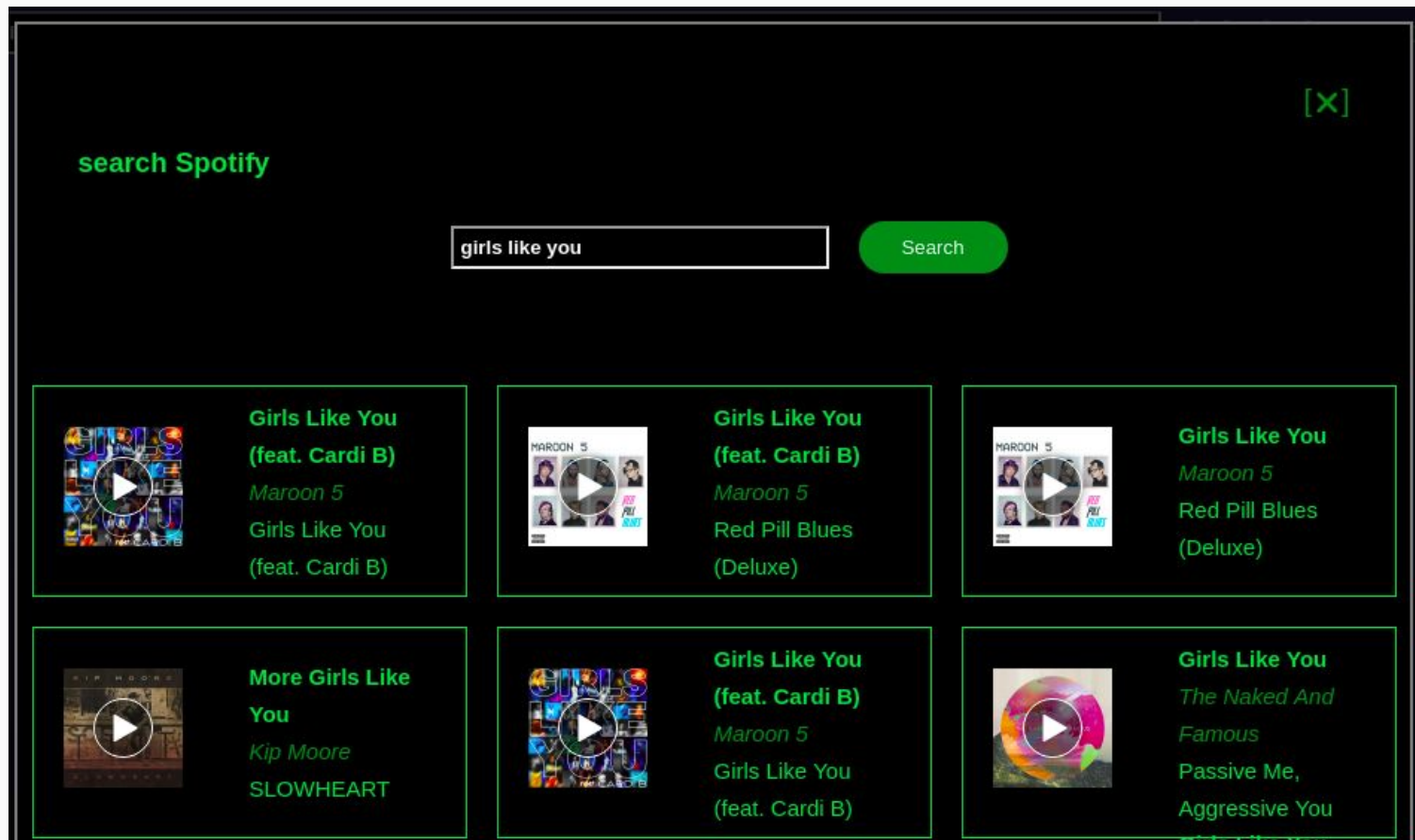
Only thing we have to do here is import and use them in **Spotify.jsx**

**** Now we have two instances of List, one is Spotify's List and the old one is UIManager's List*

```
import List from './List';

class Spotify extends React.Component {
  ...
  this.state = {
    searchTerm: '',
    searchResults: [],
  }
}
...
searchSpotify() {
  ...
  // update state with search results
  this.setState({
    searchResults: squashedResults
  });
}
...
render() {
  ...
  <List list={this.state.searchResults} />
}
```

Nice! Last thing is to add the controls in each Item to add them to our playlist



c6.5 [X] and [+] buttons on Item

To implement adding/removing items from Spotify List, we introduce the following props:

```
<UIManager>  
  <Spotify>  
    <List>  
      <Item>
```

Props passing:

```
<UIManager> toggleItemFromSpotify() ---> <Spotify> ---display_type---> <List> ---display_type---> <Item>  
                                                toggleItem()           toggleItem()
```

display_type - allows us to customize Item to have [+] instead of [X] and [Edit] (home list)

toggleItem... - when Item's [+] is clicked, item object will be passed upwards all the way to UIManager, who can add the item to our state

c6.6 implement toggleItemFromSpotify()

When an Item is clicked, UIManager can either add the item if it doesn't exist yet, or delete it if it exists already.

UIManager.jsx

- toggleItemFromSpotify()
 - use `list.some()` to determine if list contains item already
 - `createItem(item)` if exists, else `deleteItem()`
- createItem(item)
 - add item parameter, and change logic such that if null, it would get it from `state.formFields` instead
 - i.e. the item that will be added to **state** could be either from Spotify's toggleItem, OR (if null), it means the call is from ItemForm and get item from `form_fields` instead

c6.7 Denote Item X instead of + if already in playlist

```
<UIManager>  
  <Spotify>  
    <List>  
      <Item>
```

Props passing:

```
<UIManager> --isInStateList()---> <Spotify> --isInStateList()---> <List> --isInStateList---> <Item>
```

UIManager.jsx

isInStateList() - checks if passed *item_id* is already in UIManager's *this.state.list*

Implementation using `some()`, similar to logic in `toggleItemFromSpotify()`

Item.jsx

display [X] instead of [+]

if ***isInStateList()*** returns true

```
...  
<div className="add_remove">  
  <span onClick={() => this.props.toggleItem(item)}>  
    { this.props.isInStateList(item.id) ? 'X' : '+' }  
  </span>  
</div>  
...
```

Some final testing:

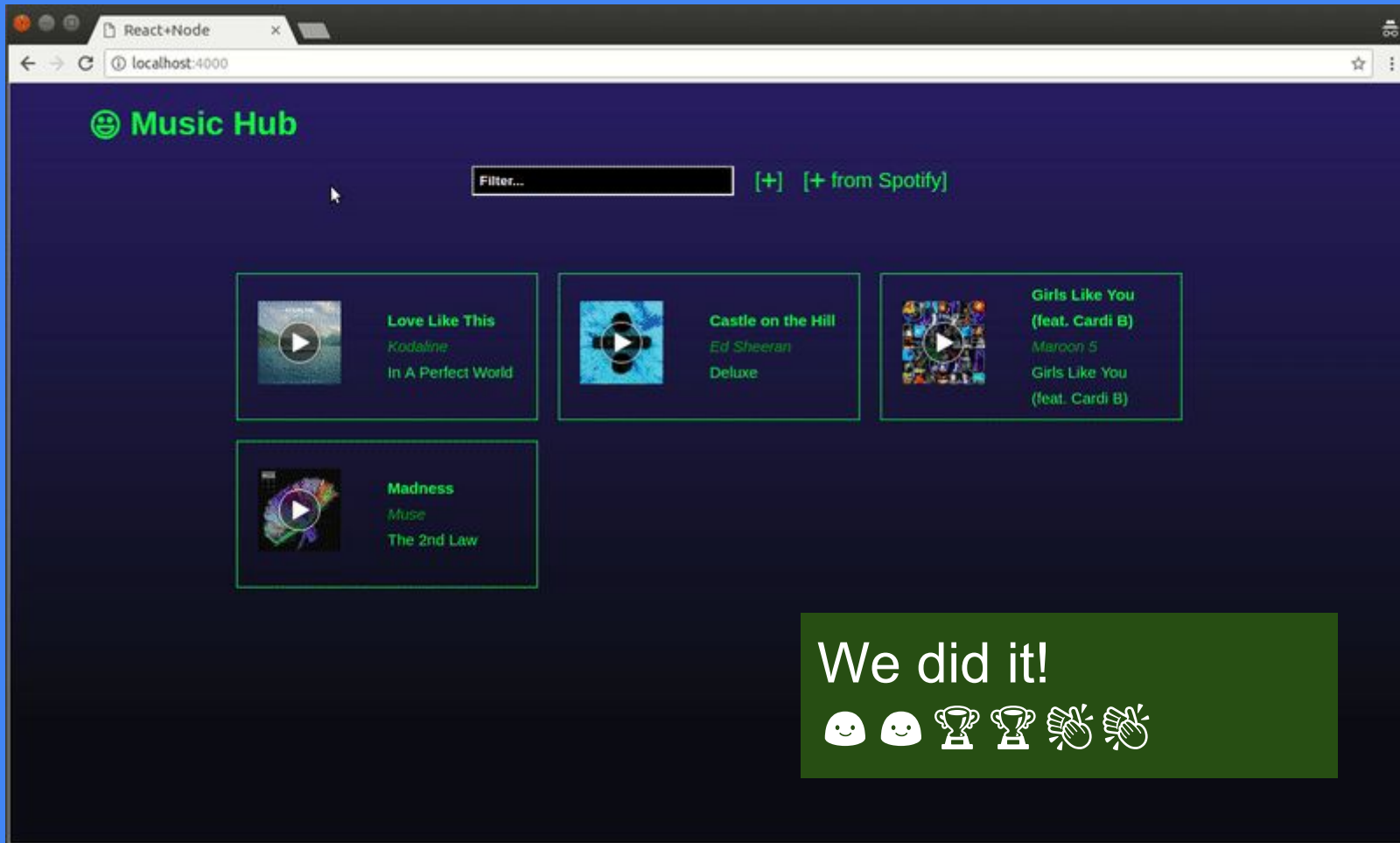
Test [+]. Here's a few sample songs

4uLU6hMCjMI75M1A2tKUQC
7GhIk7II098yCjg4BQjzvb
0FutrWIUM5Mg3434asiwkp

[+] [+ from Spotify]

Test [+ from Spotify].

Try searching and adding; removing already added items in Spotify List



We finished the app!



Questions?

Need help?

Coming up... FAQ and Advanced Stuff

Let's make this better!

After the workshop/before you leave

Please answer a short survey about the workshop

<https://goo.gl/M6Bdc3>

Please go to the code repo, and give it a star

(if it deserves one, of course!)

https://github.com/lenworld/react_workshop



Any mistakes or improvements, please submit an issue
or even better, contribute to open-source! (ping me in Slack for details)

New issue

Appendix A: FAQs and tips

- Why use a frontend framework (React, Angular, Vue) ?
- Why learn any/all of these JS frameworks?
- Why ES6?
- Any recommended resources to improve React skills?

Final tip: Beef up your CV with different technologies and projects. You'll learn a lot and it would help you get you an internship/job!

Appendix B: Intro to Advanced React Concepts

- Server stuff
- Lifecycle Methods
- Children props
- Context
- Render props
- HoC (Higher Order Components)
- Hooks
- Surprise
- Portals

Server stuff

...To learn more about backend/server/API, attend my **Node workshop !..**



Server setup used:

- *Node + Express* on the backend
- *webpack-dev-server* in the frontend.

SSR (Server-side Rendering)

This is our current SPA (Single Page Application) setup:

- There is only one page (index.html) served in / (localhost:4001/)
- All of the JS, React stuff is handled client-side, routes handled by **react-router**

Although SPA saves on network requests, you might want to consider SSR when

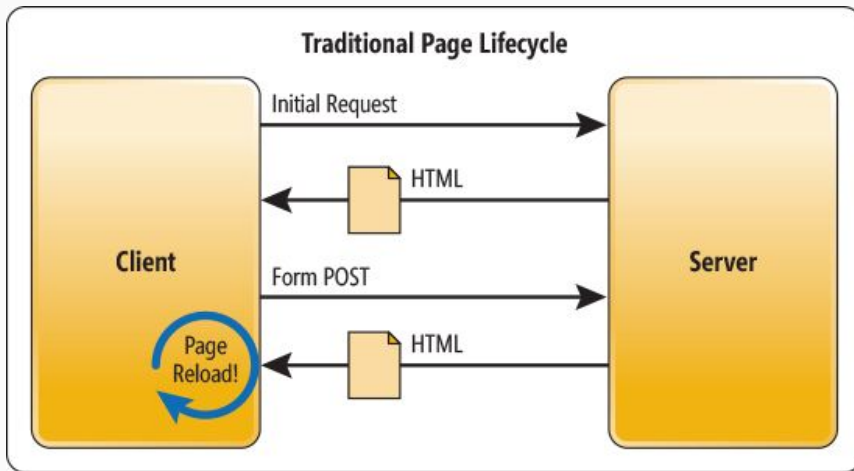
- You need SEO (e.g public-facing websites like blogs, markets, portfolio)

There are ways to combine SPA + SSR:

- in React: <https://goo.gl/9uAD1M>
- separate SSR pages from SPA page through routing
 - e.g: SSR on **website.com (/blog,/about,...)**, SPA on **app.website.com**

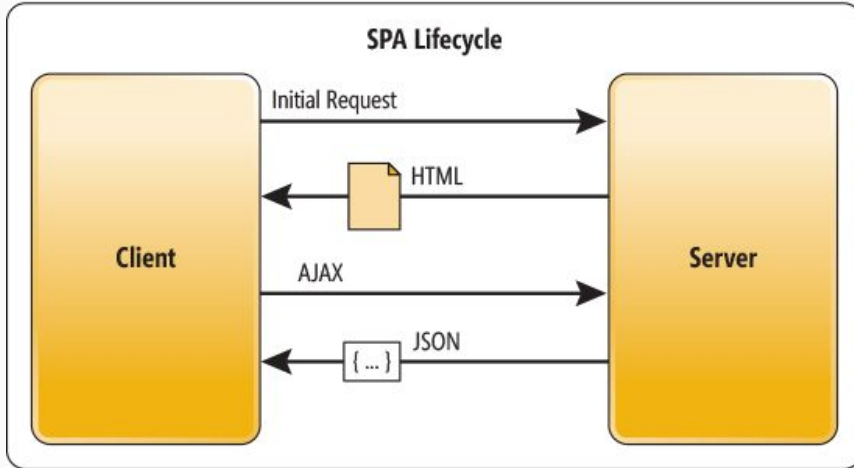
There are frameworks that provide SSR, routing, and so much more out of the box:

- Next.js, Gatsby



SSR:

- initial and subsequent page loads same time and data
- less load on client, more load on server
- good for low-processor devices



SPA:

- initial page load takes a lot of time (and data), but subsequent is fast
- most of the routing is done in the frontend, so server only takes care of data requests
- good for high-processor devices

Lifecycle methods

Full reference here: <https://reactjs.org/docs/react-component.html>

- **constructor()**
 - initialize state here
- **render()**
 - return JSX here, using *this.state* and *this.props*
- **componentDidMount()**
 - fetch data here, then use *this.setState({ data })*
- **shouldComponentUpdate()** - let React know if component should re-render
- **componentDidUpdate()** - called after re-renders (but not on initial render)
- **componentWillUnmount()** - called after component is unmounted and destroyed; put garbage collection here, when necessary

Codesandbox: **A_Lifecycle_1** <https://codesandbox.io/s/5z1rk9776p>

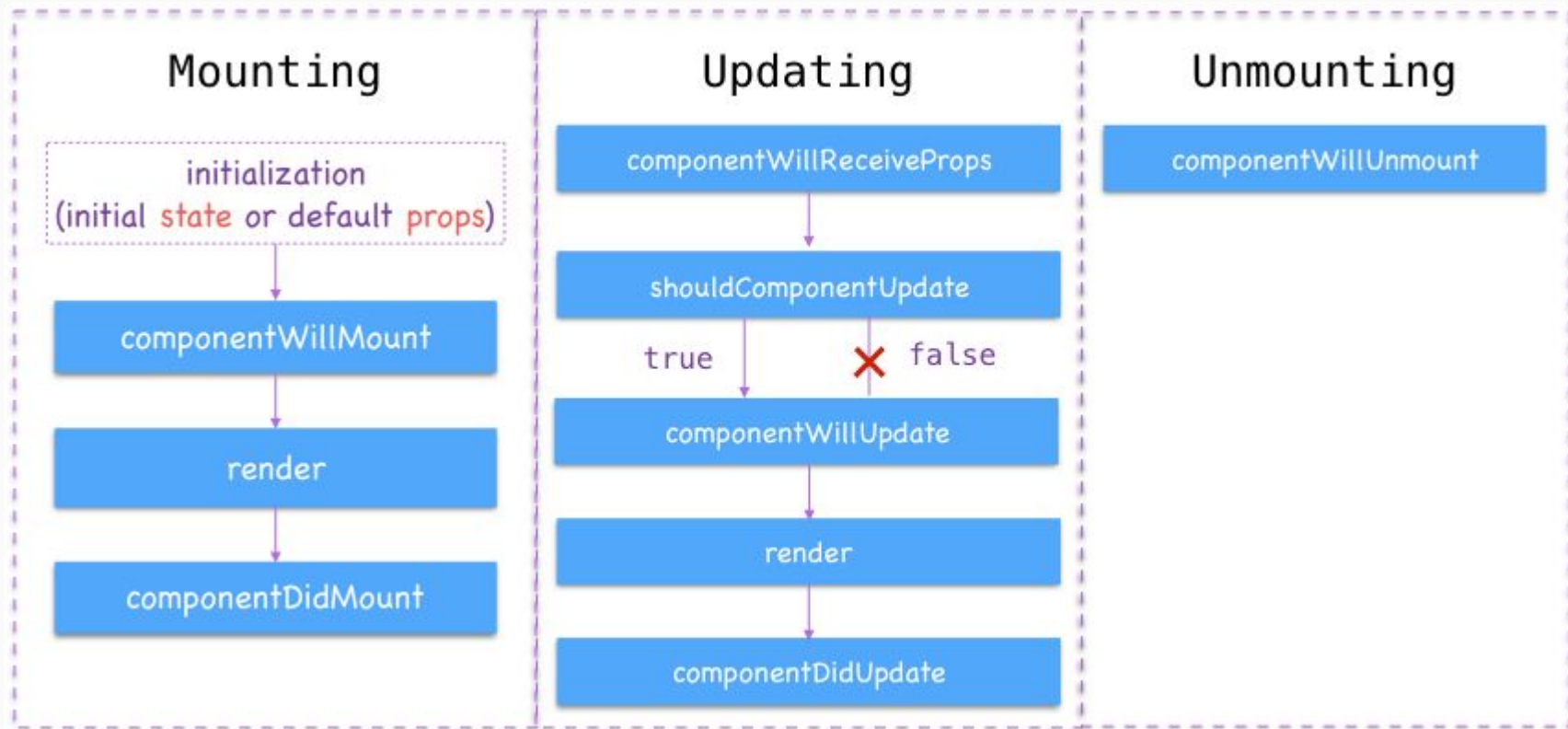


image from: <https://rangle.github.io/react-training/react-lifecycles/>

Prop-drilling and some alternatives

codesandbox: A_prop-drilling <https://codesandbox.io/s/w2xyprw937>



Prop-drilling - passing down data from ancestor to a child, when the middle ones don't need it

There are alternatives but they add some complexity to a React app. Thus, **complexity vs verbosity**. These are ordered in least complex to the most.

1. Children Props
2. Context
3. Redux

Children props

codesandbox: A_children_props <https://codesandbox.io/s/rllz7rx2n>

Instead of using the `<Component />` syntax, we can do

`<Component>`

where `<Stuff />` can be any valid JSX

`<Stuff />`

`</Component>`

then inside **Component** `render()`, we can access `<Stuff />` via ***this.props.children***

Perfect for “generic placeholders”, which could be dumb components that just renders whatever the parent passes

e.g. **Nav, Body, Container, Sidebar, Card** components

Context

codesandbox: A_context_1 <https://codesandbox.io/s/q7o3096p5w>

React Context API involves the use of a

← → ↻ <https://daveceddia.com/context-api-vs-redux/> ☆

- The `React.createContext` function which creates the context
- The `Provider` (returned by `createContext`) which establishes the “electrical bus” running through a component tree
- The `Consumer` (also returned by `createContext`) which taps into the “electrical bus” to extract the data

another example of using Context: **advanced_react_context**
<https://codesandbox.io/s/21w12jj4y>

```
render() {  
  return (  
    <div>  
      <UserContext.Provider value={this.state.username}>  
        <Nav />  
        <Body />  
      </UserContext.Provider>  
    </div>  
  );  
}
```

```
// Card is a stateless component,  
// using render props syntax  
const Card = () => (  
  <UserContext.Consumer>  
    {username => <div>{username}</div>}  
  </UserContext.Consumer>  
);
```

Redux

- A complete **state management** solution with a **data store**
- Based on concepts of **immutability** and **functional programming**
- Makes state predictable; *time travel through state*
- Testing and debugging
- simplifies read/write on entire state to local storage/Server-side rendering
- Adds significant complexity and boilerplate to an application
- Not an “addon” library; entire app needs to be restructured
- Discourages OOP, since it's built around FP

HoC and render props

codesandbox: A_hoc-and-render-props <https://codesandbox.io/s/0y09y8m14p>

These two patterns emerged from the need to effectively reuse code and share data between reusable components.

HoC (Higher Order Components) - a function that takes a component and returns a new component

Render props - a **function prop** that a (reusable) child component uses to know *what* to render. The parent component expresses the *how*.

Portals

codesandbox: A_portals <https://codesandbox.io/s/7ylx55z0zx>

Allow a parent to render children outside its hierarchy,
i.e. into a DOM node that exists outside the DOM hierarchy of the parent
component

Perfect for modals, tooltips, dialogs

Another example: rendering into a different browser window

<https://hackernoon.com/using-a-react-16-portal-to-do-something-cool-2a2d627b0202>

Suspense

For async fetch requests, we used to do:

```
if (this.state.memes.length === 0) {  
  return <div>Loading...</div>  
}
```

But for more complex data fetching and more elegant UX, it is better to use **React Suspense**. It allows you to defer rendering part of your application tree until some condition is met (e.g. data from an endpoint or a resource is loaded)

An advanced example: <https://codesandbox.io/s/k2m12p8nqr>

Hooks

Hooks let you use state, lifecycle methods and other class features, with functional/stateless components, i.e. without writing a class

Really good intro here:

<https://scotch.io/tutorials/getting-started-with-react-hooks>

Final words...

This is just the beginning of your React learning...

A few tips:

1. Get comfortable with the foundation first (state, props, event handlers) before moving to the advanced stuff, so you don't get overwhelmed
2. Fork the repo, modify it, break it, add some stuff, make it your own
3. Message me in Slack anytime

Let's make this better!

After the workshop/before you leave

Please answer a short survey about the workshop

<https://goo.gl/M6Bdc3>

Please go to the code repo, and give it a star

(if it deserves one, of course!)

https://github.com/lenworld/react_workshop



Any mistakes or improvements, please submit an issue
or even better, contribute to open-source! (ping me in Slack for details)

New issue

Thank you!!!

