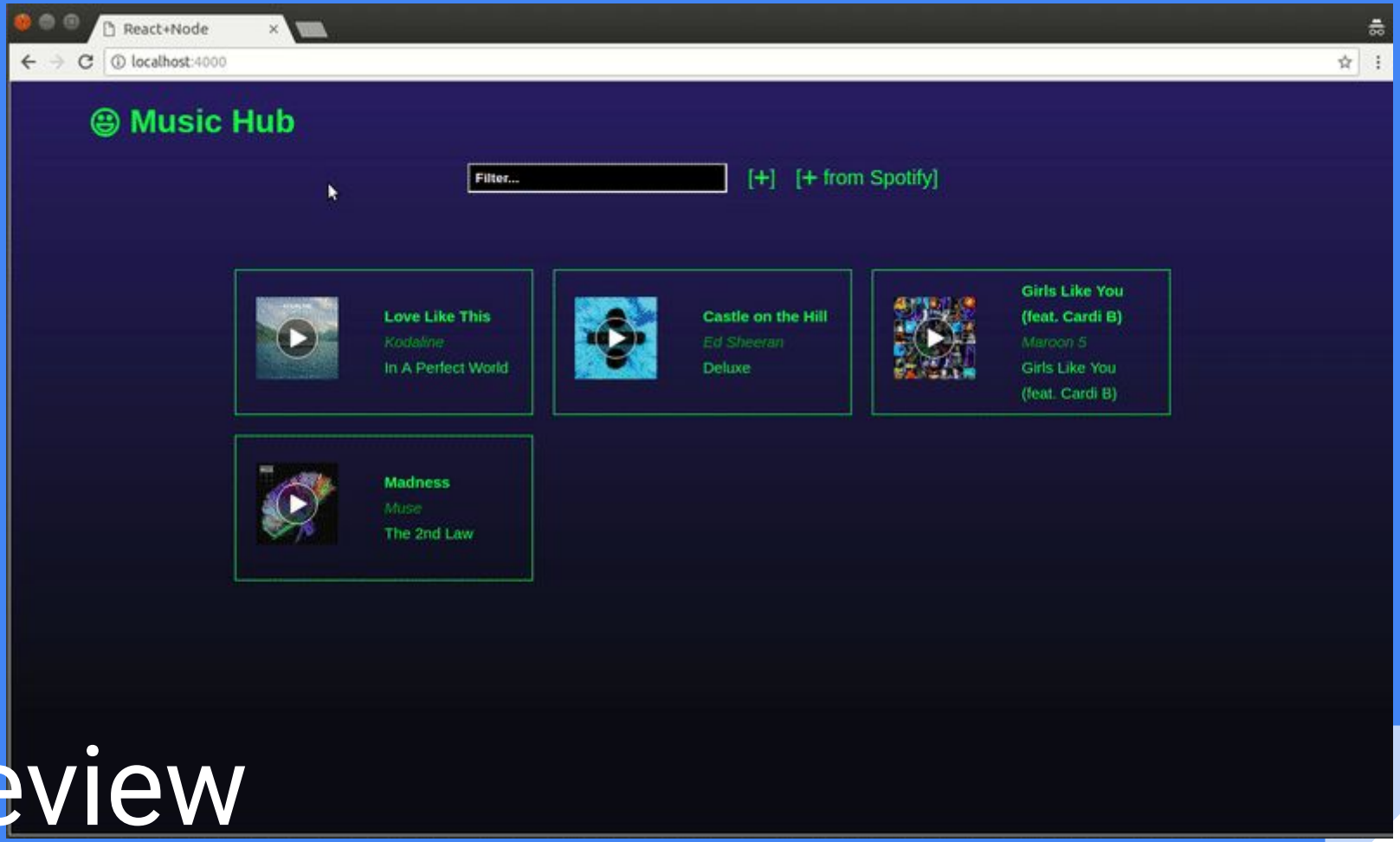
React and Node workshop

Making a full-stack web app with Spotify API


React+Node
localhost:4000
Music Hub
Filter...
[+]
[+ from Spotify]
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

Preview

me

Lenny
(Lenmor Larroza Dimanalata)

Web Developer at Autodesk

I love hackathons and coding projects

lenmorld
Github, Linkedin, lenmorld@live.com

you

- You have an interest in web dev
- You may have heard of React, Node and curious about it
- You have some knowledge (or interest) in HTML, CSS, JS, web stuff

What is this about?

Full-stack web app development with:

- React: UI (frontend) JS framework
- Node: server (backend) JS framework
- Important JS stuff for web app dev

We will discuss passively:

- Webpack
- Npm
- ES6

Depending on time, we may not touch on:

- CSS
- Advanced JS stuff

We need:

For Windows users:
Check out this doc if having problems with Node, npm installation
https://goo.gl/WN3LmA

1. WiFi! 📶
2. Join the Slack **reactnodeworkshop** https://goo.gl/buZF71
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. Install mongodb (we will use mLab but local DB is good to have)
   - Follow https://docs.mongodb.com/manual/installation/
8. Check out the cheat sheet https://goo.gl/m8i2gc for some quick info

Git setup

1. Log-in to github
2. Go to https://github.com/lenmorld/rnw
3. In github, fork branch 

   - You'll be asked to login to your github account
4. On local terminal, clone the repo

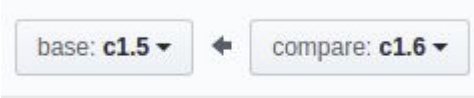
   ```
   $ git clone https://github.com/<username>/rnw.git
   ```

   or use SSH if you'd like
5. Checkout dev branch (which should be an empty slate)

   ```
   $ cd rnw
   ```

   ```
   $ git checkout dev
   ```
6. Open code editor with **rnw** as the root directory
7. **We're ready!** 🚀

If needed: exploring workshop code

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>/rnw/compare/c1.5...c1.6
   Or use 

3. Checkout code at any step, e.g:

```
$ git stash                    // (or push in a new branch)
$ git checkout <chapter>       // e.g.    git checkout c4.1
$ npm install                  // very  important to make code work
// if needed, restart terminal processes for npm start and npm run dev
```

4. Try your best to catch up. Material could go fast
5. Code is all yours after (MIT License)

Have these links open in your browser (or handy)! Also in Slack #resources

- **RNW compare** branches: e.g. https://github.com/<username>/rnw/compare/c3.3...c3.4
- **RNW files**: https://github.com/lenmorld/rnw_files/
  - Supplementary files to make our coding faster
- **Slack** - better to have Desktop client
- **Codesandbox** - testing/understanding a piece of code separate from app: https://codesandbox.io/u/lenmorld/sandboxes
- **Cheat sheet** - https://goo.gl/m8i2gc

SORRY! Resources overload! 📚📚📚

Node Basics

c1.0 Hello World console.log()

Execute *node server.js*

```
$ node server.js
>> Hello World!
```

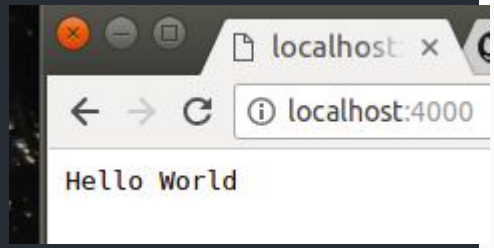
Sample directory structure:

```
react_node_workshop/
    server.js
    README.md
    ...
```

c1.1 - Hello World! server

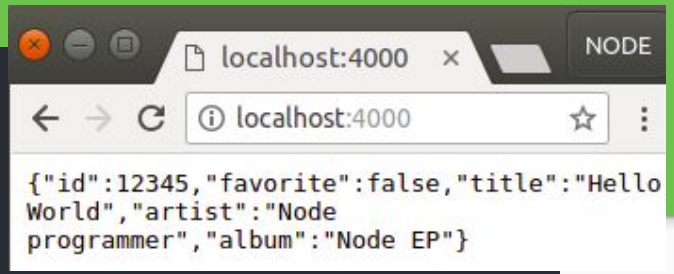
```
# to run server
$ node server.js
>> Starting server at 4000

# CRTL+C to stop server
```

```
// server.js

// import built-in Node package
var http = require('http');
var port = 4000;

var server = http.createServer(function (req, res) {    // Callback function
   // Response header
   res.writeHead(200, { "Content-Type": "text/plain" });
   // send response
   res.end("Hello World\n");
});

server.listen(port, function () {   // Callback function
   console.log("Starting server at " + port);
});
```



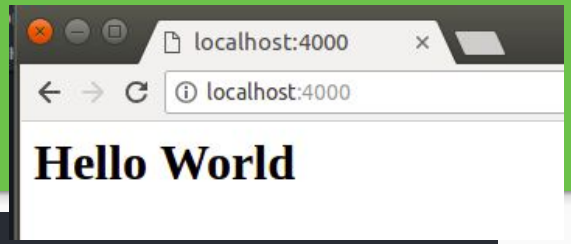
C1.2 - Hello World - JSON

```
// server.js

// import built-in Node package
var http = require('http');
var port = 4000;

var server = http.createServer(function (req, res) { // Callback function
   // Response header
   res.writeHead(200, { "Content-Type": "application/json" });

   // JSON object
   var song = {
       id: 12345,
       favorite: false,
       title: "Hello World",
       artist: "Node programmer",
       album: "Node EP"
   };

   // send JSON response to client
   res.end(JSON.stringify(song));      // JSON.stringify({a: 1}) -> '{"a":1}'
});

server.listen(port, function () { // Callback function
   console.log("Starting server at " + port);
});
```




```
# to restart server
# CRTL+C to stop server
$ node server.js
>> Starting server at 4000
```

C1.3 - Hello World - HTML




```javascript
// import built-in Node package
var http = require('http');
var port = 4000;

var server = http.createServer(function (req, res) { // Callback function
   // Response header
   res.writeHead(200, { "Content-Type": "text/html" });

   // send HTML response to client
   res.end("<h1>Hello World</h1>");
});

server.listen(port, function () { // Callback function
   console.log("Starting server at " + port);
});
```

```
# to restart server
# CRTL+C to stop server
$ node server.js
>> Starting server at 4000
```

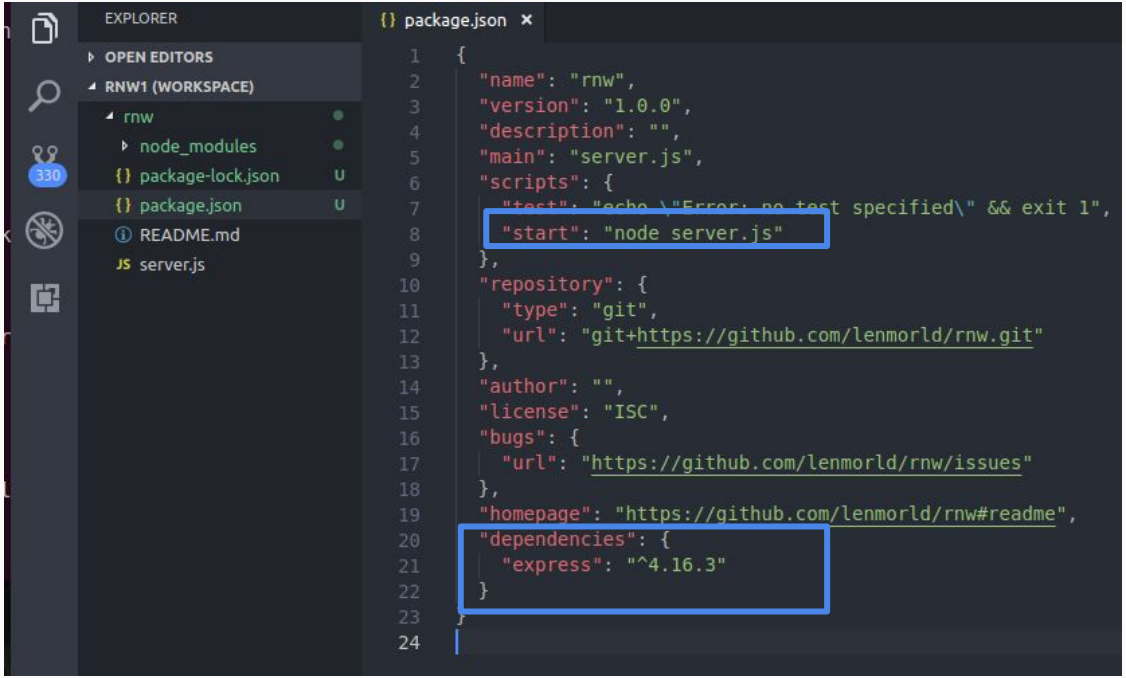
c1.4 - Using *express* middleware

Express simplifies web server stuff in Node
But first, to install any package (dependency, library) in Node, we need **npm**

```
$ npm init
# leave defaults; press Enter
until it finishes

$ npm install express
```

Examine ***package.json***

- installed dependency: express
- Npm scripts



Sidenote: Npm, package.json, node_modules

Node_modules contains all the packages locally

- **add to *.gitignore* !!!**

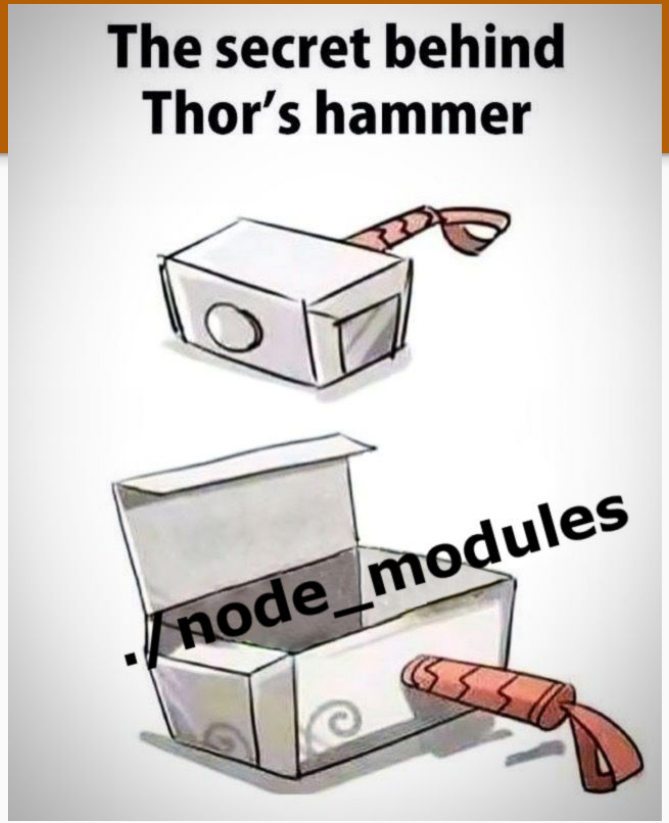
.gitignore ×

```
1   node_modules/
```

Directory structure:

```
react_node_workshop/
      .gitignore
      node_modules/
      server.js
      package.json
      package-lock.json
```

*include **package-lock.json** in the files you check in to git
This is useful in *semvar*



c1.5 - Using *express* middleware

Now we can serve **index.html** using Express

Directory structure:

```
react_node_workshop/
    server.js
    index.html
    ...
```

```
// server.js

var http = require('http');
var express = require('express'); // import express
var server = express();

var port = 4000;

server.get("/", function(req, res) {
   res.sendFile(__dirname + '/index.html');
});

server.get("/json", function(req, res) {
   res.send((JSON.stringify({ name: "Lenny" })));
});
...
```

Create ***index.html*** in project root

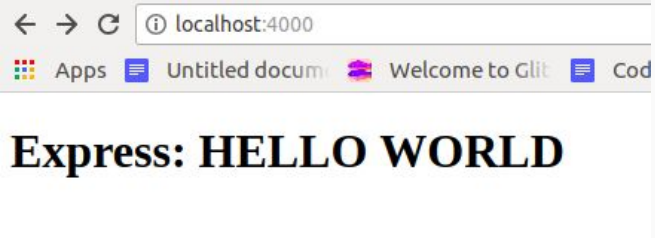
```
<!-- index.html -->

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>React+Node</title>
  </head>
  <body>
      <h1>Express: HELLO WORLD</h1>
  </body>
</html>
```

c1.5 Running server and testing

Since we have npm now,
Instead of *node server.js*, we can do this to run server

```
# to restart server
# CRTL+C to stop server
$ npm start
```

To test:



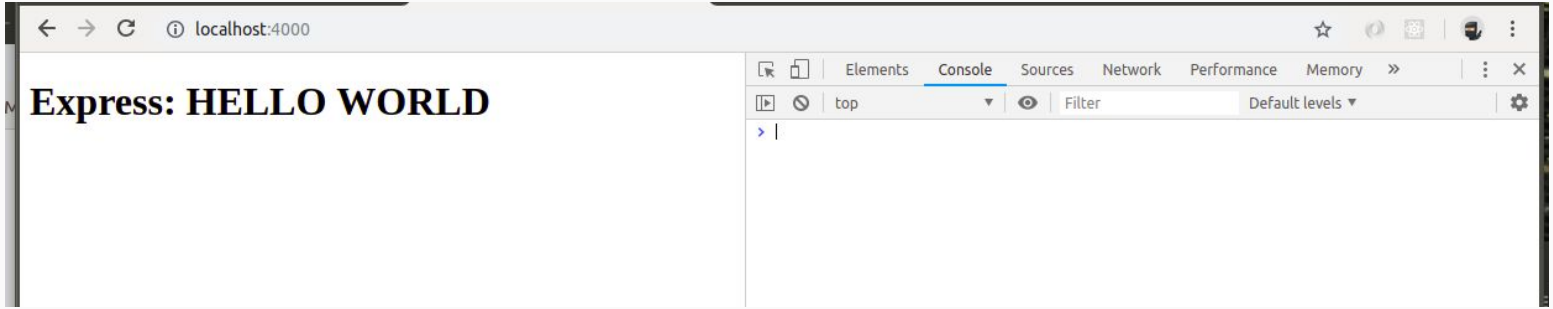

```
# in another terminal tab/window
$ curl localhost:4000/json
>> {"name":"Lenny"}
```

c1.6 auto restart server.js on changes

```
# install nodemon globally, sudo might be needed
$ npm install -g nodemon

# in package.json, change start script to
# nodemon server.js

$ npm start
```

```
// package.json
...
 "scripts": {
   "test": "...,
   "start": "nodemon server.js"
 }
...
```

Browser dev tools



Watch out for errors in **Console** tab

React and webpack setup

c2.1 Install webpack and react deps

- Create file *webpack.config.js*, from template
  - Copy from **RNW files:** webpack.config.js
  - Quick walkthrough of webpack
- Install webpack and other needed plugins

```
$ npm install --save-dev webpack webpack-cli babel-cli @babel/core @babel/preset-react
babel-loader style-loader css-loader babel-preset-env

$ npm install react react-dom
```

OR, copy deps in **RNW files:** package.json
then do

```
$ npm install
```

- Add a **dev** entry in npm scripts

```
// package.json
...
 "scripts": {
   ...
   "dev": "webpack --watch"
 }
```

c2.1 React root file at app/index.jsx

- Create folder **app/** and inside it, create file **index.jsx**
  - **app/** folder will hold all of the front-end code

Directory structure after:

```
react_node_workshop/
  ...
  webpack.config.js
  app/
    index.jsx
```

running npm and webpack together

1st terminal tab/window - BACKEND

```
# to restart server
# CRTL+C to stop server
$ npm start
```

2nd terminal tab/window - FRONTEND

```
# to start webpack in watch mode

$ npm run dev
```

3rd terminal tab/window also useful for git, cURL, npm install, etc

```
# other stuff
$ curl http://localhost:4000
```

c2.2 React setup

```
// server.js
...
server.use(express.static('public'));

server.get("/", function(req, res) {
...
```

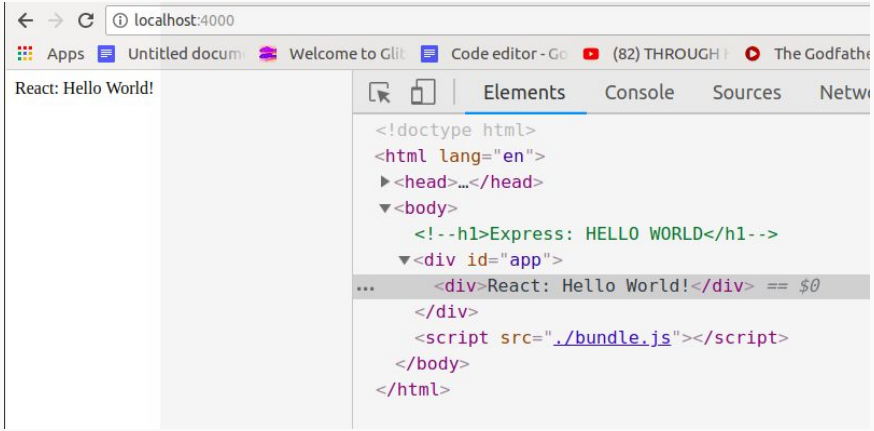
- In **server.js**, serve ***public/*** dir
- Write React code in **app/index.jsx**

```
// 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'));
```

c2.2 React setup

- Include React app in **index.html**
- We can also comment out the other HTML in the page for now

```html
<!-- index.html -->

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>React+Node</title>
  </head>
  <body>
     <!--h1>Express: HELLO WORLD</h1-->
      <div id="app"></div>
      <script src="./bundle.js"></script>
  </body>
</html>
```



C2.3 CSS files setup

Directory structure after:

```
react_node_workshop/
  ...
  public/
     styles.css
     bundles.js
  ...
```

1. create css file inside public

public/styles.css

Copy styles from this link: https://github.com/lenmorld/rnw_files/blob/master/styles.css

2. Set viewport for better responsive mobile viewing
3. include styles file from HTML head

```html
<!-- index.html -->

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta content="width=device-width, initial-scale=1" name="viewport" />
    <link rel="stylesheet" href="./styles.css" />
    <title>React+Node</title>
  </head>
  <body>
   ...
</html>
```

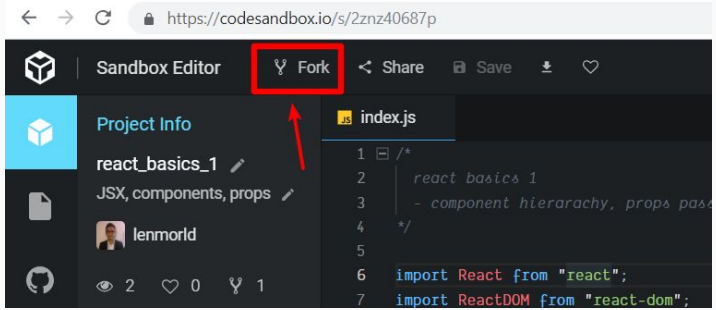
React Basics

5 minute React Intro

Codesandbox: **react_basics_1**

https://codesandbox.io/s/2znz40687p

Please fork before editing



Frontend Design

What are we building?

Music playlist web app, where user can "CRUD" songs

What components do we need?

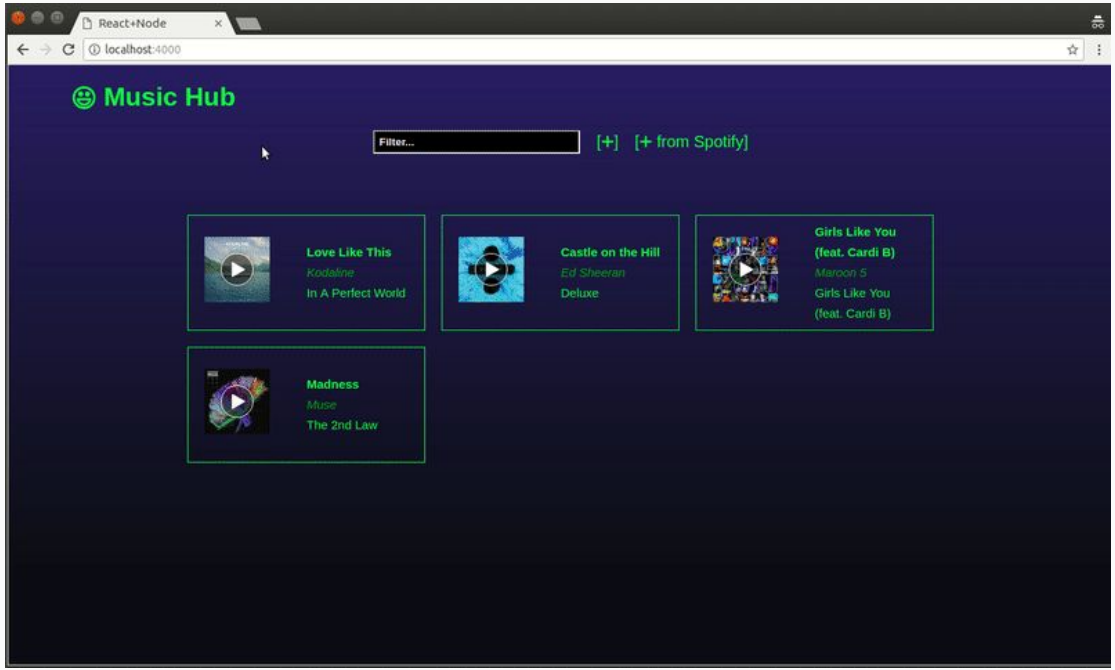
App (index.jsx) - loads React tree into DOM

UIManager.jsx - manages entire app, provides access to data (e.g. facade, API)

Header.jsx - HTML header

List.jsx - contain/manage the list of items

Item.jsx - display/manage an item



Component tree

```
App
    UIManager
        Header
        List
            Item
        ...
```

c3.1 Spotify Data

- Create file ***app/data.js***
  - Copy from **RNW files:** data.js
- List and Item data objects (model)
  - Maps to List and Item React components

```
// app/data.js

var data = {
  "list": [
    {
      "id": "0c4IEciLCDdXEhhKxj4ThA"
      "artist": "Muse",
      "title": "Madness",
      "album": "The 2nd Law",
    },
    {
      "id": "2QAHN4C4M8D8E8eiQvQW6a"
      "artist": "One Republic",
      "title": "I Lived",
      "album": "Native",
    },
     ...
  ]
}

export default data;
```

c3.2 UIManager.jsx

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

```jsx
// app/Manager.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;
```

```jsx
// app/index.jsx

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

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

c3.3 Header.jsx

- Create a new file **app/Header.jsx**
  - Copy from **RNW files** Header.jsx
- Import and render **Header** component inside **UIManager** component
  - *** Notice that we have to enclose return JSX in a ***

```
// app/UIManager.jsx

...
        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**

c3.4 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
        ...
```

State

What and why?

State is where data lives. Any change of data in state results to a re-render. This is the reason why we use React.

How?

- Include ***constructor()*** and define *this.state* inside
  - State is initialized when an instance of this component is created
- To read data, use ***this.state.<obj>***
- Only the component has access to its state, but it can pass **props** downwards for child components to render or modify the data

Props

What and why?

Props is a data object passed from a parent component to a child component.

How?

- 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 and Props analogy
- Parent and Child

Codesandbox: **state_props_analogy_1**

https://codesandbox.io/s/xp7zv30934

Please fork before editing

Component_tree:
Parent
    Child

Props passing:
Parent → $100 → Child

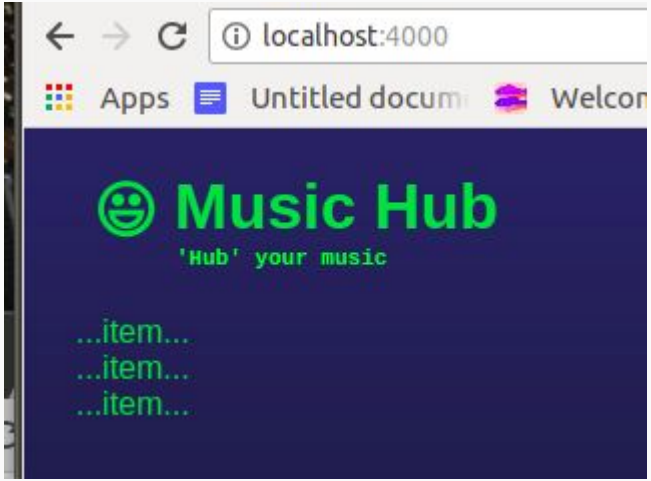
c3.5 Add state to UIManager, pass list object as props to List

```
// app/UIManager.jsx
...
class UIManager extends React.Component {

 constructor() {
   super();
   this.state = {
     list: data.list
   };
 }

 render() {
   ...
       <List list={this.state.list} />
   ...
```

```
// app/List.jsx
...
   render() {
       var list = this.props.list;
       console.log(list);
       return(
           <div>
               <Item />
           </div>
       );
   }
...
```

c3.6 Using map() to render list

```
// app/List.jsx
...
        return(
            <div>
                {
                    list.map(function(item) {
                        return (
                            <Item
                                item={item}
                                key={item.id}
                             />
                        );
                    })
                }
            </div>
        );

...
```

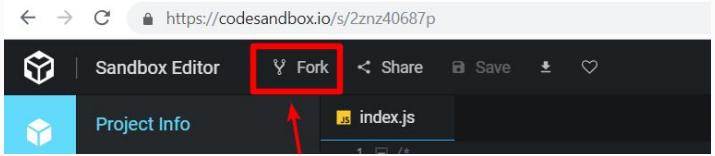


Map, filter, reduce

Codesandbox: **map_filter_reduce**

https://codesandbox.io/s/4z684jjzxw

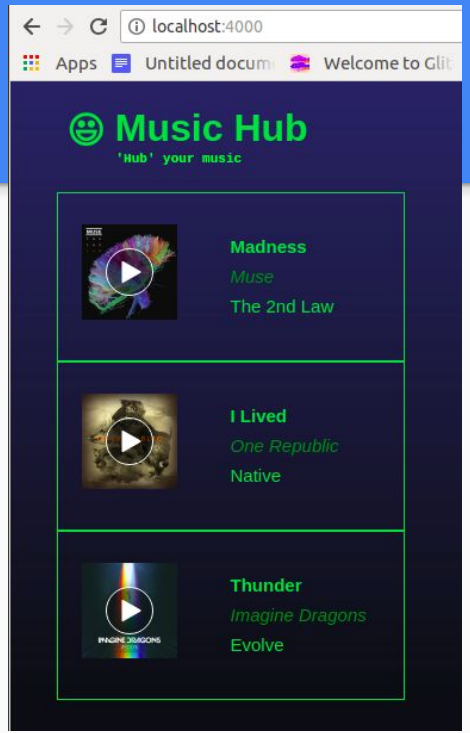
Please fork before editing



c3.7 rendering Item

*** Item can be called a **dumb / pure presentation component**
Since it's only job is to render the `item` object into JSX / HTML elements
`{item.title}, {item.artist}, {item.album}`

Dumb components can be transformed into a *stateless/functional component*



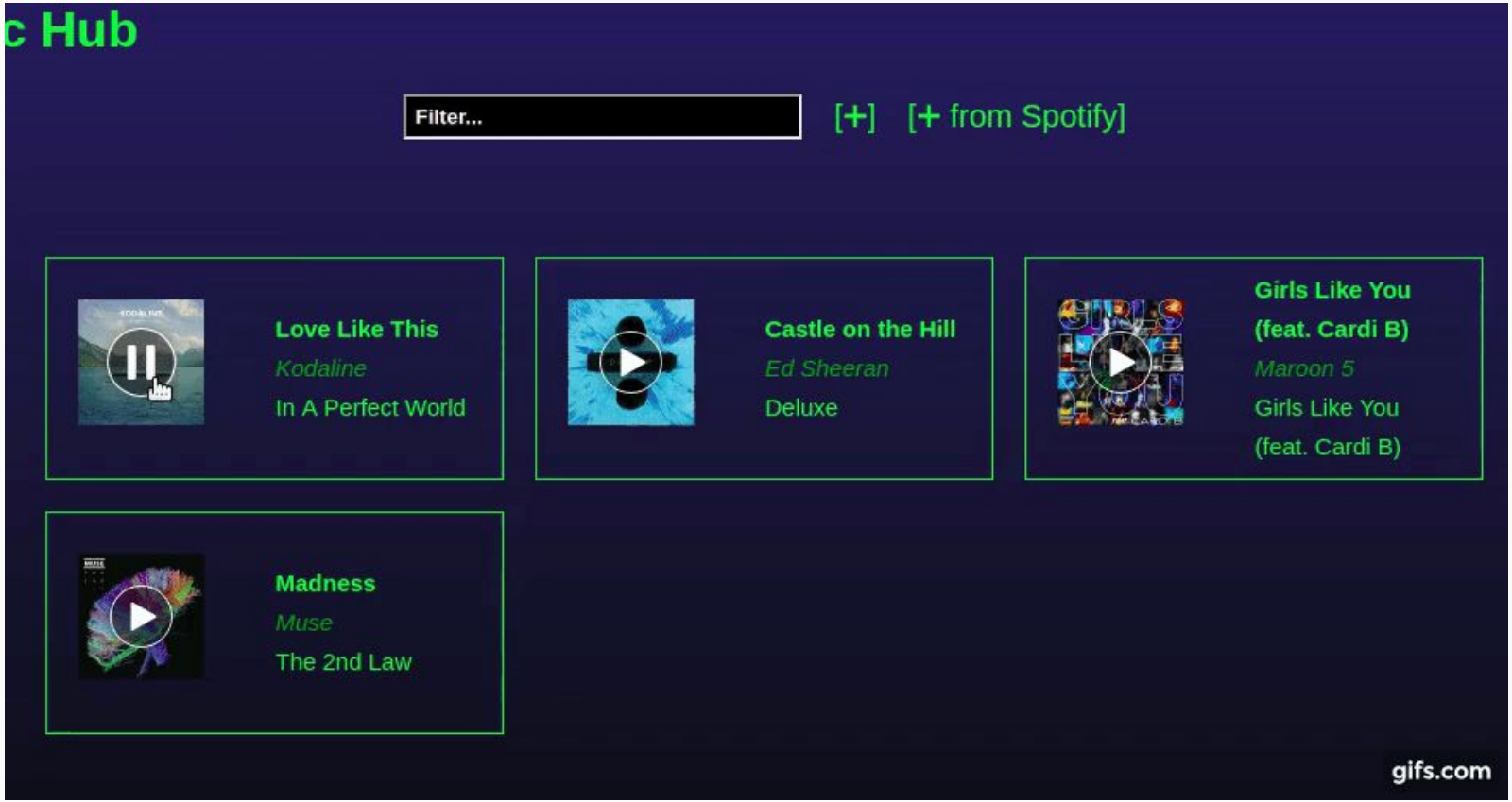
c3.8 List CSS

```
// app/List.jsx

...
        return(
            <div className="items_grid">
                {
                    list.map(function(item) {

...
```

Apply `items_grid` class to List main div

React CRUD

Searching


c Hub
Filter...
[+]
[+ from Spotify]
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
gifs.com

Events and event handlers

TL; DR - HTML elements, esp. inputs generate events, which is processed by an event handler function

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 with the event

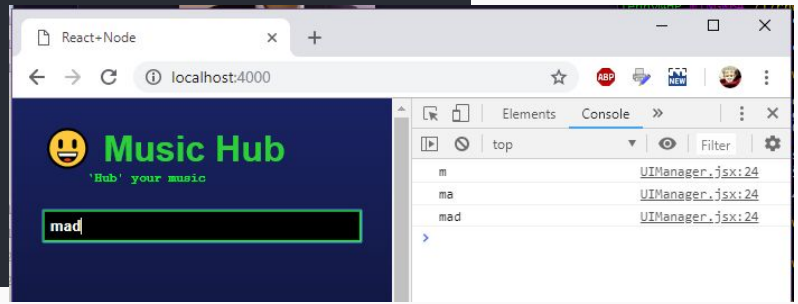
Codesandbox: **js_event_handler**

https://codesandbox.io/s/l2mw8wrj5z

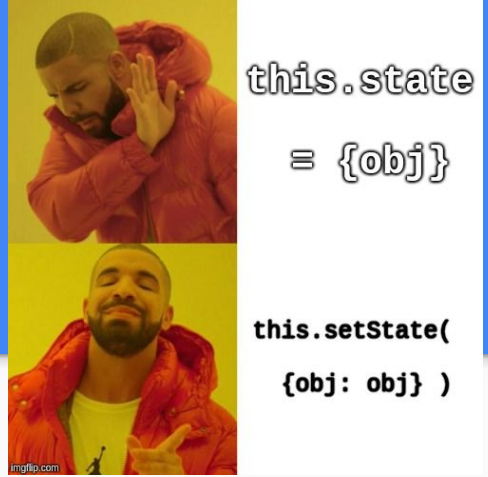
c4.1 search input and onChange event handler

```
// app/UIManager.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



c4.2 modifying state

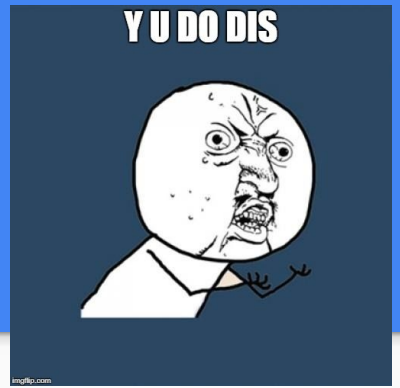


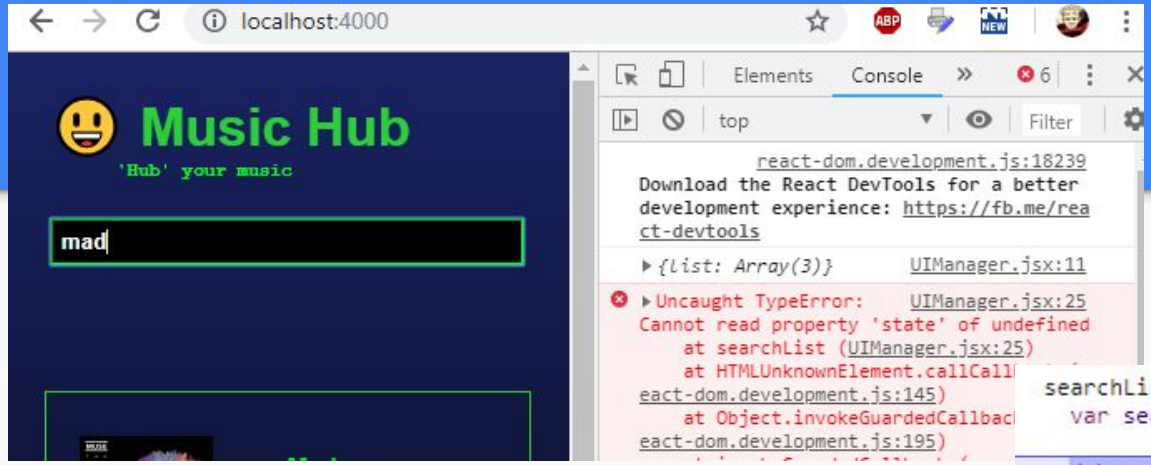
For us to filter the list based on current input:

- 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 must be async**

```
class UIManager extends React.Component {

    constructor() {
        super();
        this.state = {
            search_term: '',
            list: data.list
        }
    }

    searchList(event) {
        var search_term = event.target.value;
        // console.log(search_term);
        console.log("current search term: ", this.state.search_term);
        this.setState({
            search_term: search_term
        });
    }
```

But we're getting an error!






```
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({
```

- **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)

c4.3 using state inside event handler

```
// app/UIManager.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: **react_basics_3** https://codesandbox.io/s/3x3z1rm365

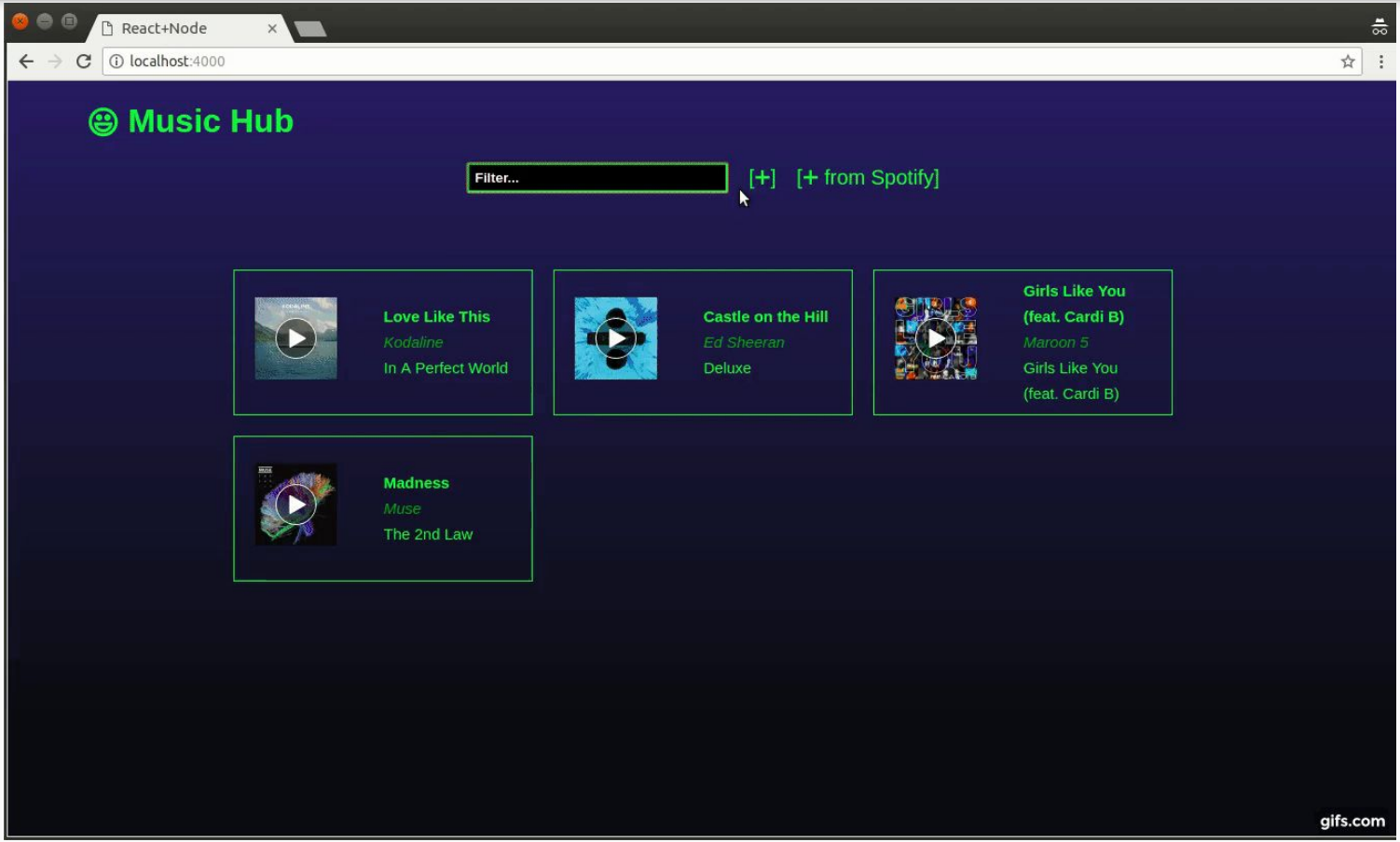
Explore the issue and solutions here

c4.4 filtering list based on *state.search_term*

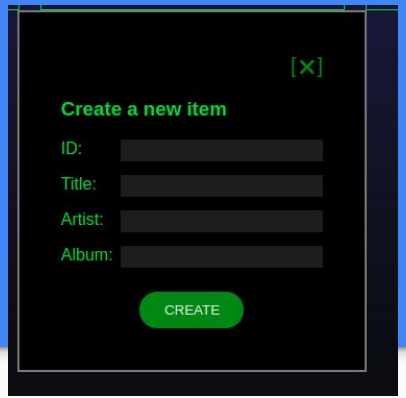
1. Create temp variables for `this.state.list`, `this.state.search_term`
2. If `search_term` empty, return full list
3. Else -> filter(): go through each item, and include in list if item's title in lowercase matches the search_term in lowercase

This is a rather expensive way of doing this,
i.e. each character typed -> full re-render of App
A nice fix is discussed later

Create / Add new Item



c4.5 ItemForm.jsx



We'll create a component that has an HTML form for creating, editing an item

- Create new file **app/ItemForm.jsx**
  - Copy from **RNW files** [ItemForm.jsx]

In **UIManager**

- add `this.state.form_fields` initializing item fields to empty string
- Import and render **ItemForm** same level as **List**
  - And pass `this.state.form_fields` as props

Function props

Remember that **Child** component cannot modify **Parent** state directly. However, **Parent** can pass a function to **Child** that it can call whenever it wants to change Parent's state. e.g:

In Parent render():

```
<Child earnMoney={(money) => this.increaseMoney(money)}>
```

In Child render():

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

Codesandbox: **state_props_analogy_2**
https://codesandbox.io/s/6jormxo65n

c4.6 events and event handlers

```
<UIManager>
        <ItemForm>

Props passing:
<UIManager> ---- onChangeFormInput(event) ---> <ItemForm>
```

In **UIManager.jsx**

1. Define *onChangeFormInput*()
    a. Pass this to **ItemForm** as a **<u>function prop</u>**

ItemForm.jsx - define these event and event handlers

1. When changing any of the 4 inputs
    a. onChange → invoke this.props.*onChangeFormInput(event)* function prop, passing the event
2. When hiding the form ([X])
        a. onClick → this.hideForm
3. When submitting form (CREATE)
        a. onClick → *this.onSubmitForm(event)*

c4.7 onChangeFormInput

Whenever user types into either ID, artist, title, album field:

1. From **ItemForm.jsx,** we are forwarding the event to **UIManager.jsx**, using function prop *onChangeFormInput*

In UIManager.jsx onChangeFormInput()

1. Copy values of *this.state.fields* (don't copy reference!) - use Object.assign
2. Modify state copy depending on which input was changed

```
// e.g. current_list_fields['artist'] = 'Artist1'
current_list_fields[event.target.name] = event.target.value;
```

3. Apply changes to state using *this.setState*

Now we are successfully tracking the input values in state

c4.8 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 operator
    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.form_fields }
          onChangeFormInput={(event) => this.onChangeFormInput(event) }
          createItem={this.createItem} />
```

In **ItemForm**, on ***onSubmitForm()*** event handler

1. Forward request to function props

c4.9 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’

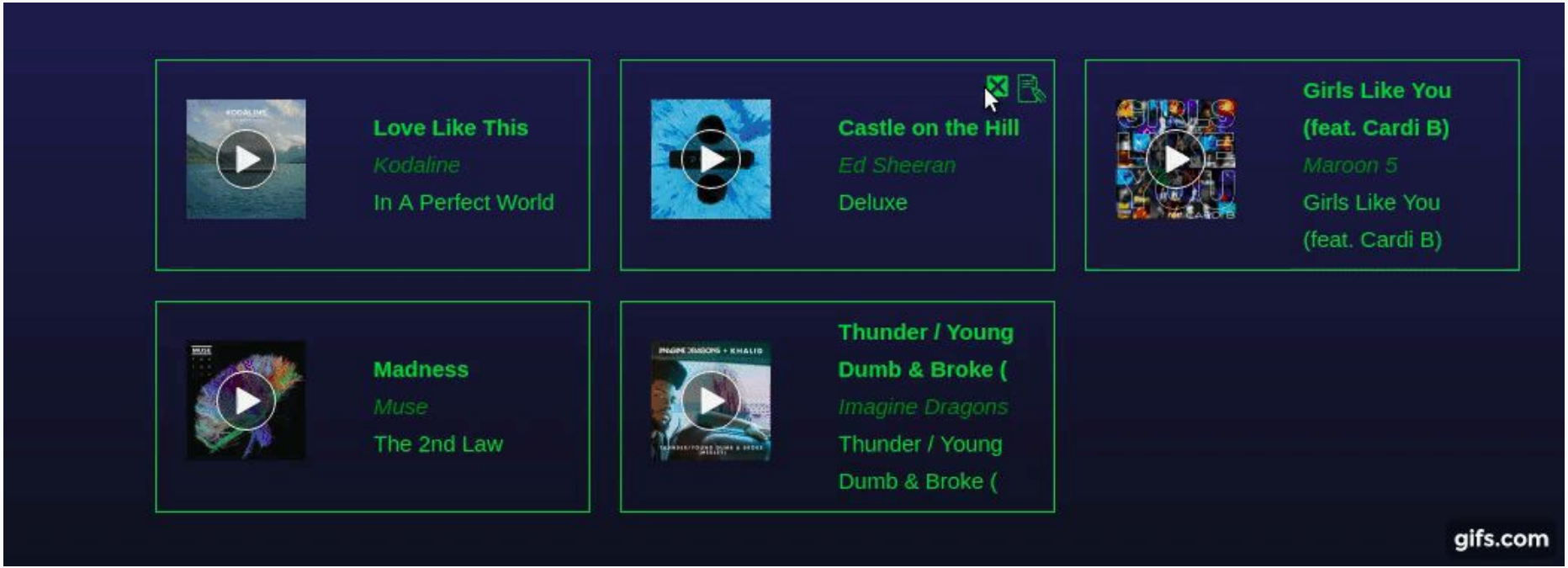
Optional: invoke after adding a new item

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”
- Get the ID in `track/ID`
- Use ID to Create new Item

Delete and Update item



c4.11 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**

c4.12 deleteItem() method

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

c4.13 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, alias **mode**
  - This is to show props is just a name

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

ItemForm.jsx

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

c4.14 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

c4.15 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`

c4.16 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!

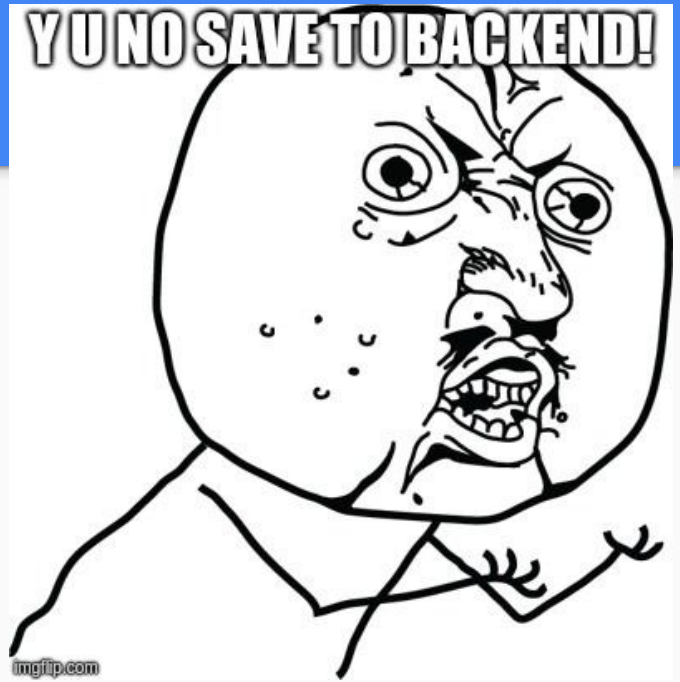
Create, Edit, Delete, Filter (Search)

Questions so far??

Recap?

Break?

But… the changes do not apply to data.js?



In frontend, we were able to manipulate the in-memory data

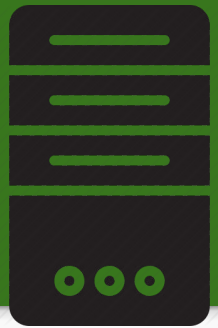
- **But when we reload the page, the changes are gone**

We will persist the changes in a data file using Node in the backend

- NOTE: Frontend read/write of local data files is usually not the way to go, because of security reasons

Node CRUD

c5 Overview of REST API



- Notice that the CRUD operations are working, but changes not applied to the **data.json** file.

- Instead of applying the changes to in-memory state in frontend, we need to persist the changes to backend, through a REST API.

- We need backend to read/write into files, databases, and external APIs
- First, we would move data file in the backend, which would be the data source of our HTTP REST API
    - Read **data.json** using **fs**

c5.1 Reading file from server-side

Directory structure after:

```
react_node_workshop/
  ...
  server/
     data.json
     utils.js
  ...
```

- Make new directory **server/** in application root
- Move data from ***app/data.js*** *(frontend)* to ***server/data.json*** *(backend)*
- Create a new file **server/utils.js**
  - This will be used for file-reading, etc
- Define readJSON() in **utils.js** using fs.readFile
- In **server.js**, enclose server functionality inside function **runServer()**, then call readJSON() and pass callback

⚠️ **UI will break, but no worries 💩. We'll fix it as we go**

c5.2 READ routes

Establish routes for fetching data

- Fetch list - gets all list items
  - `server.get("/list", function(req, res))`
- Fetch one
  - `server.get("/list/:id", function(req, res))`

```
// fetch all
server.get("/list", function(req, res) {
    res.send(json_data['list']);
});

// fetch one
server.get("/list/:id", function(req, res) {
    console.log(`GET Item ID ${req.params.id}`);
```

localhost:4000/list


```
[
  {
    "id": "0c4IEciLCDdXEhhKxj4ThA",
    "artist": "Muse",
    "title": "Madness",
    "album": "The 2nd Law"
  },
  {
    "id": "2QAHN4C4M8D8E8eiQvQW6a",
    "artist": "One Republic",
    "title": "I Lived",
    "album": "Native"
  },
  {
    "id": "5VnDkUNyX6u5Sk0yZiP8XB",
    "artist": "Imagine Dragons",
    "title": "Thunder",
    "album": "Evolve"
  }
]
```


localhost:4000/list/0c4IEciLCDdXEhhKxj4ThA


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


```
GET Item ID 0c4IEciLCDdXEhhKxj4ThA
```

Promises

I promise there will be stuff here...

...

... (after some time)

...

... here it is ---------->

```
function fetchData() {
 return new Promise(function(resolve, reject) {
   // do something that takes indeterminate time (aka
asynchronous) resulting to either error or data
   if (error) {
       reject(error);
   } else {
       resolve(data);
   }
 });
}

//ES5
fetchData().then(function(result){
   doSomething(result);
}).catch(function(error) {
   throw error;
});

// ES6
fetchData().then(result => {
   doSomething(result);
}).catch(error => {
   throw error;
});
```

c5.3 Updating frontend to fetch list from backend

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

UIManager.jsx

- Install and import `axios`
- Fetch data from backend before page is rendered. How?
  - `componentDidMount()` - React lifecycle method that runs after component mounted; good place to put network fetch requests

componentWillMount()

- `Axios.get('/list').then()` is executed when async request comes back (**promise is resolved**)
- Note the use of arrow function here since this.setState needs access to real **this** object
- 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

componentDidMount, and other React lifecycle methods

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

STUFF

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()`.

Understanding HTTP requests, URL params and Request Body

Sample commands:

```
$ curl https://jsonplaceholder.typicode.com/posts/1
$ curl https://jsonplaceholder.typicode.com/posts
$ curl -X POST -H "Content-Type: application/json" --data '{"title": "foo", "body":
"bar", "userId": 1}' https://jsonplaceholder.typicode.com/posts/
$ curl -X PUT -H "Content-Type: application/json" --data '{"title": "foo", "body":
"bar", "userId": 2}' https://jsonplaceholder.typicode.com/posts/1
$ curl -X DELETE https://jsonplaceholder.typicode.com/posts/1
```

Sample requests cheat sheet: **RNW files** http.md

Designing routes with HTTP methods

Designing: what are the methods and parameters we need for REST API

Operation	Method	URL	URL params	Request body	example
Create	POST	/list		body: {id, title, artist, album}	POST /list body: {...song details}
Read one	GET	/list/:id	:id (item ID)		GET /list/12345
Read all	GET	/list			GET /list
Update	PUT	/list/:id	:id (item ID)	body: {id, title, artist, album}	PUT /list/12345 body: {...song details}
Delete	DELETE	/list/:id	:id (item ID)		DELETE /list/12345

c5.4 Adding CRUD routes in backend

```
# for parsing request body
$ npm install body-parser
```

server.js

- Install and use body-parser
- Implement following routes:
  - CREATE: server.post("/list")
  - UPDATE: server.put("/list/:id")
  - DELETE: server.delete("list/:id")

We will use sample responses, since we will tackle file writing afterwards

Sample requests cheat sheet: **RNW files** http.md

```
// server.js
...
var path = require ('path');
var body_parser = require('body-parser');
...
 server.use(express.static('public'));
 server.use(body_parser.json());
 server.use(body_parser.urlencoded({ extended: true }));
```

Testing the backend routes using cURL

Client commands and results

```
lenny@hp:~/rnw$ curl http://localhost:4000/list/
[{"id":"0c4IEciLCDdXEhhKxj4ThA","artist":"Muse","title":"Madness","album":"The 2nd Law"},{"id":"2QAHN4C4M8D8E8eiQvQW6a","artist":"One
lic","title":"I Lived","album":"Native"},{"id":"5VnDkUNyX6u5Sk0yZiP8XB","artist":"Imagine Dragons","title":"Thunder","album":"Evolve"
y@hp:~/rnw$ curl http://localhost:4000/list/0c4IEciLCDdXEhhKxj4ThA
{"id":"0c4IEciLCDdXEhhKxj4ThA","artist":"Muse","title":"Madness","album":"The 2nd Law"}lenny@hp:~/rnw$
lenny@hp:~/rnw$
lenny@hp:~/rnw$ curl http://localhost:4000/list/something_that_doesnt_exist
{"error":"Item with ID something_that_doesnt_exist not found"}lenny@hp:~/rnw$
lenny@hp:~/rnw$
lenny@hp:~/rnw$ curl -X POST http://localhost:4000/list -H "Content-Type: application/json" --data '{"title":"My Song", "album":"My A
 "id":"daskdal2dasdk2dasd"}'
{"created":{"title":"My Song","album":"My Album","id":"daskdal2dasdk2dasd"}}lenny@hp:~/rnw$
lenny@hp:~/rnw$
lenny@hp:~/rnw$
lenny@hp:~/rnw$ curl -X PUT http://localhost:4000/list/dasdsad123da -H "Content-Type: application/json" --data '{"title":"My Song", "
:"My Album", "id":"daskdal2dasdk2dasd"}'
{"updated":{"title":"My Song","album":"My Album","id":"daskdal2dasdk2dasd"}}lenny@hp:~/rnw$
lenny@hp:~/rnw$
lenny@hp:~/rnw$ curl -X DELETE http://localhost:4000/list/dasdsad123da
{"deleted":"dasdsad123da"}lenny@hp:~/rnw$
lenny@hp:~/rnw$
```

Backend logs

```
GET Item ID 0c4IEciLCDdXEhhKxj4ThA
GET Item ID something_that_doesnt_exist
GET Item ID something_that_doesnt_exist
Create item with details: {"title":"My Song","album":"My Album","id":"daskdal2dasdk2dasd"}
Edit item with id: dasdsad123da, change to {"title":"My Song","album":"My Album","id":"daskdal2dasdk2dasd"}
Delete item with id: dasdsad123da
```

c5.5 Apply CRUD operations to data.json

- **server/utils.js**
  - Define ***writeJSON()*** **-** callback passes json_data written to file
- **Server.js**
  - Define **writeToFileAndSendResponse()**
    - Invoke *writeJSON()* passing updated list to write to file, and a callback that takes the result of write
    - Callback then sends the results to client as a response

c5.6 Implement CRUD operations

- **Server.js**
  - All Create, Update, and Delete have similar algorithms to the one in frontend (C4 React) so we can apply the logic here, replacing UI state operations with file operations
- **TESTING**
  - Use same commands as before: **RNW files** http.md
  - But this time, changes must be reflected in the **data.json**

c5.7 Apply changes to front-end

UIManager.jsx

- For each Create, Update, Delete in frontend, call corresponding backend route.
  - `axios.post(url, {json_obj})`
  - `axios.delete(url)`
  - `axios.put(url, {json_obj})`
- Effectively, we can also remove some CRUD logic in the frontend, since we moved all of these to backend. However, we still need to setState, etc. We must do setState inside callback of axios calls. **Why?**

TESTING

- App should work exactly as before, but changes are persisted now in `data.json`

Node CRUD + MongoDB

noSQL databases

```
// a document record sample in mLab

{
    "_id": {
        "$oid": "5bafebc0c5fcaf59fed82980"
    },
    "id": "5VnDkUNyX6u5Sk0yZiP8XB",
    "title": "Thunder",
    "artist": "Imagine Dragons",
    "album": "Evolve"
}
```

What and why?

- Document-based (e.g. MongoDB) database
- Represents programming objects closer than relational databases
- Speed and flexibility (think of them as O(1) hash maps/ JSON)
- Why not? Not good for ACID compliant purposes, joins

Setting up mLab and MongoDB

mLab is the quickest way to setup a MongoDB instance
Free up to 0.5 GB

For this workshop:

- we'll use my db instance, with an initial user and some data
  I'll send the credentials through Slack
- Install mongodb to our project

```
# install mongodb to our project
$ npm install mongodb
```

** If you want to setup your own:

https://docs.mlab.com/

Test DB connection

***** This only works if you installed mongodb in your local machine as defined here**

https://docs.mongodb.com/manual/administration/install-community/

```
# --- mongoDB CLIENT ----
# on another terminal, connect to db

# Linux
$ mongo <mongodb_connection_link>
# macOS
$ mongo --host <mongodb_connection_link>
# Windows
$ "C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"


# use db and start running mongodb commands
> use spot_db

> db.items.find()
# should see some JSON-like objects here with song details
```

Optional: if mLab fails - MongoDB setup

***** This only works if you installed mongodb in your local machine as defined here https://docs.mongodb.com/manual/administration/install-community/***

1. On another terminal, start local mongodb server on your machine
   - Ubuntu: `sudo service mongod start`
   - Mac: `mongod`
   - Windows: *"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"*
2. On another terminal, start mongo client as described in previous slide

```
# after starting local mongodb server and
# after connecting to server using a mongodb client

# use db and start running mongodb commands
> use spot_db

> db.items.insertOne( { id:"some_song_id", artist:"The Artist", "title": "Song
song", "album": "The Album" } )

> db.items.find()
# should see fake song we just added
```

c6.1 Server code for mongodb

1. Create a new file ***server/mongo_db.js*** for mongodb connection
2. **server/mongo_db.js**
   - Define **init_db()** - this function returns a **Promise**, which will
     i. **resolve -** if connection successful, resolve value is a db instance caught by **.then()** of calling function
     ii. **reject** - if connection error, reject value is error and must be caught by calling function in a **.catch()**
   - Export file using **module.exports**
3. **server.js**
   - Import and use mongodb.MongoCleint
   - Import **server/mongo_db.js** file
   - Replace file read call with **mongo_db.init()** and define the **then()** and **catch()** functions
   - *** If using local mongodb, use the *localhost* version of *db_connection_url*

⚠️ **App will break, but no worries 💩. We'll fix it as we go**

c6.2 db_collection.find()

FRONTEND IS BROKEN! (`/list` route not returning anything yet)
To fix frontend, our first DB operation is to fetch all data on first load of app, which is in the route `server.get("/list")`

server.js
Use *db.collection.find()* to fetch all

NOTE that result is an array
This data structure must match the Expectations of **UIManager.componentWillMount()**

```
// fetch all
server.get("/list", function(req, res) {
  // res.send(json_data['list']);
  db_collection.find().toArray(function(err, result) {
    if (err) throw err;
    res.send(result);
  });
});
```

✓ Route `http://localhost:4000/list/` should now return all items from mLab

MongoDB operations

Operation	Method	URL	mongoDB method
Create	POST	/list	collection.**insertOne**(obj, callback)
Read one	GET	/list/:id	collection.**findOne**(query, callback)
Read all	GET	/list	collection.**find**(query).toArray(callback)
Update	PUT	/list/:id	collection.**updateOne**(query, { $set: obj }, callback)
Delete	DELETE	/list/:id	collection.**deleteOne**(query, callback)

Examples:

```
obj
{
 id: "blah20",
 artist: "Artist",
 title: "Title",
 album: "Album"
}

query
{ id: "blah20" }

callback
function (err, result) {
  // process result
}
```

c6.3 Move all File CRUD calls to DB version

server.js
For simplicity, we send entire list after each operation, to maintain consistency of data between backend and frontend

UIManager.jsx
Update all *this.setState()* to match the data returned

- all instances of **response.data.list** to **response.data**
- *onChangeFormInput()* must get all item fields except _id (we can't change since this is used internally by mongo)

TESTING

- App should work exactly as before, but changes are persisted now in the DB
  - Verify that UI list is synced up with mLab items

Integrating with Spotify API

Spotify search API

Intro to API Oauth2

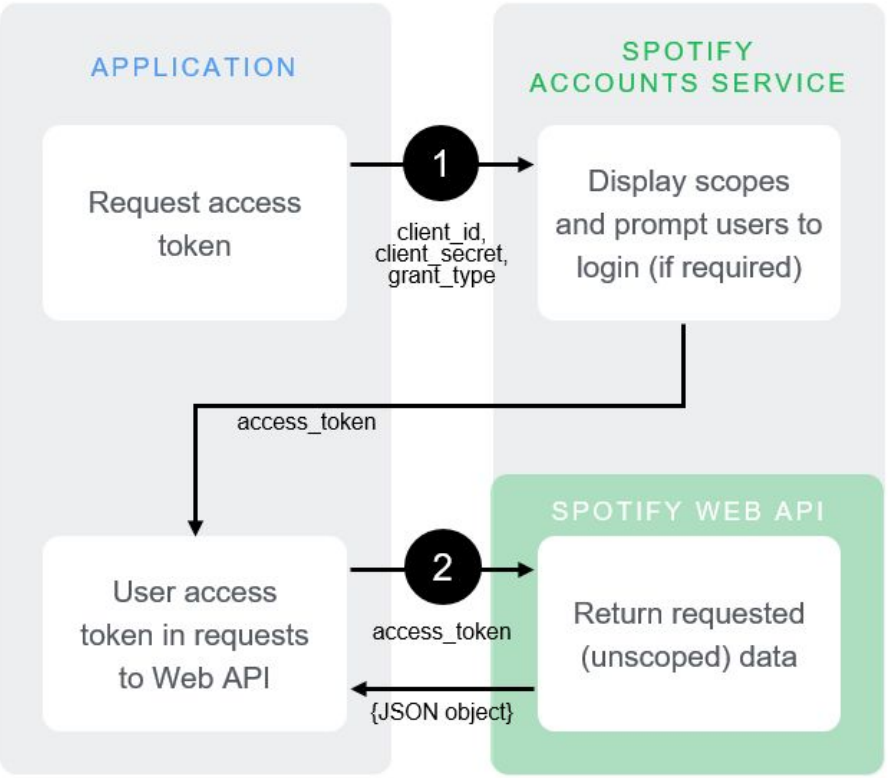
Spotify uses **OAuth2**, which requires an application to send an initial “access token request”, containing encoded user credentials (Spotify dev account). The access token (aka “bearer” access token) is good for a period of time.

The app can then attach the “bearer” access token for each request (e.g search, get album, artist, track), to prove its identity to the Spotify server, without sending credentials.

Setting up Spotify API

1. For simplicity, we will use the free Spotify developer account I setup before
   a. You could also setup your own developer account https://developer.spotify.com/dashboard

2. Authentication flow is using Client Credentials https://developer.spotify.com/documentation/general/guides/authorization-guide/#client-credentials-flow



* For those using my credentials, I will send the Base-64 string (composed of Client ID, Client secret) needed for authentication **through SLACK**

Testing out spotify credentials

Test in Terminal / Postman

```
$ curl -X "POST" -H "Authorization: Basic <base64_string>" -d
grant_type=client_credentials https://accounts.spotify.com/api/token
> {"access_token":".....","token_type":"Bearer","expires_in":3600,"scope":""}
```

We then use this access_token to do requests like

```
$ curl -H "Authorization: Bearer <access_token>"
"https://api.spotify.com/v1/search?query=gangnam%20style&type=track"

> { literally tons of data }
```

Our webapp server's job:

1. Prepare the **request**, including **encoding query params**
2. Process the **response**, which includes **filtering, preparing data for render in React**

c7.1 setup spotify API backend

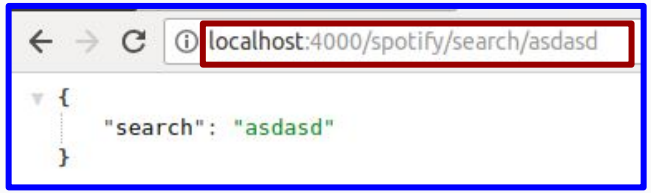
```
# qs allow proper data encoding for HTTP
$ npm install qs
```

It's better to separate API specific code, which we import in our server file.
Create folder **api** and file **api/spotify.js**

api/spotify.js

Define **spotify_routes()** - this will contain all spotify -specific routes we need

Serve a GET route `'/spotify/search/:query'`



server.js

- Import axios, qs
- Import and use ***api/spotify.js***
  - We have to pass server and db_connection object

c7.2 requesting for an access_token

spotify.js

Define **getAccessToken()** which returns a Promise

- Prepare request config containing the base-64 encoded credentials in the header
- Send request and resolve Promise with *access_token* returned by Spotify

Invoke inside route `'/spotify/search/:query'`

- Define .then(), send access_token for now to test


```
Starting server at 4000
[SPOTIFY] : searching asdasd...
[SPOTIFY] Requesting a new access token...
[SPOTIFY] Access token:  BQDfHsFvwB4QRwBiG2gPsOfktzTaG1DDmb2LlAuT5fyJoCO52U_X4Fbi6jevZ2eICYsYpCSeNnlSNmOsRnk
```

c7.3 GET request to search query string

We could now use the access_token to make a request

```
axios({
    method: 'GET',
    url: _url,
    headers: {
        "Authorization": `Bearer ${access_token}`,
        "Accept": "application/json"
    }
}).then(function(_res) {
    // inspect response data
    console.log(`search response: ${JSON.stringify(_res.data)}`);
    res.send(_res.data.tracks.items);
```

```
[SPOTIFY] : searching asdasd...
[SPOTIFY] Requesting a new access token...
[SPOTIFY] Access token:  BQAwgQci3TPoGMGLrljAgRLT
M--fwo
search response: {"tracks":{"href":"https://api.s
&limit=20","items":[{"album":{"album_type":"singl
en.spotify.com/artist/7rWz5hrtloVu09emujAeJh"},"h
09emujAeJh","id":"7rWz5hrtloVu09emujAeJh","name":
oVu09emujAeJh"}],"available_markets":["AD","AR","AT","AU","BE","BG","BO","BR","
```

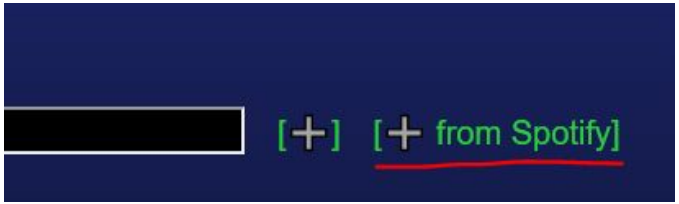
localhost:4000/spotify/search/asdasd


```
[
  {
    "album": {…}, // 13 items
    "artists": […], // 2 items
    "available_markets": […], // 65 items
    "disc_number": 1,
    "duration_ms": 8498,
    "explicit": true,
    "external_ids": {…}, // 1 item
    "external_urls": {…}, // 1 item
    "href": "https://api.spotify.com/v1/tracks/7AZmx0oEZj36SPFyC2gJ1l",
    "id": "7AZmx0oEZj36SPFyC2gJ1l",
    "is_local": false,
    "name": "qwe - Bonus Track",
    "popularity": 0,
    "preview_url": "https://p.scdn.co/mp3-preview/42144f2ccdff945472adf5c99c4fc
    "track_number": 1,
    "type": "track",
    "uri": "spotify:track:7AZmx0oEZj36SPFyC2gJ1l"
  }
]
```

c7.4 Spotify.jsx - frontend

Now that our Spotify API backend is working, we would have a Spotify component in the frontend React side that would allow us to search for Spotify tracks.
This React component utilizes our Spotify API backend.

Create new file **app/Spotify.jsx**



Spotify.jsx

Get from **RNW files** Spotify.jsx

Notice 3 event handlers that we have to implement

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

c7.5 trackSearchTerm() and searchSpotify()

Similar to how we track ItemForm fields before, we need to track search Spotify input, so when Spotify search is clicked, the search string will be in state and ready to be sent in an axios request.

Spotify.jsx - define **constructor(), trackSearchTerm()** and **searchSpotify()**

constructor() - add `this.state.search_term`, init to empty string

trackSearchTerm(event) - setState search_term to event.target.value

searchSpotify() - axios get request to our backend route `` `spotify/search/:query` ``

using `this.state.search_term` as the query

render() - add onChange event handler to search input: trackSearchTerm()

Search results in the debugger console

localhost:4000

Music Hub

'Hub' your music

[X]

search Spotify

girls like you

Search

Thunder

Imagine Dragons

Evolve

Elements Console Sources Network Performance Memory Application

Page

top

localhost:4

(index)

bundle.js

bundle.js UIManager.jsx Spotify.jsx ×

```
23   }
24
25   searchSpotify() {
26     axios__WEBPACK_IMPORTED_MODULE_1___default.a.get(`/spotify/search/${th
27       // debugger;
```

{} 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/6FRLCMO5TUHTexlWo8ym1W"}
    href: "https://api.spotify.com/v1/tracks/6FRLCMO5TUHTexlWo8ym1W"
    id: "6FRLCMO5TUHTexlWo8ym1W"
    is_local: false
    name: "Girls Like You (feat. Cardi B)"
    popularity: 94
    preview_url: null
    track_number: 1
    type: "track"
    uri: "spotify:track:6FRLCMO5TUHTexlWo8ym1W"
    __proto__: Object
```

c7.6 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**.

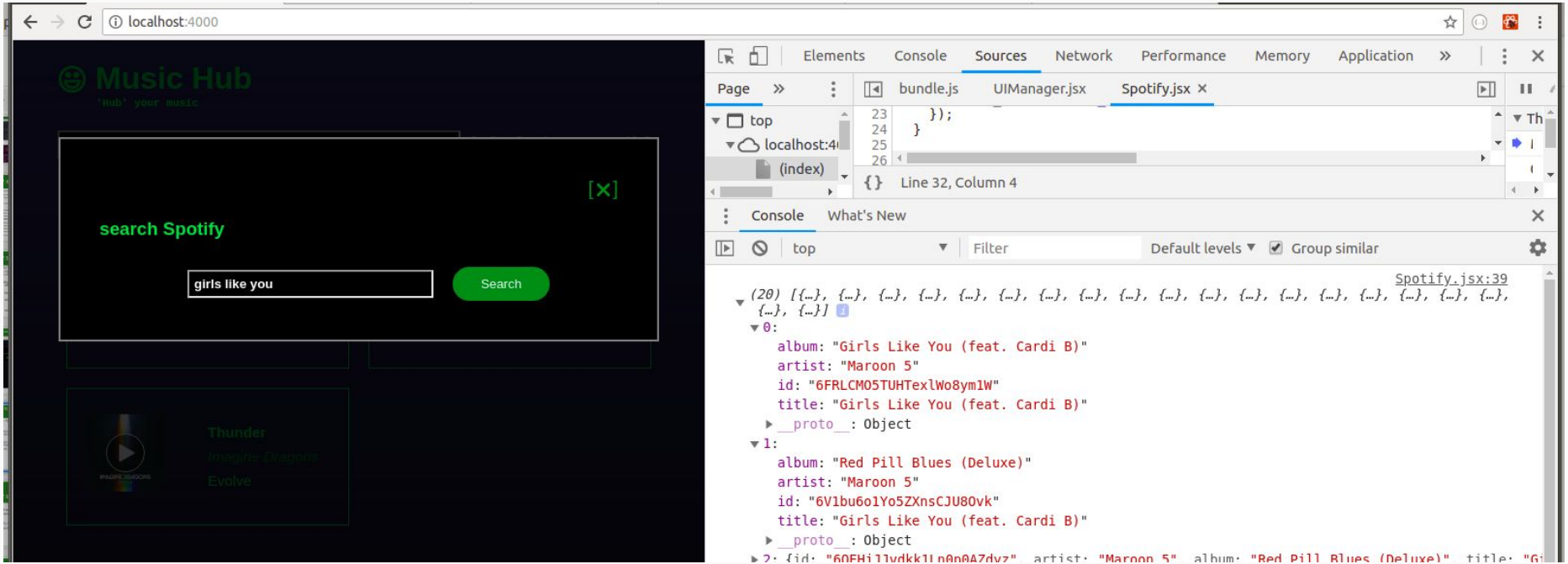
res.data array can be mapped into a new array that contains only the track attributes we need.

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

14 app/Spotify.jsx

```diff
@@ -24,7 +24,19 @@ class Spotify extends React.Component {
 24  24          axios.get(`/spotify/search/${this.state.search_term}`)
 25  25              .then((res) => {
 26  26                  // debugger;
 27  -                   console.log(res.data);
     27 +                // console.log(res.data);
     28 +
     29 +                var search_results = res.data;
     30 +                var squashed_results = search_results.map(function(track) {
     31 +                    return {
     32 +                        id: track.id,
     33 +                        artist: track.artists[0].name,
     34 +                        album: track.album.name,
     35 +                        title: track.name
     36 +                    };
     37 +                });
     38 +
     39 +                console.log(squashed_results);
 28  40              })
```

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



c7.7 We need a List and Item component...

But wait! We already have one! 😱

Hooray for reusable components! 🎉

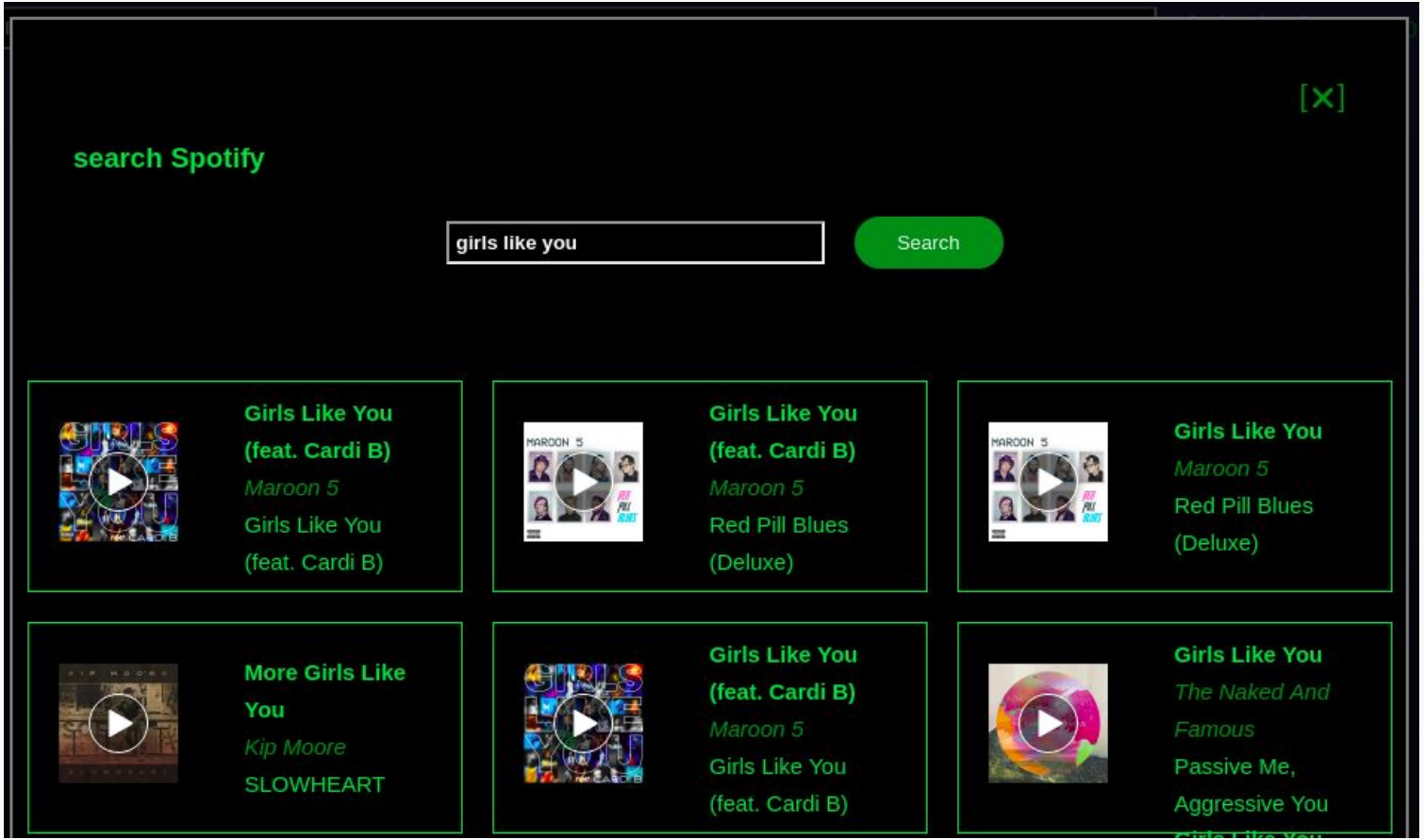
Only thing we have to do here is:

Spotify.jsx

- Import List and Item
- Add `this.state.search_results`, init to empty array
- setState `squashed_results`
- Render <List> passing `search_results` as *list* props

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

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



c7.8 [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

c7.9 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.form_fields` instead
  - I.e. the item that will be POSTed by axios (aka Create request), could be either item from Spotify's toggleItem, OR (if null), it means the call is from ItemForm and get item from `form_fields` instead

c7.10 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) ? '☒' : '+' }
    </span>
</div>
...
```

Some final testing:

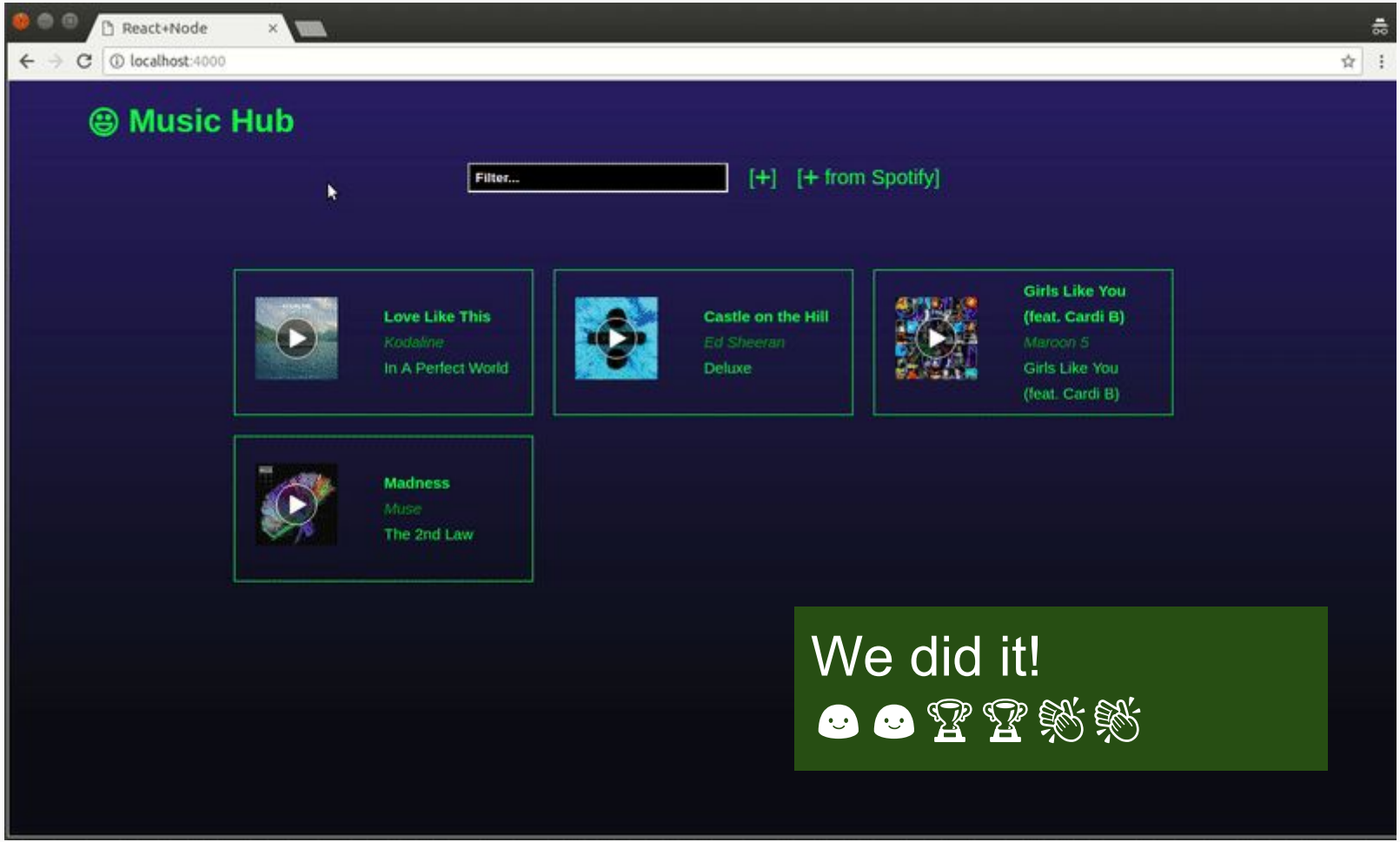
Test [+]. Here’s a few sample songs

4uLU6hMCjMI75M1A2tKUQC
7GhIk7Il098yCjg4BQjzvb
0FutrWIUM5Mg3434asiwkp

Test [+ from Spotify].

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

Note: Performance could be a bit slow, since mLab database and Spotify API depend on network. A few performance hacks are discussed in **Improvements.**


React+Node
localhost:4000
Music Hub
Filter...
[+]
[+ from Spotify]
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

We did it!

😀 😀 🏆 🏆 👏 👏

Done! 😁😁😁 🎉🎉🎉

Project at this point is at branch /**master**

Latest code updates, experiments, will be at **lenmorld/rnw/beta**

Q & A session!

Or i'll just keep talking about web 🙈

Shameless Plug!

AUTODESK®

...is hiring interns!

https://lnkd.in/dqMnkAK

We're also hiring full-time employees.

DM me in Slack if you want a list of the current positions available.

Thank you!!!