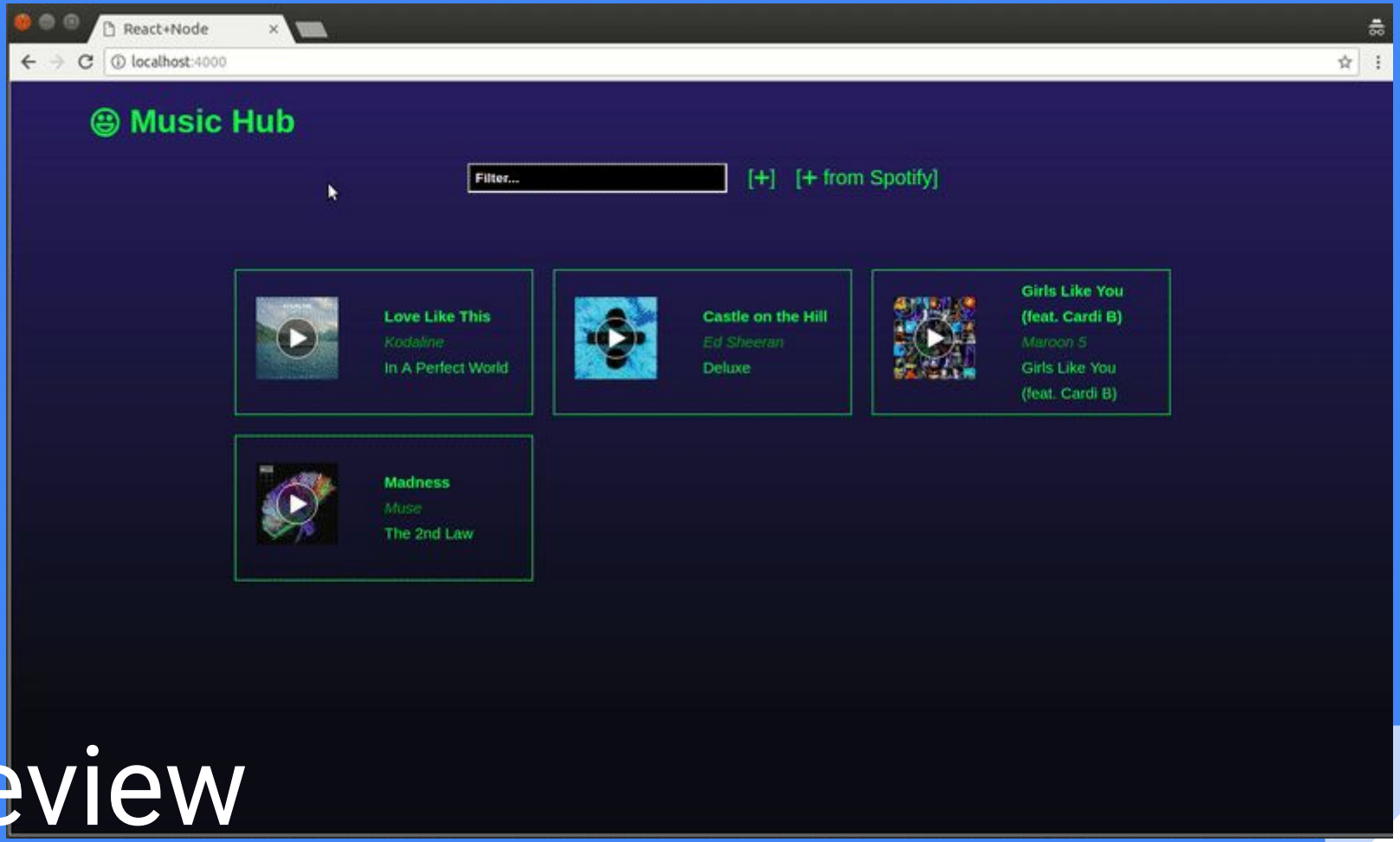
React and Node workshop

Making a full-stack web app with Spotify API

Prerequisites

Please do the following to take advantage of this workshop:

1. Join the Slack **reactnodeworkshop** https://goo.gl/buZF71
2. Install Git
3. Install Google Chrome
4. Sign up to Github (if you don't have one yet)
    - You'll need this to fork the Github Repo
5. Install node - install LTS version (8.12.x)
    - Download and install from https://nodejs.org/en/download/
    - To test: https://nodejs.org/en/docs/guides/getting-started-guide/
6. Install mongodb (we will use mLab but local DB is good to have)
    - Follow https://docs.mongodb.com/manual/installation/



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

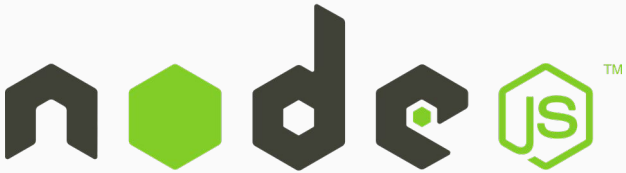
you

- **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
- Some super advanced JS stuff

Flow

- INTRO 1: Short intro on the what and whys of the tech we'll use
- INTRO 2: Some pre-req knowledge: debugger, callbacks, JS objects and arrays, client-server concepts, etc
- Chapters
  1. Node setup and basics
  2. React, webpack setup
  3. React basics
  4. React CRUD - data on frontend (memory)
  5. Node CRUD - data on backend file
  6. Node CRUD with Mongo - data on mLab/mongo DB instance
  7. Integrating with Spotify API
  8. (as time allows) QnA | node, React | JS | ES6 | code improvements

Intro 1 - Whats and whys?

JS
React
Node

Why JS ?

- Quick prototyping, visual (browser)
- Everything is becoming web-based
- Mobile app development: hybrid/native
- Web devs are in-demand!

BUT??
JS is generally a mess. So we must use best practices and frameworks

JavaScript truth table:

```
''          ==   '0'          // false
0           ==   ''           // true
0           ==   '0'          // true
false       ==   'false'      // false
false       ==   '0'          // true
false       ==   undefined    // false
false       ==   null         // false
null        ==   undefined    // true
" \t\r\n"   ==   0            // true
```

JS could be confusing though!

Non-zero value

0

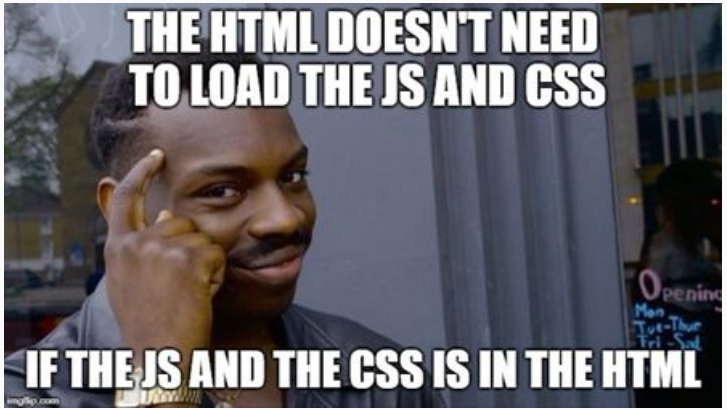
null

undefined

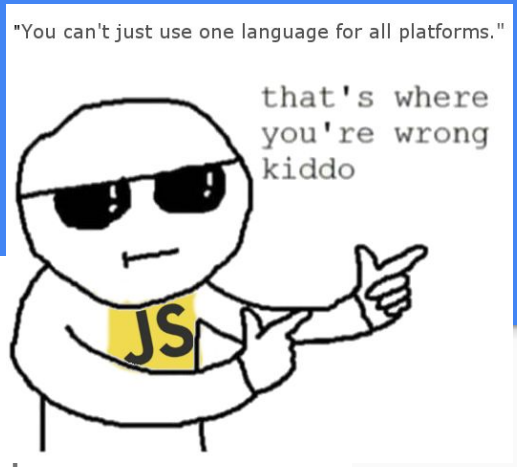
React - What and why?

React - component-based, declarative UI framework

- JS standards (JSX, ES6)
- Logic(JS) and View(HTML) glued together - less glue code
- Fast and “magical” with Virtual DOM
- More terse, less jargon to remember
- React Native: native mobile apps written in React



Node - What and why?



Node - server-side platform for developing web applications

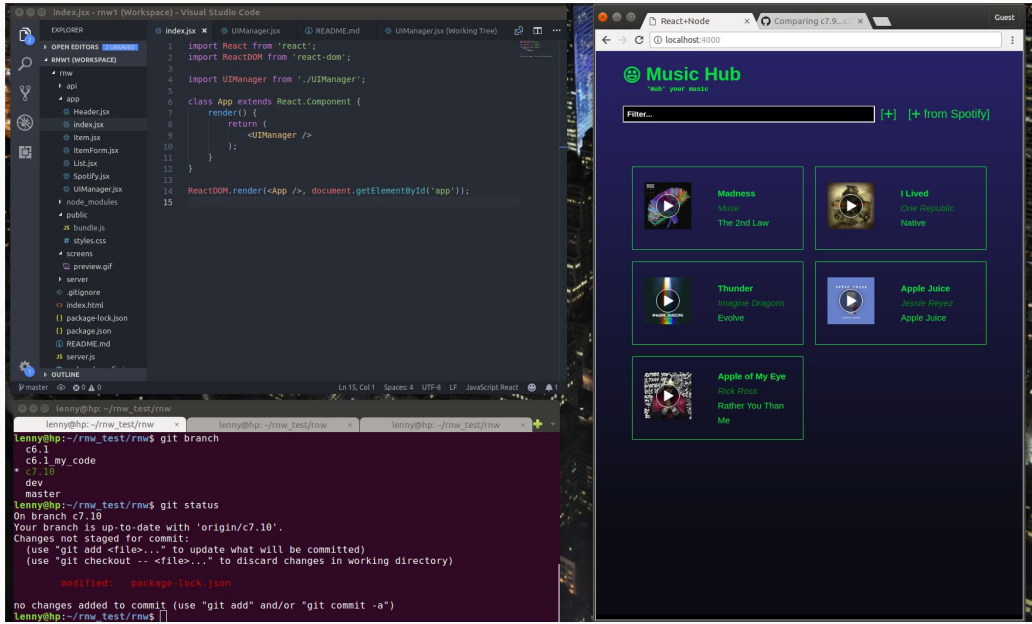
+ JS, so same language in front-end and backend
+ Non-blocking, asynchronous, event-driven, single-threaded
  + Vs Blocking architectures using threads, locks
+ Perfect for IO-intensive, real-time, streaming applications, high scalability
  - E.g. chat servers, collaborative apps, data streaming
- Not so good for CPU-intensive tasks, since single-threaded
+ Unified stack, building server from ground up
+ NPM - it’s awesome! (Yarn, too)

Intro 2 - Some super important stuff

Workspace setup

Suggested screen setup

- 1 chrome window - with multiple tabs (localhost, github, spotify search)
- 1 terminal window - with multiple tabs
- 1 code editor / IDE window



Git setup

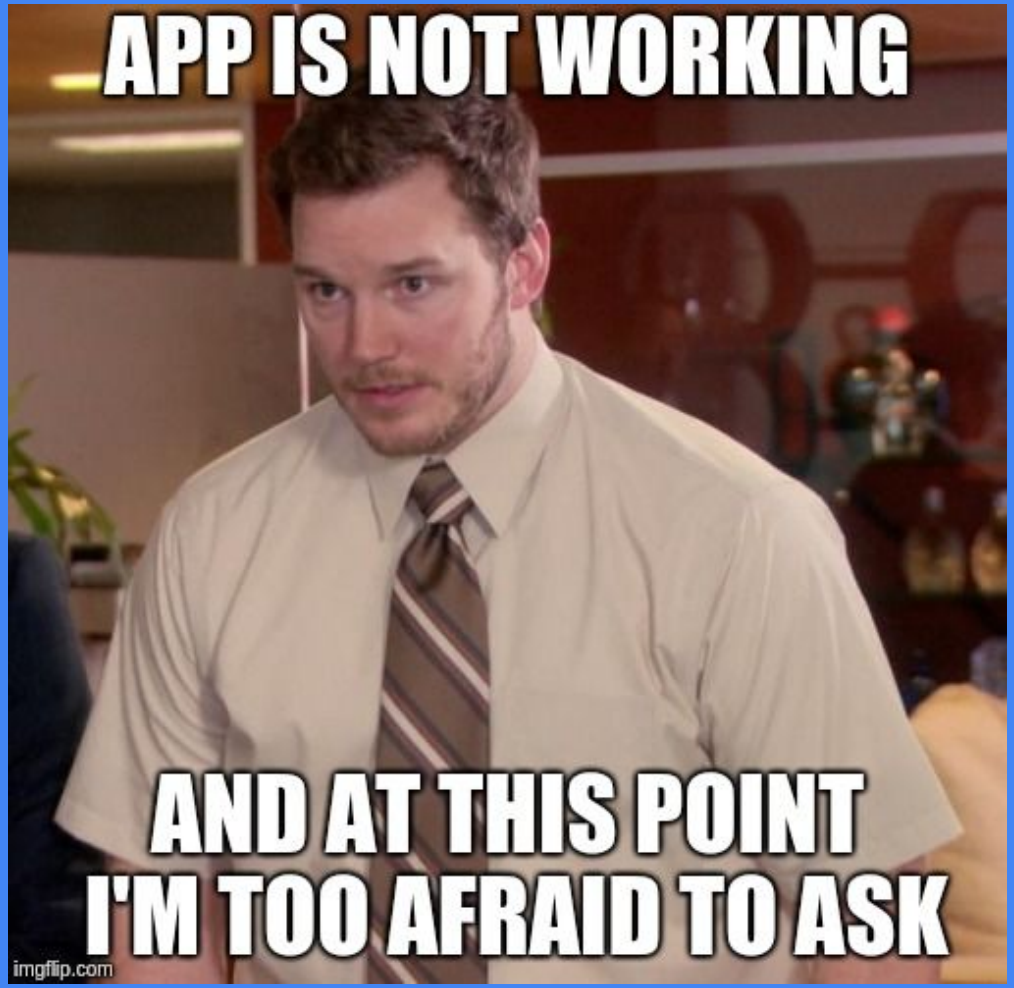
1. Log-in to github
2. Go to https://github.com/lenmorld/rnw
3. In github, fork branch

   - 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!**



Either raise it!

Or

Slack
#questions

A few reminders

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 this 

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
```

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

*** The git stuff is in the **cheat sheet** sent in Slack*

c0 - A few important topics to get started...

- Chrome Debugger and Node console
  - JSBin, Codepen, JSFiddle, etc
- Hello World in JS!
- JS Objects and arrays
- Callbacks - what and why?
- Async programming
- Clients and servers

Node Basics

c1.0 Hello World console.log()

Execute *node server.js*

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

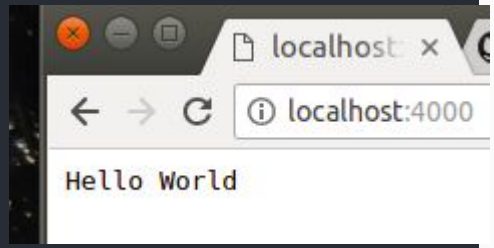
Sample directory structure:

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

c1.1 - Hello World! server

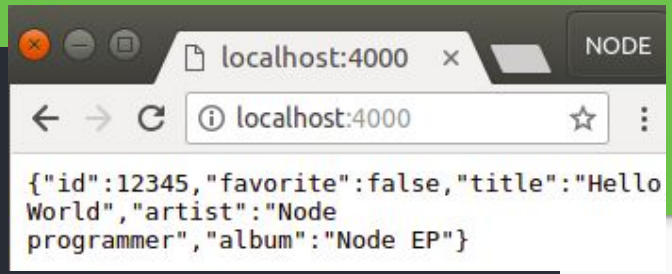
```
# to run server
$ node server.js
>> Starting server at 4000

# 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);
});
```



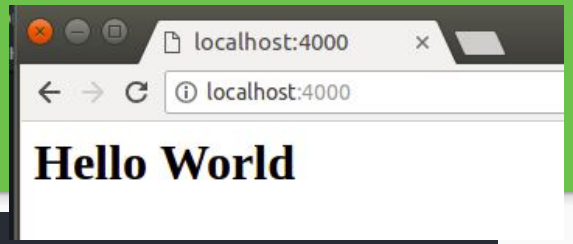
C1.2 - Hello World - JSON

```
// 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




```
// 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
```

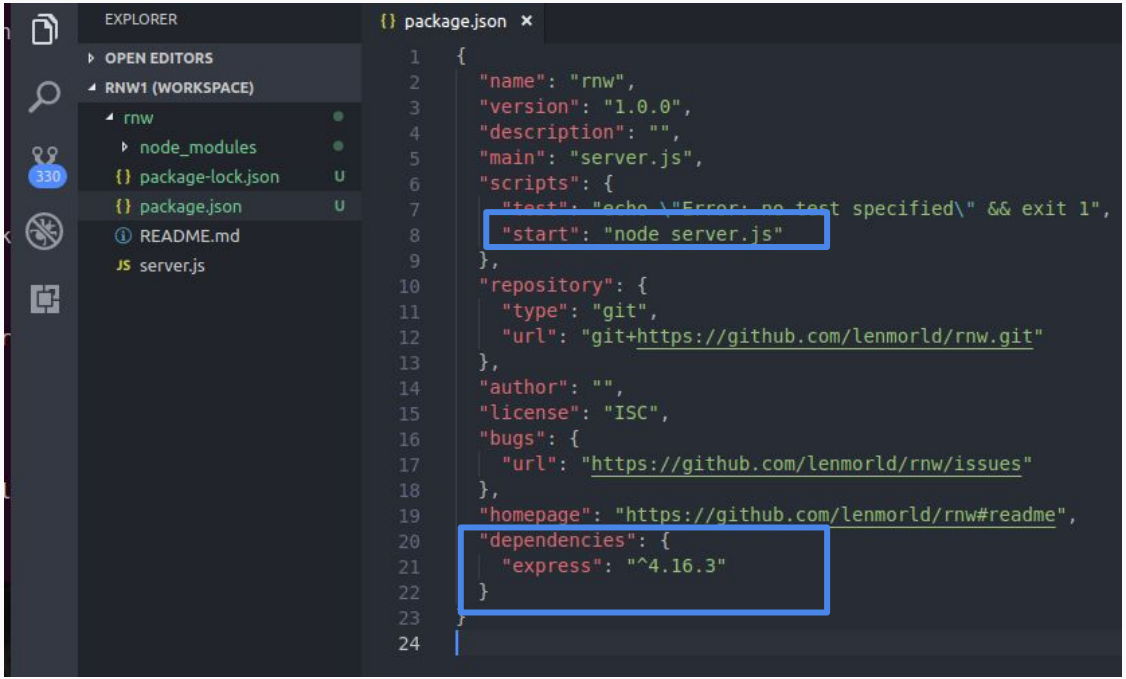
c1.4 - Using *express* middleware

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* !!!**

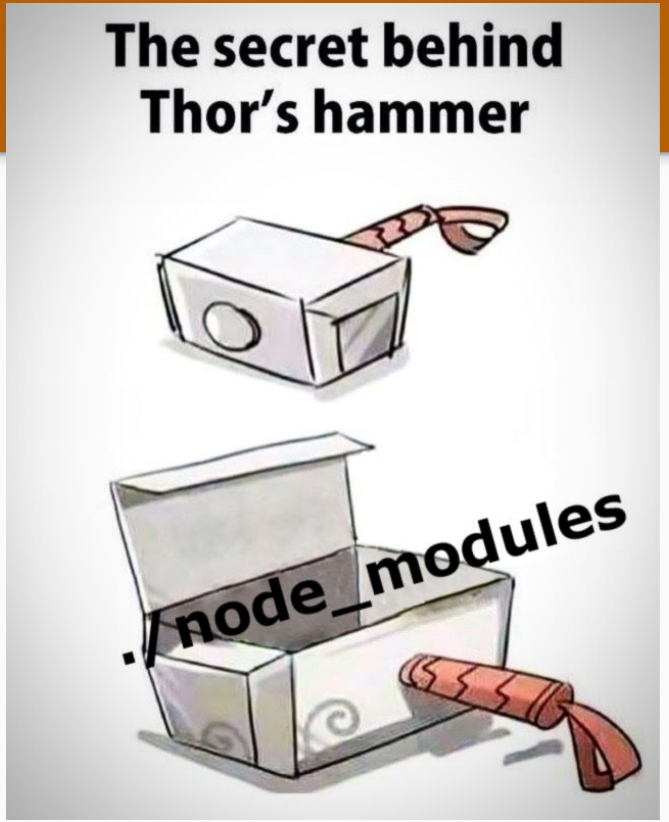
.gitignore

```
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

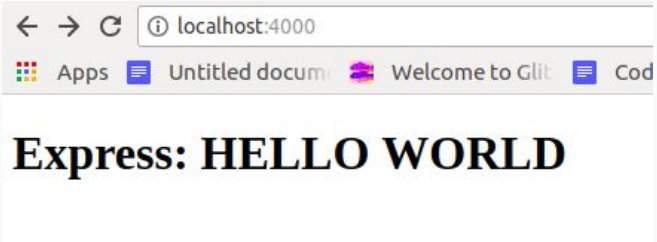
```
<!-- index.html -->

<!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:



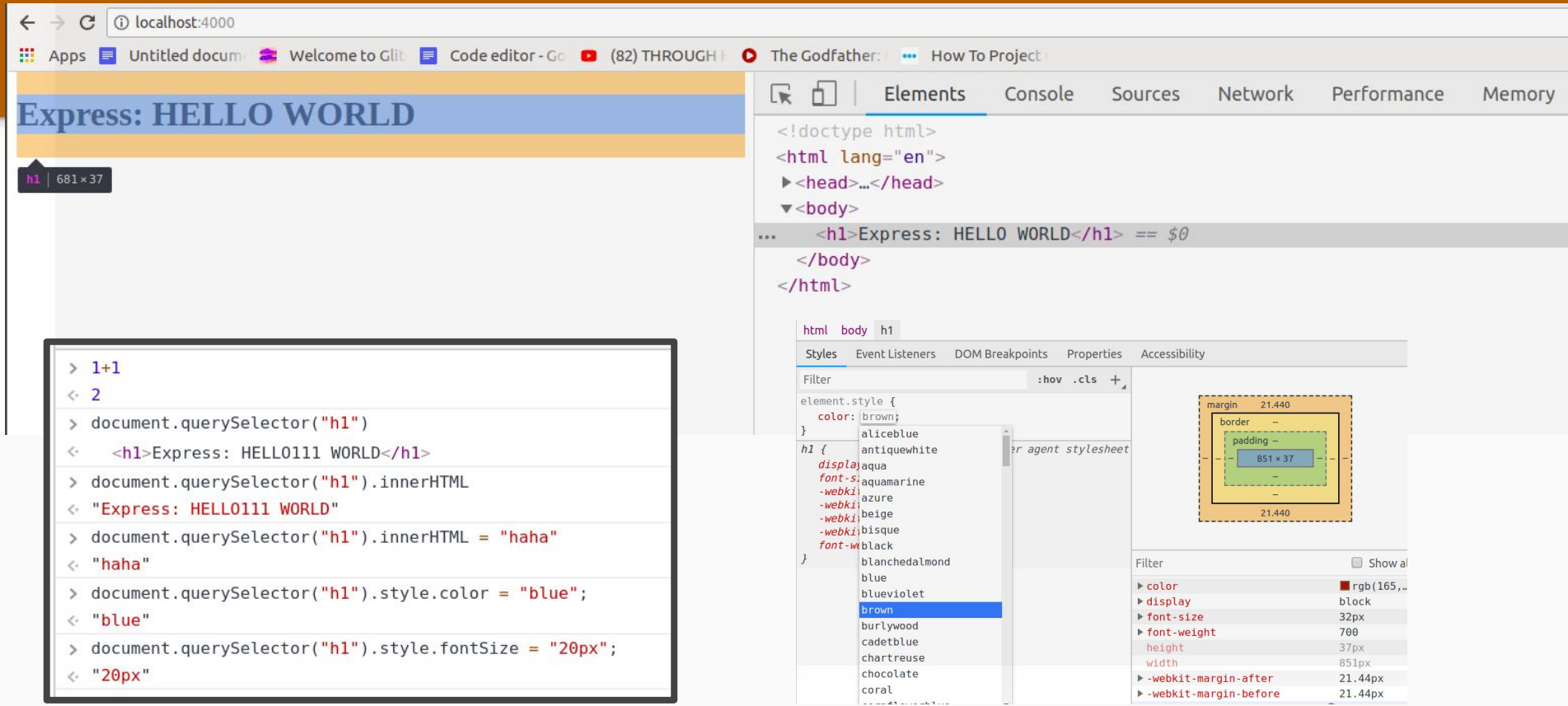

```
# in another terminal tab/window
$ curl localhost:4000/json
>> {"name":"Lenny"}
```

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"
 }
...
```

Bonus: Chrome dev tools




```
> 1+1
<- 2
> document.querySelector("h1")
<-   <h1>Express: HELLO111 WORLD</h1>
> document.querySelector("h1").innerHTML
<- "Express: HELLO111 WORLD"
> document.querySelector("h1").innerHTML = "haha"
<- "haha"
> document.querySelector("h1").style.color = "blue";
<- "blue"
> document.querySelector("h1").style.fontSize = "20px";
<- "20px"
```

React and webpack setup

c2.1 Install webpack and react

- Create file *webpack.config.js*, from template
  - Link: https://github.com/lenmorld/rnw_files/blob/master/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 react react-dom
```

- 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

$ webpack --watch
```

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) {
...
```

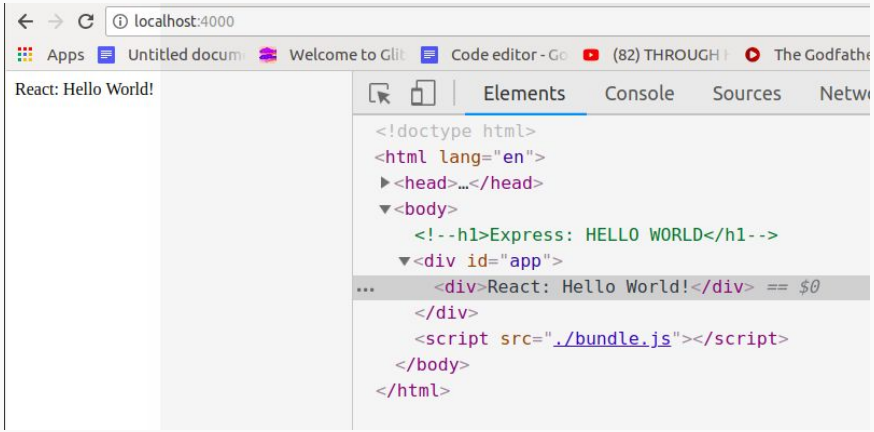
- In **server.js**, serve ***public/*** dir
- Write React code in **app/index.jsx**

```
// 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

```
<!-- 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

```
<!-- 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

Frontend Design

What are we building?

Music playlist web app, where user can “CRUD” songs

What components do we need?

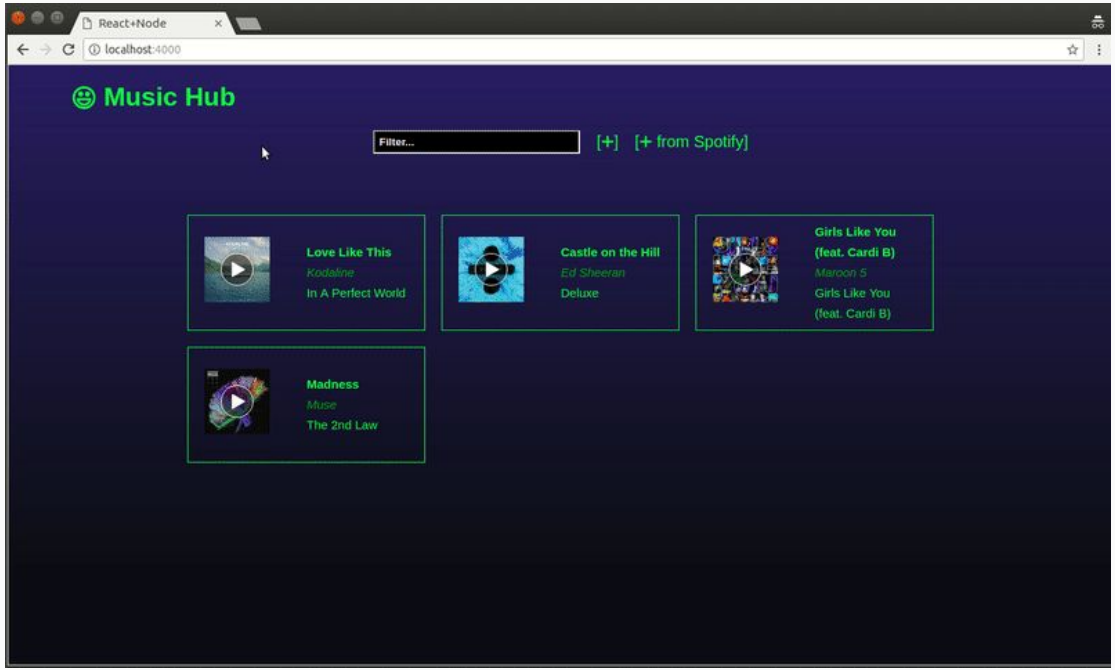
App (index.jsx) - loads React tree into DOM

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 contents from here:
  - https://github.com/lenmorld/rnw_files/blob/master/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

```
// 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;
```

```
// 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 content from here: https://github.com/lenmorld/rnw_files/blob/master/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>
      );
...
```

Good Practice: Use external stylesheets as much as possible. In react, it will save you the trouble of converting your CSS styles (e.g. font-size) into Reactified CSS (e.g. fontSize)

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
        ...
```

Github link: https://github.com/lenmorld/rnw/compare/c3.3...c3.4

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

```
class Parent extends React.Component {
    constructor() {
        super();
        this.state = {
            family_money: 300      // child doesn't have access to this!
        };
    }
    render() {
        return (
            <Child allowance={this.state.family_money / 3} />
            <Child allowance={this.state.family_money / 3} />
            <Child allowance={this.state.family_money / 3} />
        );
    }
}

class Child extends React.Component {
    render() {
        return (
            <div>I have ${this.props.allowance} today!</div>
        )
    }
}
```

Component_tree:
Parent
    Child

Props passing:
Parent → $100 → Child

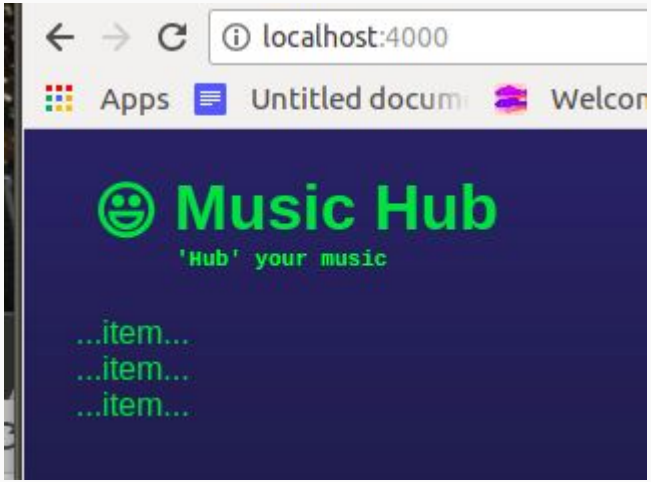
c3.5 Add state to UIManager, pass list object as props to List

```
// 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>
       );
   }
...
```

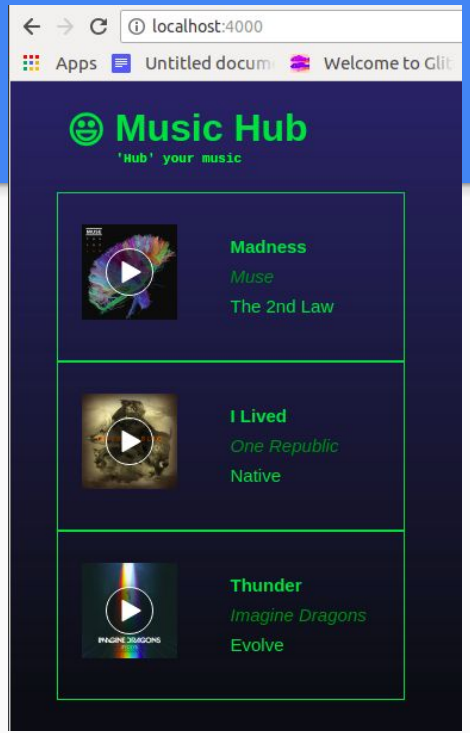
c3.6 Using map() to render list

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

...
```



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}`



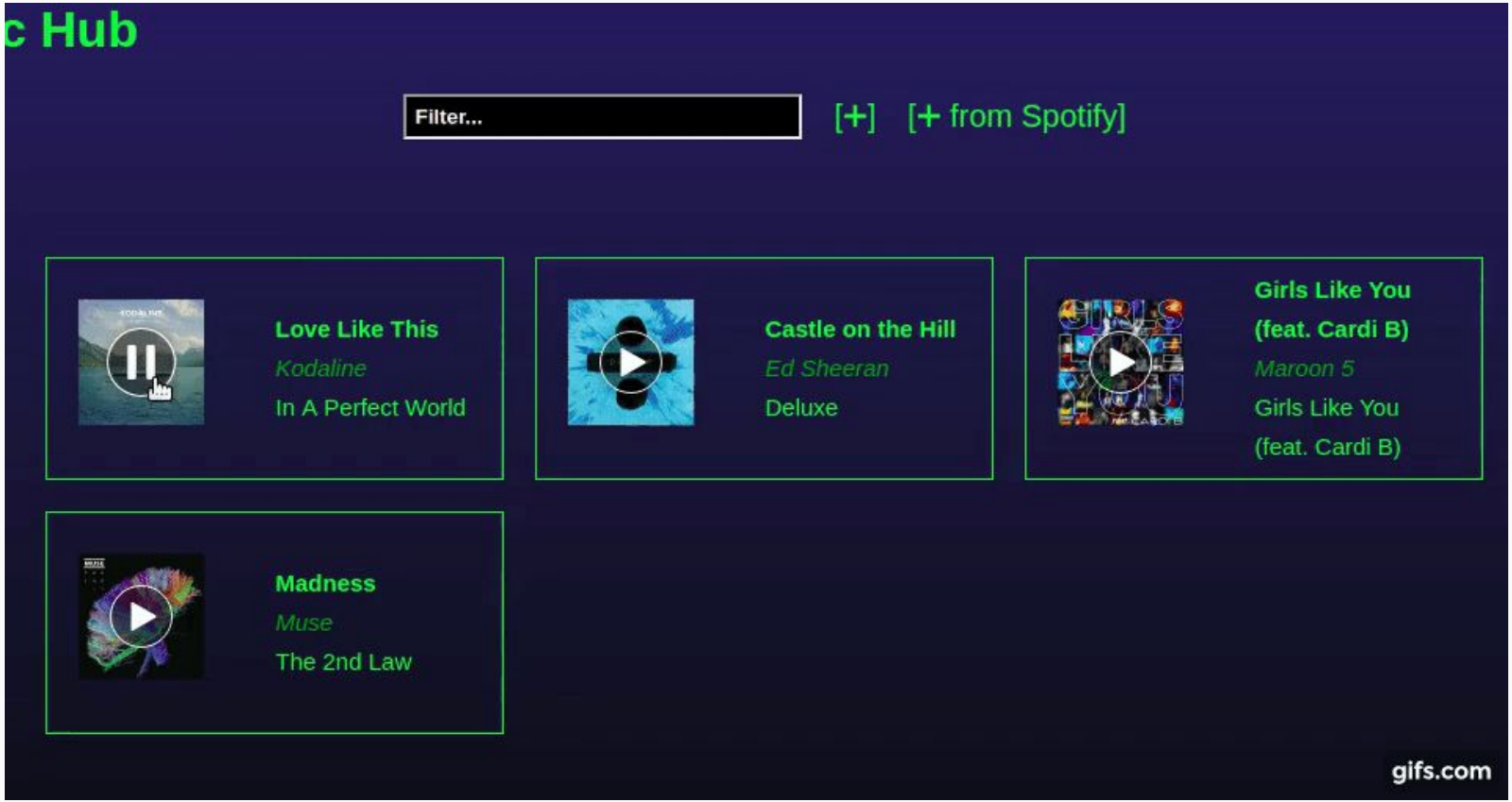
c3.8 List CSS

```
// 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 - Inputs generate event, which is processed by an event handler function

Events - provides a way to know when user clicks a button, or types in an input, etc.
When a user types in an input → a “onChange” event is generated by the input
When a user clicks a button → a “onClick” event is generated by the button

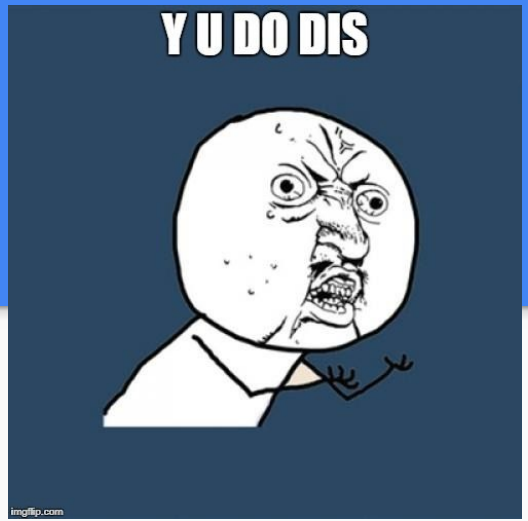
An **event handler** is a function that takes the **event** and processes it. We can define logic here, i.e. what to do when the event happens.

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={ function(event){this.searchList(event)} } />
                </div>
                <List list={this.state.list}/>
            </div>
        );
    }
...
```

But we're getting an error!

```
▶Uncaught TypeError:      UIManager.jsx:33
Cannot read property 'searchList' of
undefined
    at onChange (UIManager.jsx:33)
    at HTMLUnknownElement.callCallback (re
act-dom.development.js:145)
```



This is undefined inside the event handler

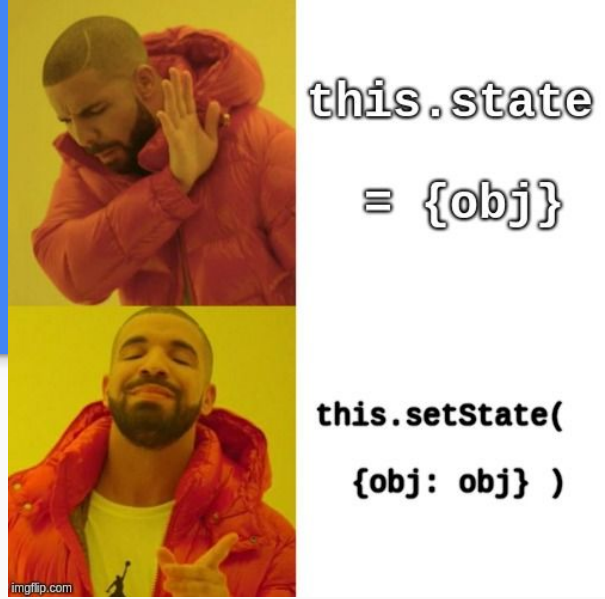
We need to use an ES6 arrow function to make sure **this** is defined inside a nested function (i.e. a function inside a function)

Full explanation: https://codepen.io/lenmorld/post/arrow-functions-and-this

c4.2 using an ES6 arrow function inside event handler

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

                <div className="options">
                    <input type="text"
                           placeholder="Filter..."
                           onChange={ (event) => {
                                        // debugger;
                                        this.searchList(event);
                                      }
                                    } />
                </div>

...
```

c4.3 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**

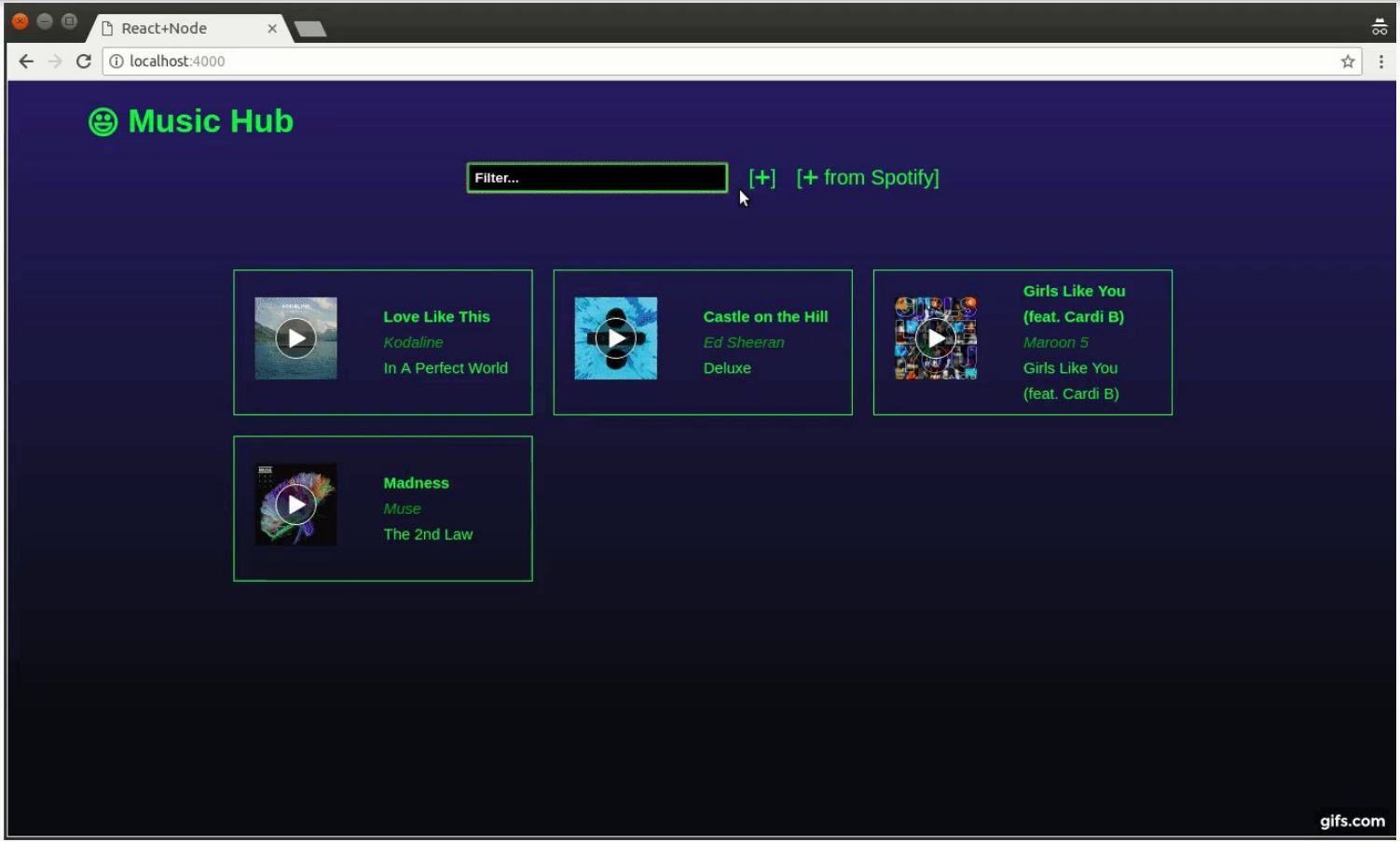
Github Link: https://github.com/lenmorld/rnw/compare/c4.2...c4.3

c4.4 filtering list based on state.search_term

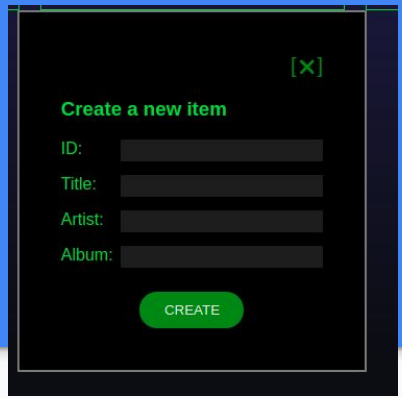
1. Create temp. Vars for this.state.list, this.state.search_term
2. If search_term empty, return full list
3. Else filter(): go through each item, and include in list if item's title in lowercase matches the search_term in lowercase

Github Link: https://github.com/lenmorld/rnw/compare/c4.3...c4.4

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 here: https://github.com/lenmorld/rnw_files/blob/master/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:

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

In Child:

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

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 **function prop**

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

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 **UIManage**r

- 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

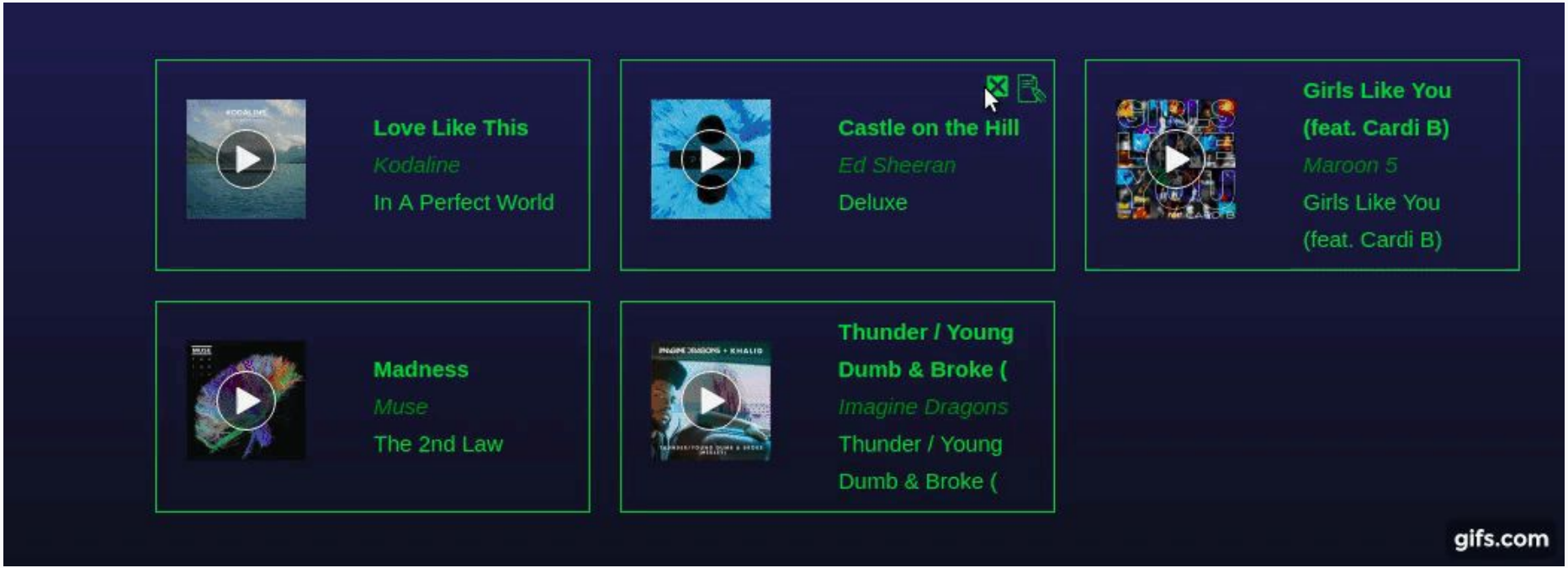
Set CSS class to “modal”

hideForm()

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

Optional: invoke after adding a new item

Delete and Update item



c4.11 Delete ☒ and Edit 

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

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

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**
- on List, no arrow functions, no args on function props since it's only a middleman

c4.12 deleteItem() method

1. Copy list values, not reference
2. Filter copy using filter(), by excluding item to delete
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 the ***form_mode*** to **ItemForm** as props, alias **mode**

```
<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. showForm()

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. hideForm() - must be defined, similar to the one in **ItemForm.jsx**

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

ItemForm:

1. Based on **this.props.mode**, invoke either this.props.create or this.props.saveUpdatedItem

c4.16 set ItemForm fields on [+] too

Notice that Create [+] does not set the correct fields on ItemForm

UIManager

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

Recap

Where to put data? State vs props

One big component or many small components

Things that could be improved in frontend (try it on your own later :D)

- Duplicate code in hideForm
- Could be just one onSubmitForm, UIManager can decide which one (create/edit) since it has the data anyways
  - Can be further optimized into one method with a switch(method) case
- Advanced: caching so spotify track won't flicker on reloads

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

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

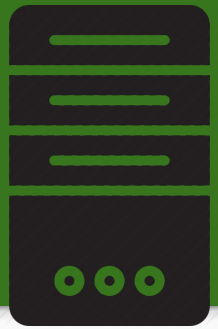
- **But when we reload the page, the changes are gone**

We will persist the changes in a data file using Node

- It is not advisable (nor easy) to manipulate data files in the frontend

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
  ...
```

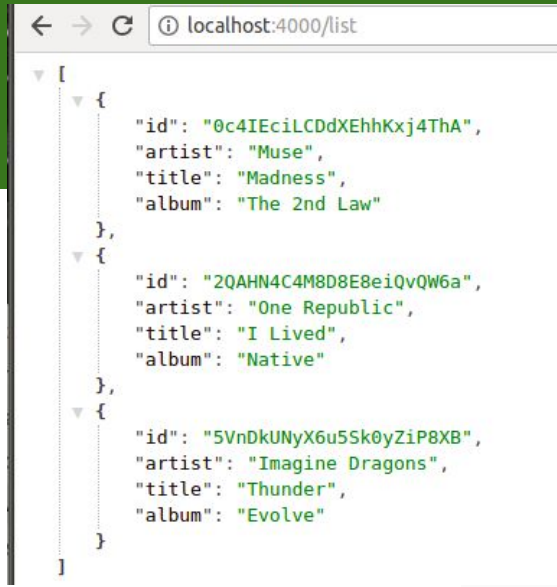
- Read file from server-side
  - Make new directory **server/** in application root
  - Move data from ***app/data.js*** *(frontend)* to ***server/data.json*** *(backend)*
  - Create a new file **server/utils.js**
    - This will be used for file-reading, etc
  - Define readJSON() in **utils.js** using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

GIthub Link: https://github.com/lenmorld/rnw/compare/c4.16...c5.1

c5.2 READ routes

Establish routes for fetching data

- Fetch list - gets all list items
  - `app.get("/list", function(req, res))`
- Fetch one
  - `app.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}`);
```


```
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

- Import axios
- Fetch data from backend before page is rendered. How?
  - There is a React lifecycle method we can use: ***componentWillMount()***

componentWillMount()

  - Axios.get(); code inside .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

Understanding HTTP requests, URL params and Request Body

File link: https://github.com/lenmorld/rnw_files/blob/master/http.md

$ 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

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

- Import body-parser and use as shown
- 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

Test requests:
https://github.com/lenmorld/rnw_files/blob/master/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 routes

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 them here (except the state stuff, of course)
- **TEST**
  - Use same commands as before: https://github.com/lenmorld/rnw_files/blob/master/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?**

Some improvements

- Produce errors when:
  - Create - Song ID is already in the data.json
  - Delete - song that does not exist (only possible in API)
- Frontend:
  - Prevent modifying ID, make it read-only on EDIT

Node CRUD + MongoDB

noSQL databases

```
// get mLab example
```

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 now, we'll just 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
   a. Ubuntu: `sudo service mongod start`
   b. Mac: `mongod`
   c. 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, init. Stuff
2. **mongo_db.js**
   a. 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()**
   b. Export file using **module.exports**
3. **server.js**
   a. Import and use mongodb.MongoCleint
   b. Import **server/mongo_db.js** file
   c. Replace file read call with **mongo_db.init()** and define the **then()** and **catch()** functions
   d. *** If using local mongodb, use the *localhost* version of *db_connection_url*

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);
    });
});
```

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
}
```

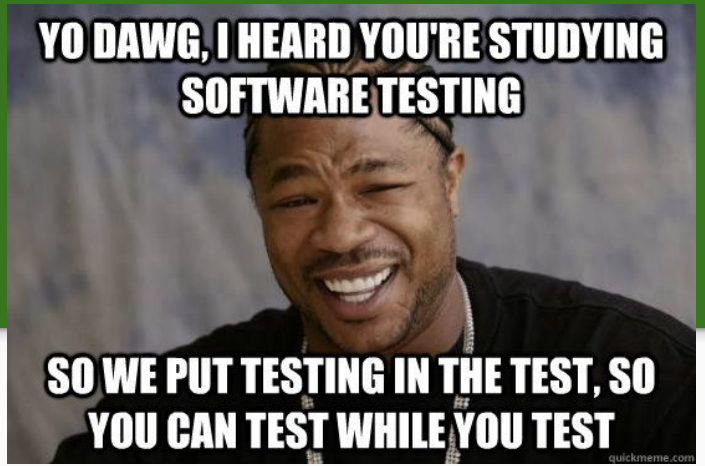
c6.3 translate all CRUD calls to db version

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



All CRUD operations should work as they were working before, and should be reflected to the mLab database (local mongo db if using local)

For /list/:id, test using browser or cURL, since we haven’t implemented this part yet in React

Improvements

- Perform some more validation on all CRUD operations performed
  - E.g. check first if item exists before Create and Delete
- Modularize duplicate code in mongo CRUD, which is sending entire list back to frontend
- Move all routes to a **routes.js** file, and use exports, require to import in **server.js**
- Challenge: improve fullstack code so backend doesn't have to return all items on a Create, Update, Delete e.g. just a message
  - The challenge here is making sure backend data and frontend data is always in-sync

Integrating with Spotify API

Spotify search API

Intro to API Oauth2

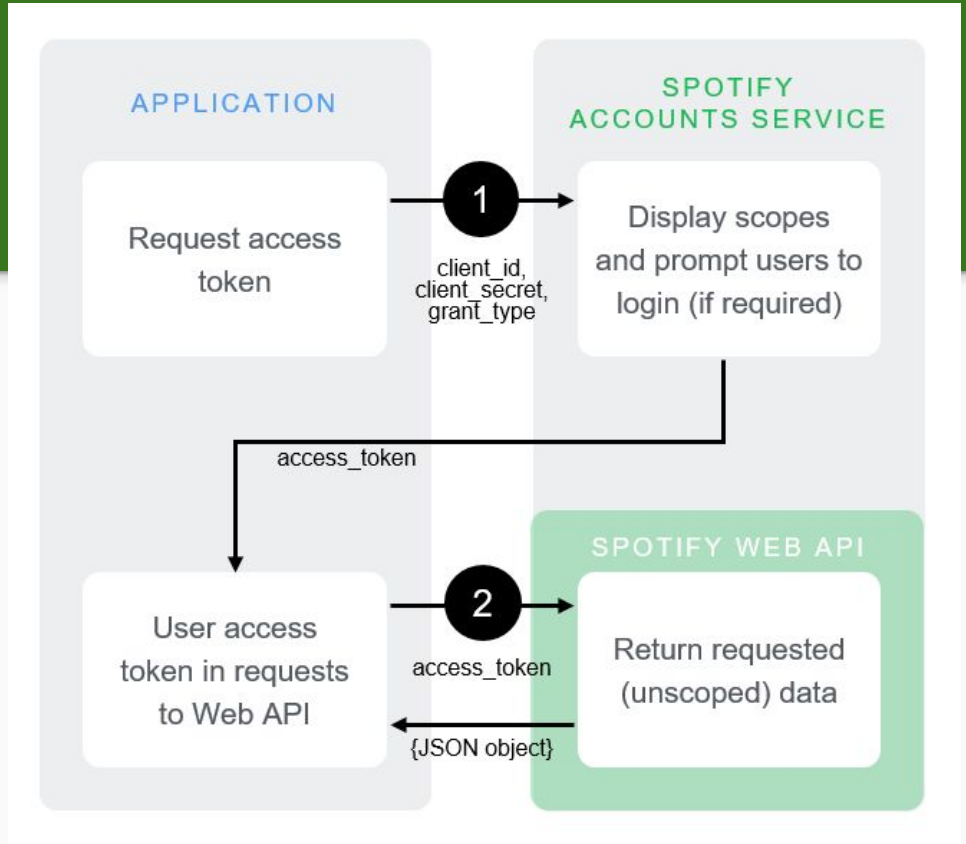
Spotify uses **OAuth2**, which requires an application to send an initial “access token request”, containing encoded user credentials (Spotify dev account). The access token (aka “bearer” access token) is good for a period of time.

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

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 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:

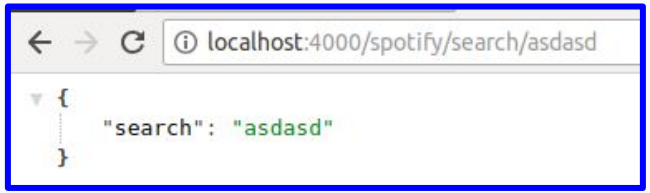
1. Prepare the **request**, including **encoding query params**
2. Process the **response**, which includes **filtering, preparing data for render in React**

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()


```
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 https://github.com/lenmorld/rnw_files/blob/master/Spotify.jsx
Notice 3 event handlers that we have to do

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

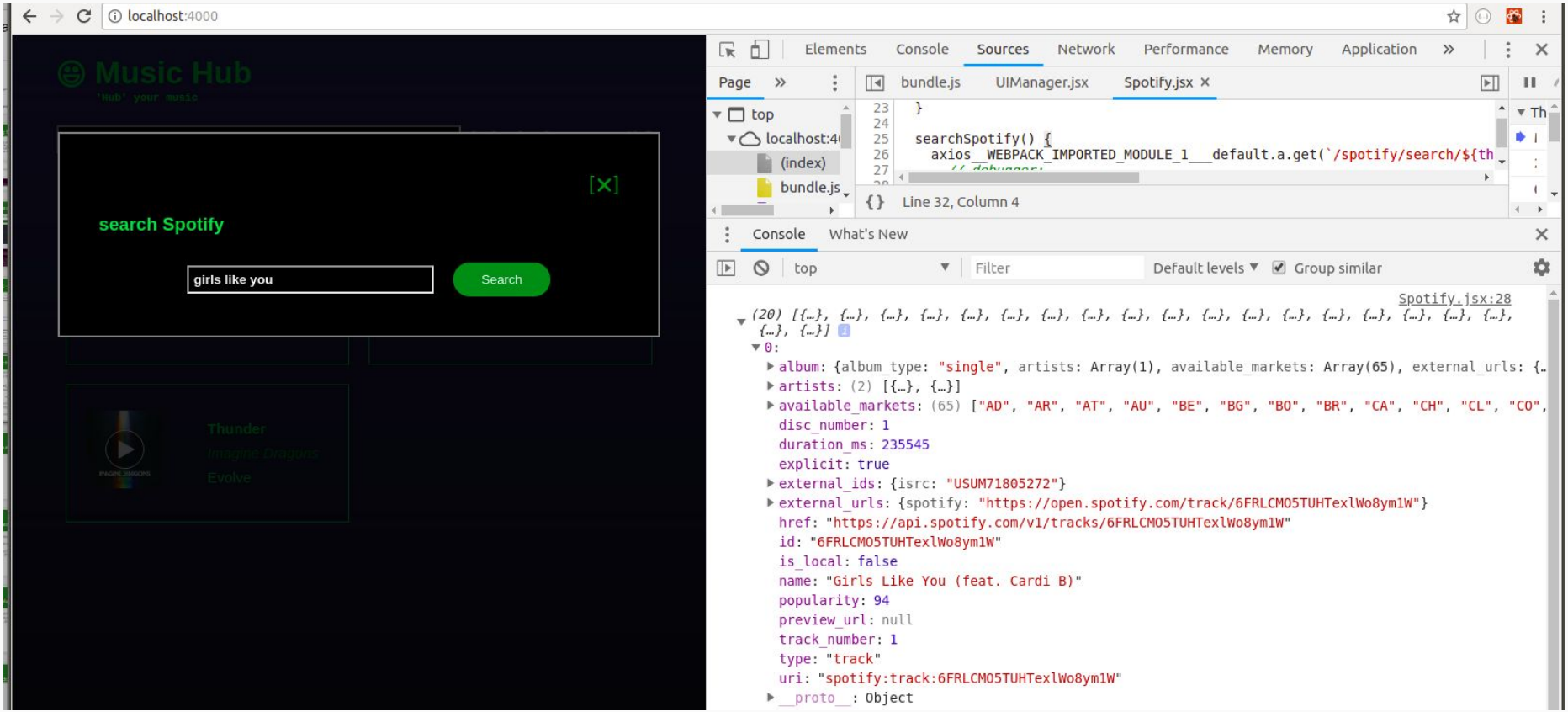
trackSearchTerm(event**)** - setState search_term to event.target.value

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



c7.6 transforming API’s response data into UI data

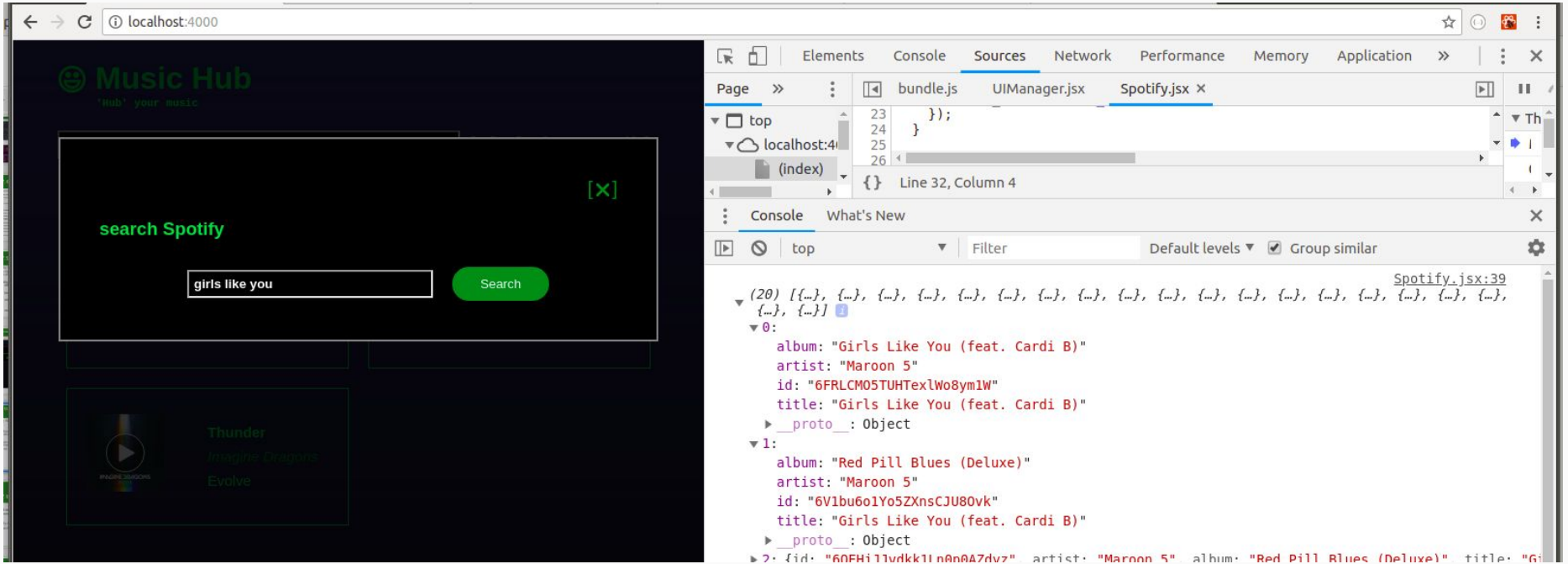
So far, this is the item format we have been using:

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",
    }
```

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!

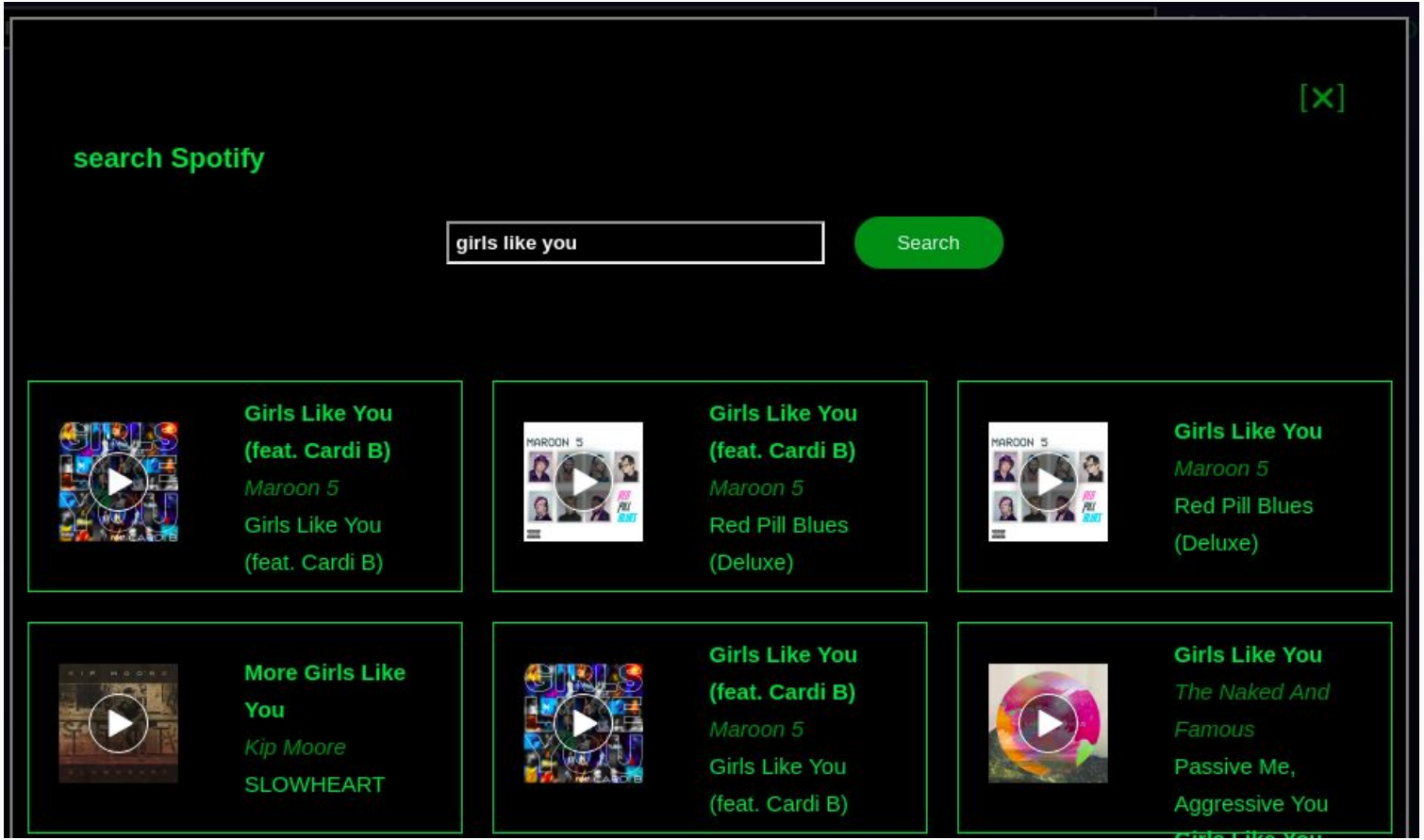
Only thing we have to do here is

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), should be either item from Spotify's toggleItem. If that is 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>
...
```

Overall Testing

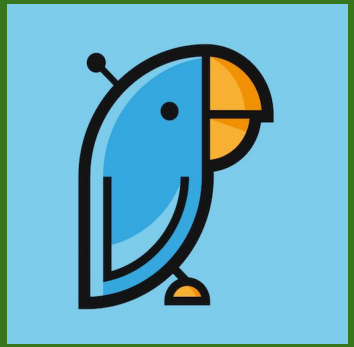
All functionalities should work

Performance could be a bit slow, depending on database and Spotify API load, as well as network latency

A few sample random songs to add (just play to get title, artist, album)

4uLU6hMCjMI75M1A2tKUQC
7GhIk7Il098yCjg4BQjzvb
0FutrWIUM5Mg3434asiwkp
20UthavN6ATWdaZGdeRPuM
1oQCO7vLjHfOiSW10Luahl

Slack → Polly



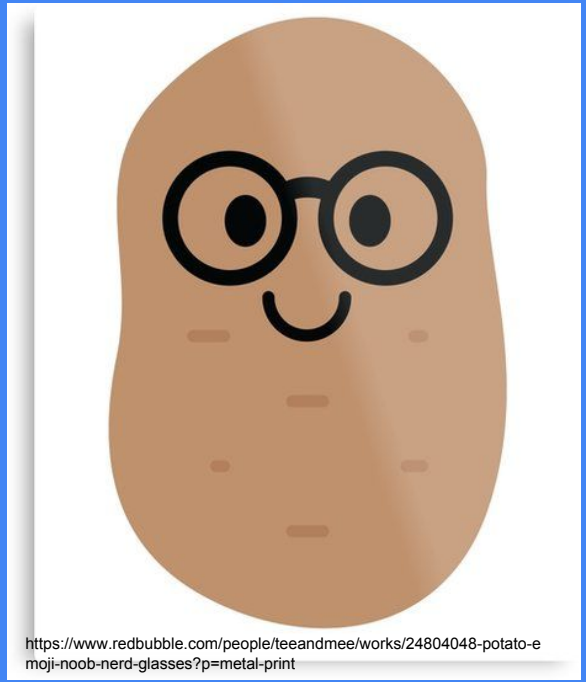
What should we do with the remaining time?

Shameless Plug!

We’re always hiring interns and full-time employees.

If you’re interested, DM me in Slack!

SORRY FOR LONG WORKSHOP! Here’s a nerdy potato



https://www.redbubble.com/people/teeandmee/works/24804048-potato-emoji-noob-nerd-glasses?p=metal-print

Thank you!!!