Dev Log

taper-web

# 1/11/18

## Initiate Project

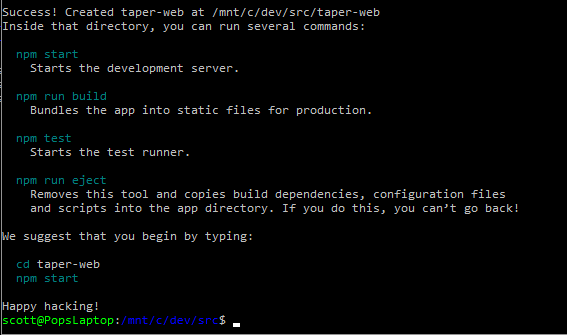
### Github Setup

* create new repo called taper-web; keep it empty - no readme, ignore, license

### Create React App

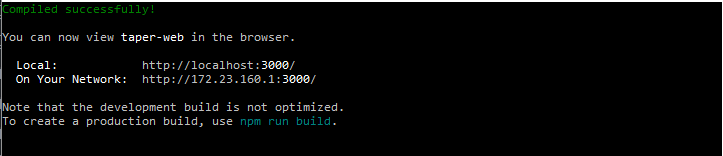
Open Ubuntu shell

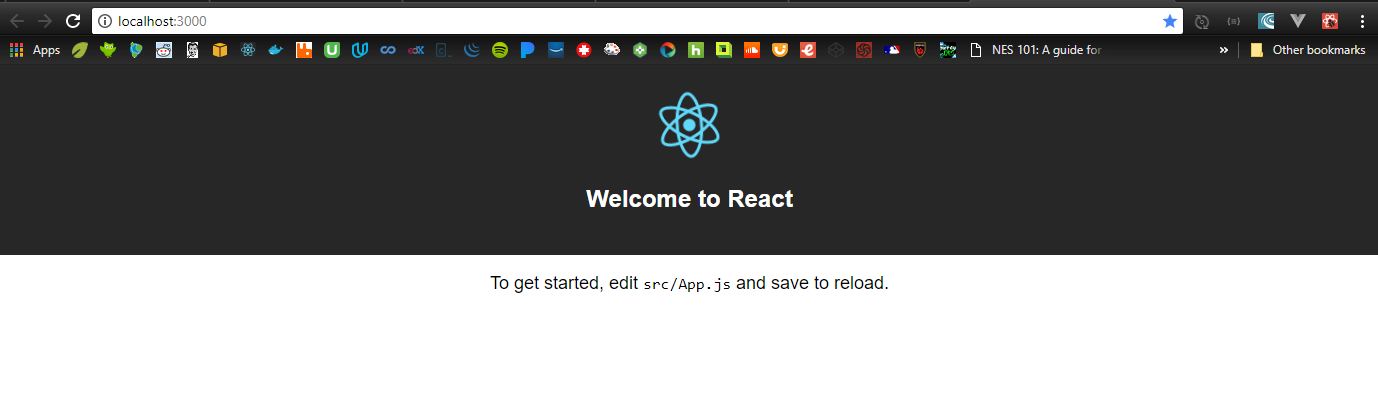
* cd /mnt/c/dev/src
* create-react-app taper-web



### Serve React App Locally

* cd taper-web
* npm start





### Git Setup

Open a new Ubuntu shell

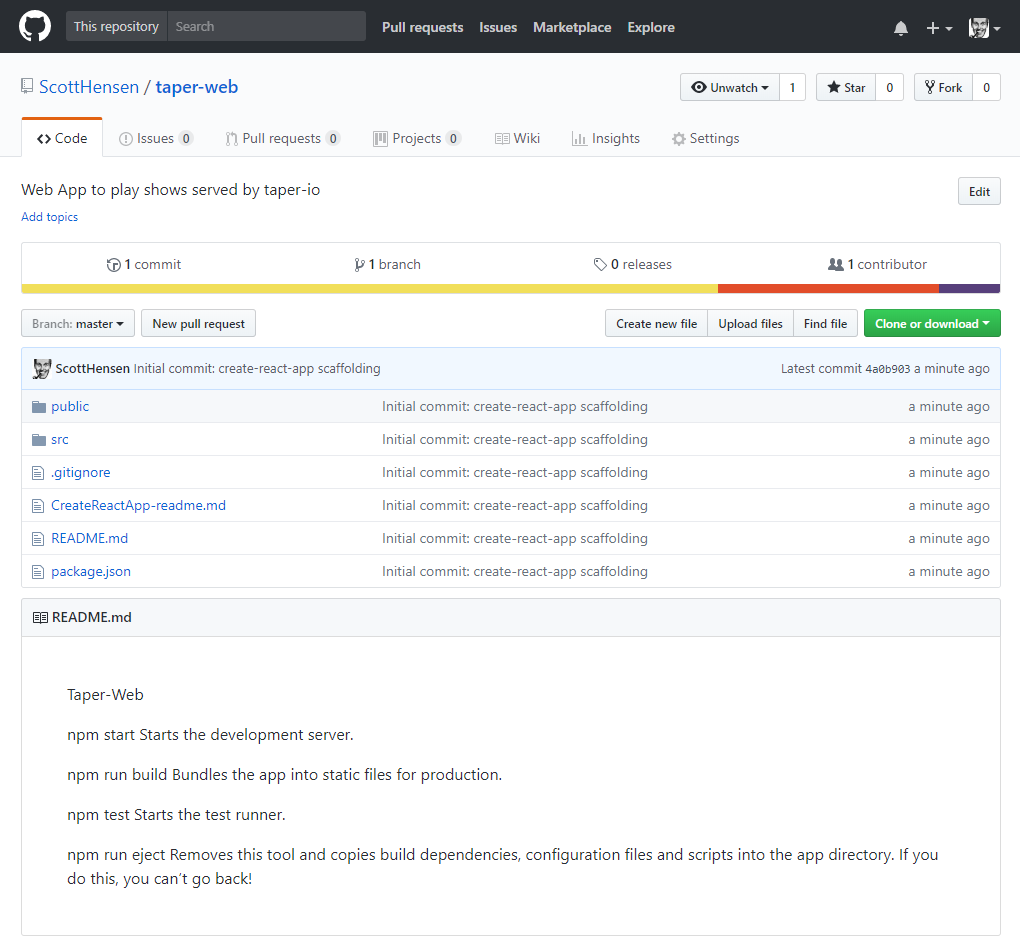
* cd taper-web
* mv README.md CreateReactApp-readme.md

(That readme is huge; it’s the one on create-react-app’s github repo. Make one for THIS project.)

* vi README.md
  + cut/paste the npm start, npm run, etc instructions into the readme
* git init
* git add .
* git commit –m “Initial commit: create-react-app scaffolding”

### Sync Git and Github

* git remote add origin <https://github.com/ScottHensen/taper-web.git>
* git push –u origin master



### Create this dev log

Woah… meta.

* mkdir doc
* save the log to taper-web/doc/devlog.docx
* git status
* git add .
* git commit –m “Add devlog doc”
* git push –u origin master

## Planning

A week of coding saves an hour of planning.

### Stories

GIVEN a user has [browser] open; WHEN the user navigates to taper web site; THEN the taper web landing page appears.

GIVEN taper landing page is open; WHEN the user clicks on an available show; THEN the show page replaces the landing page.

GIVEN the show page is open; WHEN the page loads; THEN the first song will begin playing at 70% volume.

GIVEN a song is playing; WHEN the song ends; THEN the next song will begin.

### Components

index.js

assets/

layouts/

primaryLayout

layout-helpers/

primaryHeader

sideMenu

aside

band/

tour/

show/

setlist/

set/

track/

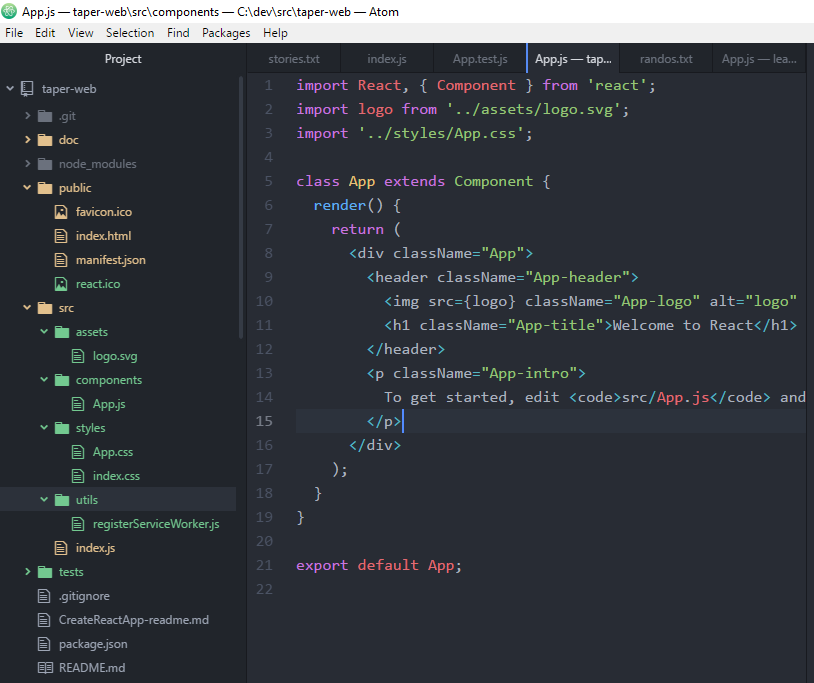
styles/

utils/

xhr

## Build Landing Page

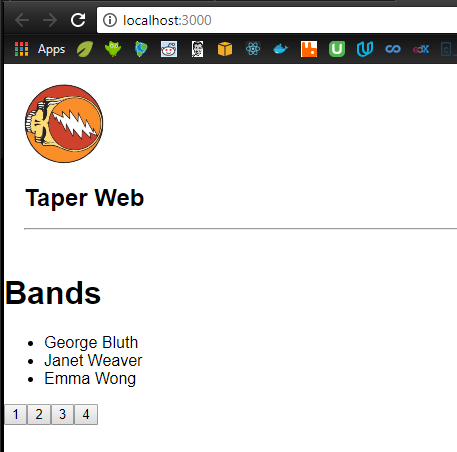
* Import project into editor
* Update manifest.json and index.html to use my title and favicon.
* Build the directory structure
* Move the files to the right directories; change the imports as I move them to keep the app working.



* Replace react boilerplate in App js with my content
* Swiped some layout and xhr code to use as boiler plate from [Brad Westfall’s phx react meetup example](https://github.com/bradwestfall/PhxReactMeetupTalk)

# 1/12/18

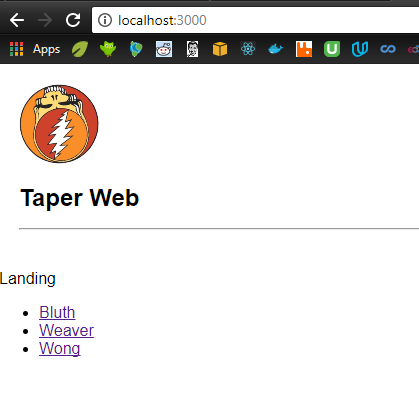
* Got missing module errors, so added these two…
* npm install –save react-router-dom
* npm install –save axios
* here’s how it looks now

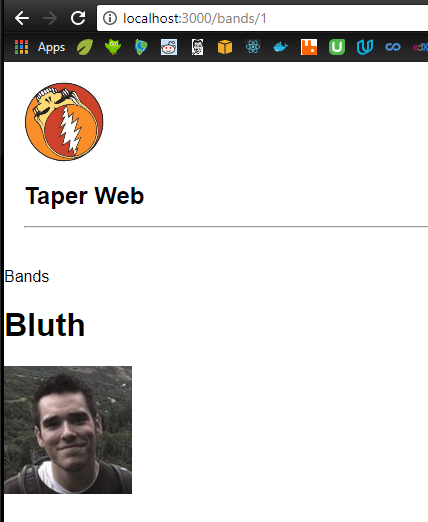


* git status
* git add .
* git commit –m “Add layout, bands page and simple xhr functionality”
* git push –u origin master

# 1/13/18

* removed the paging stuff from landing page
* added second page for bands
* added routing for the following
  + / -> landing
  + /bands -> landing
  + /bands/2 -> bands





* git status
* git add .
* git commit –m “Add routing between landing page and bands page; removed paging.”
* git push –u origin master

## Build taper back-end

### Stories

#### receive taperRequest objects

* Band
* showDate
* showContainsSong
* showVenue
* Order: first, last, random, all

#### parse them to determine how best to call archive.org

* if Band not valid, return error
* if showDate valid, use it in filter
* if showSong valid, use it in filter
* if showVenue valid, use it in filter
* if orderValid, use it in response construction
* // allow just year, just mo/year, or the whole thing; note that archive.org uses “ccyy-mm-ddThr:mn:scZ”
* // band, song, venue must be escaped
* // band, song, venue could be mapped to dictionary if that’ll help make the searches more robust

#### construct the url

* base url = https: //archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,sources&sorts=date&q=collection%3A
* append (<bandName>)
* If (showDate) append %20AND%20(<formatted showDate>)
* If (showSong) append %20AND%20(<showSong>)
* If (showVenue) append %20AND%20(<showVenue>)

#### call archive.org to get archiveResponse object

#### parse archiveResponse

* when order=first
  + title = items[0].title; venue = …
  + url = EMBED\_URL\_HDR + items[0].identifier // this is vulnerable to cross-site scripting; remember the right way to do it.
* When order=last, get items.length for index
* When random, generate a rando index
* When all

#### return taperResponse

### Hand-built archive.org get requests

#### play a random dead show.

<https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3AGratefulDead>

pick one at random from json, then build this url...

https://archive.org/embed/<identifier>&autoplay=1

#### play a random dead show with IKO IKO

<https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3A(GratefulDead)%20AND%20(Iko%20Iko)>

pick one at random from json, then build this url...

https://archive.org/embed/<identifier>&autoplay=1

#### play the last dead show with IKO IKO

<https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3A(GratefulDead)%20AND%20(Iko%20Iko)>

pick the last one from json, then build this url...

https://archive.org/embed/<identifier>&autoplay=1

#### play IKO IKO from the 1995 show at Riverport

<https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3A(GratefulDead)%20AND%20(Iko%20Iko)%20AND%20(Riverport)%20AND%20(1995)>

pick one from json, then build this url...

https://archive.org/metadata/<identifier>

find name where title="IKO IKO", then build this url...

https://archive.org/embed/<identifier>/<name><&autoplay=1

#### play every IKO IKO [in chrological order | in the 90s | at Madison Square Garden]

<https://archive.org/embed/jj2008-06-14.mk4&autoplay=1>

<https://archive.org/details/deadco2017-07-01.travitz.schoepsMK4v.fob.flac16>

#### Get all dead shows from archive.org ...

<https://archive.org/services/search/v1/scrape?fields=dir,title,date&q=collection%3ADeadAndCompany>

<https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3ADeadAndCompany>

<https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3AGratefulDead>

#### Now, just the ones that have IKO IKO in them...

<https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3A(GratefulDead)%20AND%20(Iko%20Iko)>

#### Here's the last IKO IKO show - 95 STL...

<https://archive.org/embed/gd95-07-05.sbd.9370.sbeok.shnf&autoplay=1>

#### And here's the IKO IKO from 95 STL show...

<https://archive.org/embed/gd95-07-05.sbd.9370.sbeok.shnf/gd95-07-05d1t08.shn&autoplay=1>

#### Slap this in your app, and you'll be playing der chicago show! snoit!

<https://archive.org/embed/deadco2017-07-01.travitz.schoepsMK4v.fob.flac16&autoplay=1>

<https://archive.org/embed/10-29-15DeadAndCompanyTimesUnionCenterAlbanyNy&autoplay=1>

# 1/14/18 14:30

## NOTE: Lex chatbot to Alexa Skill

<https://aws.amazon.com/about-aws/whats-new/2017/09/export-your-amazon-lex-chatbot-to-the-alexa-skills-kit/>

you can port a lex app to alexa ; see Lex [doc](http://docs.aws.amazon.com/lex/latest/dg/export.html) and [faq](https://aws.amazon.com/lex/faqs/#alexa)

## New plan

At first, I thought about building the lambda/gateway backend all by itself instead of hooking it to awsmobile, but I am deciding now that awsmobile + a very simple web front end kind of belong together, so I will let awsmobile handle do the scaffolding for me. The front-end will probably remain webby and unpolished. I will create a separate mobile/web app to consume the taper api; that’s the one that will be user-facing.

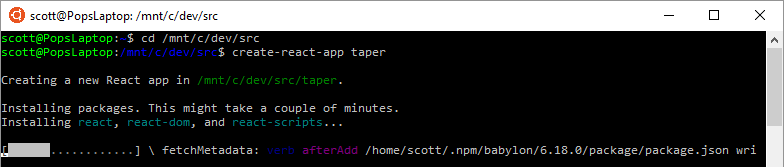
Because of this decision, I realize I’ve named the project wrong. I’m going to rename taper-web to taper. Starting from a clean slate, I will build a taper react app, put it in a new github repo, add awsmobile, build out the interface to archive.org, configure the backend’s analytics, messaging, etc.

## Start a brand new AWS Mobile Hub app

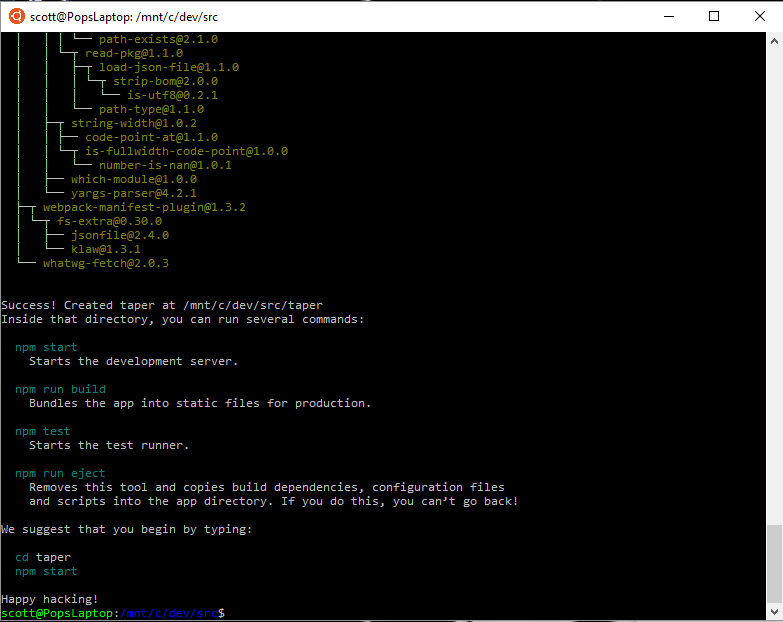
### Scaffold React Front-End

Open Ubuntu shell

* cd /mnt/c/dev/src
* create-react-app taper-web

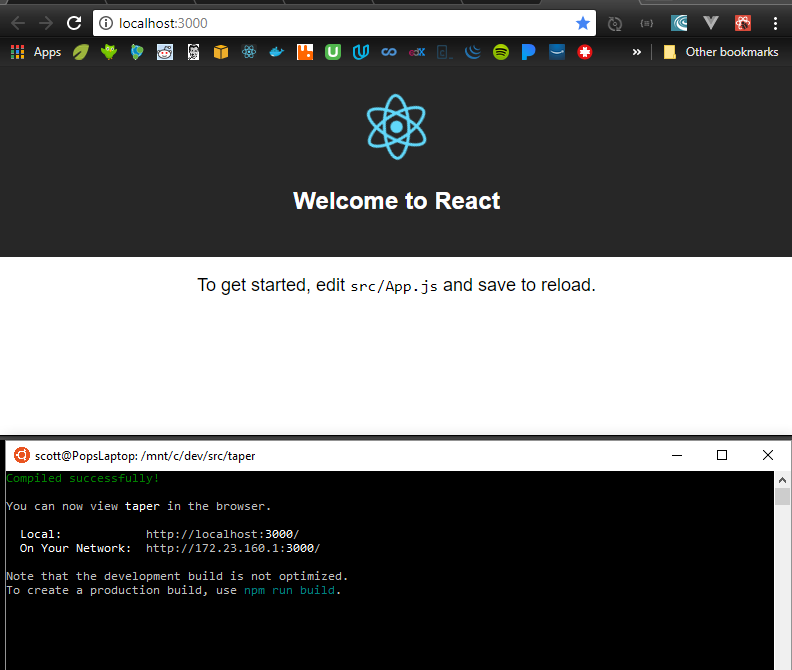


… five minutes later …



### Serve React App Locally

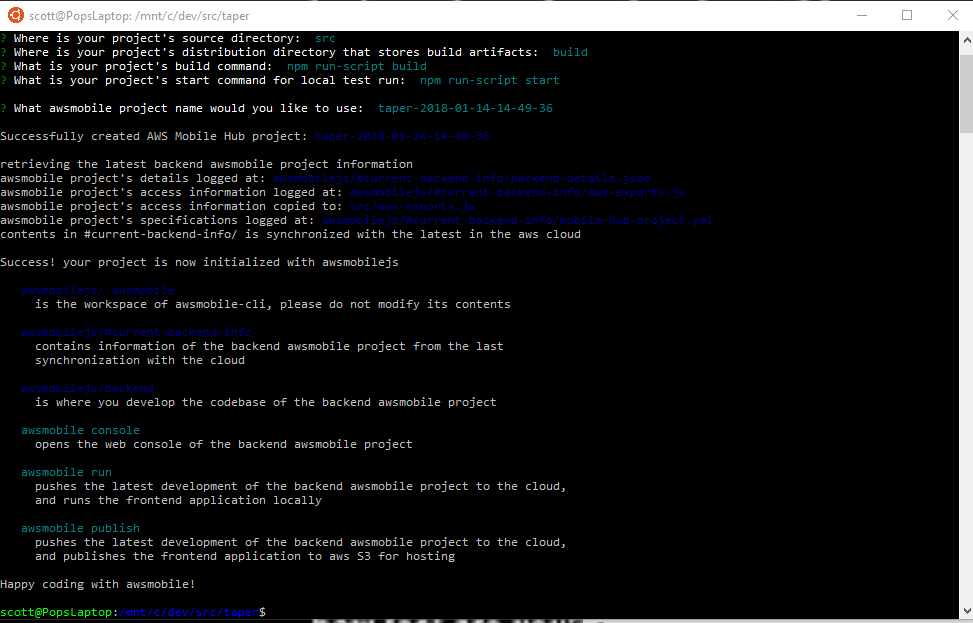
* cd taper-web
* npm start



### Scaffold Node Back-end

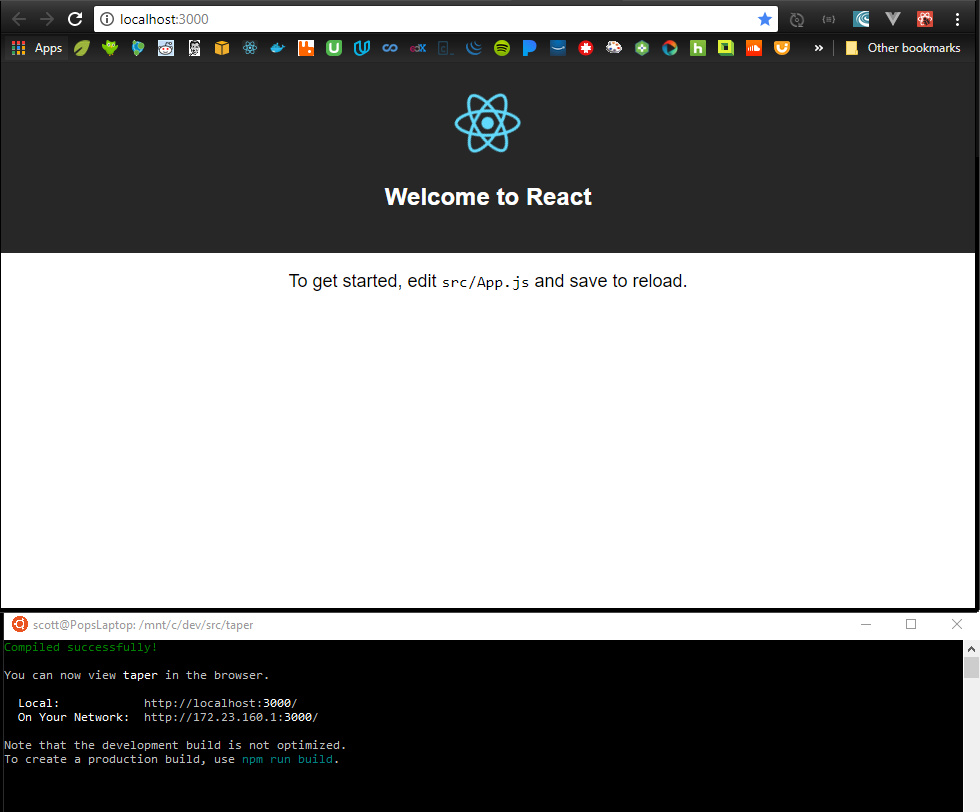
Open Ubuntu shell

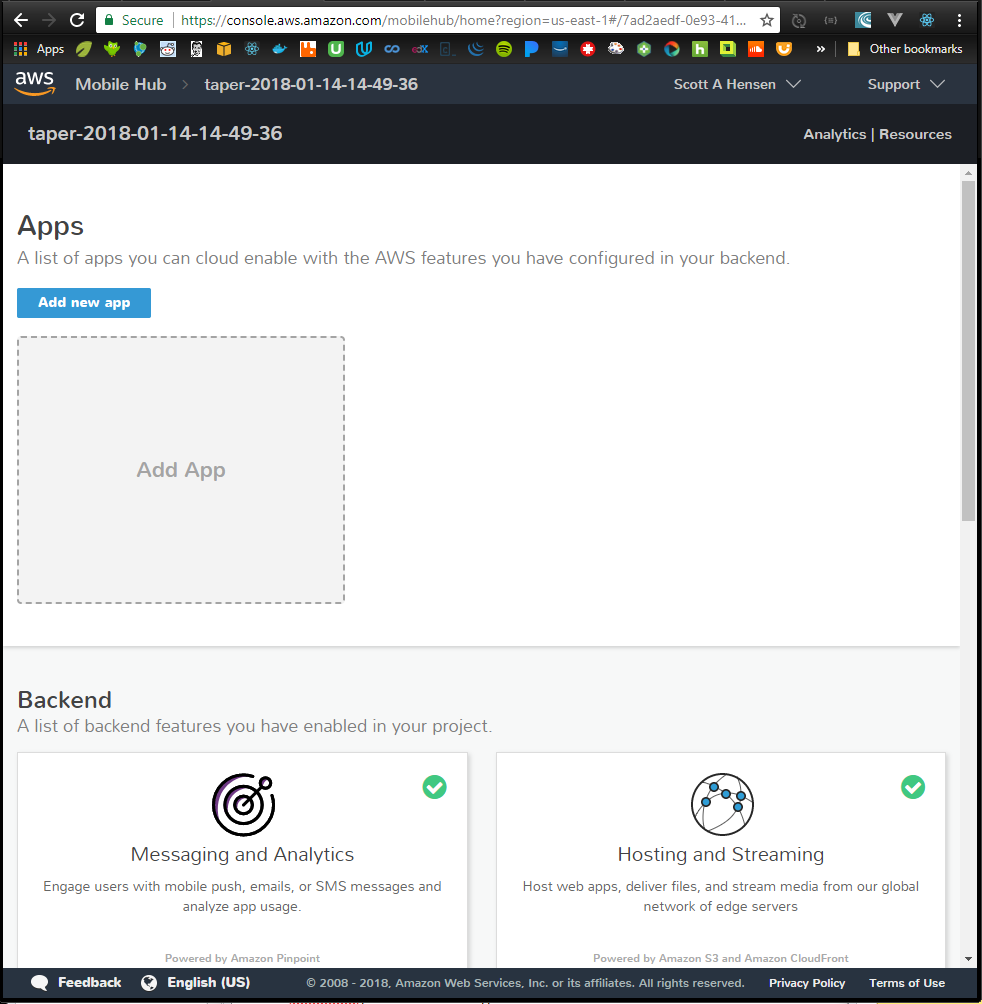
* cd /dev/src/taper
* awsmobile init

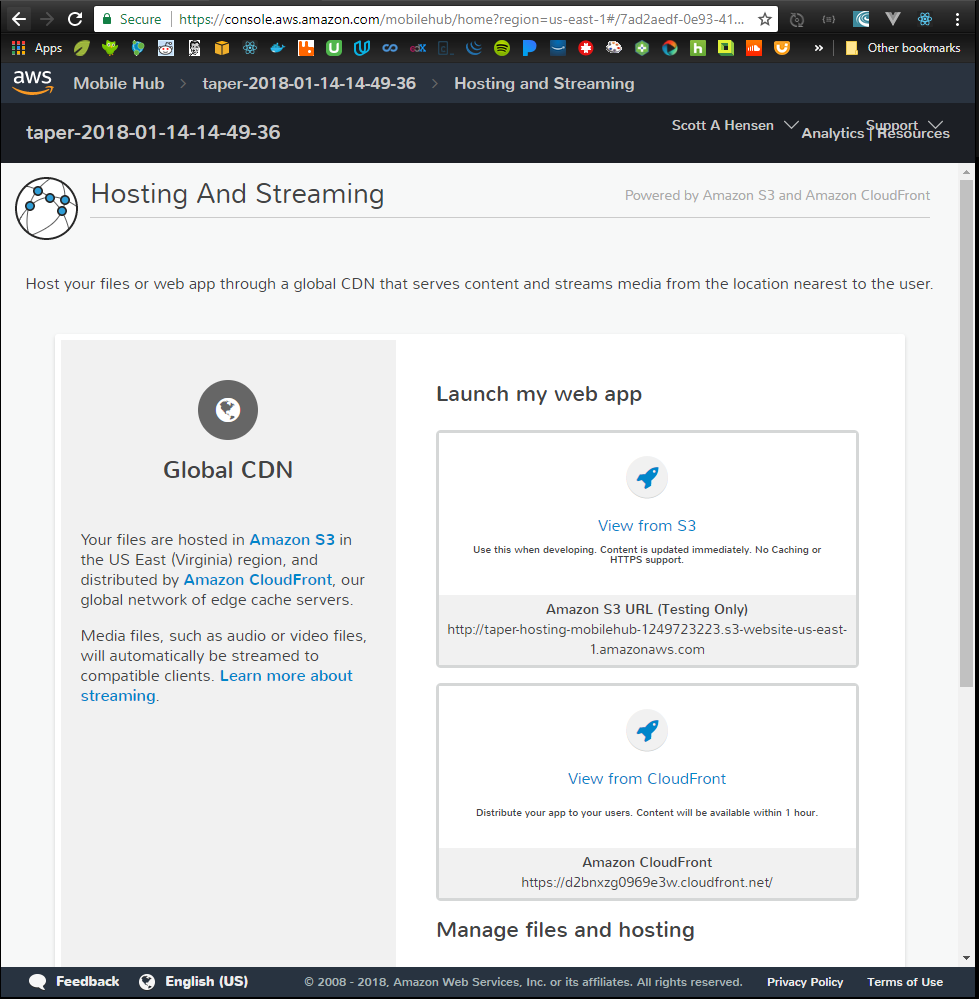


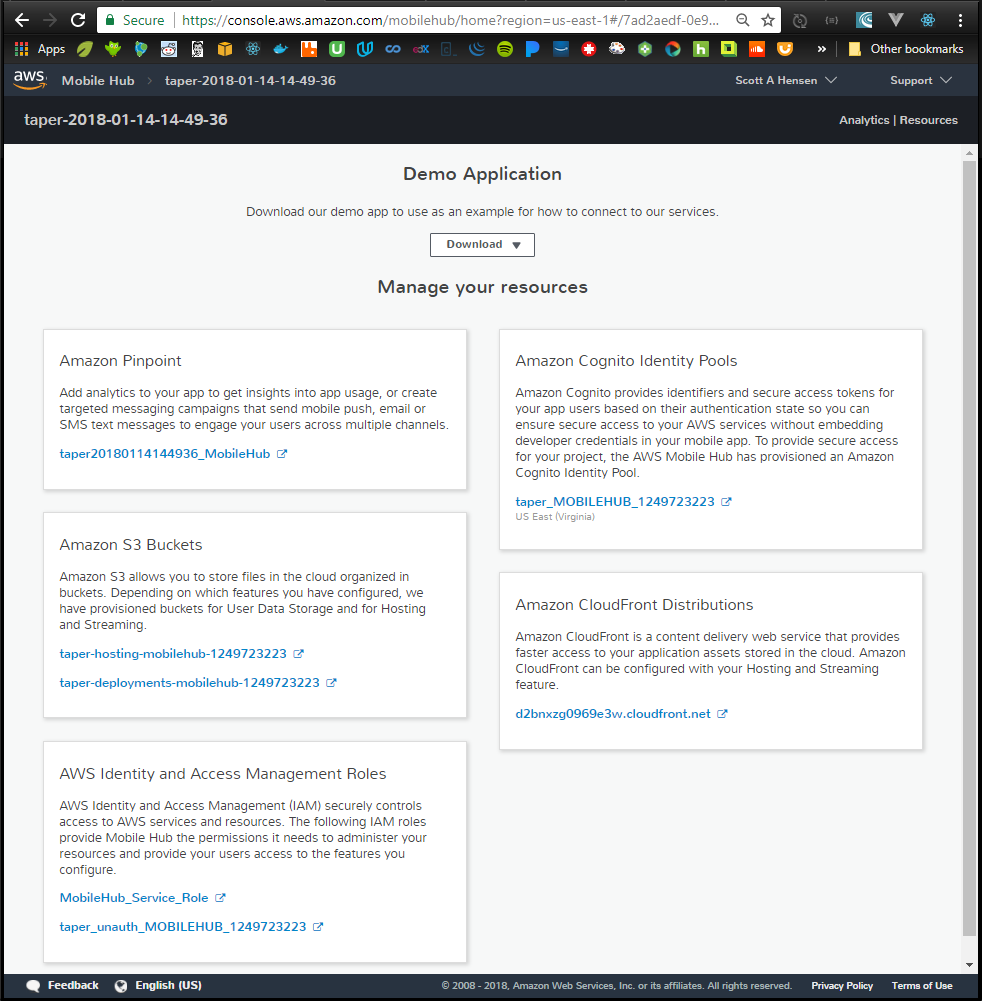
### Connect Front-end to Back-end with Amplify

* add to index.js: import Amplify from ‘aws-amplify’;
* add to index.js: import aws\_exports from ‘./aws-exports.js’;
* add to index.js: Amplify.configure(aws\_exports);
* note: this will break the locally running app because aws-amplify isn’t in node\_modules yet.
* Awsmobile run // runs npm install to load new npm\_modules, syncs local and cloud backends, pushes synced backend to awsmobile, then runs npm start



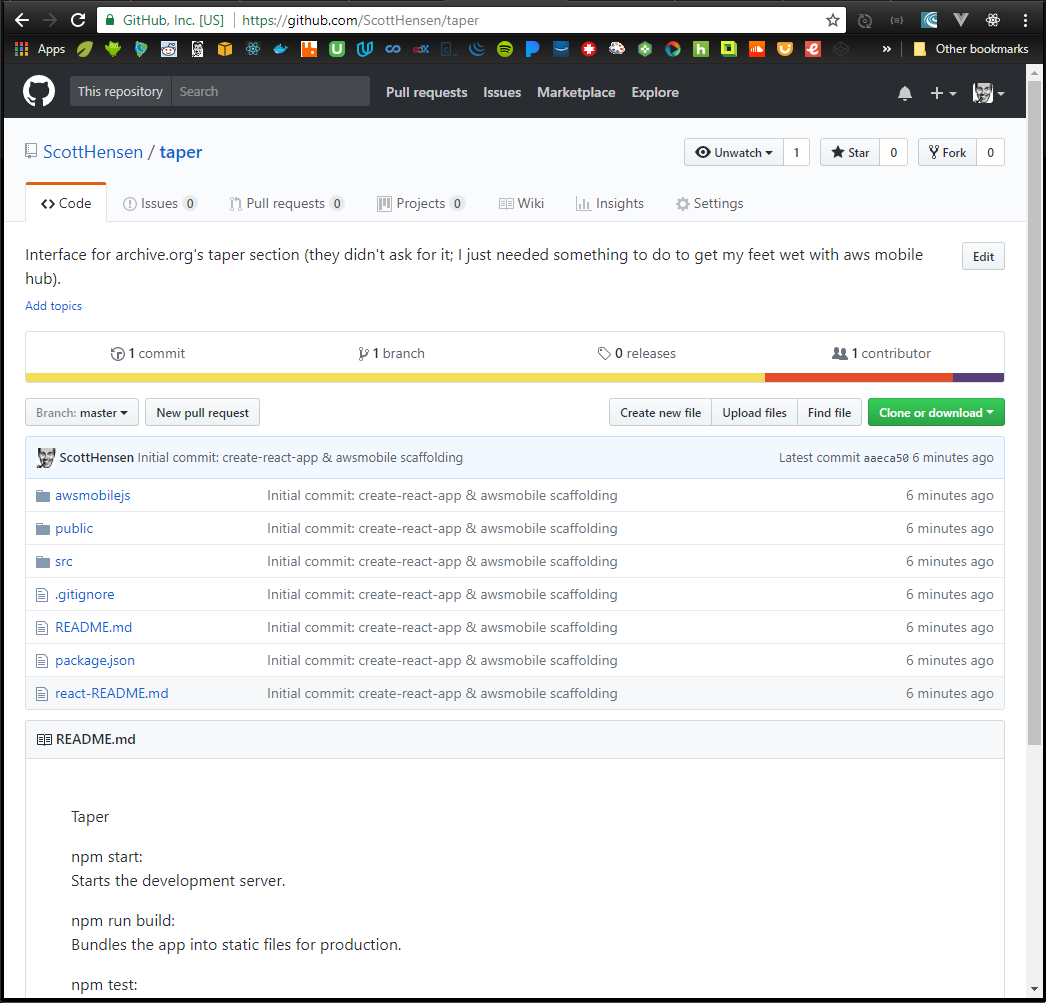






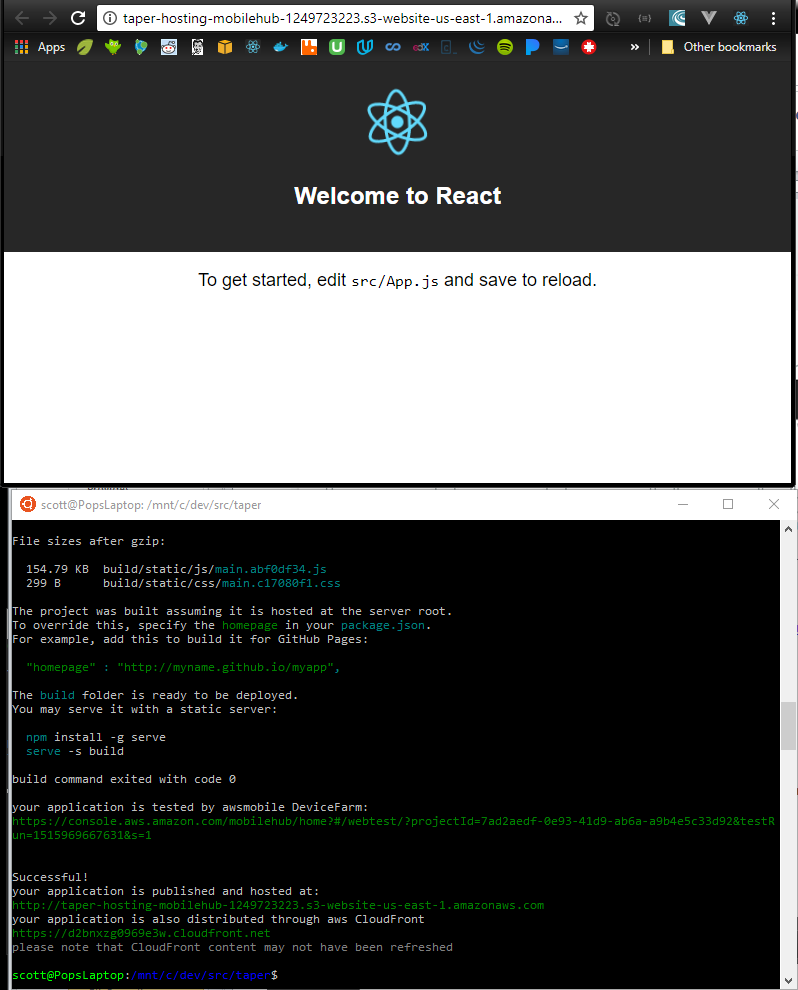
### Set up Repository

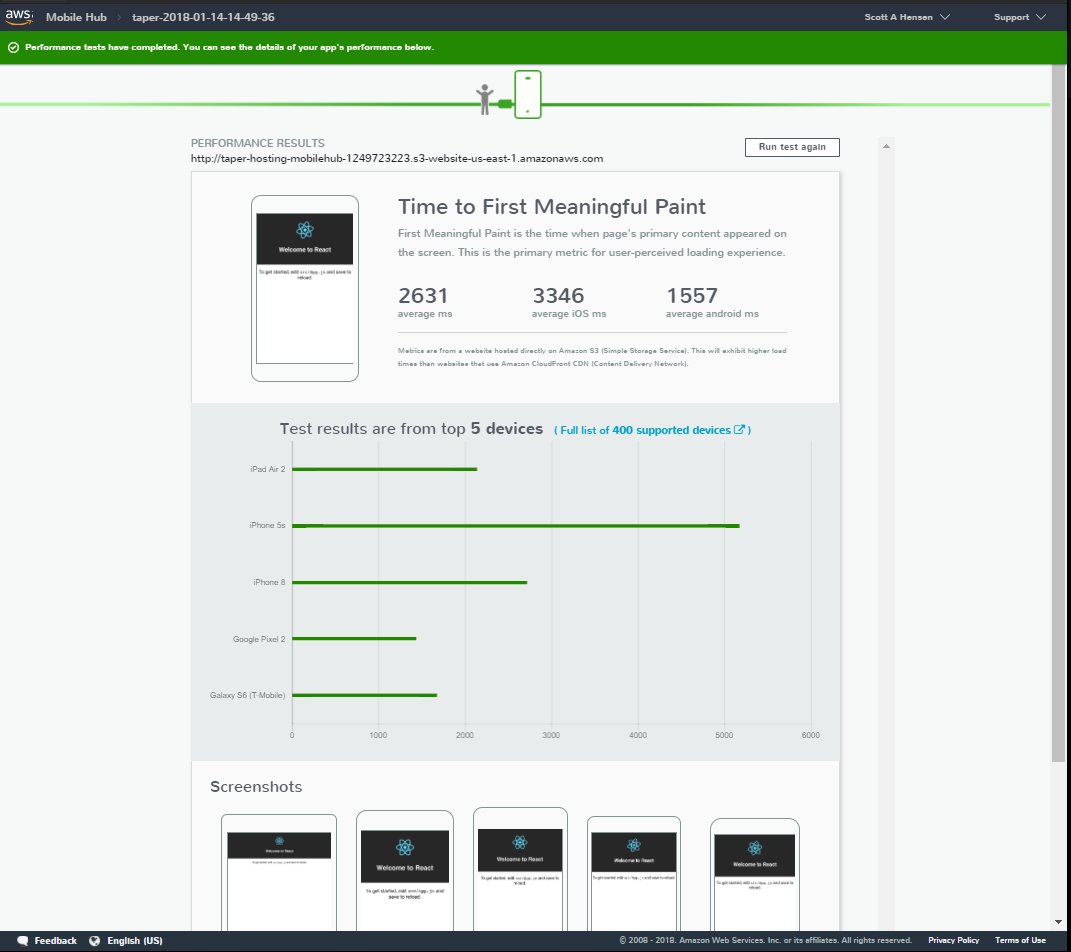
* Create new github repo – completely empty; no .gitignores, no readmes
* cd /mnt/c/dev/src/taper
* replace the huge react readme with one specific to taper
  + > mv README.md react-README.md
  + > vi README.md
  + add some basic stuff
* git init
* git status
* git add .
* git commit –m “Initial commit: create-react-app & awsmobile scaffolding”
* create github repo
* git remote add origin <https://github.com/ScottHensen/taper.git>
* git push –u origin master



### Deploy the front-end

* Pick One:
  + > awsmobile publish // does a prod build, deploys, serves it on mobile hub s3
  + > awsmobile publish –test // dash-dash; deploys it and runs it through aws device farm test service





### Finished!

* Start Time = 14:30
* Scaffolded a new react front-end with create-react-app
* Scaffolded a new node back-end with awsmobile init
* Connected the two
* Committed the working app to a repository before I started breaking things
* Deployed the front-end to aws mobile hub (the backend was deployed when it was scaffolded)
* Verified the app on aws device farm
* Stop Time = 15:50

## Build a basic front-end

* Rearrange the project folders/files to how I like them.
  + Add these folders to source: /assets, /components, /utils
  + Add these folders to components: /app, /layouts
  + Move App.\* from /src to /src/app. Fix the bugs created by the move.
  + Move logo.svg to /src/assets. Fix the bug created.
  + Move registerServiceWorker.js to /src/utils. Fix the bug created.
* Update public components
  + index.html’s title,
  + manifest’s metadata, and
  + replace react favicon with Bob
* Add primary header and layout components; hook it into app.js
* Git commit
* Stop Time = ~1hr

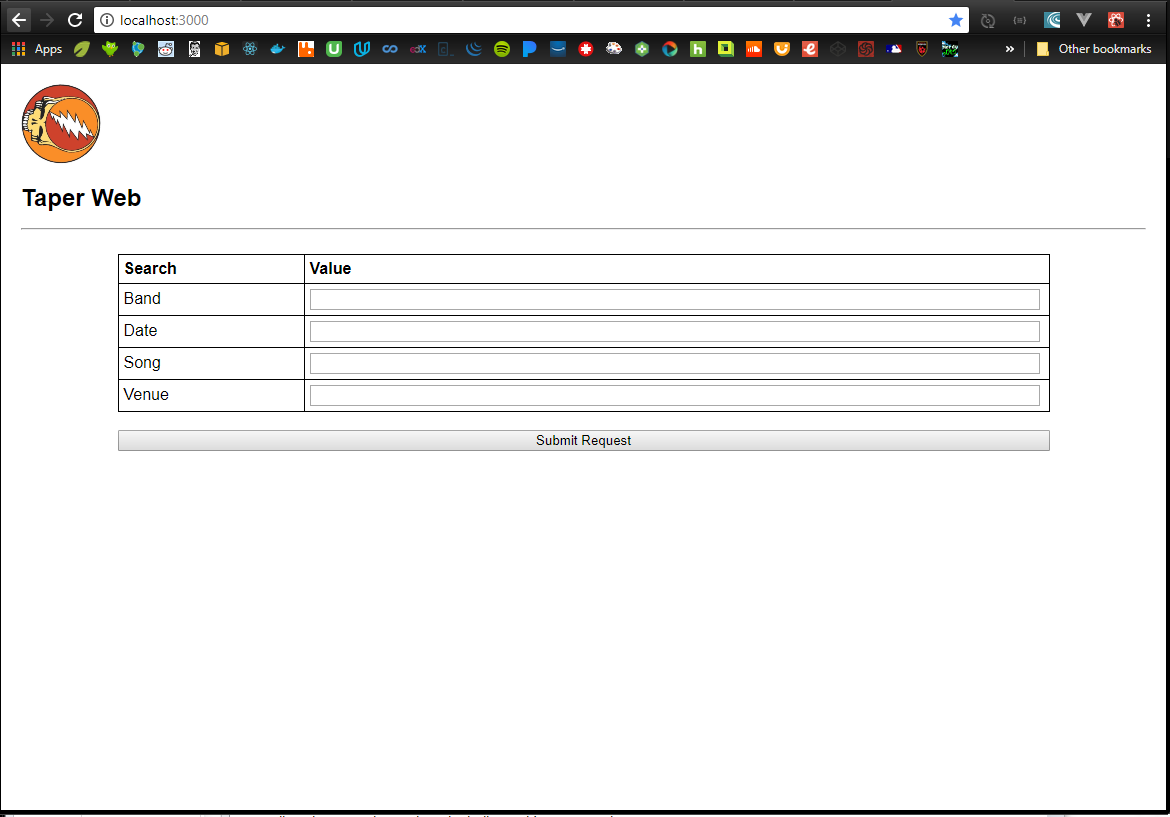
## Add archiveRequestForm component

#### Story: receive taperRequest objects

* Band
* showDate
* showContainsSong
* showVenue
* Order: first, last, random, all

##### Build it

* Make this component a class so that a state object can be maintained
* Build a label/input-value table for each search parameter
* Add handleInputChange() to update state when a value is updated
* Add submit button and handleSubmit()
* Add archiveRequestForm.css to make it a little nicer looking
* Hook archiveRequestForm into PrimaryLayout component
* Git commit
* Stop Time = ~1hr



#### Story: parse them to determine how best to call archive.org

* if Band not valid, return error
* if showDate valid, use it in filter
* if showSong valid, use it in filter
* if showVenue valid, use it in filter
* if orderValid, use it in response construction
* // allow just year, just mo/year, or the whole thing; note that archive.org uses “ccyy-mm-ddThr:mn:scZ”
* // band, song, venue must be escaped
* // band, song, venue could be mapped to dictionary if that’ll help make the searches more robust

Build It

* Add buildArchiveRequest to utils
* Hook it to ArchiveRequestForm

#### Story: construct the url

* Modified buildArchiveRequest to encode, filter, join and build final url
* Returning url to caller ArchiveRequestForm
* Git commit
* Stop time = ~2hr

## Add archiveResponse component (to dispay json)

* Added component, hooked it into ArchiveRequestForm

## Add a get api to back-end

* Added xhr to archive.org with the built searchUrl
* Got a CORS error (testing on chrome local host)
* Executed npm run build (build a prod version, so I can run it on a server)
* npm install –g serve
* serve –s build

## Add an api call to the front-end

* added. Getting cors error

## Error Will Robinson! Design Error!

The /services searches work fine from browser url bar and curl, but they are being blocked by CORS when running from my domain. Try the advancedsearch feature…

* <https://archive.org/help/json.php>
* https://archive.org/advancedsearch.php?q=licenseurl:(\*creativecommons.org\*)&fl[]=identifier,title,mediatype,collection&rows=1&output=json
* <https://archive.org/services/search/v1/scrape?fields=dir,title,date,subject,venue,source&sorts=date&q=collection%3A(GratefulDead)%20AND%20(Iko%20Iko)%20AND%20(Riverport)%20AND%20(1995)>
* [https://archive.org/advancedsearch.php?q=Grateful+Dead+Iko+Iko+Riverport&fl[]=dir,title,date,venue,source&rows=50&output=json&callback=callback](https://archive.org/advancedsearch.php?q=Grateful+Dead+Iko+Iko+Riverport&fl%5b%5d=dir,title,date,venue,source&rows=50&output=json&callback=callback)