Movi

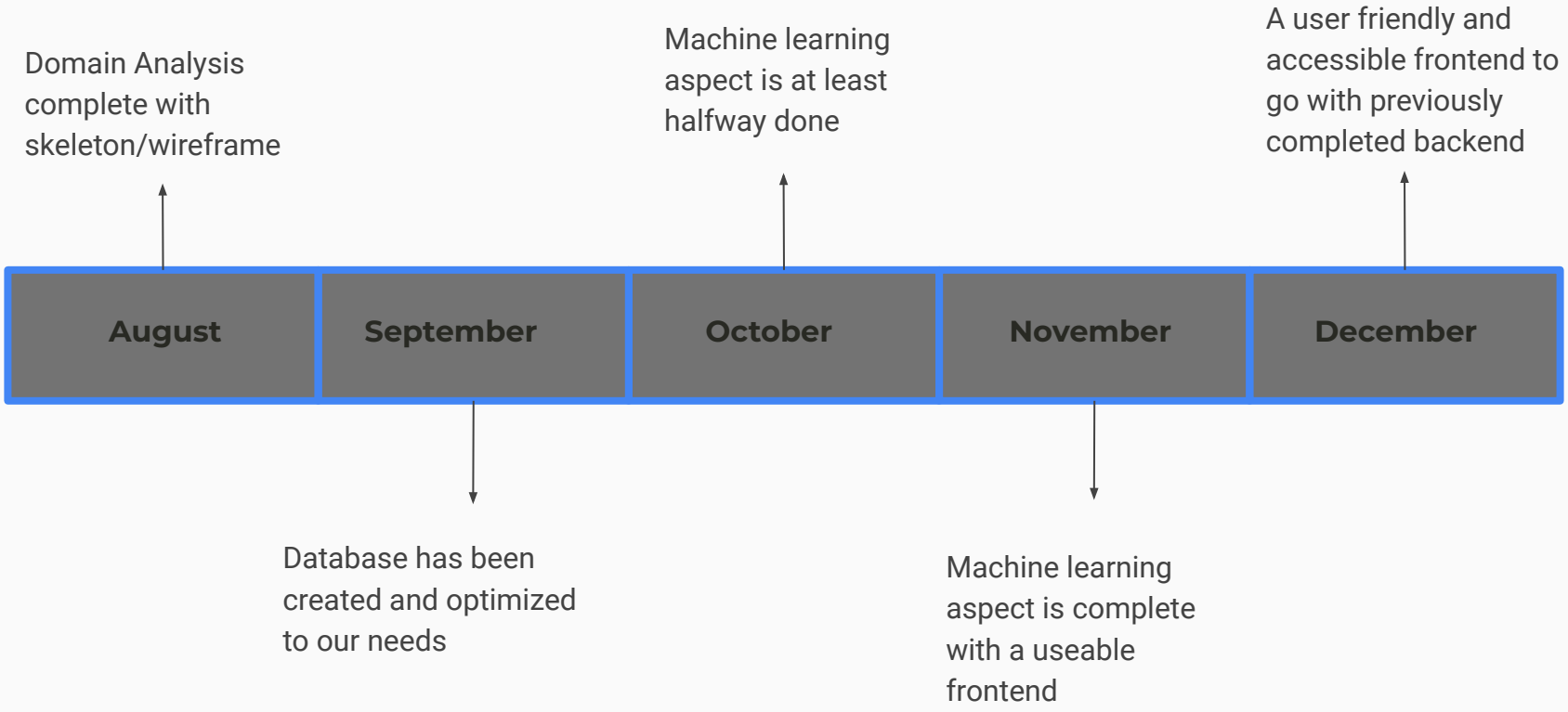
A movie recommendation web-service

By Janae Moring, Dom Mangano, and Ryan Webb

Origin Story – Janae’s Pitch

The problem: We’re bombarded with content whenever we turn on our tv’s, so much so that we may not even be able to choose what to watch.

The solution: A webapp that takes into account your own personal tastes and holds a collection of your watched movies and feeds you recommendations.



Our Original Development Timeline

Participation Goals

- Janae
  - To challenge pre-existing skills in a more formal team environment
  - Gain a deeper understanding of HTTP protocols and client/server management
- Dom
  - Learning React in order to be versed in front and back end development
  - Learning Git Workflow and working on a project with other future engineers
- Ryan
  - Improve JavaScript web development skills
  - Build a strong team-based portfolio project

Project Goals

- Developing a user-friendly front-end UI that is easy to navigate and fun to look at
- Building a robust back-end that includes API’s and aspects of machine learning that can develop as the user continues to access the web app
- Learning and building the entire project using the MERN stack, which includes:
  - MongoDB database
  - ExpressJS
  - ReactJS
  - NodeJS

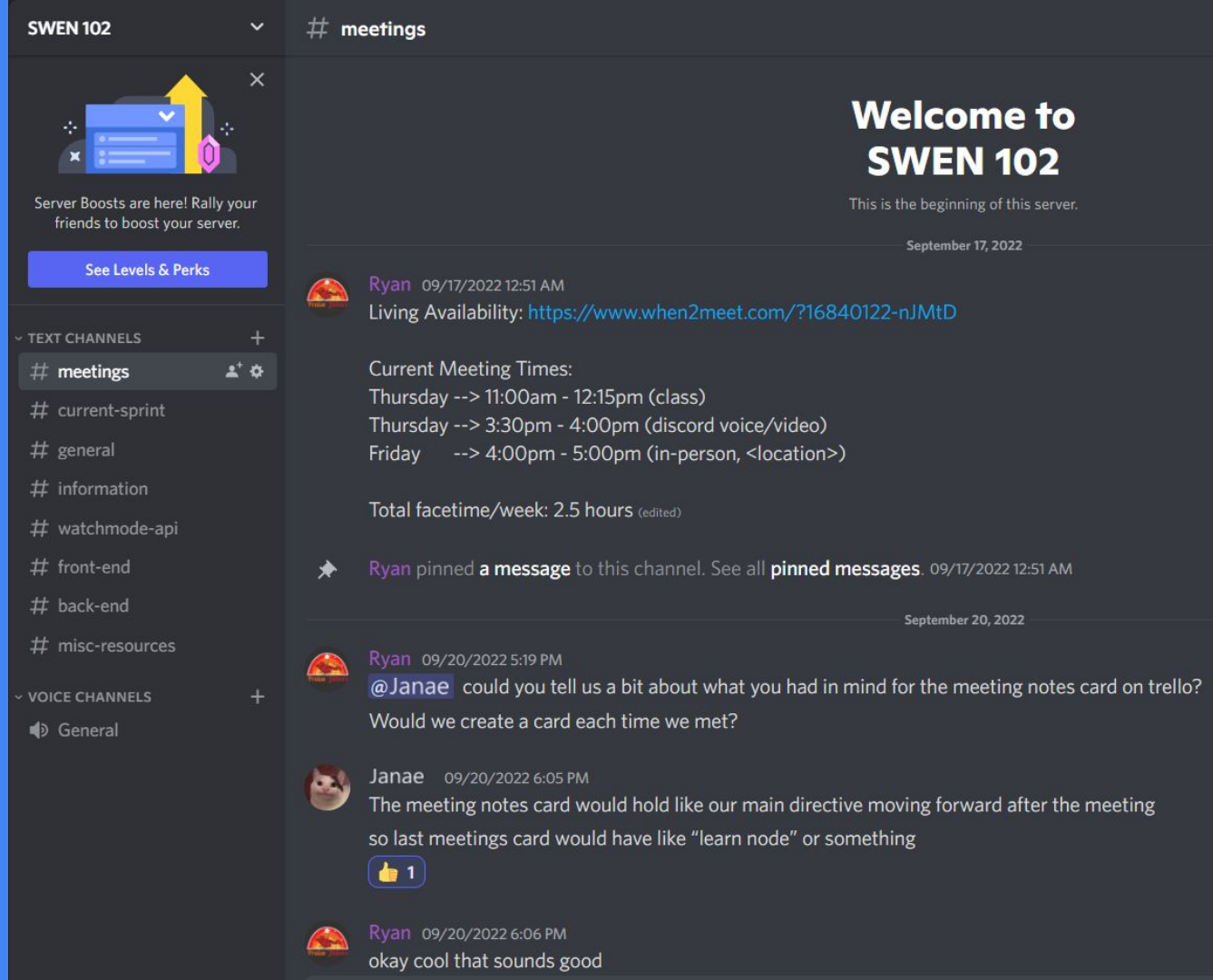
Lesson Learned

Keep scope simple and expandable

- Plan out layers of complexity
  - Allows scope to be adjusted easily during development
- Prioritize early, first-pass versions of major systems
  - Helps prevent integration hell
  - Facilitates team communication

Team Communication

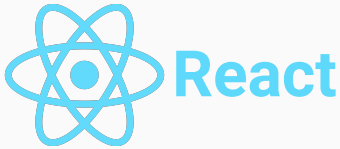
- Discord
  - Primary communication
  - Sprint information
  - Separate channels for each application layer
  - Support each other with links to learning resources
- Initially set up Trello
  - Hard to keep up with
  - Abandoned over steady communication on Discord



MERN Tech Stack

- Non-relational database
- JSON-like data storage

- Back-end web framework
- Used to build RESTful API



- Front-end UI framework
- Reusable UI components

- Back-end JavaScript server
- Lots of cool node packages

Other Technologies

- Movie database API
- Includes links to content

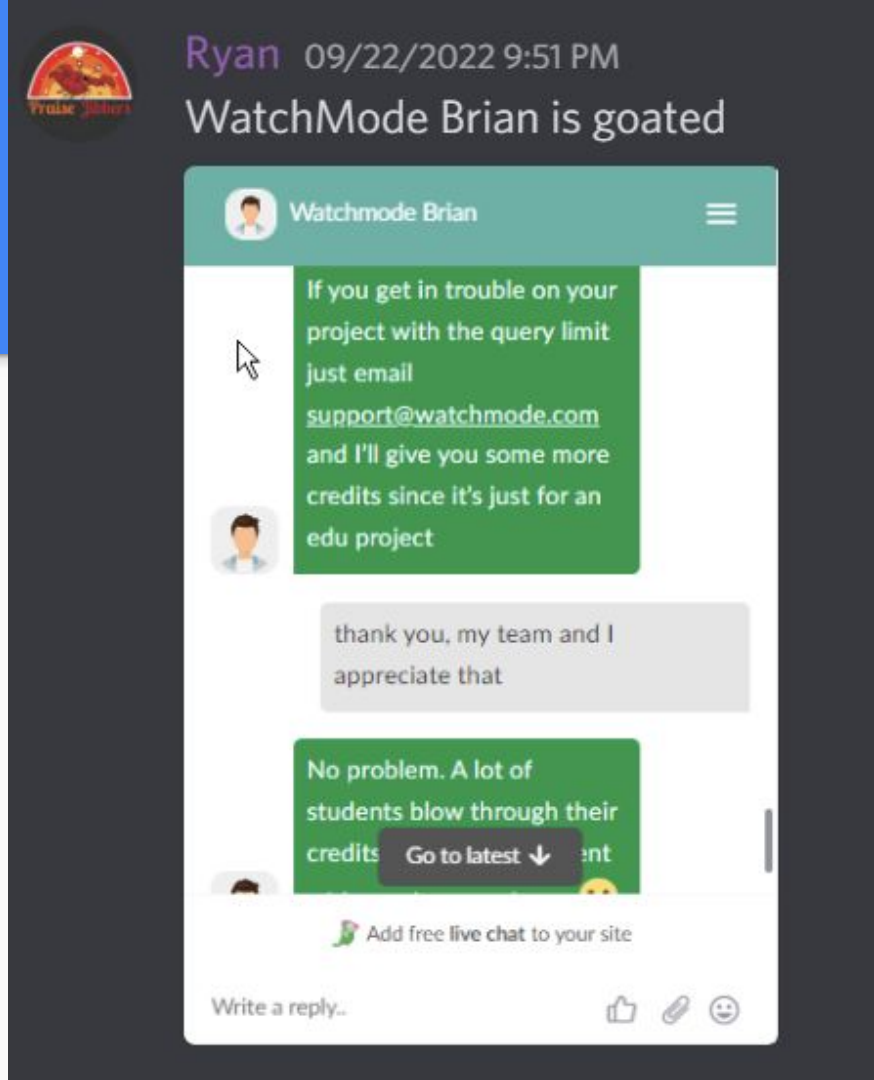
- RESTful API testing tool
- Save example responses

- Gets the job done.

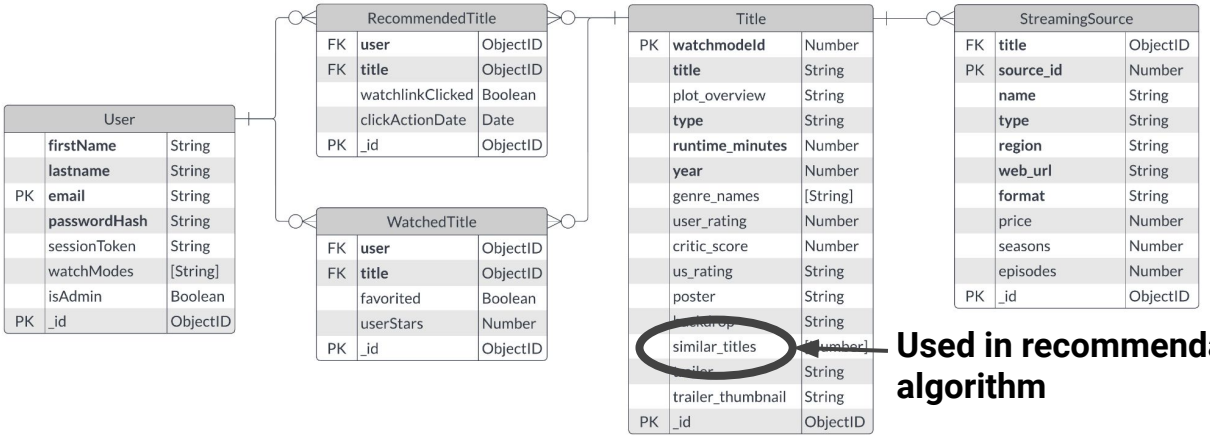
- Object document modeler for MongoDB
- Allows for elegant database interaction

Watchmode API

- Thank you Watchmode Brian!!!
  - Answered all my questions
  - So patient and helpful
- Cached API responses in MongoDB
- Most-used API endpoints
  - Title Details + Title Streaming Sources
  - List Titles
  - Autocomplete



Movi
Entity Relationship Diagram



Legend

Many to One Relationship

One to Many Relationship

FK: foreign key
PK: primary key

field: required
field: optional

The structure of this data is very relational, so best practice here is to use a relational database instead of MongoDB. Mongoose was helpful in applying rigid document schema.

Entity Relationship Diagram (ERD) – Made using LucidChart

Artifacts can quickly become out of date, updating takes time

Recommendation Algorithm

1. Create an occurrence map of similar title IDs

```
// Otherwise, get recommendations by mapping occurance      Ryan, last month • Second
// overlaps in titles similar to watched titles

// ### get a list of all titles similar to watched titles ###

// First, process a flat list of all title ids similar to titles the user has watched
let similars_all = watched.map(async (watched) => {
    return await watched.populate("title").title.similar_titles;
}).flat();

let idCountMap = {}; // Next, initialize an empty id count map (just an object).
// Increment the count of each id in the occurence list
similars_all.forEach((id) => {
    if (idCountMap[id] === undefined) {
        idCountMap[id] = 1;
    }
    else idCountMap[id]++;
});
```

2. Remove duplicates and sort by desc. occurrences

```
// create an array of unique title ids
let similars_unique = [...Set(similars_all)];

// remove titles that have already been watched by the user
// and titles that have already been recommended
similars_unique = similars_unique.filter(async (id) => {
    let title = await Title.findOne({watchmodeId: id});
    if (title == null) return false;
    return await WatchedTitle.find({title: title._id, user: user._id}) == null &&
           await RecommendedTitle.find({title: title._id, user: user._id}) == null;
});

// Sort the unique array by descending number of occurances in the map, and grab the top n
similars_unique.sort(id => 1/idCountMap[id]).slice(0, numRecommendations);

let titles = similars_unique.map(async (id) => {
    return await fetchTitleById(id);
});
```

3. Map each ID to its respective database entry

```
// Populate RecommendedTitle database collection      Ryan, last mo
titles.forEach(async (title) => {
    RecommendedTitle.create({
        user: user,
        title: title
    });
});

// Re-fetch all recommended titles to send
recommendedTitles = await RecommendedTitle.find({user: user});
recommendedTitles = recommendedTitles.map(async (recommended) => {
    return await recommended.populate("title").title;
});

res.status(200).json(recommendedTitles);
```

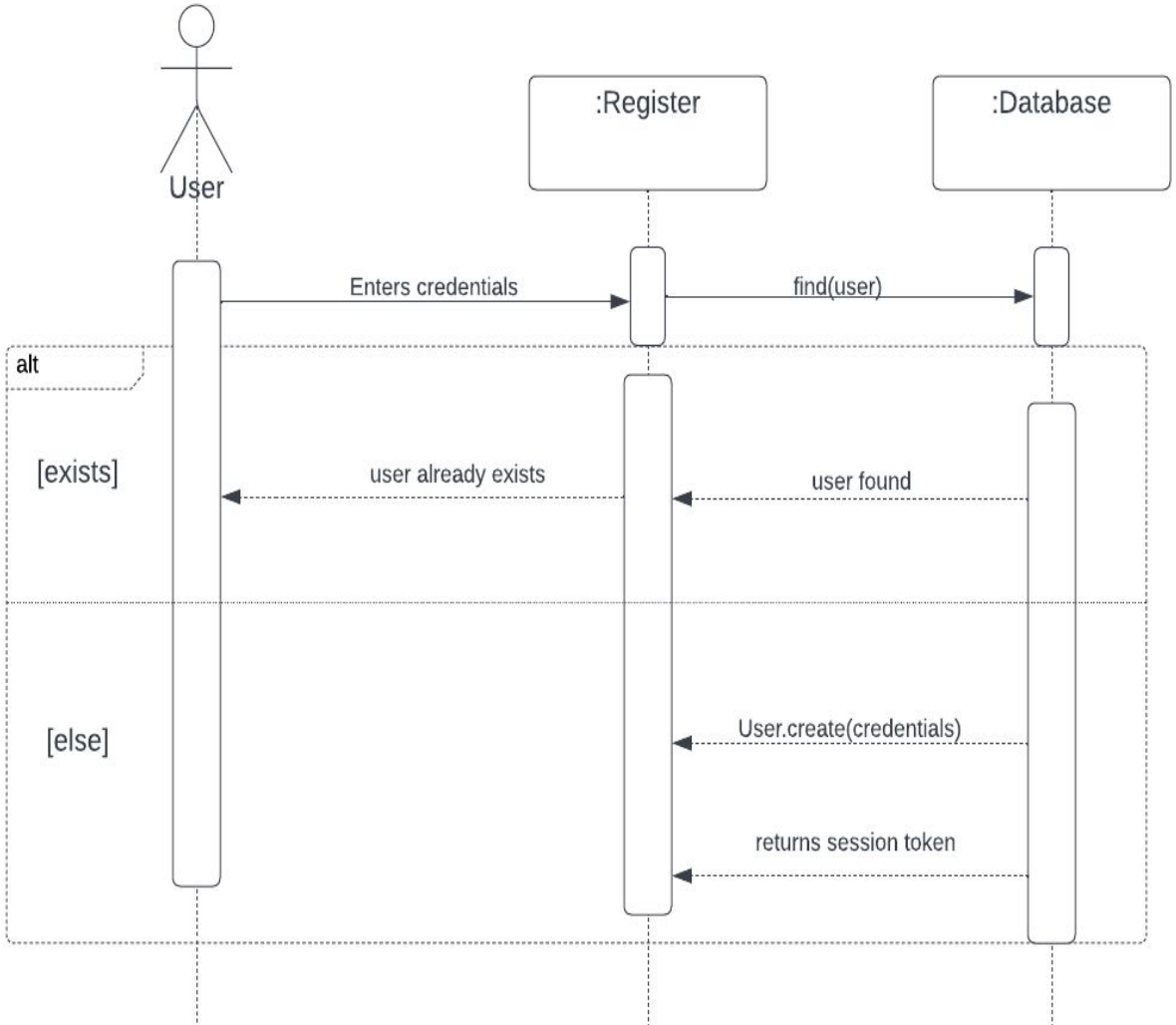
4. Return the newly recommended titles! 🥳

User Authentication

Instead of storing user data as plain text on our MongoDB server we decided to instead use CryptoJS and JSONwebtoken to securely store user data.

- All passwords on MOVI are encrypted and need a special passcode to be decrypted
- When a user is created the server automatically assigns the user a token and encrypts their password
- The token is refreshed every time the user logs out and/or every 24 hours

User Authentication

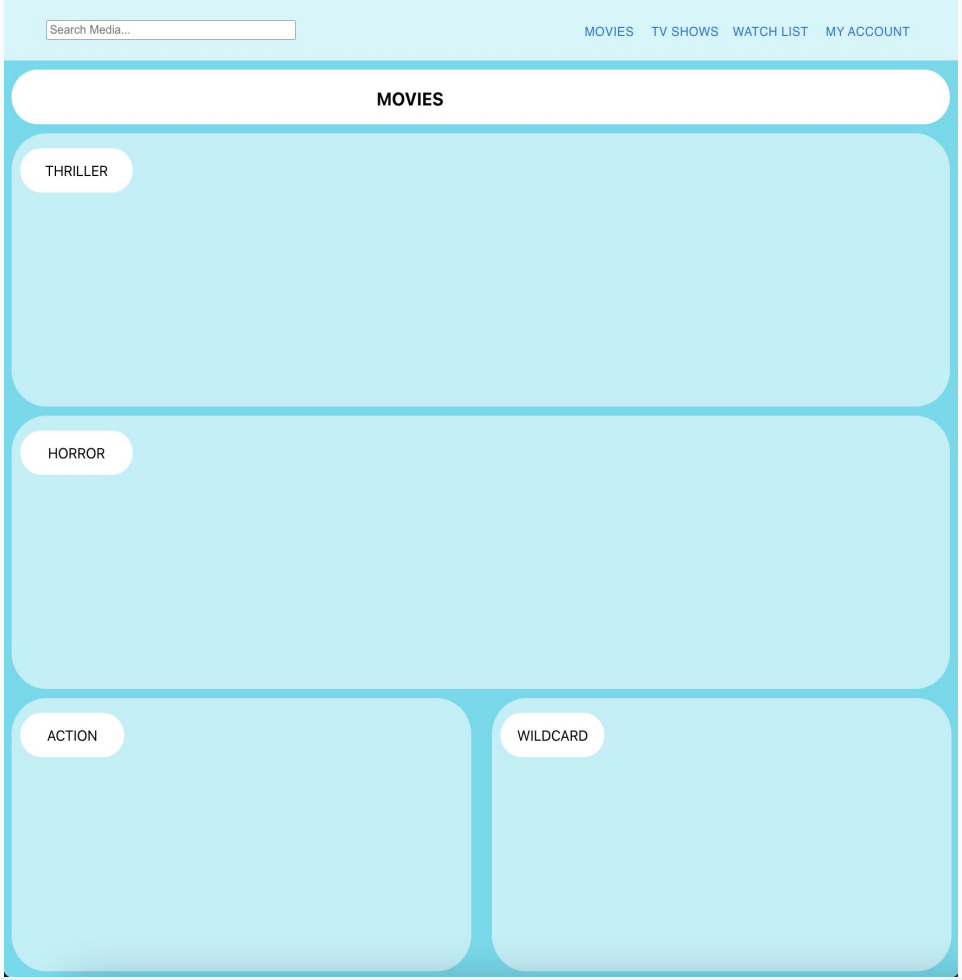


Front-end – ReactJS

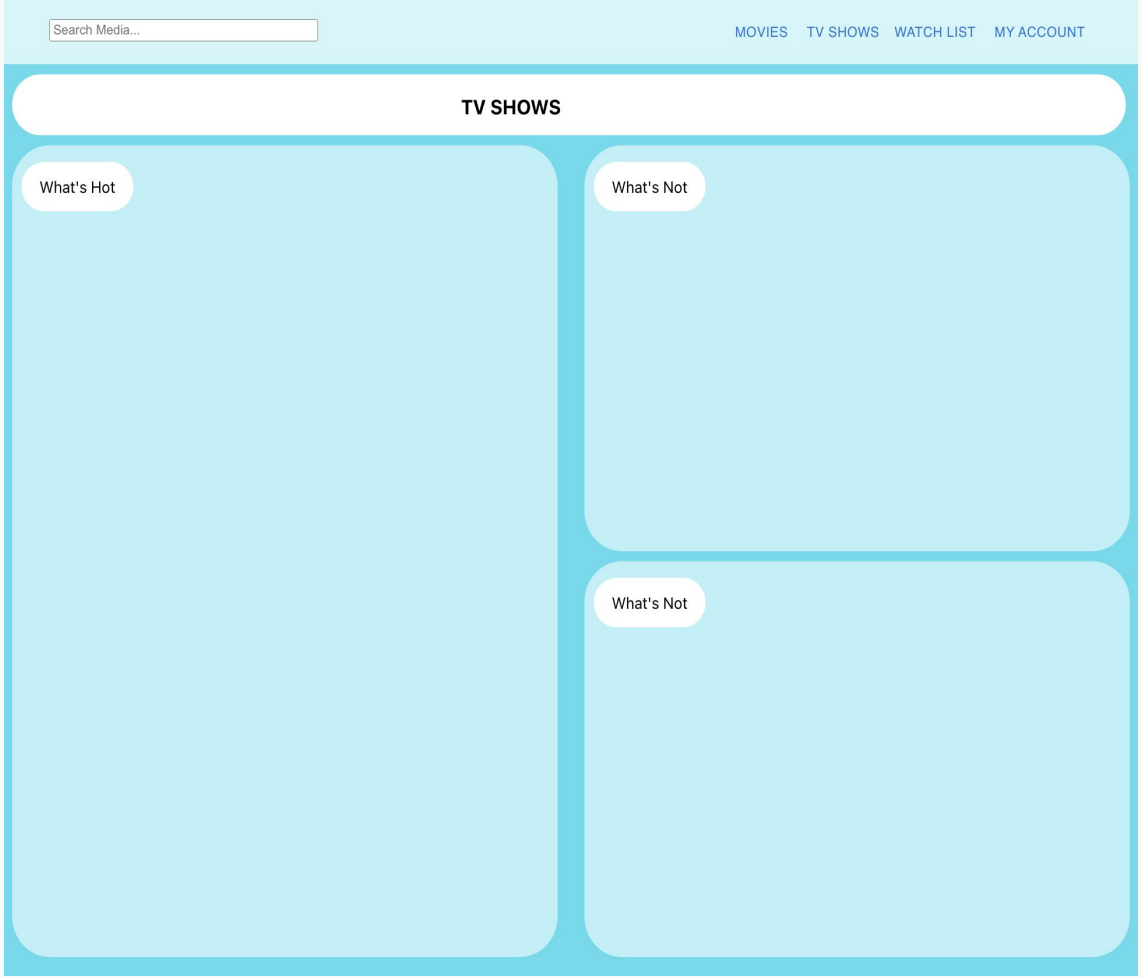
When using ReactJS, first writing the code with css and styling directly in the .jsx file was the easiest way to build the page without a ton of prior knowledge of React.

However, once it came time to split the styling into their own CSS files and major elements of each page into components, that's where React began to shine as a language.

Front-end Discover Page Wireframe



Front-end Genres Page Wireframe



Front-end TV Shows Page Wireframe

Example Components

```
import React from 'react';
import { Container } from '@mui/system';
import Header from './components/Header';
import { Helmet } from 'react-helmet';


function TVShows() {
  return (

  <div>
    <Helmet bodyAttributes={{style: 'background-color : #50dcee'}}/>
    <div>
        <Header/>
    </div>
```

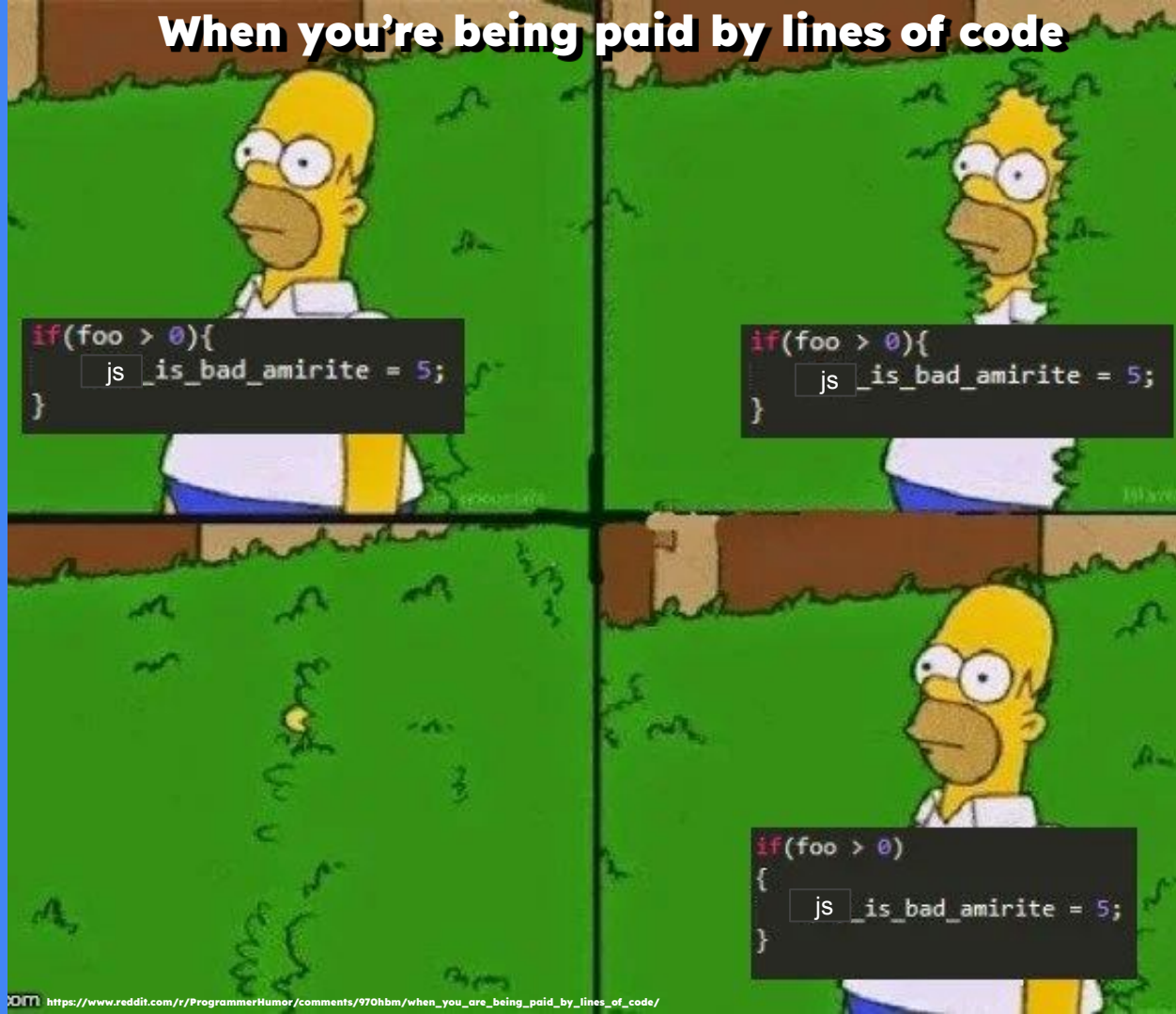
```
import React from 'react';
import SearchBar from './SearchBar';
import { Button } from '@mui/material';
import { Container } from '@mui/system';
import './HeaderStyle.css';

const Header = () => (
  <h1 className = "HeaderStyle">
      <Container className = "SearchBarStyle">
        <SearchBar/>
      </Container>

      <Container className = "ButtonStyle">
        <Button> Movies </Button> <Button> TV Shows </Button>
            <Button> Watch List </Button> <Button> My Account </Button>
      </Container>

  </h1>
);

export default Header;
```

Integration Hell

We mentioned earlier, *Integration Hell*...

A Few Metrics

- 1848 lines of code
  - 993 lines of back-end code
  - 855 lines of front-end code
  - Not counting auto-generated files
- 306 Watchmode API calls
  - Purely from testing
- 424 API credits used
  - Via endpoint chaining
- 87 git commits
  - Across four branches



Project Outcomes

- Built a capable back-end RESTful API with 13 endpoint routes.
  - REST stands Representational State Transfer - Stateless, where it accepts client information as requests, rather than maintaining active context of the client
- Integrated back-end with a normalized data model to store movie and user info
  - Also known as caching
- Created the foundation for a modern and modularized front-end UI
  - Industry standard front-end development library

Personal Outcomes

- We’ve built lasting relationships with one another after battling through this semester in both this project as well as in the SE program.
- We hope that our paths will cross again where we can be teammates on future projects.
- We all now have a strong building block for our resumes when it comes time to begin the co-op application process.

Future Goals

- Janae
  - Learn SQL databases and adapt the project to that model instead
  - Add a way for users to filter based on what streaming service they have
- Dom
  - Become more fluent in front end development using React so that I can build websites in a more efficient manner
  - Learning and practicing backend languages so that I develop into a full stack developer
- Ryan
  - Learn more about devops (deployment, continuous integration, continuous deployment)
  - Learn TypeScript and practice front-end development