Homework 2

"AJAX and data for JavaScript Applications"

The following problem set is worth 100 points total. Please submit a zip file containing all relevant code to the Canvas drop box by the due date of the assignment.

Requirements

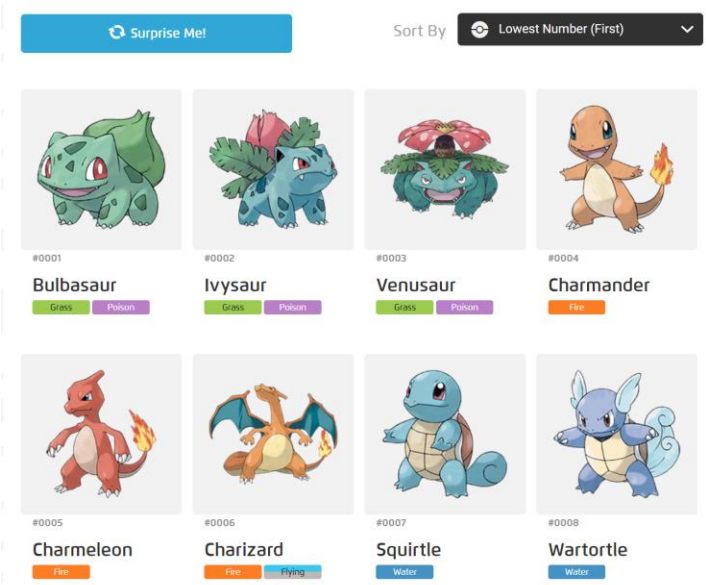
For this problem set, you are ingesting data from publicly available APIs. With this data you are producing web applications. Of the five problems listed below, implement **three** of the problems. Each program is weighted equally.

Problem 1: Pocket Monster Problem

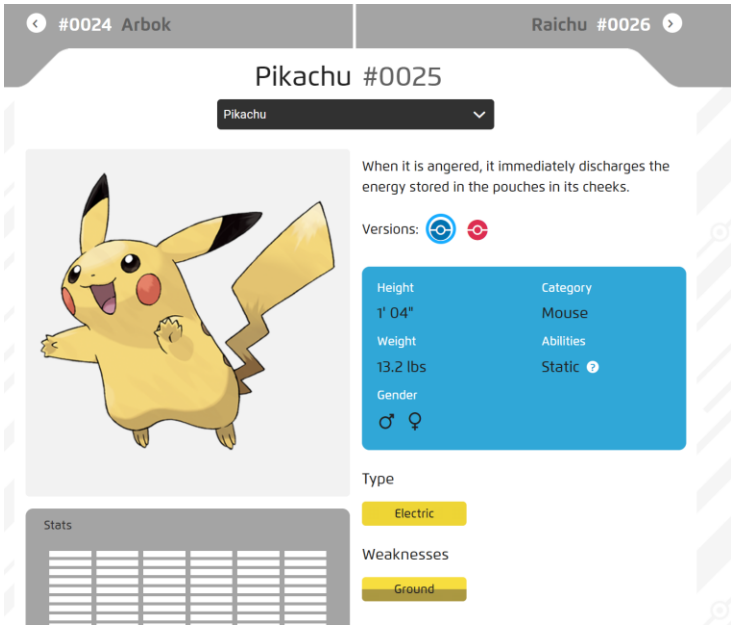
The following URL contains a JSON data set of **pocket monsters**.

https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json

Using this URL and JavaScript AJAX techniques, implement a web-based *Rolodex* for the **pocket monsters**. As an example, you could show all pocket monsters like the following:



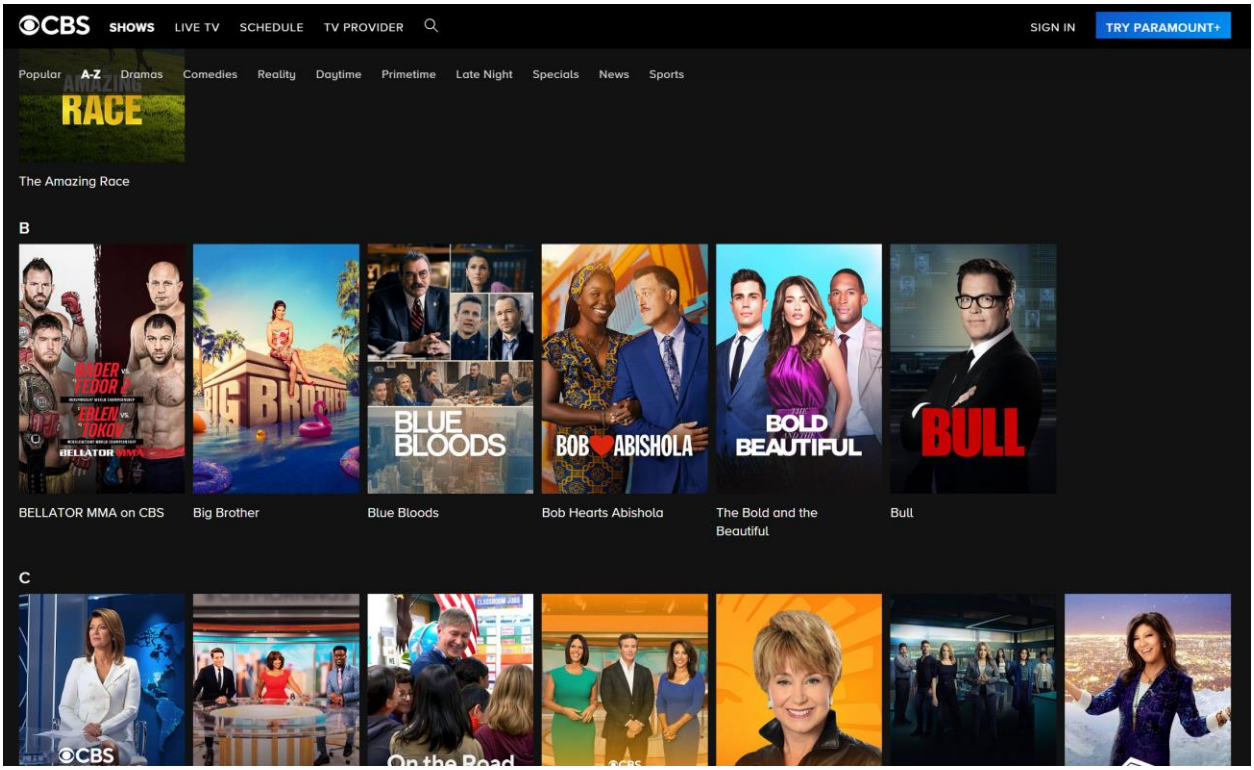
And when you click on one of the pocket monsters, you will receive detailed information about the pocket monster:



Problem 2: Television Problem

The following API exists for querying data for television shows: https://www.tvmaze.com/api

The API has a variety of services you can query, such as obtaining a list of television shows and obtaining a list of episodes for a particular show. Using the data from this API, produce an application where someone can search for shows and then from there search down and watch a particular episode (sort of how "net flicks" works.) As an example:



Problem 3: Weather Problem

Some end points might involve you creating an account to obtain a free API key. For this problem, I want you to create me a weather forecast screen similarly to how Bing does it:

You can retrieve data for your screen from **weather.gov**. Their API is documented here:

https://www.weather.gov/documentation/services-web-api

To retrieve the forecast data you'll need to make a call such as the following:

https://api.weather.gov/gridpoints/TOP/31,80

The response is a JSON data set containing information like the following:

```
  },
- maxTemperature: {
      sourceUnit: "F",
      uom: "unit:degC",
    - values: [
        - {
            validTime: "2019-02-01T15:00:00+00:00/PT11H",
            value: 11.111111111111143
          },
        - {
            validTime: "2019-02-02T13:00:00+00:00/PT13H",
            value: 11.111111111111143
          },
        - {
            validTime: "2019-02-03T13:00:00+00:00/PT13H",
            value: 14.444444444444514
          },
        - {
            validTime: "2019-02-04T13:00:00+00:00/PT13H",
            value: 3.3333333333333712
          },
        - {
            validTime: "2019-02-05T13:00:00+00:00/PT13H",
            value: 0
          },
        - {
            validTime: "2019-02-06T13:00:00+00:00/PT13H",
            value: -0.5555555555555429
          },
        - {
            validTime: "2019-02-07T13:00:00+00:00/PT13H",
            value: -5
          },
        - {
            validTime: "2019-02-08T13:00:00+00:00/PT13H",
            value: -1.6666666666666288
          }
      ]
  },
```


The endpoint breaks up data into *zones*. To retrieve the forecast zones you'll need for the above API call, you'll need to first make an API call to retrieve the zone data via latitude and longitude coordinates:

https://api.weather.gov/points/39.7456,-97.0892


```
properties: {
    @id: "https://api.weather.gov/points/39.7456,-97.0892",
    @type: "wx:Point",
    cwa: "TOP",
    forecastOffice: "https://api.weather.gov/offices/TOP",
    gridX: 31,
    gridY: 80,
    forecast: "https://api.weather.gov/gridpoints/TOP/31,80/forecast",
    forecastHourly: "https://api.weather.gov/gridpoints/TOP/31,80/forecast/hourly",
    forecastGridData: "https://api.weather.gov/gridpoints/TOP/31,80",
    observationStations: "https://api.weather.gov/gridpoints/TOP/31,80/stations",
  - relativeLocation: {
        type: "Feature",
      - geometry: {
            type: "Point",
          - coordinates: [
                -97.086661,
                39.679376
            ]
        },
```


For your screen, the user should be able to change your weather grid by entering their zip code. You'll need to convert their zip code into latitude and longitude coordinates. *MapQuest* has an open API you can use for that data as well:

https://developer.mapquest.com/documentation/open/geocoding-api/address/get/

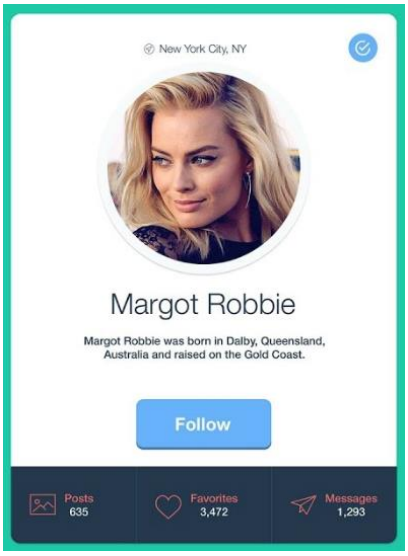
Feel free to use any image assets in your solution to your liking.

Problem 4: User Management Problem

Some web applications contain user management data. Thankfully there is an API available for you for generating random user data:

https://randomuser.me/

Using the data from the above API, arrange it in *card data* format. For instance:



The way you stack your cards (horizontally or vertically) is up to you. You can even use the card classes found in Materialize CSS or Bootstrap as well. Provide the user a way to filter on the user cards via an input tag, and your cards will *dynamically filter* as the user types input. Implement this feature using any JavaScript approach you feel fit.

Problem 5: Blogging Problem

Numerous web sites feature blogging platforms today, the most popular being WordPress. For this assignment you are to present a blog to a user, from randomly generated data via the following endpoint:

http://jsonplaceholder.typicode.com/

This API produces a number of endpoints you can use for retrieving blog post and even comment data:

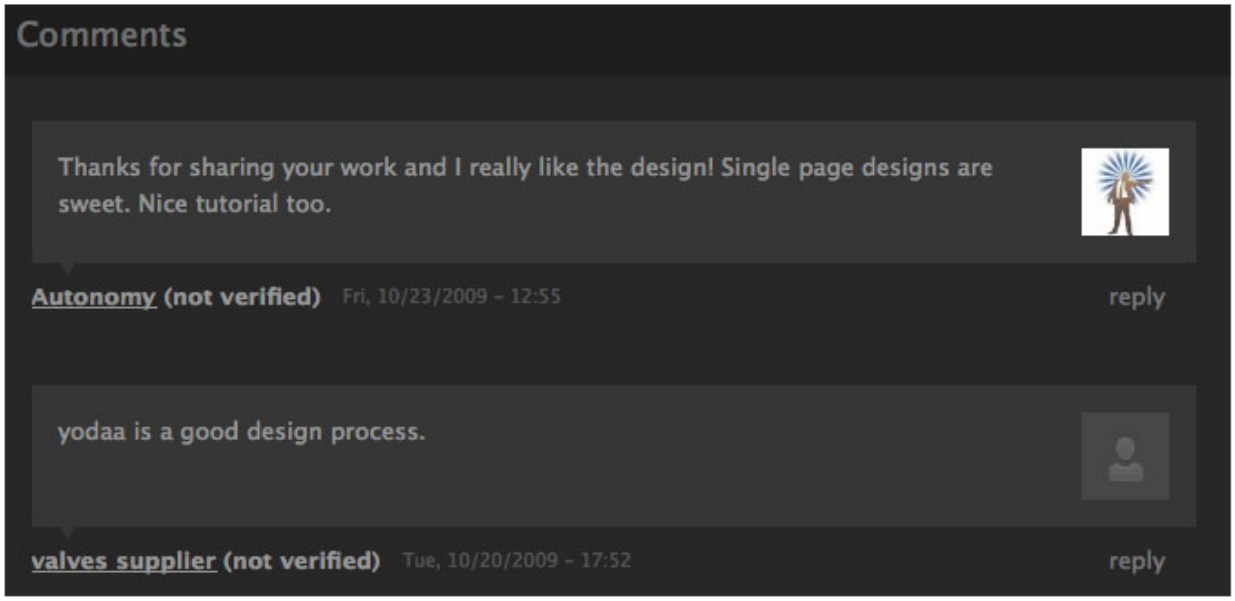
Routes

All HTTP methods are supported.

GET	/posts
GET	/posts/1
GET	/posts/1/comments
GET	/comments?postId=1
GET	/posts?userId=1
POST	/posts
PUT	/posts/1
PATCH	/posts/1
DELETE	/posts/1

Note: you can view detailed examples here.

Using data from the above endpoints, produce a blogging interface any way you like. Implement blog posts and comments into your solution. As an option, when the user first views your application you list all the blog posts and their headlines. When the user clicks on a headline, you can show the user a detailed page with comments. As an example:



Submission Requirements

For the submission, submit a zip file with all assets needed for me to run your applications. Feel free to use any technique used in class. Submit using the Canvas drop box before the due date.