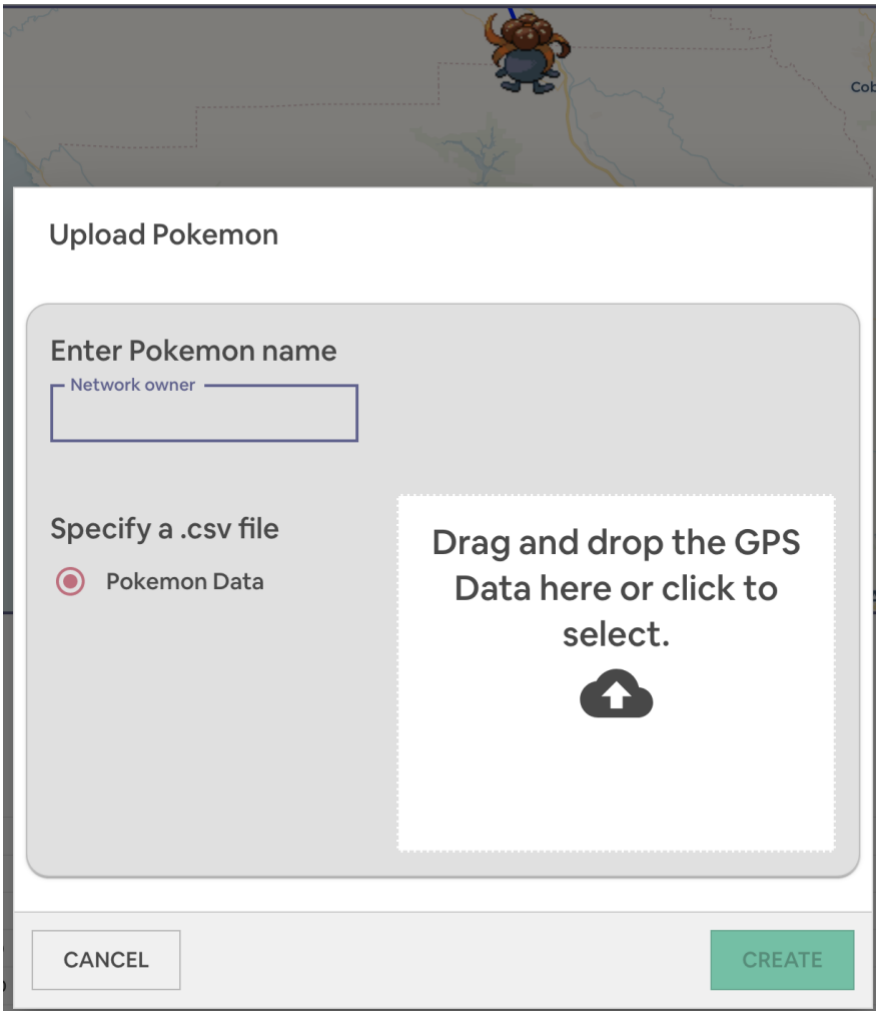
Pokemon Finder Mini Project

Project Objective:

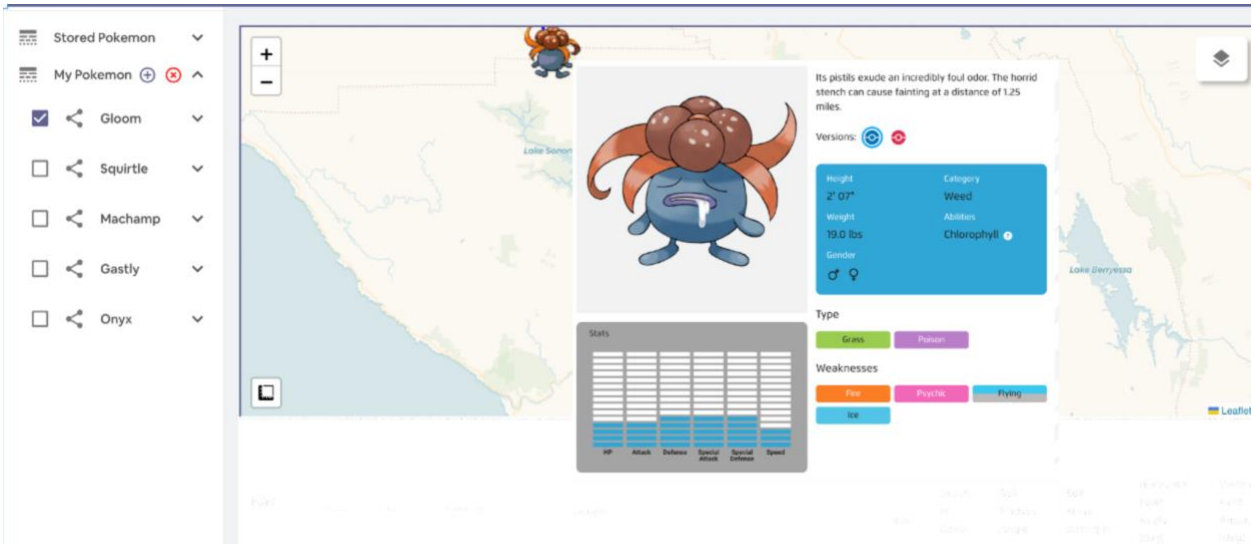
Build a Full Stack React application that utilizes a Django Rest Framework API. Display pokemon as markers on a map of California, randomly distributed but grouped by type. Implement an upload feature to add more pokemon to the map.

App Expectations:

1. Users must be able to create an account and login. Without logging in, the user should not be able to see the main page.
2. Once the user logs in, they should be redirected to the main page where a map will be displayed.
3. A parsing uploading tool should be on the main page, it should be able to parse CSV into JSON and upload it to the API. Here is an example of a dialog popup tool: Once the user clicks on create, the pokemon should be saved in the database along with data provided in the CSV. Please see below for an example of the CSV. We will test with an example CSV like the one shared within this drive.

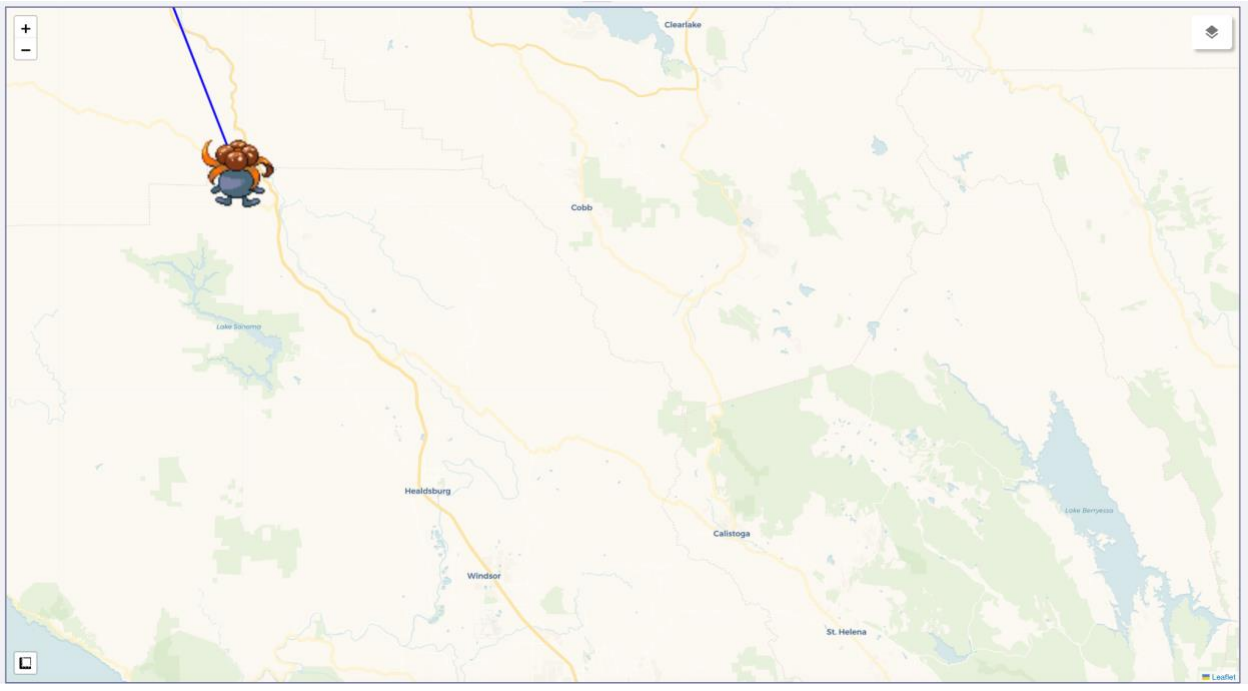


4. When a SPRITE on the map is selected, a popup should be displayed of the pokemon and their stats(Example Below)



Also an option to delete it from the list should be added to the user if they want to delete that Pokemon.

5. The Marker SHOULD BE a sprite image of that pokemon.



6. Create multiple Maps layers for the user to choose from as a base layer of the map (ex: satellite imagery, roads, topography, … etc)

7. Add a websocket in the popup of task 4 that shows the energy levels of the pokemon. The websocket data can be linked to the weather data api of the location of that pokemon, and an algorithm of your choice/creativity correlating weather and energy levels is used to display the energy levels changing. Assume a +/- 20% random variance every few seconds.
   For example, the sunnier the location, the higher the energy level.

https://openweathermap.org/api

8. Add a "How far am I from home?" button, placed anywhere based on your creativity. When the button is clicked with a selected Pokemon, the distance between the Pokemon and UCLA campus is shown.
9. Deploy the app using Docker and if issues with deployment then run docker and nginx to server applications on local port 80.

Requirements:

● Fetch 100 pokemon using the PokeAPI: https://pokeapi.co/
● List view on left side bar should be paginated
● User should not be able to see the pokemon list view until they've logged in
● User should be able to logout
● You should assign all pokemon from the API a set of coordinates. Please refer to the documents in the google drive labeled (A-J, K-Z).

TIP: The labels refer to the first letter of the Pokemon name, so for example, if we have pokemon Gloom, then we should place gloom anywhere within the Polyline. Each file (A-J) and (K-Z) are both polylines. So each pokemon should be placed within their respective polyline.

● When clicking on a Pokemon List Item, the app should spawn a pop up dialog on the map, showing the pokemon's name, portrait and other metadata such as: type, list at least 1 location name from location_area_encounters and list the most 4 recent moves learned at level 60.
● User should be able to search pokemon through pokemon list
● Use Django for the backend
● Add ability for user to favorite a pokemon
● Use Material UI.

ALTERNATIVE MARKER:

Marker color should be mapped as such:

- Fire = red
- Water = blue
- Leaf = green
- Psychic = purple
- Ground = brown
- Rock = black
- Fighting = orange

- Normal = tan
- Electric = yellow

All other types should be gray. Pokemon who are of more than one type should be colored at random between the two types that they are:

ex) Geodude -> ROCK, GROUND (BROWN, BLACK)

- In this case, geodude can be represented on the map as either brown or black

For cases where the mix types involve a type listed above and a type not listed, the color should be of the type list above.

ex) Fearow -> NORMAL, FLYING (TAN, GRAY)

- In this case, fearow must be represented as tan on the map since normal is list above and flying is not

FAQ's

1. One of the requirements is as follows: "Fetch 100 pokemon using the PokeAPI"
-My understanding is that Pokemon data is initially generated from the CSV file with subsequent API calls to the PokeAPI which are stored in the DB.

-Does the requirement indicate that the app should have the ability to fetch 100 pokemon from the csv file or do we need to prepopulate the db with 100 pokemon in addition to the CSV upload capabilities?

A: Please just fetch according to the requirements. The uploaded pokemon should be separate from the pokemon fetched using the PokeApi

2. The polyline files
-What is the relationship with the Long/Lat in the CSV file with the coordinates in the Polyline files? My understanding is that we use the CSV file to store the long/lat in the DB for each Pokemon and I have a Psyduck confusion on where the polyline files come into play.

A: Yes the pokemon from the CSV will be separate from the pokemon fetched from the API. The polyline requirements will be only from the pokemon fetched from the API.

3. There is a location in both the CSV file and a location_area_encounter from the PokemonAPI. Which data source do we use to store in the DB?
-Assuming the CSV file contains the location, do we prioritize that?

-Assuming the CSV file doesn't contain the location, do we store a random location_area_encounter from the PokeAPI?
-There is a similar situation with the "Latest Moves" column where the PokeAPI also has a set of moves.

A: You can assume that the CSV will be complete. If there is data missing you can simply ignore that pokemon. Remember that the pokemon from the API should be separated from the pokemon that the user uploads. We treat the uploaded pokemon as a custom pokemon that the user uploads.