Programming Paradigms CSE 30332

Final Project

PokePortal

Developed By: Luciano Giannini

Introduction

The PokePortal is a full-stack web application built with Flask, SQL Alchemy, Bootstrap, Javascript, HTML, and CSS. The PokePortal gives its users an interactive experience to view Pokemon encounter locations in the Kalos region from Pokemon X & Y utilizing a network graph from the AnyChart javascript library. The map allows users to see what Pokemon spawn in specific landmarks through their journey in the game. In addition to a map page, the PokePortal contains a page to show the entire PokeDex for users to view more information about each Pokemon. Users are able to create an account to save which Pokemon they have caught in the game to add to their own personal collection. Finally, there is a search page to return Pokemon from your search query to view the information about them. If you are running this application for the first time you can run python3 app.py and load the local URL, and in the terminal, you should start to see numbers

printing which means that the application is fetching the data needed to serve the website. There is also a '/configure_region' endpoint that can be used to update the data fetched from the 3rd Party API used in this project. On this page you can type 'kalos' and the page will load a success on completion.

3rd Party API & Backend Data Layout

PokeAPI 3rd Party API

The data used in the PokePortal comes from the PokeAPI. This website offers a RESTful API for accessing extremely intricate objects constructed from tens of thousands of lines of Pokémon-related data. The API gives a ton of information about Pokémon, like their kinds, moves, skills, and egg groupings, encounter locations. In addition to Pokemon information, it also contains information about each of the regions and location areas that can be explored by the player.

Backend Data Layout

In the PokePortal codebase, all of the data from the PokeAPI is formatted into Python classes stored in the models folder. This project contains four Python classes: Location, Pokemon, Region, and User.

Location Class

The Location class contains an id, name, list of Pokemon encounters, and a URL for the location relative to the PokeAPI that was used to fetch the data. In the __init__ function for this class, the name and URL are passed into the constructor and set to the class attribute, and a call to a fetchData() method is made to fetch the location information from the PokeAPI. The location information from the PokeAPI will return the id, name, region, and areas. This area data contains a list of location areas that would be visitable by the player in the game. The areas contain a list of objects that have a name and a new URL for each of the location areas. Within the fetchData() method a new call is made to get_pokemon_encounter(area) passing in the area object. In this method, the URL is used to fetch the Pokemon encounters for this specific location, and the name of the Pokemon is

appended to the class attribute. After the data is fetched the data is pushed to an SQLite database in a pushToDB() method.

Pokemon Class

The Pokemon class contains an id (PokeDex number), entry number, name, encounter locations, height, weight, types, moves, URL, and sprites. In the __init__ function for this class, the entry number and id are passed into the constructor and set to the class attribute along with the base URL for Pokemon data from the PokeAPI. Then a call to a fetchData() method is made. In this method, the id of the Pokemon is appended to the URL and a get request is made to the PokeAPI. The data returned contains id, name, height, weight, types, moves, and sprites. This data also contained encounter locations but it was for all versions of the Pokemon games which would need sufficient parsing to get the data that was needed for the Kalos region. Since the locations for the Kalos region were already fetched and Pokemon encounters were already saved to the database, a quick query to the Locations table was able to return where players could find this Pokemon. This class also contained formatting methods for types, moves, and sprites to pull the information that was needed for the PokePortal. After the data is fetched the data is pushed to an SQLite database in a pushToDB() method.

Region Class

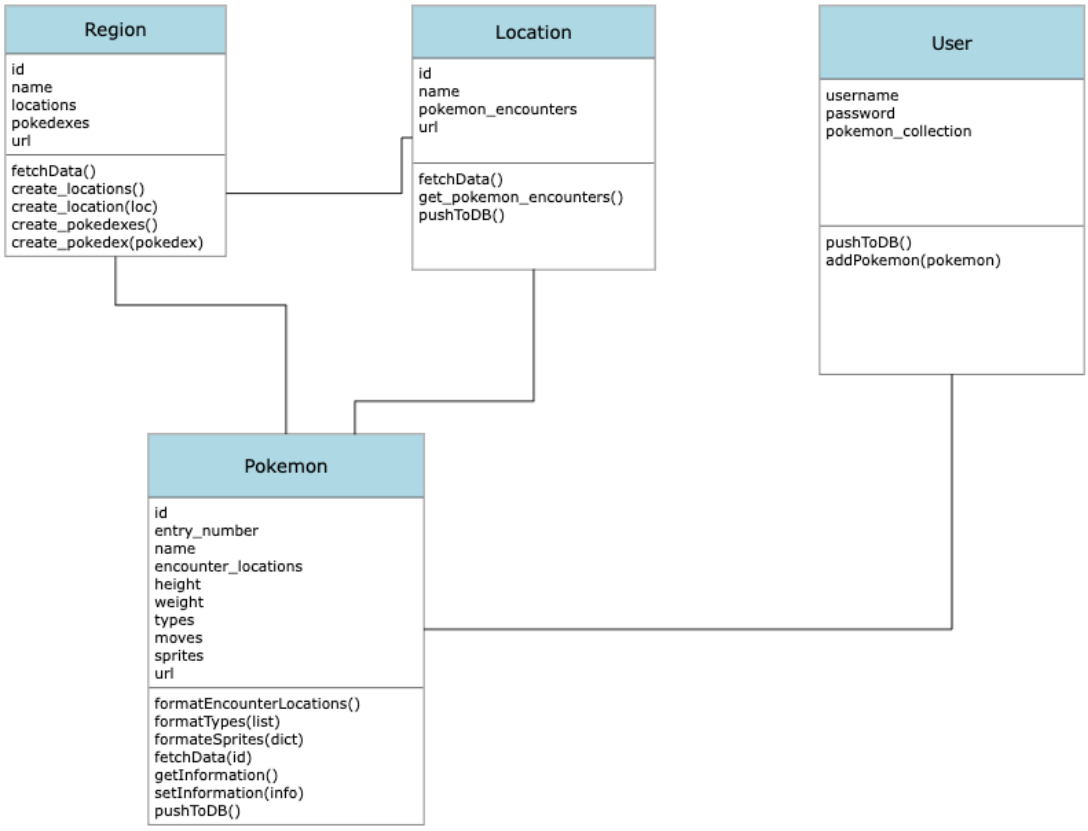
The Region class contains the id, name, locations, Pokedexes, and URL for the region relative to the PokeAPI that is used to fetch the data. This class is primarily used as the first initial data retriever for the website. It will fetch the region information, which contained the locations and Pokedexes, and uses this information to create Location and Pokemon that would be stored in the database. In the __init__ function the region is passed in and the URL is set and a call to the fetchData() method is called. In this method, a get request is called on the URL and the id, name, locations, and Pokedexes are saved to the class attributes. Then two methods are called to create the locations and Pokedexes, create_locations() and create_pokedexes(). The locations contain a list of objects that contain a name and URL from the PokeAPI. In the create_locations() method a for loop is created to loop through each of the locations in the list and create a Location passing in the name and URL. The Pokedexes contain a list of objects that contain the name and URL of the Pokedex. In the create_pokedexes() method a for loop is created to loop through each

of the Pokedexes in the region a get request is made for each Pokedex. This data returned contains Pokemon entries that are iterated through to create a new Pokemon passing the entry number and Pokemon name.

User Class

The User class contains username, password, and Pokemon collection as attributes. In the __init__ function the username and password are passed and saved to the class attributes along with saving a empty list as a string to the Pokemon collection. Then a push to the database is made to save the user to the SQLite database. This class also contains an addPokemon(pokemon) method that will append the Pokemon name to the Pokemon collection for the user to keep a history of the Pokemon they have captured.

UML Diagram



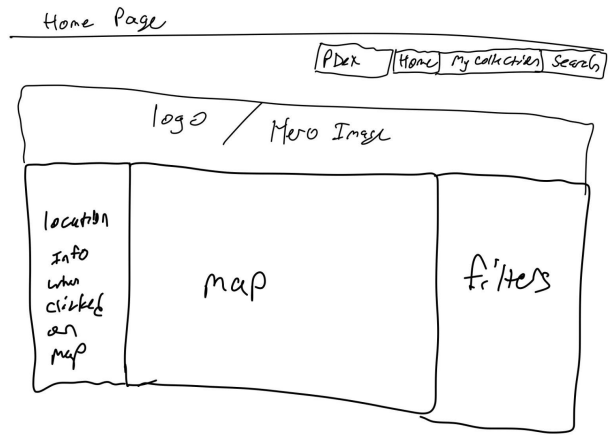
Database Schema

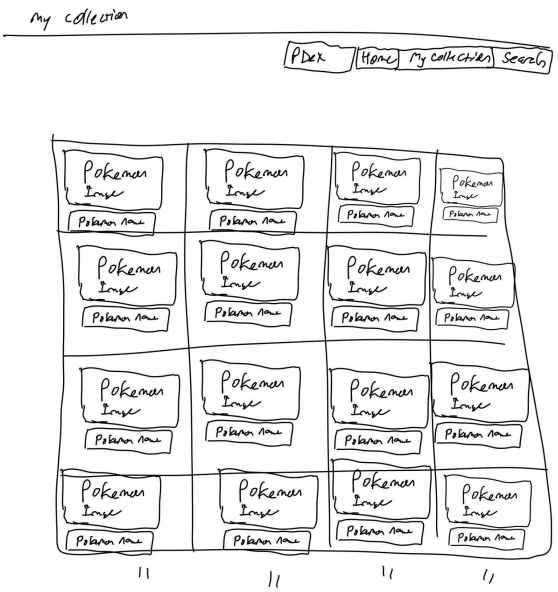
Location	User
🔑 id: Integer ♢ name: varchar(255) ♢ pokemon_encounters: varchar(255) ♢ url: varchar(255)	🔑 username: varchar(255) ♢ password: varchar(255) ♢ pokemon_collection: text

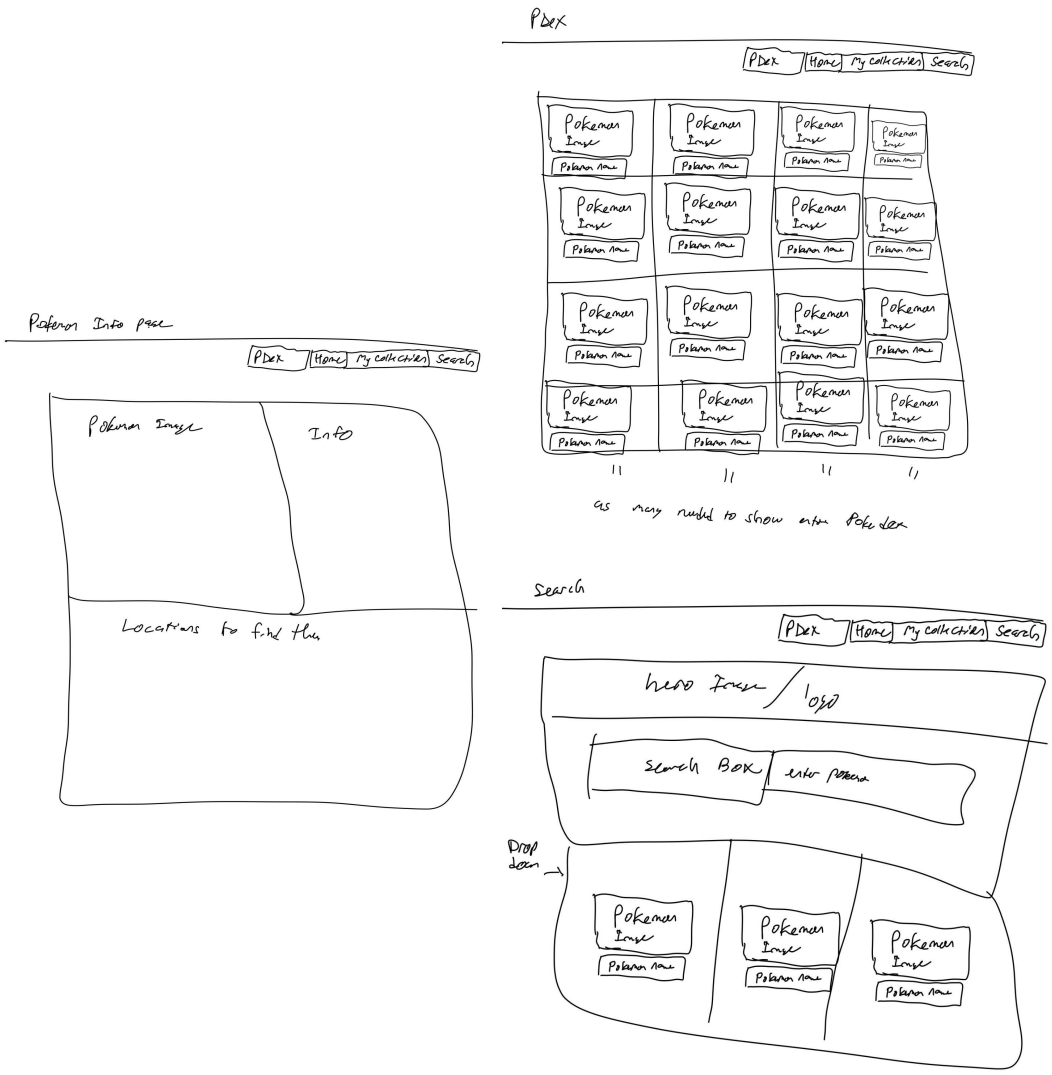
Pokemon
🔑 id: Integer ♢ name: varchar(255) ♢ entry_number: Integer ♢ encounter_locations: varchar(255) ♢ height: float ♢ weight: float ♢ types: varchar(255) ♢ moves: varchar(255) ♢ sprites: varchar(255)

Page Mockups and Final Result

Page Mockups





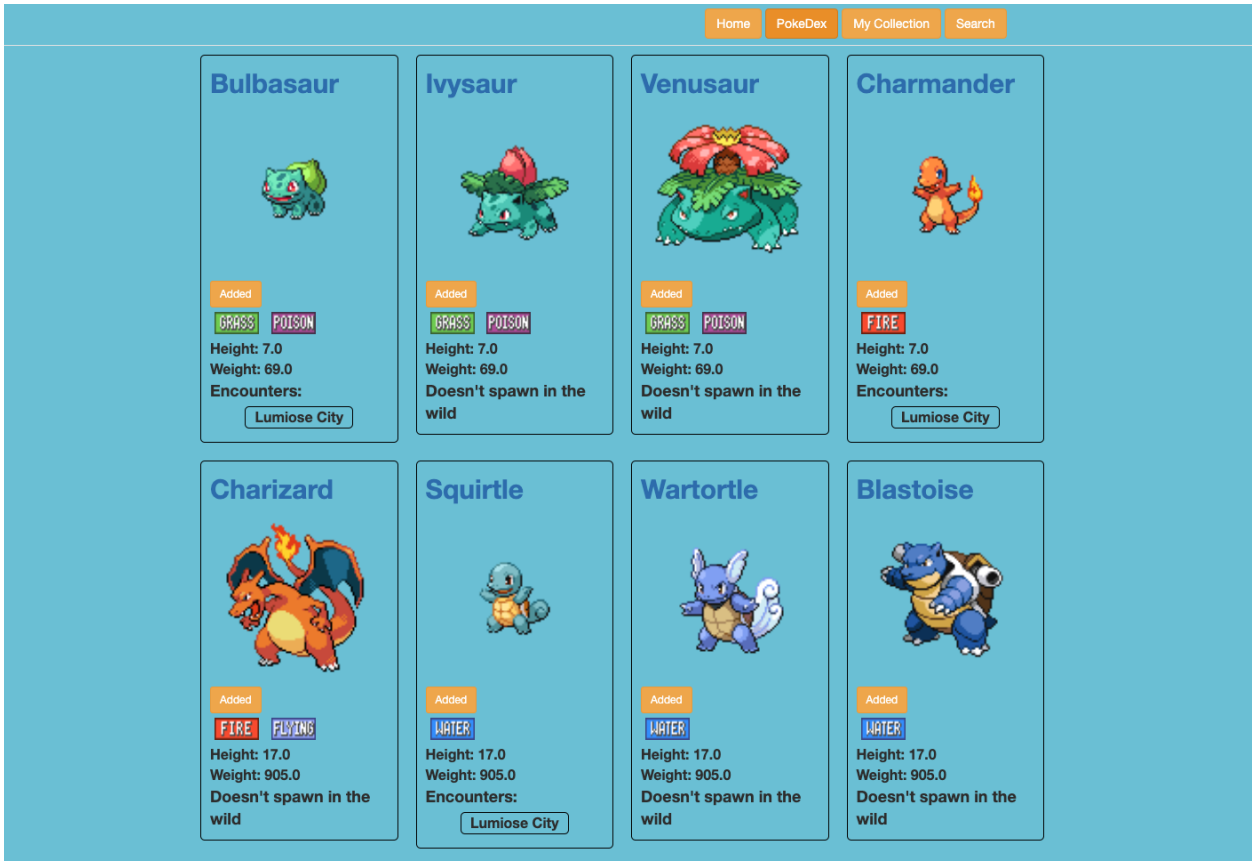


The page mockups compared to the final pages in the Pokeportal are rather similar. The search page is slightly different in containing a list of Pokemon rather than their image and name. The layout of the Pokemon information page looks a little different as well but the information detailed in the mockup is also on the final page. All of the pages contain a navigation bar to visit the different pages of the website.

/map

This is the home page of the PokePortal. It contains the hero image of the website with a Pokemon theme and the PokePortal name. Underneath the image contains the map of the Kalos region with a network graph with nodes representing each of the location areas a player can visit. If the user clicks on one of the nodes the left container will be populated with Pokemon spawns in that location. The names of Pokemon are clickable so the user can view their Pokemon information. On the left side, the user has the option to filter Pokemon encounters by type. When they click a type and press submit the page will reload and the user can click on locations to view Pokemon that spawn with that type.

/pokedex

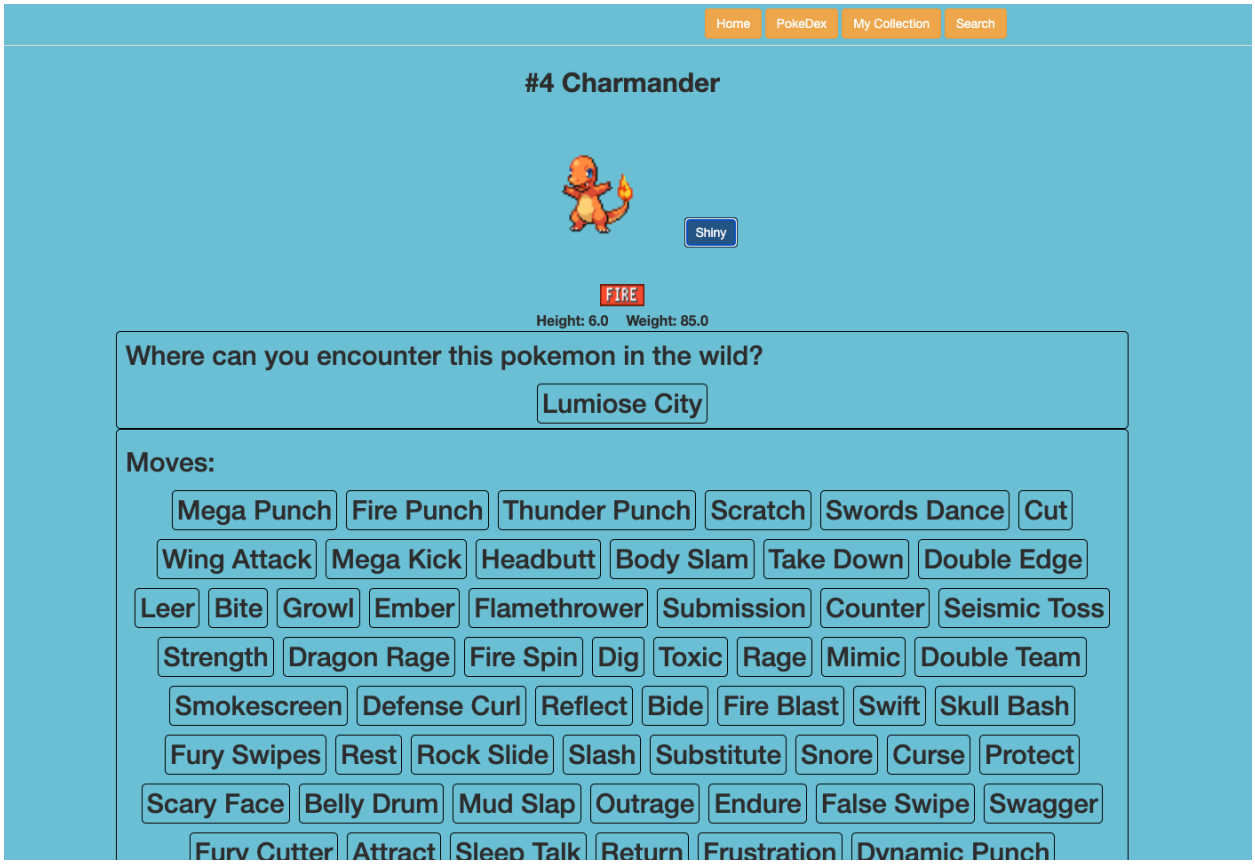


The Pokedex page consists of all the different Pokemon that are in the Kalos region. For each Pokemon you can see their image, type, height, weight, and where they spawn in the region. Also, there is a button to add the Pokemon to your collection. If the Pokemon is already in your collection the button will say ‘Added’. The name of the Pokemon is a hyperlink to that Pokemon's individual information page.

/my_collection

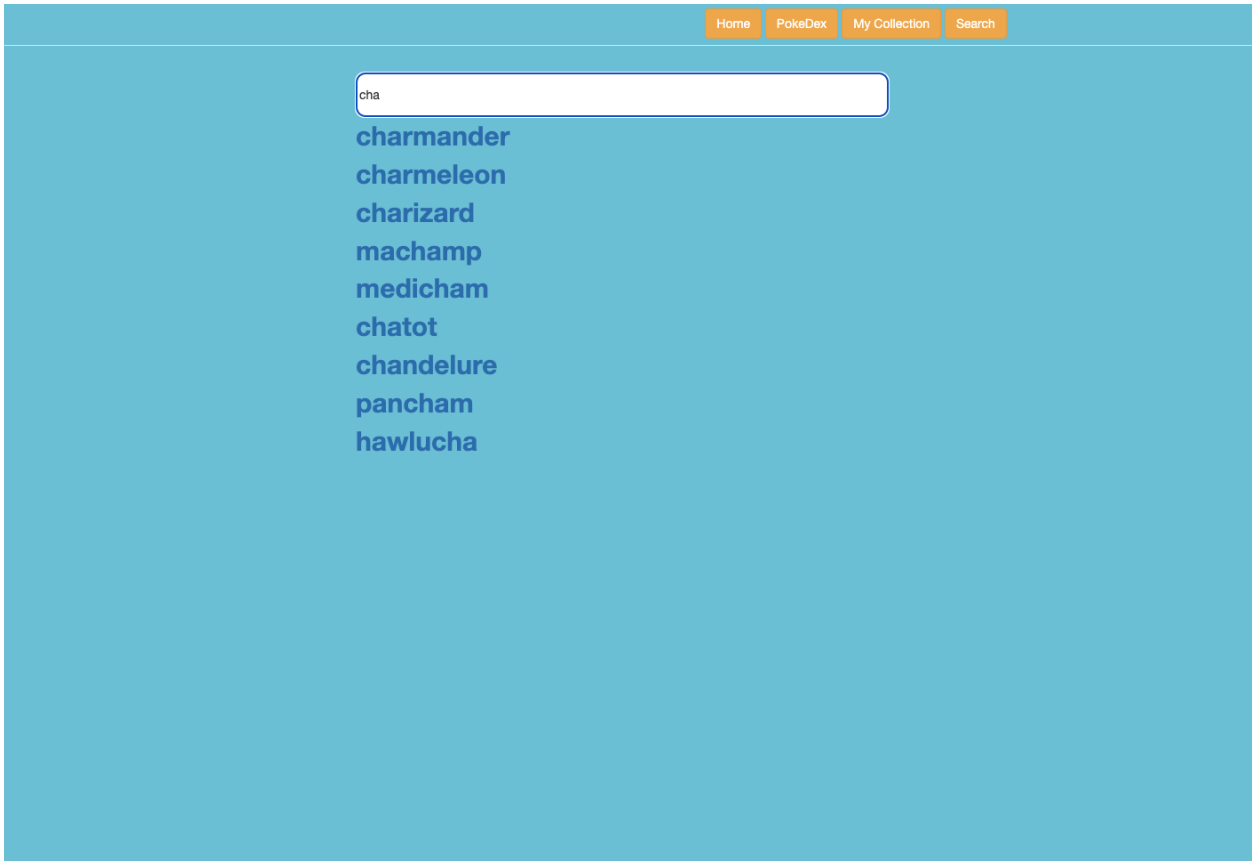
This is my collection page. This is where the user can view the Pokemon they added. The purpose of this page is to have a collection of Pokemon that shows the Pokemon that they have caught or registered to their Pokedex in the game. Again the name is clickable to view the Pokemon information.

/pokemon/<name>



This is the Pokemon information page. The user is presented with the Pokedex number and the name of the Pokemon. An image of the Pokemon is also displayed and a button ‘Shiny’ is next to it to display the shiny version of the Pokemon. Under that shows the type the Pokemon is and its height and weight. The next container, shows where a player can encounter this Pokemon in the wild. Finally, the user can see all of the moves this Pokemon can learn throughout the game.

/search



This is the search page. You can start searching for anything in the search box and Pokemon names containing your query will start to be populated in the drop-down menu. The user can click on these to view their information page.

Paradigms Used

Object-oriented programming and functional programming paradigms were used in the PokePortal codebase. For Pokemons, Locations, and Users each of these were represented as Python objects which contained data and methods to act on these objects outside of their own scope. Since the schema of my database was closely tied to how my objects were structured saving and retrieving data from SQLite was simple and concise. The objects returned from a database query were those exact objects that were created in Python so they had the attributes and methods that were expected. Some functional programming

paradigms were loosely used in my backend data code. In my Region.py file I tried to split my functions up so there was one expected return state for creating locations and Pokemon. In all of the classes, I use list comprehensions to have a clean and simplified code instead of creating for loops standalone.

Challenges

The main highlight of this project was to create an interactive map that the user could click to view Pokemon encounters at specific locations. AnyChart was the Javascript library that was used to create the network graph. Since the map was a simple image set to the background of the chart setting the positions of each location were the most challenging part of the project. My map.html code contains the many lines of creating the nodes and their specific x and y positions according to the graph that was generated. Utilizing some of my own skills with the developer tools in Chrome I was able to use the map in an editable version, position the points, and then inspect each point that had an attribute draw that contain the relative x and y components. After I did this for all the locations I reloaded the page and points were in the correct configuration but were pushed to the left and down. In my thinking, I was able to augment the position of each node in a loop by adding 10 to the x and -10 to the y after the nodes were already declared since they were hardcoded. But for some reason when I changed the x and y position after pushing to the data.nodes array nothing changed on the map, although their x and y positions have been changed when I inspected the elements. After playing around with this I tried incrementing the x and y at the place of declaration. But sadly the points didn't move but the x and y values changed. I remembered when looking at the AnyChart documentation there was one code example of the user being able to move the graph with your mouse. So there had to be a function to move the graph before it was drawn in the container. Which I eventually found, char.move(x,y) helped adjust the graph to center all the points to their location and line up with the background Kalos region image. The data from the PokeAPI was a lot so it took a bit of time to understand the format that was being returned and parse the data I needed for my website. Other than that my HTML and CSS skills were tested but I think it all came together and a useful full-stack web application was created.