Exploring Wildlife Sightings on Jefferson County Trails: A Database Web Application

Charlie Schlingman, Colin Wolff, Steven Nguyen

Date: 2nd, December 2025

CSCI 403 - Fall 2025 Professor Christopher Painter-Wakefield

Introduction and Overview

Jefferson County, located just outside of Denver, Colorado, is an extremely large and diverse place with hundreds of different species of plants, bugs, fish, mammals, and so much more. The diversity of this county is shown in its diverse ecosystems that are present throughout the entire stretch of land, which spans over 750 square miles. This county, being the doorstep to the Rocky Mountains, can be an extremely magical place when looking at species that it contains, but it can also be an extremely daunting location, as people may not understand what species make up the wonderful landscape that we see. For this project, we hope to be able to deliver an interface that users can interact with that displays the sightings of certain species, as well as helpful information for the user to identify where the sighting took place, like the park and trail name. Additionally, we hope to be able to clean this data so that the user experience is as smooth as possible, optimize the queries that are used to allow for quick access to the database, and include helpful visualizations to the user that display what parks are closely related in terms of their sighintings, as well as an interactive map that displays where a species was spotted on a map of Jefferson County, with helpful indicators that allow for easy understanding of the area that a species was found in, encouraging users to explore these areas in hopes of seeing the species in person.

Dataset

In order to achieve this goal, we first started off by gathering a strong database that we could start with to build our software. Some of the information that we needed for this project was the sightings of species in the Jefferson County area, but also information about the place that the sighting took place, like the park name and trail name, as well as information about the species that was spotted, like name, class, and genus. In order to achieve this while also aligning with one of the options of the dataset, we decided to gather 4 different tables that contained information about the sightings of species, the information about the trails, the information about the parks, and the information about the species in the dataset. Achieving this would satisfy the requirement as it gives us 4 tables that can be joined together to get a large dataset of the sightings of species along with information about the location of the sighting. After researching a couple of different datasets that were available, we were able to gather our tables from two different primary sources.

The first tables that we got were the sightings table and the species table, the former keeping track of sightings of different species that users had seen in the Jefferson County Area, and the ladder containing the information of the different species that were in the dataset. These tables were retrieved as a subset of the iNaturalist Research-grade Observations dataset [1], which is a dataset that has over 130 million occurrences of users submitting sightings of different species that they found in the wild. The dataset is peer reviewed and requires observations to meet a set of scientific grade requirements, making it an excellent choice in terms of sources. For our specific tables, we narrowed down the sightings to ones that were found only in the Jefferson County area in Colorado, as this allowed for data that was not only relevant to our goal, but also reduced the amount of data that we needed to load and clean, as loading and cleaning 130 million different rows of data would be extremely insufficient and time consuming. After gathering the data, we were able to get two separate tables, one that contained the sighting information of a specific instance where a user spotted a species in the wild, and another that contained information pertaining to all of the different species that were spotted in the dataset, containing their name, class, genus, and other identifying information.

The second source that we used to gather our other 2 data tables was the Jefferson County Colorado Open Space and Trails dataset [2], which is a dataset that uses information from the official Jefferson Country trails and parks information to give users information about the trails and parks of the Jefferson County area. From this dataset, we were able to get 2 tables that had information that would be vital for us, being the information about the trails and the information about the parks, saved to both trails and parks respectively. In the trails table, we got information on the trail name, the park that it contained in, information about the mileage and length, and coordinates that the trail was at. For the parks table, we got information about the park name, the owner of the park, the website, the area in acres, the coordinates as well, and other information about the park.

With these 4 tables now in our database, we cross reference the sightings table with the parks table and trails table in order to get exact information about the location of the sighting in the Jefferson County area, which was extremely helpful for later steps of the project. Along with doing the first join, we were also able to cross reference the species information table with the combined tables, as this allowed for more classification of the species that were found, allowing us to group sightings of similar class or genus later in our project development. In doing this, we were able to satisfy the dataset requirement stated before, as a join of these tables resulted in over 200,000 different rows of sightings, with information of the location, time, and species that was observed for each sighting. All four of these tables represent entities with significant structure, as they all provide attributes of interest for the dataset that is helpful for the user to understand the sightings that were found.

While both of the datasets that we gathered were extremely reliable and helped us immensely in our project, there were some restrictions on the use of them. One of the restrictions that we can cross for the first dataset, the subset of the iNaturalist database, was that the database was extremely large and contained almost too much information, enough to make cleaning the data a must. Some of the species that were included in the dataset were classified as bacteria and were only found very sparsely through the entire dataset. However, due to the fact that it was one of the over 20,000 different sightings that occurred in the table, it was also included in the species information, adding more clutter to the dataset that was not needed. Another restriction that we came across was in the Jefferson County Colorado Open Space and Trails database, as there were some difficulties with categorizing the trails that were not a part of a specific park. When gathering the 2 separate tables that were within the dataset, it was found there were a few trails that didn't belong to an actual park, but instead were within the JeffCo Open Space, which was not accounted for in the original dataset. This caused us to also do a significant amount of cleaning to the data, as we did not want any rows to return NaN values for things like trail, park, or species. Thankfully, we were able to solve these problems through the Data Loading and Cleaning section, which will be discussed later in this report.

Outputs

Chosen outputs:

- Data loading and cleaning
- Data visualization
- Software
- Performance Tuning

Data Loading And Cleaning

Once we gathered the necessary data for this project, we had a disparate dataset, with park and trail data, as well as observation data, but no way to correlate the two and collect useful results. It became clear it was necessary to perform some preprocessing before data was loaded into the database. The data itself was not extremely messy, and did not require extensive cleaning aside from this preprocessing, however we did omit certain columns during loading if they were simply GBIF-related metadata, or some other data not relevant to our task.

The schema we produced for this project had five total tables. The table parks stores parks, with one row per park, and geographic metadata stored as well known text. The trails table stores individual trails in the JCOS area, with one row per trail. This table has a park_id that points back to parks in the case of a trail being within a park. Another table, taxa, stores species information from the GBIF dataset, with one row per species. Sightings is the cleaned GBIF observations linked to parks, trails and taxa with derived spatial information. The last table, user_sightings, stores the sightings our app's users have recorded. These tables also have a few key relationships among one another. In trails, the field park_id is a foreign key to park.id, with the same for park_id in sightings and user_sightings. Sightings also has a foreign key for taxon_id, which references taxa.taxon_id. The explore page, the main interface in our web app, uses a view we created called explore_sightings which flattens joins across sightings, parks, taxa and trails for the queries on this page.

In order to create a dataset that had populated park and trail data for relevant sightings, we needed to use the geographic coordinates present in the observation dataset, and find some way of checking whether they were in or on a park. To accomplish this, we needed to get ArcGIS data for each park and trail. Thankfully, Jefferson County publishes this exact data on their ArcGIS hub. The trail data we used for this correlation can be found at:

https://data-jeffersoncounty.opendata.arcgis.com/datasets/00dca05ae60448d7ae2e24b6d086f485_0/explore

And the park GIS data is located at:

https://data-jeffersoncounty.opendata.arcgis.com/datasets/7b765b4fb1a34b198a08f9b62827821d_2/explore?location=39.661840%2C-105.241477%2C10.50

The actual process of correlation was done in python, working with the original dataset CSV files. In order to find the parks and trails associated with sightings, we first added a column that was simply a GeoPandas spatial point object that can be analyzed in combination with the GIS boundary data. Next, we turned our dataset into a GeoDataFrame, and performed a spatial join to correlate sightings with parks and trails. Below is a code snippet illustrating how the spatial join was performed:

```
joined = gpd.sjoin(
    wildlife_gdf,
    parks,
    how="left",
    predicate="within"
)
```

This produced a csv dataset with all the sighting information, as well as parks and trails filled in, with geometries stored in Well Known Text (WKT) format. However, this dataset did not contain data about sightings in parks or trails, it contained sighting data for a rectangular area roughly the size of Jefferson County. Due to this, out of a sightings dataset of 201,692 rows, 176,150 rows were not in a park boundary, and 186,890 rows were not near any trail. When a sighting was not in a park, or was too far from a trail (> 30 feet), the park_id and trail_id values are set to null.

When loading the data, in order to simplify the process, the raw csv data was loaded into staging tables, which had the exact same columns as the csv data. We used 4 staging tables to gather various types of data: wildlife_staging contained the original iNaturalist observation dataset from GBIF, trails_staging contained simple trail data from the JCOS Kaggle dataset, parks_staging contained simple parks data from the JCOS Kaggle dataset, and wt_staging contained the full park and trail correlated sighting dataset. Select columns of data from these staging tables were copied into new, final tables that our interface and visualizations would operate from. The final tables are as follows:

parks: Parks was sourced from the parks_staging dataset, the columns inserted were:

- park_name: text name of the park
- park_owner: text name of the legal owner of the park (e.g. City of Littleton)
- manager: text name of the recreation district that manages the park
- land_preservation_type: text describing legal preservation types
- website_url: text of the url for the park’s website
- website_photo_url: url for image data for park
- acres: double precision field representing area of the park in acres
- acquisition_date: text date explaining when a park was added
- geom_wkt: WKT format geometry data

trails: Trails was sourced from trails_staging, the columns inserted were:

- trail_name: text name of trail
- park_name: text name of park trail is in, only if in a park
- total_mileage: double precision length of the trail
- difficulty: text difficulty rating for trail ("Least Difficult, "More Difficult", "Most Difficult")
- user_type: type of allowed trail user (e.g. walking only, bikes, etc)
- total_length: double precision length in feet from GIS layer
- geom_wkt: WKT format geometry data
- park_id: integer, id of park trail is in, only if the trail is in a park

taxa: Taxa was sourced from the wildlife_staging table, it describes the various species in this dataset. The columns inserted were:

- taxon_id: bigint ID sourced from GBIF dataset
- taxon_class: text describing the biological class of the taxa
- taxon_order: text describing the biological order of the taxa
- taxon_genus: text describing the biological genus of the taxa
- taxon_species: text describing the biological species of the taxa
- scientific_name: text of the scientific name of the taxa, from GBIF
- species_key: bigint ID sourced from GBIF dataset

sightings: Sightings was sourced from the wt_staging table, the columns inserted were:

- gbif_id: bigint, unique identifier for each sighting provided in GBIF data
- taxon_id: bigint, unique identifier for each taxon, provided by GBIF data
- trail_id: ID of trail sighting occurred near, if present
- park_id: ID of park sighting occurred in, if present
- latitude: double precision latitude value
- longitude: double precision longitude value
- basis_of_record: text field describing how sighting was collected (almost all are marked as human observation)
- observed_at: timestamp type without time zone, time observation was made
- dist_to_trail_ft: double precision field generated when the spatial join was done
- species_key: bigint species identifier from GBIF, used to join to taxa.species_key when needed

user_sightings: User sightings was not sourced from any staging table, it stores any new sightings our users record. The columns present in this table are:

- id: integer, not null, primary key
- species: text data indicating species name
- observed_at: timestamp without time zone\

- park_id: integer field, collected from dropdowns in UI with parks and trails to choose from
- trail_id: integer field
- latitude: double precision
- longitude: double precision
- notes: text field for the user to input any notes
- created_by: text field to indicate who created a sighting

The spatial join with park and trail GIS layers produced a dataset with park and trail information filled in for the necessary sightings. The loading process took relatively unorganized data and turned it into a normalized database, ready for users to explore. The actual data loading SQL script can be found in Appendix A.

Data Visualization

For the data visualization section of the outcomes, we decide to do two different visualizations with the first visualization finding a better way to compare parks that had similar sightings, better categorizing them and seeing if location truly had an impact on the ecosystem, as well as an interactive map where users could input the desired species name and see sightings of that species, which are projected onto the interactive map of the Jefferson County Area.

To start, the first visualization that we decided to do was a visualization that allowed a more physical and distinct approach to categorizing parks in terms of sightings, relating them to the different ecosystems that the Jefferson County area provides and the distance between the parks that had these similar ecosystems. We decided that using a dendrogram would be the best approach for this, as dendrograms allow for grouping of similar statistics that a certain input has. This will be amazing for our project as it will allow us to get a visual chart of the similarities between gatherings, cross referencing it with the distance between the parks to see if there was a similarity. To start, the following query was used in order to get the count of a specific species as well as how many times that species was seen at each park:

```
SELECT
 p.park_name,
 ta.scientific_name,
COUNT(*)
```

```
FROM
sightings si
JOIN trails t
ON si.trail_id = t.id
JOIN parks p
ON t.park_id = p.id
JOIN taxa ta
ON ta.species_key = si.species_key
GROUP BY p.park_name, ta.scientific_name;
```

This query then allowed a clean table that gave us each park name, scientific name, and the amount of times that specific species was seen at each park. This gave us a good starting set of data, but in order to truly get it to form a dendrogram, we need to transform the data even more, pivoting it and then linking it. Pivoting the data means transforming it from each row containing one park, one species, and the amount of sightings that species had at that specific park, to each row corresponding to one park, with each column containing the sightings of a specific species, turning the data from over 5000 different rows with only 3 columns, each park and species combination and then the count, to only 27 rows and over 1300 columns, where each row pertains to a specific park and each column containing the count of that species. The code segment below takes our data and pivots it easily, making it ready for the next step:

```
pivot = test_d.pivot(index="Park Name", columns="Scientific Name",
values="Count").fillna(0)
```

The next step is the linkage step, which is where we turn the data into a form that is easily digestible by the dendrogram. A dendrogram model has to take data in a certain format so that it can be easily clustered together to compare different rows together to come up with the chart, which is why linkage is required. Linkage is a process in which the distance between two points is calculated, returning the points that had the least distance to each other. In this case, we are seeing which two parks are the closest in species sightings, with the two parks that are closest in species sightings being returned. Then, the process repeats, except the two points that were closest together are now combined into one, resulting in one less variable to account for in the comparisons, as two variables are combined. This then repeats until all of the variables have been combined, leaving us with an array that we can use for our dendrogram. Below is the code segment that computes the linkage:

```
linked = linkage(pivot, method="ward")
```

Now, we are able to build the dendrogram, setting the proper variables in order to let the visualization be properly formatted and computed:

```
plt.figure(figsize=(10, 6))
dendrogram(
    linked,
    labels=pivot.index,
    orientation="right"
)
plt.title("Park Similarity Based on Species Sightings")
plt.xlabel("Distance")
plt.show()
```

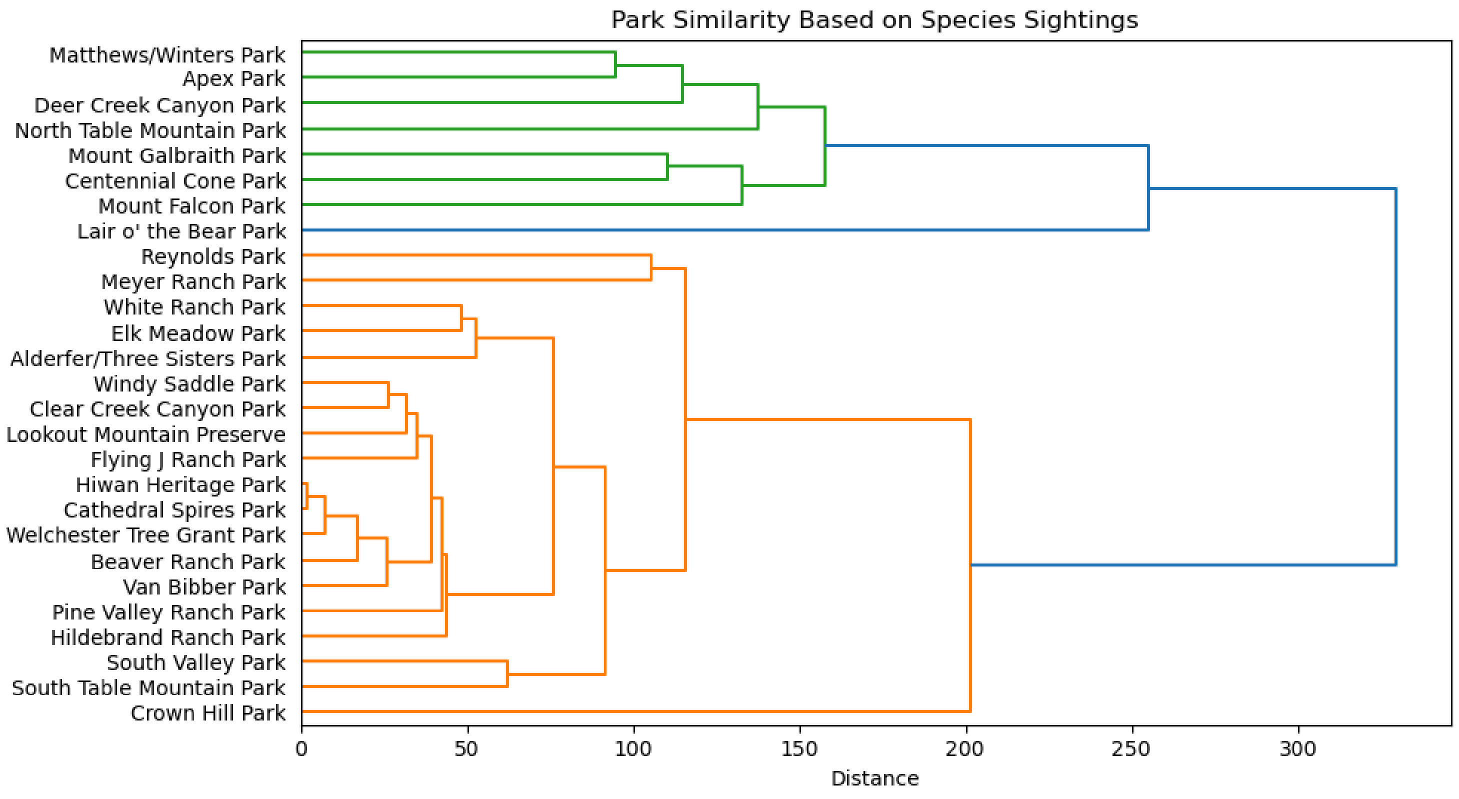


Figure 1: Dendrogram of Park Similarity Based on Species Sightings

The figure in Figure 1 was the result of the dendrogram, which gave us some really interesting results. You are clearly able to see that there are some parks that are much similar than others in terms of the types of sightings, with Hiwan Heritage Park and Cathedral Spires Park being the most similar, and some being wildly far apart, most notably the parks at the top of the chart, like Matthew/Winters Park and Apex Park. However, the most interesting discovery we can find is when we look at the actual distance between the parks that are closer together, and the parks that are further apart. For example, we decided to look to see how far apart Hiwan Heritage Park and Welchester Tree Grant Park were from each other from looking on a map, which we did by plotting a marker at the center of both parks on a map:

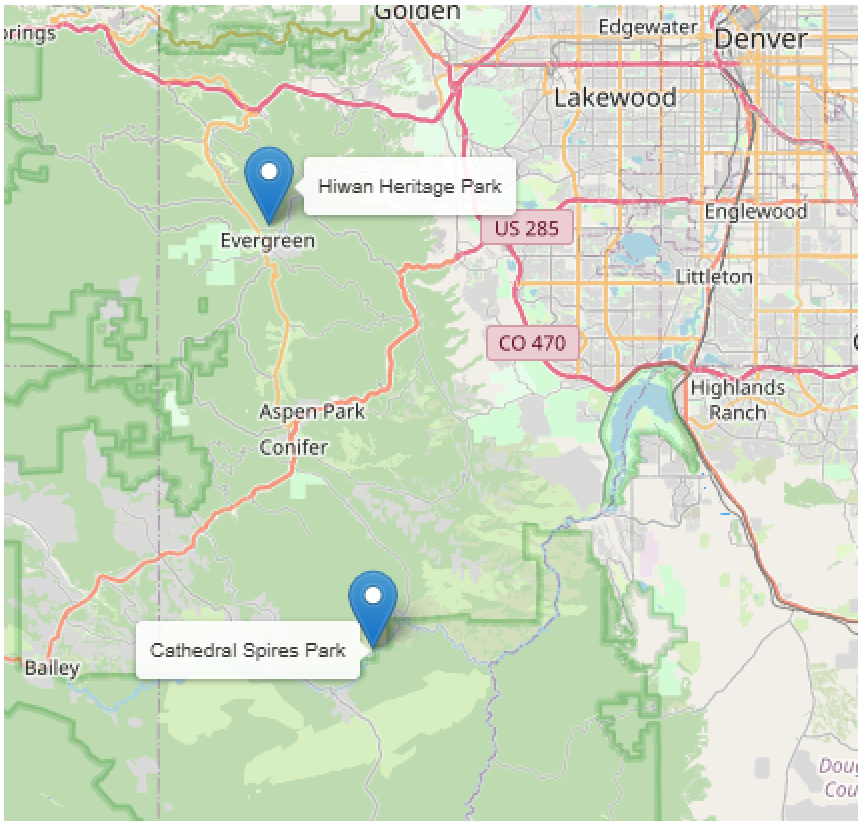


Figure 2: Visualization of Hiwan Heritage Park and Cathedral Spires Park Distance

The distance between the two of these parks does not seem like much, but it is actually about 20 miles apart, which in terms of these parks is quite large as the average distance between each park is only about 5-10 miles. The really interesting part was when we looked at the distance between two parks that were wildly different in terms of distance, with distances of over 100 between the two. An example we looked at was the distance between Matthews/Winters Park and Apex Park, who had the closest distance to each other, despite that distance being over 100. Based on this distance from the dendrogram and the physical distance between Hiwan Heritage Park and Cathedral Spires Park, you might think that the physical distance between Matthews/Winters Park and

Apex Park would be large. However, after plotting the two parks on a map, we see that this is not the case at all.

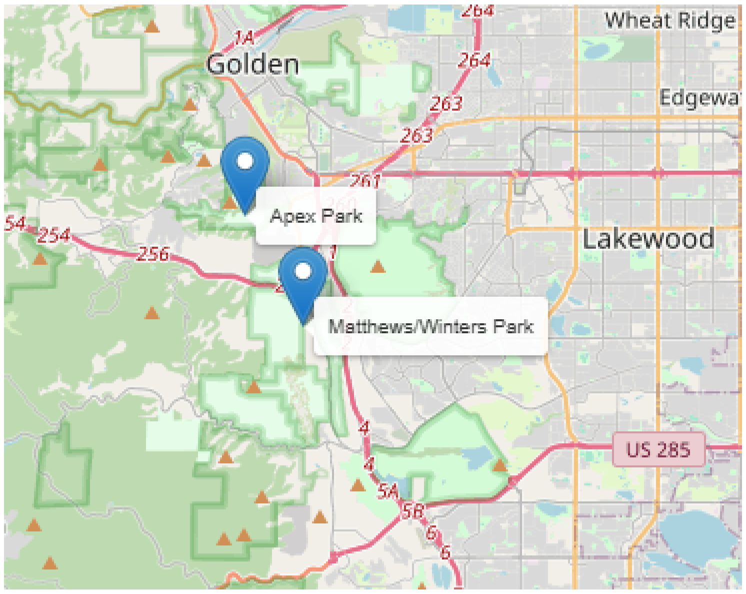


Figure 3: Visualization of Apex Park and Matthews/Winters Park Distance

Clearly, from figure 3, we are able to see that these two parks are two of the closest possible parks to each other, with the difference of the two being under 2 miles. The parks are so close that they are only separated by one of the major highways of Colorado, being I-70, which draws the border between the two.

The question that then arises from this is how are these two parks so different in species sightings, yet are so close together? And better yet, why are 2 parks that have over 10 times the distance between the two have the closest species sightings out of the almost 30 parks that were gathered? Well, the answer to these questions is surprisingly simple, human intervention. Looking back at the two figures, we can clearly see the geographical difference between the four parks. The first two parks, Hiwan Heritage Park and Cathedral Spires Park, are separated by open forest, barely any roads, small towns, and large pathways for different animals, fish, and even plants in the forms of their seeds to transfer freely. This allows for a more similar eco system, which is reflected in the sightings of each of the parks being so similar and close in distance in our dendrogram. However, when looking at the geography between Apex Park and Matthews/Winters Park, we can see a clear contrast compared to the previous two parks. Both parks are next to a lot of human intervention, with Lakewood and Golden being directly next to the parks, roads and highways separating the two, and loud traffic and construction being at the park's boundaries. This causes the species of these ecosystems to be effectively separated from each other, with very little access from one park to the other, making the sightings of different species, and thus the ecosystems of the parks, to be significantly different from each other.

The second visualization that we decided to generate from this project was a heatmap of sightings that the user selected, which was projected onto a map of the Jefferson County Area, along with markers that signified the location of different parks, allowing for the user to discover a park where a certain species was spotted at, or see what parks to avoid if a certain species was sighted there that the user could be allergic to or scared of. To do this, we first loaded a query where we prompted the user for a species name, inputting that into a query that gathers the latitude, longitude, and count of a that specific species:

```
SELECT
si.latitude,
si.longitude,
COUNT(*)
FROM sightings si
ON trails t
ON si.trail_id = t.id
JOIN parks p
ON t.park_id = p.id
JOIN taxa ta
ON ta.species_key = si.species_key
WHERE ta.scientific_name = %s
GROUP BY si.latitude, si.longitude;
```

This query then returned every single position that the inputted species was spotted at, giving us a clean data table to work with. The next step was to then input this onto a Folium map, which is an interactive map that allows the user to move, scroll, and interact with the map. We first start by centering the Folium map on the average latitude and longitude of the data table.

```
m = folium.Map(location=(df["Latitude"].mean(), df["Longitude"].mean()))
```

Then, we added a heatmap to the Folium map, which plotted each sighting by the latitude and longitude coordinates that were provided, displaying this onto the map.

```
heat_map = test_h[["Latitude", "Longitude", "Count"]].values.tolist()
HeatMap(heat_map).add_to(m)
```

Lastly, we added in a marker that was plotted at the average sighting longitude and latitude of each park, which gave us a pretty good marker of the center of each park. To do this, we first got a query that gave us every single park name, and the average latitude and longitude of the sightings at each park, grouping it by the park name.

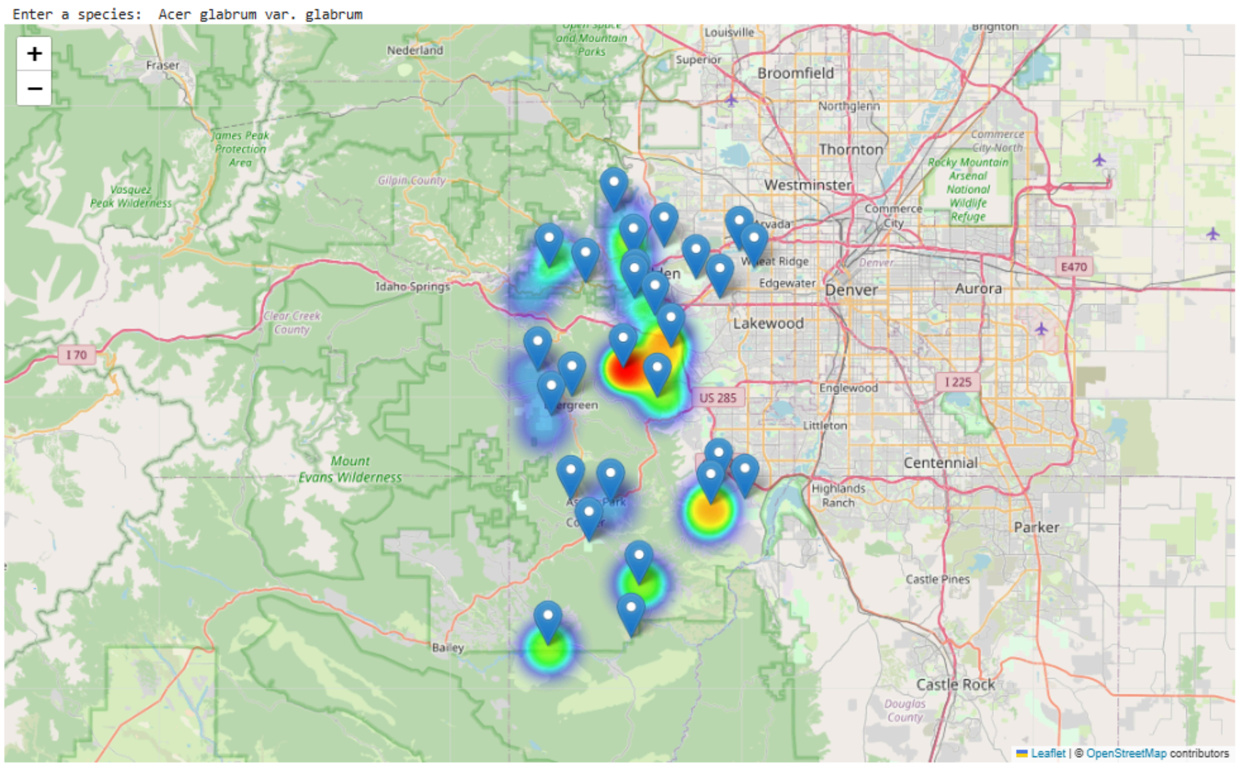
```
SELECT
p.park_name,
AVG(si.latitude),
AVG(si.longitude)
FROM sightings si
JOIN trails t
ON si.trail_id = t.id
JOIN parks p
ON t.park_id = p.id
JOIN taxa ta
ON ta.species_key = si.species_key
GROUP BY p.park_name
```

This query gave us exactly what we needed, returning the park name and a latitude and longitude coordinate pair that plotted almost perfectly onto the center of each park. We then took this query and converted it into a multi dimensional array that we looped through, plotting a marker with the name as the first element of the inner array being the name we would use, and the longitude and latitude coordinates being the second and third elements of the inner array, which accomplished with the following code.

```
for i in range(len(park_cords)):
   folium.Marker([park_cords[i][1], park_cords[i][2]], popup=park_cords[i][0]).add_to(m)
```

After running the full code and getting the proper input from the user, we are able to project a full interactive heatmap of the selected species onto a map of the Jefferson

County Area, with markers of all of the parks inside of Jefferson County properly labeled and placed.



```
Enter a species:  Acer glabrum var. glabrum
```

Figure 4: Interactive Heatmap of Sightings of a Selected Species

The visualization now allows users to see where a specific species is located, with the spots that contain more red having the most amount of sightings, the spots with blue containing the least amount of sightings, and the yellow spots being in the middle. The user can now explore the map and locate a park that contains a sighting of the species, with interactive markers telling the user which park is at which marker.

Enter a species: Acer glabrum var. glabrum

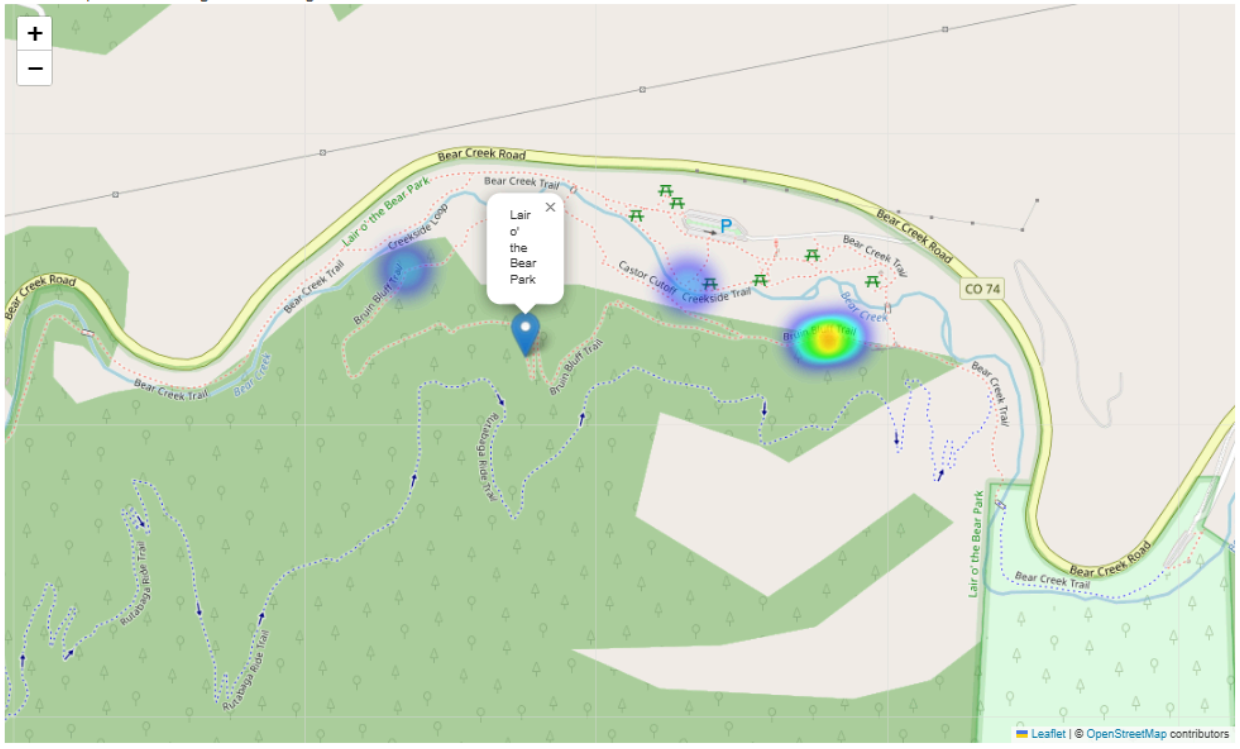


Figure 5: Demonstration of the Interactable Heatmap Zoom and Marker Functions

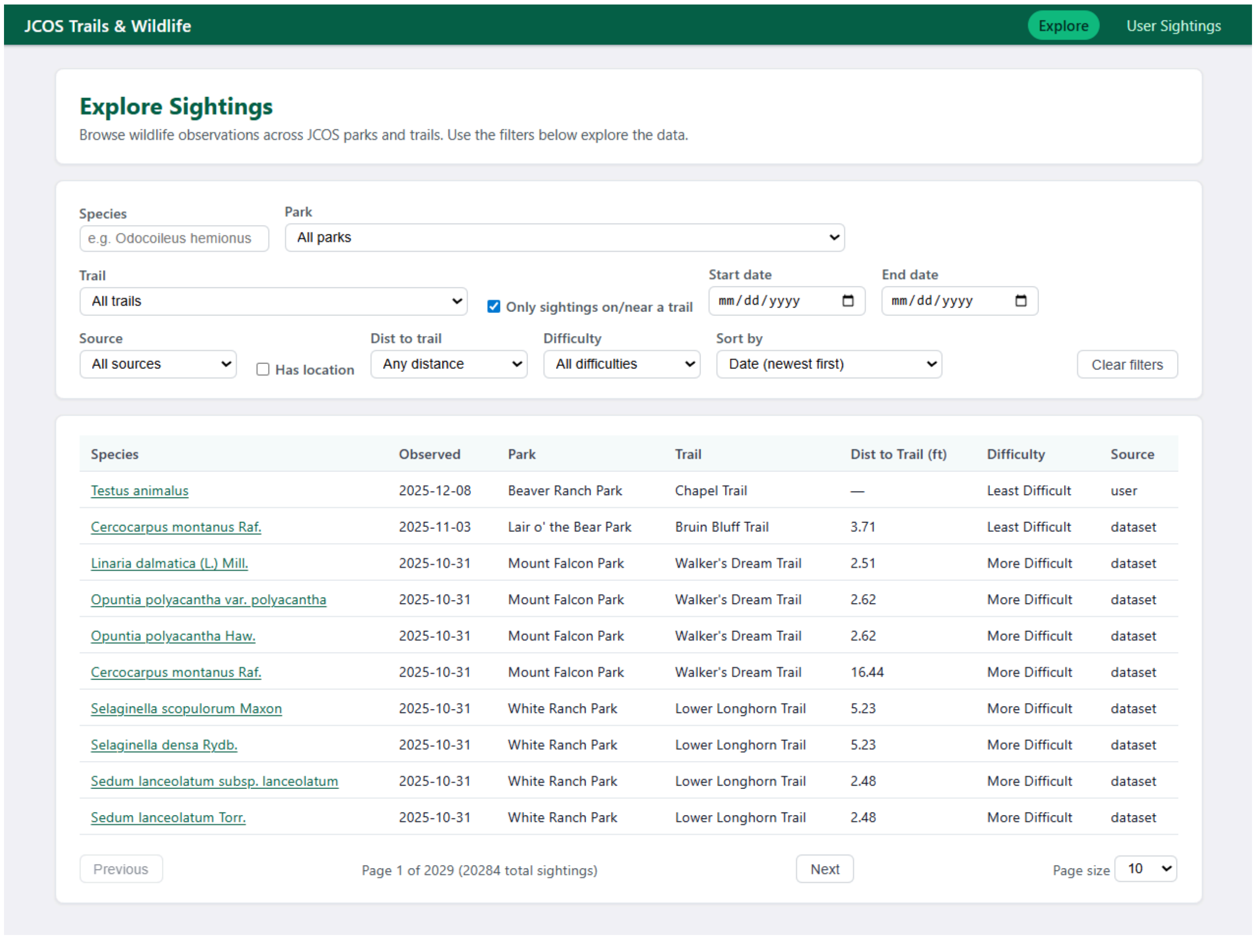
The visualization worked out amazingly and allowed not only supported user interaction and exploration of the different sightings of species across the Jefferson County Area, but also allowed for a more meaningful way to see the distribution of the ecosystems across the entire county, putting the actual sightings into something that you can see and experience, rather than just attempting to interpret a bunch of numbers.

Overall, both of these visualizations met the requirements as they were both creative ways of visualizing the data that we got from the database. In the case of the dendrogram, it allowed us to see which parks were the most similar in terms of their sightings, causing us to question why some parks with further distances were so similar, yet parks that were right next to each other were so different, challenging the idea that distance alone was the main factor in shaping the ecosystems of these parks. For the heatmap projection, it allowed us to have a much more physical interpretation of the data we got, allowing us to see where a specific species was located at on a map, rather than just looking at the park or trail it was found at, not understanding where in the Jefferson County Area that sighting actually was. These visualizations were also both used effectively and correctly, as there was no use of data conforming in order to reach the conclusions that we did from the charts. This is especially apparent in the dendrogram, as a conforming the data would not have allowed us to see that parks with similar distances had pretty different sightings, thus not prompting us to, once again, challenge the idea that distance was the only factor that influenced the sightings, allowing us to come to the conclusion that something bigger was at play.

Software

For the Software Development aspect of our outputs, we chose to make a frontend UI that enabled users to explore this data in a friendly way. We built a full stack web application that allows users to explore wildlife sightings in the Jefferson County area, with extensive filtering and sorting options. Users can also record their own sightings, backed by our normalized PostgreSQL database.

One of the main pages for the frontend is the explore page, shown below



Explore Sightings

Browse wildlife observations across JCOS parks and trails. Use the filters below explore the data.

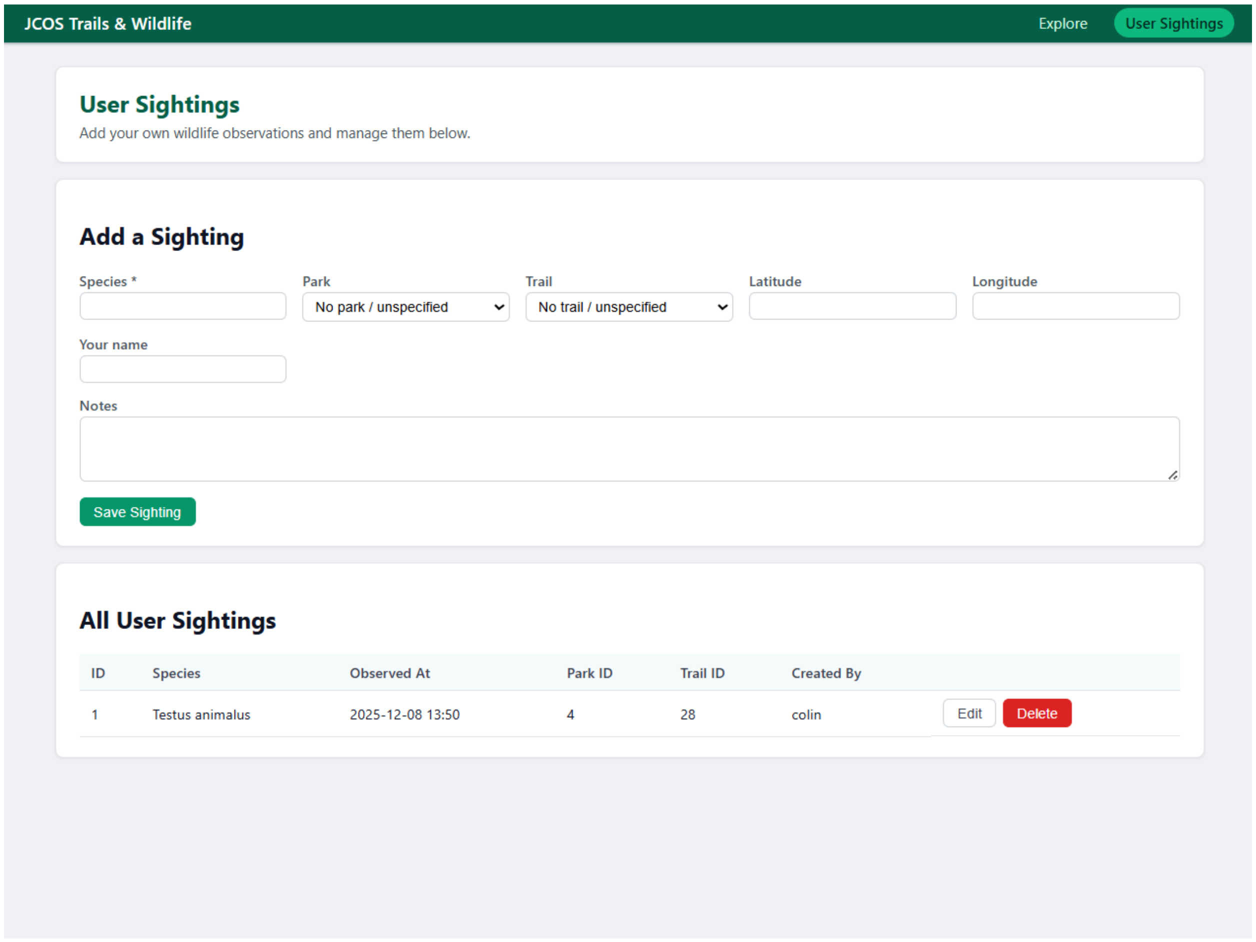
Species: e.g. Odocoileus hemionus
Park: All parks
Trail: All trails
[✓] Only sightings on/near a trail
Start date: mm/dd/yyyy
End date: mm/dd/yyyy
Source: All sources
[ ] Has location
Dist to trail: Any distance
Difficulty: All difficulties
Sort by: Date (newest first)
Clear filters

Species	Observed	Park	Trail	Dist to Trail (ft)	Difficulty	Source
Testus animalus	2025-12-08	Beaver Ranch Park	Chapel Trail	—	Least Difficult	user
Cercocarpus montanus Raf.	2025-11-03	Lair o' the Bear Park	Bruin Bluff Trail	3.71	Least Difficult	dataset
Linaria dalmatica (L.) Mill.	2025-10-31	Mount Falcon Park	Walker's Dream Trail	2.51	More Difficult	dataset
Opuntia polyacantha var. polyacantha	2025-10-31	Mount Falcon Park	Walker's Dream Trail	2.62	More Difficult	dataset
Opuntia polyacantha Haw.	2025-10-31	Mount Falcon Park	Walker's Dream Trail	2.62	More Difficult	dataset
Cercocarpus montanus Raf.	2025-10-31	Mount Falcon Park	Walker's Dream Trail	16.44	More Difficult	dataset
Selaginella scopulorum Maxon	2025-10-31	White Ranch Park	Lower Longhorn Trail	5.23	More Difficult	dataset
Selaginella densa Rydb.	2025-10-31	White Ranch Park	Lower Longhorn Trail	5.23	More Difficult	dataset
Sedum lanceolatum subsp. lanceolatum	2025-10-31	White Ranch Park	Lower Longhorn Trail	2.48	More Difficult	dataset
Sedum lanceolatum Torr.	2025-10-31	White Ranch Park	Lower Longhorn Trail	2.48	More Difficult	dataset

Previous Page 1 of 2029 (20284 total sightings) Next Page size 10

Figure 6: The explore page users are greeted with upon loading the site

The explore page’s primary purpose is to give the user a way of exploring the data in a friendly and easy to use manner. This is accomplished by providing a variety of filtering and sorting mechanisms so the user can explore different subsets of the data as needed. Users can enter a species name, and the app will search for sightings that have scientific names that are similar to the entered search text. Users can also filter by a date range, or by excluding any sightings that do not have trail data, park data or coordinates, which is important given so much of our data is off park. Users can also filter by source type to see only their recorded sightings, or by a certain difficulty or distance from a trail. Our dataset did not have very many distinct parks or trails, roughly 30 for parks, and roughly 200 individual trails. Given this restricted set of options, we decided to use a dropdown selector to allow users to sort by trails or parks, with trails getting restricted to a selected park’s trails, so users don’t accidentally filter by a trail and park that are not connected. The observation list is paginated for clarity and performance, and users can select how many items they want on a page, from 10 to 100 items. Since the data provided only scientific names of taxa that were observed, each scientific name in the list is a clickable link to a google search, which enables users to get more information on any given sighting. The other main page of the app is the user sightings page, which is where users can manage their own recorded sightings. This page is shown below.



User Sightings

Add your own wildlife observations and manage them below.

Add a Sighting

Species *	Park	Trail	Latitude	Longitude
	No park / unspecified	No trail / unspecified		

Your name:

Notes:

All User Sightings

ID	Species	Observed At	Park ID	Trail ID	Created By	
1	Testus animalus	2025-12-08 13:50	4	28	colin	Edit Delete

Figure 7: The user sightings page, which enables full CRUD with user-created sightings

The user sightings page is where full create, read, update and delete operations can be performed by the user, but only on their sightings. Users can enter a species name, and select the park or trail their sighting occurred in, with the same correlated dropdown system as on the explore page. If a sighting was too far from any park or trail, users can also enter latitude and longitude coordinates to ensure geographic information is stored. Finally, users can enter their name and any notes they may want to keep on this record. This implements the create requirement of the application. The edit and delete buttons enable the user to have update and delete ability. Of course, read is implemented in this page with the user sightings list, in combination with the explore page. Thus, this page in combination with the explore page implements the full create, read, update and delete requirements of the application.

In addition to these two pages, the frontend also has a Navbar component, which is just the header at the top of the page, present in both main components. All of these components used SightingsService, which is how the frontend communicated with the backend. The backend was implemented in Java Spring Boot, exposing endpoints at /api/sightings and /api/user-sightings to allow the frontend to access and perform operations on the database.

The backend is a Spring Boot REST API that sits between the Angular frontend and the PostgreSQL database, enabling communication between the two. This backend uses a two data-access approach. Spring Data JPA is used for full create, read, update and delete access on the user_sightings table. Spring's JdbcTemplate was used for the read-heavy endpoints that query the sightings used by the explore page, as users are not adding, removing, or changing any data on this page. Both of these approaches rely on prepared statements or parameter binding to protect against attacks like SQL injection.

The JPA-based user sightings system relied on a UserSighting Spring Entity, which was annotated with the table name (user_sightings) and the schema. The fields in this entity correspond directly with columns in the table user_sightings. This was combined with a JPA repository, which automatically provides CRUD methods (e.g. findAll(), save(), findById(), deleteById()) that use prepared statements under the hood to protect against injections while creating SQL queries. The service layer that actually queries the database wraps the repository and works in DTO (UserSightingDto) instead of entities. These DTO (Data Transfer Object) essentially just store column data as member variables in the object. This service implements the actual CRUD methods, with the example of create shown below:

```
public UserSightingDto create(UserSightingDto dto) {
        UserSighting entity = new UserSighting();
        applyDtoToEntity(dto, entity, true);
        UserSighting saved = repo.save(entity);
        return toDto(saved);
}
```

As we can see, this service implements each CRUD operation by first converting the DTO sighting into an entity, and then passing that entity to the JPA built-in save method. The service also provides the method to convert DTO to entity, shown below:

```
private void applyDtoToEntity(UserSightingDto dto, UserSighting e, boolean
isCreate) {
        e.setSpecies(dto.species());
        if (dto.observedAt() != null) {
            e.setObservedAt(dto.observedAt());
        } else if (isCreate && e.getObservedAt() == null) {
            e.setObservedAt(LocalDateTime.now());
        }

        e.setParkId(dto.parkId());
        e.setTrailId(dto.trailId());
        e.setLatitude(dto.latitude());
        e.setLongitude(dto.longitude());
        e.setNotes(dto.notes());
        e.setCreatedBy(dto.createdBy() != null ? dto.createdBy() :
"anonymous");
    }
```

This function is also where we enforce defaults on timestamp and created by. If an observation time isn't provided by our user, we replace this with the current time, and if an identity isn't provided, we fill with "anonymous" to avoid excessive null values.

We also wanted our users to be able to add, edit and delete their own sightings, but we did not want the user to be able to edit or delete any observations that were sourced from the GBIF dataset, as these are already research-grade observations, and allowing editing on this data could only degrade the usefulness of the application. As a result of this, we store user sightings in a separate table, called user_sightings, which users can edit, delete, update and view records at their convenience. This table is accessed by the frontend using the endpoint exposed by UserSightingController, which serves /api/user-sightings. This endpoint is a classic REST controller with the following methods available:

GET /api/user-sightings -> lists all user sightings
POST /api/user-sightings -> create a new sighting
PUT /api/user-sightings/{id} -> update a sighting
DELETE /api/user-sightings/{id} -> delete a sighting

The explore page is a controlled input environment, in that the user is not only purely viewing data, but is also tuning what they view with filter selectors and dropdowns and sorting options. In code, ExploreSightingDto and ExploreSightingPageDto represent the explore_sightings database view that joins sightings, trails, taxa and parks into a flat structure. The explore endpoint returns a paginated result. This endpoint is implemented in ExploreSightingController exposing GET at /api/sightings, which calls the explore service where pagination and filtering occur. Queries were built in the following manner:

```
StringBuilder where = new StringBuilder(" WHERE 1=1 ");
List<Object> params = new ArrayList<>();

if (speciesFilter != null && !speciesFilter.trim().isEmpty()) {
    where.append(" AND LOWER(species) LIKE ? ");
    params.add("%" + speciesFilter.trim().toLowerCase() + "%");
}
if (Boolean.TRUE.equals(onlyWithTrail)) {
    where.append(" AND trail_id IS NOT NULL ");
}
if (parkId != null) {
    where.append(" AND park_id = ? ");
    params.add(parkId);
}
if (trailId != null) {
    where.append(" AND trail_id = ? ");
    params.add(trailId);
}

// ... Continues with source, hasLocation, maxDistToTrailFt, difficulty,
dates, omitted for brevity ...

String countSql =
    "SELECT COUNT(*) FROM cwolff.explore_sightings" + where;
Long totalObj =
    jdbc.queryForObject(countSql, params.toArray(), Long.class);
```

Once this query was built, it gets passed off to JDBC query as follows:

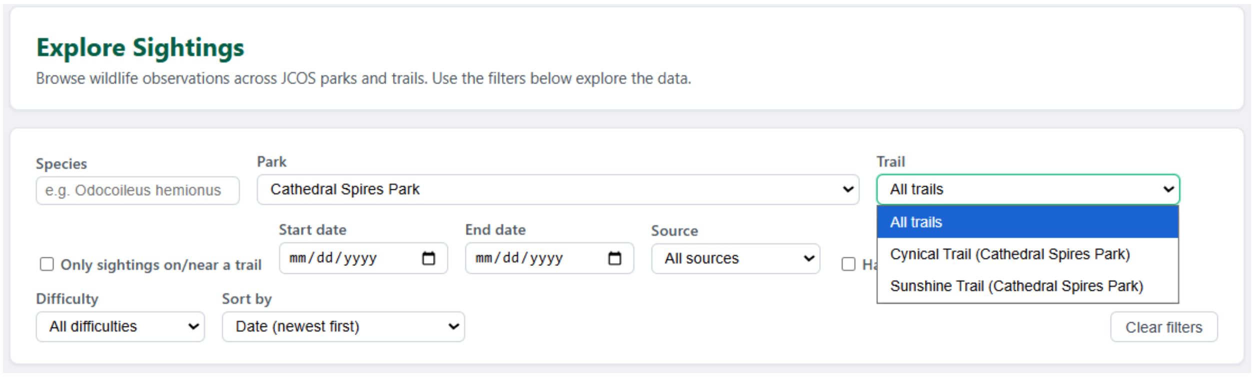
```
String orderBy = buildOrderBy(sort);
String dataSql = """
    SELECT id, source, species, observed_at, latitude, longitude,
basis_of_record, dist_to_trail_ft,
    park_id, park_name, trail_id, trail_name, difficulty, user_type,
notes, created_by
```

```
    FROM cwolff.explore_sightings
    """ + where + " " + orderBy + " LIMIT ? OFFSET ?";
List<Object> dataParams = new ArrayList<>(params);
dataParams.add(size);
dataParams.add(offset);

List<ExploreSightingDto> items =
    jdbc.query(dataSql, this::mapRow, dataParams.toArray());
```

All user inputs go into the params list, and are bound as ? parameters by JdbcTemplate. buildOrderBy(sort) returns one of a fixed set of order by strings.

In order to avoid confusion between users and the actual list of parks and trails present in the database, we used dropdown selectors to allow users to pick a park or trail on the explore page to filter by. These dropdowns contain every valid park and trail, which is possible because the number of distinct parks is <50 and the number of distinct trails is <300. In order to avoid a messy, huge dropdown when possible, when a user selects a park, the trails dropdown only shows trails within that park, this is shown in the image below.


Explore Sightings

Browse wildlife observations across JCOS parks and trails. Use the filters below explore the data.

Species: e.g. Odocoileus hemionus
Park: Cathedral Spires Park
Trail: All trails
- All trails
- Cynical Trail (Cathedral Spires Park)
- Sunshine Trail (Cathedral Spires Park)

Only sightings on/near a trail
Start date: mm/dd/yyyy
End date: mm/dd/yyyy
Source: All sources
Ha
Difficulty: All difficulties
Sort by: Date (newest first)
Clear filters

Figure 9: Filter page, showing trail options dynamically updates with park selection

Performance Tuning: Loading Filtered Sightings List

When looking at the huge dataset, it is almost instinct to optimize it in some sort of way! With that we wanted to improve the performance of loading a filtered and paginated list of animal sightings for the trail and date range. It's a key user action in the web application, where we want users to quickly be able to see the recent sightings on certain trails. The dataset contains approximately 200,000 rows with the sightings table including the columns trail_id, observed_at, timestamp, and other keys that link to the trail and park details.

Initially, query performance when we were filtering sightings by trail and date was very slow and is a great example of when users browse the long date ranges for the most popular trails. This of course leads to longer load times which we all know can be annoying in any situation as it degrades the user experience.We followed the canonical query pattern in this case!

```
SELECT
  s.gbif_id,
  s.observed_at,
  s.url,
  tx.taxa_species,
  t.trail_name,
  p.park_name
FROM sightings s
JOIN taxa tx ON s.taxon_id = tx.taxon_id
JOIN trails t ON s.trail_id = t.id
JOIN parks p ON s.park_id = p.id
WHERE
  s.trail_id = trailId
  AND s.observed_at BETWEEN startDate AND endDate
  AND s.is_deleted = false
ORDER BY s.observed_at DESC
LIMIT : pageSize
OFFSET  pageNumber * pageSize;
```

We can see that above we fetch the sightings that are filtered by a specific trail ID and a user-selected date range, followed by it being sorted from the most recent observation timestamp. In order for us to improve the querying efficiency, we made a composite index on the sightings table!

```
CREATE INDEX idx_sightings_trail_date
```

```
ON sightings (trail_id, observed_at DESC);
```

The index supports the filtering of both trail_id and observed_at range and what helps is the ORDER BY observed_at DESC clause as it enables the quick retrieval of the rows users would want in that moment.

With the new implications that had been made, we had a range of results! Before making the index we saw that the query made a sequential scan across the sightings table. This led to the execution times of around 800 milliseconds for the average queries that cover several months of data on a moderate trail. After we applied the index, we saw that the execution times dropped all the way down to about 90 milliseconds, which shows a 88% reduction in the amount of time it needs to run. We also implemented pagination to limit the result set to 50 rows per page which further improved response times by reducing the data transfer and rendering to the user. By creating a targeted composite index that is on the same page as core query filters and sorting, as well as applying pagination, the time it took to complete a request significantly improved as the responsiveness of filtered sightings loads quicker! This performance tuning helps contribute to a better user experience and scalable querying of large datasets.

Target	Dataset Amount	Before Index	After	Improvement by percentage	Performance Tuning Application
Load sightings for Trail #5 (popular trail), Jan 1 2023 - Dec 31 2024, page 1 / first 50 rows	~15,000 matching rows	820 ms	92 ms	89% faster	Composite index (trail_id, observed_at DESC) + pagination
Load sightings for Trail #12 (less popular trail), Summer 2024, page 1 / first 50 rows	~2,100 matching rows	340 ms	45 ms	87% faster	Composite index (trail_id, observed_at DESC) + pagination

Load sightings for Trail #5, page 2 / first 50 rows	~15,000 matching rows	850 ms	88 ms	90% faster	Index + OFFSET pagination efficiency

Another performance tuning idea we implemented was one to get Species-Filtered Sightings per Trail! Other than just loading all sightings for a trail and date range, another possibility users can do is filtering the dataset to a specific species like a wolf or elk on a certain trail on a date or a date range. The query gets the users answers, to questions like "Show only the elk sightings on this trail last winter" or another example is "Show sightings of a pigeon (particular bird species, I don't know why a user would pick a pigeon, but roll with it) this spring." The query below adds another filter on s.taxon_id from the query shown before and it still filters and sorts primarily by the trail_id and observed_at.

```
SELECT
  s.gbif_id,
  s.observed_at,
  s.url,
  tx.taxa_species,
  t.trail_name,
  p.park_name
FROM sightings s
JOIN taxa tx ON s.taxon_id = tx.taxon_id
JOIN trails t ON s.trail_id = t.id
JOIN parks p ON s.park_id = p.id
WHERE
  s.trail_id = trailId
  AND s.taxon_id = taxonId
  AND s.observed_at BETWEEN startDate AND endDate
  AND s.is_deleted = false
ORDER BY s.observed_at DESC
LIMIT pageSize
OFFSET pageNumber * pageSize;
```

We also are using the same composite index made previously as it still provides most of the benefit for the species-filtered query. The database first uses trail_id of the index to quickly get the only rows for the trail that we want, instead of scanning all 200,000 or so sightings. In the subset, it uses observed_at ordering from the index to find the rows in the date range wanted that's already sorted by newest first. The taxon_id becomes a filter used for the much smaller set of rows that is returned by the index. This is significantly better than applying it over the whole table. The index lowers the amount of rows that has to be scanned and checked with the taxon_id. Even though taxon_id isn't part of the index, the composite index still reduces the work for the query. In the first query we have trail + date range using idx_sightings_trail_date and in the second query we have trail + taxon + date range. This is also sped up by idx_sightings_trail_date. This means we have 2 performance tuning queries that are both tied to the same indexing strategy and both benefiting from reduced scanned rows in order to have better user experience!

Conclusion

Appendix

A: Data loading script

```
CREATE TABLE IF NOT EXISTS parks (
    id integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    park_name text,
    park_owner text,
    manager text,
    land_preservation_type text,
    website_url text,
    website_photo_url text,
    acres double precision,
    acquisition_date text,
    geom_wkt text
);

CREATE TABLE IF NOT EXISTS trails (
    id integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    trail_name text,
    park_name text,
    total_mileage double precision,
```

```sql
    difficulty text,
    user_type text,
    total_length double precision,
    geom_wkt text,
    park_id integer
);

CREATE TABLE IF NOT EXISTS taxa (
    taxon_id bigint PRIMARY KEY,
    taxon_class text,
    taxon_order text,
    taxon_genus text,
    taxon_species text,
    scientific_name text,
    species_key bigint
);

CREATE TABLE IF NOT EXISTS sightings (
    gbif_id bigint PRIMARY KEY,
    taxon_id bigint,
    trail_id integer,
    park_id integer,
    latitude double precision,
    longitude double precision,
    basis_of_record text,
    observed_at timestamp without time zone,
    species_key bigint,
    dist_to_trail_ft double precision
);

CREATE TABLE IF NOT EXISTS user_sightings (
    id integer GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    species text,
    observed_at timestamp without time zone,
    park_id integer,
    trail_id integer,
    latitude double precision,
    longitude double precision,
    notes text,
    created_by text
```

```
);

ALTER TABLE trails
    ADD CONSTRAINT trails_park_id_fkey
    FOREIGN KEY (park_id)
    REFERENCES parks(id)
    ON UPDATE CASCADE
    ON DELETE SET NULL;

ALTER TABLE sightings
    ADD CONSTRAINT sightings_park_id_fkey
    FOREIGN KEY (park_id)
    REFERENCES parks(id)
    ON UPDATE CASCADE
    ON DELETE SET NULL;

ALTER TABLE sightings
    ADD CONSTRAINT sightings_trail_id_fkey
    FOREIGN KEY (trail_id)
    REFERENCES trails(id)
    ON UPDATE CASCADE
    ON DELETE SET NULL;

ALTER TABLE sightings
    ADD CONSTRAINT sightings_taxon_id_fkey
    FOREIGN KEY (taxon_id)
    REFERENCES taxa(taxon_id)
    ON UPDATE CASCADE
    ON DELETE SET NULL;

ALTER TABLE user_sightings
    ADD CONSTRAINT user_sightings_park_id_fkey
    FOREIGN KEY (park_id)
    REFERENCES parks(id)
    ON UPDATE CASCADE
    ON DELETE SET NULL;

ALTER TABLE user_sightings
    ADD CONSTRAINT user_sightings_trail_id_fkey
    FOREIGN KEY (trail_id)
```

```sql
    REFERENCES trails(id)
    ON UPDATE CASCADE
    ON DELETE SET NULL;

INSERT INTO parks (
    park_name,
    park_owner,
    manager,
    land_preservation_type,
    website_url,
    website_photo_url,
    acres,
    acquisition_date,
    geom_wkt
)
SELECT
    ps.park_name,
    ps.park_owner,
    ps.manager,
    ps.land_preservation_type,
    ps.website_url,
    ps.website_photo_url,
    ps.acres,
    ps.acquisition_date,
    ps.geom_wkt
FROM parks_staging ps;

INSERT INTO trails (
    trail_name,
    park_name,
    total_mileage,
    difficulty,
    user_type,
    total_length,
    geom_wkt,
    park_id
)
SELECT
    ts.trail_name,
    ts.park_name,
    ts.total_mileage,
    ts.difficulty,
    ts.user_type,
    ts.total_length,
    ts.geom_wkt,
    p.id AS park_id
FROM trails_staging ts
LEFT JOIN parks p
    ON ts.park_name = p.park_name;

INSERT INTO taxa (
    taxon_id,
    taxon_class,
    taxon_order,
    taxon_genus,
    taxon_species,
    scientific_name,
    species_key
)
SELECT DISTINCT
    NULLIF(ws.taxonkey, '')::bigint AS taxon_id,
    ws.sighting_class AS taxon_class,
    ws.sighting_order AS taxon_order,
    ws.genus AS taxon_genus,
    ws.species AS taxon_species,
    ws.scientificname AS scientific_name,
    NULLIF(ws.specieskey, '')::bigint AS species_key
FROM wildlife_staging ws
WHERE ws.taxonkey IS NOT NULL
  AND ws.taxonkey <> '';

INSERT INTO sightings (
    gbif_id,
    taxon_id,
    trail_id,
    park_id,
    latitude,
    longitude,
    basis_of_record,
    observed_at,
    species_key,
    dist_to_trail_ft
)
SELECT
    wt.gbifid::bigint AS gbif_id,
    tx.taxon_id AS taxon_id,
    tr.id AS trail_id,
    pk.id AS park_id,
    wt.decimallatitude AS latitude,
    wt.decimallongitude AS longitude,
    wt.basisofrecord AS basis_of_record,
    NULLIF(wt.eventdate, '')::timestamp AS observed_at,
    wt.specieskey::bigint AS species_key,
    wt.dist_to_trail_ft
FROM wt_staging wt
LEFT JOIN parks pk ON wt.park_name = pk.park_name
LEFT JOIN trails tr ON wt.trail_name = tr.trail_name AND tr.park_id =
pk.id
LEFT JOIN taxa tx ON wt.specieskey::bigint = tx.species_key;
```

References

[1]“iNaturalist Research-grade Observations,” Gbif.org, Feb. 2012, doi: https://doi.org/10.15468/ab3s5x.

[2] P. Mooney, “Jefferson County Colorado Open Space and Trails,” Kaggle.com, 2021. https://www.kaggle.com/datasets/paultimothymooney/jefferson-county-colorado-open-space-and-trails/data?select=open_space_trails.csv (accessed Dec. 09, 2025).