Wing Watch Final Design Document

Elevator Pitch

Have you ever been to Gravelly Point in the afternoon on a weekend? If you have then you'll remember seeing people talking, enjoying a meal, or just biking through. While these technical wonders of aviation have become more commonplace and less novel than they were nearer to the birth of the industry, many still flock to locations like these to marvel at the spectacle that is man-made flight. These enthusiasts are the perfect customer for my product Wing Watch.

While the idea behind tracking planes and creating interfaces to tell which are nearby at the exact moment has been innovated many times. There has been less thought about finding specific spaces in which you can view this air traffic, and that is where my app innovates. Using air traffic data to map flightpaths of departing and arriving flights, we can locate certain spots that have both high air traffic, good visibility, and general nice ambiance.

Wing Watch will be a web app that will use an algorithm find potential plane spotting spots so that users might be able to enjoy an afternoon of plane spotting from a more remote location. Using my app, users will be able to locate potential spots near them, visit them, and then rate them on qualities such as average altitude, distance from flightpath, and more. Wing Watch will result in the discovery of many new plane spotting spots, some of which will even more accessible to those living in locations further away from an airport.

While some might argue that disclosing the locations of these places might ruin their serenity, we believe that showing these places off will create a greater connection between the plane spotting community and nature.

Wing Watch isn't just about finding the best spots; it's also about empowering the community to contribute and collaborate. Users will cultivate a community that loves plane spotting. They will engage with other plane spotters and share their knowledge by leaving comments, ratings, reviews, and tips on specific spots, helping others plan their outings more effectively. This interactive element transforms Wing Watch from a mere utility to a dynamic platform that celebrates the beauty of aviation and encourages meaningful connections among people who share this passion.

Technical Summary

Flight paths are not the most difficult thing to track. The final approach of an aircraft is the last leg of any given journey. In a normal airport under good visual conditions, the final approach will start anywhere from a half mile to two miles before the runway when the plane gets lined up. Depending on wind conditions or other external factors, sometimes the designated landing path will switch to a different approach. But even then, the plane can still be observed. It would be quite easy to draw 2 lines stretching of each runway for approximately 2 miles in each direction. But this approach clearly leaves something to be desired.

WingWatch is not an app for finding landing and takeoff paths, it is an app for finding planes. For finding them at their lowest altitude, or their most frequent landing times, WingWatch will be able to find spots that do not just coincide with flight paths, but optimize the plane spotting experience based on a multitude of variables. WingWatch uses real live flight data pulled from a network of ADS-B receivers to track data and filter it out based on plane spotting preferences. The data is tracked in a 30 $mile^2$ area around the selected airport. All flights that are departing and arriving from the airport will have their locations, timestamps, altitudes, and other variables saved into a csv file. This will then be filtered out further by restricting a height limit of between 4921 feet and 10 feet in altitude so that we can isolate planes that are flying low but have yet to land. The tracked data is then put through a cluster algorithm called DBSCAN (Density-Based Spatial Clustering of Applications). This algorithm groups data points based on their density and identified clusters in high density regions. We can adjust the variables to change the minimum size of a cluster and the value of epsilon which represents the distance threshold between two datapoints to group them in the same cluster. Given our large sample size and input parameters, we might expect anywhere from fifty to 100 clusters. This is considerably larger than the number of descent paths, but luckily because DBSCAN makes no attempt to limit clusters by a maximum value of points, most of our clusters will contain the minimum number of points and a select few clusters will contain upwards of 1000 points. When we further filter our data to only include the top few clusters, we will notice patterns. Our clusters will be long straight lines of points ranging anywhere from less than a mile to around 2 miles long. They won't be perfect of course; these lines may branch out in sections. Just because we can reliably predict that the densest clusters will correspond to the most common flight paths, does not mean we can ignore outlier data. We will need to filter our data even further. The points that adhere to our predicted flight paths will be the points in the densest part of the cluster, so to ensure we are only using the most accurate data we can filter out the 10% least dense points leaving only the strongest most relevant datapoints. Next, our flight paths will need to be plotted into a formula using polynomial regression to create a flightpath. Along this path we will find points along this line spaced evenly 0.3 miles apart, from each point on the line we search for nearby parks in a 0.3 radius using the google maps API. Once we have a list of parks we will filter them further, we use a minimize function on each park to find the closest distance between it and the nonlinear flightpath, then we filter our all parks greater than 0.3 miles away. Lastly, we will need to create custom attributes for our data, this includes the distance from flightpath we calculated to filter our data, and faverage altitude, which takes the average altitude of the flight data collected in a 0.3 radius around the spot.

Once our data collection is complete, we will begin work on an app so that users can view and interact with our collected data.

The limit of this project is not defined by us, plane watching has been around for over a century and to say WingWatch has the final word on plane spotting would not be right. WingWatch is a tool for the community to use as they wish, which is why I would like to allow as much freedom to users of the app as I can. Users will be able to search for locations, favorite locations, leave ratings and reviews, and even add their own locations that they feel the app has overlooked. Users may also even be able to connect with other plane spotters, view reviews from specific accounts, and maybe even connect with them by direct message. This is still quite a distant milestone, and we face many challenges ahead. App development will take some time before we can release a minimum viable product and even more time before we can say we are finished, but we hope to have a functional product by May.

Project Summary

User Stories

As a user, I would like to be able to provide my location, so I can find plane spotting places in my area.

As a user, I would like to be able to filter places by attributes such as average altitude or distance from flight path, so I can find spots that cater to my specifications.

As a user, I would like to be able to rate, review, and favorite places, so that I can share my own experience.

As a user, I would like to see other people's ratings and reviews, so that I can use external information to choose where I want to go.

As a user, I would like to be able to see the spots represented as a map along with my location, so I can visualize the locations of these places.

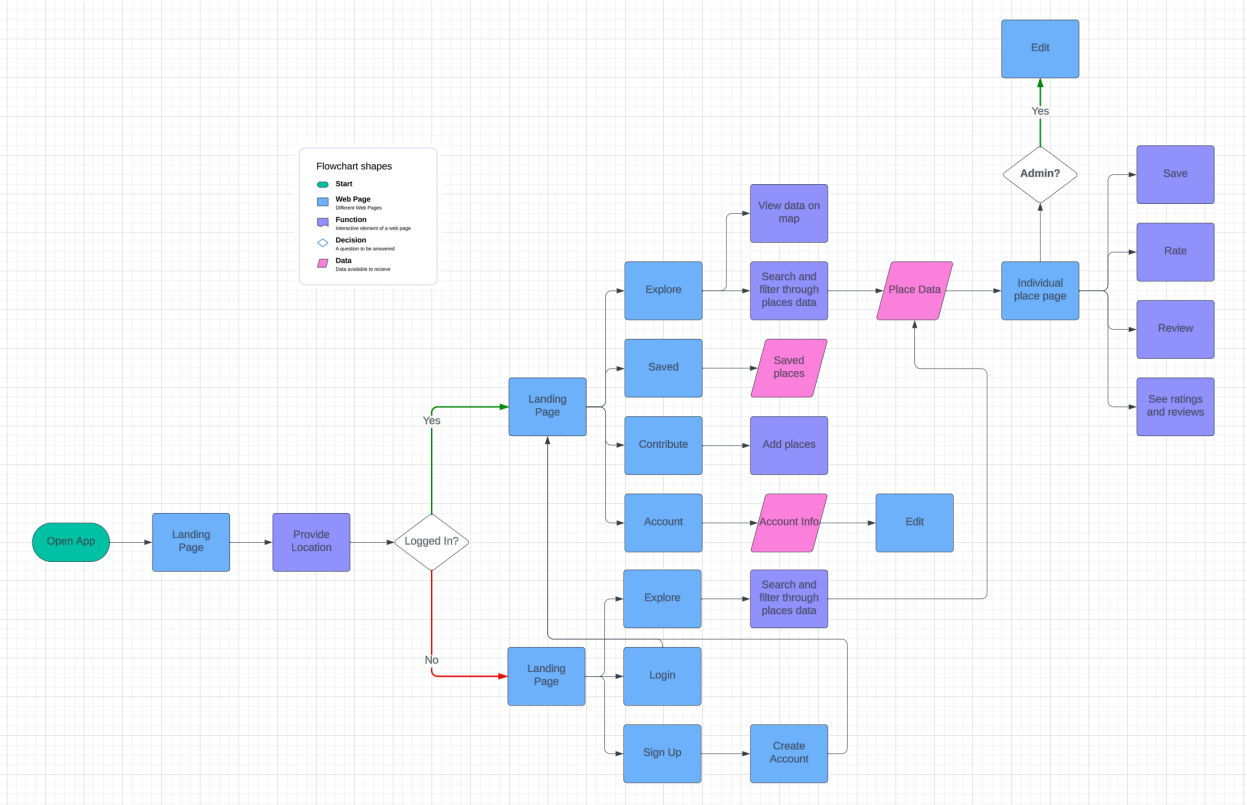
As a user, I would like to be able to externally share places, so that I can engage my friends with plane spotting.

As an admin, I would like to be able to remove reviews if I find the content offensive, so I can cultivate a more inclusive community.

As a user, I would like to suggest the addition of places, so that users can engage with the app even further.

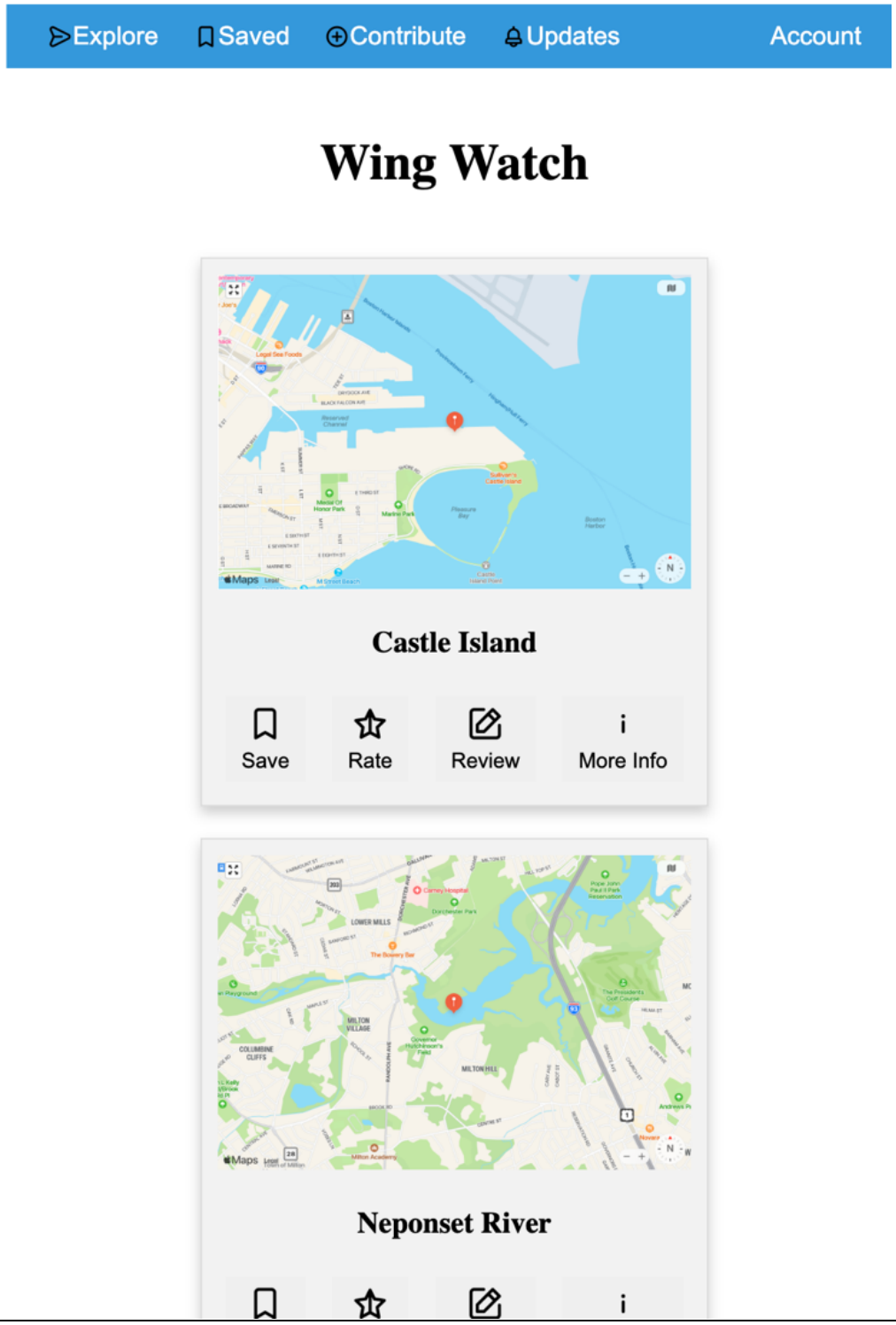
As an admin, I would like to be able to add places, and review, accept, or deny suggested places from users, so that I can help cultivate a more user driven space.

Flow Diagram

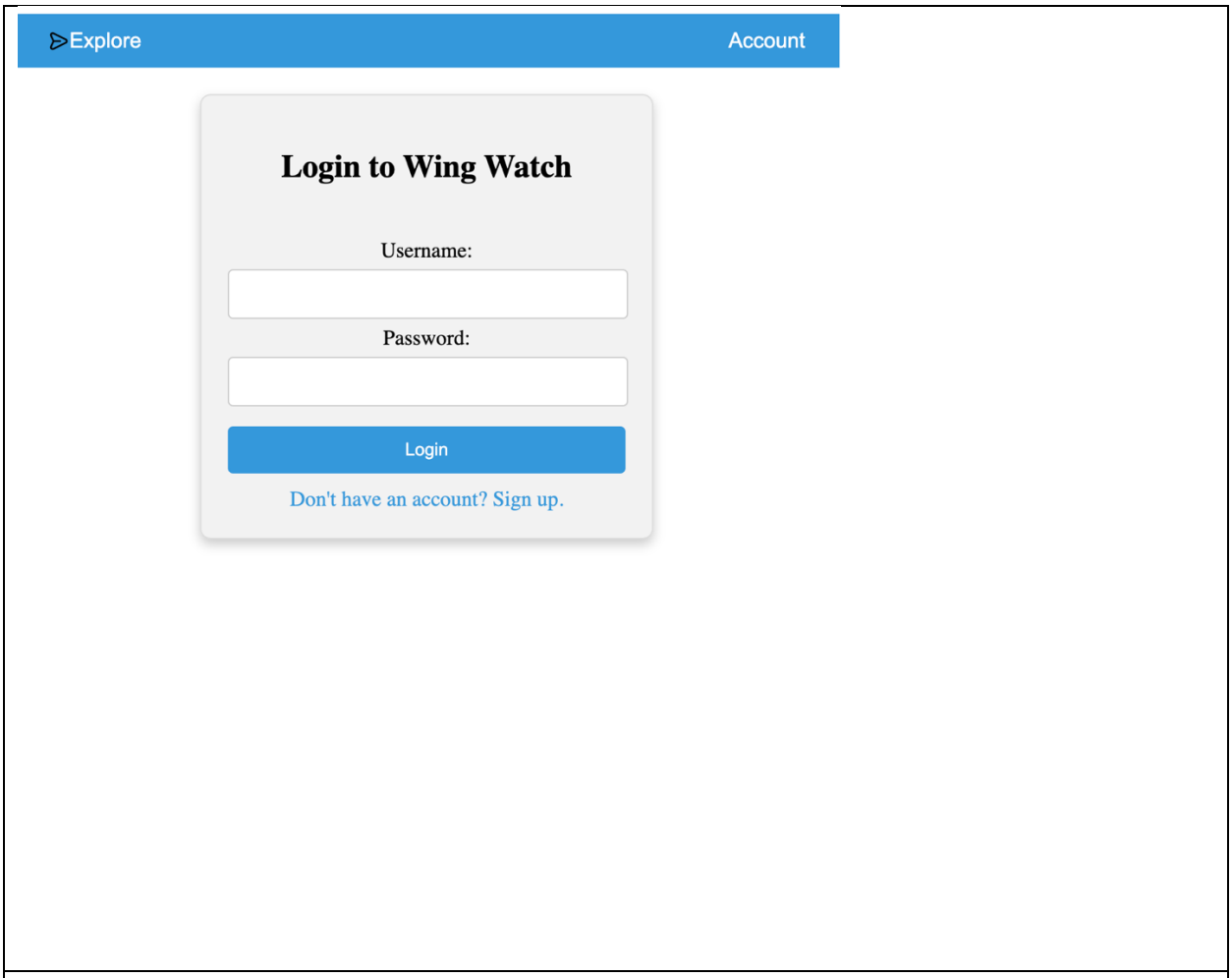


Mockups/Wireframes

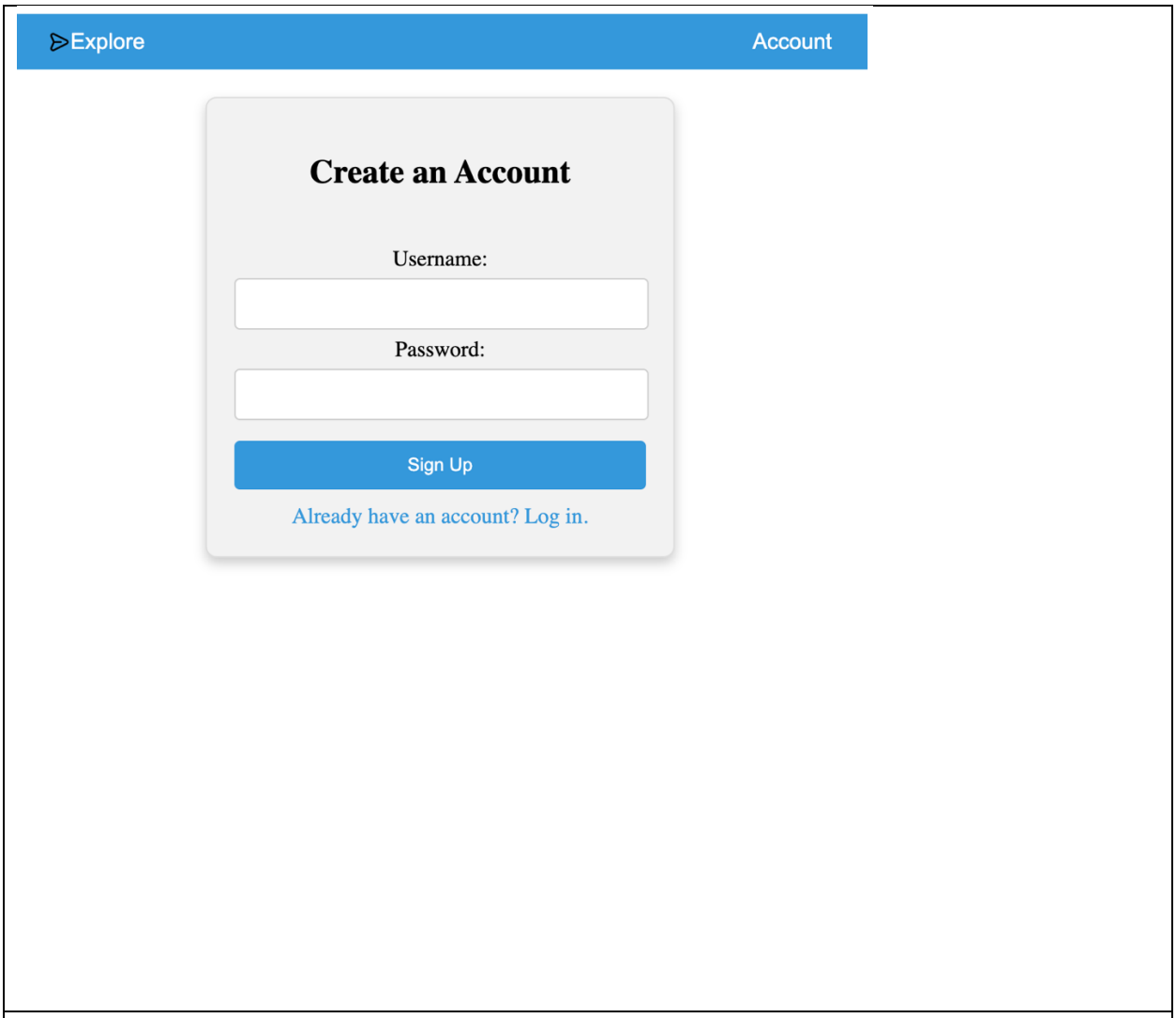
Landing Page


Explore
Saved
Contribute
Updates
Account
Wing Watch
Castle Island
Save
Rate
Review
More Info
Neponset River

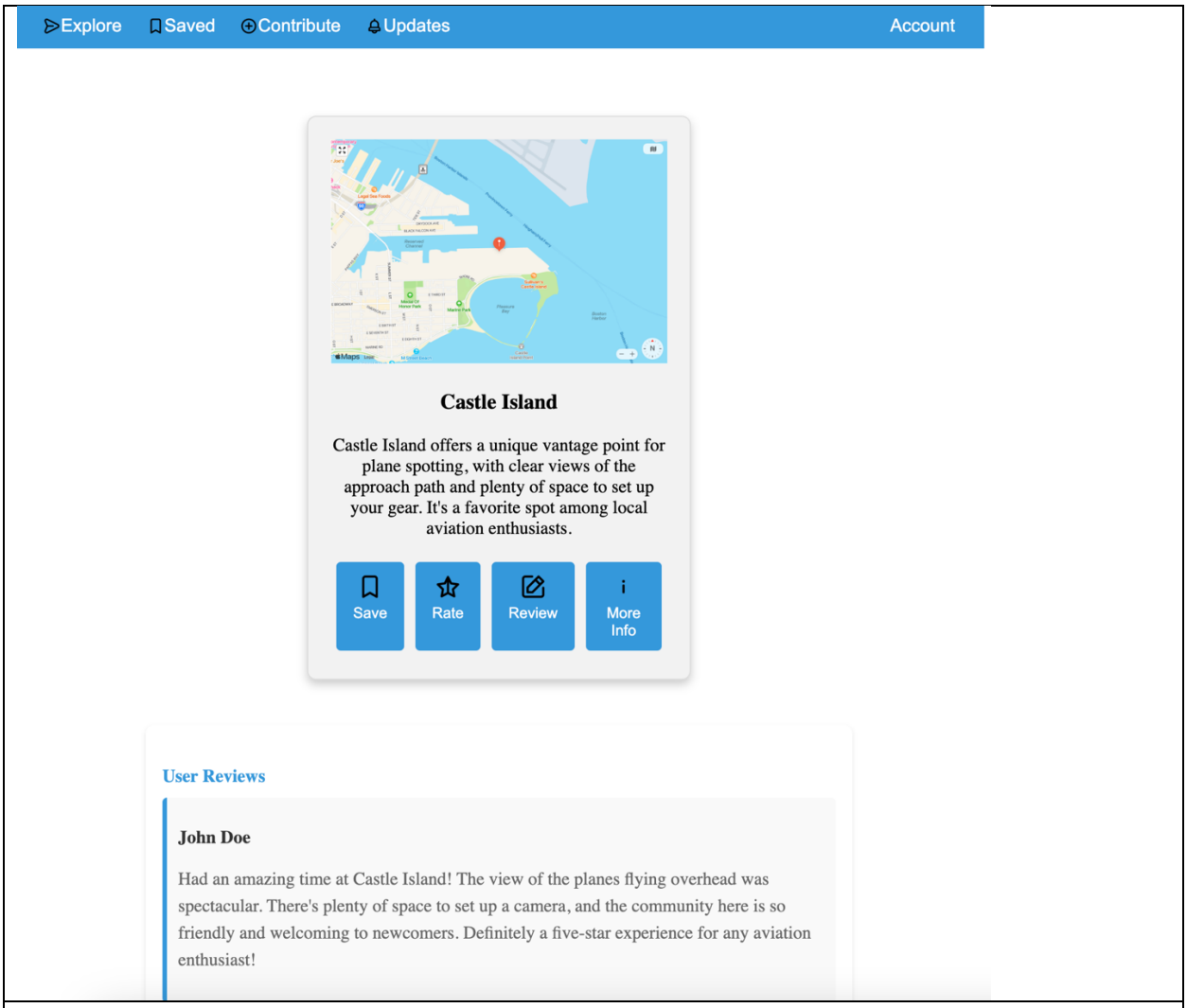
Login Page


Explore
Account
Login to Wing Watch
Username:
Password:
Login
Don't have an account? Sign up.

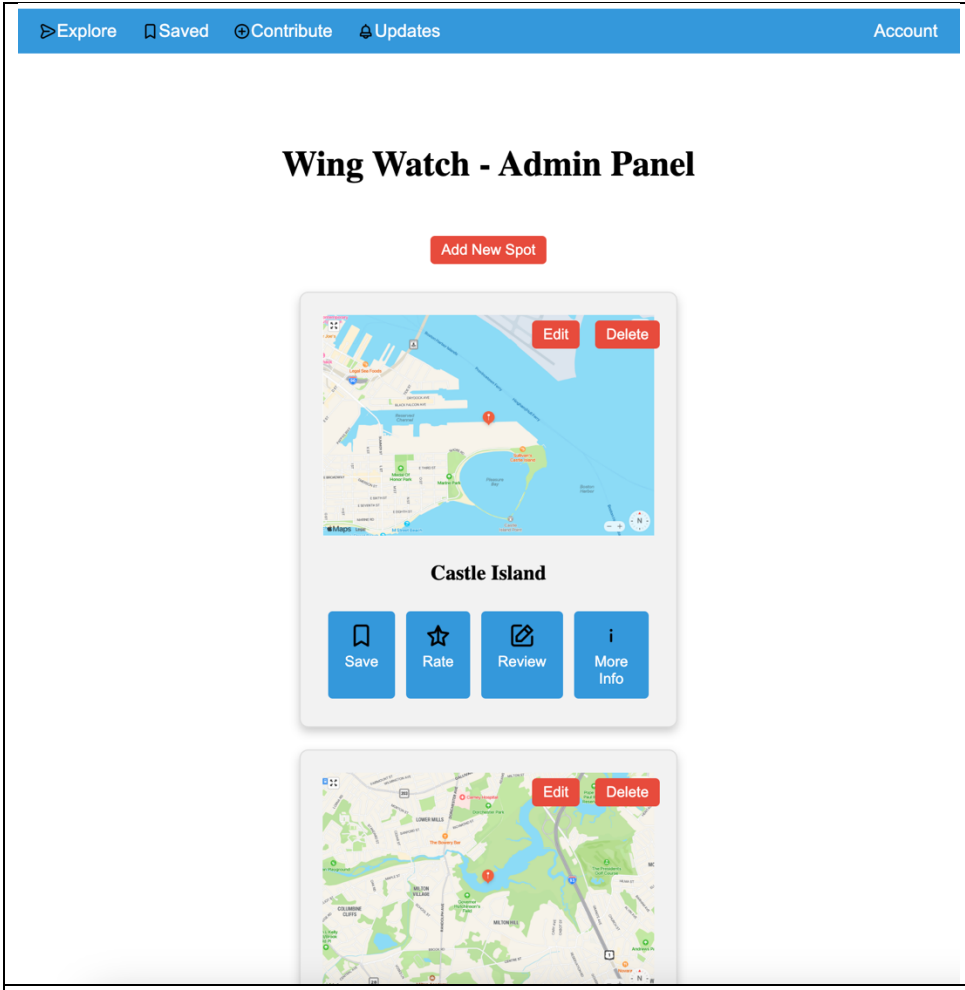
Sign Up Page



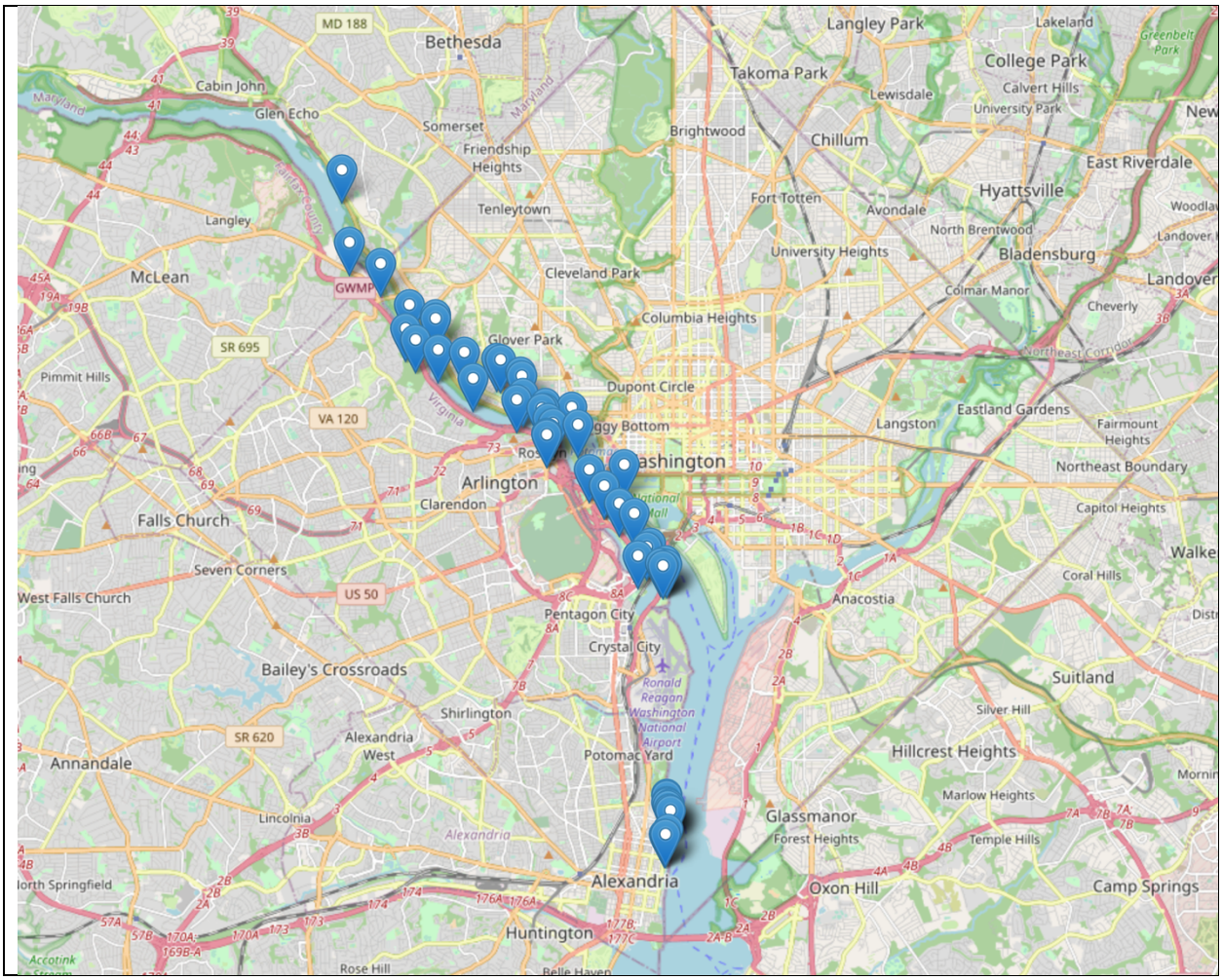
Individual Spot Page


Explore
Saved
Contribute
Updates
Account
Castle Island
Castle Island offers a unique vantage point for plane spotting, with clear views of the approach path and plenty of space to set up your gear. It's a favorite spot among local aviation enthusiasts.
Save
Rate
Review
More Info
User Reviews
John Doe
Had an amazing time at Castle Island! The view of the planes flying overhead was spectacular. There's plenty of space to set up a camera, and the community here is so friendly and welcoming to newcomers. Definitely a five-star experience for any aviation enthusiast!

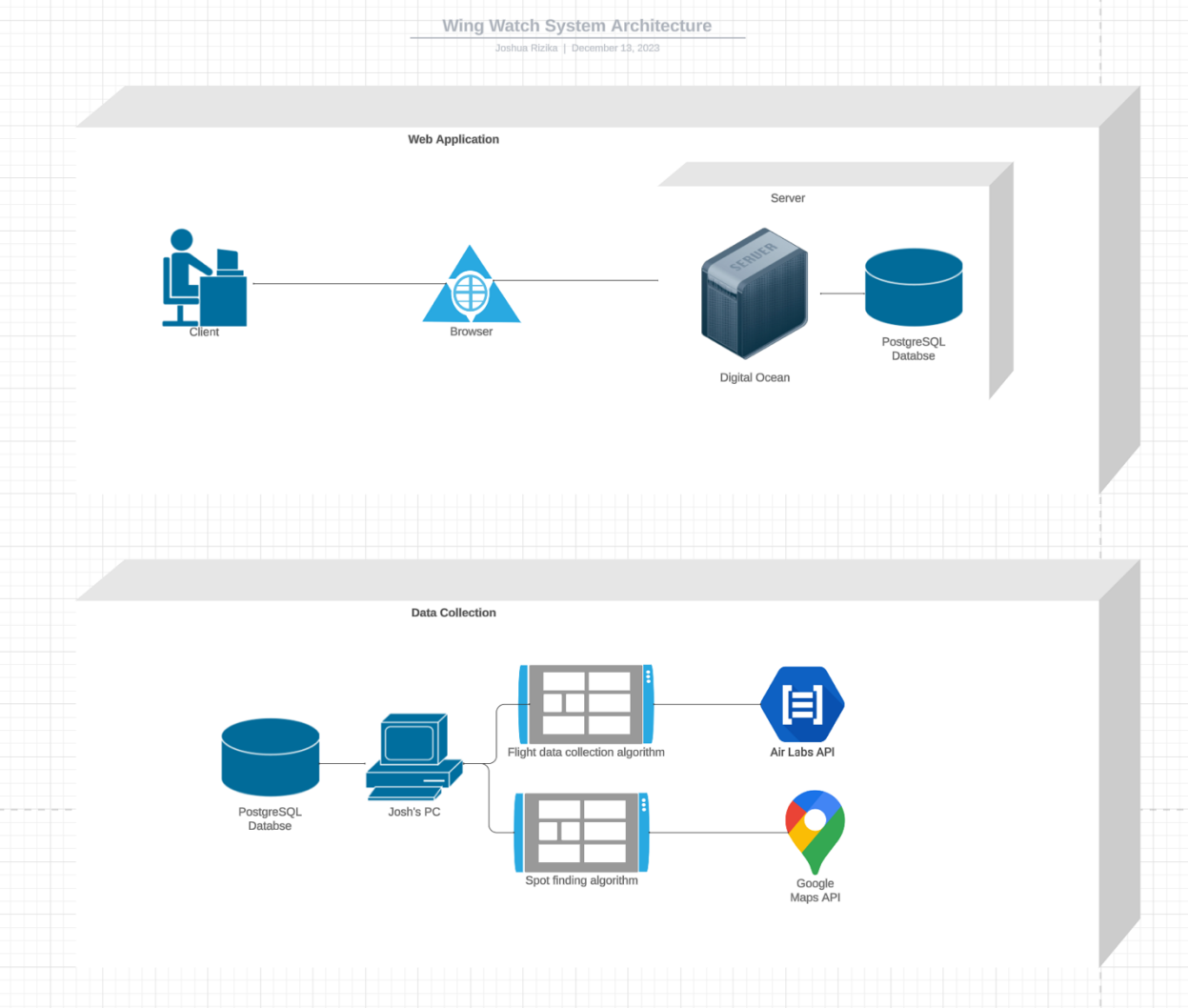
Admin Page



Map of Resulting Places



Technical Specifications



External API's and Frameworks

Air Labs API

Goal

- Get live flight data around the specified airport.

Description

This API is used in the flight_data.py file in the code. It connects to the data collection functionality and is called twice every 10 seconds. The parameter of the bounding is generated by the calculateCoordinatesWithinDimensions function, and the parameter of the arriving or departing iata code is given provided before runtime.

Endpoints Used

- GET https://airlabs.co/api/v6/flights?api_key=ed9f2aab-ab95-4805-9da8-b43eaf96a836&bbox={minLat}, {minLong}, {maxLat}, {maxLong}&dep_iata=BOS&_fields=flight_icao, airline_icao, lat, lng, alt, dep_iata, arr_iata, status
- GET https://airlabs.co/api/v6/flights?api_key=ed9f2aab-ab95-4805-9da8-b43eaf96a836&bbox={minLat}, {minLong}, {maxLat}, {maxLong}&arr_iata=BOS&_fields=flight_icao, airline_icao, lat, lng, alt, dep_iata, arr_iata, status

Google Maps API

Goal

- Get nearby locations within a radius of a coordinate pair.

Description

This API is used in the spots.py file in the code. It connects to the spot finding functionality and is called after the trendline has been plotted. The parameter includedTypes is hardcoded to “parks”, the parameter maxResultCount is hard coded to 20, the parameters latitude and longitude are taken from each equidistant point along the calculated trendline for each cluster, and the parameter radius is taken from the interval distance between points on the trendline. All parameters are passed into this API to get the result.

Endpoints Used

- POST https://places.googleapis.com/v1/places:searchNearby
  - Raw Body (JSON):

```
{
 "includedPrimaryTypes": ["park"],
 "maxResultCount": 20,
 "location_restriction": {
 "circle": {
    "center": {
     "latitude": 39.8563,
     "longitude": -104.6764},
    "radius": 30000
  }
 }
}
```

Algorithms

Flight Paths

Goal

- Collect flight data around the specified airport.

Description

- This algorithm uses threading to run on multiple different airports simultaneously. At the beginning of each thread, we access or create a .csv file. We perform two API calls in 10 second intervals. Each return all flights within a 30-mile box around the airport, but one returns all departing flights from the airport and the other returns all arriving flights. AirLabs, the company providing the API, gives elevation expressed in meters meaning we must make corrections to the values. We convert the altitude from meters to feet and then subtract the elevation of the airport to get an altitude relative to the targeted airport. Lastly, we add the results to the .csv file and then repeat until we reach 5000 rows of data.

Clusters

Goal

- Filter the datapoints into density-based clusters representing flightpaths, then map a trendline along the remaining data.

Description

- Using the data gathered from the flight path algorithm we use the clustering algorithm DBSCAN (Density -Based Spatial Clustering of Applications with Noise), which sorts datapoints into clusters, or ignores them as noise. We take the two largest clusters along with any other clusters with at least 100 datapoints and we are left with the most popular flightpaths for that airport. For each cluster that we are left with we perform a kernel density estimate on the data to get a ranking of the clusters datapoints by density. Using this we can filter out the bottom 20% of datapoints to ensure that we are only using the core points. Then we perform polynomial regression on the remaining data to find a trendline that corresponds with an estimate of the average path taken by planes on their arrival or departure. I plan to incorporate Convolutional Neural Networks (CNNs) and Transformers to make my algorithm more precise and intuitive for handling flight and location data. These methods are expected to enhance pattern recognition and trend forecasting, leading to more reliable predictions and insights.

Spots

Goal

- Get parks within a certain distance of each plotted trendline.

Description

- Using the trendlines calculated in the clusters algorithm we can calculate a series of spots along the line at equal intervals. Since this is a polynomial line, we use calculus to calculate the arc length in intervals. Then we use those points as our basis for our search. We perform a Google Maps Places Nearby Search API call to get all parks within a radius equal to our point interval distance and add all our park results into a set which will exclude any duplicates. Next, we calculate spot specific attributes such as the distance from the flightpath, distance from the airport, and the average altitude of

nearby flights. To get the distance from flight path we must use a minimize function which takes different values of x along the line, calculates the y value for those values, and then calculates the distance between the spot and the point on the line. The minimize function will return the smallest distance value it gets as the closest distance to the path. The distance from airport function uses an easy function that takes the two coordinate pairs, the spot, and the airport, and returns the distance between them in miles. The average altitude function searches for all flights within a radius equal to the interval distance used for the points on the path and returns the average altitude for all those flights. Sometimes this will return nan as it is possible for my datapoints to not appear uniformly throughout the cluster, so if on our first attempt we do not return an average altitude we increase the search radius by 0.1 miles and try again. Once we have compiled all the data and all its attributes, we are left with a list of parks, along the most popular flightpaths for any given airport.