Wing Watch Project Description

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.

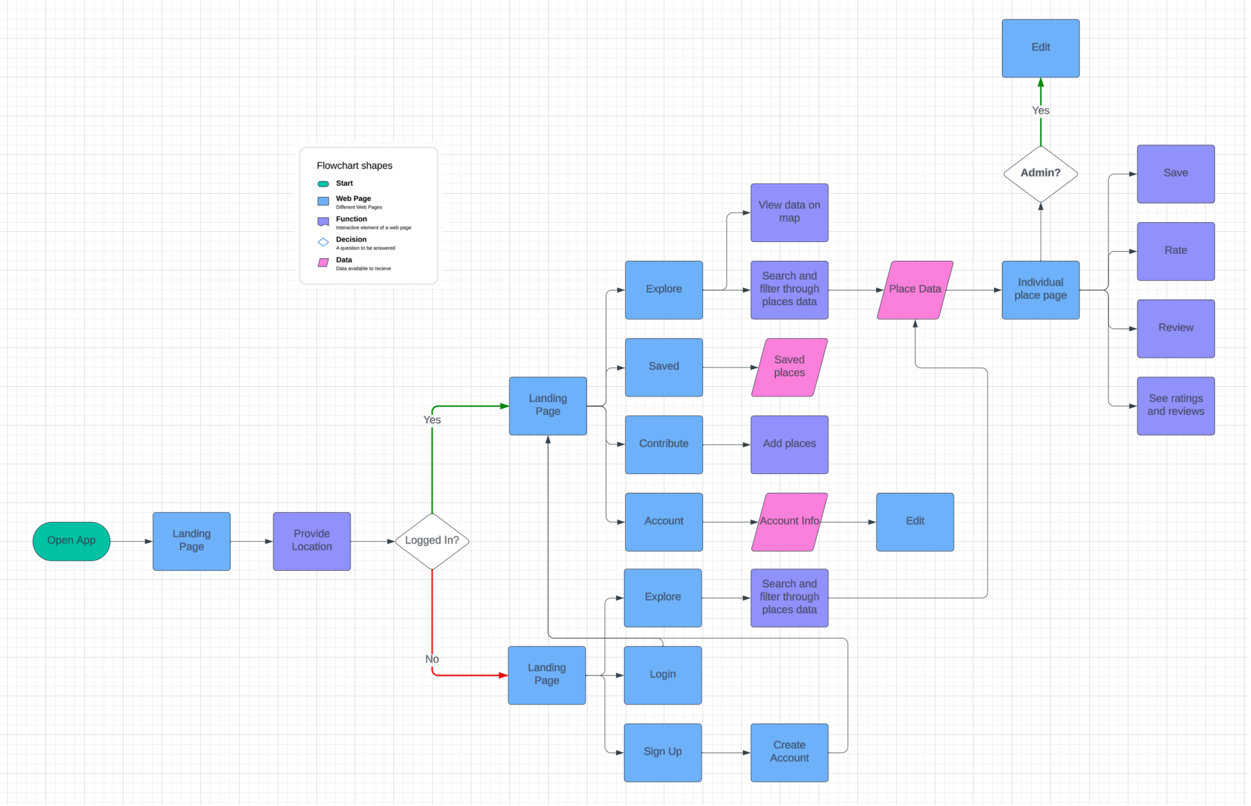
As a user, I would like to be able to externally share places, so that I can engage my friends with plane spotting.

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.

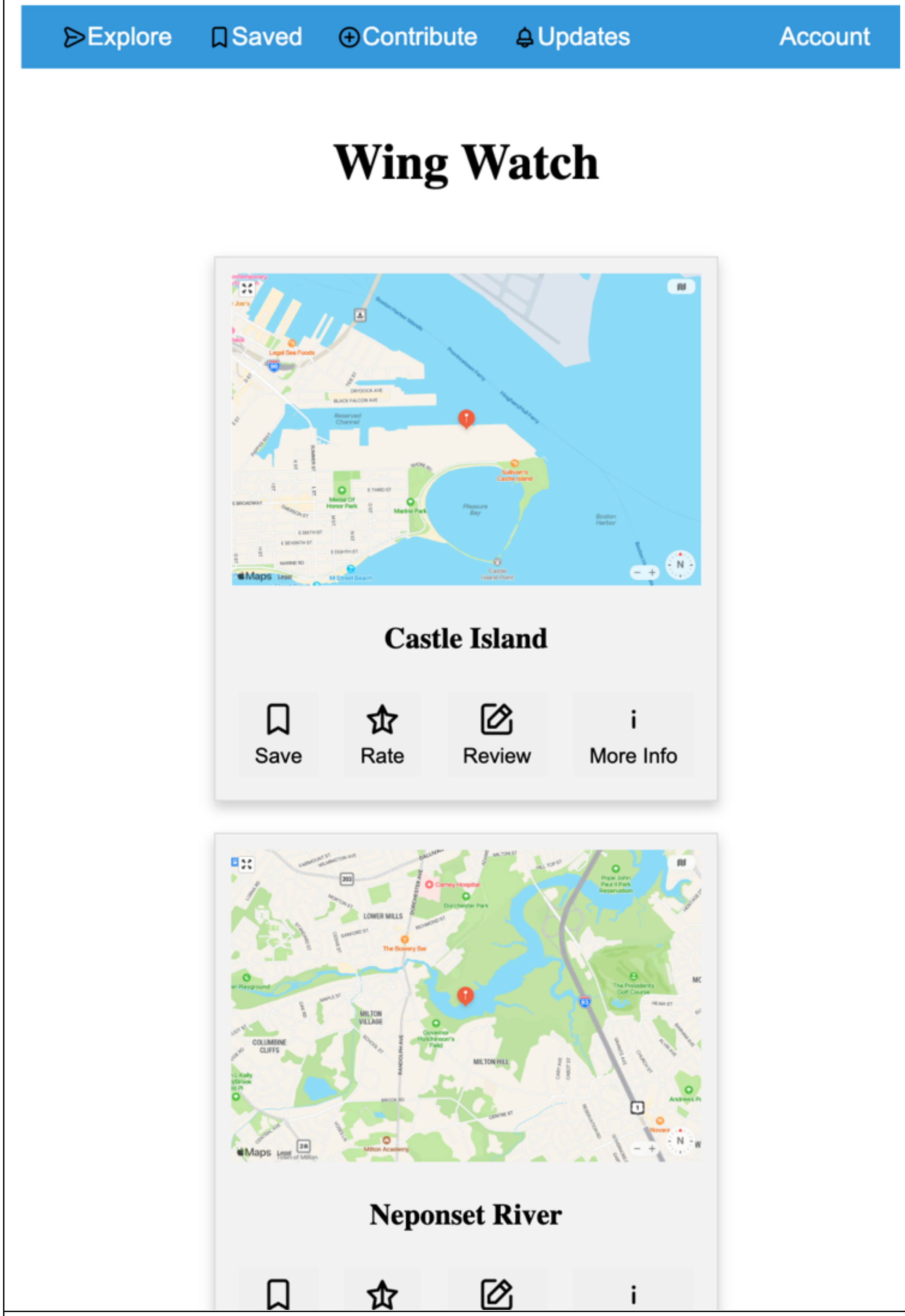
As an admin, I would like to be able to add places, and review, accept, or deny suggested places from users, so that I can help cultivate a more user driven space.

Flow Diagram



Mockups/Wireframes

Landing Page

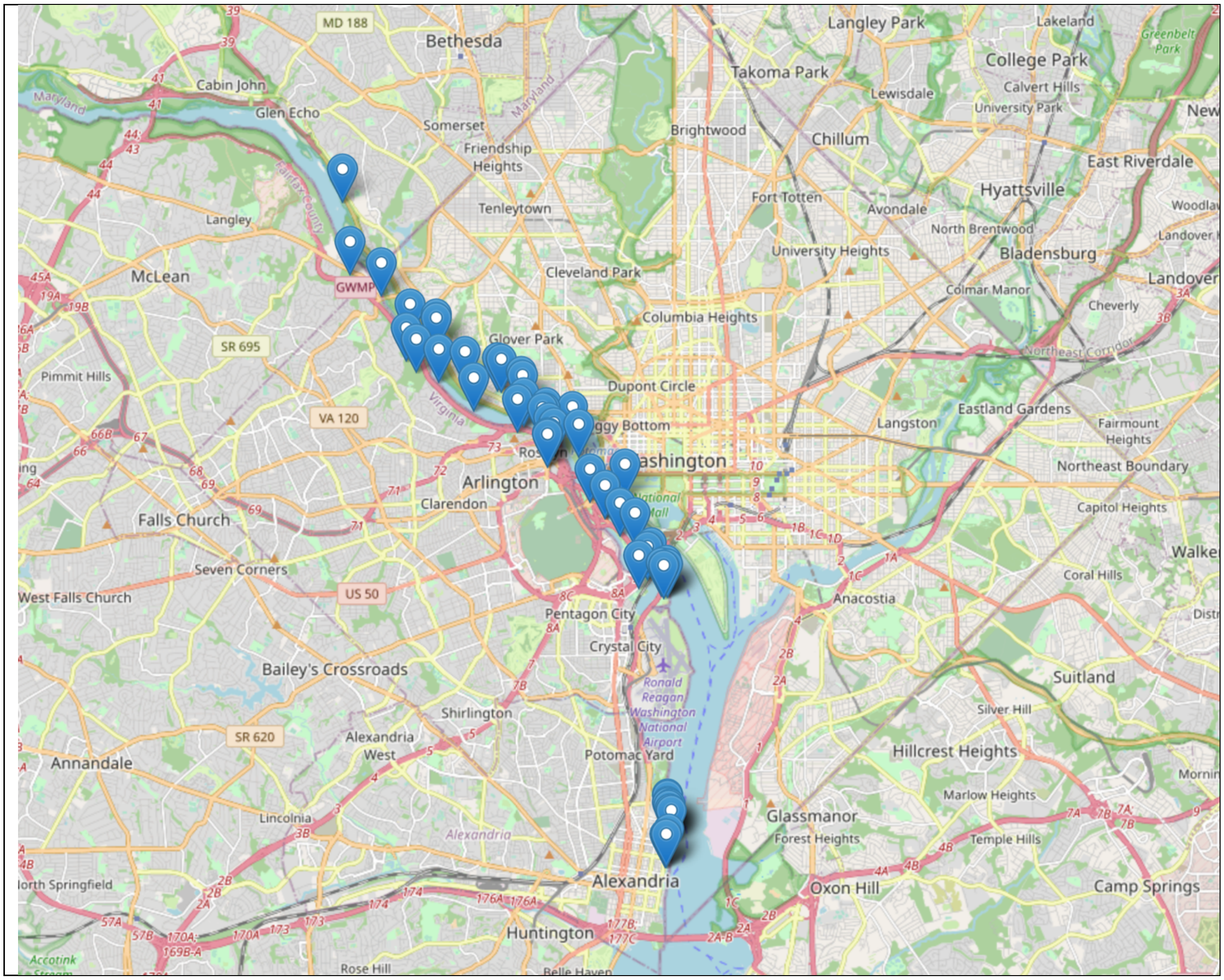


Wing Watch

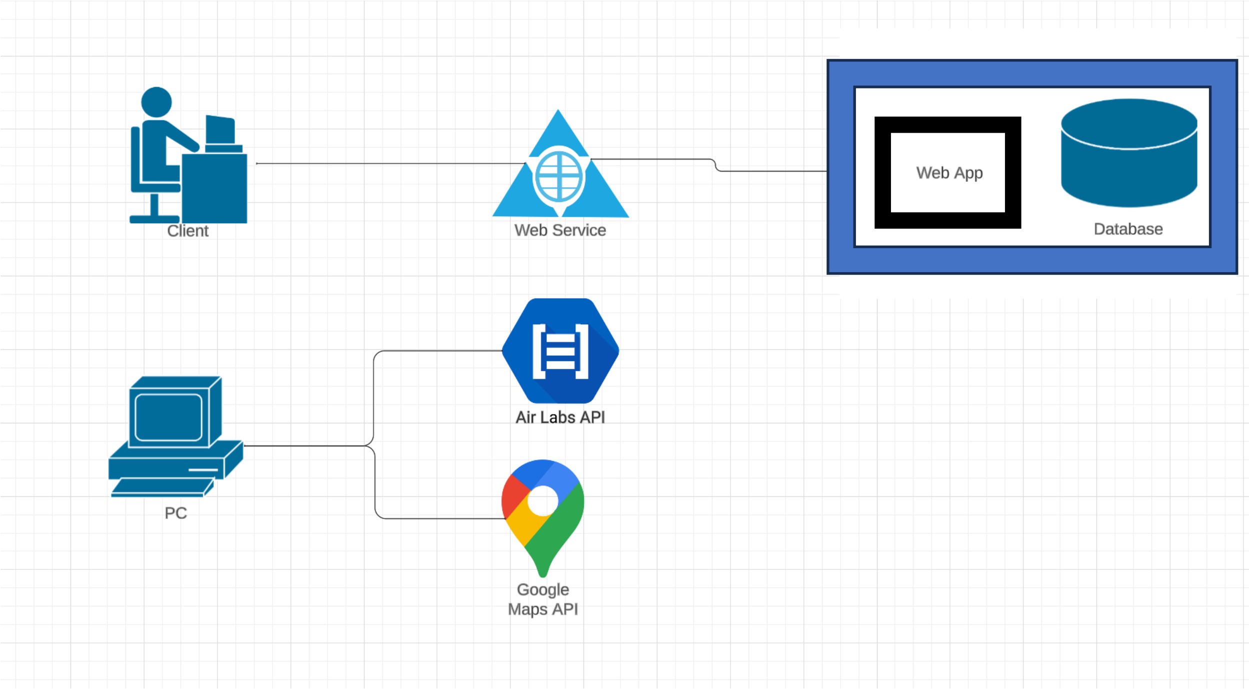
Castle Island

Neponset River

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.

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.