Welcome to Bird App!

# Introduction

The goal of our application is to help a user who is interested in bird watching to plan a trip from scratch with only their starting address. The user must choose a specific bird species and/or location that interests them, and then get back directions to the birdwatching spot. In terms of architecture, our application runs on serverless synchronous API RESTful architecture. The contributions of our tool are that it is user-friendly, and it releases the user from the burden of interacting with more complex APIs or understanding more complex outputs. Our tool integrates several APIs in one, interacts with a cloud database that is integrated in our architecture system, and manipulates the outputs of APIs such that they’re easily readable.

# API Architecture Design

Figure 1 shows the architecture of our API. As shown in the figure the client interacts with an API that we have created through API Gateway. When the client calls our API through its corresponding API URL, it automatically triggers the corresponding lambda function that is associated with the specific API resource method. The lambda functions will then interact with external APIs or with our RDS and perform the necessary computations or extract the necessary data. The lambda functions also have access to the necessary access keys for the Simple Email Service (SES) method to increase security for our service. The reason we chose lambda functions is because we don’t use big data in our computations and so the limitations in terms of computation time (<15 min) do not affect us. The four external APIs that our application interacts with are as follows:

1. [Geoapify- Forward Geocoding API](https://apidocs.geoapify.com/docs/geocoding/):

This API was used to convert addresses to global latitude and longitude coordinates.

1. [Geoapify- Forward Routing API](https://www.geoapify.com/routing-api/):

This API was used to return the optimal route from x to y given a specified preferred mode of travel (walk, bike, train, car).

1. [Ebird API](https://documenter.getpostman.com/view/664302/S1ENwy59):

This API was used to retrieve recent bird observations in an area.

1. [Nominatim API](https://nominatim.openstreetmap.org/search):

This API was used to convert addresses to global latitude and longitude coordinates.

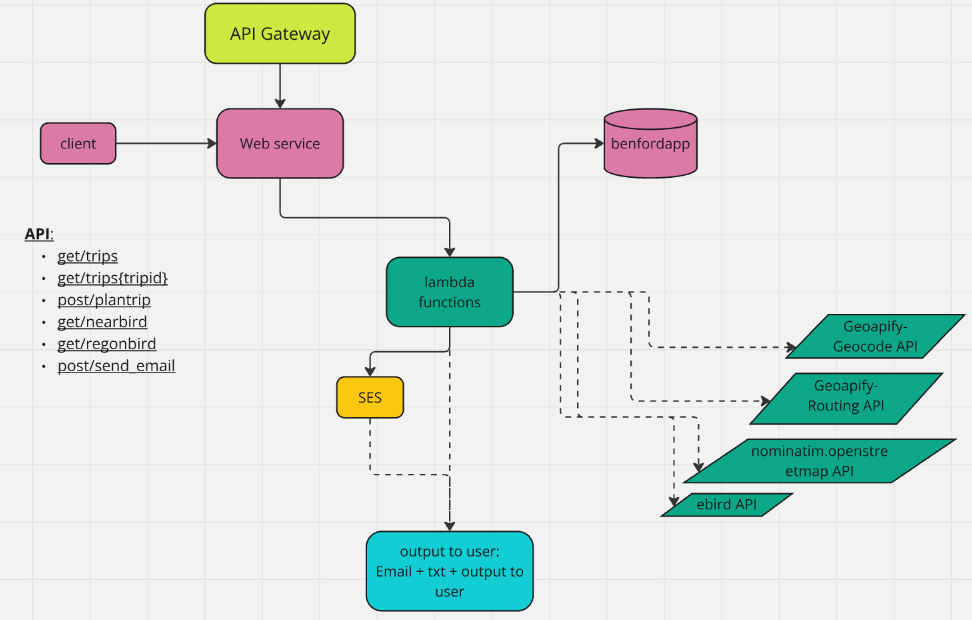


Figure 1: API Architecture for the Application

# Server-side

## Relational Database Service (RDS)

We utilized the RDS component of AWS where we built a table to save past trips’ data based on all previous requests by past users to guide new and existing users. The table was created using sql. It included the following columns:

* Trip ID
* Common bird name
* Startin location
* Ending location
* Transportation mode
* Distance of the trip in Km
* Instructions to arrive at destination

## Non-trivial Operations:

Our application includes several computational and non-computational operations, some of which are listed below:

* Integrated and interacted with 4 external APIs for data.
* Extracted, processed, and combined data, then uploaded it in a database and generated downloadable text files.
* Send emails through lambda function using AWS Simple Email Services (SES).
* Sorted json files of bird observations based on most recent observations or the highest number of observations at one location.
* Computed the distance between two sets of coordinates manually using the Manhattan distance approximation, which is commonly accepted by civil engineers for grid-like networks such as those in the US.
* Computed the durations between the time of request and the time of observations and filtered out observations that are not recent enough (older than 24 hours).

## Lambda Functions & API Configuration

Our application is built on 6 lambda functions and two API urls with 5 total methods. The first API URL ( <https://ssgmv6u3w1.execute-api.us-east-2.amazonaws.com/proj_prod>) integrates 5 lambda functions as methods as listed below:

1. **Trip planner (/plantrip POST):** 
   * Interacts with our RDS to save data
   * Interacts with Geoapify- Geocode API
   * Interacts with Geoapify- Routing API
   * Computes the distance of the trip
2. **Download Trip (/trips{tripid} GET):**
   * Interacts with our RDS to extract data and report to user
3. **Past Trips (/trips GET):**
   * Interacts with our RDS to extract data and report to user
4. **Nearby Birds(/nearbird GET):** 
   * Interacts with nominatim.openstreetmap API
   * Interacts with ebird API
   * Sorts the observations by highest number of observations per location
   * Reports to user
5. **Region Birds(/regionbird GET):**
   * Interacts with ebird API
   * Sorts the observations by the most recent
   * Computes the duration between the time of request and observation
   * Filters out old observations (more than 24 hours ago)
   * Reports to user

The second API url (<https://rpo6ydyluh.execute-api.us-east-2.amazonaws.com/prod>) integrates one lambda function:

1. **Send\_email (/send\_email POST)**
   * Gets recipient’s email address and trip details from Client.
   * Sends user email with Trip details.

# Client-Side

The client only needs two configuration files that include the two needed API urls which are provided in two files (benfordapp-client-config.ini & email-client-config.ini) which get called by the corresponding functions in the db\_test.py file which the user runs. When the user runs the db\_test.py file, they will get prompts for 5 options:

1. To show past trip request by all previous users + Send email with a selected trip details and instructions;
2. To plan trip to the location of interest for bird watching, by different modes;
3. To download trip details and instructions;
4. To see nearby birds within 50 km radius from your address;
5. To see birds in the region of your request.

The ideal order is to select functions 1, 4 or 5 first to see the available options in terms of bird watching locations and then to select option 2 to plan a trip, followed by 3 or 1 to save the details of the trip.