## horizontal line



Rescue Our Friends (ROF)

An interactive web application that helps connecting pet seekers and their new friends

CIS550 - Project Final Report

28.04.2021

**─**

Project Completed by:

Di Hu | Yunyang Li | Eva Liu | Derrick Mo

# Introduction

According to the latest studies published by The American Society for the Prevention of Cruelty to Animals (ASPCA), approximately 6.5 million companion animals are entering US animal shelters nationwide each year. While approximately 50% of these pets were adopted each year thanks to ASPCA and their allies’ continuous efforts and relentless dedications, the other not-so-lucky half, approximately 3.3 million, stayed behind the steel bars with no faults to be found. Unfortunately, a significant contributor of this number is us, the one who gave them intimated first names, referred them with humanized pronouns and offered them many personifications. Nevertheless, we ended up sending them to the local animal shelter due to our situational inconveniences.

Fortunately, there are many ways we can help our beloved counterparts. We can choose to adopt from animal shelters instead of purchasing from incentivized breeders. This change helps to reduce the number of pets that will eventually go to the animal shelters and increase the number of adoptions. To contribute our parts to the cause, we want to create a user friendly web application that helps connecting pet-seekers and abandoned pets. We hope through our innovative approaches and friendly recommendation systems, pet-seekers can conveniently locate their ideal pets and bring our beloved friends to have a place called home again.

In a nutshell, our web application has three main functionalities:

1. Help pet seekers to identify pets with their desired attributes such as animal type, pet breed, color age, and etc. in their local area and display these candidate pets and their respective animal shelters on the Google Map Interface. Users are able to save their search criteria by logging in and resume their search the next time.
2. Help pet seekers to find appropriate and affordable pet related items from our selected e-commerce platforms (Due to time constraints, we only have eBay for now). This function is especially helpful for the first-time pet owner and can serve as a pet item checklist to create a smoother adoption experience and help create a cozy environment for the adopted pet.
3. Help to raise awareness by creating a separate fun fact page displaying interesting and entertaining information about pets in our databases. Whether you are interested in learning how your state or county does in terms of lowering pets in the animal shelters or how abundant the financial resources are changing owners’ behaviors, you can find some clues here.

Last but not the least, here are the pet lovers who are making this happened:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| **Di Hu and Selena**  diihuu@seas.upenn.edu | **Yunyang Li and Chopper**  liyy@seas.upenn.edu | **Eva Liu**  **MacArthur & Curry**  yubeiliu@seas.upenn.edu | **Derrick Mo and Momo**  moderr@seas.upenn.edu |

# 

# Technology and Architecture

The front-end of the website will be designed using HTML, CSS, JavaScript, React, and other suitable tools and libraries to allow users to interact with the website. We also plan to use existing web development tools such as Google Maps and third-party APIs to facilitate better user experience.

The back-end of the website will be constructed using Python to perform data cleansing, using JavaScript including Node.js framework to perform data queries, execute pet selection algorithms, and bring the results back to the front-end. There will be two types of data: Dynamic data pulled from the third-party API tools so the user can see timely search results; static data pulled from our database architect running on AWS to supplement query results. The final database design will be implemented based on the requirements in the later project milestones.

In summary, we used the following technologies to create the web application:

Frontend: React, Recharts, Redux

Design: CSS, bootstrap

Backend: Node.js, Express.js, MySQL, Python/Pandas

API: Google Maps, Petfinder API fetch

Data scraping: BeautifulSoup, Pandas

# 

# Data

\*\* Data cleaning code for Animal data and Organization data are provided via link to Colab

Our data are collected from various sources and pre-processed before they are uploaded to the AWS database to facilitate smoother user experience and reduce computational loads. For example, the income data file pulled from the IRS website originally contains 153 schemas and 165,935 tuples where most of the schemas are irrelevant to our exercise. After preliminary cleaning, the database contains only 15 schemas and 27,558 tuples, which is still relatively large but much more manageable when we are performing nested joins. Please consult below for the full data list and their primary usages:

## eBay data table

**Source**: eBay [https://ebay.com/](https://developer.ebay.com/)

**Description**: We built a scraper to scrape data from ebay.com. This data table contains common pet associated items that can be found on eBay. This table is directly linked to Pet Supplies and Pet Type Mapping tables below.

**Statistics**: 4 keys, 6 schemas, and 1,482 tuples

**Usage**: Once the user has completed the search criteria and triggers the pet finding algorithm, the application will pull related items based on the type of pets. For example, if the user wants to find a dog, then the app will return dog food and other dog supplies.

## Organization data table

**Source**: PetFinder API - <https://www.petfinder.com/developer/>

**Description**: This table stores all available animal shelters’ information on the PetFinder API. The table contains the name of animal shelter, contact emails and addresses.

**Data cleaning using Colab:**

<https://colab.research.google.com/drive/18vRMVfGDps6h686ymsAhCy9tBMmn9v__?usp=sharing>

**Statistics**: 1 key, 6 schemas, and 3,000 tuples

**Usage**: We use the location information available in this table to show all available for adoption pets in the same area and display our results on the Google Map interface.

## Animal data table

**Source**: PetFinder API - <https://www.petfinder.com/developer/>

**Description**: Animal data are pulled directly from the API and pre-processed before upload to AWS. The table contains detailed attributes such as shelter, type, breed, and age, etc.

**Statistics**: 3 keys, 28 schemas, and 89,916 tuples

**Data cleaning using Colab:**

<https://colab.research.google.com/drive/18vRMVfGDps6h686ymsAhCy9tBMmn9v__?usp=sharing>

**Usage**: Animal data will be used in three primary ways: First, match customers’ search criteria with adoptable pets in the table and return pets that match closest to the user’s desired attributes; second, once animals are selected, the system will trigger the shop recommendation system and return eBay items that matches with the user’s selected animal type and breed; last but not the least, these data will be used in fun fact generator and finds interesting facts about our beloved animals.

## Income Data data table

**Source**: Internal Revenue Services (IRS) - <https://catalog.data.gov/dataset/zip-code-data>

**Description**: This is a publicly available database and is widely used in understanding local demographics and policy related analysis such as income gap and affordability analysis.

**Statistics**: 1 key, 15 schemas, and 27,558 tuples

**Usage**: We are trying to understand if abandoning a pet or adopting a pet is positively or negatively related to an individual or a family’s overall income level by examining the population over adoptable pets on state and local county level. This data table is mainly used in populating fun facts and the “did you know” section.

## Pet Supplies Mapping data table

**Source**: Rescue Our Friends (ROF)’s proprietary data table!

**Description**: This table serves as an interim table and contains categories for our supplies.

**Statistics**: 2 keys, 3 schemas, and 24 tuples

**Usage**: This table is used in linking search criteria and eBay recommendation system.

## Pet Type Mapping data table

**Source**: Rescue Our Friends (ROF)’s proprietary data table!

**Description**: This table serves as an interim table and contains categories for animal types.

**Statistics**: 1 key, 2 schemas, and 4 tuples

**Usage**: This table is used in linking search criteria and eBay recommendation system.

## Zip Code Mapping data table

**Source**: Google - <https://developers.google.com/maps/documentation/javascript/places>

**Description**: Provide zip code matching with other tables and shows locations on the Google Map interface. The data format for this table meets our map’s requirements well.

**Statistics**: 1 key, 7 schemas, and 42,633 tuples

**Usage**: Please note our organization data table, animal data table and income data table are containing similar information. However, given the size of the animal data table and the size of the income data table are considerably larger, having a separate zip code matching table allows us to increase the speed of complicated queries.

# 

# Database

## DDL Considerations

1. Each table has at least one primary key that references to one or more tables. As SQL database automatically generates indexes for primary keys, this implementation helps to facilitate query efficiency.
2. Most schemas in our tables were renamed before uploading to AWS database and we have agreed on the following naming conventions:

* Decided not to use camel case but underscore since SQL does not have active uppercase and lowercase differences; therefore, we believe using underscore (\_) to be easier to recognize schemas with two or more words.
* Using any acronyms is strongly discouraged unless there is a strong reason to do so for internal consistency and review clarity. This agreement works well and we have only been using two acronyms in our DDL: ZipCode which is a commonly used acronym and AGI for income as the full definition of AGI is too lengthy while we want to avoid using “income” which is easy to be misrepresented.
* Given the restrictions of our system that does not allow reserved words to be used (e.g. state) and duplicate schemas so we agreed on duplicating the last letter when a reserved word is being used (in income data table, we used “statee” instead of “state”) and adding a number when there are two or more schemas that have the same name (in animal data table, we have two addresses and one is named add1 and the other one is named add2 ).

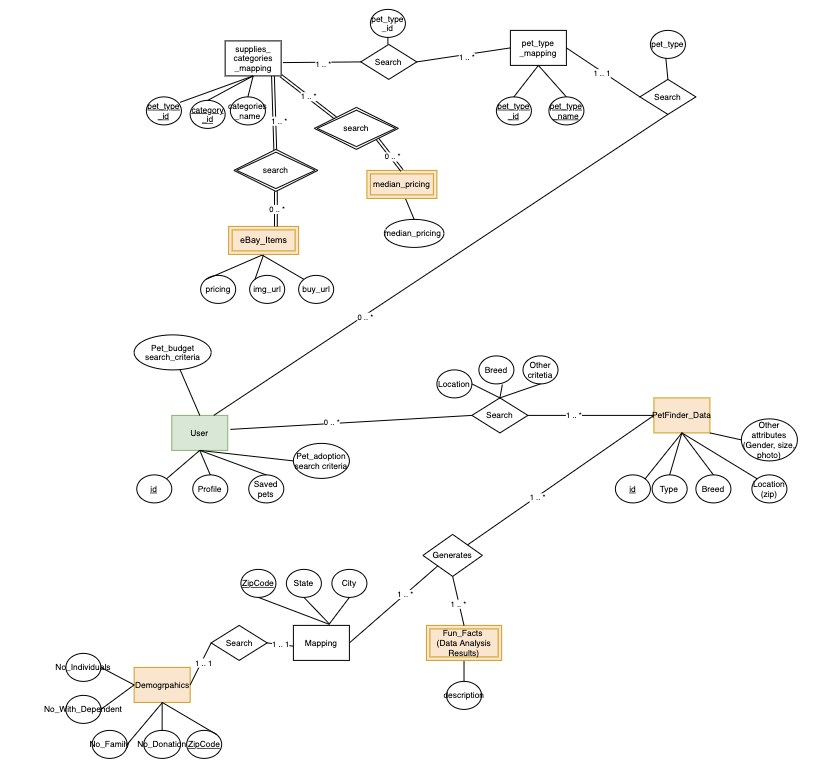
## 

## 

## 

## 

## ER Diagram



# 

# 

# Queries

**Query 1: funFactWithColors**

WITH CAT AS

(SELECT ani.color AS color, Count(\*) AS count\_cat

FROM pets.animalData AS ani

WHERE color <> "none" AND ani.animal\_type = "Cat"

GROUP BY ani.color

ORDER BY ani.animal\_type, count\_cat desc

),

DOG AS(

SELECT ani.color AS color, Count(\*) AS count\_dog

FROM pets.animalData AS ani

WHERE color <> "none" AND ani.animal\_type = "Dog"

GROUP BY ani.color

ORDER BY ani.animal\_type, count\_dog desc

),

HORSE AS (

SELECT ani.color AS color, Count(\*) AS count\_horse

FROM pets.animalData AS ani

WHERE color <> "none" AND ani.animal\_type = "Horse"

GROUP BY ani.color

ORDER BY ani.animal\_type, count\_horse desc

),

RABBIT AS (

SELECT ani.color AS color, Count(\*) AS count\_rabbit

FROM pets.animalData AS ani

WHERE color <> "none" AND ani.animal\_type = "Rabbit"

GROUP BY ani.color

ORDER BY ani.animal\_type, count\_rabbit desc

),

ALLCOLOR AS (

SELECT ani.color AS color, Count(\*) AS count\_all

FROM pets.animalData AS ani

WHERE color <> "none"

GROUP BY ani.color

ORDER BY count\_all DESC

LIMIT 15

)

SELECT ALLCOLOR.color, IFNULL(CAT.count\_cat,0) AS cat, IFNULL(DOG.count\_dog,0) AS dog, IFNULL(RABBIT.count\_rabbit,0) AS rabbit, IFNULL(HORSE.count\_horse,0) AS horse

FROM ALLCOLOR

LEFT OUTER JOIN CAT ON CAT.color = ALLCOLOR.color

LEFT OUTER JOIN HORSE ON HORSE.color = ALLCOLOR.color

LEFT OUTER JOIN DOG ON DOG.color = ALLCOLOR.color

LEFT OUTER JOIN RABBIT ON RABBIT.color = ALLCOLOR.color

**Query 2: getStatesWithFamiliesLoveLargeDogs**

Family ratio: Based on IRS's number of family returns over total number of returns

Size Matrix: Calculate average pet size score where small = 1, medium = 2, large = 3 and x-large = 4

Multiply family ratio and size matrix and sort in a descending order

States that are displayed on the list are the state that have most families that love large dogs

SELECT inc.state, inc.familyWeight as Family\_Weight, score.score as Size\_Score,

inc.familyWeight \* score.score as indexScore

FROM

(SELECT inc.statee as state, sum(inc.n\_JOINt\_returns) / sum(inc.n\_total\_returns) as familyWeight

FROM pets.incomeData inc

group by inc.statee) inc

JOIN

(SELECT small.state1 as state,

(small.small / (small.small + med.med + large.large + xlarge.xlarge) \* 1 +

med.med / (small.small + med.med + large.large + xlarge.xlarge) \* 2 +

large.large / (small.small + med.med + large.large + xlarge.xlarge) \* 3 +

xlarge.xlarge / (small.small + med.med + large.large + xlarge.xlarge) \* 4) as score

FROM

(SELECT ani.state1, count(PetSize) as small

FROM pets.animalData ani

WHERE petsize = 'small'

group by ani.state1) small

JOIN

(SELECT ani.state1, count(PetSize) as med

FROM pets.animalData ani

WHERE petsize = 'medium'

group by ani.state1) med

JOIN

(SELECT ani.state1, count(PetSize) as large

FROM pets.animalData ani

WHERE petsize = 'large'

group by ani.state1) large

JOIN

(SELECT ani.state1, count(PetSize) as xlarge

FROM pets.animalData ani

WHERE petsize = 'extra large'

group by ani.state1) xlarge

on small.state1 = med.state1

AND small.state1 = large.state1

AND small.state1 = xlarge.state1) score

WHERE inc.state = score.state

order by inc.state

**Query 3: getAnimalNames**

Simply return top 10 names of each pet by state and city

Step 1: Choose a type FROM dog, cat and rabbit and a state FROM 50 states

Step 2: Send the SELECTion to the database and return top 10 cities

Step 3: Choose a city FROM the 10 cities list

Step 4: Return top 10 names given the input

WITH dognames AS(

SELECT ani.petname AS name1, ani.animal\_type, count(\*) AS count

FROM pets.animalData ani

WHERE ani.animal\_type = "Dog"

GROUP BY name1

ORDER BY count DESC

LIMIT 60

),

catnames AS

(SELECT ani.petname AS name2, ani.animal\_type, count(\*) AS count2

FROM pets.animalData ani

WHERE ani.animal\_type = "Cat"

GROUP BY name2

ORDER BY count2 DESC

LIMIT 60

)

SELECT dognames.name1,dognames.count AS count\_dog,catnames.count2 AS count\_cat

FROM dognames

INNER JOIN catnames ON dognames.name1 = catnames.name2

ORDER BY dognames.name1

**Query 4: getIncomePopulationAnimalsByState**

SELECT ani.state1, SUM(inc.income)/1000 AS income, SUM(inc.population)/1000 AS population, count(\*) AS animal

FROM pets.animalData ani

JOIN (select zip, inc.AGI AS income, inc.n\_individual AS population

FROM pets.incomeData inc

FROM pets.zipcodeMapping zip

ON inc.zipcode = zip.zip) inc

ON inc.zip = ani.zipcode

GROUP BY ani.state1

ORDER BY ani.state1

**Query 5: getEbayData**

SELECT \*

FROM (ebayData e

JOIN petTypeMapping p

ON e.pet\_type\_id = p.pet\_type\_id)

JOIN petSuppliesMapping s

ON (e.pet\_type\_id = s.pet\_type\_id AND e.category\_id = s.category\_id)

WHERE e.pet\_type\_id = '${pID}' and e.category\_id = '${cID}'

# 

# Performance Evaluations

We tested our queries’ processing efficiencies in both SQL WorkBench running environment and in our web application’s running environment. While we did not set a hard cap during our beta testing period, we have softly agreed that each query should not have a processing speed to be longer than 0.5 seconds in the SQL WorkBench running environment and target a max of 1.0 - 1.5 seconds in our website’s running environment. In the actual testing, we ran each query 10 times and removed the longest and shortest processing time and then calculated the average processing speed. In the end, we are glad to report that we were able to meet our initial goals of < 0.5 seconds in the SQL WorkBench running environment and < 1.5 seconds in web application running environment.

|  |  |  |
| --- | --- | --- |
|  | **SQL WorkBench**  **Processing Speed\*** | **Web Application**  **Processing Speed\*\*** |
| **Query 1: funFactWithColors** | ~0.203 seconds | ~0.897 seconds |
| **Query 2: getStatesWithFamiliesLoveLargeDogs** | ~0.392 seconds | ~1.152 seconds |
| **Query 3: getAnimalNames** | ~0.266 seconds | ~0.975 seconds |
| **Query 4: getIncomePopulationAnimalsByState** | ~0.402 seconds | ~1.186 seconds |
| **Query 5: getEbayData** | ~0.047 seconds | ~0.663 seconds |

\*Calculated using application provided timer in the output console

\*\*Calculated using Date().getTime() method in the JS running environment

# 

# Technical Challenges

1. **Hooks.**

We learnt some basic React.js, but we had never tried Hooks. With Hooks getting more and more popular, we decided to utilize Hooks in our project. It took a long time for us to truly understand useState Hook and useEffects Hook. We spent hours to solve the problem of how to use data immediately stored in useState. After we got used to Hooks, we found it is really convenient compared to the old way of using State.

1. **Recharts.**

This recharts library is specialized for data visualization in JavaScript language. The library requires a specific format of data input. This puts a strict requirement on the SQL data output format.

1. **Tinder card library**

Though the library itself is quite user-friendly, the application of the Tinder card library was challenging since we need to make it match with our design and information.

1. **Google Maps API**

We used the functions like display/zoom by location and put self-defined markers on the map. It’s challenging and fun to apply it to our Node.js library.

Functional wise, location search results cannot be strictly limited to specific geographic regions. This limitation restricted us from using the address, and we alternatively use Geographic coordinates to locate our markers.