# Final Project Write-Up - CIS450/550

Lucy Chai, Jordan Hurwitz,
Margaret Nolan, Francisco Selame

June 4, 2017

## 1 Introduction

*DogFinder* is web-app that simplifies the dog adoption process. A simple quiz is used to determine user preferences and find dog breeds that are suitable for them, while a database of zip codes and coordinates is used to determine areas near the user. This information is used to query a database of dogs available for adoption, in order to find dogs that are both close by and good matches. Users are able to create profiles in order to save their favorite dogs.

## 2 Modules and architecture

A combination of different technologies were used for this project. MySQL and DynamoDB databases, hosted using AWS, were used for data storage. Vogels was used a s DynamoDB data mapper, Node.js was used for server-side scripting, and Express.js was used to handle routing. The front-end was built using embedded javascript, HTML, and CSS.

We structured the project into the following sections:

- Models: A javascript file is included for each of the databases. These files handle setting up the connection with AWS and include the queries.
- Routes: Handles routing logic for all get and post requests
- Views: Contains EJS files that make up the front end. A "components" submodule contains views that are used throughout the website
- Images: Contains image files used in the view files

## 3 Key features of the project

Some of the project's key features include:

- Core Functionality:
  - Users are able to log in to their profiles, and take a personality quiz, from which their preferences are inferred. Users are then able to

input their zip code and see a list of dogs available for adoption that best match their needs, and are close by. This involves a query to determine nearby zipcodes, a query to determine compatible dog breeds, and query to the pets table to find pets that are both nearby and compatible, and a query to the user table if the user wants to save the pet.
  - Users are also able to directly search for dogs in their geographical proximity without having to take the personality quiz.

- Additional Features
  - API Integration: A Twitter embedded search timeline was used as a motivating example, displaying posts with the hashtag #puppy. Facebook social plug-ins were used to enable users to share content and support adoption organizations.
  - User profiles were implemented, in order to allow users to save the state of their search by saving and managing their favorite dogs. Passwords are hashed before they are stored in order to ensure security.
  - Validator, an npm package, as well as custom regex checks, were used in order to avoid SQL injections from malicious users.

## 4 Technical challenges and how they were overcome

- We extracted the data about adoptable pets via the Petfinder API. However, the API limits the amount of queries we can make so we couldn't pull all of their data. Thus, we decided to first use HTML scraping to get the 1000 most populous zip codes, and then queried the Petfinder database based on these zipcodes.

- We intended to use the vogels wrapper for our dynamoDB queries. While this worked well for the users database, we found that we needed more complex queries for the pets table that could handle selections on multiple optional characteristics. This was in addition to the simpler queries of getting a pet based on pet ID. Thus, we decided to migrate all of the pets queries off of vogels and use the aws-sdk node module to handle dynamodb queries to the pets table instead.

- Because many of our queries relied on data from both the SQL and NoSQL database instances, we were forced to find a way to pipeline the results of one database call to another in an efficient manner. To overcome this, we devised a callback system in the database files to pass data between instances.

## 5 Data instances

*DogFinder* uses the following databases:

- MySQL:
  - Zipcode: This table includes a list of US zipcodes, and their latitudes and longitudes. Table uses zipcode as a clustered index.
  - Breed: This table includes a list of dog breeds, along with a score for 31 different characteristics for each breed, such as friendliness and intelligence. Table uses username as a hashkey. Table uses breedname as a clustered index.
- DynamoDB:
  - Pets: This database contains a list of pets available for adoption, and key characteristics like their locations and descriptions. Table uses pet id as a hashkey, and zipcode as sort key.
  - Users: This database contains a list of usernames, hashed passwords, and profile information like dogs saved. Table uses username as a hashkey.

## 6 Data Sources, Cleaning and Import

Data sources:

- MySQL:
  - Zipcode: A CSV file was obtained from federalgovernmentzipcodes.us which included all us zip codes and a set of attributes for them. Zip codes, latitudes, and longitudes were then extracted from this file.
  - Breed: BeautifulSoup was used to HTML scrape dogtime.com, in order to determine characteristic scores for over 200 breeds.
- DynamoDB:
  - Pets: The Zipcode database was used to determine the most populous zip codes in the US. Then the petfinder.com API was used to query those locations for available pets. The API calls returned JSON objects, which were then used to populate the database.
  - Users: Database was populated during web-app usage, as user profiles were created.

Cleaning and Import: Boto3 and MySQL Connector were used to interface with AWS during the data import process. Breed scores and zip code tables were also manually cleaned, and missing data values were inserted.

## 7 Performance Evaluation

In order to understand the application's latency, we explore how the execution time of sample queries on our two SQL tables change when the database is locally hosted, or hosted on AWS (both with and without a cache). The following average latencies are 5-trial averages, and are presented in milliseconds:

Table 1: Average Execution times, in milliseconds

| | Local Execution | RDS Execution (No Cache) | RDS Execution (Cache) |
|---|---|---|---|
| Breed | 0.7172 | 179.23 | 33.4 |
| Zip Code | 45.58 | 189.8 | 45 |

As expected, local execution is the fastest, followed by execution on RDS with an active cache, and then RDS execution without a cache. The difference is execution times between local and RDS instances is due to the delay in sending and receiving network requests, as evidenced by the largely comparable no-cache, RDS execution time across both the Breed and Zipcode tables.

Although it proved difficult test DynamoDB locally, we measured the average latency over multiple record requests to be 144 milliseconds, which was faster than the RDS instance query time without caching.

## 8 Future Expansion

With additional time, there are a couple of extra features which could be implemented to improve *Dogfinder*. Some of these include:

- Google Geolocation API can be integrated into the application in order to determine the user's zip code without explicit input
- Dog suggestions could be refined by building a recommendation engine based on the dogs users save to their profiles. This would mean user preferences could be determined beyond the results of the quiz. Machine learning methods like decision trees or logistic regression could prove useful here.
- Automatic queries to the petfinder API could be included in the application so that the database of available pets remains consistent with the petfinder database.