CS373: Software Engineering

10 am, Group 3 - Pets4me Phase II

Connor Sheehan, Rosemary Fortanely, Cristian Garza,
Robert Hrusecky, Andrew Cramer, Dean Torkelson

Section 1: Motivation

The motivation behind Pets4Me was the lack of a single aggregator for pets that are up for adoption in the Austin area. There is never a shortage of animals that need homes and families, but finding the perfect pet requires going to many different shelter websites, visiting many in person, or spending time calling multiple sources for information. However, with Pets4Me, all the data a future owner needs (be it on the shelters, different breeds, or the animals themselves) is presented in one single location, making it easier for animals in need to find their forever home.

Section 2: User Stories

2.1: User Stories for us:

1. **More instances for each model**
    a. As this is a requirement for phase 2, we implemented this. At the time of reporting, we were still only displaying our 3 hardcoded instances of each model, but now we have hundreds to thousands of pets, hundreds of breeds, and many dozens of shelters that users can view.
2. **Make the "adopt a pet today" on homepage responsive**
    a. Our homepage reads “Make a New Friend! adopt a pet today” above our mission statement. We decided to make it such that the link takes you to a random animal’s page, as a sort of “I’m feeling lucky” option.
3. **Interconnectedness of the instances**
    a. As this is a requirement for phase 2, we implemented this. Dog Breed and Cat Breed are the only two models that are not interconnected, but between Shelter and Pet, everything else is interconnected.
4. **Image Sizes for Instance Pages**
    a. At the time of reporting, some of our CSS was off and because of this, images would be stretched weird, cropped weird, or potentially slightly overlap other graphical elements of the page. After we started getting all our data from the APIs, the image sizes were more uniform and this was no longer an issue.
5. **Add Pagination to the Model Pages**
    a. As this is a requirement for phase 2, we implemented this. Our customers suggested a 3x3 grid of cards for each model homepage, and we initially took their advice, but eventually decided that a 4x3 grid fit the style of our page better and allowed users to see more data at once.

2.2 User Stories for CrashSafe:

1. **Include many instances of car brands**
    a. As a user, I would like to see more than 3 instances of car brands. I would like to see a sample selection of these brands on the model homepage. I would like to be able to click on some of the data for it to go to the individual page for the brand.
2. **Fix image resizing on about page for mobile**
    a. As a mobile user, I want the images to behave as expected and all be the same size. Currently, some images are large and others are small. This negatively affects my User Experience.
3. **More data fields in the instances**
    a. I would like to know more about the cars that receive their respective ratings. More data fields like what type of car (crossover, sedan, truck, etc.), car weight, and/or popularity would be great information to have.
4. **Homepage Image Load Time**
    a. As a user, I would like the background image that appears on the homepage to load faster. You can see this issue by visiting crashsafe.me and click on the Home button. You will notice that a gray background appears while the next new image loads.
5. **Many Car Brands**
    a. As a user, I would like to see more car models in the car models page. I think this should first be populated by the most popular cars of this year. It might also be nice for the models to automatically be sorted by rating.

Section 3: RESTful API

GET /models/pets

This endpoint will return all instances of individual pets.

GET /models/breeds

This endpoint will return all instances of breeds.

GET /models/shelters

This endpoint will return all instances of shelters.

GET /models/pets/{id}

This endpoint will return the pet that has the id specified by the parameter.

GET /models/breeds/{id}

This endpoint will return the breed that has the id specified by the parameter.

GET /models/shelters/{id}

This endpoint will return the shelter that has the id specified by the parameter.

We chose these endpoints based on a recommendation from a TA on Piazza.

Section 4: Models

For our three models, we decided on 1) the individual pets up for adoption, 2) the different breeds of dogs, 3) breeds of cats, and 4) the shelters where you could adopt the animals.

Model 1: Pets

The Pets model contains the "dog-ographic" information on the individual animal. We included information about the pet itself, such as name, breed, size, gender, and age. Each pet model also contains information about the adoption, such as where the animal currently is and a link to the shelter's website. From a pet's page, you can be linked to the page for the breed, as well as the page on our website for the shelter.

Models 2 & 3: Dog and cat breeds

The Breeds model contains the information that someone who has never heard of it would need to get an idea of what the breed is like, such as its name and nicknames, species, origin, average lifespan, size, typical traits, and how heavily it sheds. From each breed page, you'll be shown a few links to animals that are up for adoption that match that breed, as well as the shelter that has the most animals of that breed. Though breeds are conceptually very similar, we have them split between dogs and cats in our code due to major differences in the individual attributes for each.

Model 4: Shelters

The Shelter model has essentially every piece of relevant information that someone looking to adopt would need. Including its location, available hours, contact information like emails and phone numbers, adoption policy and mission statements, and of course their social media. If

there is any information that someone is looking for that we don't have, there is a link to the shelter's website on the page.

Section 5: Tools

The primary tools we used were React, TypeScript, Marvel, Postman, Bash, Mocha, Selenium, and Mapbox. Within React, we also used react-bootstrap and Material UI, as these both have a similar look and feel and allow us to easily implement complicated components. We chose to work in TypeScript as opposed to JavaScript to reduce the likelihood of type errors and silly human mistakes. We used Marvel to create mockups of the frontend for us to model our designs after. Postman was used to design our API. Bash was used to deploy our website. Selenium and Mocha were both used to test our frontend. Lastly, Mapbox was used to display a map of a shelter's location on its instance page.

Section 6: Hosting

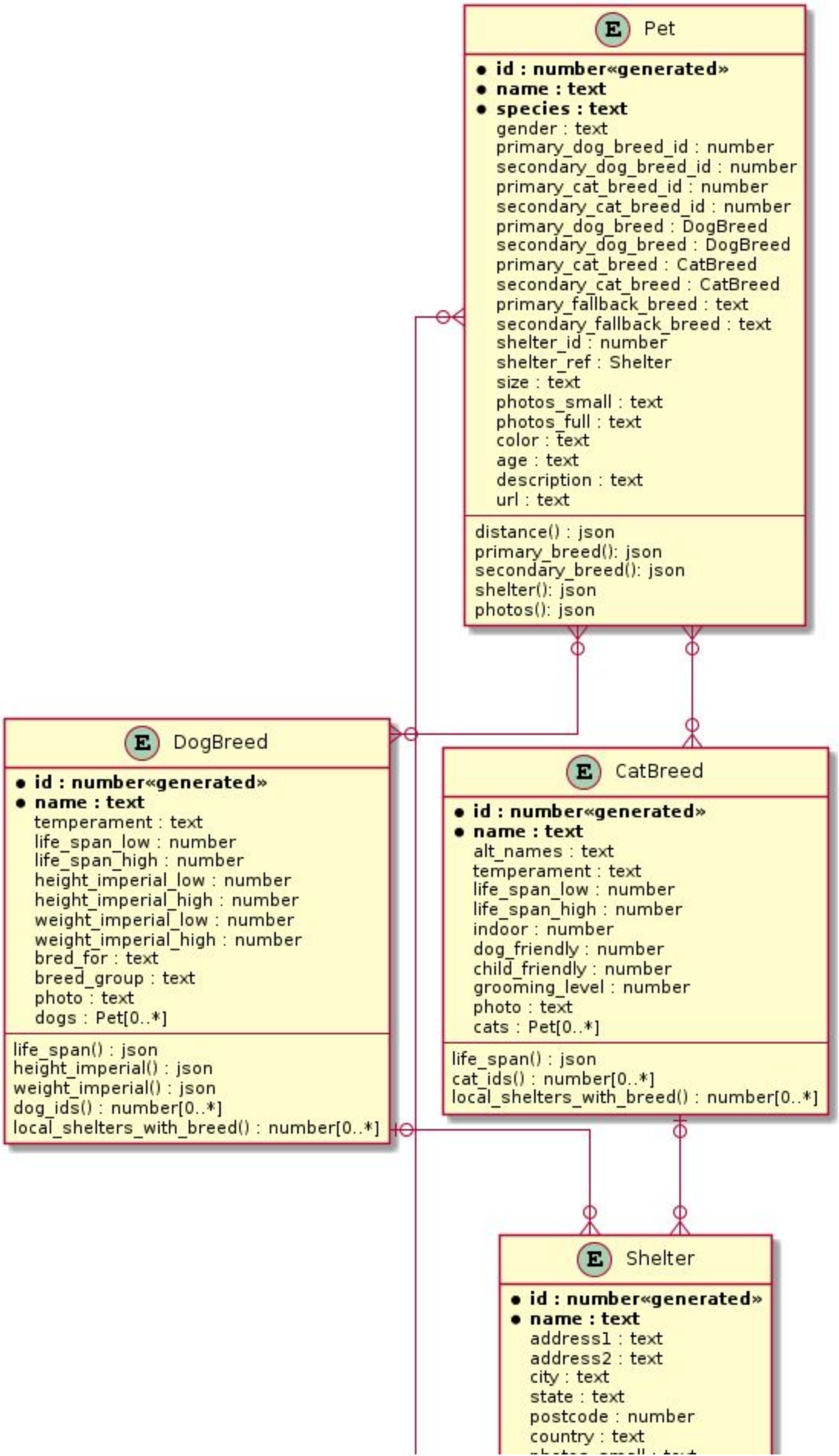
We decided to host our website through GCP. We had the option of hosting through the Compute Engine and the App Engine of which we chose the latter. Although it was more difficult to set up (in our opinion, as with compute engine all that was needed was to start the server process in the VM), App Engine is free as long as your app stays within quotas set by Google. Also, App Engine provides more scalability features and it is easier to re-deploy once the project is set up. We got our domain name through NameCheap. HTTPS certificates are managed automatically through GCP App Engine by a feature known as "Google-managed SSL." This is advantageous because we no longer needed to know how to set up HTTPS or renew certificates manually. We found that the frontend instance needs the API key for our GitLab repository to aggregate statistics on the about page.

Section 7: Pagination

On our website, we implemented pagination as a query parameter to the backend that returns a certain quantity of data after a certain offset. This way, users only see 12 cards at a time, but they can load more if they want to. We not only let users go forward and backward one page at a time, but we show them the total number of pages, and let them go +/- 2 pages (boundaries permitting) at a time, or to the first and last.

Section 8: Database

Our database is hosted in a Google Cloud Platform PostgreSQL instance. We constructed the definitions in python using SQLAlchemy. There are four tables: Pet, DogBreed, CatBreed, and Shelter. Each contains the data, methods, and connections outlined in the UML diagram below. The connections for Pet are a primary and secondary dog breed, as well as a primary and secondary cat breed, and the shelter it belongs to. The connections for Dog Breed are a list of pets that belong to that breed group, and a list of local shelters with that breed. The connections for Cat Breed are a list of pets that belong to that breed group, and a list of local shelters with that breed. The connections for Shelter are a list of pets that it contains, and the dog breed and cat breed that occur the most at that shelter.


Pet
• id : number«generated»
• name : text
• species : text
gender : text
primary_dog_breed_id : number
secondary_dog_breed_id : number
primary_cat_breed_id : number
secondary_cat_breed_id : number
primary_dog_breed : DogBreed
secondary_dog_breed : DogBreed
primary_cat_breed : CatBreed
secondary_cat_breed : CatBreed
primary_fallback_breed : text
secondary_fallback_breed : text
shelter_id : number
shelter_ref : Shelter
size : text
photos_small : text
photos_full : text
color : text
age : text
description : text
url : text
distance() : json
primary_breed(): json
secondary_breed(): json
shelter(): json
photos(): json
DogBreed
• id : number«generated»
• name : text
temperament : text
life_span_low : number
life_span_high : number
height_imperial_low : number
height_imperial_high : number
weight_imperial_low : number
weight_imperial_high : number
bred_for : text
breed_group : text
photo : text
dogs : Pet[0..*]
life_span() : json
height_imperial() : json
weight_imperial() : json
dog_ids() : number[0..*]
local_shelters_with_breed() : number[0..*]
CatBreed
• id : number«generated»
• name : text
alt_names : text
temperament : text
life_span_low : number
life_span_high : number
indoor : number
dog_friendly : number
child_friendly : number
grooming_level : number
photo : text
cats : Pet[0..*]
life_span() : json
cat_ids() : number[0..*]
local_shelters_with_breed() : number[0..*]
Shelter
• id : number«generated»
• name : text
address1 : text
address2 : text
city : text
state : text
postcode : number
country : text

Section 9: Testing

9.1: Selenium GUI Testing

We used Selenium to test various components of our GUI, such as ensuring that links (model pages, “adopt a pet today” button) take users to the appropriate location.

9.2: Mocha Frontend Testing

We used Mocha to test the logic on our frontend. Our tests range from basic rendering-checks and logic validation to more complicated tests where we verify that certain parts of the page load correctly when we receive bad or incomplete data from the APIs we call.

9.3: Postman API Testing

We used Postman’s testing suite to ensure that our API responds to normal HTTP calls and returns expected data.

9.4: Python unittest Backend Testing

We used Python’s built-in unittest functionality to test a myriad of things rooted in the backend, mostly API-centered. In addition to our Postman tests, we have some Python tests that ensure our internal API is performing as expected, as well as tests that confirm the liveness of our external API data sources (i.e. we’re not being rate-limited, the websites haven’t died, etc.)