

CS373: Software Engineering

10 am, Group 3 - Pets4me Phase IV

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:

We did not receive any user stories from our Customer group.

2.2 User Stories for CrashSafe:

1. Subtext in home page overlap

- a. To replicate: As a user I would like to see the subtext in the home page not overlap when resized. The font could be resized like it is when the page is smaller. It could also be fixed by increasing the size of the textbox.

2. Clear Comparisons Button for Test Provider Page

- a. As a User I would like to be able to clear the instance comparisons in 'Test Provider' model page. This would be useful so the user won't need to refresh to remove selected instances when the 4 instances limit is hit. I think an implementation similar to that of the 'car model' and 'car test' comparison would be good.

3. Clear Comparisons Button for Car Brand Page

- a. As a User I would like to be able to clear the instance comparisons in 'Car Brands' model page. This would be useful so the user won't need to refresh to remove selected instances when the 4 instances limit is hit. I think an implementation similar to that of the 'car model' and 'car test' comparison would be good.

4. Favicon Update

- a. As a user, I'd like to have a better grasp of CrashSafe's branding. When I look at my tabs, I see that CrashSafe's favicon is still the generic React logo. An updated favicon would help me be able to identify and navigate to CrashSafe.

5. Make Car Model pages visible

- a. As a user, I'd like to peruse the Car Models. However, when going to <http://crashsafe.me/carmodel>, I get the error `TypeError: rowValue is undefined`. Please resolve this error so I can view that model.

Section 3: RESTful API

GET /filter

This endpoint will return all data needed to populate filter menus.

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. We also include images of each breed and a youtube video on 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 and to count the number of tests each of us created. 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. However, we found that it was cheaper to host our database on Compute Engine as the Cloud SQL service is not included with the free tier. 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.

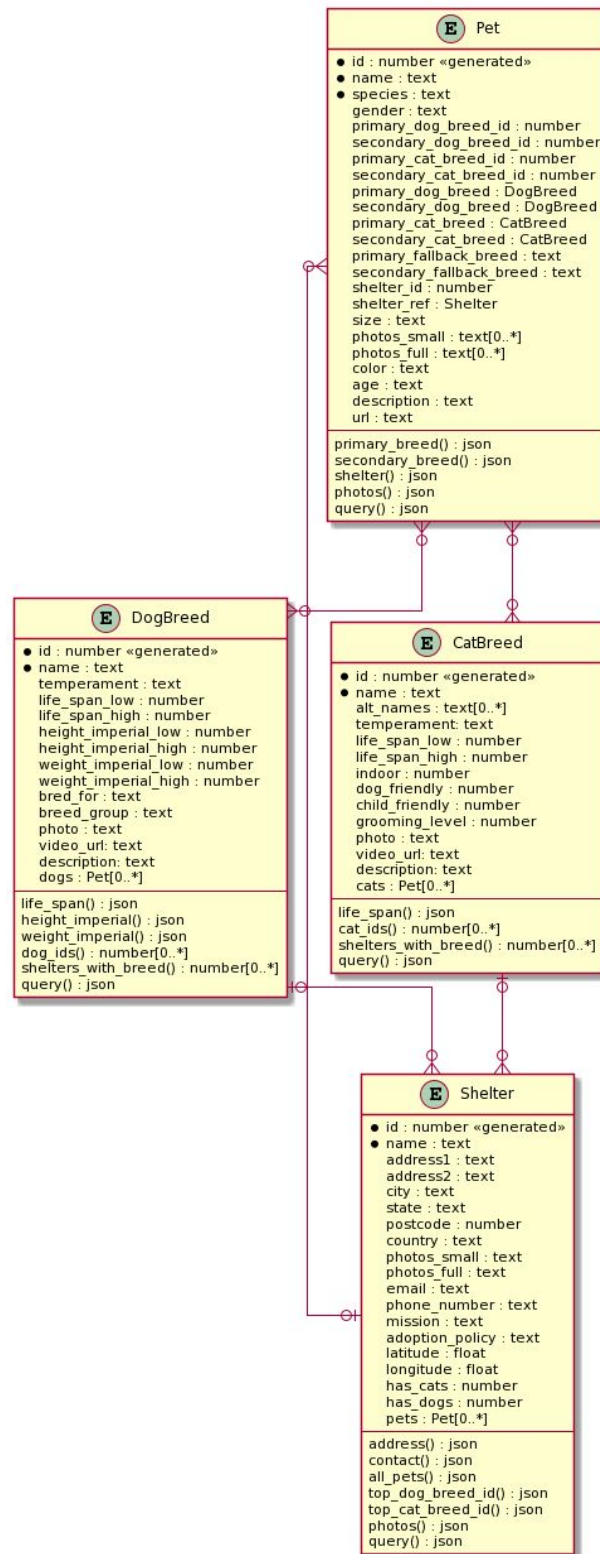
Section 7: Pagination

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 Compute Engine instance with a Postgresql database installed manually. 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.



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.)

Section 10: Filtering/Sorting

Filtering: Filtering can be done across multiple types of quantitative data in our models. However, some of the quantitative data appears to be qualitative because the APIs present it in a qualitative fashion - for example, a pet’s “age” is not presented numerically, but as either “Baby”, “Puppy”, “Adult”, or “Senior”. Since we have a small number of finite values for these fields, it’s still fairly easy for users to filter by those values. Otherwise, we provide dropdown, multi-select menus for string values (breed, species, etc.) and sliders for numerical values (height, lifespan, distance, etc.). We pass this information to the backend to filter using machinery built in to Flask Restless. To accomplish this, the frontend call has to include the name of the table we’re filtering on, the value to filter by, and an operator that details how to use the value for filtering. The backend also has a few custom query parameters implemented alongside flask-restless for sorting and filtering, as it didn’t give us enough functionality by default for all of the features that we wanted to implement.

Note: Not many zip codes have shelters. We suggest starting at 78702 to see that filter work.

Sorting: Similar to filtering, we allow the user to sort by a variety of fields on each model in either ascending or descending order. We achieve this by passing a value with the filter information that describes which value we want to sort by, as well as the direction we want to sort in.

Section 12: Searching

We allow the users to search in two different ways from the same search bar - they can either search the entire website from one base search (The “Pets4Me” option), or they have the option to search based on individual models. We display the information for them in a way that matches how we display it on model homepages (with cards) and with the query keywords highlighted for them. For the global search, we split the results up based on model type. To search, simply enter your query and **press enter** on your keyboard. Search is handled on the backend side through a query parameter that you can use to provide a search string to each endpoint. The backend then constructs an appropriate database query based on the search string content, and returns the result to the frontend.

Section 13: Visualizations

We provide several visualizations of our website's data on the about page. The first carousel of visualizations is for our website, and the second carousel is for our developer, CrashSafe. In our visualizations you can see a couple of interesting things. First, we have many instances of the top few breeds, and then a large number of breeds with very few instances. This suggests that some types of dogs are more much more common in shelters than others. This could be because the breeds are unpopular and difficult to adopt out, have a high abandonment rate, or because they make up much of the stray population in Texas. As for cats, we have an even more stratified distribution, with one breed having almost all the instances while every other breed trails behind at less than 10% of its population. The number of pets per shelter visualization can be a little misleading, as it suggests that many shelters only have one pet, but really it shows a limitation of our data collection method because we only pull a finite number of pets from PetFinder™. To collect shelters, we just store each new shelter that we find when getting a pet from PetFinder™; we make no attempt to find all pets for a given shelter.

In the CrashSafe visualizations we can see the overall safety ratings of around half the brands in their database. Most of them have at least a 4.0 rating, with a very gradual decline from 5.0 to 4.0, however the few that go below that drop in rating much more rapidly. The second visualization shows how average safety ratings have improved over time, showing encouraging

growth since 2002. In the final graph we see an overall positive correlation between user rating and overall safety rating, which might mean that improving vehicle safety has a positive effect on customer opinion.

Section 14: Presentation

<https://www.youtube.com/watch?v=oE28pBfLF-0>