

**University of Toronto**  
**Department of Computer Science**  
**CSC309: Programming on the Web Winter**  
**2016**

Project	Pet Care
Team Members	Rahal Ranasinghe : c6ranasi David La : g4davidl Taewoo Kim : g5kimtae Tzu-Chiang Weng : c5wengtz
Instructor	Mashiyat, Ahmed Shah
TA	Kyriakos Georgiou
Due Date	Wednesday, April 6, 2016

# 1. Introduction

In this assignment, we designed and implemented a web application for a sharing economy called PetCare. PetCare is an online service which connects individuals who are looking for pet sitters to take care of their pets and pet sitters in their community who are experienced in taking care of pets. For the implementation of this web application we used Node.js as the web server, Express.js as the back-end web framework, Angular.js as the front-end framework and MongoDB as the database. We used SASS to write our CSS code and our design is completely responsive. We have also deployed the web application on Heroku cloud platform. In this document we describe the detailed design of our application, security measurements we took, performance improvements, web application architecture, REST API design and testing done for the application.

## 2. Detailed Design


### 2.1 High-level View

PetCare is an online service where individuals who are looking for someone to take care of their pets can connect with pet sitters in their community who are experienced in taking care of pets. Users can register on the system and look for a pet sitter or offer their pet sitting services, or both.

There are two types of postings that users can create:

1. Pet postings ([http://localhost:3000/new\\_pet\\_posts](http://localhost:3000/new_pet_posts))  
Pet postings are for users who are looking for a pet sitter to take care of their pet. They must first register a pet ([http://localhost:3000/new\\_pet](http://localhost:3000/new_pet)) and specify information such as the price, date, and location.
2. Pet sitter postings ([http://localhost:3000/new\\_petsitter\\_posts](http://localhost:3000/new_petsitter_posts))  
Pet sitter postings are for users who want to provide their pet sitting services to other users. They must provide information such as the types of pets they are willing to take care of, their price range, and experience.

Users who create postings will have the option to upload their own images to be used as the posting thumbnail, which will be uploaded to an Amazon S3 server.



### Looking for a kind pet sitter.

★★★★★ 2 Reviews

📍 Toronto, ON

🐾 Dog

\$ 50 per day

📅 May 15 2016 to May 30 2016

🕒 2 years old

💬 Apply

🗉 Review

**Gender** Male

**Breed** Labrador Retriever

**Pet Supplies** Toys, Kennel, Clothes

**Medicine or Special Diet** N/A

Looking for someone to take care of my dog while I am out of the country.


Sample posting page

On the posting page, users who are logged in can apply or review the posting.

## 2.2 Description of different sections

### 2.2.1 User / Profiles / Authentication

Our user profile component contains all of the information for the user, including user's pet, active posts, closed posts, and reviews. Users are able to update their public information and their profile pictures when they log in. Under the pet section, users can edit their pet's information and see the reviews for each pet. They can also message other users or file a report when they think the content of a certain user is inappropriate.



**Leonardo Dicaprio**  
☆☆☆☆☆ 0 Reviews  
Location New York, USA  
Email leonardo.dicaprio@gmail.com

Edit


I love my pets.

Pets

Postings

History

Review




**iPhone5**  
☆☆☆☆☆ 0 Reviews  
Cat  
3 years old

Gender Neutered Male  
Breed Manx

Edit

He loves tuna.



**Earth**  
☆☆☆☆☆ 0 Reviews  
Cat  
4 years old

Gender Intact Male  
Breed Manx

Edit

### User Profile with public information

No file chosen



Leonardo Dicaprio  
☆☆☆☆☆ 0 Reviews  
New York, USA  
leonardo.dicaprio@gmail.com

Save

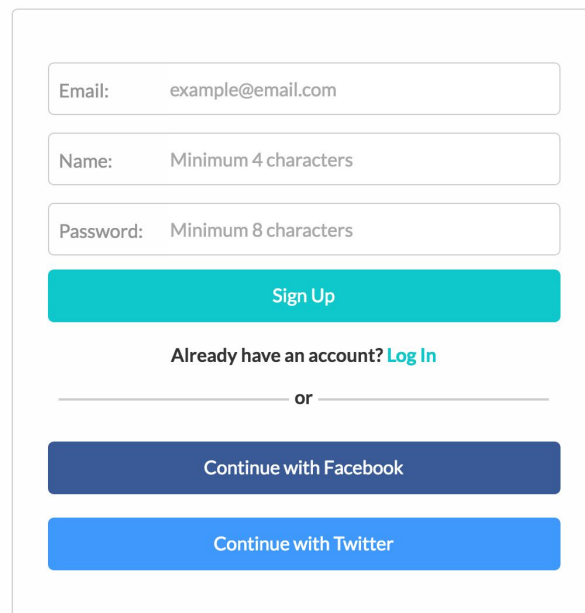
Cancel

I love my pets.

### User Profile Edit Mode

We provide three ways to authenticate user. The first is by username and password. Users have to set their username and password when they register, and will have to use these information to log in. The second ways is through facebook authentication. User can connect their accounts to their facebook account by granting permission to access their facebook id and access token, which will be used to verify their identities in the future. The third way is through twitter

authentication, which works in the same way as facebook authentication. We use third-party libraries 'passport', 'passport-facebook', and 'passport-twitter' to implement our authentication strategies. When the user register, their default role is 'regular', which means they can only access and edit to their private account information. In the admin console, the user whose role is admin will be able to promote the regular user to admin. To access certain restricted pages, such as reporting and messaging other users, or making posts, the users must login to continue; otherwise, they will be redirected to the login page. If the user tries to access other users' private information, such as message and application inbox, they will be redirected to their own message and application inbox.



Sign up form with three ways of authentication

### 2.2.2 Rates / Comments

Our Reviews (Rating/Commenting) component is integrated with other pages like /petsitter\_posts/:id, /pet\_posts/:id, /users/:id.

A user can review another user for their service. When giving a review to a user, the review form displays a rating scheme where users can give 'stars' to other users (min 0 stars and max 5 stars). Also they can add a comment as a part of the review. Once the reviews is submitted, we calculate the average rating for the user who was reviewed and display the new average user rating value on his/her profile along with a link to all the reviews that user received.


Review

Rating:  
★ ★ ★ ★ ☆

Review:  
Im very satisfied with the service. I totally recommend her

Close Submit

Review form (Rate and Comment on a user)



Jennifer Lawrence


★ ★ ★ ★ ☆ 2 Reviews

Location Toronto, ON

Email jennifer@gmail.com


Display average rating and link to all reviews

Pets Postings History **Review**



Christian Bale  
Toronto, ON

★ ★ ★ ★ ☆ 2016-04-05  
Im very satisfied with the service. I totally recommend her



Christian Bale  
Toronto, ON

★ ★ ★ ☆ ☆ 2016-04-05  
Not the best pet sitter out there. But she only charged \$10/hr

Read all reviews for the user

### 2.2.3 Recommendation System / Search

Our search system searches for postings that users submit to the server. There are two types of postings, “Hire a Pet Sitter” and “Offer Pet Sitting”. “Hire a Pet Sitter” has postings that offer pet sitters and “Offer Pet Sitting” has postings that look for pet sitters for their pets.



Search pages on the nav bar - “Hire a Pet Sitter” and “Offer Pet Sitting”

A dark gray search bar with three input fields: 'Pet' with placeholder text 'Enter pet type ...', 'City' with placeholder text 'Enter city ...', and 'Price' with placeholder text 'Enter maximum price ...'. To the right of these fields is a teal 'Search' button.

Search bar

The search results are based on user’s queries (pet type, city, price) and the user’s data if the user is logged in. In the initial search pages, instead of having empty results, the search system recommends postings based on user’s data (location, type of user’s pet) if the user is logged in. If the user is not logged in, it gets the location from The Google Maps Geocoding API (See Additional Information section). It may take some time until the page loads result postings.

### Recommendations near your location.

Make a new post

Previous 1 Next



#### Looking for a kind pet sitter.

★★★★★

📍 Toronto, ON

🐾 Dog

\$ 50 per day

📅 May 15 2016 to May 30 2016

🕒 2 years old

Apply

Looking for someone to take care of my dog while I am out of the country.... [See More](#)

Initial recommendation in “Offer Pet Sitting” when user is logged in. Search based on user’s data.

## Recommendations near Toronto.

Make a new post

Previous

1

2

Next



### Pet Sitter Near Toronto. I offer pet sitting in the GTA area

★★★★☆

📍 Toronto

🐾 Dog, Cat, Bird, Rabbit, Fish

💰 \$ 20 - 50 per day

📅 May 01 2016 to May 30 2016

🕒 5 years experience

Apply

I have raised many types of pets. This May, my friends and I are going to offer pet sitters. We plan to offer this in any place in Canada.... [See More](#)

Initial recommendation in “Hire a Pet Sitter” when user is not logged in. Toronto is acquired from the Google api.

Search algorithm with 3 input queries (pet type, city, price):

- Postings that are matched with more input queries get higher priority in recommendation.
- If the input queries are not given, do not filter out the postings but rather, less priority is given to the postings than the postings that are matched with the given queries.
- Excludes the postings that the user created.
- Notify to the users whether they already applied to postings.
- The price query in “Hire a Pet Sitter” is the maximum price that the user would like to pay to a pet sitter. The price query in “Offer Pet Sitting” is the minimum price that the user would like to get paid by offering pet sitter.
- The 3 input queries are case-insensitive.

By clicking the title of the postings or “See More” link, it goes to the detailed posting page. Under the target posting, there are recommendation postings based on the search that the user previously made.



Previous12Next



Somone who likes a dog

★★★★☆

📍

Ottawa

🐾

Dog

📅

May 01 2016 to July 30 2016

🕒

1 years old

Apply

I have one dog in Ottawa and I am going to another city for this summer. Unfortunately, I can't go with her. Looking for someone who likes dog and would like to spend this summer with her.... [See More](#)

The first search result in “Offer Pet Sitting” with queries (dog, Ottawa, 10)

Previous1Next



Looking for a kind cat sitter in Vancouver.

★★★★☆

📍

Vancouver

🐾

Cat

📅

May 27 2016 to May 30 2016

🕒

4 years old

Apply

I will travel Vancouver with my cat but I want to travel alone for just a few days.... [See More](#)

The first search result in “Offer Pet Sitting” with queries (cat, Vancouver, 50)

Previous12Next



Sitter Near Montreal

★★★★☆

📍

Montreal

🐾

Dog, Cat, Bird, Rabbit, Fish

📅

May 01 2016 to May 30 2016

🕒

5 years experience

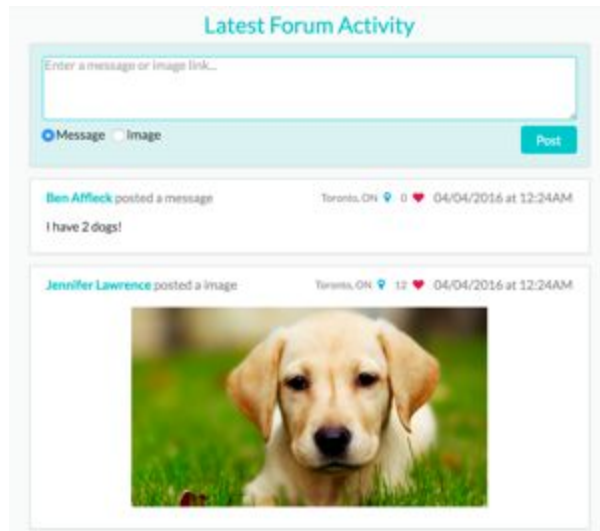
Apply

I have raised many types of pets. This May, my friends and I are going to offer pet sitters. We plan to offer this in any place in Canada.... [See More](#)

The first search result in “Hire a Pet Sitter” with queries (cat, montreal, 40)

## 2.2.4 Social Network

Our social network component of the site is a public forum where users can post messages or pictures. The posts contain information about the user's location, the posting date, and any images the user has added. It is similar to other social networks such as Twitter and Facebook in that other users can 'like' a post and the newest posts are shown at the top.



Forum activity page

## 2.2.5 Admin

<http://localhost:3000/admin>

Note: user must be logged in as an admin to access this link

Users who are admins can access the admin console by click on their username in the upper left corner and clicking on 'Admin Console'.



Admin users can access the Admin Console from their dropdown menu

At the top of the console, admins can view statistics of the site, including the number of users, postings, and report.

Admin Dashboard			
	73 users		279 posts
	25 reports		

### Overall site statistics

Admins have the capability of editing any user's info, banning the user, and making a user an admin.

Users					
Name	Profile	Message	Edit Info	Ban User	Admin
Ben Affleck	<a href="#">View</a>	<a href="#">Message</a>	<a href="#">Edit</a>	<a href="#">Ban</a>	<a href="#">Remove admin</a>
Jennifer Lawrence	<a href="#">View</a>	<a href="#">Message</a>	<a href="#">Edit</a>	<a href="#">Ban</a>	<a href="#">Make admin</a>
Christian Bale	<a href="#">View</a>	<a href="#">Message</a>	<a href="#">Edit</a>	<a href="#">Banned</a>	<a href="#">Make admin</a>

### Admin functions on users.

Admins can also edit any posting information, as well as delete the posting.

Postings			
Date	View	Edit Posting	Delete
Apr 3, 2016 6:59:19 PM	<a href="#">Pet Post 1</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
Apr 3, 2016 6:59:19 PM	<a href="#">Pet Post 2</a>	<a href="#">Edit</a>	<a href="#">Delete</a>

### Admin functions on postings.

Admins can see any reports that have been made and perform and necessary actions such as banning the user, and responding to the user.

Reports				
Date	Reported by	Reporting	Read Report	Message
Apr 3, 2016 6:59:17 PM	<a href="#">Christian Bale</a>	<a href="#">Jennifer Lawrence</a>	<a href="#">Read</a>	<a href="#">Message</a>
Apr 3, 2016 6:59:17 PM	<a href="#">Jennifer Lawrence</a>	<a href="#">Christian Bale</a>	<a href="#">Read</a>	<a href="#">Message</a>

### Admin functions on reports.

## 3. Security

### 3.1 Password security by salting/hashing passwords using the ‘Passport’ module.

Storing the user passwords in the database as plain text format is not secure. If an attacker was able to gain access to the system, he/she can easily read all the username-password combinations. Hashing is a method to encrypt the password before storing it in the database. This way if an attacker gain access to the database, he cannot see the plain text passwords. However, it is still possible for attackers to perform dictionary attacks where he/she use a list of passwords to generate possible password hashes. To prevent this, ‘Passport’ uses a method called ‘salting’. Salting is a method to concatenate the user’s password with a random string called the salt before hashing. In the database we then save the salt and the hash value instead of the plain text password. This is much secure than since it makes dictionary attack against arbitrary users much harder.

Testing:

To test this, we checked the database entries for newly created users in our mongoDB database by performing the following steps.

- We first registered two users named ‘Ben’ and ‘Jennifer’ to our system with the same password, 12345678.
- Open mongo daemon by issuing the ‘mongo’ command
- Query the database entries for all the users by issuing ‘db.users.find({})’

Results:

```
{ "_id" : 1, "updated_at" : ISODate("2016-04-03T19:40:46.306Z"), "created_at" : ISODate("2016-04-03T19:40:46.306Z"), "salt" : "866d6d344886556f0ede9f847c68a21fae99c12cab6b864f81c5a870b39cb80e", "hash" : "672615c8a9472474a197140a34d44c0b42581ce8762b3736f3a1fd5318ed9c9f887abb25e4c82fb4b10542b801f19c81e6608b2352a9cdefb247ce19a4633d2a4b2353821c3786c785238b936e977cb5d651810df785c8718e517258aad13fddc13183ede9e91f8cf1f3a23ba84c51da2d9612ba6d6b64ca188cf3e4184f5d24a188a712a0cb63a1c89fd2296cb168f7645d90f967f9643421a70e6aa7f0b9220d8241b697fe06025a01bdf9d339d363ce63bf9d0ca2a846a7081dd369ac70d06fd44522f7042b2e8416c2e97957b459c19d56f86dfe2a510930e98baa72f9024cf7f1c8ef764b6db8cee736128b1b9a4c408a5e86e681ccf393d6976848ebc0736dc7ba39d8fa27801a243cafed2129e4827e495bd3dfb1b3dbcd4df60a556defaaab252dd3545445f69a86976cbf967854157bd4900a9d5371be35564caabf7225b3ad2bd0c172ad5c439729b0739aa400c4911d4fd93b1ef3eeab194c8aa0f97961b3fa4f5c8f57a7332831d99a0ce20458cbfdce103b4834e36b3188928f20d57c2f85c7e720a9036093d1d266c466a17d60efdda5dadaa445862fbfb1e002d9fe60b7e422a8174b8e5bda36211c0b48974677c515f88ae84d467edd20bd596cfa7d18988564a30d0e9a6540df82b930720848a53c68cd2761540a9d35172f44880694d8b5cc8b3a89ec90aaab56525b568698bdca252c2a01b4", "name" : "Ben Affleck", "username" : "Ben", "email" : "admin@gmail.com", "location" : "Toronto, ON", "role" : "admin", "banned" : false, "rating" : 0, "_v" : 0 }
```

Salt and hash values in database for user ‘Ben’

```
{ "_id" : 2, "updated_at" : ISODate("2016-04-03T19:40:46.842Z"), "created_at" : ISODate("2016-04-03T19:40:46.842Z"), "salt" : "4e2cadfadf52b4f1a8c80e7c5505ffdc34c9ad418f5351a8379979a4a82544a", "hash" : "98db742c34fa543ecf61b06879b313aca0f5432a9c4c2ffa0d2d619d30615d723573ac17d95aaeff22adddba77f4fb41b6e1c9bc55ec5b5b269e4b9438883203008f0886d408ec6ab560984d7860c763cb4e95bb5849849ee5e80586e3cc0bb0e2ad317154622c2a81b830c05a5bdc825a92bbd7667e548c3618bc788ca8dde93b798f7c4a6d2f89ed0576616d45ac4fd732b0b074f1b64c201ef9ed8cba2dbac3f9970a379e68bb322aba9812f5c104336ea93d0ba026ed3b10ad13020f950fa594b4c0aad53d6d4201ffc36efebc83aa223c057450267b81bcea7f9b22866e59c5232c0251ac511a6f563bc9764075f9aad18914ee4f49b3c8a4c5ab931d56e9f91c8edd8a753f775713b455cbff2026f3e7d4eb012c20e30bf453cb8b0e86ae6c3bfcd1d0ee8fe44792f305cde59dc40da5ac95ae9757c4804c46f0d12889f90894239d6a37d3b616b1c3b106d2fd8cdad94303c4eedcf9fb1b1c34f6e6ff77cc4e98b893cd9366e48b1438d089d9d10e4f2073a0606c04d8ae1bdb5a82a39568bcbef6b2bb8c5faaf6a365cafdf627183b4837681e5c104931e88a60685e65ed1386adb780413e0d180c645aa2990c7f7b00b2253b874a6ac780d757e5b6e27600f206a5c7e60b3fc88f511dfcc38d2b17c8757d09c2213a9590826e4db937bdc27a6542a58e3c3c99d22bd37d5f46cbf8befac93748338d82be44a1645", "name" : "Jennifer Lawrence", "username" : "Jennifer", "email" : "Jennifer@gmail.com", "location" : "Toronto, ON", "role" : "regular", "banned" : false, "rating" : 4, "_v" : 0 }
```

Salt and hash values in database for user ‘Jennifer’

We were able to verify that for the same password, 12345678, the hash value that was stored in the database was different. Also, If an attacker gain access to this information, he/she will not be able to see the plain text passwords of the users.

### 3.2 Cross-site scripting attack (XSS) prevention by using Helmet.js module

Cross-Site Scripting (XSS) attacks are a method of client side script injection, where attackers inject malicious script to trusted web sites. Helmet module protect Express applications by setting various HTTP headers and disabling various HTTP headers. Helmet module mitigate XSS attacks by setting the X-XSS-Protection HTTP header of the HTTP response. In addition to this it disables the X-Powered-By header of the response. X-Powered-By header is enabled by default and attackers can use this header to detect apps running Express and then launch specifically-targeted attacks.

Testing:

We logged the response headers of our server responses to a simple GET request before and after implementing the Helmet XSS security.

```
app.get("/layouts/home.html",function(req,res){
  console.log(res._headers);
  res.render("layouts/home.html");
});
```

Code segment for logging the response header

Results:

```
{ 'x-powered-by': 'Express' }
```

Headers Removed

```
{ 'x-content-type-options': 'nosniff',
  'x-frame-options': 'SAMEORIGIN',
  'x-download-options': 'noopen',
  'x-xss-protection': '1; mode=block' }
```

Headers Added

We're able to verify that x-powered-by header was removed and x-xss-protection header along with other headers (used for security) were added to the response header. Hence securing our application from XSS attacks and also at the same time hiding information useful for attackers.

### 3.3 Cookies and request forgery

When a user logs in, the server sends a unique user-id (that is generated by database when a user creates an account) which will be used for searching user's data in the database. Then, the client saves the user-id in a browser's cookies. Some rest apis send requests to server with cookies values. Some of them request for users' private data. However, it is easy to change cookies from browser's console. A user can change his/her user-id in cookies to other user-ids (the user can easily guess other user-ids since user-id is natural number). The server does not know whether the user-id is changed by the user or not. As a result, the user can see other users' private information and also change it. In our website, the malicious user can see other users' messages and applications. The user can also send applications, messages, and reviews as if they are sent from other users. For example, the user can send a message to other users with administrator user-id and ask for personal information.

To prevent this, when a user logs in, server creates a temporary random number and saves (maps) it with user-id in the database. The number is used as authentication token. The server sends token with user-id back to client. When client sends requests that involve user-id, it sends the token as well. Then, the server authenticates the request by checking user-id and the token are matched with the mapping in the database.

In this way, the server can know whether users use their own user-id or not. Even if a user knows exact user-id of other users, they cannot use it without corresponding token. It is hard to guess the token value, since it is randomly generated every time when users login. Moreover, the token is only valid when the corresponding user is logged in. In other words, if a user is not logged in, no one can access to the user's data.

Testing:

Change user-id cookies from browser console and send http request to get user 1's messages.

```
> document.cookie="userID=1"
< "userID=1"
```

Result:

```
▶ GET http://localhost:3000/api/messages/1/0.05654555605724454 angular.min.js:100
401 (Unauthorized)
```

Unauthorized, since the user-id does not match with the token.



## 4. Performance

### 4.1 Performance Improvement Techniques

#### Cache-control

We set a cache-control header when we serve our static files, which tells the browser to cache the file for as long as we specify. This reduces additional HTTP requests by using the cached files. We set the max age to 86400000 milliseconds. This means the browser will cache these static files for 86400000 milliseconds.

#### GZIP Compression

We used a Node.js compression middleware module called 'compression' which use a gzip to compress the response bodies for all request that traverse through the middleware. By compressing the size of the response body, the application will use less network bandwidth. This will reduce the response time because now the total size of the response is smaller.

#### Minified files

We used minified versions of the JavaScript and CSS files which reduces the file size by removing unnecessary parts of the file such as spacing and line breaks. This reduces the download and execution time.

#### Stylesheets at the top

We put the CSS files at the top in the <head> tags so that they load first and the page will be rendered with the styles loaded.

#### Network (Split Resources Across Servers)

We upload our images to Amazon S3, which has been designed for reliability and speed. If this site were to be hosted on a server, hosting the images separately on Amazon S3 will reduce the download time for the images. Uploaded images also have a restricted size to prevent large image downloads. If our images are coming from different domains, this can also allow the browser to download the assets in parallel. Other than default profile pictures, our website hosts very few images to avoid any additional HTTP requests.

#### CDN for libraries



We load our JavaScript and CSS libraries from CDNs provided by Google Hosted Libraries and BootstrapCDN. This will reduce the loading time of the website because if a user has previously visited a site that has also loaded the same resource from the same CDN and it is still cached, they will not have to download the resource again when they visit our website.

## **4.2 LOCUST performance testing**

We also used an open source load testing tool called LOCUST to calculate performance of our web applications and for load testing. We performed these tests when the application is running on the Heroku cloud platform. To run these tests we first installed locust and wrote python code that simulate the user behaviour. Then we tested how our system behave when 50 simultaneous users are accessing the system. This python script and how to run the LOCUST tests are written under PetCare/test/my\_locust\_file.py file.

First we simulated 50 users making 6392 requests without GZIP and Caching implemented on our application. Then we implemented GZIP Compression and Cache-control as we described above. After running the tests again, with 50 simulated users making 7115 requests, we saw a x1.7 increase in the average response time of our application (242ms/142ms). See the below LOCUST output for more test results.

In addition to these performance increases, we also found out that the failure rate for 50 users accessing our application simultaneously is 0%. A summary of locust results are shown below.



Statistics Failures Exceptions Download Data

Type	Name	# requests	# fails	Median	Average	Min	Max
GET	/	915	0	150	237	37	2545
GET	/api/users/2/reviews	947	0	140	253	56	3170
GET	/forum	887	0	150	250	37	3256
GET	/pet_posts	863	0	150	247	41	2835
GET	/petsitter_posts	910	0	140	245	37	3469
GET	/signin	956	0	150	244	39	5462
GET	/users/1	914	0	140	220	40	4177
Total		6392	0	150	242	37	5462

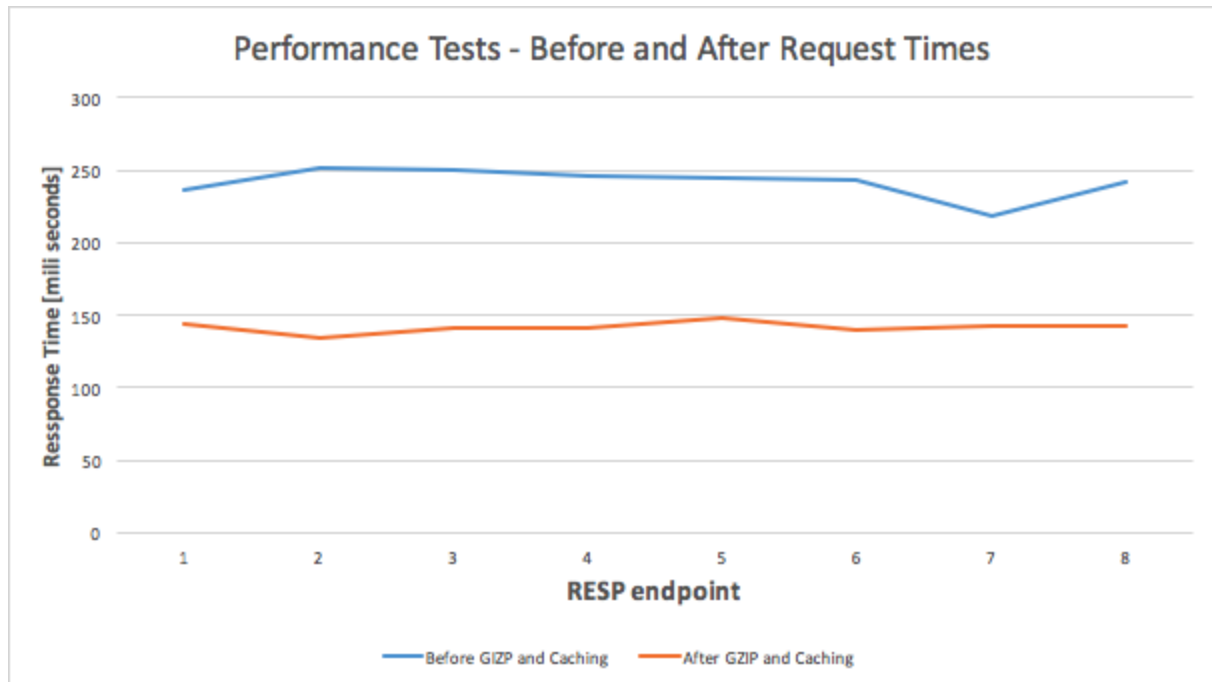
50 simulated users making 6392 requests **before** GZIP and Caching



Statistics Failures Exceptions Download Data

Type	Name	# requests	# fails	Median	Average	Min	Max
GET	/	1008	0	120	144	51	1643
GET	/api/users/2/reviews	1008	0	120	136	60	656
GET	/forum	1036	0	120	141	59	659
GET	/pet_posts	1033	0	120	142	49	1078
GET	/petsitter_posts	1011	0	120	148	54	2349
GET	/signin	1033	0	120	140	60	2767
GET	/users/1	986	0	120	143	58	780
Total		7115	0	120	142	49	2767

50 simulated users making 7115 requests **after** GZIP and Caching



Average response times before and after GZIP and Caching

### 4.3 jMeter Testing

jMeter is a load testing tool that we use to test the performance of our application hosted on the Heroku cloud platform before and after optimization.

We created a test plan to test 50 users and a loop count of 5 to repeat the test 5 times. We simulated users visiting 6 pages that an average user would request when viewing on our site, including a REST API call.

Sample Test

Thread Group

Visit PetCare Homepage  
Visit PetCare Forum  
Visit PetCare Posting Search Page  
Visit PetCare Posting Page  
Visit PetCare User Profile  
Visit PetCare Signup Page  
Graph Results  
View Results in Table  
View Results Tree

WorkBench

Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

☒ Continue
☐ Start Next T

Thread Properties

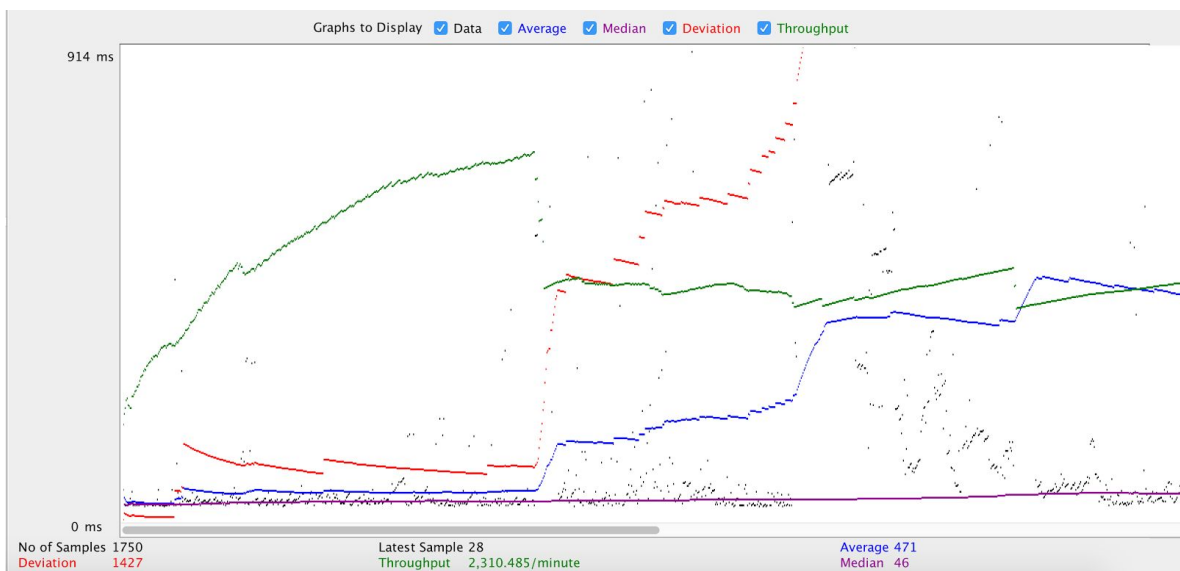
Number of Threads (users): 50

Ramp-Up Period (in seconds): 20

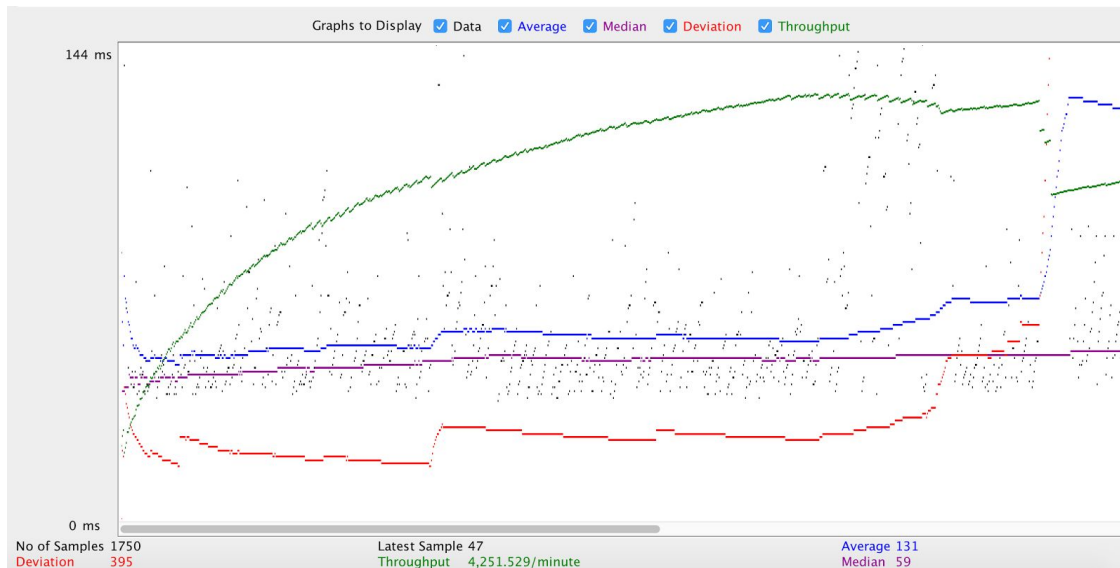
Loop Count: ☐ Forever 5

☐ Delay Thread creation until needed  
☐ Scheduler

jMeter Test Settings



Average response time of 471 ms before optimization



Average response time of 131 ms after optimization

Our jMeter test results showed an increase from an average response time of 471ms to 131ms. This is consistent from the results we got from testing with LOCUST.

## 5. Architecture

In our development process, we follow the MVC (Model-View-Controller) design pattern to build our application. Our application was built using Node.js, Express, AngularJS, and MongoDB. A directory diagram of our application is shown below. We put angular files and static files in the 'client' directory, and the node component files in the 'server' directory in order to separate front-end and back-end development. Based on our understanding of MVC structure, in the model subdirectory, there are database schema model files which define data structures and create a model which we can use to interact with MongoDB. In the view subdirectory, there are based html files which users see on their screens. In the route subdirectory, there are route files which handle requests from user, interact with model to get desired data and pass it back to the view.

We use AngularJS for our front-end framework because of its two-way data binding feature, and it allows us to build custom web components. Angular uses client side MVC pattern. In every of our controller, the \$scope object was injected to be used as the model. The model contains the data to be displayed in the view, and the functions that are invoked based on user input or events. Since AngularJS has a two-way data binding feature, elements in the view can bound to the model(\$scope) directly. Any change made in the view may also affect the model, or vice versa. The controller is responsible for pulling the model used by the view and setting up the dependencies or handle event from the view. In the controller, we get the desired data from the

database by making ajax calls. We use the directive ‘ng-view’ to include the rendered template of the current routes into the main layout (index.html) file. When the route changes, the view and controllers changes with it according to the configuration in the config.js file.

```
<!-- file structure -->

- server                <!-- Include all files for node components-->
----- config          <!-- Configuration files -->
----- models          <!-- Database Schema Models -->
----- routes          <!-- All routes are handled here-->
----- views           <!-- Based view -->

- public               <!-- Include all files for angular components and static files-->
----- app.js          <!-- Angular app -->
----- config          <!-- Angular configuration -->
----- assets          <!-- Images and stylesheets -->
----- controllers     <!-- Angular controllers -->
----- partials        <!-- Angular views -->
----- services        <!-- Angular services -->

- test                 <!-- Include all test files-->
----- my_locust_file.py <!-- LOCUST performance testing-->
----- test.js         <!-- Mocha tests-->

- data                 <!-- Include all data files-->
----- default-data.js <!-- Default data-->

- package.json         <!-- npm configuration to install dependencies -->
- server.js            <!-- Node configuration -->
- README.md
```

PetCare directory tree

## 6. REST API Documentation

Method	URL	Parameters	Description
POST	/auth/register	User's name, username and password are posted in the http POST request body	Register the user with username and password.
POST	/auth/login	User's username and password are posted in the http POST request body	Verify user's username and password
GET	/auth/logout	-	Logout the user

GET	/auth/status	-	Return if any user is currently logged in
GET	/auth/twitter	-	Login or register the user with twitter account
GET	/auth/twitter/callback	-	Callback Url for twitter authentication
GET	/auth/facebook	-	Login or register the user with facebook account
GET	/auth/facebook/callback	-	Callback Url for facebook authentication
GET	/api/users/:id	id: The numerical ID of the desired user	Returns the user information for user specified by id.
GET	/api/users	-	Returns a collection of all user information.
PUT	/api/users/:id/ban	Id: The numerical ID of the desired user	Updates the user's 'banned' field to 'true' Returns the Status code 200 and Modified:1 as a json object
PUT	/api/users/:id/role	Id: The numerical ID of the desired user	Updates the user's role to 'admin' or 'regular' depending the their previous role
GET	/api/users/:id/pets	id: The numerical ID of the desired user	Returns a collection of pets belonging to the user specified by id.
GET	/api/users/:id/posts/:status	id: The numerical ID of the desired user status: The status (open, closed) of the post	Returns the a collections of posts with specified status belonging to the user specified by id.
GET	/api/users/:id/reviews	id: The numerical ID of the desired user	Returns a collection of reviews sent to the user specified by id.
PUT	/api/users/:id	id: The numerical ID of the desired user	Updates the information of the user specified by id.
POST	/api/sitterpostings/:id/reviews	id: The numerical ID of the pet sitter post. Review, Rating and User Id	Query the user ID of the user who created the post, Update the reviews for that user and calculate the new average rating for the

		of the user who is making the review in the request body.	user.
GET	/api/sitterpostings/:id/rating	id: The numerical ID of the pet sitter post.	Return the average rating of the user who made the post
POST	/api/petpostings/:id/reviews	id: The numerical ID of the pet post. Review, Rating and User Id of the user who is making the review in the request body.	Query the pet ID of the pet referred to by this post, Update the reviews for that pet and calculate the new average rating.
GET	/api/petpostings/:id/rating	id: The numerical ID of the pet post.	Return the average rating of the pet referred to by this post
GET	/api/pets/:id	id: The numerical ID of the desired pet	Returns the pet information for the pet specified by id.
POST	/api/pets	Name, user, type, breed, gender, description, rating, photo	Creates a new pet.
GET	/api/petpostings/:id	id: The numerical ID of the desired pet posting	Returns the information for the posting specified by id.
GET	/api/petpostings	-	Returns a collection of all pet posting information.
DELETE	/api/petpostings/:id	id: The numerical ID of the desired pet posting	Deletes a pet posting and returns a Status code 200 and ok:1 as a json object
POST	/api/petpostings	User, title, duration, price, supplies, description, thumbnail, pet	Creates a new pet posting.
PUT	/api/petpostings/:id	id: The numerical ID of the desired pet posting	Updates the information of the posting specified by id.
PUT	/api/petpostings/:id/:st	Id: the numerical ID of the	Updates the status of the posting to 'open'



	atus	desired pet posting	or 'closed'.
GET	/api/sitterpostings/:id	id: The numerical ID of the desired sitter posting	Returns the information for the posting specified by id.
GET	/api/sitterpostings	-	Creates a new sitter posting.
PUT	/api/sitterpostings/:id	id: The numerical ID of the desired sitter posting	Updates the information of the posting specified by id.
PUT	/api/sitterpostings/:id/:status	id: The numerical ID of the desired sitter posting	Updates the status of the posting to 'open' or 'closed'.
GET	/api/reports/:id	id: The numerical ID of the report	Query the database for the report identified by the given id. Return the report as a json object.
GET	/api/reports	-	Return all the reports in the database as a json array
POST	/api/reports/	Report message, user id who is creating the report and the user id of the user is is being reported in the request body	Creates a new report and add it to the database.
GET	/api/users/:userId:token/news	userId: a key in User db table, token: authentication value for userId	Get the number of unread (new) messages and applications
GET	/api/petpostings/:pet/:location/:min_price/:userId	pet: type of pet, location: city, min_price: minimum price of posting, userId: a key in User db table	Get search results of pet sitter finding postings.
GET	/api/sitterpostings/:pet/:location/:max_price/:userId	pet: type of pet, location: city, max_price :maximum price	Get search results of pet sitter offering postings.

		of posting, userId: a key in User db table	
GET	/api/applications/:userId/:token	userId: a key in User db table, token: authentication value for userId	Get applications of the userId.
POST	/api/applications	Posting id, Application text(string), user who is creating the application is posted in the http request body	Creates a new application and add it to the database.
PUT	/api/applications/:app_id/read	app_id: a key in Application db table	Updates read status of the application
GET	/api/messages/:userId/:token	userId: a key in User db table, token: authentication value for userId	Get messages of the userId.
PUT	/api/messages/:msg_id/read	msg_id: a key in Message db table	Creates a new message and add it to the database.
POST	/api/messages	Message text(string), user id who is creating the message, user id who is receiving the message are posted in the http request body	Updates read status of the message
GET	/api/forumposts	-	Returns a collection of all the forum posts.
POST	/api/forumposts	User id who is creating the forum, type of forum, message, image url for the forum are posted in the http request body	Creates a new forum post.

PUT	/api/forumposts/:id/like	id: The numerical ID of the desired forum post	Increases the 'likes' by 1 on the forum post specified by id.
POST	/api/upload	A single image file under 1MB	Uploads a file to an Amazon S3 server.

## 7. Testing

### 7.1 Mocha automated testing

We used the mocha JavaScript test framework to test the functionality of our REST APIs and other functions. We wrote 17 test cases using the Mocha test framework which tests the main functionalities of our web application.

The instructions for running the test cases can be found inside the PetCare/test/tests.js file.

We categorized these 17 test cases into 5 main test suits.

1. Test cases that make http GET requests
2. Test cases that make http POST requests
3. Test cases that make http DELETE requests
4. Test cases that make http PUT requests
5. Test cases that test for validity of user inputs

We setup our mocha test script to automatically open the node.js web server on port 8989 before running these test and close the web server after the test cases are done. We have also wrote code for removing any data that is added to the database after performing POST or PUT requests.

```
GET Request Test Suite:
  Make an http GET request to /api/users/:id
    ✓ Should respond with 200 (46ms)
  Retrieve pet data for a specified pet
    ✓ should respond with 200
  Retrieve pet posting data for every posting
    ✓ should respond with 200
  Access to another user's messages
    ✓ should respond with 401, unauthorized
  Access to another user's applications
    ✓ should respond with 401, unauthorized

POST Request Test Suite:
  Make an http POST to /auth/register
    ✓ Should create new user (1008ms)
  Make an http POST to /api/login
    ✓ Should login the user (483ms)
  Make an http GET to /api/logout
    ✓ Should logout the user
  Create a new pet
    ✓ should create new pet and return pet information
  Create a new sitter posting
    ✓ should create new sitter posting and return posting information
  Send a message
    ✓ should create a new message

DELETE Request Test Suite:
  Delete post
    ✓ Should post and delete pet post

PUT Request Test Suite:
  Ban a user
    ✓ Should create new user and ban that user (980ms)
  Like a forum post
    ✓ should increase the like on a forum post by 1
  Close a sitter posting
closed posting
    ✓ should set a sitter posting status to closed

Valid Integer Test Suite
  Valid integer tests
    ✓ should return true if value is an integer
  Invalid integer tests
    ✓ should return true if value is an integer

17 passing (3s)
```

Mocha test results

## 8. YouTube Link

[https://www.youtube.com/watch?v=IN4HoH-9u\\_o](https://www.youtube.com/watch?v=IN4HoH-9u_o)

## 9. Heroku Link

<https://petcare-service.herokuapp.com/>

## 10. GitHub Link

<https://github.com/davidla25/PetCare>

Note: You must be added as a contributor to access the repository

## 11. Additional Information

### The Google Maps Geocoding API

For the initial search page, the search system recommends postings based on user's location. If the user is not logged in, the search system uses Google's Geocoding API to get user's location. Here is an example url that requests json object for geolocation of latitude 43.659 and longitude -79.39:

<http://maps.googleapis.com/maps/api/geocode/json?latlng=43.659,-79.39&sensor=true>

Here is part of the response json object:

```
{
  "long_name" : "Toronto",
  "short_name" : "Toronto",
  "types" : [ "locality", "political" ]
},
```

By using this api, search system can recommend postings to users although they are not logged in.

---

**README.pdf** - need to make this a separate pdf

### Setup on local machine

1. Install nodejs
2. Install npm
3. Install mongoDB 3.2 Community Edition  
OS X: <https://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/>  
Windows: <https://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/>
4. On OS X/Linux, copy the Amazon AWS credentials file into ~/.aws/credentials  
on Windows, copy the credentials file into C:\Users\USERNAME\.aws\credentials  
This file contains the Amazon AWS credentials used for uploading images to Amazon S3 storage.  
**NOTE:** You can find the `credentials` file inside the `a4p2.zip` in the `PetCare` directory
5. Execute ``npm install`` to install the dependencies listed in the `package.json` file.

## How To Run

1. Open two terminals.
2. In one terminal, run ``mongod --dbpath ./mongodb_data`` to start the mongoDB daemon
3. In the second terminal, run ``node data/default-data.js`` to import the data  
And then run ``npm start`` to start the nodeJS serve
4. Go to <http://localhost:3000/> for the interface.

## How To Run Test Cases

Instructions on how to run the Mocha test cases can be found under `PetCare/test/test.js` file

## Admin and other user credentials

After running the `default-data.js` script as described above, users can sign in one of the following users by using these credentials. Users can also sign up with their own account.

### Admin Credentials

email: admin@gmail.com  
password: admin123

### Default user 1 credentials

email: jennifer@gmail.com  
password: 12345678

### Default user 2 credentials

email: bale@gmail.com  
password: 12345678

## List of URLs

All the REST API end-points are listed in section 6 of the `report.pdf` document.

URLs that does not require sign in

/

/forum

/petsitter\_posts **NOTE:** Default search results may take time to load depending on the network

/pet\_posts **NOTE:** Default search results may take time to load depending on the network

/pet\_posts/:id

/petsitter\_posts/:id

/users/:id

URLs that does require sign in

/new\_pet\_posts

/new\_petsitter\_posts

/users/:id/applications

/users/:id/messages

**NOTE:** You have to sign in to make a Review or to Apply

URL that only Admins can access

/admin

**NOTE:** Admins can access the Admin Console by clicking on their name on the upper left corner and clicking 'Admin Console'

## **TODO:**

Lisa

- MVC

- authentication

- user profile

  - add close posting

  - add open posting

- redirect to admin on login

- add test cases

Taewoo

- search

- geolocation

- add test cases

David

- make video

Rahal

- readme pdf

- replace default profile picture

- add performance python file
- update test case information

Everyone:

- remove console.log() from all files (server and angular)
- add test cases
- add comments

**Before submission:**

- Add performance files to zip
- Add readme.pdf
- Add report.pdf
- Add test folder