"The Great Game Share"

GROUP 0203:
Alana Tinney (c3tinney)
Connor Yoshimoto (g4yoshi)
William Le (c3lewill)
Jordan Martel (g4martel)

DESCRIPTION
Our application is a game sharing site. At its most fundamental level it provides a platform for users to sell or rent games to those interested in them. A common scenario for people playing video games, known as gamers hereafter, is that they accumulate a collection of games that they no longer wish to play. Typically, this is due to the fact that the gamer has completed said game but it may also be due to simple dislike. Regardless of the reason, a gamer typically builds a significant collection of these games, with the most prolific gamers having collections of upwards of a hundred games. At this point in time, these games hold little value to the gamer herself but may represent a significant value for someone who has not played the game. In this fashion, our site hopes to capitalize on the unused collections of games that people possess by allowing said people to sell or rent these games to people who have an interest in them. Our site provides a marketplace for games, allowing users to quickly and easily sell, rent or purchase games.

DESIGN
The application can be divided into frontend, server and database sections. When an action occurs on the frontend, the server reacts accordingly, and stores or receives data to and from the database.

The frontend sections accounts for all the HTML, CSS and asset files. It also includes several view files created by the handlebars module. The following pages are included in the frontend:

iii. The list of pages and UI elements of your project.
Note html files are only a template. The project uses handlebars with views/layouts to display the information
adminupdate.html - used by the admin to change information of a specified user
login.html - this is the page seen when logging in or signing up to the page
mainpage.html - this is the main page that is seen when loading the site, or after logging in.
myuserpage.html - user’s profile page containing all of their info
posting.html - this page allows the user to create a new game posting
product.html - this page contains information about a posted game
search.html - contains the search results that respond to a game search
update.html - allows the logged in user to edit their profile information
userpage.html - contains the information of the selected user to be viewed. Allows another user to rate them

usersearch.html - The page where the user inputs a username to search for
The server section is a single JavaScript file, server.js, that works with the data from the frontend and database. The server will format data correctly so that it can be stored into the database. Similarly, it will also deal with information retrieved from the database and send it to the frontend. Also, the server takes care of authentication, as well as input sanitation (see security section below). In order to access the database, the server works with the DB.js file to make the required function calls.

The database is represented by a single file DB.js. This file contains all of the functions that will be used to access the database. The functions are available for the server.js file to call. The database section has 3 main sub-sections:

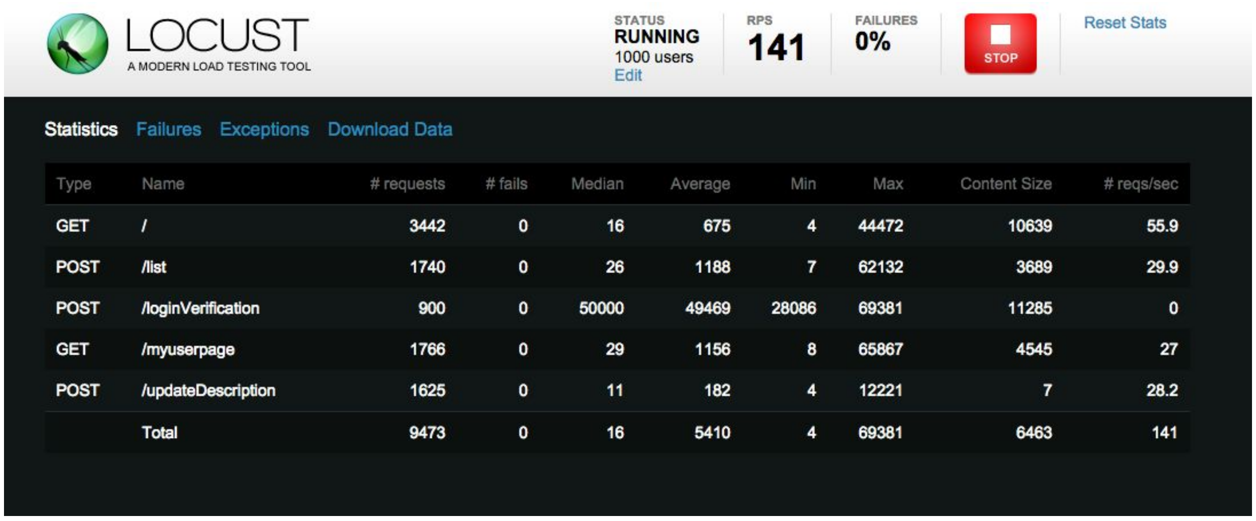
1) Users - stores information about a registered user. This section also takes care of retrieving information about a user from the database. It will store username, email, password (hashed) and the admin type of the user.
2) Posts (Game postings) - stores information about game postings. When a new posting is made by a user, the database stores information such as title, tags and the username of the user that created it. This section is used when searching for games, adding games and renting games
3) Reviews - stores information about user reviews. When a user reviews another user, a new “review” is created in the database. This stores important information about the review such as the rating, description and usernames of the reviewer and reviewee.

SECURITY AND TESTING

Authentication and XSS are taken care of by the server. Using the nodejs bcrypt module, the server hashes and stores all passwords in the database. When a user attempts to login, the server will use the bcrypt module to compare the password entered with the hashed password corresponding with the requested username. The sanitize-html module is used to take care of XSS. Whenever input is given to the server, that input has all tags removed using the module in order to prevent any script or other formatted data input from entering the database. This prevents script from being loaded onto the page when read from the database at a later point.

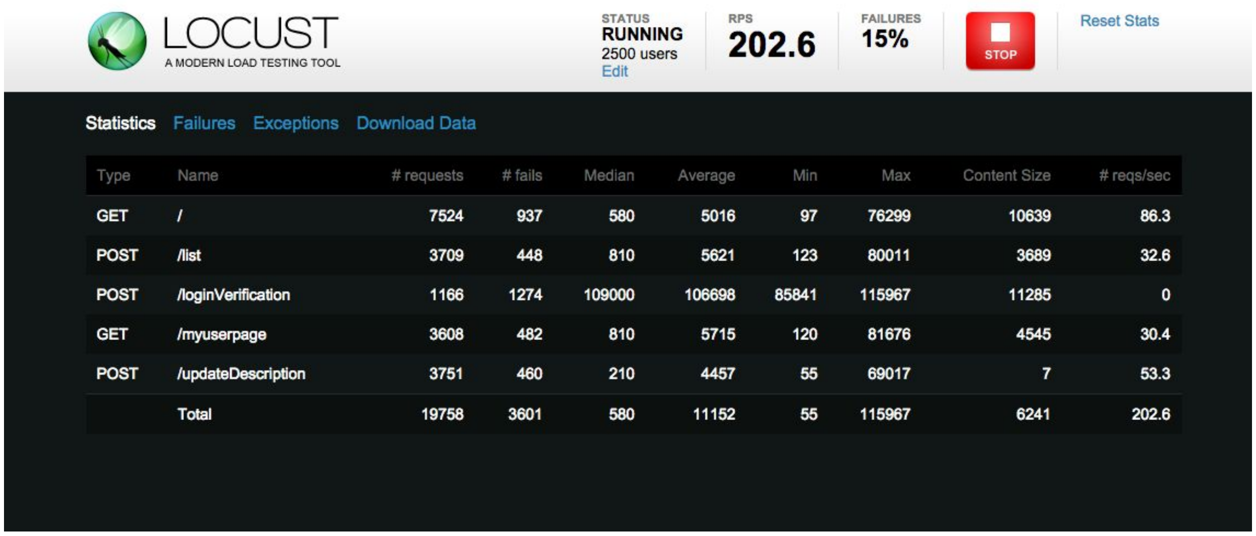
PERFORMANCE

To improve our website’s performance we took advantage of express.js’s built-in cache control. We allowed our public folder files to be cached for multiple days, so our static files (like css) do not need to be retrieved from the server after every request. Originally we started to receive a sizeable number of fails with 2500 users, but with caching we reduced the fail per cent from 15% to 4% at peak usage.
Below is a series of locust statistics. The first two are from our original website before we added any optimization. With a peak of 1000 users averaging 141 requests per second, there were no fails and everything worked as it should.



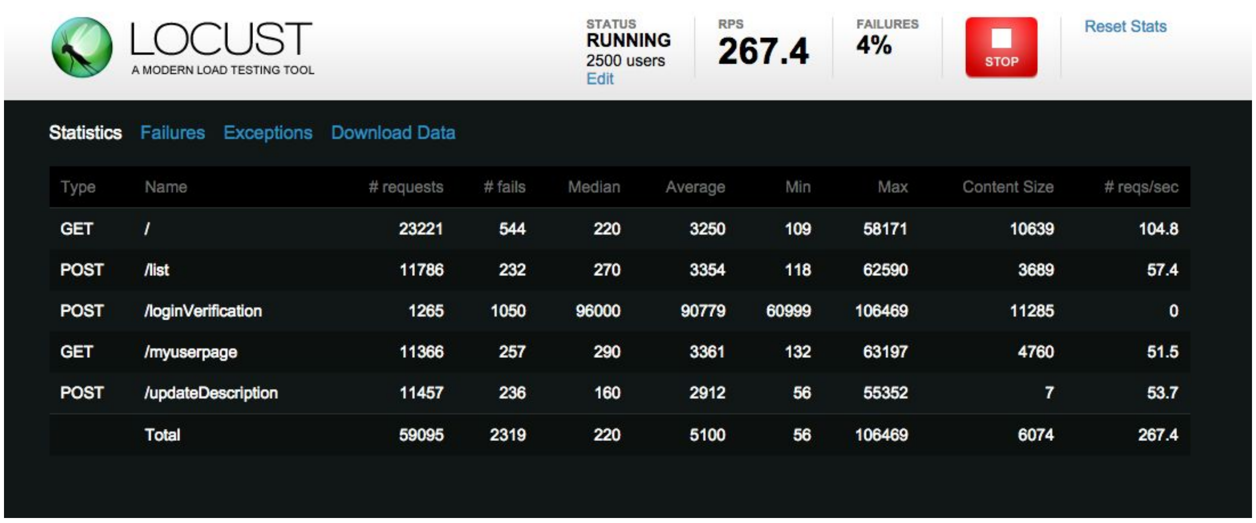
Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/	3442	0	16	675	4	44472	10639	55.9
POST	/list	1740	0	26	1188	7	62132	3689	29.9
POST	/loginVerification	900	0	50000	49469	28086	69381	11285	0
GET	/myuserpage	1766	0	29	1156	8	65867	4545	27
POST	/updateDescription	1625	0	11	182	4	12221	7	28.2
	Total	9473	0	16	5410	4	69381	6463	141

However at 2500 peak users with 202 requests per second, we were get a 15% failure rate for requests, as shown below.



Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/	7524	937	580	5016	97	76299	10639	86.3
POST	/list	3709	448	810	5621	123	80011	3689	32.6
POST	/loginVerification	1166	1274	109000	106698	85841	115967	11285	0
GET	/myuserpage	3608	482	810	5715	120	81676	4545	30.4
POST	/updateDescription	3751	460	210	4457	55	69017	7	53.3
	Total	19758	3601	580	11152	55	115967	6241	202.6

The following locust test is after we added our optimization, it was able to handle the 2500 users better with a higher RPS rate of 267 and a much lower failure rate of only 4%.



Type	Name	# requests	# fails	Median	Average	Min	Max	Content Size	# reqs/sec
GET	/	23221	544	220	3250	109	58171	10639	104.8
POST	/list	11786	232	270	3354	118	62590	3689	57.4
POST	/loginVerification	1265	1050	96000	90779	60999	106469	11285	0
GET	/myuserpage	11366	257	290	3361	132	63197	4760	51.5
POST	/updateDescription	11457	236	160	2912	56	55352	7	53.7
	Total	59095	2319	220	5100	56	106469	6074	267.4

Video Demo
http://www.screencast.com/t/cuAE2n65VFue