REPORT

INFORMATION

DEVELOPED BY

Albert Calzaretto (g5ex)

Roman Polyanovsky (g4roman)

Umut Akkaya (g4akkaya)

Yaroslav Taben (g4tabeny)

PROJECT DESCRIPTION

Our project is a social economy specifically allowing users to post, rate, and comment on Minecraft (a video game) servers. When a user adds a server, our systems automatically query the server for various gameplay parameters, and generate a dynamic, periodically updated page for the server that shows the online players, the player history, and if the server supports it, a dynamic live overview of the server topography powered by DynMap. The server page allows users to comment on servers. If a user is known by our system to have played on a server, then the comment will show as "verified". Users can also "like" servers, in order to get recommendations of similar servers on their home page.

SECURITY

As part of the security of the applications, the passwords are double hashed, first round is MD5 hash, and then then we use bcrypt with 2^8 rounds to hash the password that has already been previously hashed by MD5 (8 bit hashing on top of MD5 hash). For added protection against unauthorized links that would delete servers or do other potential damage to the database, we provide security on the front and back end to make sure that the user is an admin/super admin in order to do so. Having the two layers builds a wall between what the user can do and can't do to the database. Additionally, we use the express-validator module to sanitize and escape all user input received by any controller.

TESTING

Many of the tests were and can only be conducted via user testing. We did however use mocha to test the basic tests on possible vulnerabilities that could get into the server, which was one of the frameworks we used to test, as well as basic server recovery on invalid inputs. The majority of the testing for our application was however manually trying to break the website, and user experience testing which cannot be done properly in Mocha which would require a user to be logged in.

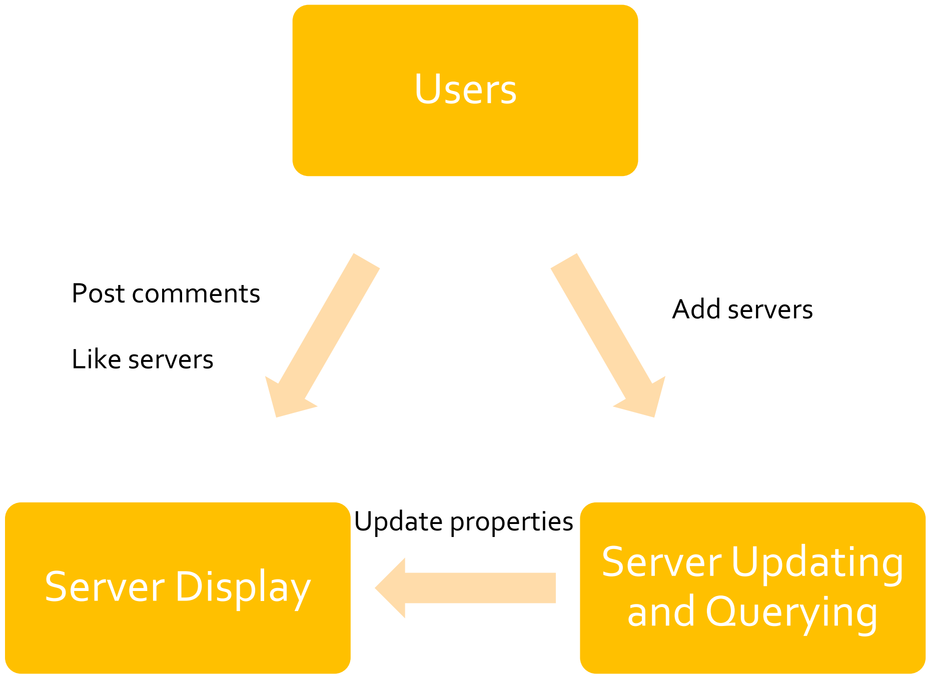
PERFORMANCE

In order to improve performance, we use two approaches: The first is the built-in cache control system from express, which send 304 results in order to indicate to a browser that it should use its cache. The second is the use of the express-compression module which compresses our server HTML responses, as well as static resources with zlib, resulting in much smaller transfer sizes.

DESIGN

i. Our system employs open source node modules mcquery and minecraft-protocol. When a "server stat" (updating of server parameters) is requested by either a user adding a server or the background update daemon, the system attempts to connect to the server through mcquery, and perform a "full stat" which updates all server properties supported by Minecraft's Query Protocol. If the stat fails (since not all servers enable query protocol), the system falls back to a basic server ping using minecraft-protocol, which provides a few (much less than a stat) parameters about the server.
The server update system works in tandem with the server management/comment/rating system which allows users to post comments on servers and like them. This system also generates recommendations of servers for players based on the parameters of servers that they liked.
Lastly we have our user management system which allows creation (and login/access) of accounts through our system, Google account login, and Minecraft account Login.

ii. The systems work together in a very simple interaction scheme:



iii. Home page (/)
Local Registration (/user/register)
Local Login (/auth/local/login)
Google Login (/auth/google/login)
Minecraft Login (/auth/minecraft/login)
Server List (/server/list)
Add Server (/server/create)

View server (/<ip>/<port>)

OTHER

In order to be good Samaritans on the web, we responded with the appropriate headers with every request made to the server, from a 503 to a 404 and the default 200.