View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by La Salle University Digital Commons

La Salle University
La Salle University Digital Commons

Mathematics and Computer Science Capstones | Scholarship

Spring 5-18-2018

# RCON Administration tool designed for use with Garry's Mod and Source Servers

John Gibbons
*La Salle University*, gibbonsj3@student.lasalle.edu

Follow this and additional works at: https://digitalcommons.lasalle.edu/mathcompcapstones


Part of the Graphics and Human Computer Interfaces Commons, Other Computer Sciences Commons, and the Programming Languages and Compilers Commons

Recommended Citation
Gibbons, John, "RCON Administration tool designed for use with Garry's Mod and Source Servers" (2018). *Mathematics and Computer Science Capstones*. 39.
https://digitalcommons.lasalle.edu/mathcompcapstones/39

This Thesis is brought to you for free and open access by the Scholarship at La Salle University Digital Commons. It has been accepted for inclusion in Mathematics and Computer Science Capstones by an authorized administrator of La Salle University Digital Commons. For more information, please contact careyc@lasalle.edu.


RCON Administration tool designed for use with Garry's Mod and Source Servers

https://play.google.com/store/apps/details?id=gibbons.gcon

https://gitlab.com/gibbonsjohnm/gcon

# Table of Contents

1 Executive Summary ..... 4
1.1 Summary ..... 4
1.2 Problem ..... 4
1.3 Motivation ..... 4
2 Planning ..... 6
3 Design ..... 7
3.1 Activities/Layouts ..... 7
3.1.1 AddServerActivity ..... 7
3.1.2 ListServerActivity ..... 8
3.1.3 AboutServerActivity ..... 9
3.2 SQLite ..... 11
4 Development ..... 12
4.1 NetworkUtilities ..... 12
4.1.1 AuthUpdater ..... 12
4.1.2 IPQuery ..... 12
4.1.3 JSONReader ..... 12
4.1.4 PlayerQuery ..... 13
4.1.5 RCONAuth ..... 13
4.1.6 RCONCommand ..... 13
4.1.7 ServerPing ..... 14
4.1.8 SteamQuery ..... 14
4.2 Log Receiver ..... 14
4.3 Log Server ..... 15
4.4 External Libraries - Android ..... 15
4.4.1 Steam-Condenser ..... 15
4.4.2 Glide ..... 15
4.5 External Libraries – NodeJS/NPM ..... 16
4.5.1 Srcds Log Receiver ..... 16
4.5.2 Log Server ..... 16
5 Technical Difficulties/Lessons Learned ..... 17
5.1 Logging ..... 17
5.2 Asynchronous Tasks ..... 17

6 Lessons Learned ..... 19
6.1 Testing ..... 19
6.2 CI/CD ..... 19
6.3 Saying “No” ..... 19
7 Moving Forward ..... 20
7.1 Firebase ..... 20
7.2 Google Play Console ..... 20
7.3 SonarQube ..... 21
8 Future Development ..... 22

# 1 Executive Summary

## 1.1 Summary

The proposed project is an Android application designed to interact with the Source RCON protocol. Defined below:

*The Source RCON Protocol is a TCP/IP-based communication protocol used by Source Dedicated Server, which allows console commands to be issued to the server via a "remote console", or RCON. The most common use of RCON is to allow server owners to control their game servers without direct access to the machine the server is running on. In order for commands to be accepted, the connection must first be authenticated using the server's RCON password, which can be set using the console variablercon_password.*

The application will be designed entirely in Android/Java, utilizing libraries and APIs for querying and interacting with Source Dedicated Servers (https://developer.valvesoftware.com/wiki/Source_Dedicated_Server) and Steam user profiles (https://developer.valvesoftware.com/wiki/Steam_Web_API).

The application will report information about servers to the user, including host name, game, game mode, map, pings, connected players, logs, commands, among many other things.

The application will use SQLite for local databases, containing information related to servers, including hostname/ip, game port, and RCON password. A separate nodejs application for sending and storing server logs will be required for users who wish to view their server's logs in the Android application.

The project will be stored in GitHub or GitLab and will utilize the CI/CD processes offered for automated build, testing, and delivery to the Google Play Store. This can also be accomplished in Jenkins, using Docker build slaves. Depending on research and performance, one of those avenues will ultimately be selected.

## 1.2 Problem

Currently, the only way to interact with game servers remotely is via RCON browser tools or outdated Java applications. Several Android applications exist, but do not possess all of the features discussed above. They are especially lacking detailed player information, as well as real time logging data. A game server is like any other server; it requires monitoring, alerting, and upkeep. For owners and administrators who wish to monitor their server(s) on the go, a mobile application is necessary.

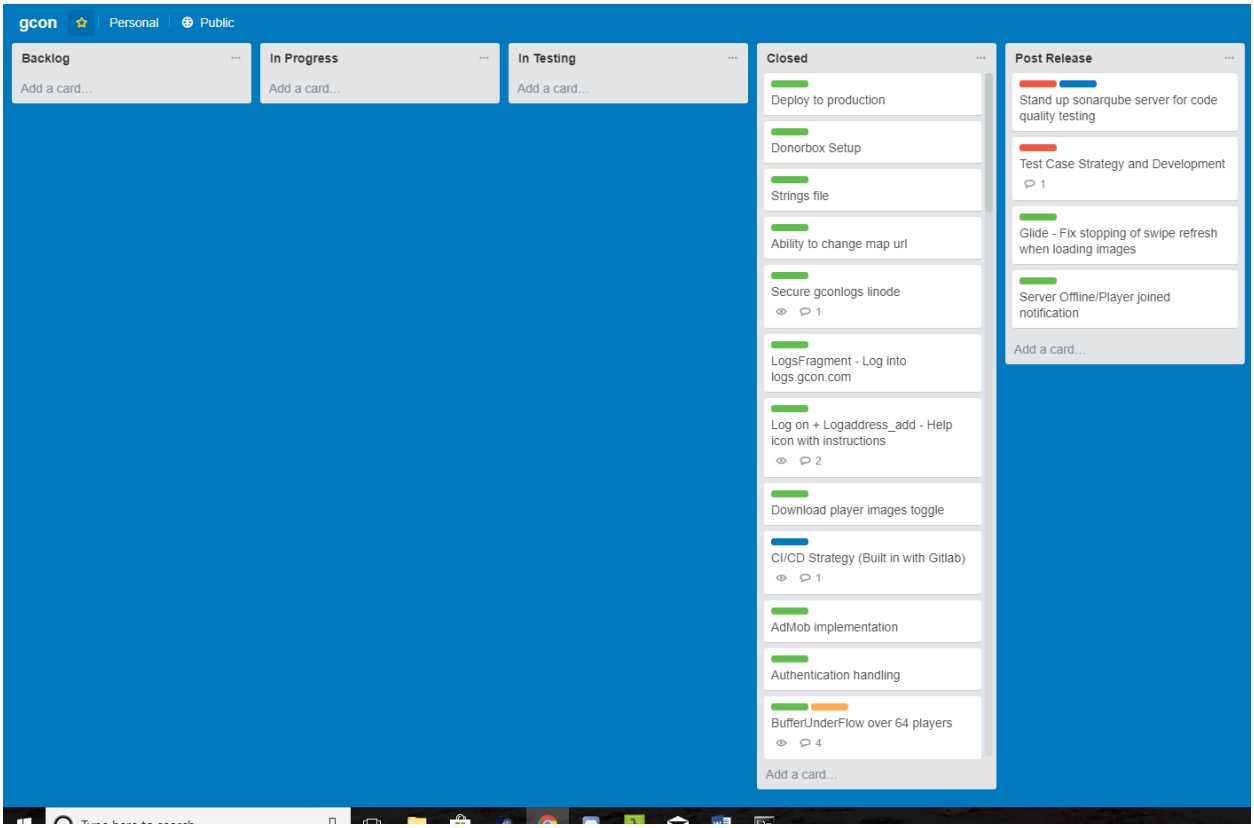
## 1.3 Motivation

The main focus and development point of gcon was Garry's Mod (https://gmod.facepunch.com/). I've been playing this game for over 4 years now and have logged nearly 1000 hours. Most of that time was spent creating, fiddling with, and hosting servers for both friends and strangers. It became an instant passion, allowing me to game while also channeling my creativity via Lua programming (https://www.lua.org/about.html) and Linux administration. The only thing lacking was a remote, mobile

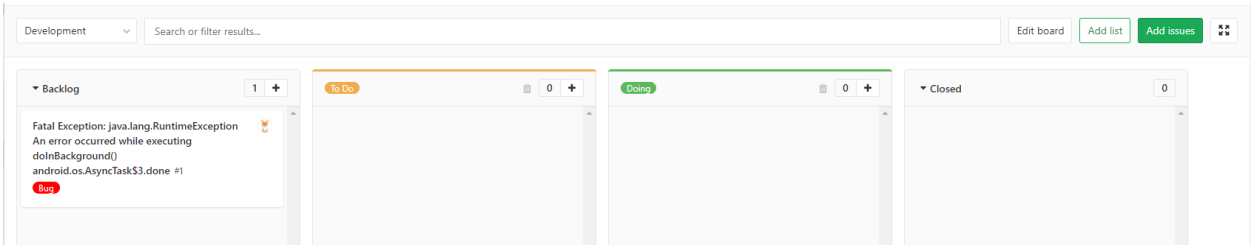
administration tool. To check my servers without being physically on them, I had to either go to clunky URLs not built for mobile devices or write custom python scripts without any GUI. The lack of a strong mobile market was another large factor.

# 2 Planning

Despite being the only developer on the project, I felt it was important to map out tasks to realistically determine timelines for completion. I decided to adopt an Agile-like methodology to accomplish my goal. Using a Trello board (https://trello.com/b/xIMb6n07/gcon), I created tasks to complete as well as sub headers to appropriately organize the tasks. The sub headers included: “To do”, “In Progress”, “In Testing”, “Completed.” I tried to stick to those tasks throughout the project, adding items as I progressed.




My plan is to eventually move this list to GitLab’s built in Boards feature. I’ve already begun using it for issue tracking and would love to setup a feature that created an issue in GitLab for every crash event in the app.



# 3 Design

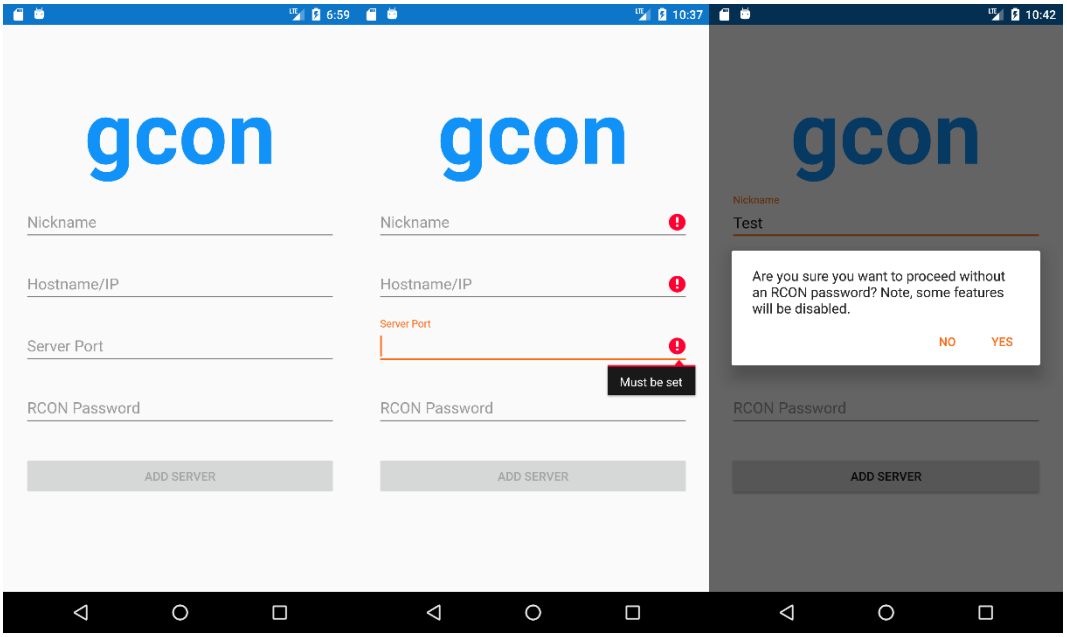
## 3.1 Activities/Layouts

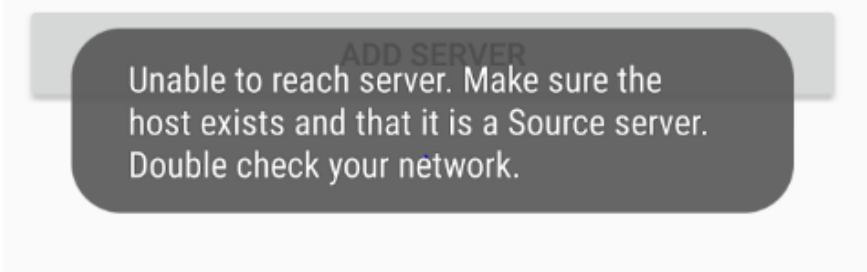
The first step in any mobile application is usually the design of the “screens.” In Android’s case, these are created using layout xml files (https://developer.android.com/guide/topics/ui/declaring-layout) that are later linked in code to Activity classes https://developer.android.com/reference/android/app/Activity). Because I opted for a fragment based structure, gcon only required the development of 3 activity classes. An overview of each class is provided below.

### 3.1.1 AddServerActivity

Upon fresh installation of the application, the first screen the user sees is the AddServerActivity. Until the user successfully adds a Source server, they cannot progress past this screen. 3 of the 4 fields listed in the above screenshot are required: Nickname, Hostname/IP, Server Port. RCON Password is an optional field; however, without one, the user will be unable to utilize all features of the application. The user is notified when the password field is left blank.

Additionally, if the server the user entered doesn’t exist, or is not a Source server, they will receive a Toast notification informing them of common things to verify.



Upon inputting a Source server, the application will either create a local SQLite database called gcon.db or add a new entry to the existing database. Once that completes, the user is directed to the ListServerActivity.

### 3.1.2 ListServerActivity

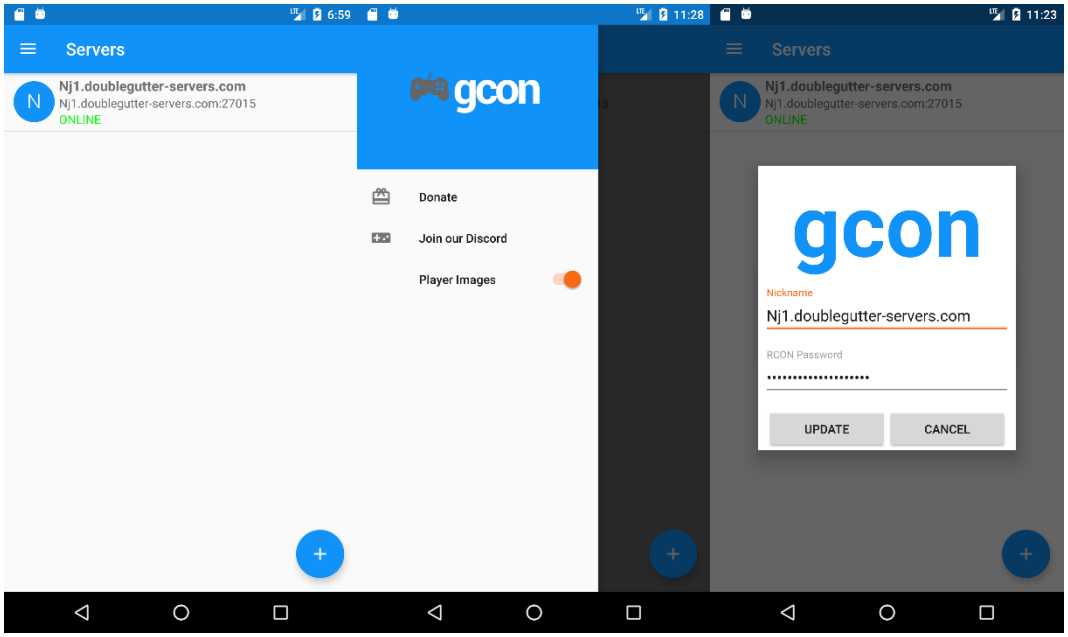
The ListServerActivity is what could be considered the “main” activity of the application. If the user has added a valid Source server to the database, they will be directed to this screen when opening the app.

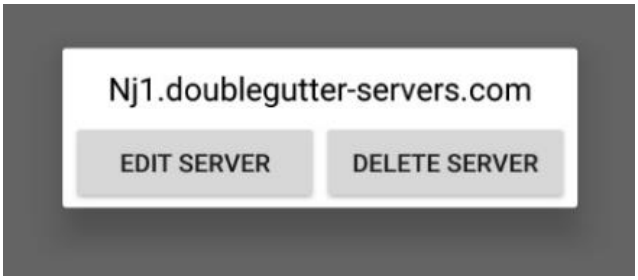
The screen contains a ListView, which scrapes data from the local SQLite database and puts it in readable form. It also provides the status of the server (ONLINE/OFFLINE).

The user can edit or delete an existing server or add a new one, using the blue floating action button at the bottom of the page.

Finally, the activity contains a navigation drawer for a link to my donation page, the community’s discord (https://discordapp.com/) server, and a toggle option to download player images.

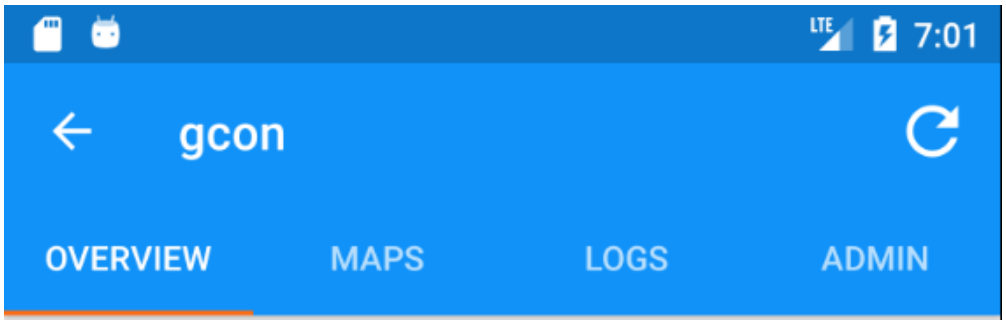
Each server in the list is clickable. If the server is online at the time of the onClick event, the user will be directed to the AboutServerActivity.



### 3.1.3 AboutServerActivity

This activity serves a handler for a ViewPager for 4 Fragment classes. The reason for this design was to achieve a tab based layout.

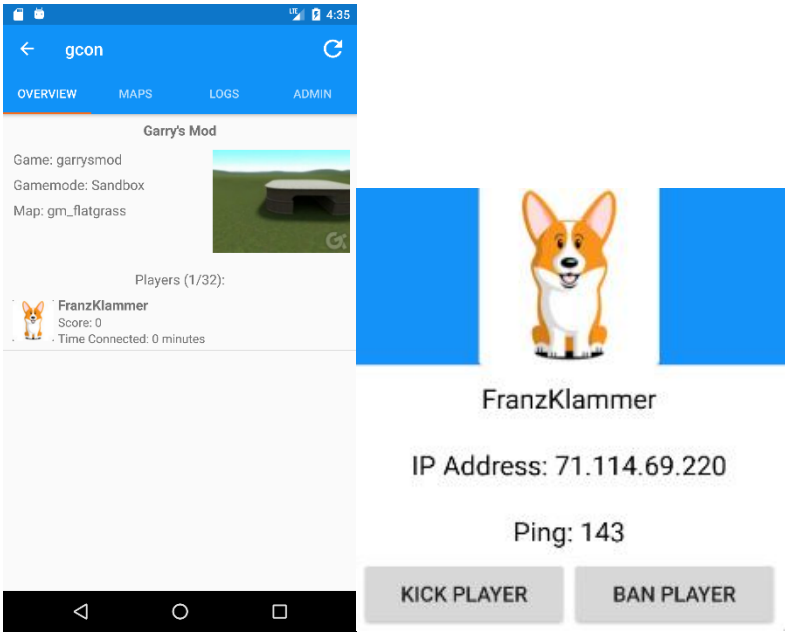



#### 3.1.3.1 SummaryFragment

The Summary Fragment contains all the essential information about a server: Game, Game mode, current Map, and connected player information.

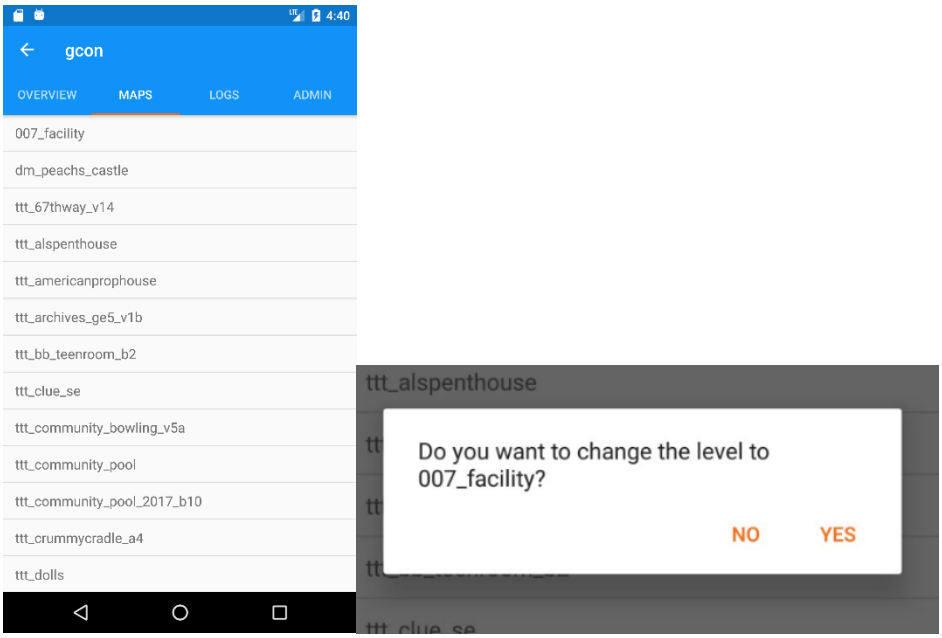
The current map image is pulled from https://www.gametracker.com/ and uses the game mode and map name to attempt to grab the appropriate image. The user can change the name of the game mode to adhere to game tracker's default, if images aren't loading properly.

Each player is contained in a ListView that holds the player profile image (downloaded from https://store.steampowered.com/), how long the player has been connected, and their current score. If the server is successfully authenticated, more information about a player will be available after the onClick event, including IP address, ping, and the ability to kick or ban the player.



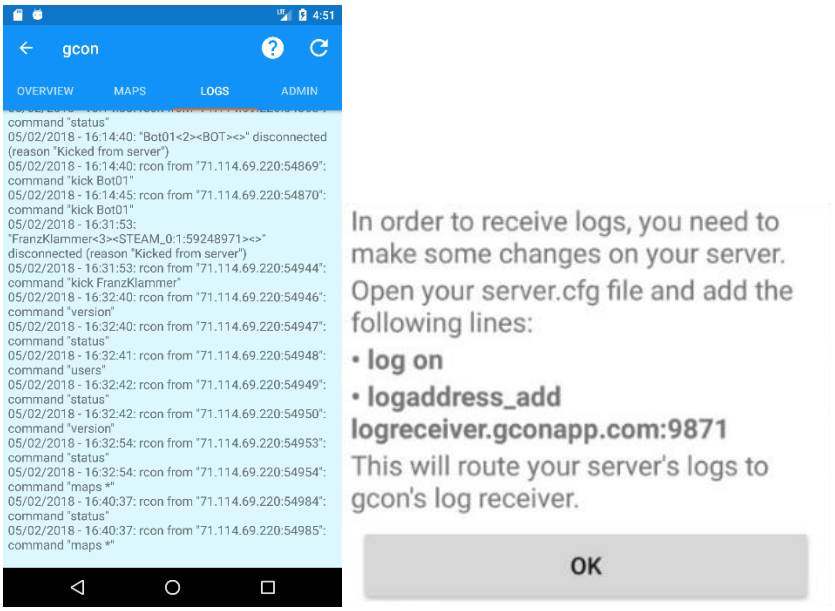
### 3.1.3.2 MapsFragment

The MapsFragment uses the RCONCommand class to determine all the current maps on the server. It runs “maps *” to get a full list of maps, and then parses that information into the ListView. Each map is clickable, and the user can change the current map on the server to any available in the list.



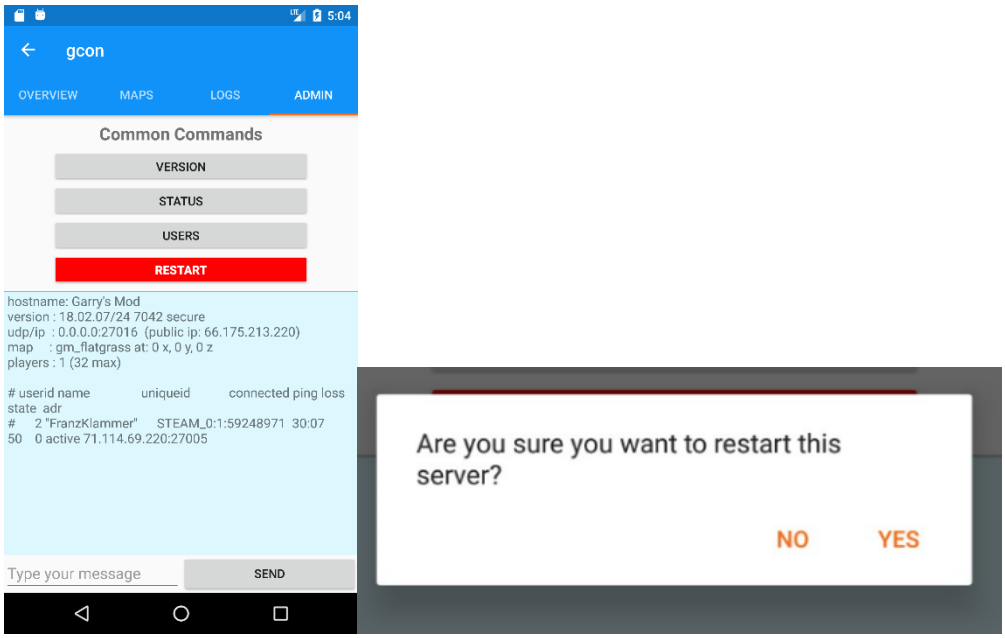

### 3.1.3.3 LogsFragment

The LogsFragment directly integrates with gcon’s log receiver. If the server is authenticated, this class will attempt to locate the server’s log file on the log receiver, using the server’s IP address and Port. If the file is located, the last 20 lines of the log are parsed and displayed in a TextView. The user can refresh the log at any time. For cases where the log can’t be located, the user will be notified of common ways to fix the problem.



#### 3.1.3.4 AdminFragment

The AdminFragment provides a way to run common commands on the server, such as version, status, and users. It also allows users to attempt to restart their servers, in case of unexpected crashes or maintenance. Its main feature, however, is the ability to run any custom command on the server. Each command, custom or built-in, will print its output to a TextView.




## 3.2 SQLite

https://developer.android.com/training/data-storage/sqlite

The SQLite structure for this application is very simplistic, yet critical. It is stored in a file called gcon.db and has the following attributes:

```
public static final String TABLE_NAME = "rcon";
public static final String COLUMN_NAME_HOSTNAME = "hostname";
public static final String COLUMN_NAME_NICKNAME = "nickname";
public static final String COLUMN_NAME_PORT = "port";
public static final String COLUMN_NAME_FULL= "full";
public static final String COLUMN_NAME_PASSWORD = "pass";
public static final String COLUMN_NAME_AUTHENTICATED = "auth";
public static final String COLUMN_NAME_GAMENAME = "gamename";
private static final String SQL_CREATE_ENTRIES =
        "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
                FeedEntry._ID + " INTEGER PRIMARY KEY," +
                FeedEntry.COLUMN_NAME_HOSTNAME + " TEXT," +
                FeedEntry.COLUMN_NAME_NICKNAME + " TEXT," +
                FeedEntry.COLUMN_NAME_PORT + " INT," +
                FeedEntry.COLUMN_NAME_FULL + " TEXT," +
                FeedEntry.COLUMN_NAME_PASSWORD + " TEXT," +
                FeedEntry.COLUMN_NAME_AUTHENTICATED + " TEXT," +
                FeedEntry.COLUMN_NAME_GAMENAME + " TEXT )";
```

This database is accessed constantly throughout the lifecycle of the application.

# 4 Development

## 4.1 NetworkUtilities

Each of the classes in the NetworkUtilities package interacts with the Source Server, using the Steam-Condenser, or with gcon's log receiver. Every action is performed via a nested ASyncTask (https://developer.android.com/reference/android/os/AsyncTask) to run network code in a background thread. The classes are used throughout the lifecycle of the application.

### 4.1.1 AuthUpdater

Runs when a user edits the server to use a new password or when a required authenticated activity happens and the server is no longer authenticated. Updates the authentication field in the SQLite db with the appropriate authentication status.

```
values.put(FeedReaderContract.FeedEntry.COLUMN_NAME_AUTHENTICATED, "false");
        db.update(FeedReaderContract.FeedEntry.TABLE_NAME, values, "_id=" +
sqlHandler.getRows("_id").get(position), null);
```

### 4.1.2 IPQuery

Runs on LogsFragment to return a list of IPAddresses from the SourceServer object. The IPv4 address is used when retrieving logs from the log receiver.

```
SourceServer test = new SourceServer(server);
        ipAddresses = test.getIpAddresses();
        return ipAddresses;
```

### 4.1.3 JSONReader

Runs on LogsFragment. Authenticates with the log receiver, parses the log, and returns the last 20 lines to be printed in the logs TextView.

```
try {
            InputStream is = new URL(params[0]).openStream();
            try {
                BufferedReader rd = new BufferedReader(new InputStreamReader(is, Charset.forName("UTF-8")));
                String jsonText = readAll(rd);
                JSONObject json = new JSONObject(jsonText);
                return json;
            } finally {
                is.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (JSONException e) {
            e.printStackTrace();
        }
```

### 4.1.4 PlayerQuery

Uses a SourceServer's getPlayers() method to return a hashmap of player information including name, time connected, and score. If the server is authenticated, more information about the players is retrieved, such as IP Address, SteamId, ping, etc.

```
SourceServer Server = new SourceServer(server);
if(auth && dlPlayerImages ){
        return server.getPlayers(pass);
    }
    else{
        return server.getPlayers();
    }
```

### 4.1.5 RCONAuth

Attempts to authenticate with the server and returns a Boolean indicating whether or not the attempt was successful. Occurs during the initial creation of the server if an RCON password is specified, as well as when a server is edited under the same conditions.

```
public class RCONAuth {
  public boolean getAuth(String server, String pass) {
    try {
      SourceServer test = new SourceServer(server);
      test.rconAuth(pass);
      return test.isRconAuthenticated();
    } catch (SteamCondenserException e) {
      e.printStackTrace();
    } catch (TimeoutException e) {
      e.printStackTrace();
    }
    return false;
  }
```

### 4.1.6 RCONCommand

Used primarily in MapFragment (to return a list of maps and execute a map/level change) and AdminFragment (to run built-in and custom commands).

```
String response = null;
    try {
      SourceServer server = new SourceServer(serverName);
      server.rconAuth(pass);
      response = server.rconExec(command);
    } catch (TimeoutException e) {
      e.printStackTrace();
```

### 4.1.7 ServerPing

Returns the host's current ping to the server. Primarily used to determine if a server is offline or not.

```
while (true) {
        try {
            SourceServer test = new SourceServer(server);
            ping = test.getPing();
            break;
        } catch (SteamCondenserException e) {
            e.printStackTrace();
            if (++count == maxTries) break;
        } catch (TimeoutException e) {
            e.printStackTrace();
            if (++count == maxTries) break;
        }
    }
    return ping;
```

### 4.1.8 SteamQuery

Runs on SummaryFragment. Returns a Hashmap of server information including name, game mode, current map, connected players, maximum players, among many other things.

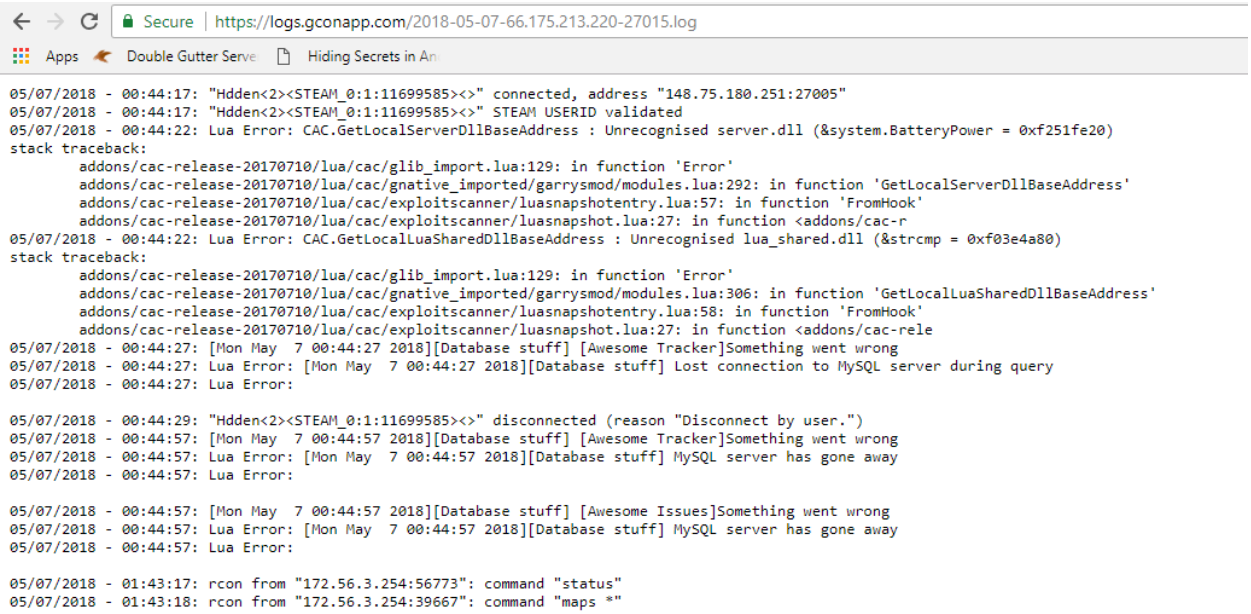
```
while (true) {
        try {
            SourceServer test = new SourceServer(server);
            serverInfo = test.getServerInfo();
            test.disconnect();
            break;
        } catch (SteamCondenserException e) {
            if (++count == maxTries) break;
        } catch (TimeoutException e) {
            if (++count == maxTries) break;
        }
    }
    return serverInfo;
```

## 4.2 Log Receiver

Gcon's log receiver takes advantage of Srcds Log Receiver to receive any logs from Source Servers configured to send servers to the receiver's URL (logreceiver.gconapp.com:9871). Each server gets its own daily log file.

## 4.3 Log Server

The log server lives on a Linode VPS (Virtual Private Server) and can be reached via https://logs.gconapp.com. It can only be accessed via password. In order to not use too much storage, the log server purges files older than 24 hours.


## 4.4 External Libraries - Android

### 4.4.1 Steam-Condenser

https://github.com/koraktor/steam-condenser/

The Steam Condenser is a multi-language library for querying the Steam Community, Source and GoldSrc game servers as well as the Steam master servers. Currently it is implemented in Java, PHP and Ruby.

Example:

```
SourceServer test = new SourceServer(server);
serverInfo = test.getServerInfo();
```

### 4.4.2 Glide

https://github.com/bumptech/glide

Glide is a fast and efficient open source media management and image loading framework for Android that wraps media decoding, memory and disk caching, and resource pooling into a simple and easy to use interface.

Example:

```
Glide.with(getContext())
        .setDefaultRequestOptions(requestOptions)
        .load("https://image.gametracker.com/images/maps/160x120/" +
sqlHandler.getRows("gamename").get(position) + "/" +
serverInfo.get("mapName").toString() + ".jpg").into(mapImg);
```

## 4.5 External Libraries – NodeJS/NPM

### 4.5.1 Srcds Log Receiver

https://www.npmjs.com/package/srcds-log-receiver

A library to receive logs directly from a Source dedicated server (srcds) via its UDP log transport (logaddress_add). As these logs are sent live during the game, this allows you to build interactive real time systems that react to in-game events.

### 4.5.2 Log Server

https://www.npmjs.com/package/log-server

A simple log server that receive text and dumps it into a file. Every log file will have a daily representation, meaning that you can easily delete old logs.

# 5 Technical Difficulties/Lessons Learned

## 5.1 Logging

A monitoring app, especially one designed for game servers, would be nearly useless without any sort of logging capability. Unfortunately, this turned out to be one of the more difficult things to implement. The RCON protocol does not provide this type of functionality and neither did Steam-Condenser. A few other libraries did; however, none of the solutions were designed for mobile. My original intention was to have each device be its own receiver, but that required the network the device was on to have a specific port open. This, for obvious reasons, is nearly impossible to guarantee.

Ultimately, I decided on an external Srcds Log Receiver. All logs live on a web server only accessible via password. End users will not be able to view their logs on the portal. I imagine many of my potential users will not be fond of having their server's logs on a web server. So, I left the choice up to them. Gcon will not attempt to forward logs to the log receiver unless the user performs specific actions on their game server. This should alleviate some concerns.

## 5.2 Asynchronous Tasks

On the Android platform, all network activity has to occur in a background thread. This is accomplished by using an Asynchronous task. At first, I was only using the AsyncTasks to perform a network activity using doInBackground() and return the value. This is not the intended use case. You are supposed to perform tasks prior to the desired network call and then return the value to the UI using OnPostExecute(). Once I figured this out, app performance improved significantly. An example of a proper AsyncTask below:

```
public class ChangeMap extends AsyncTask<String, Void, String> {
    int mapPosition;

    public ChangeMap(int mapPosition) {
        this.mapPosition = mapPosition;
    }


    @Override
    protected String doInBackground(String... params) {
        RconCommand execCommand = new RconCommand(getActivity(), position);
        String response = execCommand.getResponse(serverNames.get(position), serverPass.get(position),
"changelevel " + maps.get(mapPosition));
        return response;
    }

    @Override
    protected void onPostExecute(String response) {
        if (response == null) {
            Toast.makeText(getActivity(), R.string.unable,
                    Toast.LENGTH_SHORT).show();
        } else {
```

```
            Toast.makeText(getActivity(), getString(R.string.map_change) + maps.get(mapPosition) + ".",
                    Toast.LENGTH_SHORT).show();
        }
    }
```

# 6 Lessons Learned

## 6.1 Testing

Android has a wide array of options for testing. Unfortunately, I decided to put that off to the end and never got it fully featured. I had to rely on manual/alpha testing, firebase, and the internal testing google play provides when releasing an app. Test-driven development is something I need to start embracing to build the best application I possibly can. Down the line, I hope to make testing integrated into the build process and stop builds/deployments when a test case fails.

## 6.2 CI/CD

"Continuous integration, delivery, and deployment, known collectively as CI/CD, is an integral part of modern development intended to reduce errors during integration and deployment while increasing project velocity. CI/CD is a philosophy and set of practices often augmented by robust tooling that emphasize automated testing at each stage of the software pipeline. By incorporating these ideas into your practice, you can reduce the time required to integrate changes for a release and thoroughly test each change before moving it into production."

The ideas of Continuous integration, delivery, and deployment go hand-in-hand with a solid testing framework. While the initial release of an application via google play console requires a decent amount of manual work, future iterations do not. I believe I can easily leverage GitLab's CI/CD pipelines (https://docs.gitlab.com/ee/ci/) to automate testing and releasing.

## 6.3 Saying "No"

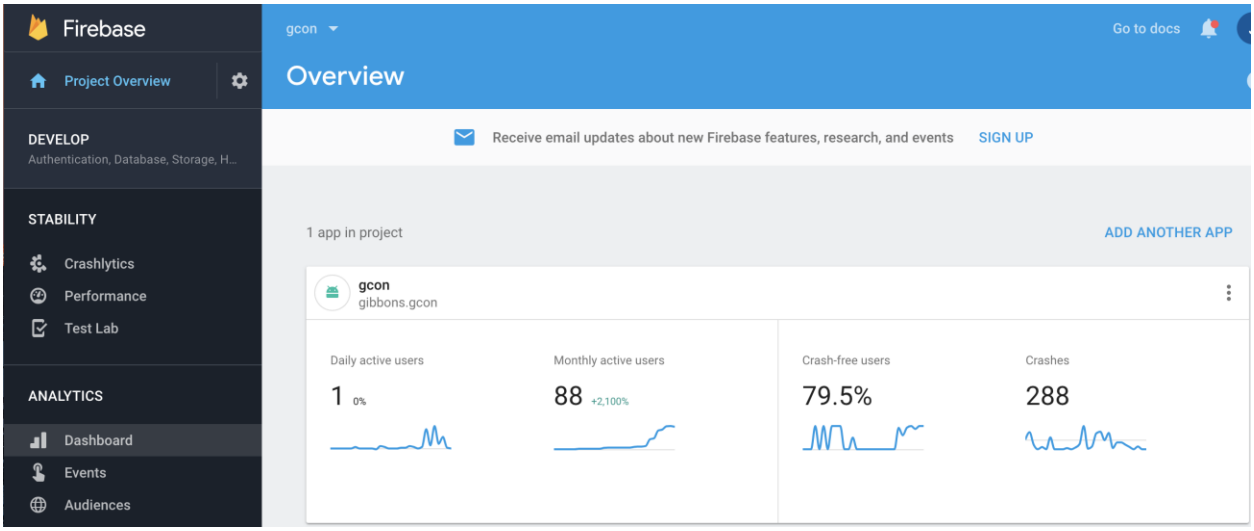
When I first came up with the idea for gcon, it was more featured than the version I released to the play store. While I still have hopes and development plans for those features, I decided it was best to focus on the essentials and produce an MVP, or minimum viable product (https://en.wikipedia.org/wiki/Minimum_viable_product). This is a development technique that companies have begun to adopt. It involves releasing software that is developed enough to satisfy early adopters. Features are then added based on feedback received. Sometimes, my idea for a feature isn't what's right, so I decided to let my user base help me figure out what works and what doesn't.

# 7 Moving Forward

## 7.1 Firebase

“Firebase helps you build better mobile apps and grow your business.”

I began using Firebase towards the end of my development cycle. I needed a way to monitor my app performance, especially crash events, while testing it on the go. Firebase provided a seamless way to do this. It integrates directly with the build process and creates a portal for the application in the Firebase console:



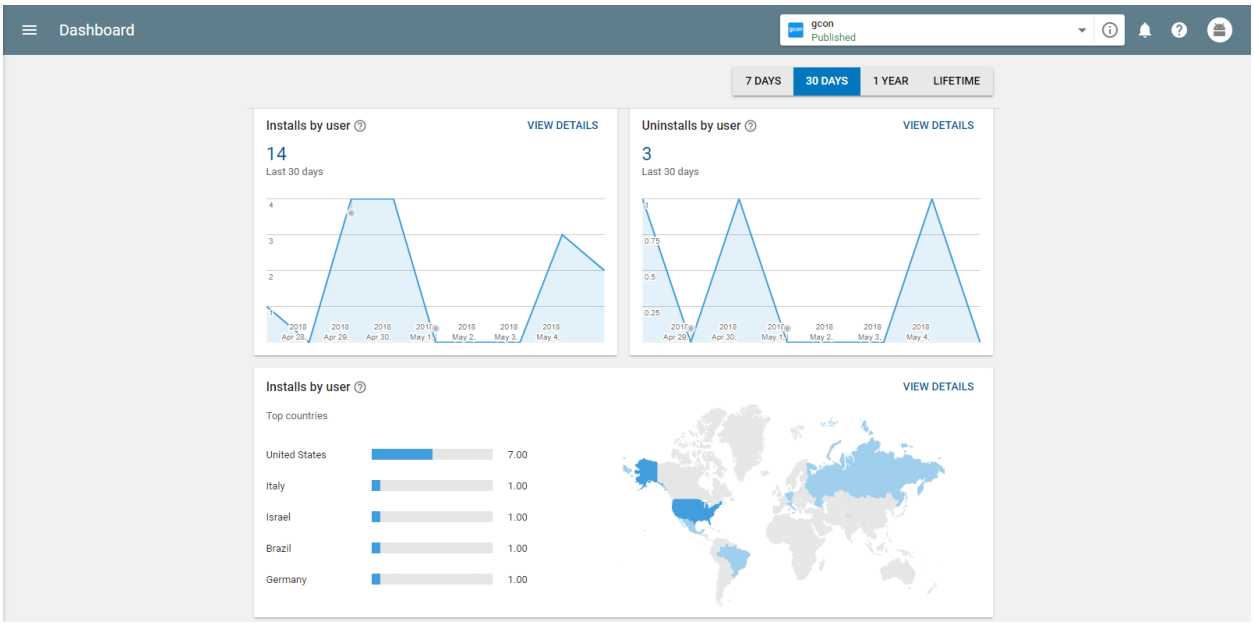

The best feature of Firebase, in my opinion, is that it emails you after it receives a crash event. This will allow me to monitor crashes and continuously improve.

## 7.2 Google Play Console

“Publish your apps and games with the Google Play Console and grow your business on Google Play. Benefit from features that help you improve your app’s quality, engage your audience, earn revenue, and more.”

The google play console is a “one-stop shop” for all android developers. You can test your app, publish, manage releases, view crash reports, downloads, user statistics, and many more essential items.

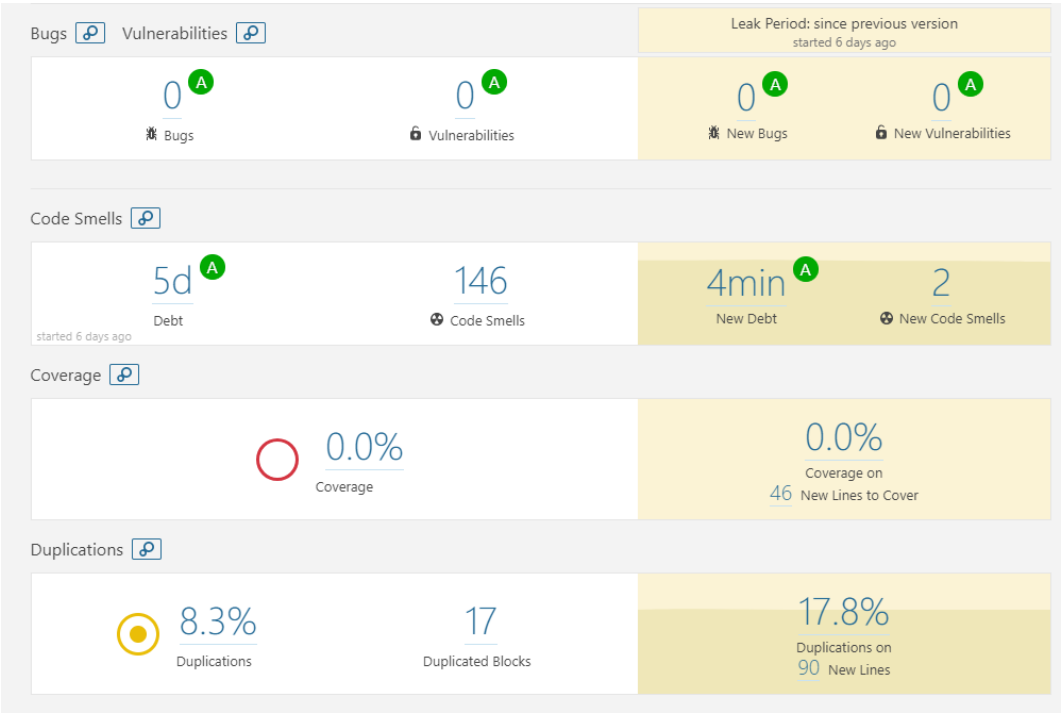
The console has already allowed me to see what audience I’m reaching and how effective each release has been.




## 7.3 SonarQube

"SonarQube provides the capability to not only show health of an application but also to highlight issues newly introduced. With a Quality Gate in place, you can fix the leak and therefore improve code quality systematically."

SonarQube has helped me identify critical bugs, security vulnerabilities, duplications of code, and areas for improvement in my application. I've already started a git branch to address the issues it finds. I plan on keeping my SonarQube dashboard clean throughout the lifecycle of gcon.



# 8 Future Development

To me, gcon is something that will always be a work in progress. In other words, I think there will always be room for improvement. I have many ideas to expand the app, as well as improve some of its current features. I hope to take the crash reports, feedback from google play reviews, and discussions on reddit to continue to make gcon the best at what it does.