FortNotification

We've got you covered.
The easiest way to keep track
of the items you want
is now here.

s3588952 s3602940

FortNotification

Team Members:

s3588952 – Isaac Underhill
s3602940 – Alex Rosenzweig

Contributions:

50/50

Links:

Github: https://github.com/isybub/fortnotification

FortNotification: http://www.fortnotification.appspot.com

Summary

The purpose of our project is to use the Fortnite API, and quite a few of Google's APIs to create a meaningful web application that fills a current gap in the market. This is a convenience product which aims to help games save time, similar game tracking products have proven to be both successful and profitable in the past. An example of a product that is similar for a different game is CSGO Exchange. For Fortnite, there currently does not exist an alternative to FortNotification, meaning it has the potential for marketing, advertising and real-world usage.

Introduction

Fortnite is a currently popular video game, that can be played on many different machines and computers. It has a player base of *approximately 125 million people.*

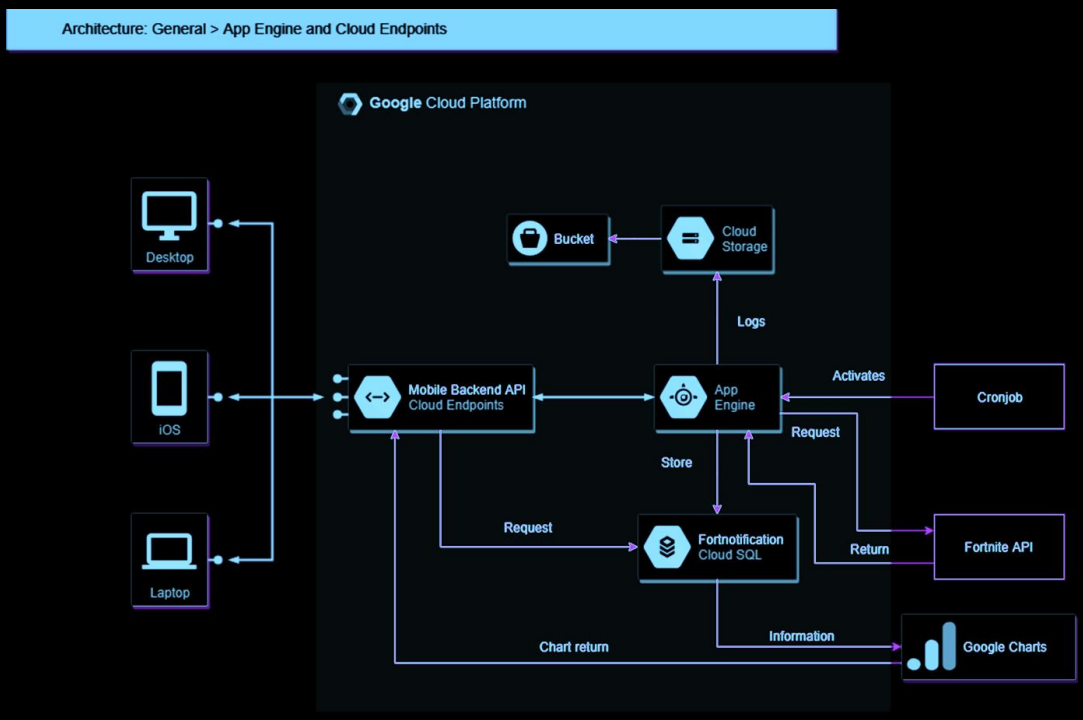
This game has an internal "Store", where players can spend virtual currency to purchase "Skins" and "Items". These skins and items are rotated through the store on a 24-hour basis, and no two days will have the same items. There are currently nearly *400 items* in the game that will be sold on the store, and as such it can take quite some time for certain items to arrive on the daily rotation.

This leads to an issue. Players want a specific item. Players will have to log into the game *every single day* for an undetermined amount of time to purchase that specific item.

Here's where FortNotification comes in. A user will log in with Google Authentication, list the few items they wish to purchase, and each day as the store rotates, if it contains one of the items that the user wants, they will receive an email ASAP describing the items in the store.

This means users can rest easy knowing they will be able to purchase the item they want, and *not have to log into the game* every single day to check.

Architecture



The website is hosted on Google Apps Engine, using the SQL API, Google Storage API, Charts API, User Service API, and the Fortnite API. Most of these services are backend, and the user will not notice, aside from google charts.

Technology Stack

HTML5/CSS/Javascript:

We chose to use the base versions of HTML, CSS and Javascript for the front end of our system as we wanted to keep the system as lightweight as possible. Because we didn't have to load a lot of Javascript or CSS libraries our website loads very quickly which makes is a huge advantage for users with a slow connection. Furthermore we did not lose out on achieving any of our desired functionality by not adding these libraries and as such for this particular project we viewed this front end language / framework stack as optimal

PHP:

We chose to use PHP over other backend languages like Python and Node primarily due to familiarity with the language. Both developers on our team had worked with PHP in the past and as such it would reduce the amount of development resources needed to achieve our goals. Furthermore PHP was entirely compatible with all the frameworks / API's we wanted to work with and as such there were no features our choice of backend language prevented us from utilizing. An added bonus of using PHP was it made it much easier to learn and understand all the API's we ended up using to power our application.

SQL:

We chose SQL over other backend languages (Such as Mongo or other NoSQL variants) as we viewed it as the easier to manage and maintain technology. As such we were happy to lose out on a little bit of speed, in our queries in exchange for the added readability and maintainability of our users data. Furthermore due to most of the queries where a join is involved not happening due to user input, the loss of performance would never be noticed by the end user and as such would not be a future scaling issue.

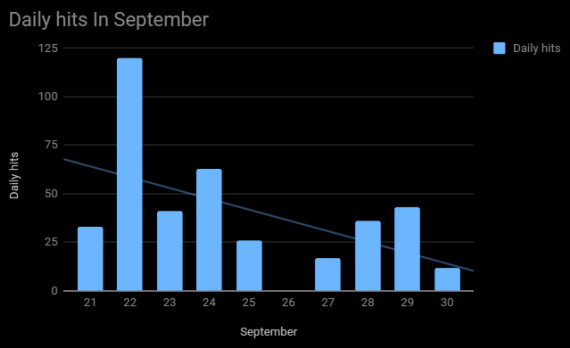
API Use

For the majority of our API use, we are looking towards the future. Fortnite has upwards of 100 million players; if our website to capture *even 0.1%* of this player base, it would be *100,000* users. A such, these extreme figures must be taken into consideration. Processing data locally with 100,000 separate entries, and possible thousands of accesses a day, would not scale well.

Storage API:

We use the google storage API to log the user's movements on the website. Files are sorted by year, month, day, and then logged. The information is login time, with a unique google user ID, and the user's email. Once a user has submitted items to be requested, another logging event occurs, this time documenting the items requested, the User ID, and email.

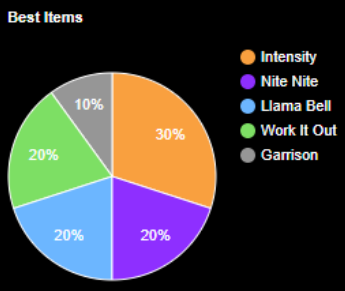
This usage is important to track statistics about website usage, and to confirm the availability of certain items. For example, we will run an algorithm that can create a histogram of website usage to confirm the popularity and direction of usage. Using this we can determine whether we will need to prepare for more scaling or start slowing down production, saving money.



You can see the trendline going down as we slow down production of the website, less website resetting and less hits, however once in full production, these numbers will likely have a *positive trend line.*

Charts API:

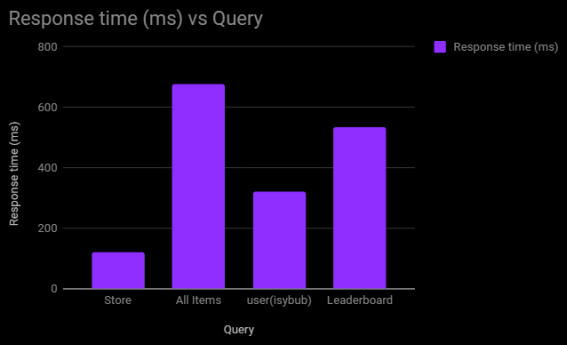
Google Charts API is one of the most powerful online chart creators, simple to execute and extremely fast. We have done several tests during the production and development of the website, we found there is *no noticeable time* between page load, and chart load. This is partially due to the hosting on App Engine, and the *high availability* of all google servers, in all locations. Here we have an example chart from the website, which shows the current most popularly requested items. This is useful for new users to see what other players enjoy and are looking for; New sign-ups will appreciate the reminder of a few popular items they may have forgotten about



Fortnite API:

The core service in this assignment, the Fortnite API. We use this API to access the daily Fortnite store, to compare with our user's requests, before sending out emails. We have completed several tests against the various functions of this API to ensure the performance of it. The graph here displays 4 different tests:

1. *The only function we require, daily store items*
2. *The most intensive function on the API, listing all current Store items*
3. *Requesting specific user statistics*
4. *Requesting "Leaderboard"*

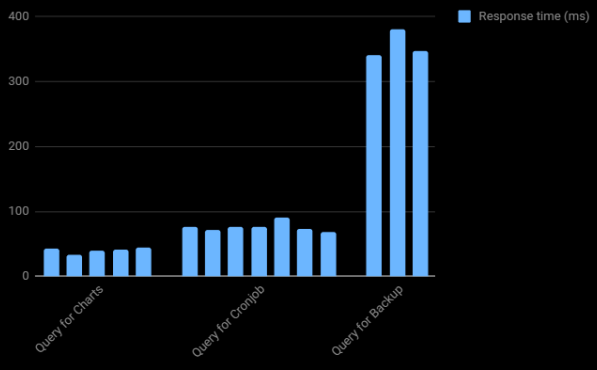


User Service API:

Login. It's one of the most important aspects to any website that has any form of service. Google's user service API makes this one of the easiest functions to implement. It is simply a few lines of Google's code, and there you have it. User information, easily accessible, and a login button that is *easy and simple to use*. The login through Google API also provides users with a sense of security as the login system is made by a brand they know they can trust.

SQL API

As mentioned before, there is the possibility of an extreme count of users, it is a clear option then, to use either Hadoop MapReduce, Google BigData, or the google SQL API. We decided due to familiarity to continue with the SQL API, as this would be quick to develop for. In terms of scaling for the future, the difference between Google's SQL API, and Google's BigData for this simple database was minimal, and as such a transfer of data and usage would be easily computed. We duplicated our data in the SQL database and ran a few of our commands on it, here are the results with *200,000 data entries*. The backup query takes some time to run, this is fine as it is not a forward-facing query, as such, it does not affect performance for users.



What's so cool about all this?

From the user's point of view, the process is extremely laidback and low on interaction. All they need to do is login with google, which can be one click if they are already using a browser they are logged in with. After this, they type the list of items they wish to be notified of, and that's it. An extremely simple two step process will give us all the information we need to complete our side of the transaction.

Speaking of our side of the transaction, how does that work?

Once the user has submitted their data, we transform it into something we can easily insert into the database. Once stored, this is the end of the submit process; After this, the cronjob is the next action to occur server side, based on the user's submit action.
This action is run *every day* a few moments after the Fortnite store has been updated, quite a few things are going on during this cronjob. First, we set up a new connection with the Fortnite api, check that it is working, and send a post request to get the daily store. Once this has been returned to the *PHP*, we fill a variable with the json response, and then decode it. Following this, we tokenize the decoded response, for individual item processing, and store it in an array of strings. We then select each user from the database, and store their items in a *PHP* variable, comparing the two arrays now, we create a third array, that is *the matches* between the store items and the user's requested items. We begin crafting the string that will be sent in the email to each user with matches.

```
$msg = "Hello! The fortnite store currently has an item that you want!";
$msg .= "\nYou were looking for:\n";

$tempDat = $db->prepare("select name FROM items WHERE uid = \"".$user['id']."\"");
$tempDat->execute();
$data = $tempDat->fetchAll();

$str = "";
foreach ($data as $row) {
    $str .= $row['name'].", ";
}
$str = substr($str, 0, -2);
$msg .= $str;
$msg .= "\n\nThe item store has:\n";
```

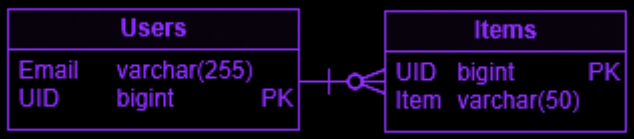
After a few more things get appended to the string, the email is sent out with the *PHP mail*() function. This is the general transaction from our side, there are a few more pages that are useful but not necessarily important to speak about. We have condensed the website down from what used to be about 1,000 lines of code, to about 500 lines.

How does our database work?

SQL Example tables

Users	
Email	UID
Isybub6@gmail.com	106148742696891494647
example@gmail.com	113916595269314910627
eggy@gmail.com	102394564853386161998
fortnite@gmail.com	108255305514332138299

Items	
UID	Item
106148742696891494647	Llama Bell
106148742696891494647	Fresh
113916595269314910627	Pumpernickel
113916595269314910627	Llama Bell



Due to the light database design, we have decided it is unnecessary to use BigData or MapReduce for such a simple layout. The performance of the SQL engine will be monitored, however; If we see any significant changes in response speed, we will easily and quickly move the database over to BigData.

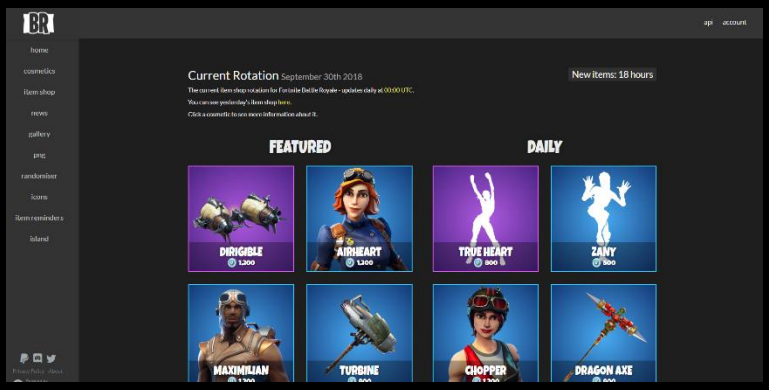
Google's User Service API contains a fantastic feature, the Unique Google ID. That is, every user that has a google account, has a unique, permanent ID tied to their account. This is what we use for the primary keys in both tables, and is a unique way of identifying users whilst avoiding sequential user IDs.

```php
use google\appengine\api\users\User;
use google\appengine\api\users\Userservice;
$user = UserService::getCurrentUser();
$uid = $user->getUserId();
$uem = $user->getEmail();

include "db.php";

$tempDat = $db->prepare("select name FROM items WHERE uid = \"".$uid."\"");
$tempDat->execute();
$data = $tempDat->fetchAll();
```

This snippet of code will get the current user, put their email and user id into variables, and then run an SQL query on it, using the User Id as a key to find the information that we are looking for; The items that a user has requested in the database.

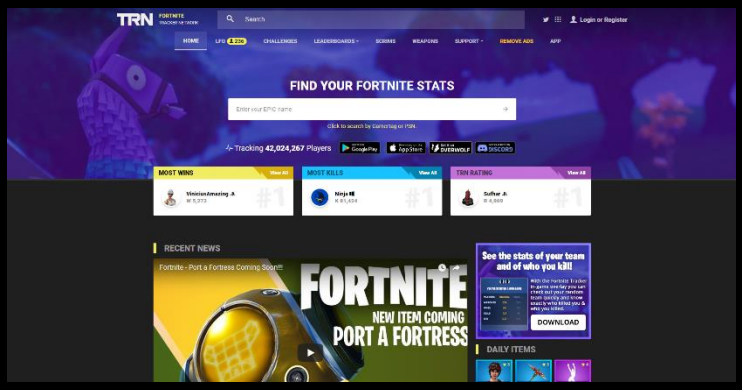
The first 5 lines are littered throughout the codebase, and the google User ID from the API is used extensively within the website, for different functions requiring a uniquely identified user. This function has dramatically reduced the necessary code for identifying users.

Related work



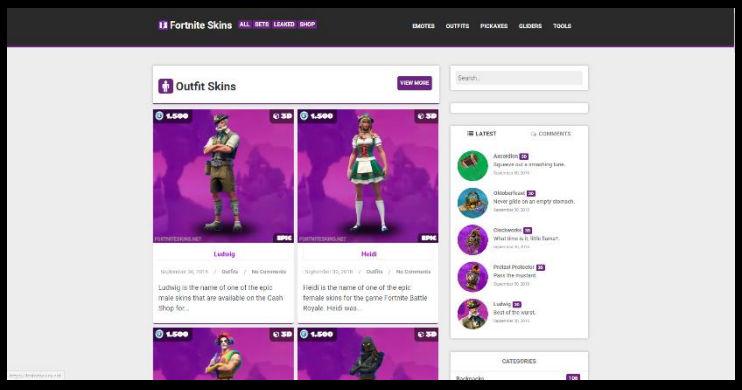
https://fnbr.co/shop/

A hub to view the store items online rather than logging into the game. This still takes time, and a good memory to check each day. Lacking a notification feature.



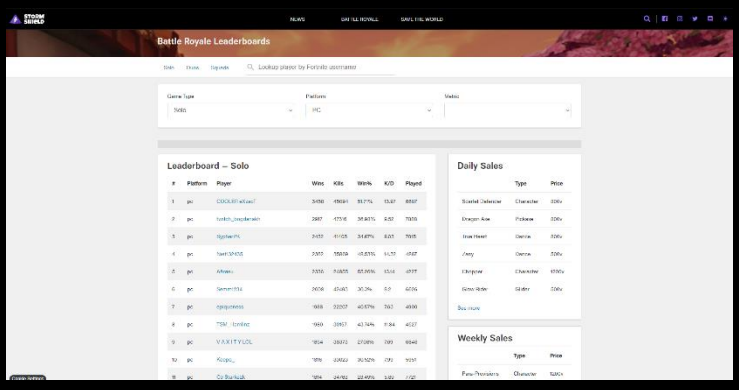
https://fortnitetracker.com/

Using a very similar Fortnite API, players can get their stats of the last season and compare with other players. Similarly lacking a notification feature.



https://fortniteskins.net

Also using Fortnite API, listing all of the items that have ever been available in the store ,and the ones in the store right now.



https://www.stormshield.one/pvp

Fortnite Tracker clone – slightly less favourable website design, however simpler to look at with less on the screen at once.

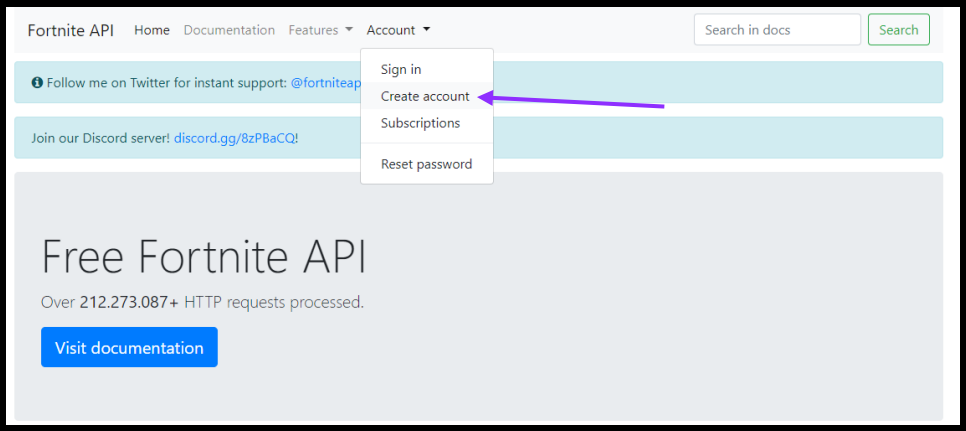
Step 1.

Clone the repository.

```
Isaac@isaac MINGW64 ~/Desktop/Filing Cabinet
$ git clone https://github.com/isybub/fortnotification.git
Cloning into 'fortnotification'...
remote: Enumerating objects: 2986, done.
remote: Counting objects: 100% (2986/2986), done.
remote: Compressing objects: 100% (2127/2127), done.
remote: Total 2986 (delta 525), reused 2967 (delta 524), pack-reused 0
Receiving objects: 100% (2986/2986), 9.98 MiB | 320.00 KiB/s, done.
Resolving deltas: 100% (525/525), done.
Checking out files: 100% (2539/2539), done.

Isaac@isaac MINGW64 ~/Desktop/Filing Cabinet
$
```

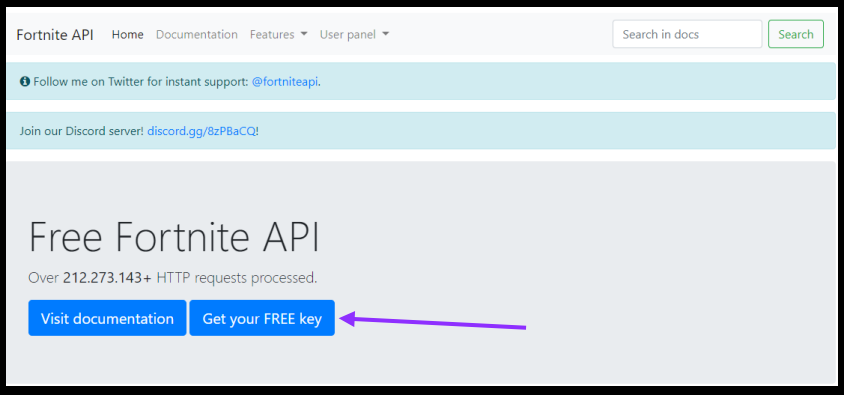
Step 2.

Create an account on https://fortniteapi.com



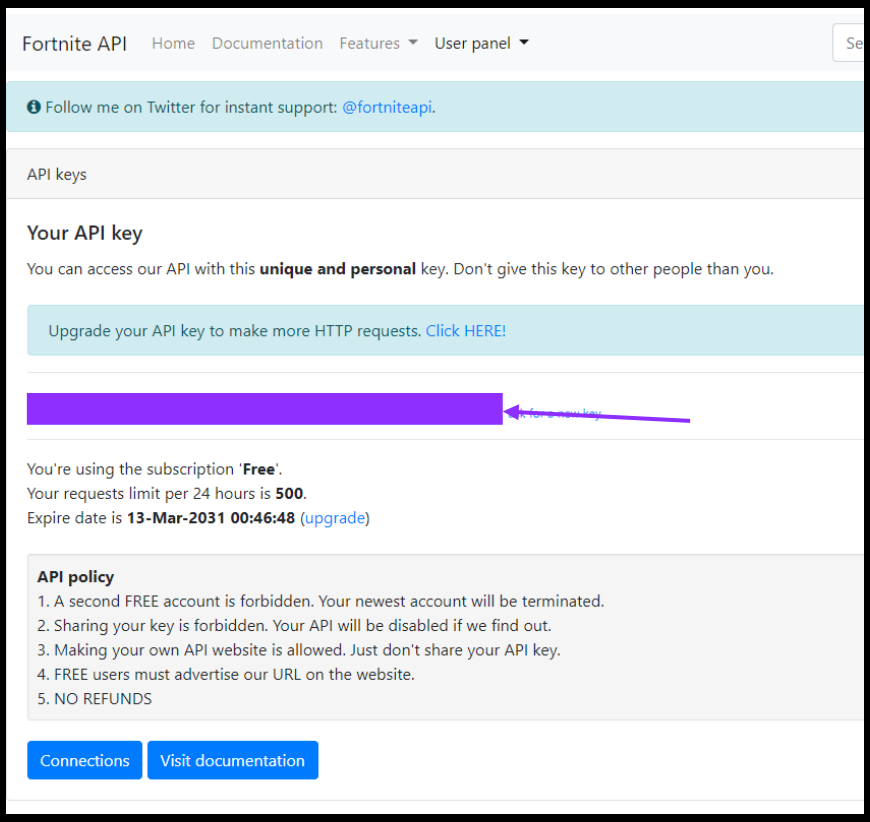
Step 3.

Once logged in, “Get your FREE key”.



Step 4.

Copy your new key to clipboard.



Step 5.

Paste into the setKey function in your local files.

```
$api = new FortniteClient;
$api->setKey("                              ");
$store = getStore($api);
function getStore($api){
    $return = json_decode($api->httpCall('store/get', ['language' => 'en']),true);

    if(isset($return->error))
    {
        return $return->errorMessage;
    }
    else
    {
        return $return;
    }
}
```

Step 6.

Follow the Cloud App Deployment PDF tutorial from RMIT'S Cloud Computing class, tutorial #3.

https://rmit.instructure.com/courses/17760/files/4498035/download?wrap=1

Note. Make sure you call the instance name "FortNotification".

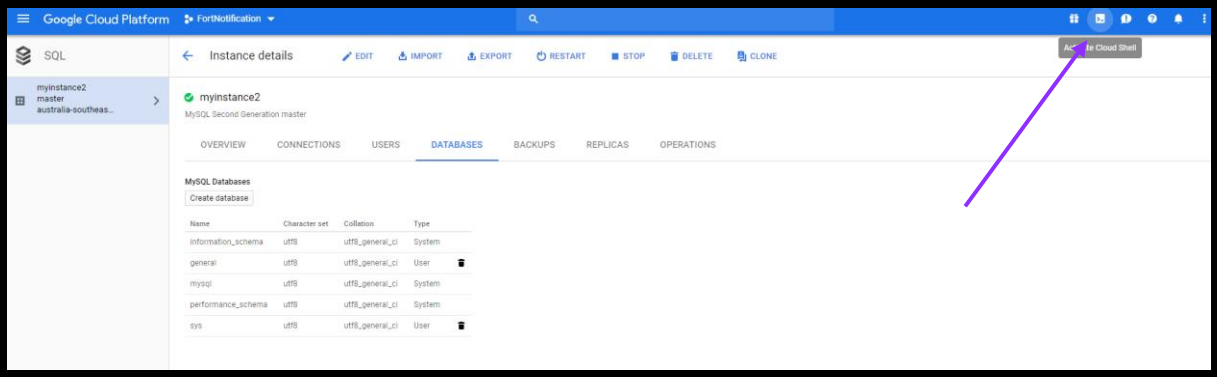
Step 7.

Follow Google Cloud's documentation to create an SQL instance on your app engine.
https://cloud.google.com/sql/docs/mysql/create-instance

Note. Ensure you name the instance what the app.yaml has been referenced to. In this case "myinstance2". Also name the password "myinstance2" as per app.yaml.

Step 8.

Open Cloud shell



Step 9.

Connect to your newly created instance on cloud shell.

```
isybub6@cloudshell:~ (fortnotification)$ gcloud sql connect myinstance2 --user=root
Whitelisting your IP for incoming connection for 5 minutes...⠹
Whitelisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 93941
Server version: 5.7.14-google-log (Google)

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Step 10.

Create new database “general”.

```
MySQL [(none)]> CREATE DATABASE general;
Query OK, 1 row affected (0.17 sec)
```

Step 11.

Enter that database.

```
MySQL [(none)]> USE general;
Database changed
```

Step 12.

Create the users table.

```
MySQL [general]> CREATE TABLE users (email VARCHAR(255), id bigint);
Query OK, 0 rows affected (0.19 sec)
```

Step 13.

Create the items table.

```
MySQL [general]> CREATE TABLE items (name VARCHAR(255), uid bigint);
Query OK, 0 rows affected (0.21 sec)
```

Step 14.

Close everything

```
MySQL [general]> Ctrl-C -- exit!
Aborted
isybub6@cloudshell:~ (fortnotification)$
```

User Manual

Alright! Once all of this has been completed, you, and your users are ready to sign up. Simply open your web browser and navigate to https://www.fortnotification.appspot.com

1. *Click on the "Sign in or register" Button.*
2. *Click "Go to my subscribed items!".*
3. *Enter a list of items you wish to be notified of, for example: "Fresh, Work It Out".*
4. *Click submit.*

And you're all done!

FortNotification will now send you an email when the item store has the items you asked to be notified of. It's that simple.

If you wish to change the items at any point, follow these exact same instructions, and the items you have already requested will be shown, and you can add, remove or keep the same items.

References

1. https://cloud.google.com/sql/docs/
2. https://cloud.google.com/appengine/docs/php/
3. https://cloud.google.com/appengine/docs/standard/php/googlestorage/
4. https://cloud.google.com/appengine/docs/flexible/php/using-cloud-storage
5. https://cloud.google.com/php/getting-started/using-cloud-sql-with-mysql
6. https://cloud.google.com/sql/docs/mysql/quickstart
7. https://cloud.google.com/appengine/docs/standard/php/mail/
8. https://cloud.google.com/appengine/docs/standard/python/config/cronref
9. https://cloud.google.com/storage/docs/
10. https://cloud.google.com/appengine/docs/standard/php/googlestorage/advanced
11. https://cloud.google.com/php/getting-started/using-cloud-storage
12. https://cloud.google.com/pubsub/docs/?hl=en_GB
13. https://cloud.google.com/storage/docs/access-public-data
14. https://cloud.google.com/storage/docs/access-control/lists
15. https://googleapis.github.io/google-cloud-php/#/docs/google-cloud/v0.36.0/storage/bucket
16. https://fortniteapi.com