CE-4020-004

Team Omicron

Milestone 4 Cycle Report

Submitted April 7th, 2019

# Executive Summary

LogiSteps is a full stack application that is designed to collect, process, and display user fitness data in a seamless, self-powered construct. LogiSteps allows a user to pair their Bluetooth enabled smart sole with their mobile device and stream data to the cloud in a manner that is unobtrusive and relies very little on the user. Prior to beginning spring quarter, four milestones were set, with the previous being completed late due to miniaturization of design. The completion of this fourth milestone, and hence, fouth development cycle, symbolizes the ability to store user data in a Google Cloud SQL instance, as well as host the web application in a Google Cloud Compute Engine. With this added functionality, the Logisteps server (web and mobile endpoints) and its statically hosted files are accessible to any user in the world. Using Google Cloud’s available resources, the app can be scaled up to meet demand.

# Introduction

The Logisteps project has completed its fifth week of development in the final quarter of development. The major goals of this quarter are completion of system integration, which includes physical construct to sensors, sensors to microcontroller (both data and power), microcontroller to Android application, Android application to web server, and web server to user. Logisteps initially began the quarter on track, but fell slightly behind in week 3, and has since then adjusted and recovered lost time that occurred in week 3. The main focus for the project during this development cycle (week 4 – week 6) the completion of full stack data transmission, and world-wide online interaction. Full stack communication is tracked as a separate milestone and won’t be documented in this milestone. A separate milestone document will be prepared for documentation of milestone 3.

The remainder of this document will go into further detail regarding risks to the project, as well as possible contingency plans that are ready to be implemented if needed, overall performance of the team regarding project plan, the deliverables created for this fourth development cycle, updates to development and planning, followed by a final conclusion. The completion of this fourth milestone marks a significant point in Logisteps progress, as it makes the product usable in a production environment.

# Discussion

The completion of this development cycle caps off a major deliverable required for usability of the Logisteps system (for end users) – ability to access resources served on a publicly accessible server. During this development cycle, progress has been made in full-stack data transmission, but this functionality will be documented in a separate milestone report. Further discussion will describe the process of configuring the Logisteps web technology for deployment on Google virtual machines.

## Google Cloud Hosted Technologies

### Google Cloud SQL Instance

In order to save user data, including user-specific details and step data being streamed from a remote device, a database technology must be established to put the data into long-term storage that is easily accessible for consumption by technologies such as web servers and mobile applications. In the previous quarters, research determined that a SQL database would be best for storing user data, and the data models were designed and implemented for a local development environment. A local web server would then communicate with the local database to retrieve user data and save step data. While this worked for development purposes, it does not meet the requirements for the Logisteps production environment. If the database was not hosted online, each end user would be required to setup their own SQL server, and expose it publicly to allow for mobile application communication. This user experience would be unacceptable, considering the target audience; to adequately satisfy the requirements of the end user, Logisteps needed to deploy and manage its own cloud instances of the database technology.

In the fall quarter, exploratory research was conducted which concluded that the Google Cloud Platform would provide the functionality needed by the Logisteps application, while also providing an abundant number of additional resources and advantages, such as free tier development, scalability, security, etc. Having already decided the technology for hosting the Logisteps application online, the first step was to make the database available online for use by the server written using the Django framework. The database is a dependency of the web application, so the database needed to be deployed *prior*to the deployment of the web application.

Moving the database to the Google Cloud Platform was a relatively simple process, and the following steps describe the process of doing so.

#### Creation of the SQL Instance

The first step in making the database online was the creation of a Google Cloud SQL instance. This was done by navigating to <https://console.cloud.google.com/sql/instances> and clicking the “Create Instance” button, as shown in figure 1 below.

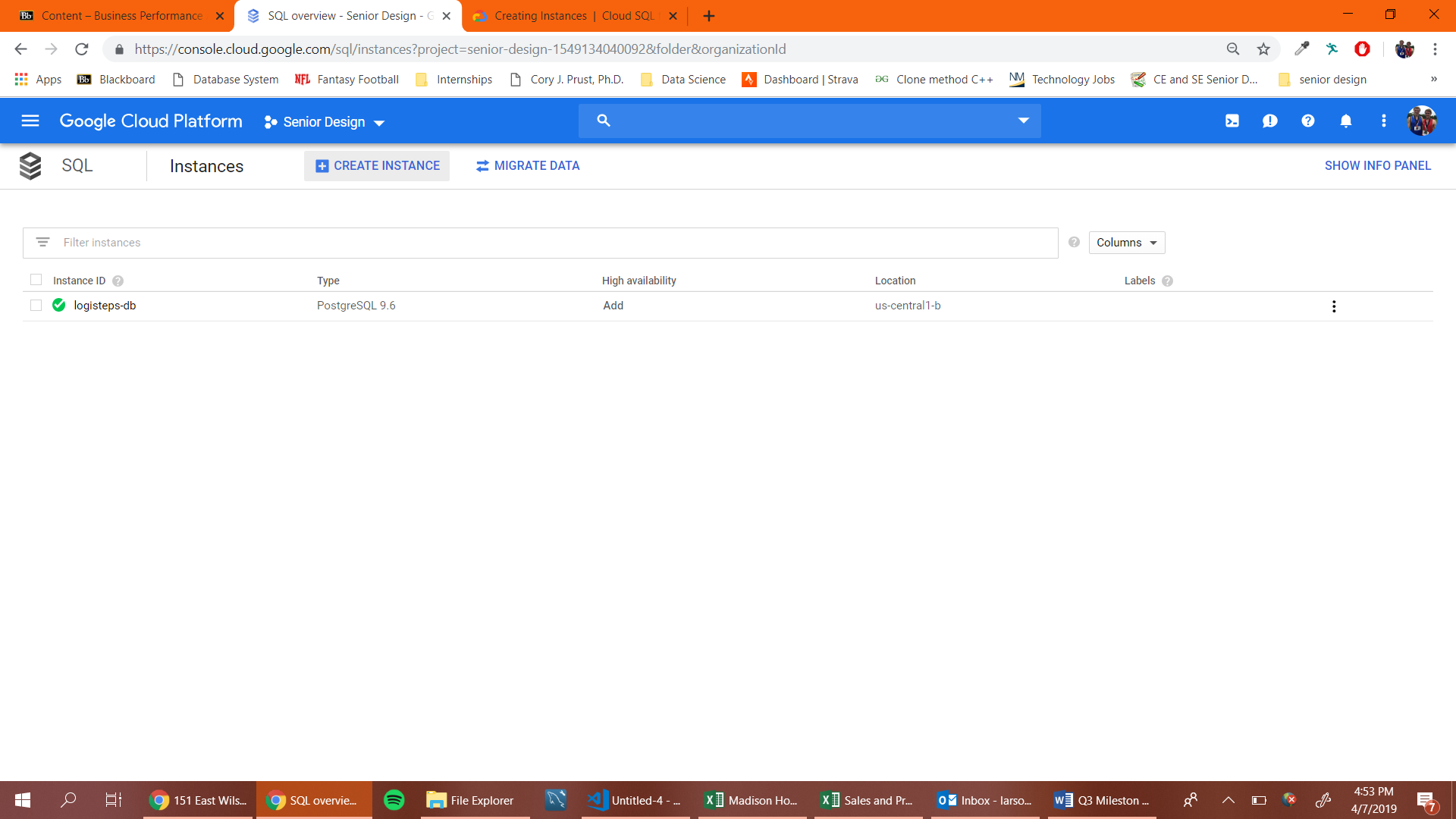




Figure - Creation of GCP SQL Instance.

After clicking the create instance button, a screen asks whether the database should be a MySQL or PostgreSQL instance. To work with timescaleDB, which provides a scalable SQL implementation, the PostgreSQL engine was chosen. Next, the configuration screen shown in figure 2 is presented.

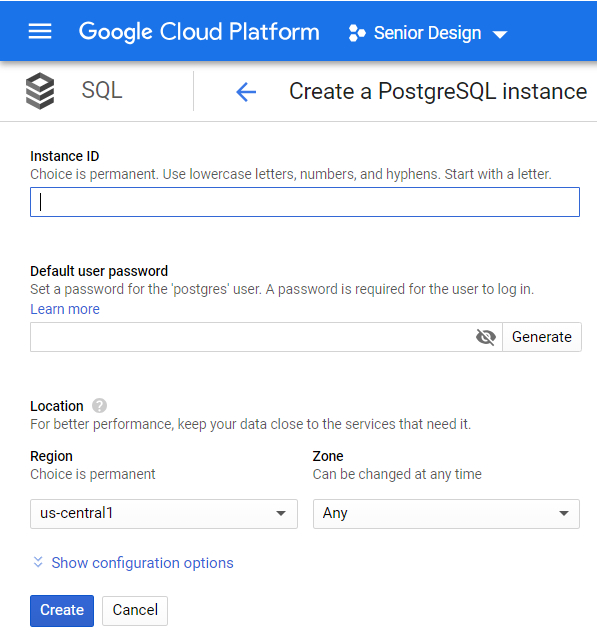


Figure - GCP PostgreSQL configuration.

For the Logisteps application, an instance ID of Logisteps\_db was chosen, a secret password was created, and the rest of the options were left as their default values. Using a “us-central1” region ensures that the database is hosted in a geographical area near the Midwest, ensuring fast connection speeds.

After clicking the “create” button, creation of the PostgreSQL instance began, and completed after a couple minutes. At this point, an empty database was created, and still needed a database and user instance. To create a user for the Logisteps application, a user was created by navigating to <https://console.cloud.google.com/sql/instances/logisteps-db/users> and clicking the “create user account” button. As shown in figure 3, a username and password were required, and a user with a username of “logisteps” was created.

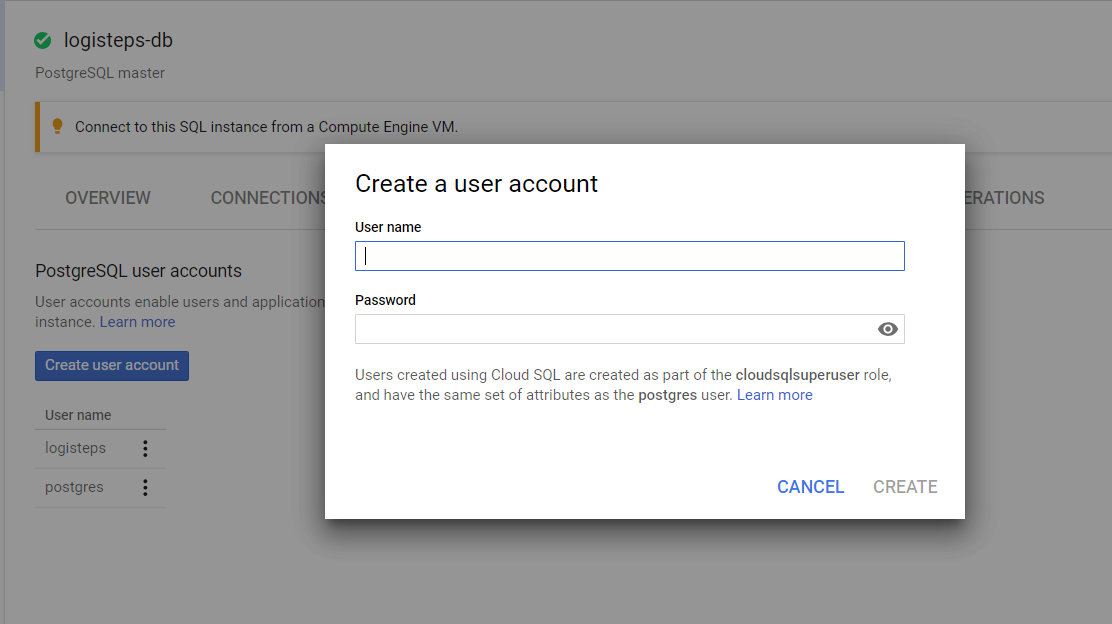


Figure - Creation of a GCP SQL user.

The next step required the creation of a database other than the default database created when the PostgreSQL instance was created. A new database was created by navigating to <https://console.cloud.google.com/sql/instances/logisteps-db/databases> and clicking the “create database” button. A configuration screen, as shown in figure 4, asked for the name of the new database, and the name Logisteps was used.

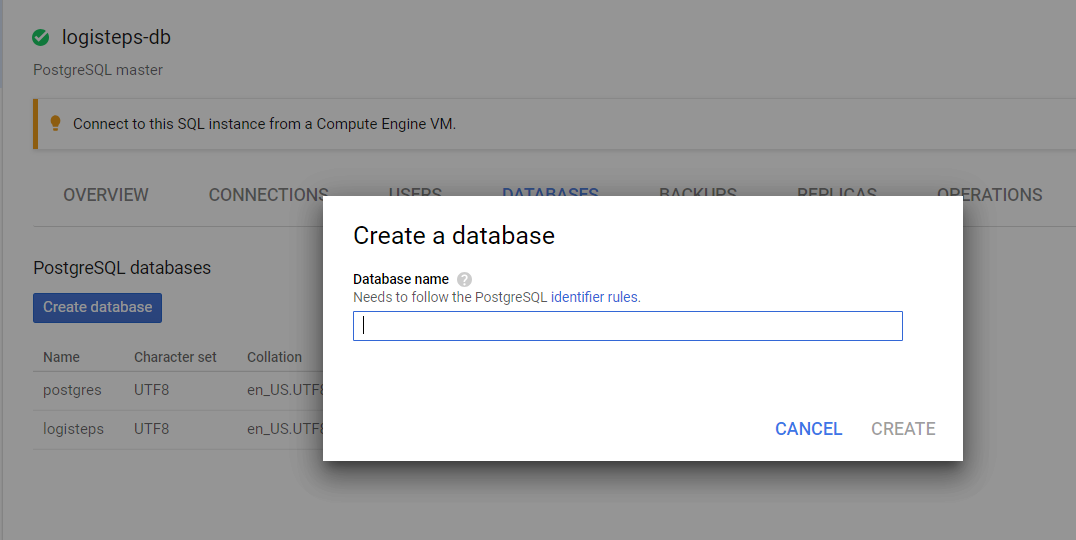


Figure - Creation of a CGP SQL database.

At this point, a GCP PostgreSQL instance was create, with a Logisteps user and database. The last remaining step was to create the database tables required by the Logisteps application. Django has a feature which creates database schema based off the Django data models, which was used to create the required database tables. This step required migrating the locally run Django web application from a local PostgreSQL instance to the cloud instance. Doing this required the use of a Google tool known as the Google SQL Proxy. This tool makes it possible to connect a locally running application to a GCP SQL instance.

The first step was enabling the API required by the Cloud Proxy. This was done by navigating to <https://console.cloud.google.com/flows/enableapi?apiid=sqladmin&redirect=https://console.cloud.google.com&_ga=2.247154700.-982376191.1548382701> and selecting the correct Google project. Upon successful enablement of the API, the message shown in figure 5 was shown.

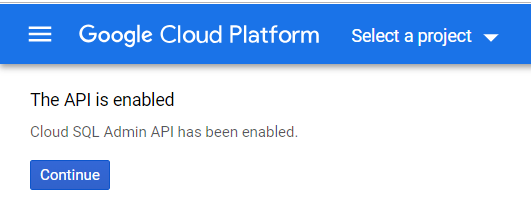


Figure - Cloud SQL Admin API enablement.

Once the API had been enabled, the Cloud Proxy was downloaded from <https://dl.google.com/cloudsql/cloud_sql_proxy_x64.exe> and moved to the root of the web application. Using the Google Cloud SDK Shell, the proxy was ran using the following command.

./cloud\_sql\_proxy -instances=senior-design-1549134040092:us-central1:logisteps-db=tcp:3307

At this point, the Cloud Proxy was serving the cloud database on local port 5432, making it possible to run the web server locally, while maintaining a connection to the cloud instance. This made it possible to run Django migrations, which configures the database according to the Django data models.

The last step required prior to running Django migrations was configuring the web servers settings.py file to properly connect and authenticate with the Cloud Proxy. To do so, the DATABASES object was changed to the following:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'HOST': '127.0.0.1',

'NAME': 'logisteps',

'USER': 'logisteps',

'PASSWORD': '●●●●●●●●',

'PORT': '3307'

}

}

At this point, the web application could be run locally with a connection to the cloud database. The next step was running database migrations. To do this, the following commands were run in the root of the web server project.

py /manage.py makemigrations

py /manage.py migrate

Following successful execution of these commands, the GCP SQL instance was properly configured for use by Logisteps applications.

Currently, the Logisteps database is running at 35.224.157.85 with 3.75 GB of memory and 10 GB of SSD.

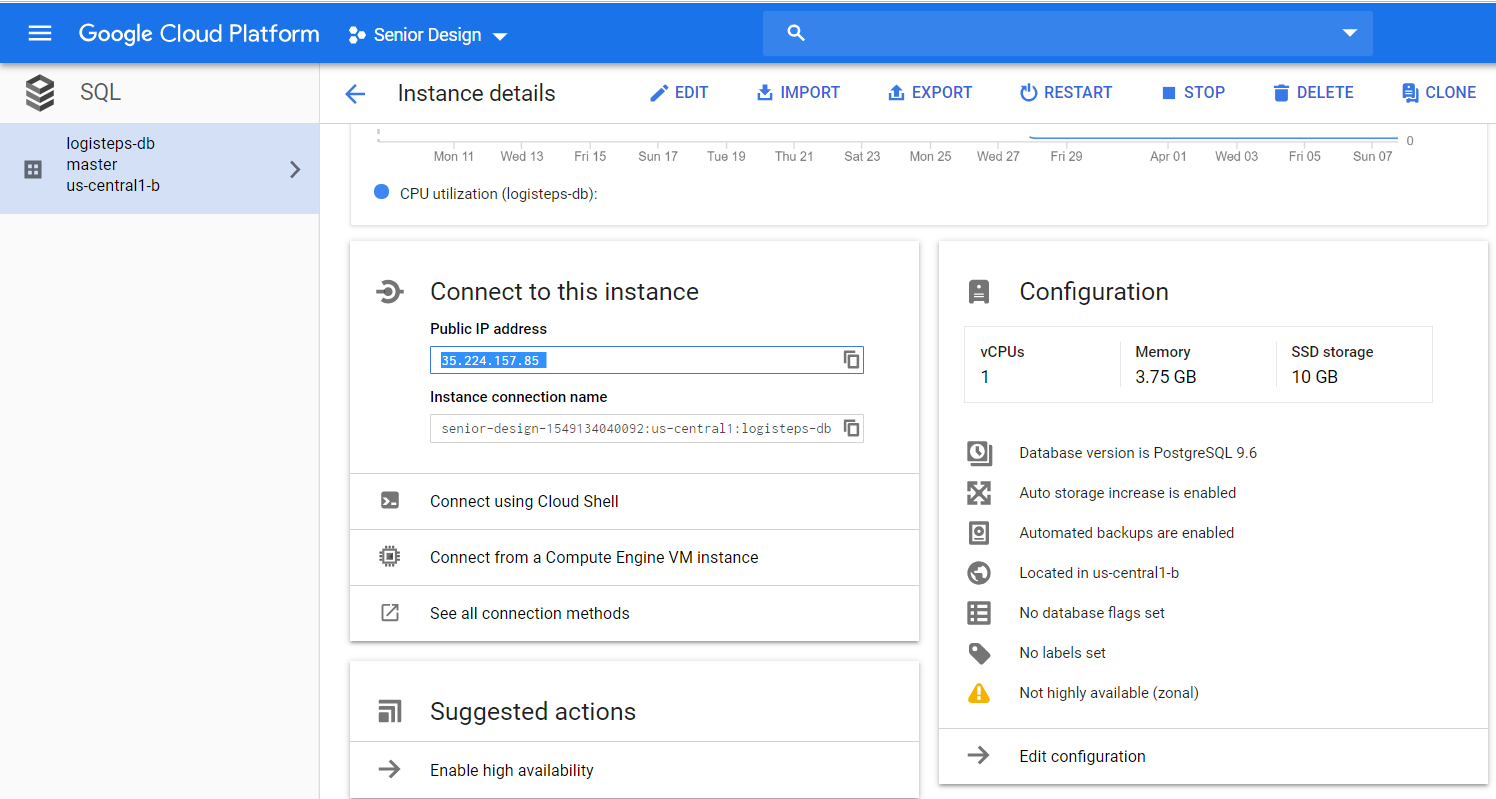


Figure - Overview of CGP SQL instance.

### Google Cloud Compute Engine

After the Logisteps database was successfully deployed to the Google Cloud infrastructure, the next step was to deploy the rest of the Logisteps web technology. This step was a much larger task and involved a significantly larger number of configuration steps and configuration files. The following sub-sections describe the process followed when deploying the web application and its related features.

The technology used to host the web application was the Google Cloud Compute Engine. The compute engine is a Google Cloud technology that allows execution of custom software in a virtual machine. The virtual machine is configured to only operate while actively serving requests and go to sleep when no longer needed to reduce power consumption and cost.

#### Configuration Files

The first step in deploying the Logisteps web technologies was creating the configuration files required by the Google Cloud Platform. The configuration files help the Google VMs determine the proper settings to use so that the custom application is properly executed. The first file created was app.yaml. This file was placed in the root of the project and was configured as follows.

***Note: These configuration files were obtained by following the tutorial located at*** [**https://medium.com/@BennettGarner/deploying-a-django-application-to-google-app-engine-f9c91a30bd35**](https://medium.com/@BennettGarner/deploying-a-django-application-to-google-app-engine-f9c91a30bd35)**.**

# [START django\_app]

runtime: python37

handlers:

# This configures Google App Engine to serve the files in the app's

# static directory.

- url: /static

static\_dir: static/

# This handler routes all requests not caught above to the main app.

# It is required when static routes are defined, but can be omitted

# (along with the entire handlers section) when there are no static

# files defined.

- url: /.\*

script: auto

# [END django\_app]

The next step involved creating a main.py file in the root of the project. This file helps the Google App Engine start the application. The file needed to appear as the following code snippet.

from mysite.wsgi import application

# App Engine by default looks for a main.py file at the root of the app

# directory with a WSGI-compatible object called app.

# This file imports the WSGI-compatible object of the Django app,

# application from mysite/wsgi.py and renames it app so it is

# discoverable by App Engine without additional configuration.

# Alternatively, you can add a custom entrypoint field in your app.yaml:

# entrypoint: gunicorn -b :$PORT mysite.wsgi

app = application

Next, a requirements.txt file was generated to list the dependencies of the Django app. This file is needed so that the Google Compute Engine can install the software libraries used by the Logisteps application. At the time of initial deployment, the requirements.txt file was the following.

Django==2.1.7

djangorestframework==3.9.0

psycopg2==2.7.6.1

pytz==2018.6

Note: These dependencies should be updated as future releases are available.

Next, the settings.py file needed to be updated to tell the Google Compute Engine where to find the web applications static files, such as HTML, javascript, CSS, etc. To do this, a STATIC\_ROOT was added to the file, as shown in the following code snippet.

STATIC\_ROOT = 'static'

The last step in configuration file creation was verifying the contents of the /mysite/wsgi.py file. The file was verified to ensure it held the following content.

"""

WSGI config for mysite project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see

https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/

"""

import os

from django.core.wsgi import get\_wsgi\_application

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'mysite.settings')

application = get\_wsgi\_application()

Following this, all of the configuration files required by the GCP were created.

#### Collecting Static Files

During development of the Django web application, dozens of static files were created to render content for client browsers. While all static content is placed in static folders, the folders were separated among several different Django projects. The three projects created during development of the Logisteps application were the default “mysite” project, the “logisteps” project, and the “logisteps\_api” project.

Prior to deploying the web technology, all static files needed to be located in a single static folder for the Google Compute Engine to access. Django provides a convenient command for locating and gathering all static files in a single folder. To do this, the following command was run from the root of the web application project.

py /manage.py collectstatic

At this point, all static files were now located in a single static folder, located in the root of the project, as shown in figure 7.

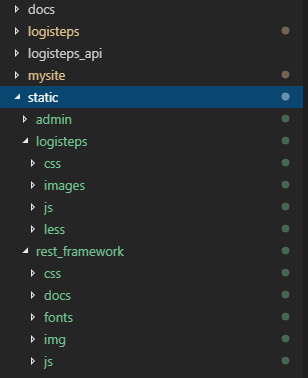


Figure - static files located in the root of the project.

#### Application Deployment

At this point, all configuration was complete, and was nearly ready for deployment. Prior to deploying the application, the DATABASES object in settings.py was changed to reference the cloud database directly, rather than referencing it through the Cloud Proxy. The following code snippet shows the configuration required for this.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'HOST': '/cloudsql/senior-design-1549134040092:us-central1:logisteps-db',

'USER': 'logisteps',

'PASSWORD': '●●●●●●●●',

'NAME': 'logisteps'

}

}

After this was done, the application was deployed using the Google Cloud SDK Shell by running the following command.

gcloud app deploy

Upon completion of this command, all of the web technology was hosted in a Google Cloud VM. This included.

* Web Server
* Mobile App REST API
* Static client files
* User data

The user web application can now be accessed from anywhere in the world by navigating to <https://senior-design-1549134040092.appspot.com/>.

## Data Encryption

By deploying the application using the Google Cloud Platform, all web traffic is automatically encrypted using HTTPS, ensuring that user data, credentials, and fitness data are protected from others.

# Risks to the project

With the recent developments in the project, including online accessibility of web technology and full stack data communication, little risks remain for the project. Most of the remaining work involves fine tuning and improvement of existing software. The only major risk that the project still faces is the possibility of damaged hardware, but steps have been taken to mitigate this risk. As of 4/5/19, additional parts were purchased for the case in which hardware needs to be repaired or replaced.

# Deliverables

The following list summarizes the deliverables completed for milestone four. This is a summarized list of all deliverables and not an exhaustive list of every single task completed.

* Google Cloud SQL Instance
* Google Cloud Compute Engine – hosting web technology.

# Development Plan / Plan Update

The completion of this milestone comes a week earlier than its initial target date. This puts the project slightly ahead of schedule. This means that additional resources will be focused on fine-tuning of the software. Other than this, no other changes have been made to the project development plan.

# Conclusion

The completion of the fourth milestone, and fourth development cycle, brings about a slight reduction in risk to the project. By enabling online access of the web technology, it simplifies the development process and creates an environment acceptable for end user usage. The coming weeks of the project will focus on fine-tuning of existing software, and completion of poster design.