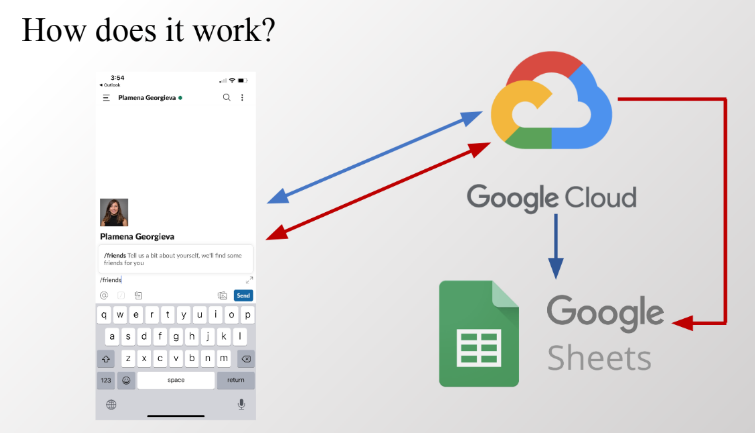
# Leveraging Cloud Technologies for Rapid Development

## Abstract

In a world with virtually free commoditized cloud computation, development of new technologies can be expedited thanks to the delegation of large aspects of the development to those off-the-shelf cloud products.

In this project, we will leverage Google’s cloud offerings to build a basic, yet functional Slack application in under a week.

## Design

* **User Interface**

We use [Slack](https://slack.com/) as a UI. This choice completely obviates the need for fullstack development. We leverage Slack’s UI builder, and its API for communicating with our Backend.

* **Backend**

We use [Google Cloud Functions](https://cloud.google.com/functions/) to implement a webapp capable of receiving http post requests, running python code, and serving responses.

There is a free option for apps that are under a certain usage limit, as well as $300 worth of free credit for the first 12 months of use.

* **Database**

As our data is simple and relational, and we use [pandas](https://pandas.pydata.org/) for data processing, a simple [google spreadsheet](https://www.google.com/sheets/about/) is enough to serve as a database. We use [gspread](https://gspread.readthedocs.io/en/latest/) to connect between any python code’s environment and our spreadsheet. This is especially useful for the development stage, as uploading our code to the Cloud Function takes 2 minutes each time. A local installation of [Anaconda](https://www.anaconda.com/), allows us to develop our code on a [Jupyter Notebook](https://jupyter.org/), which lets us test and develop our code in a streamlined fashion.

**Note - Design Flaws:**

1. In our design, each user entrypoint (the initial point of contact between the user and our app) – a [slash command](https://api.slack.com/interactivity/slash-commands), is connected to a different Cloud Function. It would have been cleaner and enabled faster deployment to have a single Cloud Function for all entrypoints, and adding an initial routing method that routes each request internally to the right method in our “server code”, rather than multiple cloud functions defined on the same codebase, but with different methods.
2. Our app has two main user flows, and one heavy computation. Of the two user flows, one happens once per user (onboarding) and the other may happen an indefinite amount of times (query), and the results may differ depending on if other users onboarded since the last usage.

In the current design, the heavy computation happens at the end of the second flow, which is unnecessary. A simple modification to the codebase would compute the results for all users after each onboarding event to our database, s.t. each query simply retrieves the pre-computed result.

## Process

* **Data Collection and Manipulation**

1. As part of a different school project, 21 students filled a [survey](https://docs.google.com/forms/d/19g60Q0D6pg0KI--M6nu0mzt-VWAv6Eb_gSIUvvvB8wg/edit) asking personal questions.
2. Those results were projected to a numeric space using pandas, the code can be revised [here](https://github.com/ct-studio-buildboard/F19-T075/blob/gabby/slack_app/DAL.py#L66).
3. A cosine similarity score was computed on the resulting vectors. The code can be revised [here](https://github.com/ct-studio-buildboard/F19-T075/blob/gabby/slack_app/DAL.py#L77).
4. The results were benchmarked using our knowledge of the subject pool, to realize that %62 of our simplistic algorithm’s friend recommendations are of actual friends in real life.

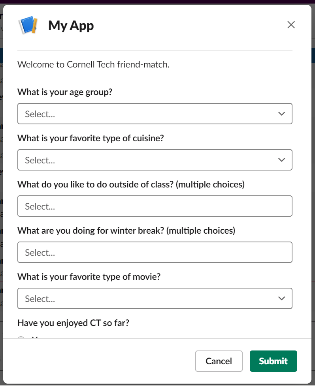
* **DevOps – Implementing the full connectivity flows**

1. To build a simple slack app with a google functions backend we followed [this](https://cloud.google.com/functions/docs/tutorials/slack) tutorial.
   1. As a code gotcha – the environment variables need to be configured. See [this q&a](https://stackoverflow.com/questions/59120476/gcp-slack-tutorial-slash-command-failed-to-deploy-google-cloud-function) if you have a problem.
2. To enable python code to connect with a google sheet, we followed [this](https://www.twilio.com/blog/2017/02/an-easy-way-to-read-and-write-to-a-google-spreadsheet-in-python.html) tutorial.
   1. Note that we used the very same cloud project from before for the credentials
   2. There was another gotcha, fixed by [this q&a](https://stackoverflow.com/questions/56084171/accessing-google-sheets-api-with-python) (needed the second scope).

* **Functionality**

1. To build a slack modal, use [this slack-provided](https://api.slack.com/tools/block-kit-builder?mode=modal) tool.
   1. There’s no tutorial on how to create a survey-like modal, but with the help of Ryan Sydnor we were able to understand that it’s crucial to use “input blocks” in order to get the user’s input when they submit.
   2. Thanks to Wenqin Tang for providing an initial modal code.
   3. Code can be revised [here](https://github.com/ct-studio-buildboard/F19-T075/blob/gabby/slack_app/views.py#L5).
2. For all other functionalities (Adding data to the database, Responding to the user, and other) see the [full codebase](https://github.com/ct-studio-buildboard/F19-T075/blob/gabby/slack_app/).

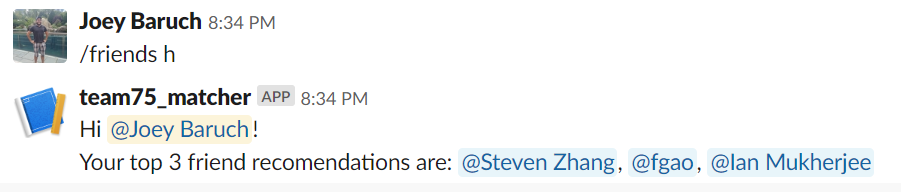
## Result

* **Onboarding**

By calling “/addme” in any channel, a survey implemented as a slack modal pops up, and the user onboards themselves by filling it and submitting.

This prompts one of our Cloud Functions to persist the data as a new row in our spreadsheet.

* **Query**

By calling “/friends” in any channel, a Cloud Function performs a [cosine similarity](https://en.wikipedia.org/wiki/Cosine_similarity) computation across all users and returns the top 3 most similar other users.

## Code

The development code may be found [here](https://github.com/ct-studio-buildboard/F19-T075/tree/gabby/slack_app), and as we work out the last few kinks in the app, it will hopefully move to the master branch [here](https://github.com/ct-studio-buildboard/F19-T075/tree/master/slack_app).

Special thanks to Ryan Sydnor, Wenqin Tang, and Gabby Facquet for their help developing this project.