Initial API Design

API Design Summary

The following document will explain the various technologies which the team will be using and the technology stack which the team had agreed upon.

For web scraping and web crawling components of the API, the team has decided upon the usage of google Cloud Run for hosting the web scraper, API and API documentation, and the usage of Google Firebase as the database for the scraped website. The language chosen to run the scraper is Python for its ease of prototyping and powerful libraries (BeautifulSoup and Scrapy) for web scraping. Several other language merits were judged and considered below.

Here we'll discuss what our API's design is. What information will be available etc.

Software Architecture

The software architecture that the team decided for this project is outlined as below and the reasoning in why we chose the following software systems.

<insert system architecture diagram here that outlines (in general) the entire system>

Web Scraper Design

Platform:

Regarding the web-scraper tech stack that the team will use, the team narrowed between two Cloud services, Amazon Web Services (AWS) and Google Cloud (GCP). GCP was ultimately chosen given three factors.

The first factor was the ease of entry, relative to AWS, GCP difficulty of starting is lower as they have better documentation (as agreed by the team) which means that the team could set up a prototype faster.

The second factor is cost. Given the team's budget, the team is required to use a free tier of these services. AWS does provide a 12-month free trial but will begin to charge as soon as the free tier is up whereas GCP will have a cap that will prevent the team from spending more than they need. On top of this, GCP also offers $300 of credit which can be used for its services.

Finally, it is familiarisation. The team has familiarity with Google services and all have Google Accounts which means we could start to develop fast. AWS would require a developer account which the team has no experience.

Google Cloud Storage was chosen over Google Firebase Realtime Database. Firebase has better capabilities when dealing with a high volume of user-generated content which our API is not. Google Cloud Storage has high scalability and high storage which can deal with the large number of queries expected from an API.

Language:

Research narrowed down the potential languages which the team could use to build their web-scraper and web-crawler is given to research into their web scraping development support. The final choice of the language is determined by three main criteria; speed, scalability, and ease of use.

C/C++ has decent support for building a basic web scraper. However, given the poor state of the CDC website, we would need good support for web crawler development as well which the language does not have good support for. Additionally, the languages take considerably longer to code in.

PHP was automatically declined given that the team has no good experience in the language and given our time frame speed of development was important.

Node.js, when compared to Python's standard library, Python's library is more complete and has more in-depth scraping libraries which would aid the team with faster development. Node.js benefit towards non-blocking event which is not great for something which operates for a long time such as scraping a website The team also has a lack of familiarity with javascript and its framework.

Python was the language that the team agreed upon for the web scraper. The members involved are familiar with the language and it has three well-known and well-supported libraries; BeautifulSoup and Scrapy being the primary ones. BeautifulSoup has allowed for high-speed scraping which is beneficial for keeping up with real-time updates.

<insert architecture diagram of web scraper>

API Design

The team considered two approaches regarding querying the database with the API and the scraper:

1) Have the API periodically scrape the website every time there is an update to the CDC website and update the database accordingly.
2) Everytime the API is called the web scraper and scrapes the requested data accordingly.

Ultimately the team chose the first approach for two reasons. The first is that if the queries are tied to the scraper, in the event that there is a large number of queries, this could result in high traffic on the CDC website themselves and incur high financial costs. The second reason is that we believe that parsing API queries into database queries would be easier in terms of translating that info rather than taking the parameters of the queries and passing them into the scraper.

The tech stack chosen for the API is NodeJS and ExpressJS deployed on Cloud Run. The reasons for Cloud Run being chosen will be explained later under the Deployment heading.

ExpressJS was chosen as the API framework due to its minimalistic nature ensuring ease of development. NodeJS was chosen due to ExpressJS being a NodeJS framework, but also for educational reasons. Considering NodeJS' popular industry usage and extensive documentation as the back-end environment for APIs the team deemed it beneficial in the long-term to learn and use it. Python's Flask library, a micro web framework, was considered, however ExpressJS' more popular usage for RESTful APIs and its ease of use caused the team to choose the latter.

<insert architecture diagram of API stack>

In terms of the API's design we will not be using authorisation methods such as HTTP Basic, JWT, API Keys or OAUTH. We concluded that the time to implement these features did not confer any noticeable benefits. We posited that if this were a production API where DDoS and high traffic usage were a real .

<insert examples of possible HTTP interactions, parameters, etc.>

<create a really cool time-based diagram>

API Documentation Website Design

In terms of the API's documentation, the team shall use Spotlight.io over SwaggerHub. The main reason for this are the following:

- Ease of use: Stoplight.io offers a very user-friendly GUI to generate documentation.
- Cooperation: Stoplight.io allows up to 5 free contributors, whereas SwaggerHub's free tier is capped to only 1.

In terms of the design of the API documentation, we will also attempt to have the API conform to the OpenAPI standard. The reason for this decision are the following benefits:

- Following an industry standard will result in a more organised and well-thought-out API.
- OpenAPI can already be easily generated by either SwaggerHub or Stoplight.io

<insert architecture diagram of API Documentation>

Deployment

The choices available for deployment were numerous, but ultimately the Google Cloud Platform (GCP) was chosen. Google Cloud Run was chosen for the Web Scraper, API, API Documentation and Website, and Google Firestore was chosen for the database.

For the Scraper, API, API Documentation and Website, Google Cloud Run was chosen over candidates such as Cloud Function, Compute Engine or AWS EC2 due to its robust offering of a VPS-like service in a serverless format. Essentially, Cloud Run offers a VM-like device similar to Amazon's EC2 with the added benefit of being serverless, meaning that the team only pays for what is needed, and additional server instances are created automatically by Cloud Run to scale upwards. This automated scaling and the fact that it's a VPS enables great flexibility in the applications we can create on the platform, and thus why the team solidified it as our final choice.

For the database, Google Firestore was chosen due to its abundance of documentation, tutorials, but also primarily because it is also part of the Google Cloud Platform, making the managing of the deployment easier.

Additionally, GCP was a more appealing choice than AWS due to its $300 free trial and its useful spending cap on some of its services, helping ensure the team doesn't incur any extraneous costs.

DOCUMENT DRAFT END

“1. Describe how you intend to develop the API module** and **provide the ability to run it in Web service mode

2. Discuss your current thinking about how parameters can be passed to your module and how results are collected. Show an example of a possible interaction. (e.g.- sample HTTP calls with URL and parameters)

3. Present and justify implementation language, development and deployment environment (e.g. Linux, Windows) and specific libraries that you plan to use.”