Project Management Information

Organisation and Responsibility

Name	zid	Major responsibility	Strengths
Abhyudit Gupta	z5196145	Front-end, Project plan	Python, React, JS, HTML,CSS
Jiahui Luo (Lacey)	z5158415	Front-end, Project plan	JavaScript, React, Python
Haoran Xu (Matthew)	z5134675	Back-end, API design	Python, Flask
Yueru Duan (Ellen)	z5210986	Back-end, API design	Python, Flask
Ayaan Adil	z5213315	Back-end, API design	Python, web-scraping, Flask, Django

Based on the overall work of the project and the strengths of every member, our team was divided into two parts basically, front-end and back-end. Three of the members are mainly focused on building the API, and two of the members are responsible for the implementation of the web application. Since the first half of the project is mainly about building API, the front-end team will do work on the report and Swagger documentation. While during the second half of the project, the back-end members will work on the report, and then we may all collaborate to help in the implementation of the web application and efficient integration of multiple API's.

Roles of Team Members in each Deliverable

Abhyudit & Lacey	Ayaan, Ellen, Mattew
Deliverable 1: ● Management information report ○ Team responsibilities and work management ○ Decision making ○ Management tools ● Design Details Report ○ Justifying language ○ Justifying deployment host ○ Software Architecture	**Deliverable 1:** ● API design ● Implementation & Justification of language, deployment and development environment. ● Web-scraper testing ● Parameter passing ● Exploring cloud functions ● Design Details report
Deliverable 2: ●	**Deliverable 2:** ●

Decision Making

During the project, decision making will be down to a vote after having a meeting to list all the alternatives and evaluate the advantages and disadvantages of every possible option. If members have conflicting decisions, we will go with the majority and further consult our mentor for what's best.

Group environments can be challenging for many reasons and if we encounter any problems during the project, we will as a team resolve the issue and try our personal best to solve it. Following may be the hurdles during the project:-

- Unresponsive API's.
- New exposure to cloud services and new frameworks.
- Development expectations and outcome.
- The poor performance of a team member or their missed deadlines.
- Communication gap with team members or our mentor.

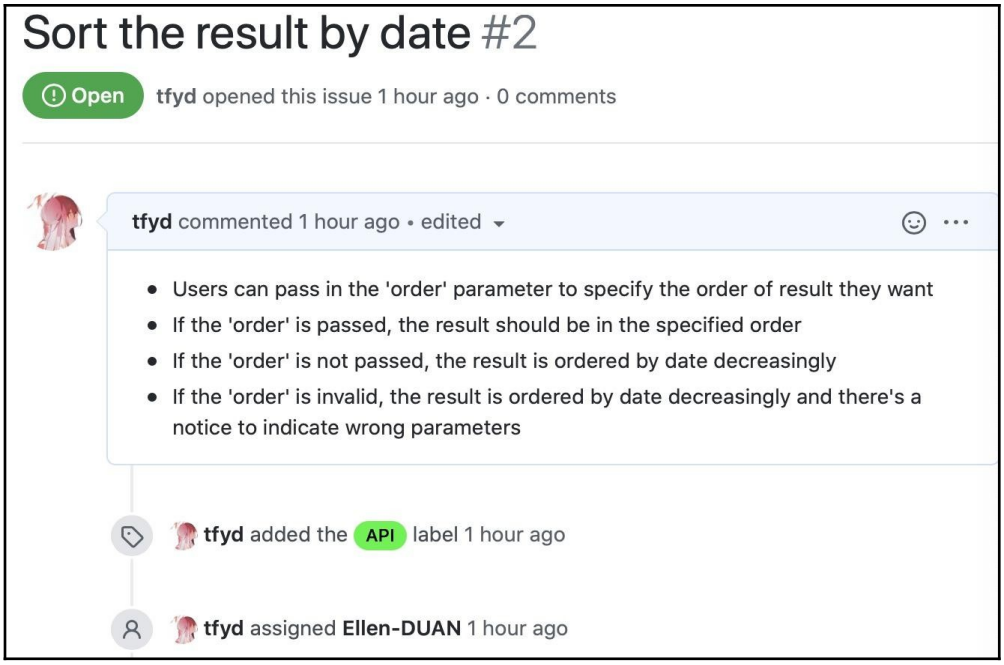
Communication

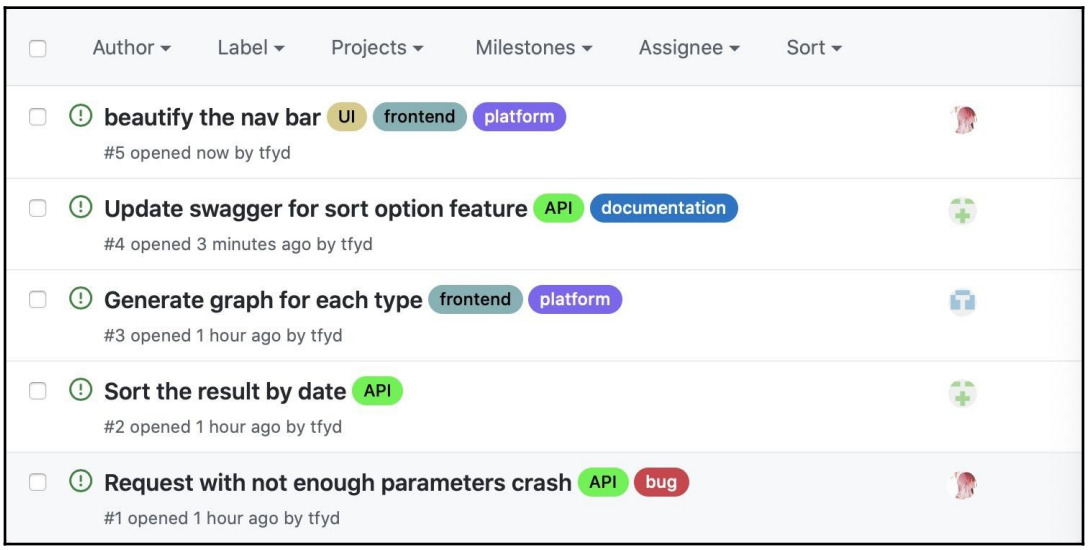
Since the members are located in different countries currently, most of the communication is conducted online.

- Weekly meetings
    - We will have at least two meetings every week using MS Teams. One of them discussing the progress and follow-up plans of the whole team with the mentor, and the other meeting talks about every member's progress and problem as well as assigning the following job if needed. The team members working on the same component may have meetings within a smaller group.
    - The most preferable meeting time is decided by doing polls on messenger group chat.
- Instant communication
    - We also have a Facebook messenger group for instant communication. If anyone encounters a problem, it is more efficient to ask in the group chat than waiting for the weekly meeting. If the issue is not solved within the group, we contact our mentor, Richard Liu.

Management Tools

- Task Management
  - The Github Issue Board is used for task control.
  - During each phase, the issues are created firstly.
  - The requirement and test criteria are written in the issue and the issue is assigned to one or more of the team members before actually starting to code.
  - When there are bugs, new issues are created for that.
  - All issues are clearly labelled.
  - The issue is closed when it has been implemented and tested.





- Code Management
  - We use Github as the tool for code management and version control. Each new feature will start from a new branch, and Github Pull Request is used to ensure the code in the master branch is the latest runnable version.

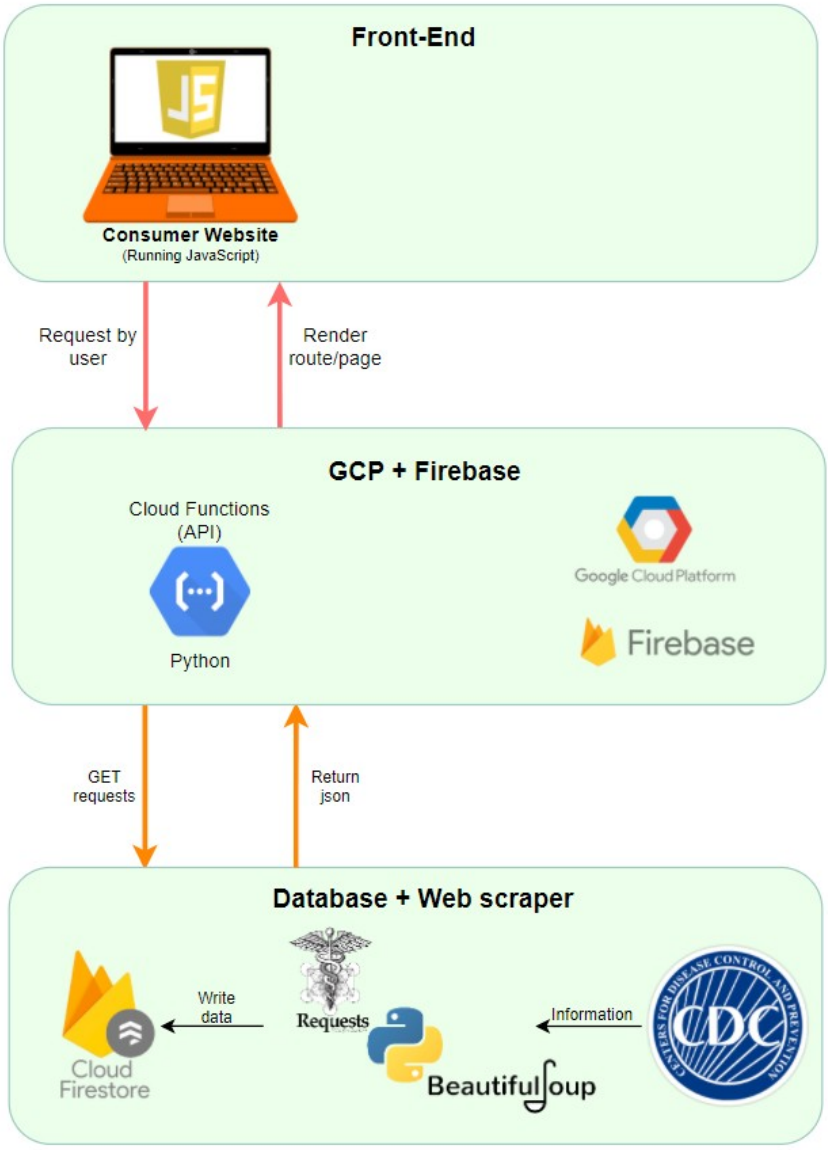
- Documentation
    - For documentation, we use google docs to collaborate with each other for the reports and other documents, such as API design. When there are conflicting ideas on the document, those are commented respectively on google docs, which are later resolved.
    - The final document will be uploaded on GitHub.

API DESIGN/DESIGN DETAILS

Development of API module and Web App

Objective :
The objective of the project is to extract information from the CDC (Centre for Disease Control) Website using a web scraper and then make it available on the web in an organised manner using an API. Once we have constructed the API we have to make a Web App that will use this API and other similar API's to present detailed information to users about disease outbreaks.

Software Architecture:
Here is a diagram illustrating the overall functioning of our Web App -



Therefore, the main components of the project include :

1) Web scraping
2) Development of API
3) Construction of Web App

Detailed information about each of the design choices mentioned below will be elaborated on in later parts of the report

Web Scraping:
The Web scraper will extract information from the CDC website on a regular basis. We will be using a combination of BeautifulSoup and the Requests library to conduct our scraping. The web scraper will be hosted as a separate cloud function. For each run the scraper will create disease reports and add them to our database.

Development of API:

The development of the API involves 3 important design decisions -

1) Hosting the API
2) Storing the information
3) Documentation of the API

Hosting the API:
Our API will be hosted using Google Cloud Functions. Hence, given the function's URL, anyone on the web can query it using any web browser.

Storing the data:
The cloud function will make calls to a firebase database (which is a real time database provided by Google) and the database will return the disease reports in the specified JSON format.
The Firebase database will be populated by the Python web scraper.

Documentation: The documentation for the API would be made using Swagger. Any user who wants to use the API can use the swagger documentation to view the various calls that they can make.

Construction of Web App:

We will be using ReactJS for the frontend of our project. The backend will be made with python. We will be using GCP to host the website. The website will use a combination of 2-3 APIs to present disease reports. The user will enter the required parameters and the website will render a page that will show various graphs and charts displaying details of the outbreak the user searched for.

Comparison of various Web Scrapers

	Scrapy	Requests	Beautiful Soup	Selenium
What is it?	Web scraping framework	Library	Library	Library
Purpose	Complete web scraping solution	Simplifies making HTTP requests	Data parser	Scriptable web browser to render javascript
Ideal use case	Development of recurring or large scale web scraping projects	Simple non-recurring web scraping tasks	Simple non-recurring web scraping tasks	Small-scale web scraping of javascript heavy websites
Built-in Data Storage Supports	JSON, JSON lines, XML, CSV	Need to develop your own	Need to develop your own	Customizable
Available selectors	JCSS & Xpath	N/A	CSS	CSS & Xpath
Asynchronous	Yes	No	No	No
Javascript support	Yes,via Splash library	N/A	No	Yes
Documentation	Excellent	Excellent	Excellent	Good
Learning curve	Easy	Very easy	Very easy	Easy
Ecosystem	Large ecosystem of developers contributing projects and support on Github and StackOverflow	Few related projects or plugins	Few related projects or plugins	Few related projects or plugins
Github stars	32,690	34,727	-	14,262

Having compared and studied the various web scrapers offered in Python. We made the decision to use **BeautifulSoup and Requests** because of the following reasons -

1) The pages that we have to scrape do not have repetitive content. The content is not placed in the same tags on different pages. Hence eliminating the need to use Scrapy.

2) BeautifulSoup has selectors that are much easier to use as compared to other scrapers.

3) The CDC website has preloaded content and hence we do not need a website that handles javascript. This eliminates the need to use Selenium.

4) In the end, BeautifulSoup is very easy to use and the team feels comfortable in experimenting with it.

Database:

For storing our data we will be using a real time firebase database provided by google. There are several reasons for choosing firebase for our project listed below -

Measures	On-Premises Database	Cloud Database / DBaaS
Reliability	Reliable, and Private.	More reliable but not necessarily Private.
Scalability	Limited scalable.	Unlimitedly scalable.
Speed	Faster, but may fail in any point of time (in care of hardware failure).	Faster and will be up always.
Deployment	Deployment takes time.	Deployed within no time.
Cost Effectiveness	Lots of capital required to setup on-premises database as a service.	Pay only for what you Use. Highly Cost effective. No Overhead cost involved.
Maintenance	High on Maintenance Cost. All cost to the company. HW, technicians, DBA's and other infrastructure.	No Maintenance Cost. Pay for what you use.
Setup Cost	Entire Setup cost is to be borne by the Company.	Entire Setup cost is borne by Vendor. The company pays only for the Service.
Security	Highly Secure and Controlled.	Highly Secured as per Vendor.

After analysing the comparison between on premise database options and cloud storage, we chose cloud storage for the following reasons -

1) Requires a huge amount of capital which we as students do not have.
2) Cloud services are free for initial use and easily expandable. Hence perfect for a student project.
3) It also serves the purpose of experimenting with a Cloud database.

All points above indicate that a cloud database would be more suitable for us -

Vendor	Pros	Cons
AWS	• Enterprise friendly services with a good rating from CIO's • An established market leader • Robust partner ecosystem • Substantial Market place with large collection software and services • Higher availability zones • AWS now spans 76 Availability Zones within 24 countries and has announced plans for nine more Availability Zones and three more AWS Regions in Indonesia, Japan, and Spain.	• No demonstrated support for Hybrid cloud Outposts is still in its nascency) • Cost management • Overwhelming options
Azure	• Integration with Microsoft tools and software • Broad feature set with a deeper knowledge of enterprise needs • Hybrid cloud support • Azure is generally available in 53 regions around the world, with plans announced for 8 additional regions. Across 140 countries	• Downtime of regions which has affected global businesses • Less flexible with non-windows server platforms • Central network connectivity and management • Incomplete management tools
GCP	• Designed for cloud-native businesses • Commitment to open source and portability • Deep discounts and flexible contracts • DevOps expertise • Google is available in 24 regions and 73 zones with plans to expand to more locations.	• No integrated backup options • Fewer features and services enterprise focus • No option for native DR replication tools

Although AWS is the market leader in providing all kinds of cloud services. We chose GCP real time database because of the following reasons -

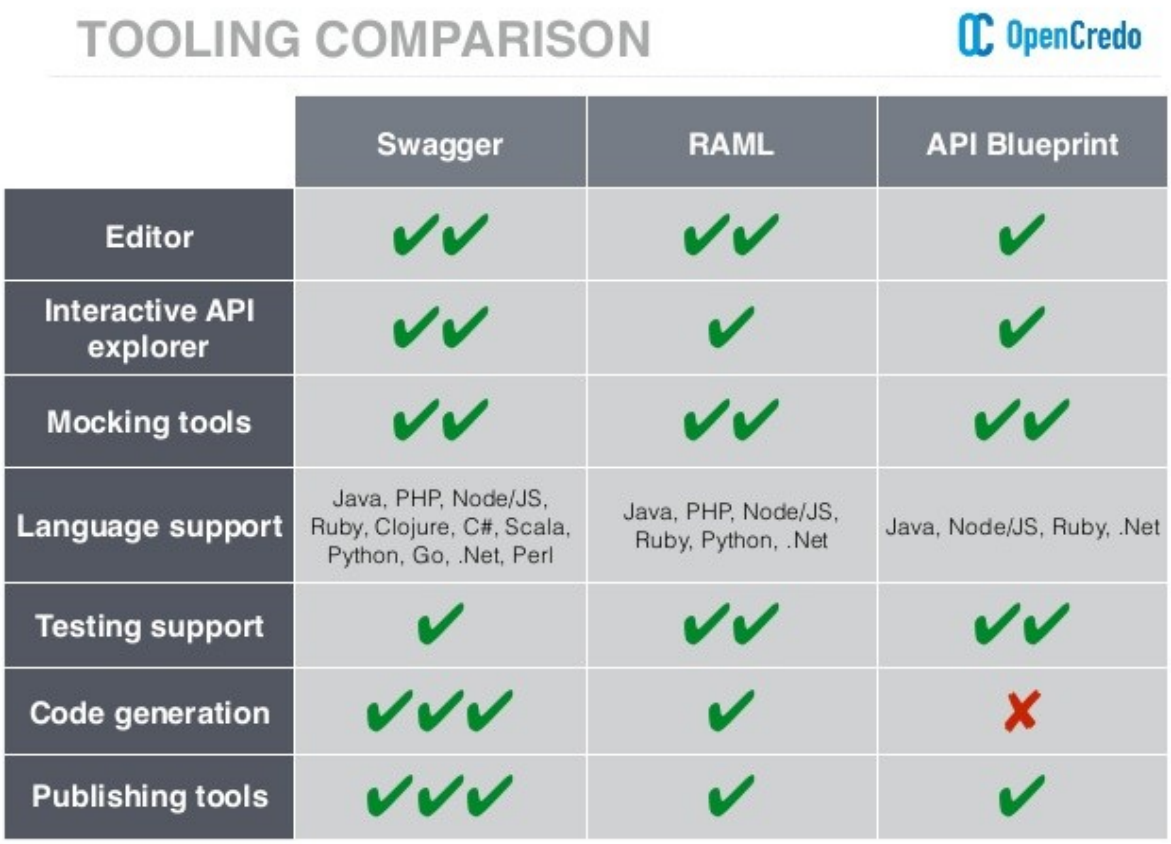
- AWS offers a multitude of in depth facilities which were not required for our specific use case of a small project. The options it offers are overwhelming for beginners. AWS is a better fit for larger corporations.

- The GCP real time database stores data as JSON and is synchronised in real time to every client as compared to other google databases. This works perfectly for us as our API has to return the data in a JSON format as well.

- Since we were using google cloud functions to host our API. It was only suitable to consider the google real time database option since they belong to the same platform.

- Microsoft Azure as mentioned above can be potentially problematic for Non Windows platforms. Given that a significant portion of the team uses Mac we felt we should not consider it.

API Documentations:



The above diagram shows a comparison of various API documentation platforms. We chose Swagger to document our API because of the following reasons -

1) Easy to use
2) Recommended by the university
3) Has a beautiful UI and makes documentation very straight forward

Passing Parameters and making calls to the API

There are 3 major ways of passing parameters to any API -

- *As part of the **URL-path** (i.e. /api/resource/**parametervalue**)*
- *As a **query argument** (i.e. /api/resource**? parameter=value**)*
- *Or as **JSON** objects as part of url (i.e. /api/resource?)*

We will be using the query argument method to pass parameters because of two reasons -

1) Google Cloud Functions do not support passing parameters as part of the URL path.
2) Industry best practises indicate that query arguments are often used for specifying search terms.

The main parameters to be passed in are -

1) start_date
2) end_date : The dates will be of the format “yyyy-MM-ddTHH:mm:ss”
3) location : example “Japan”
4) keywords : example “COVID,ZIKA”

The API will not work if any of these parameters are empty or None.

Sample Calls to API/Cloud Function

Curl sample get request

```
curl -I https://australia-southeast1-seng3011-306108.cloudfunctions.net/test_cloud_functions-2/index?start_date=2020-01-03xx:xx:xx&
end_date=2020-03-05xx:xx:xx&location=japan&key_terms='covid'
```

Corresponding 200 OK Response/

Sample Disease Report:

```
{
    "url" : "https://www.who.int/csr/don/17-january-2020-novel-coronavirus-japan-ex-china/en/",
    "date_of_publication": "2020-01-17 xx:xx:xx",
    "headline": "Novel Coronavirus – Japan (ex-China)",
    "main_text": "On 15 January 2020, the Ministry of Health, Labour and Welfare, Japan (MHLW) reported an imported case of labor
    "reports":[
        {
            "event_date":"2020-01-03 xx:xx:xx to 2020-01-15",
            "locations":[
                {
                    "country": "China",
                    "location": "Wuhan, Hubei Province"
                },
                {
                    "country": "Japan",
                    "location": ""
                }

            ],
            "diseases":[
                "2019-nCoV"
            ],
            "syndromes":[
                "Fever of unknown Origin"
            ]
        }
    ]
}
```


An example of a 404 error : When the syntax might not be correct.


```
{
    "status-code": 404,
    "error": "Not Found"

}
```


An example of a 400 error: It means that the server was not able to process the request.


```
{
    "status-code": 404,
    "error": "Not Found"

}
```


Justification for Tech Stack of Project

Architecture

Architecture is the most fundamental and critical aspect to consider for this project since it determines how the web application is hosted online and which programming languages and frameworks to use. In the following subsections, traditional server architecture is compared with serverless architecture and their benefits are shown in Table 1, and in Table 2, different serverless products are compared against each other.

Server vs serverless

Server	Serverless
● No cold starts ● No vendor lock-in so one can choose any programming languages ● No function timeouts which is critical for long-running tasks ● Advanced monitoring	- Low cost - Scale automatically to cope with traffic surges - Less effort for maintaining project infrastructure - Simplified backend code - Reduced packaging and deployment complexity

Table 1: Benefits of server and serverless architecture.

Even though the traditional server architecture gives complete freedom over programming language choices, popular programming languages such as Python, Java, Node.js and Golang are supported by serverless architecture. In addition, Python and Go can considerably lower cold start times, and there are no time-critical tasks involved in this project. Thus, cold starts appear less problematic when considering serverless architecture. The simplicity resulting from serverless architecture is appealing given that we are new to web hosting and time constraints. Therefore, we will use serverless architecture.

Serverless products and providers

	GCP Cloud Function	AWS Lambda	Microsoft Azure Function
Programming languages supported	Node.js, Python, Golang, Java, .NET, Ruby	Java, PowerShell, Golang, Node.js, C#, Python, Ruby	C#, JavaScript, F#, Java, PowerShell, Python, TypeScript.
Maximum execution time	9 mins	15 mins	10 mins

Scalability	Automatic scaling	Automatic scaling	Automatic scaling
Function limit	1000 functions per project	Unlimited	Unlimited
Concurrent executions	Upto 80 Concurrent executions	Upto 1000 Concurrent executions	Capped at 10X the number of core on the VM
HTTP(s) invocations	HTTP trigger	API Gateway	HTTP trigger

Table 2: Comparison of some main features of different serverless products

From Table 2, it is clear that these serverless products are generally similar to each other and thus, the UI of their web portals really affects our decision. Comparing Figure 1, Figure 2 and Figure 3, it is clear that Google Cloud Platform has the most simple and intuitive web portal. Thus, since our team has no prior experience with serverless products and is given limited time to complete the project, we will use Cloud Function provided on Google Cloud Platform.

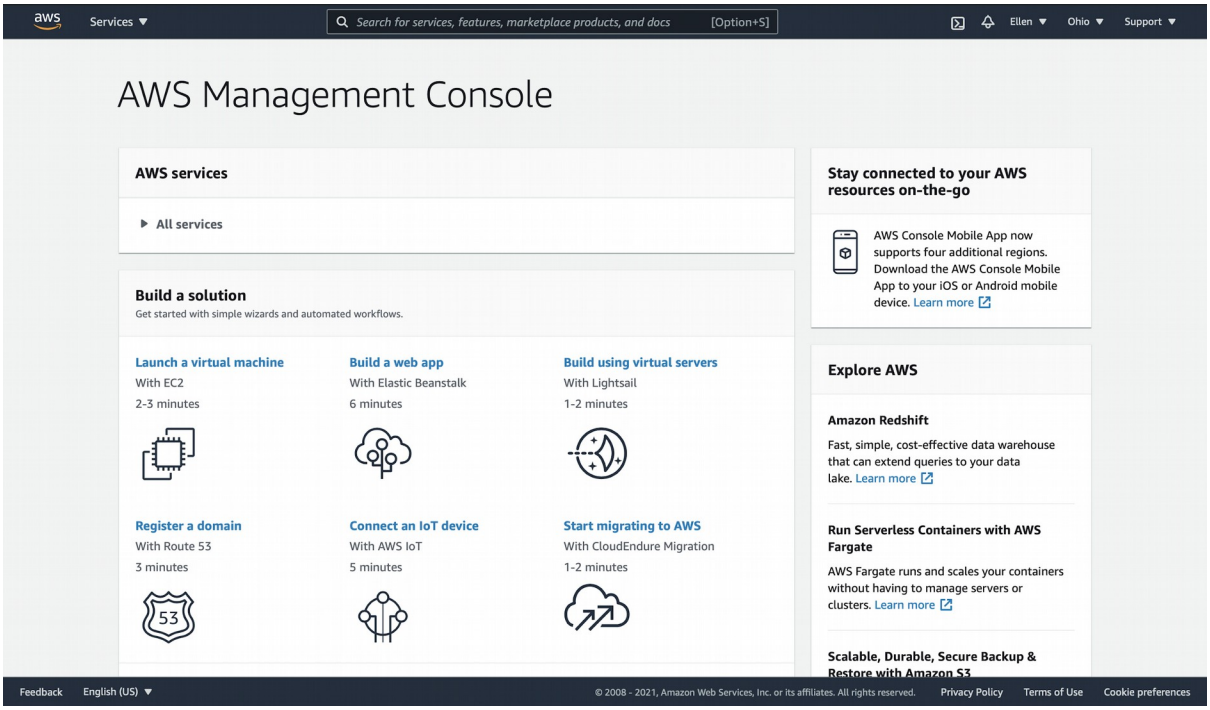


Figure 1: AWS web portal

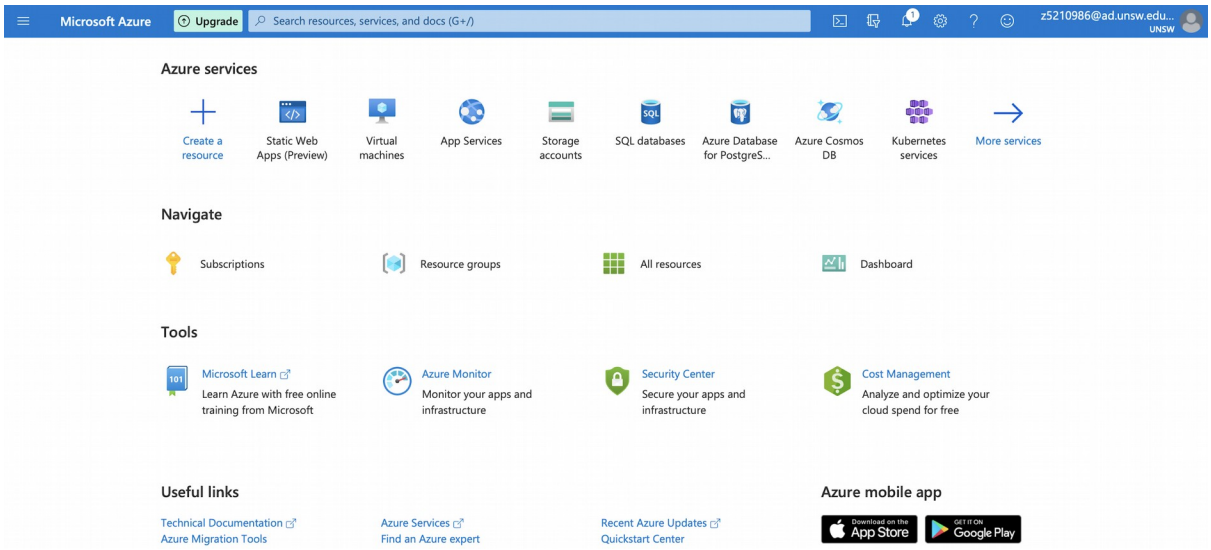
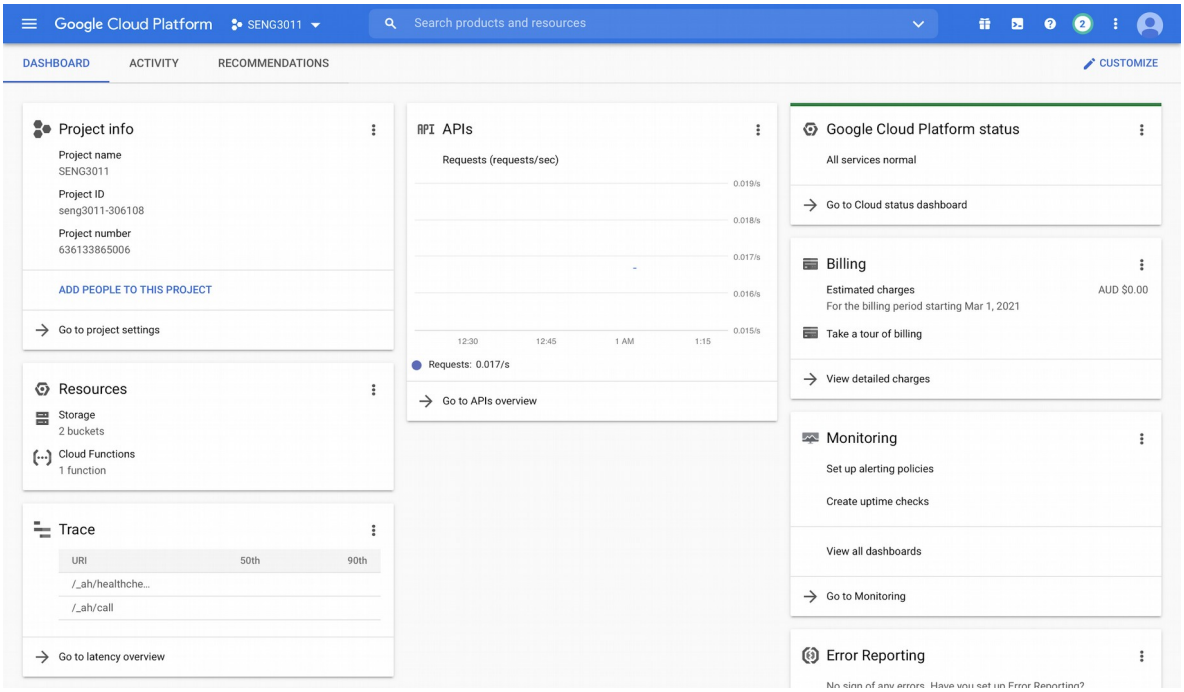
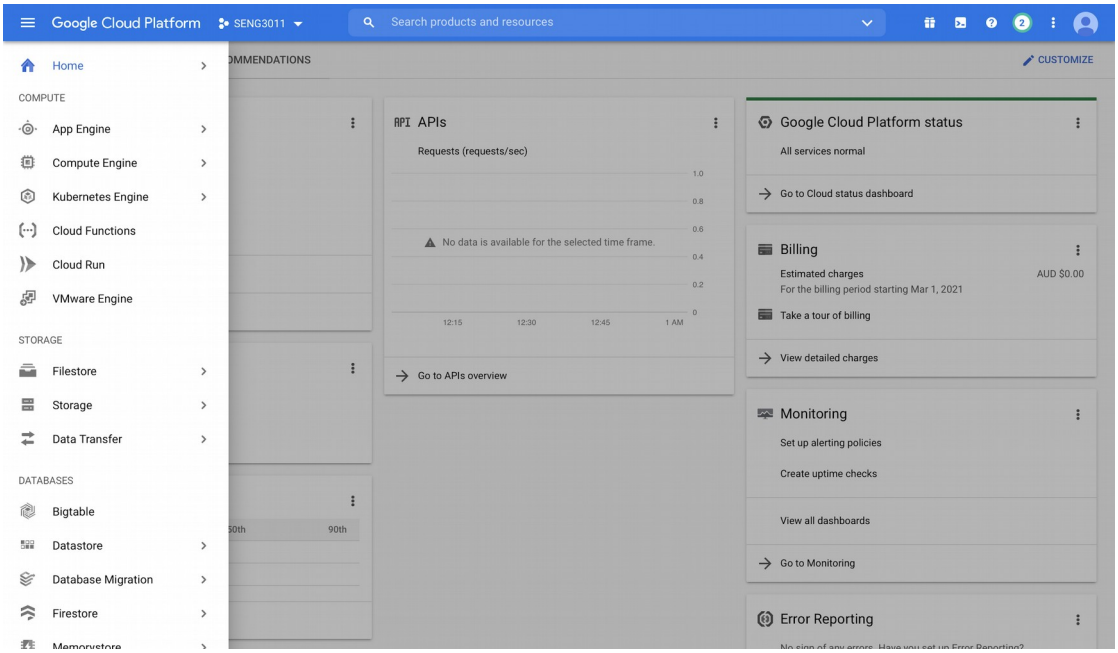


Figure 2: Azure web portal



(a)



(b)

Figure 3: Google Cloud Platform web portal

Frontend

React, Vue, Angular, vanilla JS

Backend

Language	Speed	Firebase/ Community support	Group Familiarity	Integrations, APIs, SDKs	Frameworks & Libraries support
Python			5/5	3/5	
Javascript			3/5		
Node			1/5	3/5	
Java			5/5	4/5	
C#			2/5	2/5	

Python vs Javascript vs Node.js vs Java vs C++

Python:

- Good flexibility→ several Python implementations integrated with other programming languages
- Many frameworks available - Scrappy will be used for our web scraping
- High level, object-orientated
- Dynamically typed, known to be concise and readable → make developing apps faster than using languages like Java
- Open-source language → multiple libraries available

JavaScript

-

Java

- Memory consuming and slower than compiled languages i.e. C++ → run 5PHP sites in equivalence for one java site → expensive hosting

Node.js

- Inefficient in handling CPU-intensive apps- “generating audio, video, editing graphics are some concurrent requests which cannot be currently managed”

C#

- Low security
- Complex to debug
- Bootstrap issues
- C# is not used on the client-side because there isn’t good infrastructure available.
- Easier to create templates using other languages → implementation is more difficult

Python was ultimately chosen as the main language for our project. Given it’s flexible nature, it is best able to integrate with all other frameworks chosen within our web stack. Also, all team members are highly familiar with using Python. Additionally, it has multiple libraries and frameworks available which will be essential given the dynamic range of features our website will have i.e. web crawling, graphing, charts, other APIs.

Deployment Environment

Deployment Hosts

Below are the comparison of deployment hosts we considered:

Host	Price	*Performance	Security	Suitable for production	Industry relevance
Localhost	Free of cost	Not acceptable, can only be run on one machine	Not secure, For LAN access only	No, only accessible by the person running the machine	N/A- Not used for production deployment
AWS	12 months free, ~4.8USD/month	2vCPU, Memory: 0.5 GB	Highly secure. Prefered by most businesses	Highly suitable for large organisations. Provides great features	Highly used, ranked on top for market share with 61% (as in 2019)
Google Cloud Platform + Firebase	free + $300 (additional cost incurred) ~3.32USD/month (customer-friendly pricing)	2vCPU Memory: 0.6GB, Fast	Very secure. Google gives security high priority	Not the best option for big businesses. Although recommended for startups. Is growing significantly.	Ranked 3rd with 6% market share (as in 2019)
Microsoft Azure	12 months free trial, ~4.7USD/ month	2vCPU, Memory: 0.5GB	Highly secure	A great choice for businesses. Rapidly scalable and feature-rich cloud	Used moderately, ranked 2nd with market share of 17.6% (as in 2019)

Reasons why we choose FireBase and GCP Cloud Functions

Firebase is a set of solutions: Data Storage, Authentication, Hosting, User activity analysis, Crash data collection, etc. Using firebase we can actually avoid using several different SDK.

One of its solutions, Cloud Firestore is a NoSQL database that can handle huge volumes of data and has excellent expandability, which is precisely what we need to store data collected from continuous web scraping. It's cloud-hosted, easy to maintain and integrates with cloud functions and other tools on GCP.

Cloud Functions for firebase is a serverless framework, which can automatically run backend code in response to events triggered by Firebase features and HTTPS requests. It's easy to integrate with other tools on the Firebase platform and it has zero maintenance cost as the Firebase automatically scales up computing resources to match the usage patterns of the users.

In general, Firebase is charged based on the usage and starts for free, so it's definitely the economic solution for students to use.

Development Environment:

Visual Studio Code as our primary IDE

- It has an easy-to-use interface, powerful functions like syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets etc.
- Integrated Terminal for easy git pull and push and Build-in Marketplace that lets us install libraries we need in a very convenient way.
- Support on MacOs Linux and Windows, easy to code in any circumstance**.**

Operating System:

We will be using MacOS to build and test the back-end, it has a nice terminal and ssh to conveniently transfer files between local and server, and it's the most commonly used by our teammates. Front-end and Web Scraper will be both coding using Python 3.7 and Beautiful Soup 3.9.3 and hosted on GCP cloud function, which will be compatible with Windows, Linux and MacOS.

Cross-browser compatibility

- Keeping our code simple is the best and simple way to achieve this, as the simplicity in coding is a fundamental principle in any circumstance. Simple code means less bug and easier to debug as well as transfer the website between browsers.
- Add DOCTYPE at the beginning to avoid ‘quirk mode’. DOCTYPE is an instruction at the beginning of your code that tells browsers which language you used to write your code. And “quirk mode” is a backup plan for browsers to render old websites that don't have a DTD or outdated DTD version below HTML4, that is not the best way we intended to render our pages. Thus DOCTYPE is an important element we need to remember.
- Create CSS reset style sheets at the beginning of our CSS style sheets, to ensure different browsers can easily identify their style sheet and implement the appropriate default values throughout automatically. Which will make the display properly on any browser.
- After all, we will still keep testing across browsers while coding the front-end to make sure the code is compatible and prevent deviation.