We Byte- Deliverable 1

Note: Data source is promedmail.org

1.1 API Development

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

1.1.1 API Web Scraping & Libraries

Since our data source (promedmail.org) doesn't have an API, we will be developing a web scraper in order to extract any appropriate information and aid with the development of our own API. Through this process, we'll be able to download the html from a given page using headers and/or parameters and ultimately query this information to build our database.

We plan to use BeautifulSoup and Urllib in order to create the web scraper and use the libraries, Flask and Flask-RESTful, to build our REST API.

1.1.2 API Testing & Documentation

To comply with the project specification, we intend to use stoplight.io to document our API, and have all endpoints return JSON objects.

Additionally, to make sure our API functions as desired we will be utilising Python's testing modules to ensure the API is free from bugs, is reliable, and easy to use guaranteeing the quality of our product. In particular, we will be using pytest for unit testing, and the requests module for automated testing of our endpoints. To ensure thorough testing, we will also be using the pytest-cov library to ensure maximum (as close to 100% as possible) code coverage. Finally, we will be using Postman for manual verification and debugging during development. By using stoplight.io to map out the expected behaviour of our API prior to implementation, we can ensure that the endpoints are implemented the way clients (the other groups) expect them to be and that the API serves its purpose.

1.1.3 API & Information Storing

We plan on using an noSQL database, more specifically Amazon's DocumentDB, to store the information that our API provides. The web scraper we will develop in conjunction with the API will extract relevant data from promedmail.org and further store the information in the database.

To ensure the efficiency of our API, and reliability of the information, we currently intend on running the web scraper once a day to update the database and reflect current statistics regarding any outbreaks. However, as development progresses the method in which the database may be change to ensure utmost efficiency of our program.

1.1.4 API as a Web Service

To run our API as a web service and allow other users to gain access to it, we plan on hosting our API using the Serverless framework to deploy into AWS Lambda and API Gateway.

1.2 API Parameters

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)

WeByte Stoplight.io
[https://sim.stoplight.io/docs/seng3011-webyte/PHASE_1/API_Documentation/reference/WeByteAPI.v1.json/paths/~1search_keywords/get]

To get an even more detailed code view of this (and our models) please view out github repository master branch (SENG3011-WeByte/PHASE_1/API_Documentation/reference/WeByteAPI.v1.json).

1.2.1 Parameter Passing

Parameters will be passed to our API through URL parameters. For example, passing parameters 'location' and 'keywords' to a keyword search API route hosted at webyte.com, would be done using a GET request with the path:

www.webyte.com/keyword_search?location=sydney&keywords=covid&time=2021-03-03

This utilises the standard URL format to query our RESTful API.
A request like this will return a JSON object whose exact format will be defined in our stoplight.io docs for each API route. We plan to use the return data types laid out in the project specification so as to assist integration with other teams API's, but we may expand on this list of types with our own as necessary.

1.2.2 Example API Interactions

As per the specification, all API requests will generate JSON outputs, an example route and response(s) is available on our stoplight.io page.

1.2.3 Location

As per the specification, one of the parameters a user can input will be the location of an outbreak. The various methods in which a location can be accessed are analysed below:

Method	**Cost**	**Usage Ease**	**Taxonomies**	**Source**
String	Free	Extremely easy	No	
Geonames	Free	Relatively easy	Yes	https://www.geonames.org/export/ws-overview.html
Google	Free with low	More complex	Yes	https://developers.google.co

Places	usage, but requires billing information			m/maps/documentation/places/web-service/search

From our research, we have determined Geonames to be the stronger option as it has taxonomy functionality and it is simpler to set up and use in comparison to Google Places. To represent a location using Geoname, we can simply store the Geoname ID (a unique Integer) and use the Geoname's taxonomy to store any relevant locations.

1.3 Development and Deployment Environment

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

1.3.1 Implementation Language: Python

	Python	**PHP**
Team familiarity	5/5	0/5
Web Frameworks	Many web frameworks	Access to mature frameworks
Speed	Interpreted language, slower	Faster than Python
Open Platform	Designed with features/libraries to facilitate data analysis and visualisation	Platform independent
Web Service Libraries	Easy integration with flask and/or django	Large array of libraries available
Industry Relevance	Industry Leader, many deployment options	Open source, easy development
DataBase	Doesn’t support database connectivity as much as PHP but RDBMS like PostgreSQL, MySQL, SQlite are still accessible	Able to access more than 20 different databases
Dependency management	Pipenv	Php composer
Readability	Indentation requirements makes it makes it very readable	Highly documented- classic approach
Debugging	Faster	Slow
Usability	Very versatile - able to use it for web development, robotics, science etc	Mainly used for web development

Python has the greatest appeal for developing an API given its flexible nature, ability to integrate seamlessly with many web frameworks, cross-platform design and simple integration into the rest of the chosen web stack. Additionally, all team members are highly familiar with Python, making it the best language to use to ensure our team is easily able to collaborate and develop across multiple platforms. Finally, the abundance of modules and libraries that come alongside Python means it has added versatility and will aid with adding any additional features. Despite the advantages that come along with using PHP and other backend programming languages like access to mature frameworks, speed and community support, the teams’ familiarity with Python and simultaneously its simple integration with Flask means it would be the best suited language for our project.

1.3.2 Web Framework: Flask

	Flask	**Django**
Team familiarity	5/5	1/5
Size of project	Preferred to develop smaller apps esp REST apps as minimal coding is required	Preferred to develop larger applications with a good code base
Flexibility	Large range of external libraries and add-ons	In built features and modules - less freedom and control but Django is older - more support
Support	Inbuilt support for sessions	Larger community of users
Structure	Allows freedom to implement whichever modules/libs desired. Implements bare minimum upon use	Consists of all tools and/or packages for full stacked applications and provides database interfaces, directory structure etc immediately
Difficulty	Simple, minimalist, easy to learn	“Steeper learning curve”
Database	Supports relational DBs & noSQL	Supports a number of relational DB including SQLite, PostgreSQL, MySQL, Oracle
Python	Both easy to integrate with python	
Routing	The request object is global in Flask and much easier to access	1. request matches a URL pattern 2. request object(HTTP request) is passed to a view 3. view is invoked 4. must pass request object around to access it
Other	- Easier for rapid prototyping and stand alone tasks - easy to bootstrap and maintain	- need experienced resources to maintain

Given the entire team has interacted with Flask and Python together, and the simplistic nature of the framework, we decided to use Flask. In addition to this, the ability of Flask to use a large range of external libraries and modules with it and its easy integration with both relational databases and NoSQL means it would satisfy the versatility required to build both the API and web application for our project.

1.3.3 Deployment Environment

Platform Provider: Amazon Web Services

There are many cloud platforms to choose from, however, AWS was chosen over the others due to the reasons listed below:

- **Prevalence in industry**: it is by far the most used singular cloud service. With a market share of 33%, with its next biggest competitor Azure at 13%.
- **Large ecosystem**: of over 200 cloud products. Making it easy to deliver more complex microservices.
- **It's free tier:** offers an abundance of database storage and compute time. More than enough for this project
- **It is very heavily documented:** and has many tutorials, which will help make it easier to learn

Platform: Serverless Platform

We decided to implement our API with a serverless architecture. This meant we would be using AWS Lambda and API Gateway, although to abstract the configuration on AWS and ease the development and deployment processes we decided to use the **Serverless framework**.The reasoning for this is listed below:

- **Easily scalable**: We expect that the usage of our API will have long periods of inactivity then spikes of high activity (such as demo day). Using Lambda provides a lot of horizontal scaling, it defaults at 1000 concurrent requests.
- **Cost effective**: It only gets run when it is triggered, this saves on a lot of expense compared to using a server model in which the server must have long periods of uptime. The free tier of Lambda includes 1 million requests and 400 000 GB-seconds of compute time a month, as long as each function is limited to a compute time of 5 minutes. This plenty for the scope of this project
- **Simple**: The Serverless framework has an extremely easy set up. Especially in comparison to ECS, which has significantly more configuration. By using this framework, the development and deployment environments can be simply handled and we can focus on writing code.
- **Flask Compatibility**: Flask is a web framework our team is extremely familiar with, it is therefore desirable that our platform is compatible with it. AWS Lambda is not compatible with Flask without complex configuration, but by using the Serverless framework we can easily use Flask.
- **Modular**: Each Lambda function is designed to serve a singular purpose. This allows us to design a much more decoupled design, making it much more simpler and more manageable. As opposed to an entire EC2 container which would run many different aspects of the API, and therefore a simple bug could compromise the entire API more easily.

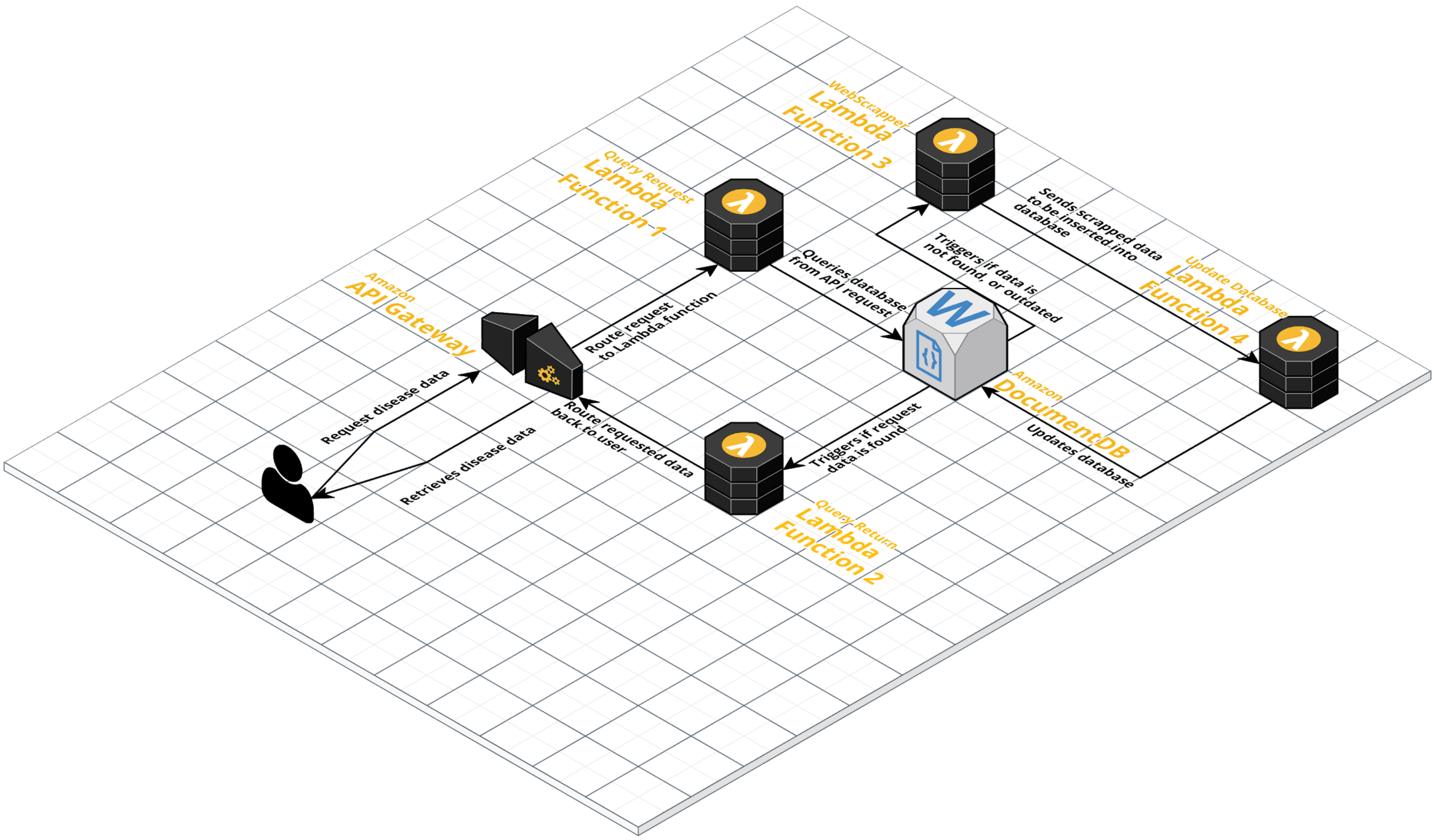
Database: DocumentDB

DocumentDB is a noSQL structured database. The reasons why this noSQL database was chosen were:

- **Requires no predefined schema**: this is important considering that the data on promed is loosely structured
- **Since there are different data types on promed (reports, articles etc..)**: noSQL allows for the ability to store documents with different structures. This will also be helpful when we start to incorporate more unstructured data from other API's such as social media posts etc..
- **BASE compliance**: SQL is more suited for ACID compliance, whereas noSQL is better for BASE compliance. SInce the data being requested doesn't require such rigid compliance (unlike databases for bank transactions), the BASE compliance is more than enough to provide the necessary data for our users.
- **MongoDB compatibility**: Shares similar syntax to MongoDB, which is an area of expertise for our team.
- **Easily integrated into the deployment:** DocumentDB is an AWS product, making it very easy to use with our Lambda functions.

Deployment Architecture

The below architecture describes an initial draft of how our serverless API could be implemented.


WebScrapper
Lambda Function 3
Query Request
Lambda Function 1
Amazon
API Gateway
Sends scrapped data to be inserted into database
Update Database
Lambda Function 4
Triggers if data is not found, or outdated
Queries database from API request
Route request to Lambda function
Request disease data
Retrieves disease data
Route requested data back to user
Triggers if request data is found
Amazon
DocumentDB
Updates database
Query Return
Lambda Function 2

1.3.4 Development Environment

A strong development environment ensures that all our developers' environments and deployment environments are consistent. They need to be consistent in their:

- Packages and package version
- Language version
- Operating System

It's also important to note that since we are using AWS resources in our endpoints such as DocumentDB, our development environment should be able to access them.

We will be using venv to maintain a consistent language version and package versions. venv allows us to create lightweight virtual environments, with their own Python binary and installed Python packages.

We will be using the Serverless framework to maintain the operating system and access AWS resources. Using the framework, we can deploy a version of the API to any arbitrary environment, for example dev-Tony. This allows us to test the API in an environment identical to the deployment environment.

1.3.5 Libraries

Below is a list of the libraries that we are aware are necessary. Keep in mind that as the development progresses, there is an increased likelihood of the libraries being used changing.

Web Framework

Flask is a lightweight web framework. We will be using Flask and Flask RESTful to build our API.

Web Scraping

As mentioned in section 1.1, in order to aid with the web scraping functionality, Python's Urllib and Beautiful Soup libraries will be used. Urllib allows us to open URLs in order to process them. Beautiful Soup parses the HTML from opened URLs and allows us to extract relevant information.

Natural Language Processing

spaCy is an open sourced natural language processor which can be used to pull locations out of text, aiding with the process of taking in location as a parameter. It can also be used later to distinguish between different reports within articles.

Testing

We will be utilising Python's testing modules to ensure the API is robust.

- pytest for unit testing.
- pytest-cov library to ensure maximum code coverage
- requests module for automated testing of our endpoints
- Postman for manual verification and debugging during development

1.4 Project Plan

Provide a project plan showing team member responsibilities, work arrangements and any information team members will be using to coordinate their activities. You should also mention any software tools used by the team to assist project management.

As a team, we've decided to have weekly meetings from 1-1:40pm on Tuesdays, followed by our mentor session from 1:40-2pm. These meetings will be used to discuss progress, solve any imminent issues, as well as plan upcoming work and split tasks for any future deadlines.

In order to best articulate team member responsibilities and plan out any deliverables and/or sprints, Trello, Teams(and in built Microsoft OneNote) and TeamGantt will be used as all platforms allow for easy real time collaboration. Trello in particular will be used for the management of sprints and the breakdown of tasks amongst team members, whilst TeamGantt will allow for the careful planning and consideration of all project requirements .

A link to our Trello board is as follows: https://trello.com/b/1oBZ0DCl/seng3011-we-byte.

A link to our Gantt Chart is as follows: https://bit.ly/3bYwfdW

During the development of our applications we intend to use Figma (Web APplication frontend) and stoplight.io (API documentation) as it will allow our team to design,prototype, test and collaborate our ideas in real time.

Finally, to ensure seamless code collaboration and version control during the development of the API and web application, our team will be using GitHub.

A link to our GitHub repository is as follows:
https://github.com/Harsimran-Saini/SENG3011-WeByte.

As we get further into the project, the responsibilities of each team member for the relevant deliverable will become more apparent and hence will be updated accordingly.

Name	**Email**	**Experience**	**Work Arrangements**
Jaiki Pitt	z5213775@unsw.edu.au	• Backend: Python, Flask, SQL, noSQL, mongoDb, Java, C, OOP • Frontend: Javascript, ReactJS, jQuery	Tuesday-Friday. Except Thursday 12-1

James Bradley	z5210562@unsw.edu.au	● Backend: python, SQL, Java, C, C++, OOP, Optimization using Genetic algorithms. ● Frontend: ReactJS	Mostly available Wednesday, Thursday, Sunday. Can't do Friday / Saturday before 5:00pm
Sruti Desai	z5260319@unsw.edu.au	● Backend: Flask, Python, SQL, Java, PostgreSQL ● Frontend: JavaScript, jQuery, HTML/CSS	Available to meet after 6pm Monday, Tuesday, Thursday and Friday Weekends if necessary
Lachlan Harvey	z5258871@unsw.edu.au	● Flask, python, react, scraping	Mostly free Tuesday, Wednesday, and weekends if necessary Other weekdays free after 6pm
Harsimran Saini	z5208912@unsw.edu.au	● Worked on using APIs with flask and Python in SENG2011. ● Confident with use cases and business requirements	Free to meet up on weekdays after 5 and on Tuesdays between 1 to 1:40.