# University of New South Wales

## SENG3011_Coders : Design Details

The API is based on scraping data from the web and presenting it on a web service.

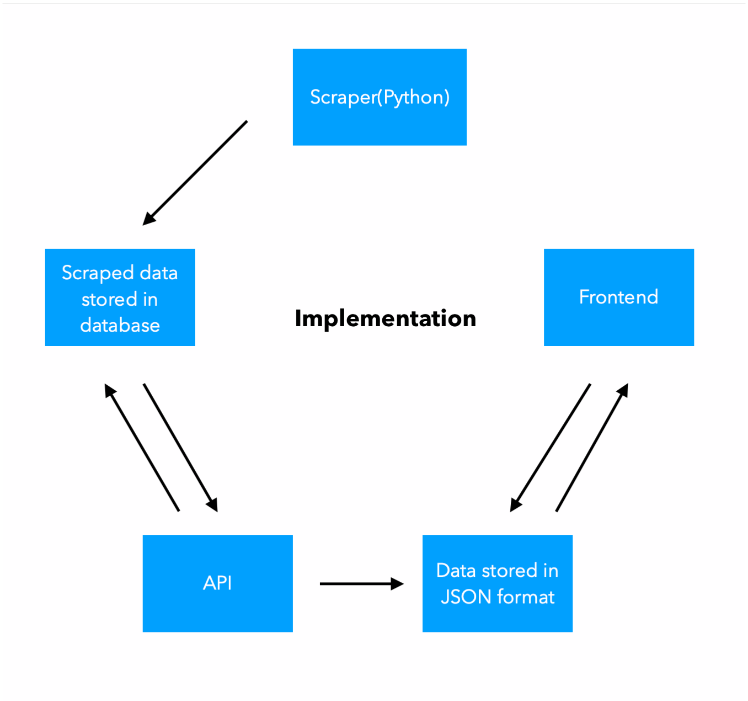
**Describe how you intend to develop the API module and provide the ability to run it in Web service mode.**

**Design Details :** The design is divided into 3 parts,

**Scrapper :** The web scraper will extract the data out of www.cdc.com.gov and store it in the database (created in sql).

**API :** The API (using RESTFul) will get a request (GET request) from the front-end in the form of JSON. It will then access the database and return the response in the form of JSON.

**Front-End :** It will be the main platform where the user will be able to interact. It will create a JSON as per the query of the user and call a GET request to the API using the JSON as body of the request.



**Implementation :** Since the project is divided into 3 parts,

**Scraper :** For this, the team used BeautifulSoup library of Python to parse the site and extract the relevant data. We decided to extract some sections of the Data Source provided and then the data was stored into the database.

**Database :** A basic SQL database made using SQLite3 on python. With various functions to manipulate, add and get data.

**API :** A RESTful API is created in Python. Since the main function of this API is to show the data from the database as per request. It only has one endpoint which is 'show'. It receives a JSON file as per its request it parses through the database and has 2 possible responses-

**200 :** Which means that the data was found and thus a JSON response is created and passed back to the front-end.

**404 :** Which means that the data wasn't found and thus an error response is created as a JSON and passed to the front-end.

**400 :** Which means that the page was not found.

**Front-End :** It's a web-based GUI which creates JSON query and then uses that to create a GET request to the API and then displays the response accordingly.

**Challenges / Shortcomings :** One of the major issues we as a team faced was communication and division of task.

We have created a very basic api and scrapper. We haven't made our scrapper capable of scrapping reports from the websites, but it is something we are looking forward to implement, and update the API accordingly.

A challenge we faced is in the hosting of the swagger documentation, as well as working on the google cloud platform.

Connecting the database using sqlite also took a bit longer time than expected.

**Testing :** The API will be tested on behavioural, Contractual and Result API testing grounds. The API will be examined whether it delivers expected behaviour and handles unexpected behaviour properly or not. It will be tested to see if tasks defined in specifications have been achieved by code with given right inputs and date format.

As a whole it supports the intended cases that it was designed to solve. Since we are going to use Python Flask in which constantly testing the api is fairly simple we can easily make test cases and then run the code through pytest and other test files.

There were various tests written in the test script -

**Test1 :**

For a completely wrong input,

```
Example -

{

        location = “”,

        key_terms = “”,

        'start_date': "",

        ‘end_date’:""

}
```

for this particular input structure the output status code should be

400 - BAD REQUEST

**Test 2 :**

For a correct input style for available output,

Example -

```
{

	'location': “Chicago",
	'key_terms': "Salmonella , hedgehogs”,
	'start_date': “2018-12-19T05:45:10",
	‘end_date': "2019-05-21T05:45:10"
}
```

There are a few different things to test -

2.1

Check the Status Code -

200 - OK

for a correct response the response code should be 200

2.2

Check  the dates

This is the test to check if the outputs date is inside the range of dates given.

The way we check this is by making 3 date time objects and comparing between them

2.3

Check for bathing key terms with the headlines

This is a test which takes every key word supplied by the user and checks if the output’s headline contains each of those words.

**Test 3**

This is the check for an input whose correct output is not present in the database.

```
Example- { 'location': “Chicago",
           'key_terms': “Cancer",
           'start_date': “2018-12-19T05:45:10",
           'end_date': "2019-05-21T05:45:10"
}
```

In this we check that the status code returned by the API is

404 - NOT FOUND.

We do the following tests checking the various possibilities of inputs with the outputs.

**Extensions :** The team is wishing to further enhance the user experience with additional features like a map with affected areas for the period selected. The website would include a displayed list of alerts in a particular location.This provides an easy understanding the scope of outbreak. These features can help to make an application for mass welfare in future.

**Optimisation:** The team will work on the optimisation of the API to make it efficient and speed up the scraping process. Excessive API calls can cause inefficient code. Team will optimise API and handle under fetching and over fetching problems. We will always aim to reduce redundancy in code and unnecessary requests made.

An API acts as an interface between two different applications so that they can communicate with each other. Web service facilitates interaction between two machines over a network. With RESTFul web services, there is a natural mapping between the HTTP methods.

**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).**

**<u>Passing parameters to the API module</u>**

Passing the parameters to the API module from the front end would be using JSON data type. The front end would take the data in from the user and create a JSON like -

Parameters = {

"location": "Chicago",

"key_terms": "Salmonella",

"start_date": "2019-01-19T05:45:10", "end_date": "2019-01-28T05:45:10" }

This is then passed to the API as a GET request.
R = requests.get('localhost:5000/show', params=Parameters)

**<u>Collecting API results</u>**

The API then receives the JSON file, searches through the database for the possible match of the Parameters.
The result is then curated into a JSON. Example,

sample_result = {
'url' : "www.cdc.gov/salmonella/typhimurium-01-09/index.html",
'date_of_publication' : "2019-01-25",
'headline' : "Outbreak of Salmonella infections linked to pet

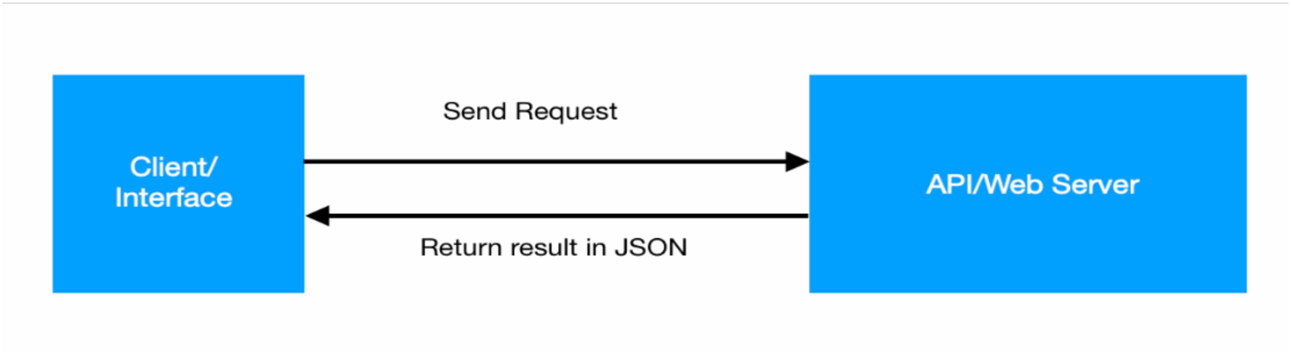
hedgehogs ",
'main_text' : "CDC and public health officials are investigating a

multistate outbreak of salmonella infections linked to pet hedgehog ", ‘reports' : "NULL" }

The result can be either some data like the one above or a failed response like,

error_result = { ‘error' : '404',

'details' : "couldn’t find any report on the given set of keywords" }



As the above figure illustrates, the HTTP request will be sent to the Web Server using the API and then the result will be returned to the client in JSON object sent in in the body of the HTTP response.

**Present and justify implementation language, development and deployment environment and specific libraries that you plan to use.**

In choosing which implementation language the team decided to use Python, we have considered both NodeJS and Python which are the two most popular languages in backend development, with the former being very popular with startups.The languages were compared and the most suitable was selected. We chose Python for the reason that Python’s advantages were more suited to our project’s needs and outweighed those of NodeJS. For example, Python’s BeautifulSoup library will enable us to scrape data more efficiently from CDC as HTML tags may differ from article to article which can be difficult to fetch.

| Python | NodeJS |
|---|---|
| Python allows tasks to be achieved with fewer lines of code. | Better performance overall: event-based programming makes its processes run faster. |
| Python provides more advanced web scraping frameworks such as Beautiful Soup which incorporates parsing trees and can parse less than well formed HTML. | Only Javascript is required to code the Frontend and Backend as opposed to multiple languages for the Front and Backend. It allows tasks to be completed faster in smaller teams. |
| Python also has a richer standard library for server side development which supports such operations as argument parsing, creating temporary files and unit testing. | NodeJS is better suited to real time web applications as it is based on Chrome's V8 engine which is very powerful and fast. |
| Error handling means debugging takes less time and is very easy in Python in comparison to other languages. | Node's non-blocking I/O allows maximum usage of CPU and memory. If there is intensive I/O operations in the application and there is a lot of data being shuffled between the backend and frontend, Node is the better choice. |

We narrowed down the list to two Python frameworks, Django and Flask and selected the most suitable one. In the end the requirements of our project needed freer access to libraries as well as more flexibility in our implementation, so the team decided to chose Flask.

| Flask | Django |
|---|---|
| Flask is built on minimalism, enabling more flexibility for functionality. | Django has inbuilt features and templates. |

| | |
|---|---|
| Flask has no browsable API option unlike Django so a less inconvenient alternative is to use Swagger in front of the API. | Django’s framework creates HTML pages to execute all API endpoints and for browsing. |
| As a result, its performance is generally faster. | |
| Much more freely able to integrate Flask with NoSQL databases such as MongoDB which stores data in JSON. | |

For this project, the team wanted greater versatility in implementation as we planned to use a wide range of libraries with BeautifulSoup being one of them.