SENG3011 - LetsHD Deliverable1 Report

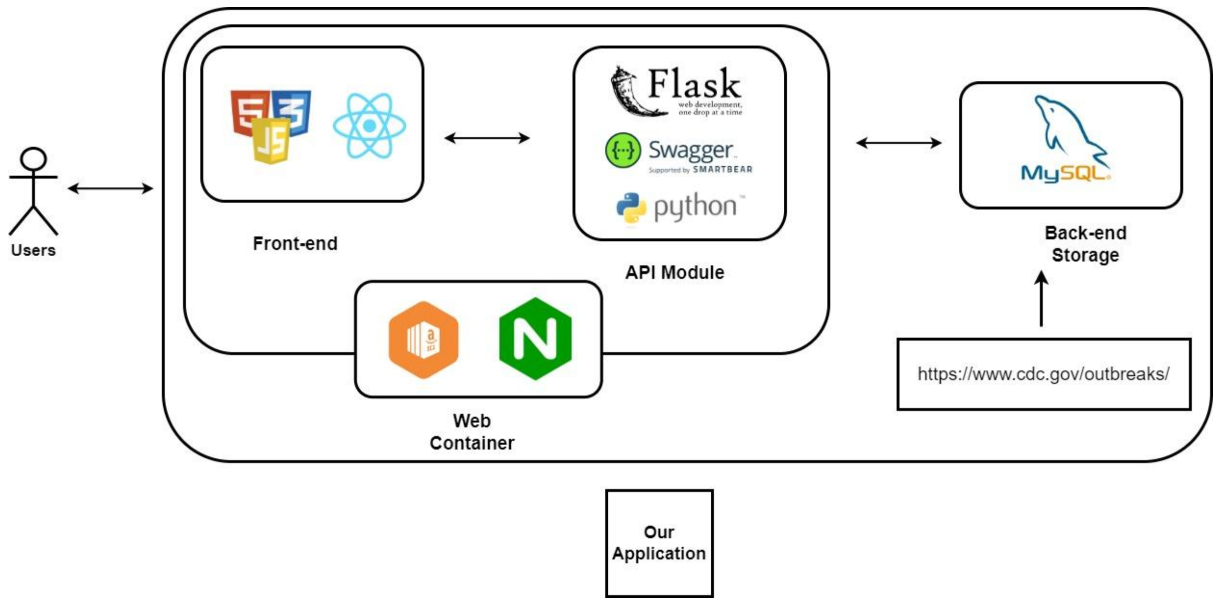
By -

Jiaxin Liu(z5191691), Zhiyuan Yin(z5191561), Yilin Zhu(z5212995), Ziyu Zhou(z5212919) Jing Deng(z5213505)

Table of content

Design Details

API module design and Web support



(SAD Diagram)

API Module development

In order to enforce stable and efficient communication between our front-end infrastructure with our database, a well-organized API is inevitable. As shown in the software architecture diagram above, the chosen programming language and framework for the API modules and back-end storage are Python, the Flask web framework and MySQL database. Python and the Flask web framework together will provide a portal to receive and handle different requests coming from all kinds of clients, these requests will be typically querying for existing data entry inside our MySQL database. Depending on the permissions of the clients and the requests, the API should either reject the requests or proceed to return desired information. The CDC API as well as any necessary scraping will be put into use in order to populate our SQL database with meaningful and accurate information.

Web Support

For deploying our API services to the web and making it publicly available, we have chosen to use the Flask web framework. The Flask web framework offers straightforward implementation to declare how our API should behave when different HTTP requests are received. Once all of our intended behaviors of the API are integrated with corresponding Flask web handlers, launching the Flask server inside our EC2 instance will then deploy our API to the address where our EC2 is located.

The Rationale

Python

Python being one of the most used languages across ranges of different fields in the industry has its very own advantages. Firstly, it provides developers the options to run the program with particular dependencies in a virtual environment which is extremely efficient when multiple developers are working together and iterating the API in fast-paced development cycles. Additionally, being a popular option in the field of API development, support from the community is widely available on the internet, thus reducing the difficulties of any necessary learning. Lastly, Python is highly compatible with other frameworks and tools as it provides built-in functions to efficiently interact with external entities such as MySQL databases and the Flask framework. Other programming languages such as JAVA and JS provide benefits of their own, but are not as light-weighted as Python due to their compatibility and readability issues.

Flask

Although there is a wide range of different web frameworks available for Python, Flask was an obvious option for our development team. The Flask framework is well known for its lightweight and easy to negotiate characteristics, it is effortless to set up and integrate into any system. Additionally, it is extremely scalable meaning the potential of the application is not restricted. And most importantly, the Flask framework happens to be the most familiar framework to our back-end development team.

Swagger

Documentation plays an important role in communicating efficiently how our API behaves to the users. Swagger documentation is a documenting platform that specializes in explaining the behaviors of an API. It is capable of showcasing all available endpoints provided by the API in detail. It is highly customizable, developers have the options to group endpoints by characteristic and provide samples inputs or outputs. Incomparably, Swagger also provides viewers the ability to send sample requests to examine the API in real time.

MySQL database

As many databases as there can be, they mainly come in two types, either document-based or schema-based. We have chosen to integrate schema-based databases with our system as it provides more emphasis on the correctness of the data entries, the additional flexibility provided by document-based databases is not appreciated in this particular project as most of the entries our application will be storing comes in fixed form. Out of ranges of schema-based databases available, we decided to proceed with MySQL database as it smoothly interacts with Python and is highly portable which fits our rapid development cycles. The database sizing and debugging issues that come along with MySQL are not concerning as the amount of data our development team are planning to store is not extensive, and debugging can be achieved through the use of MySQL workbench.

API parameters considerations

Foremost, we are using the RESTful API architecture and design principles, so it is necessary for us to build an API that follows the constraints of the standard. Thus, the parameters of the API we use should also follow the constraints, otherwise improper design of parameters will straightly delay the delivery of the project.

Besides, the open data that we can have are from the following website: http://www.cdc.gov/outbreaks/ and also navigate to the articles about the outbreaks on it. The specific steps of how the API works can be demonstrated as the following steps.

1.Retrieving parameters from the user

To grab the search conditions offered by the user, we will be using a HTTP GET request with an HTML form. The parameters will include:

- period_of_interest, which should be composed of 2 inputs:
  - start_date
  - end_date

  and both of the inputs would have the same format "yyyy-MM-ddTHH:mm:ss”
- key_terms, which can be referred to the name of a specific disease
- location, which can be referred to a specific city in a specific country.

A successful request will return a response to GET/results. Otherwise, if there is something wrong from the parameters input by the user, like the start_date is in the future, there will be an error box to inform the user to try again. A possible input is shown below.

- **GET/search**

  Parameters:

```
{
  "period_of_interest": {
    "start_date": "2022-01-01T00:00:00",
    "end_date": "2022-02-02T23:59:59"
  },
  "key_terms": [
    "e coli"
  ],
  "location": "US, Ohio"
}
```

Responses:

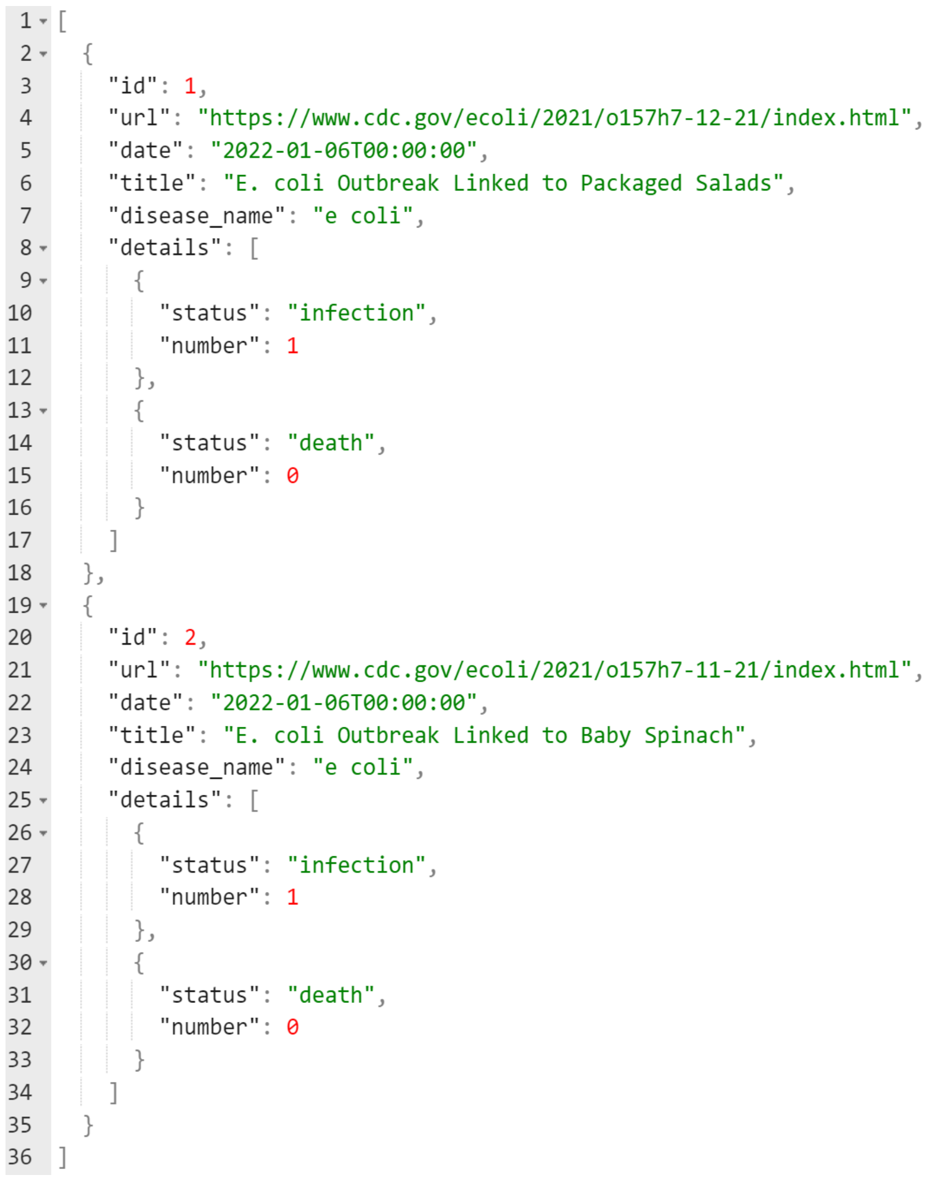
Code	Mean
200	successful
400	invalid input
404	not found
500	internal server error

2.Retrieving data from the website

After the user has submitted a valid request, the API will search the related articles from the CDC, which is the website mentioned above, grab the related details then return the response of the GET request to the user. The data will be returned as JSON format, and with a list of report objects. Each object will have some relevant information about it, including report ID, url, date, title of the report, the disease name and other outbreak details like the number of people infected, number of deaths, etc.

Some possible interactions are shown as follows.

- **GET/result/**
  Result:



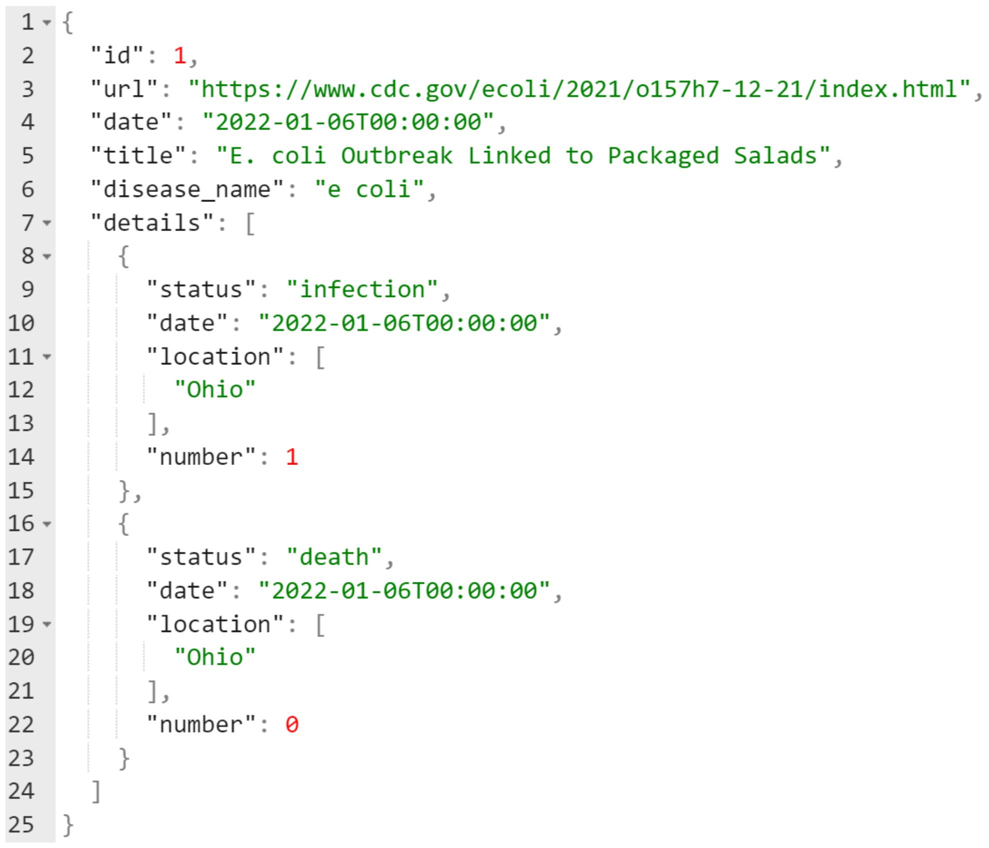
```
[
  {
    "id": 1,
    "url": "https://www.cdc.gov/ecoli/2021/o157h7-12-21/index.html",
    "date": "2022-01-06T00:00:00",
    "title": "E. coli Outbreak Linked to Packaged Salads",
    "disease_name": "e coli",
    "details": [
      {
        "status": "infection",
        "number": 1
      },
      {
        "status": "death",
        "number": 0
      }
    ]
  },
  {
    "id": 2,
    "url": "https://www.cdc.gov/ecoli/2021/o157h7-11-21/index.html",
    "date": "2022-01-06T00:00:00",
    "title": "E. coli Outbreak Linked to Baby Spinach",
    "disease_name": "e coli",
    "details": [
      {
        "status": "infection",
        "number": 1
      },
      {
        "status": "death",
        "number": 0
      }
    ]
  }
]
```

Responses:

Code	**Mean**
200	successful
404	not found
500	internal server error

- **GET/result/{id} (a specific report selected in the list above)**

  Result:



```
{
  "id": 1,
  "url": "https://www.cdc.gov/ecoli/2021/o157h7-12-21/index.html",
  "date": "2022-01-06T00:00:00",
  "title": "E. coli Outbreak Linked to Packaged Salads",
  "disease_name": "e coli",
  "details": [
    {
      "status": "infection",
      "date": "2022-01-06T00:00:00",
      "location": [
        "Ohio"
      ],
      "number": 1
    },
    {
      "status": "death",
      "date": "2022-01-06T00:00:00",
      "location": [
        "Ohio"
      ],
      "number": 0
    }
  ]
}
```

Responses:

Code	**Mean**
200	successful
404	not found
500	internal server error

As for data processing and storing, we plan to use MySQL as the DBs of our API by considering its security and portability. Compared with other mainstream DBs on the market, MySQL is portable and maintains high performance at the same time. It is well established for use with python and flask for many years. Besides that, we also plan to develop our scraper as a tool which can assist us to manipulate the DBs and retrieve information from the CDC website.

Once we have the scraper developed, the scraper will parse HTML code to collect relevant information and store it in our DBs. The scraper will run to make sure all the data in DBs is timely and concurrent every time when users send requests to our API. If there was any upgrade detected on the website, the scraper can refresh the corresponding part in DBs so that users can access the newest data.

Development environment considerations

Implementation Languages

As discussed in the section “Design Details - API Module Development” our development team decided to use the combination of Python and the Flask framework for our API services. The back-end database of our choosing is MySQL, justification for choosing is also presented in the earlier section “Design Details - The Rationale - MySQL database”.

For front-end infrastructure, JavaScript and the React framework was chosen, as it is flexible, compatible and easy to navigate. The React framework also provides a range of external libraries built by other React enthusiasts which can significantly reduce our front-end building workload and focus on other elements of the project. The combination of HTML5 and CSS3 will also be used to create styled web templates.

Deployment environment

Once the product is completed, tested and polished, the product will be launched as a web application using the AWS cloud hosting service. We can achieve cloud hosting by using the EC2 service provided by AWS. EC2 is a virtual machine service which allows us to host both our front and back-servers non-stop without allocating physical resources. The operating system of the virtual machine will be Ubuntu linux, as it provides maximum flexibility for users working in the development environment.

While the EC2 service sets up the portal to host our server to the public, we also need an extra layer to serve our front-end content to the public. Nginx is one of the two most common open source software that is made for web serving in the world, it is capable of reverse proxying, load balancing, media streaming and much more. By serving our web application through Nginx, it can efficiently manage the resource which ultimately means more end-users can connect to our application securely and stably at once since CPU, RAM and bandwidth of the virtual machine is better managed.

External API

The only external API our development team will be including in our Application is the CDC API which allows our back-end team to fetch data from the CDC website to store in our database.

High Level Architecture Recap

Architecture Recap		
Architectural Component	**Software Component**	**Software Details**
Cloud serving	AWS EC2	Virtual machine hosted at a certain remote address that allows external users to interact with our application.
	Nginx	Power web serving tool to optimize resource management
Frontend	Javascript	Most used language in the web development industry as it smoothly integrates with most web-dev software.
	React	One of the most used web development frameworks, provides rich and powerful external libraries to build out the application, lightweight and compatible.
	HTML5	The lower level web page templates used to present users with information.
	CSS3	Markup language used to add styles to the webpage.
Backend	Flask and Python	Web framework and programming language that build the communicating portal between the database and client
	Swagger Documentation	Documentation tools used to explain the behavior of our API

		service
	MySQL	Lightweight schema-based database that will be used to store our data entries
External API	CDC API	API provided by the CDC website to fetch information

Management Information

Key Objectives

Due to the limited development time our team has access to, our team decided to use the ‘divide and conquer' strategy along with the ‘Agile’ development approach. We identified the four main parts of the projects to be the following:

1. **Reports** and **documentation** - These are mostly written materials that are required to be submitted or stored somewhere for later references.
2. **Backend infrastructure -** These include the database we are using to store data as well as the API we need to build in order to communicate the database with the frontend of our application.
3. **Designs -** This refers to the UI and UX design of our application.
4. **Frontend infrastructure -** This is the window we build for our end-users to interact with our backend functionalities with ease.

Mangament tools

In order to deliver the product within time limit and with exceptional quality, Our development team has also decided to integrate additional tools to assist with our workflow. A list of chosen tools are as follow:

1. **GitHub** - GitHub was chosen to be our primary version control tool. GitHub is a powerful platform that allows developers to share and edit the same codebase across different machines which is extremely handy when multiple developers are working on the same project. On top of code sharing it offers developers the ability to switch freely between each iteration of the codebases. Last but not least, it also provides a convenient visual interface for developers without command line experience.
2. **Jira** - As mentioned above, our team has decided to implement our solution using the ‘Agile’ development approach, as its rapid changes and short period of development best suits our use case. Jira is a convenient tool that allows the development team to visualize both the overall ‘Agile’ development progress and the progress for any given sprint by simply flicking between the backlogs and current sprint boards.
3. **Confluences** - Similar to GitHub, confluences is another powerful collaborative editing platform that gives each developer the permissions to add, remove and alter documentations. It is a compact platform that gives developers the ability to take concise and meaningful notes efficiently.
4. **Google Drive** - Google drive is an online text editing tool that allows multiple members to edit a certain document at the same time through different locations and devices. Additionally, it also provides an easy interface for commenting and change suggestestion hence It is chosen to be our primary tool for editing lengthy documents such as reports.
5. **Microsoft Team**s - Due to the current Covid outbreak, our team has decided to migrate physical meetings to online meetings. Since the course is delivered through Microsoft Teams, and it provides a stable online meeting platform.

6. **WeChat** - WeChat is a lightweight mobile application that allows team members to have regular short conversations through text or voice messages.

Members’ Roles & Responsibilities

With key objectives in mind and management tools selected, we were able to carefully identify the strength of each team member and assigned each member with tasks according to their confidences. Details for roles and responsibilities for each member are as follow:

Reports and **documentations**	**Backend infrastructure**	**Designs**	**Frontend infrastructure**
All members	Jiaxin Liu Zhiyuan Yin Yilin Zhu	Ziyu Zhou	Jing Deng

Apart from completing corresponding development, team leads are also selected based on the differences between members' expertise as shown in the chart below.

Head leader	**Backend lead**	**Design lead**	**Frontend lead**
Zhiyuan Yin	Jiaxin Liu	Ziyu Zhou	Jing Deng

Each team lead is responsible for communicating and allocating work to each team member according to the situation and documenting necessary changes. The head leader is responsible for communicating with other team leaders outside of meetings to ensure teams are making reasonable progress and making schedule adjustments accordingly.

Communication & Meeting Schedules

To ensure leaders and members are aware of the progress and therefore stay on top of schedules, communications and regular meetings are inevitable. Regular meetings will take place on every Tuesday and Friday, each team leader is expected to report progress or any roadblocks encountered. Members are also expected to reach out for assistance if necessary outside of meetings via Wechat and Microsoft Team.

Progression Estimation

Below is a visual representation of how our team envisioned the project’s lifecycle.

	Week2	Week3	week4	Week5	Week6

Ideas brain-strom	🟦	🟦			
Forming requirements		🟦			
Deliverable 1	🟧	🟧			
UI / UX designs			🟨	🟨	
Deliverable 2			🟦	🟦	
Backend Dev			🟪	🟪	🟪

	Week6	Week7	Week8	Week9	Week10
Deliverable 3	⬛	⬛	⬛		
Backend Dev	🟪	🟪			
Frontend Dev		🟥	🟥	🟥	
Deliverable 4				🟦	🟦

Phase one is where the entire team will come together to brainstorm the scope of the application and come up with requirements that development teams can refer to later on. Once the requirements are set, the design team lead by **Ziyu Zhou** and the backend development team lead by **Jiaxin Liu** will simultaneously start working according to the requirements. When the design for the frontend infrastructure is inplace and got the tick of approval from all members, the frontend development team lead by **Jing Deng** will proceed to build out the frontend infrastructures. Deliverable reports throughout the development are expected to be completed by the entire team and will roughly take 2 weeks to complete.

However, as our development team are still in the early stages of the project, future changes and alternation of the requirements or schedules are extremely likely, thus the progression plan above only serves as a rough estimation.