SENG3011(22T1) Deliverable 2
Design Details

Group - 1 Group 2 Group 3 Group 4

Joel Ivory (z5162834)
Hoshang Mehta (z5255679)
Lachlan Sutton (z5308261)
Jiaqi Zhu (z5261703)
William Wan (z5311691)

Table of Contents

Part 1 - Tech Stack

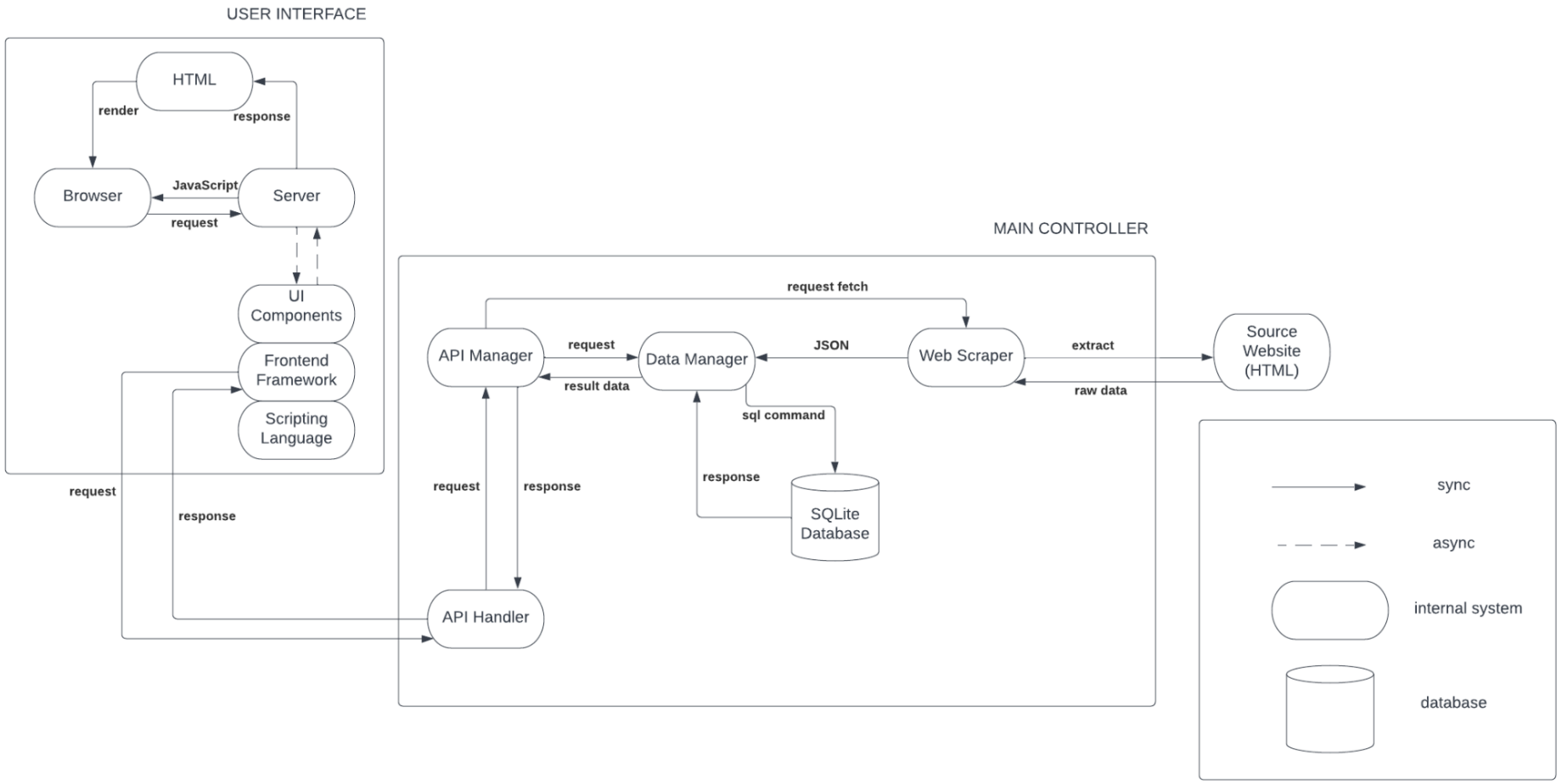
1.1 Updated Tech Stack Table

There have been dramatic changes to our chosen tech stack over the past few weeks since we worked on the project. Below is the updated summary table of the tech stack that we are using for deliverable 2. We will talk more about the reasons for the changes in later sections.

Figure 1. Summary of Tech Stack

Project Components		Languages / Frameworks / Platforms	Suitable Libraries
Backend programming	Web Scraper	Python	Beautiful Soup
	Database	SQLite	
	RESTful API	Django	Requests
	Documentation	Swagger	JSON
Frontend programming	Web Design & User Interface	HTML, CSS, JavaScript	React React Redux Materials UI
Testing	Unit Testing	Python, Pytest	-
	User Acceptance Testing	Python	-
Deployment	Hosting service	AWS Elastic Beanstalk	-
Project Management	User Requirements	Jira	-
	Software Architecture	UML	-
	Task & Issues Management	Jira	-
	Project Roadmap	Jira	-
	Centralisation of Documentation & Product Requirements	Confluence	-
Deployment/ Testing Environment	Windows		

1.2 Architecture Diagram



Part 2 - Backend

2.1 Scraper

2.1.1 Design Solutions

For web scraper, we used the **web scraping library: Beautiful Soup 4 and its dependencies (requests library, html.parser)**. The scraper code is in **scraper.py** under the directory /PHASE_1/API_SourceCode/. A branch called **scraper** is used to develop, maintain and test the scraper.

The resulting output of the **scraper.py** file is a **data.json** file, containing a complete json object of all data collected from the World Health Organisation URL (https://www.who.int/emergencies/disease-outbreak-news). There are three sets initially defined to be utilised by the scraper; '*disease_set*' contains a list of known diseases that could be in an article, '*syndrome_set'* similarly contains a list of syndromes that could be in an article and lastly '*months*' contains a list of each month in the year.

In terms of additional libraries and dependencies used by the scraper, we have imported and utilised **requests**, **pandas**, **json**, **time** ('strptime' in particular) and **calendar**. To assist the function of the scraper, we have defined a '*dateConverter*' function to turn our scraped dates into the format '*yyyy-mm-dd* or *yyyy-mm-ddT00:00:00*' to match the date-time format that our input takes.

The web scrapers' procedure is as follows:

1. Iterate through each page (from 0..n) using *requests.get()* function and perform the following at each page:
2. For every outbreak title listed, the scraper will parse and split the title and relevant information with try-except methods to handle various edge cases.
3. The captured data is then stored in various dictionaries as follows:
    a. *'locs_list'*: stores a list of location objects which include the country and location data for an outbreak, the list sits within the *'report'* dictionary.
    b. '*report':* contains the diseases, syndromes, event date and location list of the relevant outbreak article. Report objects relevant to the same article are stored in a Reports List within each '*Article*' dictionary.
    c. '*article':* contains all core information scraped from each article listed on the WHO page. This includes; the URL, date of publication, headline of article, main text body in article and a list of reports.
4. Ultimately, the above dictionaries are captured as a list of articles which is then written to a **data.json** file and ready to be loaded into the database.

2.1.2 Alternatives

We choose Beautiful Soup 4 as our library of choice due to the ease of use that it provides. The two other competitors that we considered were **Scrapy** and **Selenium**.

Beautiful Soup 4 has many helpful functions that make data extraction from websites much easier than if we had used the other two options. One such function is the html.parser that is

used in conjunction with Beautiful Soup 4. This function allows us to easily extract specific information from a website.

In addition, Beautiful Soup 4 is extensively documented and used throughout the community. This gives us an extensive amount of documentation and past questions to use if we need help.

This library also has an active community, which we can call upon if we are confused about any of the functions of the library. This further eases the use of the library.

Beautiful Soup 4 also has a shallow learning curve, due to it having many inbuilt functions that make extracting data easier than the other two (Scrapy Vs Selenium Vs Beautiful Soup for Web Scraping., 2022).

2.1.3 Drawbacks of the Chosen Solution

Although Beautiful Soup 4 is easy to use and well documented, there are some downsides to using it in comparison to the other options.

While Beautiful Soup 4 excels in ease of use, it lacks in methods of customisation. Scrapy's architecture is far more suited for more complex projects and advanced users due to the ability to add custom functionality.

In addition, through research we have found that Beautiful Soup 4 is the slowest scraper out of the 3 options, with Scrapy the fastest and Selenium second. To combat the slower scraping speed, multithreading can be utilised but it is a more complex process.

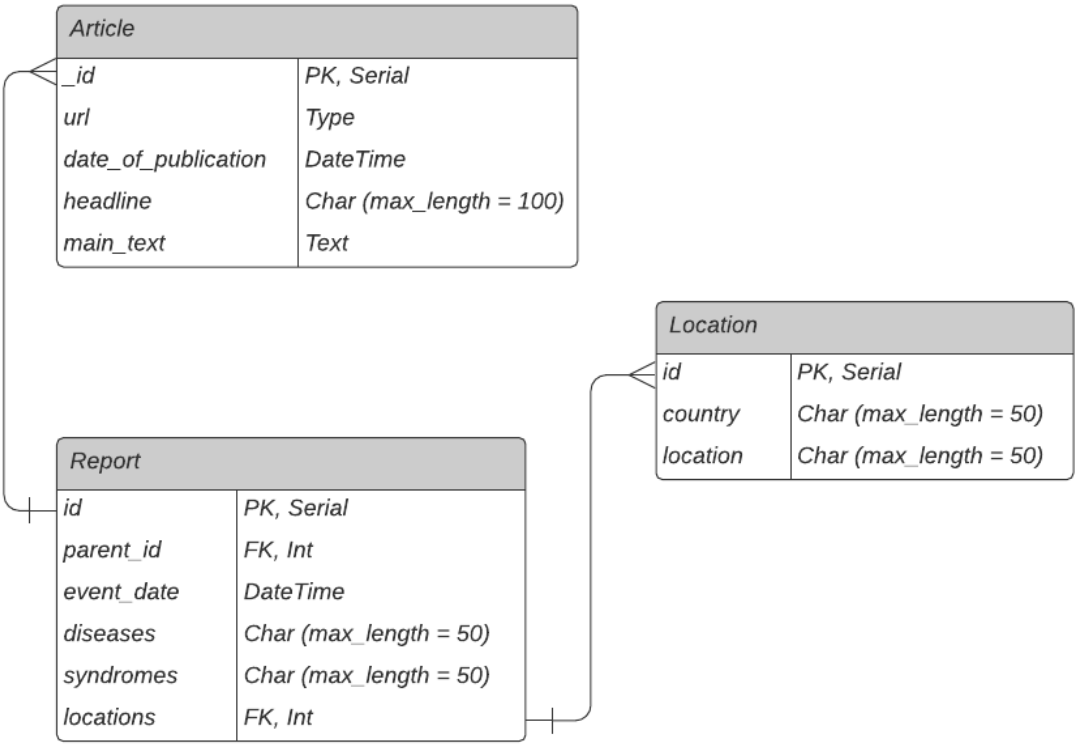
Beautiful Soup 4 also has many required dependencies. This may limit the system types that the scraper can run on, as a depencandy might be incompatible with a system.

2.2 Database

2.2.1 Design Solutions

We decided to use the **Django supported SQLite ORM** for implementing the database. We think SQLite is easy to learn, easy to use and good enough for the API as we expect the API to handle low to medium traffic HTTP requests (mostly they are GET requests) and the database to store only a couple hundreds of pages of data.

The initial design for the database structure is shown in the ER diagram below. We assume every report will only have one disease, one syndrome and one location, which may not always be the case. However, doing so significantly reduced the time for designing the API endpoint, making filters, and debugging for connecting the database to the API. We also made assumptions for the text length and nullability of attributes, which can be found in /scraper/models.py. Eventually, we will update the database structure and SQL schemas to satisfy more complex input datasets.



2.2.2 Alternatives

SQLite implementation is at similar difficulty as the other relational database management systems such as **PostgreSQL** and **MySQL**. While all these mainstream DBMSs are compatible with and supported by Django, the main advantage of using SQLite is its lightweight and fast performance.

Potentially, we could implement a NoSQL database and we thought about **MongoDB**. However, MongoDB is known to lack the support for joins as a relational database, and we will need joins to handle filtering through multiple related tables (for later deliverables), thus it's not an ideal solution (Understanding the Pros and Cons of MongoDB, 2022).

2.2.3 Drawbacks of the Chosen Solution

The drawback of SQLite is that it's not suitable for high-volume websites and very large datasets, and it also cannot handle high concurrency for writing as it will only allow one writer at any instant in time. This will potentially restrict the scalability of this project (Appropriate Uses For SQLite, 2022).

2.3 API Framework

2.3.1 Design solutions

Django App

Django is a python-based open source framework that is ideal for our API design. Django's ability to be fully functional with the REST API architecture is one of the reasons it was in

consideration, as well as the decent amount of documentation online and support for any bugs.The rest framework has a built-in API browser for testing endpoints and Django with rest framework provides facility of authentication and permission rules with the framework, which makes it an ideal solution for our project. Django is also relatively immune to cyber attacks, especially SQL injection which is again a huge bonus for us since we have to be running SQL queries securely and don't want data manipulation.

2.3.2 Alternatives

Fast API
FAST API is a web framework for python that is used for developing REST API. Although FastAPI has many advantages of Django like how it helps in validating the datatype from the developer even in nested JSON requests, it is a relatively new framework and there is less documentation online. Therefore FAST API is not the best design solution in regards to availability of documentation or help from the community in case the team runs into a bug or a problem, which isn't a problem for Django. Another disadvantage of the FAST API is that in the development of the applications we need to tie everything together in the application which causes the main file to become very long or crowded. This isn't a problem in the Django framework and thus makes it a lot more approachable as a design solution.

Flask
Flask is a python web framework that also manages HTTP requests. Although Flask is the easiest to pick up and use as compared to FAST API and Django, Flask doesn't have the same amount of functionalities as Django does, and is too simple of a framework to be considered as a viable use case. Flask's network applications have a single source that needs to handle the request one after another regardless of multiple requests at a time which makes the application more time consuming, which is not a problem for the Django framework. Thus, despite Flask's approachability, Django is the better framework to use in our case.

2.3.3 Drawbacks of the chosen solution

Django unfortunately does have some minor drawbacks such as Django frameworks having no conventions, often when configuring "on-the-go" the components are mismatched, although this is a very minor problem and can be overlooked. The requests for every individual process also makes the Django development process slower, but again this is very minor and not too drastic.

Part 3 - Deployment

3.1 Design Solutions

We decided to use **Amazon Web Services (AWS)** for deployment of our solution. This choice was made as AWS has great features and pricing for small-scale development and deployment, which fit our needs well. Additionally, AWS is extremely widely adopted, meaning that any issues we may have had during the process of deployment would be easy to research and solve.

3.2 Alternatives

Due to the current demand for web applications, there is a vast range of deployment options for these applications. Because of this, care must be taken in the choice of deployment service so that the best service is chosen. Below is a list of the considered deployment services for our solution:

Replit:
Replit is a commonly used service for both development and deployment of applications, offering tools suitable for the entire development process. Due to this, some members of the team have some past experience with using Replit. Unfortunately, however, the tool proved to be fairly unreliable and unsuitable for collaborative development. For example, the tool would frequently crash while multiple people were using the development environment at once, meaning large amounts of data loss were fairly common.

Azure:
Azure is a deployment service created by Microsoft, and is slowly becoming the most largely adopted deployment services for web applications. It is exceptionally useful as a deployment service, providing a complete tool suite to provide the developer full control over the deployment of their app. Due to this control, however, Azure is more of an expert-compatible suite in both pricing and verbosity than one facilitating a simple deployment process.

AWS:
AWS is by far the most widely adopted deployment service, providing both great features and pricing for first-time users. Due to its wide adoption, any issues that a developer may come across are usually well documented and thus easy to solve. However, due to the wide range of features, AWS is fairly impenetrable for a first-time user, and has a level of control that leads to a more complex and harder to understand experience.

Heroku:
Heroku is a simple deployment service that is widely used for free deployment. It is relatively easy to set up and use, but unfortunately does not have an option for long-term storage, and so is unsuitable for our scraper system.

GitHub pages:
GitHub pages is an attractive deployment solution due to its ability to run directly on a GitHub repository, which we are already using for our development. GitHub pages is also extremely widely adopted, allowing much more support for any potential issues we might have in development. However, there is no option for external storage in GitHub pages, and so this deployment service is unsuitable for our scraper solution.

3.3 Drawbacks of the Chosen Solution

As written above, AWS is a very advantageous service for our deployment, but there are some considerable drawbacks to the service. Due to the wide range of options that AWS offers, it is difficult to know which option to choose for our specific use-case. This leads to considerably increased setup and deployment maintenance time, as extensive research must be done to learn which option best suits our needs.

Additionally, to improve the consistency of the service, there are strict requirements on the setup of a deployment environment, with configuration files being required for specific

features. Unfortunately, however, these are scarcely documented in the deployment centre of AWS, and so external research must be done to find out which of these to use.

Part 4 - Frontend

4.1 Design Solutions

We chose to develop the frontend by creating a **React app** and programming in **HTML, CSS and JavaScript**. We expect to use **React Redux** and **Materials UI** these libraries for managing states and UI components at frontend. We aim to build a single-page application with React.

4.2 Alternatives

The two alternatives we considered for frontend are **Vue.js** and **Angular**. These two are both popular frameworks for web development. Since this project is fairly small and doesn't require admin functions such as register and login, we consider ease of use over performance when selecting the framework for frontend.

Both Vue.js and React are suitable for building single-page applications. However, we prefer React because it is simple to use by following the tutorial and we can create elements that contain HTML and JavaScript at the same time with React JSX, while Vue is a bit more complex to set up and has limited plugins available online.

For Angular, since it has the most complex project structure out of the three, and seems to be overpowering for this project, we ditched the idea. Another reason is because none of us can program in TypeScript or have knowledge about this framework, which will potentially increase the time for us to debug for frontend if anything goes wrong.

4.3 Drawbacks of the Chosen Solution

For this project, React doesn't have any significant cons and other minor drawbacks are yet to be determined in the development for the project.

Part 5 - Challenges Faced

5.1. Web Scraper

In developing and implementing our web scraper, we faced two key challenges.

Firstly, within the report object defined within our scraper file, the '*event_date*' variable is currently being set to only the first date that isn't the publishing date within an article. We might need to develop some other algorithms to make sure the extracted event date is actually the date for the reported event.

Secondly, we encountered a challenging corner edge case error where the event date was extracted in the format of "20183-09-07" because of a superscript next to the date that was also extracted. As a result, an extra checking step is added to avoid extracting the superscript.

5.2. Database

Regarding the database, we encountered some challenges when trying to establish a MongoDB connection locally. The error said "...connection is forcibly closed…". Weirdly, we could directly import our JSON data in data.json as a collection into the database created on MongoDB, but we couldn't import using a python script and the above error occurred. It is unknown why these issues were occurring (even though our data structure seems fine) and it was hard to debug only based on the error message. Hence, the idea of using MongoDB was scrapped and we decided to pursue our next best alternative, SQLite, which has worked well so far.

Additionally, many changes for the database have been tied to the data structure that is written into the **data.json** file from the web scraper. As a result, if the scraper returns a different structure, the database must also be updated to accommodate the new structure. Similarly, the methods to stringify data from the database will also need to be changed.

Lastly, the complexity of web scraped data ultimately determines how we design the SQL schemas. When there are more tables, there are more relationships that need to be handled. This can greatly increase the complexity for filtering search results.

5.3. Deployment

During the scoping phase of deployment our team faced a multitude of options to choose from, which proved to be a significant challenge in getting our web application deployed. Secondly, upon selecting AWS as our method of deployment we faced the challenge of navigating AWS' extremely verbose and particular needs in order for the deployment to work. The folder structure required to run our software on AWS was drastically different to the one used in development, and so a system to convert one to the other was needed. Lastly, AWS is unable to use poetry and so package management had to be converted into another format.

References

1. Sqlite.org. 2022. Appropriate Uses For SQLite. [online] Available at: <https://www.sqlite.org/whentouse.html> [Accessed 19 March 2022].
2. Medium. 2022. Scrapy Vs Selenium Vs Beautiful Soup for Web Scraping.. [online] Available at: <https://medium.com/analytics-vidhya/scrapy-vs-selenium-vs-beautiful-soup-for-web-scraping-24008b6c87b8> [Accessed 18 March 2022].
3. knowledgenile. 2022. Understanding the Pros and Cons of MongoDB. [online] Available at: <https://www.knowledgenile.com/blogs/pros-and-cons-of-mongodb/#MongoDBTransactions> [Accessed 19 March 2022].