SENG3011(22T1) Deliverable 1
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 - The API Module

1.1. Architecture Design

The API module will be developed to a **resource-oriented** design, namely the REST (REpresentational State Transfer) approach.

1.2. API Endpoint Naming Conventions

In order to ensure ease of use and consistency of the API, some conventions should be used in the naming of endpoints.

URIs should refer to a specific object or group of objects rather than an action. This ensures that the address can support multiple operations. For example both a GET and DELETE request are valid on a URI of /reports/, but only GET is valid on /getReports/

As URIs represent a noun, they should aim to use as few words as is feasible. URIs should be lowercase when only containing one word and camelcase otherwise. For example, /reports/ and /diseaseReports/ are valid, but /Reports/ and /diseasereports/ are not.

Hierarchy in addresses should only be used if all higher level resources **fully** encapsulate all low level resources i.e. there should never be a resource in a higher level of the hierarchy that is more specific than one lower than it. For example, /reports/statistics/ must refer to statistics for reports **only**.

Part 2 - Passing Parameters and Collecting Results

2.1. Passing Parameters to the API

There are several parameters that are required to be sent to the API, including date range, disease name, location, top n amount of returns. They can be passed to the API as input in JSON format:

```
{
    "disease":"ebola"
    "date" : {
            "start": 1646031254
            "end": 1646117872
    }
    "location":{
           "country": "Spain"
           "city": "Madrid"
    }
    "maxReturn": 100
}
```

2.2. Asynchronous Operations

In some circumstances, such as high load or high complexity of a request, the API may take a considerable amount of time to respond to a request. This can be slow, as the client will wait on this request until it is completed.

To solve this, requests will be made in two stages. A request will be ‘created’ by issuing a POST request with JSON data to further specify the parameters of the request. This will queue an operation in the backend, and return a **request token**, which can be used to track the progress of the operation through GET requests. GET requests can be periodically issued until the operation is either completed or cancelled. Once completed, a response will be issued to the GET request containing the result of the operation. Requests may also be cancelled before completion by issuing a DELETE request with the request token.

2.3. Versioning

An important factor to consider when working with the API is version control. API versioning is integral to keeping the software stable as well as when a new system needs to be added and changes the current integration. Thus, versioning the API will also allow us to constantly monitor for changes and keep track of iterations that we can fall back on in case the system breaks. To do this with Django, URL versioning can be used to be able to keep up with the changes the team makes with each integration.

2.4. Response Codes

To be able to utilise the REST framework properly, proper status codes will be used to make our code more readable and easier to follow. Examples can be seen in the following:

Information 1xx:
`HTTP_100_CONTINUE`
`HTTP_101_SWITCHING_PROTOCOLS`

Successful 2xx:
`HTTP_200_OK`
`HTTP_201_CREATED`
`HTTP_202_ACCEPTED`

Redirection 3xx:
`HTTP_300_MULTIPLE_CHOICES`
`HTTP_301_MOVED_PERMANENTLY`
`HTTP_302_FOUND`
`HTTP_303_SEE_OTHER`

2.5. Sample Requests

2.5.1. Requesting an Article

Request:

```
POST /api/articles HTTP/1.1
{
   "disease": "ebola"
   "date" : {
            "start": 1646031254
            "end": 1646117872
    }
}
```

Response:

```
HTTP/1.1 202 Accepted
{
   "location": "api/articles/queue/12345"
}
```

Request:

```
GET /api/articles/queue/12345 HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
   "status": "pending"
   "results": []
}
```

Request:

```
GET /api/articles/queue/12345 HTTP/1.1
```

Response :

```
HTTP/1.1 200 OK
{
   "status": "complete"
   "results": [
      {
          "date": "2010-10-10"
          "title": "Useless Article"
          "content": "Lorem ipsum dolor sit amet...
              ...
              "
      },
      ...
   ]
}
```

2.5.2. Requesting a Report

Request:

```
POST /api/reports HTTP/1.1
{
   "disease": "spanish flu"
   "location":{
           "country": "Spain"
           "city": "Madrid"
    }
    "maxReturn": 100
}
```

Response:

```
HTTP/1.1 202 Accepted
{
   "location": "api/articles/queue/12345"
}
```

Request:

```
GET /api/reports/queue/12345 HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
   "status": "pending"
   "results": []
}
```

Request:

```
GET /api/reports/queue/12345 HTTP/1.1
```

Response:

```
HTTP/1.1 200 OK
{
   "status": "complete"
   "results": [
      {
          "date": "2010-10-10"
          "title": "Useless Report"
          "content": "Lorem ipsum dolor sit amet...
              ...
              "
      },
     ...
   ]
}
```

2.5.3. Internal Server Error

Request:

```
POST /api/reports HTTP/1.1
{
   "disease": "spanish flu"
   "location":{
           "country": "Spain"
           "city": "Madrid"
    }
    "maxReturn": 100
}
```

Response 500:

```
HTTP/1.1 500 Internal Server Error
```

2.5.4. Unknown Endpoint Error

Request:

```
POST /unknownEndpointError HTTP/1.1
{
   "unknown": "1"
}
```

Response 404:

```
HTTP/1.1 404 Not Found
```

2.5.5. Invalid Request Type Error

Request:

```
POST /api/reports HTTP/1.1
{
   "unknown": "1"
}
```

Response 400:

```
HTTP/1.1 400 Bad Request
```

Part 3 - Implementation Details

Our team dedicated an entire working session towards determining the right technology stack for our project. We first conducted an assessment of our past experiences to gauge what programming languages, API frameworks, libraries, deployment environments and project management tools we were all the most confident with.

Figure 1. Survey of team members past experience

Tech stack	Hoshang	Jiaqi	Joel	Lachlan	William
Backend	Python	Python	Python Java	Python Java	Python
Database	PostgreSQL SQLite MySQL	PostgreSQL SQLite	CSV PostgreSQL	PostgreSQL	PostgreSQL SQL
API Framework	Flask	Flask, FastAPI	Flask	Flask	Flask
Frontend	HTML CSS JavaScript	JavaScript React HTML CSS	HTML CSS JavaScript	JavaScript, React	JavaScript React HTML CSS
Deployment	-	-	-	Repl.it	Repl.it
PM tools	GitLab Trello Notion	GitLab Trello Notion	Trello	GitLab Trello	Gitlab

As shown in the table above, Python was found to be the most common language of expertise amongst our team members for backend programming. In addition to this experience, our team considered the availability of Python libraries such as ‘NumPy’, ’Pandas’, ‘BeautifulSoup’ and ‘Scrapy’ which could be significantly useful for implementing the web scraper and managing data resources in our project.

Our team also brainstormed and came up with a list of other possible options which we can use for the project. Combined with the tech stack we already have experience with, below is a summary of the possible tech stack we could use for the project.

Figure 2. Summary of Possible Tech Stack

Project Components		Languages / Frameworks / Platforms	Suitable Libraries
Backend programming	Web Scraper	Python, Scrapy	Beautiful Soup
	Database	PostgreSQL, SQLite, MongoDB, MySQL, CSV	Pandas, NumPy
	RESTful API	Django, Flask, Fast API	Requests
Frontend programming	Web Design & User Interface	HTML, CSS, JavaScript, Vue.js, Angular	React React Redux Materials UI
Testing	Unit Testing	Python, Pytest, Testify	-
	User Acceptance Testing	Python, Robot framework	-
Deployment	Hosting service	Repl.it, Heroku, GitHub Pages, AWS	-
Project Management	User Requirements	Jira	-
	Software Architecture	UML	-
	Task & Issues Management	Jira, Kanban board, GitHub	-
	Project Roadmap	Jira	-
	Centralisation of Documentation & Product Requirements	Confluence	-
Deployment/ Testing Environment	1 MacOS, 1 Linux, 3 Windows		

We paired the results above with our research into the most appropriate languages and libraries for web scraping, API and insights application development. Eventually, we filtered the options and finalised on the tech stack that we want to use in this project.

Figure 3. Finalised Tech Stack for the Project

Project Components		Languages / Frameworks / Platforms	Suitable Libraries
Backend programming	Web Scraper	Python	Beautiful Soup
	Database	CSV	Pandas, NumPy
	RESTful API	Django	-
Frontend programming	Web Design & User Interface	HTML, CSS, JavaScript	React React Redux Materials UI
Testing	Unit Testing	Python, Pytest	-
	User Acceptance Testing	Python, Robot framework	-
Deployment	Hosting service	AWS	-
Project Management	User Requirements	Jira	-
	Software Architecture	UML	-
	Task & Issues Management	Jira, Kanban board	-
	Project Roadmap	Jira	-
	Centralisation of Documentation & product requirements	Confluence	-
Deployment/ Testing Environment	1 MacOS, 1 Linux, 3 Windows		

3.1. Backend

Since we decided to use **Python** for backend development, all the frameworks and libraries chosen have to be compatible with Python.

3.1.1 RESTful APIs

Our team picked **Django** for development of the RESTful APIs. Why Django? According to Net Solutions, Django is better to use over Flask when planning to build robust APIs with the REST framework (Goyal, 2021). Additionally, it will have a faster development time (not needing to develop from scratch like with Flask) which is required due to the fast-paced nature of Trimesters.

3.1.2 Web Scraper

Our team decided upon **BS (Beautiful Soup)** as our web scraper, due to a number of positive reasons. The first one was its ease of use, as from viewing a few reference programs implementing BS as a web scraper, it only required a few lines of code to start working. Another big plus for BS is its large popularity in the python package community, which allows our team to easily be able to find solutions to problems we may encounter with BS through many articles online.

Compared with Scrapy, which is a more complex framework, BS as a Python library is more beginner-friendly. Although Scapy is said to be faster than BS in performance, BS is considered to be a better choice for smaller projects (Choudhury, 2020).

3.1.3 Database

Since this project only requires scraping from one web source, we chose to use CSV files as the ‘database’ to store the data and use the Python libraries **pandas** and **NumPy** to operate on the data.

3.2. Frontend

The main reason why we chose **React** for frontend development is because it makes creating dynamic websites easy with JSX (JavaScript Extension), which allows us to render specific components. This makes React perfect for making an efficient one page website and the best choice for frontend for this project. Aside from that, our team leaned more towards React because it has an easier learning curve compared to the others. As suggested by one of our team members, we will also use **React Redux** for easier state management at frontend.

All UI component libraries are relatively similar in terms of installation and usage, so our criteria revolved around the levels of customisation and aesthetics that the library provided. We chose **Materials UI** as the UI component library of choice because it provides an accessible, aesthetic and customisable library of reactive components that we can use at our disposal. MUI also contains many UI widgets that will allow us to represent data effectively with appropriate visual aids.

3.3. Development and Deployment Environment

For development, our team will be developing on local machines, in a Windows environment. We chose Windows as our development environment due to its compatibility with the frameworks we plan to use and the fact that all group members natively run windows on their local machines. As a result, Windows is the most convenient development environment for the project.

As shown in figure 3, our web application will be utilising a hosted service, namely **Amazon Web Services (AWS)** for deployment.

We had to choose between a few hosting services as shown below:

Heroku
Heroku provides a free service to host websites in the form of dynos which give users a number of free hours to host per month (Heroku Platform Reviews & Ratings 2022, 2022). While Heroku provides a free service, there isn't much documentation regarding hosting and no one in the group has had any prior experience with using it. In addition, Heroku's hosting can often be slow and inconsistent.

Github pages
While it provides a free hosting service, Github pages does not allow us to use a database. This makes it unsuitable for the project as the group has planned to use SQL as the primary storage method.

Repl.it
Eventhough Repl.it provides a free hosting service, it takes a considerable amount of time to start up the website. In addition, through anecdotal experience, some members of the group have experienced frustrating bugs relating to collaborative work and hosting with this service.

Amazon Web Services
AWS provides a fast and free hosting service. On top of this, many of the group members have had experience with hosting a website on AWS, which makes it easy to use.

3.4 Testing

Our team concluded that **Pytest** would be the best unit testing framework as it remains as the most popular python testing package due to its ease of use and large usage of various exceptions (pytest: helps you write better programs — pytest documentation, 2022).

For user acceptance criteria testing, our team decided upon using the **Robot Framework** to conduct these tests. This is because Robot Framework is very versatile as it allows the user to perform command-line, web and GUI testing. This will also help test our Django Framework to assure us we are on the right path.

Part 4 - Software Architecture Diagram

Figure 4. Software Architecture (with components)

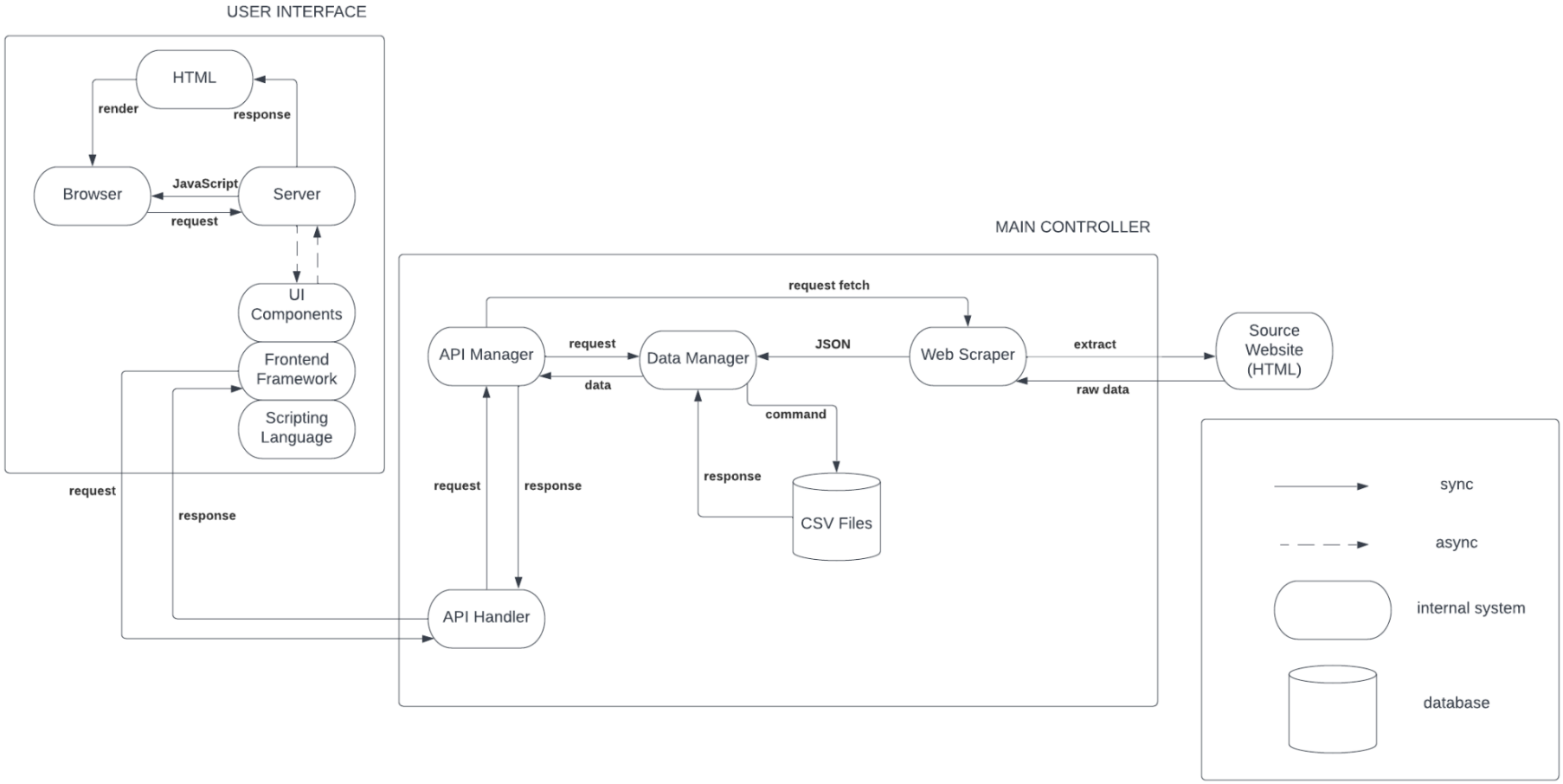
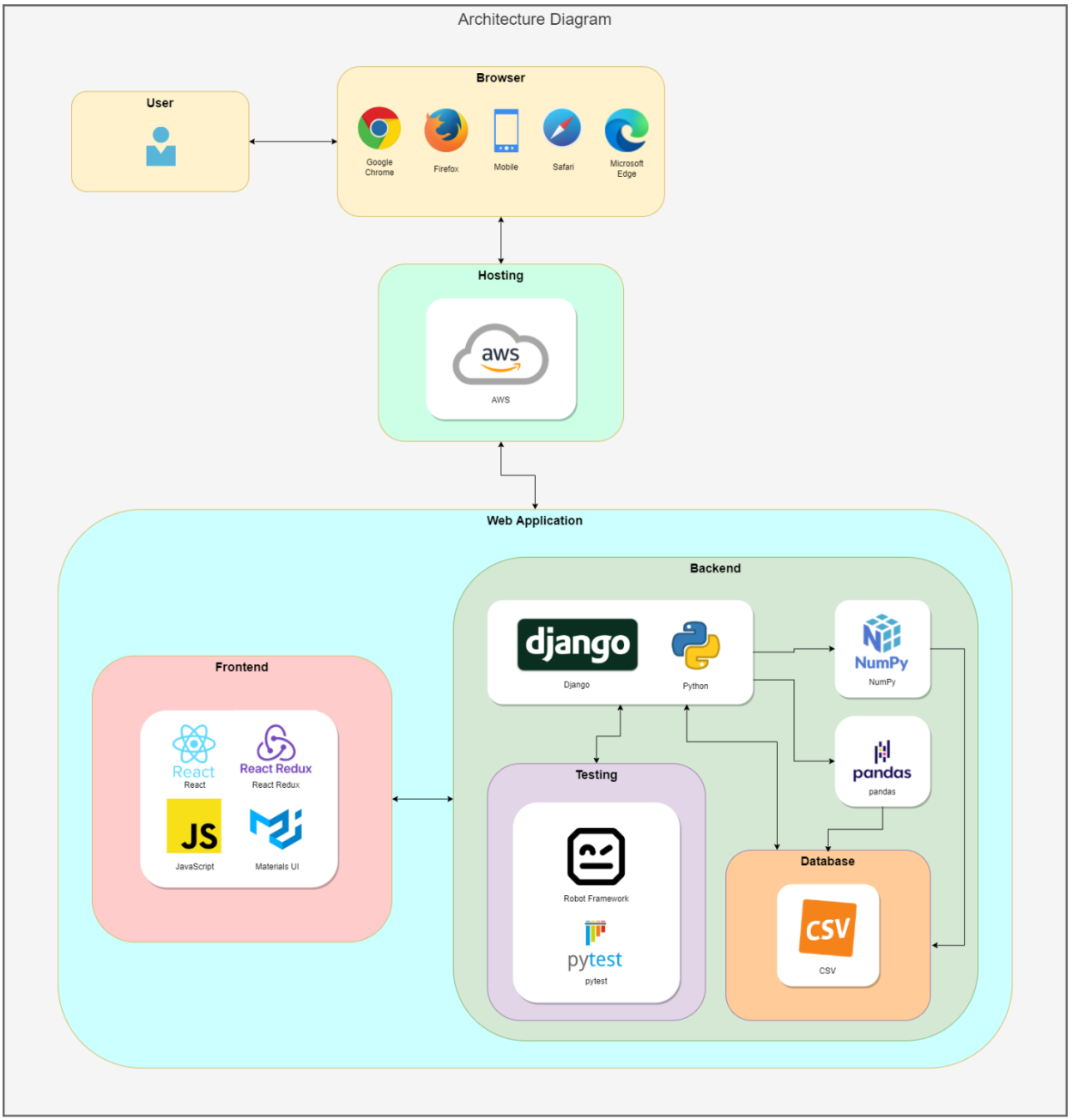


Figure 5. Software Architecture (with tech stack)



References

1. Choudhury, A., 2020. *Scrapy VS BeautifulSoup: A Comparison Of Web Crawling Tools*. [online] Analytics India Magazine. Available at: <https://analyticsindiamag.com/scrapy-vs-beautiful-soup-a-comparison-of-web-crawling-tools/#:~:text=Scrapy%20is%20an%20open%2Dsource%20framework%2C%20whereas%20Beautiful%20Soup%20is,the%20developer%20what%20they%20need.> [Accessed 3 March 2022].
2. Goyal, S., 2021. *Flask vs Django in 2022: Which Framework to Choose and When?*. [online] Insights - Web and Mobile Development Services and Solutions. Available at: <https://www.netsolutions.com/insights/flask-vs-django/> [Accessed 3 March 2022].
3. TrustRadius. 2022. *Heroku Platform Reviews & Ratings 2022*. [online] Available at: <https://www.trustradius.com/products/heroku-platform/reviews?qs=pros-and-cons#overview> [Accessed 3 March 2022].
4. Docs.pytest.org. 2022. *pytest: helps you write better programs — pytest documentation*. [online] Available at: <https://docs.pytest.org/en/7.0.x/> [Accessed 3 March 2022].