SENG3011 Engineering Workshop 3

Deliverable 1

ANALYTICS PLATFORM FOR PREDICTING EPIDEMICS: The Project Plan

By team4masters

Navid Bhuiyan (z5260384), Jin-Ao Olson Zhang (z5211414), Adrian Borjigin (z5205791), Aaron Guan (z5208046), Kieran Nguyen (z5208469)

Contents

1.1 Project Background

1.2 Project Specification

1.3 Initial Design Considerations
- 1.3.1 Overall Software Architecture
- 1.3.2 API Design
  - 1.3.2.1 API Usage
- 1.3.3 Scraping Service Design
- 1.3.4 Database Design
- 1.3.5 Platform Design

1.4 Project Management
- 1.4.1 Project Roadmap and Kanban Board
- 1.4.2 Role Delegations
- 1.4.3 Agile Practices

1.5 Conclusion

1.1 Project Background

The Integrated Systems for Epidemic Response (ISER) which is a NHMRC Centre for Research Excellence located at UNSW, has requested to design a scraping, storage, search and reporting system to automate outbreak detection and surveillance. As part of the first deliverable, the report will break down our software and platform design that fit the major aspect of the report gathering and search aspects of the project specification. Our project management practices are also described here to allow for flexibility if requirements or our understanding of the project changes.

As for now, we mainly have designed our system around the use of the cloud, Amazon Web Services (AWS) as part of our initial design to allow for modular, flexible development and scalability.

1.2 Project Specification

The project outlines a web application to automate reporting to assist with epidemic and pandemic surveillance. Also with the project expanding to a web application extracting analytics and insight from different third party sources such as social media, media outlets and so on, the system needs to be designed in order to deal with large loads of random, inconsistent and unstructured data and be able to create insights on top of it.

Keeping this in mind, realising the project on top of our project specification, we have interpreted additional requirements:

- Application needs to be able to handle large data throughput
- Requires the ability where the system can adapt to different types of data for insights
- Speed and reliability are both equally important to provide insights
- The need to easily scale (especially horizontally) is important as well

With these additional considerations, we focus on a system where it is possible for modular development.

1.3 Initial Design Considerations

1.3.1 Overall Software Architecture

Analysing the specification, and the need to develop a scalable application for fast and reliable insights, the entire software stack will be based on a container driven architecture on a Iaas and PaaS platform. Why are we specific about using this, is for one it balances both control, configuration and scalability to allow the entire app to be developed to suit the needs of the project. This approach also balances the viable performance the development team can get out of it per resource, providing a way to optimise and minimise costs. AWS EC2

Container service was chosen as the overall platform, as it provides a great balance of pre-configuration for scaling and customizability for application development, as it is compatible with a lot of the major web app technologies (both frontend and backend) we have chosen for this application.

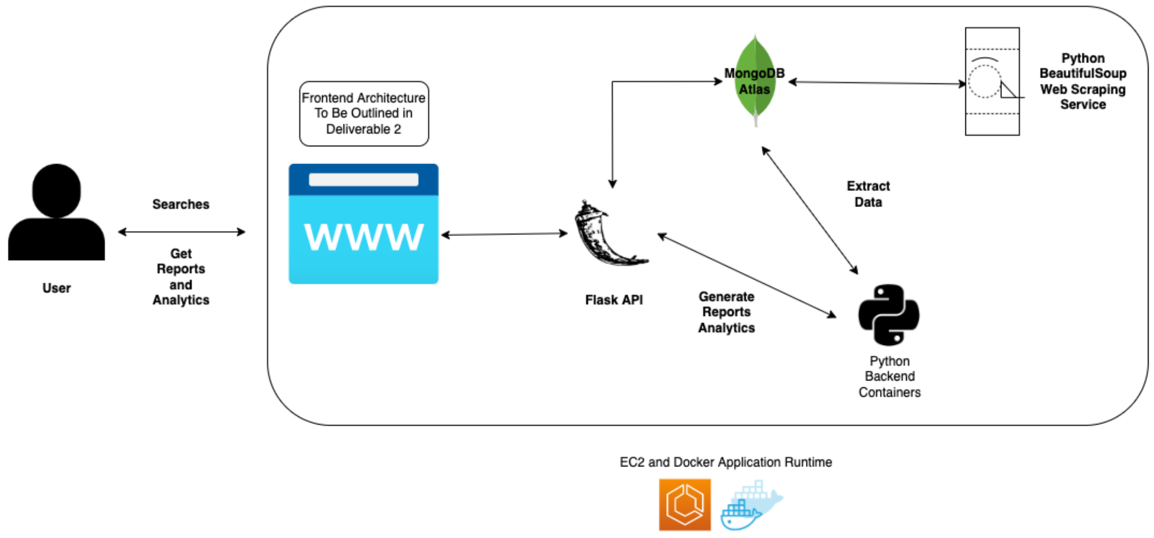

User
Searches
Get Reports and Analytics
Frontend Architecture To Be Outlined in Deliverable 2
WWW
Flask API
MongoDB Atlas
Python BeautifulSoup Web Scraping Service
Extract Data
Generate Reports Analytics
Python Backend Containers
EC2 and Docker Application Runtime

Image 1. Backend Architecture Diagram (excluding frontend architecture).

To comment on committing to a Docker and EC2 architecture, this ensures its native support on AWS Elastic Container Service (ECS). This also allows the use of optimised images only which provides less overhead, dedicating resources to a task. Refer to Platform Design for more information in considering this platform architecture.

Down below in the table provides an overview of the languages, software, runtimes and we shall be using in the project. For the majority of our backend, we shall be dedicating to using Python, due to its powerful application in data analytic, artificial intelligence and overall usability in creating internet services. This simply allows us to expand our services later down during development

Table 1. Overview of the utilised backend technologies.

Technology	**Version**	**Reasoning**
Python 3	3.8.12	Most stable version of Python, with the maximum compatibility with extensive libraries such as BeautifulSoup, Numpy, Pandas, OpenCV, TensorFlow etc. Run's best on Dockerized Applications

Docker Runtime	Python 3.8.12-slim	Lightweight runtime which enables the use of Python and dedicated resources for only Python related tasks.
Flask	2.0.3	Most latest and compatible release of Flask which can run on Python 3.8.12. Also provides a light framework to develop our API.
MongoDB Atlas	MongoDB 5.0	SaaS available to be hosted on AWS ECS, which allows to customise scaling and the structure of data using a readable frontend platform.

In the topics further below we expand our reasoning of the different aspects of the app.

1.3.2 API Design

Considering the overall design of our API, even if it is defined in the spec, there are a few modifications and additions created, since we believe there are additional endpoints which are useful to collect user information, particularly search history to enable the user to keep track of what they had extracted and researched.

Down below are the design considerations of the API and its possible endpoints.

Table 2. API Documentation for Possible Endpoints

Route name	**Route description**	**CRUD Method**	**Parameters**	**Return values**	**Exception**
/search	The user provides the search terms as query strings to find the disease reports related to given search terms. Search terms must have - Period of interest - Key terms - Location - (optional) Timezone,	GET	(key_terms, location, Start_date, End_date, Timezone)	{[report_json]}	Value Error when - Start date is later than End date - Date does not followed the format - key terms does not followed the format

	default to UTC if not provided Period of interest contains - Start date - End date				
/search/frequent-key	Show the top five key terms searched by users	GET	N/A	{keys}	
/search/frequent-key/update	once have new search history, update the frequently key	POST	(keys)	{is_success}	
/search/history	Records the last five searches	GET	N/A	{records}	
/healthcheck	Indicates the aliveness of the service -Do all functions are normal -Do we have enough database usage	GET	N/A	{is_alive}	

Table 3. API Documentation for Variable Names

Variable name	**Type**	**Variable name**	**Type**
key-terms	string(AAA,BBB,CCC)	Start_date/end_date	string(yyyy-mm-ddTHH:mm:ss)
Timezone	string	keys	list of string
records	list of string	is_success	boolean

report_json	JSON format Object	is_alive	HTTP response code (200 if OK, 503 if service is unavailable)

1.3.2.1 API Usage

Primary functionality of the API is searching for reports through a GET request to the API

- Input arguments get passed as query strings/ parameter (period of interest, key_terms and location)
    - Query strings are used as the inputs will all be simple strings or arrays of strings rather than more complex objects that need to pass more information in (requiring json object), which could require a body. Alternative would be to pass in a json withpath the arguments but this seems unnecessary.
- Queries the database for articles that contain reports in them, then forms a json in the backend to be returned to the user
- Returns 200 if passes, 400 if no date inputs or syntax errors found, 204 if no articles are found.
- **Sample API Call:**
    - GET http://127.0.0.0:8080/search?key=zika,outbreak&location=sydney,NSW&start_date=2015-10-01T08:45:10&end_date=2015-11-01T19:37:12&timezone=+01:00
- Key features:
    - Input parameters have been passed in as query strings in the URL.
    - Key terms and location can have multiple terms, which should be separated by commas to allow them to be parsed separately in the backend of the API.
    - Start and end dates must follow the yyyy-MM-ddTHH:mm:ss format.
    - Timezone is an optional parameter based on UTC time, defaulting to UTC+00:00 and ranging from -12:00 to +14:00

Other routes provide additional information about the service, such as finding the most frequently searched key terms, in order to assist developers implement further functionality with the API, such as having a headline item for frequently searched terms.

Additional routes may be added depending on development time. Currently, API functionality has been limited to the barebone necessities of the specification, however may be extended in further iterations. Possible extensions may include geolocation to expand API searches (e.g. including reports that mention “Sydney” when the user searches for New South Wales) or running statistics when calling the healthcheck endpoint (as opposed to simple HTTP response).

Naming Conventions

URIs have been kept as simple nouns and verbs to provide clarity of functionality for each method. All characters are lowercase for simplicity, words separated by hyphens.

Django vs Flask

There are two python frameworks used to develop API which are Django and Flask. Django is a full stack framework used for quickly developing concise apps.

Django is designed to minimise actual coding during the app design process. Instead of writing code, developers can define options with forms, tables, and other interfaces.

Flask is a micro-framework, meaning that it's designed to perform a more limited role than a full-stack framework like Django.

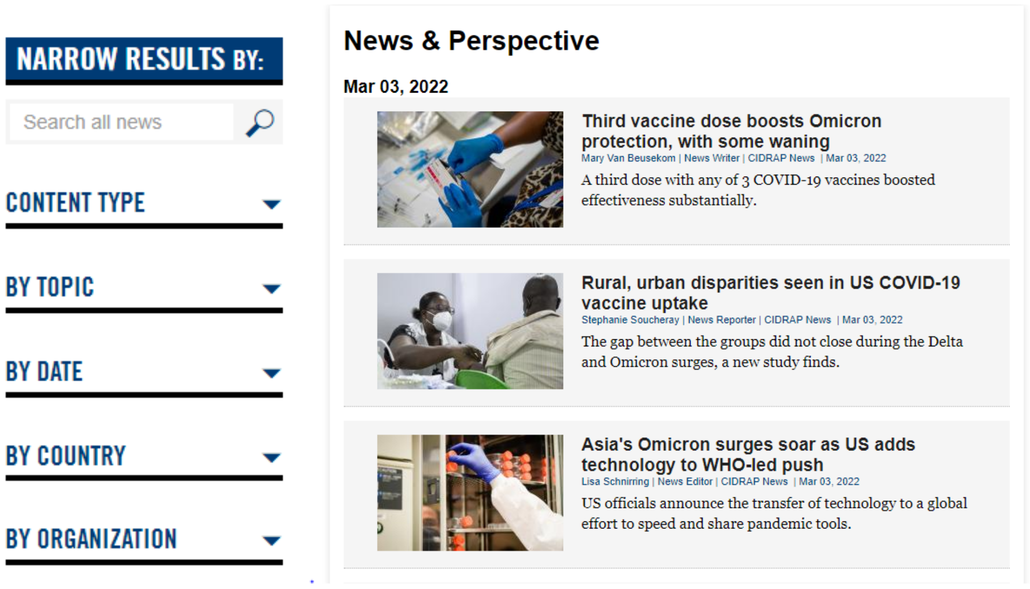
Table 4. Django vs Flask

Framework	**Advantages**	**Disadvantages**
Django	Fast, full featured, scalable, versatile, and secure.	Steep learning curve and not suitable for small projects.
Flask	Flexible database support for frameworks such as NoSQ, microframework and better support for API development.	Not suitable for complex systems and less overall support than Django.

We deemed Flask as more suitable for our project due to the greater support for our chosen architecture (NoSQL, more support for API development) and ease of use, as all team members have used Flask in COMP1531, reducing the learning required to complete the API to allow for faster development. Given the time constraints and sprint-based nature that we have outlined in our timeline, we believe the simplicity of Flask will allow us to provide additional features and focus on proper functionality to mitigate the disadvantages of not having Django’s full featured capabilities.

1.3.3 Scraping Service Design

We have chosen to use https://www.cidrap.umn.edu/ to scrape our data as the articles on this website are all clearly categorised by date, disease topics and by country. While the site contains some dynamic elements, the articles are all clearly displayed in HTML, which makes it easy for us to extract the data.



News & Perspective

Mar 03, 2022

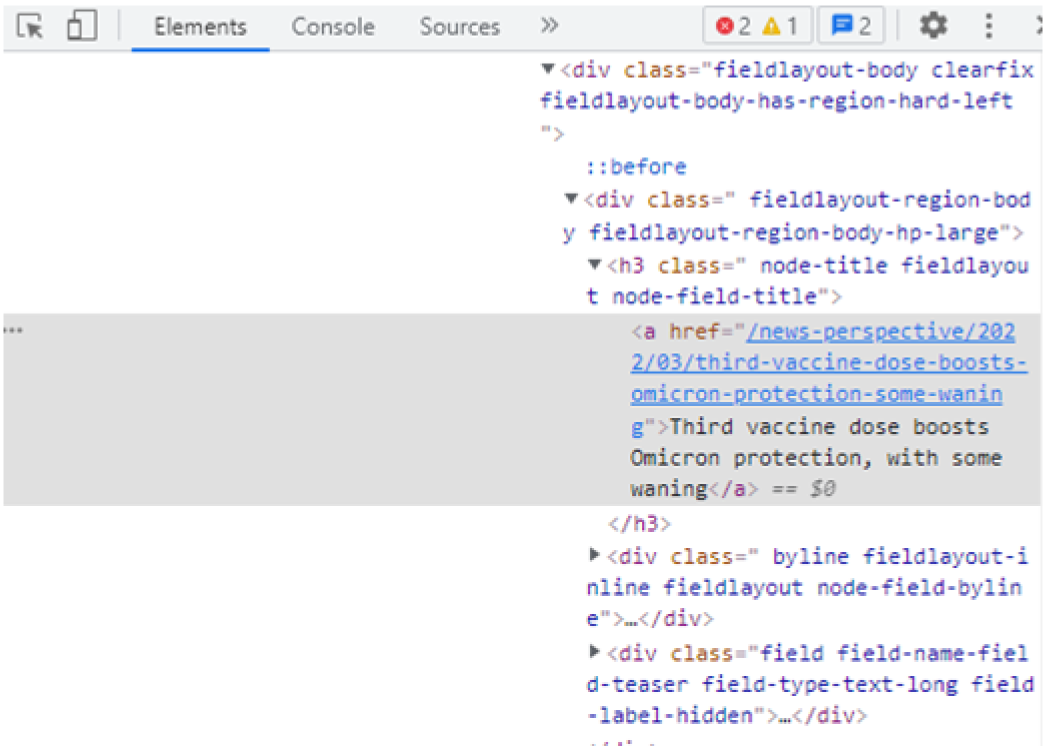
Third vaccine dose boosts Omicron protection, with some waning
Mary Van Beusekom | News Writer | CIDRAP News | Mar 03, 2022
A third dose with any of 3 COVID-19 vaccines boosted effectiveness substantially.

Rural, urban disparities seen in US COVID-19 vaccine uptake
Stephanie Soucheray | News Reporter | CIDRAP News | Mar 03, 2022
The gap between the groups did not close during the Delta and Omicron surges, a new study finds.

Asia's Omicron surges soar as US adds technology to WHO-led push
Lisa Schnirring | News Editor | CIDRAP News | Mar 03, 2022
US officials announce the transfer of technology to a global effort to speed and share pandemic tools.

Image 2. The layout of the CIDRAP news and perspective page

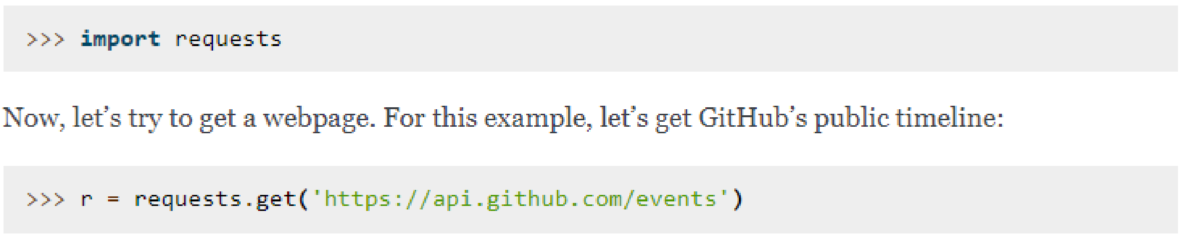
With BeautifulSoup we first deal with the issue of getting all the necessary URLs on the site. To achieve this, we can write a simple algorithm to recursively save any “href” tags into an array and repeatedly open those links and save them into the array as well. This will allow us to end up with all the possible links on the CIDRAP site, i.e., all the pages we will want to scrape. Obviously, we will filter out certain links such as any that lead us outside the CIDRAP domain and remove any duplicates.



```
▼<div class="fieldlayout-body clearfix fieldlayout-body-has-region-hard-left">
  ::before
  ▼<div class=" fieldlayout-region-body fieldlayout-region-body-hp-large">
    ▼<h3 class=" node-title fieldlayout node-field-title">
        <a href="/news-perspective/2022/03/third-vaccine-dose-boosts-omicron-protection-some-waning">Third vaccine dose boosts Omicron protection, with some waning</a> == $0
      </h3>
    ▶<div class=" byline fieldlayout-inline fieldlayout node-field-byline">…</div>
    ▶<div class="field field-name-field-teaser field-type-text-long field-label-hidden">…</div>
```

Image 3. A href link on the CIDRAP home page

We can next import the Requests python library which will allow us to get the HTML for the corresponding page we are looking at..



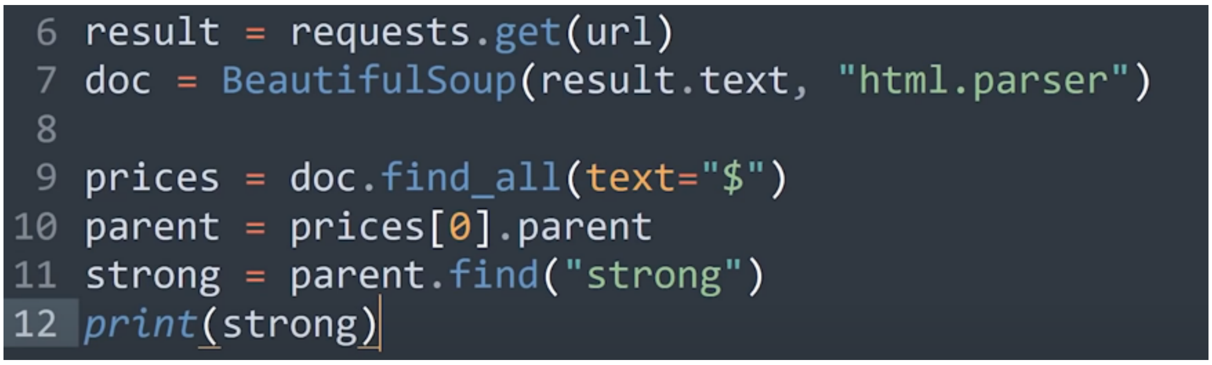
```
>>> import requests
```

Now, let's try to get a webpage. For this example, let's get GitHub's public timeline:

```
>>> r = requests.get('https://api.github.com/events')
```

Image 4. Get method from the Request library

From here, BeautifulSoup offers functions to allow us to filter through the page for any specific tags. The find_all method allows us to not just filter through certain tags, but also certain characters in the text, which will be helpful in scraping for the correct diseases and locations. We are then also able to simplify any nested elements allowing us to drill down to plain text.



```
result = requests.get(url)
doc = BeautifulSoup(result.text, "html.parser")

prices = doc.find_all(text="$")
parent = prices[0].parent
strong = parent.find("strong")
print(strong)
```

Image 5. An example where BeautifulSoup attempts to scrape through a page finding any text with "$" and finding the "strong" tag nested within

Once the entire domain has been scraped, we can compile our results together and parse it through into our database.

It was decided to use Python to build our minimum viable product (MVP) as Python contains many libraries that are dedicated for web scraping and our team is familiar with this language. Python will also be compatible with our framework. There were three main Python libraries that we have considered to use for web scraping: BeautifulSoup, Selenium and Scrapy.

	Pros	**Cons**
Scrapy	A complete framework Very fast and efficient	Requires some setup and is not easy to use

	Requires no dependencies ● Works on all operating systems ● Asynchronous capabilities More extensibility compared to BeautifulSoup Plenty of documentation available Works best with complex projects	Not user friendly for beginners
BeautifulSoup	User friendly, simple to set up ● Easy to pick up and learn Can extract HTML and XML elements Works best with smaller projects	Depends more on external Python libraries ● import bs4 ● Makes it more difficult to transfer code from other projects or machines Slow web scraper ● Requires the multiprocessing library to run faster Inefficient for larger sites
Selenium	Developed to facilitate web testing ● Web scraping was a by-product of it ● Can be used for both web scraping and testing, making it versatile Can scrape JavaScript ● Most sites now use JavaScript to load dynamic content	Slow at scraping large amounts of data Not intentionally designed to be a web scraper ● Has a steeper learning curve ● Not as user friendly

We ultimately decided to use BeautifulSoup as our main scraper due to it being easier to learn, setup and implement compared to the other two libraries. BeautifulSoup is also more suited to a small project like ours and shouldn't have issues handling the amount of data parsed in despite its limitations in regards to efficiency.

Web scraping on just a single computer takes a long time because of limitations on our internet network and can significantly slow down other programs that are also running at the same time. Using AWS EC2 resources for scraping will help to minimise the time spent on scraping by dividing the task and assigning it to hundreds of servers and will save the extracted data on the cloud rather than our local computer.

1.3.4 Database Design

Our software architecture includes a database to allow for interaction between our API and web scraper, whereby the web scraper will continuously update the database with new articles daily, and the API can query the database to find articles based on key terms which appear in the articles as sorted by the web scraper.

For the database, our team had the choice between NoSQL and SQL.

Upon consideration of the pros and cons for both architectures, we decided to go with NoSQL, specifically MongoDB Atlas due to the following factors:

- SQL has stronger data integrity and places a large emphasis on consistency of data, which is critical for systems such as e-commerce transactional systems or sales. Comparatively, NoSQL is able to scale horizontally much more efficiently, using minimal resources to store the data, which our team deemed more important due to the high volume of data which we will be processing through the web scraper.
- SQL has tight data constraints and design considerations which must be factored in but allows for more complex queries, whilst NoSQL has greater freedom but may struggle with more complex queries. The nature of the task will not require many complex queries, as we are simply querying for data with keywords, however aspects of the design and specification are subject to change, and thus NoSQL is preferable in this regard
- More flexibility to adapt to different types of data as extracted from various different sources or articles
- Team members have already used MongoDB in the past, thus reducing the learning required to implement this component of our design

1.3.5 Platform Design

Our platform will mainly be native on AWS EC2 Container Services, because of a few key things which allow for a clean integration of CI/CD for automated testing, deployment and scale.

The IaaS and PaaS paradigms are present in this service, which enables us to both develop our own platform but configure it in a much more intuitive way. This allows for faster development, focusing on the core product rather than infrastructure set up. We just simply set up the services and develop them. However cost can also be considered, and we can restrict certain resources which may not anticipate much use and traffic, but have the flexibility to expand our services further if needed. Docker is also a popular runtime which enables optimised instances running for its desired task with minimum overhead.

Also with easy GitHub integrations, we can design our own containers to automate the testing of our product both functionally and non-functionally, just using Python.

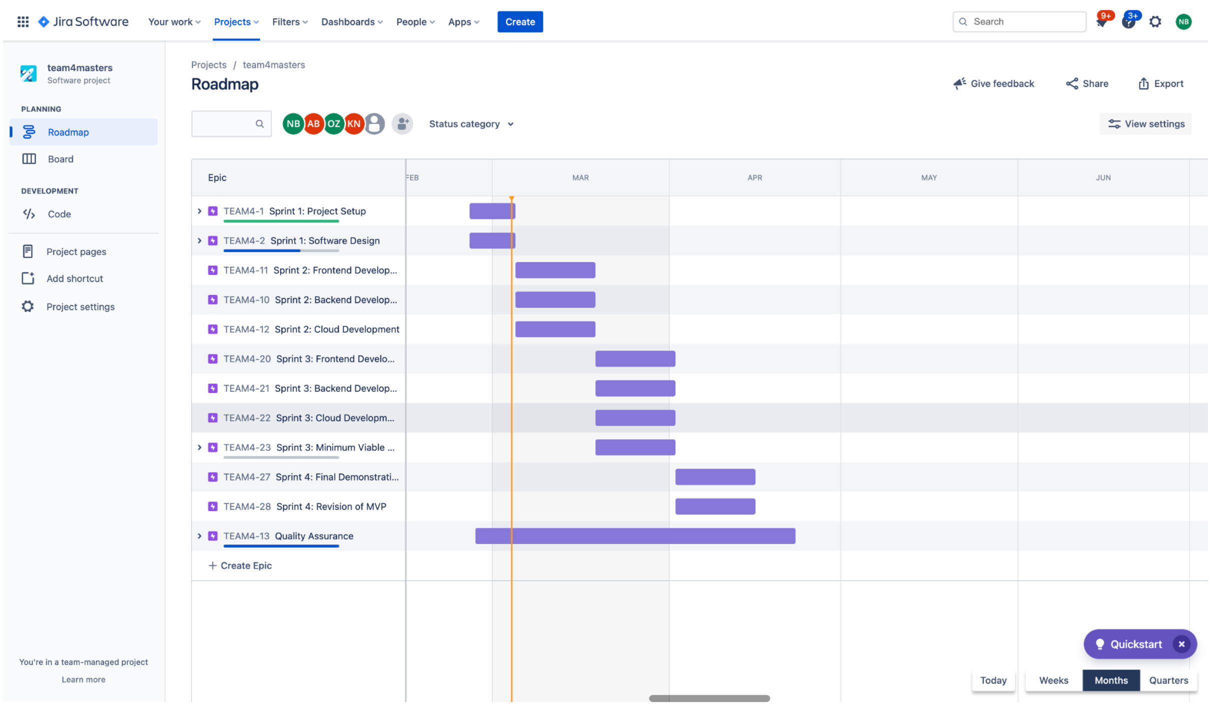
1.4 Project Management

Our team has aligned itself with utilising Agile methodologies to manage the project, whilst adapting to new requirements.

1.4.1 Project Roadmap, Kanban Board and Confluence

Our project timeline is split into four key sprints. Sprint 1 focuses on planning out the project and documentation required to design the app, Sprint 2 then fully focuses on development, and sprint 3 then concludes development for our final MVP. All our sprints have a duration of fourteen days. For each sprint, they are broken down into certain aspects and issues, such as frontend, backend and cloud development with the only exception of quality assurance, as that will always be on-going throughout the project (with both manual and automated testing).

Image 6. Jira Roadmap


Projects / team4masters
Roadmap

Epic
TEAM4-1 Sprint 1: Project Setup
TEAM4-2 Sprint 1: Software Design
TEAM4-11 Sprint 2: Frontend Develop...
TEAM4-10 Sprint 2: Backend Develop...
TEAM4-12 Sprint 2: Cloud Development
TEAM4-20 Sprint 3: Frontend Develo...
TEAM4-21 Sprint 3: Backend Develop...
TEAM4-22 Sprint 3: Cloud Developm...
TEAM4-23 Sprint 3: Minimum Viable ...
TEAM4-27 Sprint 4: Final Demonstrati...
TEAM4-28 Sprint 4: Revision of MVP
TEAM4-13 Quality Assurance

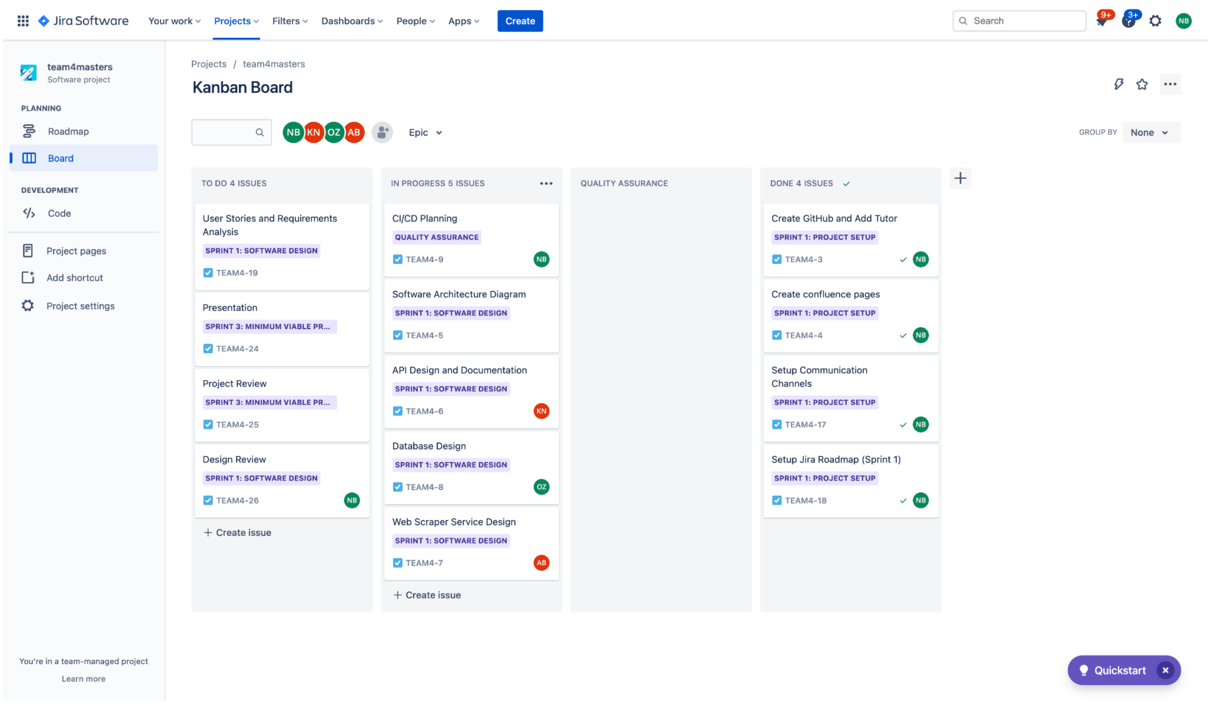
Although all of our sprints do not coincide with the project deliverable deadlines as mentioned in the following table, we have mainly aimed at producing a fully fledged MVP by the final deliverable. We have however scheduled our sprints so that our core MVP shall be completed before Deliverable 3, so we have the resources to add on additional functionality or refine current functionality.

Table 7. Sprint and Deliverable Timeline

Sprint	Deliverables	Deliverable Deadline
Sprint 1	Deliverable 1	4/03/2022 at 5 pm
Sprint 2	Deliverable 2	18/03/2022 at 5 pm
Sprint 3	No Deliverables, although expect core MVP to be completed.	Not Applicable
Sprint 4	Deliverable 3	14/04/2022 to 08/04/2022 (Exact timing TBA)
No Sprint in Session	Deliverable 4	25/04/2022 at 5 pm

To manage on-going tasks, the Scrum master adds tickets to the Jira Issues, to keep track of the overall progress of each Sprint, and adds core tickers of the next sprint one week in advance.

Image 7. Kanban Board

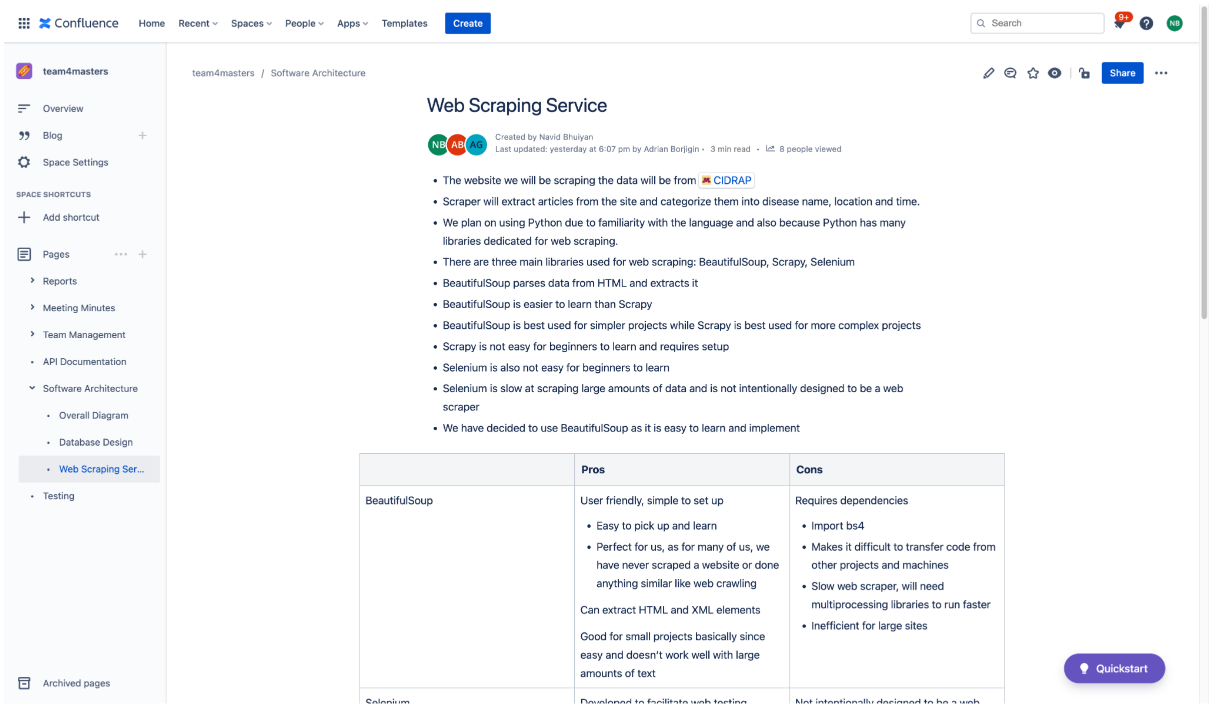


Kanban Board

TO DO 4 ISSUES	IN PROGRESS 5 ISSUES	QUALITY ASSURANCE	DONE 4 ISSUES
User Stories and Requirements Analysis — SPRINT 1: SOFTWARE DESIGN — TEAM4-19	CI/CD Planning — QUALITY ASSURANCE — TEAM4-9 — NB		Create GitHub and Add Tutor — SPRINT 1: PROJECT SETUP — TEAM4-3 — NB
Presentation — SPRINT 3: MINIMUM VIABLE PR... — TEAM4-24	Software Architecture Diagram — SPRINT 1: SOFTWARE DESIGN — TEAM4-5		Create confluence pages — SPRINT 1: PROJECT SETUP — TEAM4-4 — NB
Project Review — SPRINT 3: MINIMUM VIABLE PR... — TEAM4-25	API Design and Documentation — SPRINT 1: SOFTWARE DESIGN — TEAM4-6 — KN		Setup Communication Channels — SPRINT 1: PROJECT SETUP — TEAM4-17 — NB
Design Review — SPRINT 1: SOFTWARE DESIGN — TEAM4-26 — NB	Database Design — SPRINT 1: SOFTWARE DESIGN — TEAM4-8 — OZ		Setup Jira Roadmap (Sprint 1) — SPRINT 1: PROJECT SETUP — TEAM4-18 — NB
+ Create issue	Web Scraper Service Design — SPRINT 1: SOFTWARE DESIGN — TEAM4-7 — AB		
	+ Create issue		

All of our documentation is also sorted on Confluence, as it provides the hub for all developers to go through and understand our app.

Image 8. Confluence Page Example



Web Scraping Service

Created by Navid Bhuiyan
Last updated: yesterday at 6:07 pm by Adrian Borjigin • 3 min read • 8 people viewed

- The website we will be scraping the data will be from CIDRAP
- Scraper will extract articles from the site and categorize them into disease name, location and time.
- We plan on using Python due to familiarity with the language and also because Python has many libraries dedicated for web scraping.
- There are three main libraries used for web scraping: BeautifulSoup, Scrapy, Selenium
- BeautifulSoup parses data from HTML and extracts it
- BeautifulSoup is easier to learn than Scrapy
- BeautifulSoup is best used for simpler projects while Scrapy is best used for more complex projects
- Scrapy is not easy for beginners to learn and requires setup
- Selenium is also not easy for beginners to learn
- Selenium is slow at scraping large amounts of data and is not intentionally designed to be a web scraper
- We have decided to use BeautifulSoup as it is easy to learn and implement

	Pros	Cons
BeautifulSoup	User friendly, simple to set up • Easy to pick up and learn • Perfect for us, as for many of us, we have never scraped a website or done anything similar like web crawling Can extract HTML and XML elements Good for small projects basically since easy and doesn't work well with large amounts of text	Requires dependencies • Import bs4 • Makes it difficult to transfer code from other projects and machines • Slow web scraper, will need multiprocessing libraries to run faster • Inefficient for large sites
Selenium	Developed to facilitate web testing	Not intentionally designed to be a web

1.4.2 Role Delegations

We mainly have decided on these roles to allow for effective management and development. Roles were formed based on interest and skill sets of certain members, and the team shall be sticking to these roles for the rest of the project.

Name	Role	Delegation
Navid Bhuiyan	Project and Tech Lead	Project management, task delegation and responsible for the overall design of the app. Gets involved in all aspects of development.
Navid Bhuiyan	Scrum Lead	Manages and designs agile practices within the team.
Adrian Borjigin	Full-stack Developer	Responsible for designing the scraping and data services for the frontend.
Aaron Guan	Backend Developer	Responsible for designing and developing the backend architecture for our scraping services and APIs.
Jin-Ao Olson Zhang	Backend Developer	Responsible for designing and developing the backend architecture for our APIs and cloud services.

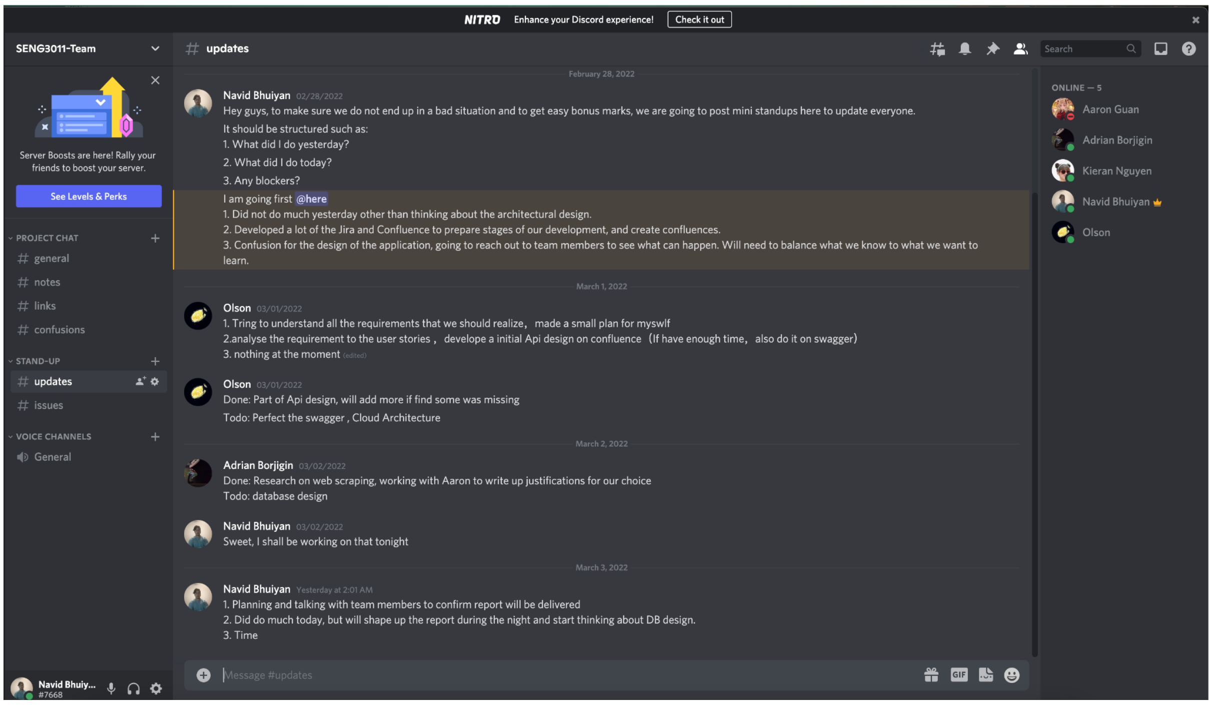
Kieran Nguyen	Backend Developer	Responsible for designing and developing the backend architecture for our APIs and database.

1.4.3 Agile Practices

Our key agile practices we maintain are meetings twice a week, daily standups, outlining progress per sprint's requirements and each project member managing their allocated ticket on the Kanban board.

Our meetings are often coordinated on Monday and Friday at 10 am, to cover our overall progress in the week and to enable sessions where the team can ask questions. These meetings are often flexible and offer the time to do pair programming and to discuss initiatives and bugs. Our daily standups are all managed on the Discord platform, as each developer has allocated themselves available on the platform to track their own progress daily. If no one attends the stand-up then it is up to the discretion of the Scrum Master to catch up with team members on their progress and to prompt them to record on Discord.

Image 8. Stand-up Channel



updates

February 28, 2022

Navid Bhuiyan 02/28/2022
Hey guys, to make sure we do not end up in a bad situation and to get easy bonus marks, we are going to post mini standups here to update everyone.
It should be structured such as:
1. What did I do yesterday?
2. What did I do today?
3. Any blockers?

I am going first @here
1. Did not do much yesterday other than thinking about the architectural design.
2. Developed a lot of the Jira and Confluence to prepare stages of our development, and create confluences.
3. Confusion for the design of the application, going to reach out to team members to see what can happen. Will need to balance what we know to what we want to learn.

March 1, 2022

Olson 03/01/2022
1. Tring to understand all the requirements that we should realize， made a small plan for myswlf
2.analyse the requirement to the user stories， develope a initial Api design on confluence（If have enough time， also do it on swagger）
3. nothing at the moment (edited)

Olson 03/01/2022
Done: Part of Api design, will add more if find some was missing
Todo: Perfect the swagger , Cloud Architecture

March 2, 2022

Adrian Borjigin 03/02/2022
Done: Research on web scraping, working with Aaron to write up justifications for our choice
Todo: database design

Navid Bhuiyan 03/02/2022
Sweet, I shall be working on that tonight

March 3, 2022

Navid Bhuiyan Yesterday at 2:01 AM
1. Planning and talking with team members to confirm report will be delivered
2. Did do much today, but will shape up the report during the night and start thinking about DB design.
3. Time

1.5 Conclusion

To conclude, the project overall has considered the infrastructure and software level design where we can now commence development for Deliverable 2. However missing from this assessment are key design aspects of the database such as the schema, which need to be properly fleshed before any database development commences. Also cost considerations have not been put into account to add why we chose this architecture over others, especially when we argue our AWS EC2 Container Service native app balances both customizability, easy configuration and cost.

However, with what do have in terms of design and our management practices, we can build it along the way and hopefully provide fully supported and enriched reporting and a strongly built API for Deliverable 2.