

Deliverable 1: Design Details

Development of API module and ability to run in Web service mode

For this project, we have decided to use a RESTful API due to the simplicity and flexibility it provides. It can handle multiple types of calls and return data in various formats that will best suit the needs for this project. Also, it doesn't require a lot of overhead or require any specific procedures that need to be learnt to use it, thus making it the best option for our team.

Our team has also decided to use the Flask framework as it is very simple and intuitive to use and provides a wide range of tools and libraries that will help build the web application. Its micro-framework structure is best suited for this project for many reasons including that it is simple, lightweight, flexible, scalable and works well with RESTful apps. All team members have previous experience using Flask as well. In comparison to other frameworks, this was the most appropriate choice for our project's needs.

Finally, we have decided to use AWS LightSail. Though the majority of the web services researched provide relatively similar features such as providing databases to utilise, storage, quick deployment, scalability, simple interface and good documentation, AWS has a wide range of resources and a large IT community to help online, which will be very beneficial for our team. It is also known to be very suitable for small projects and the costs are relatively cheap

Comparison of Options Available:

<u>Option</u>	<u>Pros</u>	<u>Cons</u>
Architecture		
REST or RESTful (Representational State Transfer)	<ul style="list-style-type: none"> • Uses Bandwidth Efficiently • Simple to understand and learn • Scalable • Lightweight • Has distinct and clear end points • Multi-language client 	<ul style="list-style-type: none"> • Can't maintain states within sessions. • There's no contract between the client and server <ul style="list-style-type: none"> ◦ not recommended when transferring

	<p>compatibility</p> <ul style="list-style-type: none"> • Works well with existing operating environments (like firewalls) • Has higher level libraries for frameworks and widgets. • REST just has the HTTP processing overhead. • Can return XML, JSON, YAML etc. • flexible due to loose guidelines <ul style="list-style-type: none"> ◦ made to suit developers needs 	<p>confidential data</p> <ul style="list-style-type: none"> • Due to its flexibility, it requires detailed documentation to understand each API call
SOAP (Simple Object Access Protocol)	<ul style="list-style-type: none"> • Can work with any application layer protocol, such as HTTP, SMTP, TCP, or UDP. • Security, authorization, and error-handling are built into the protocol • Doesn't assume direct point-to-point communication <ul style="list-style-type: none"> ◦ Good for complex transactions that require high security. 	<ul style="list-style-type: none"> • More Bandwidth needed, which can slow page download • XML used in this is expensive to parse • Most SOAP stacks use SAX parsing which creates a big overhead <ul style="list-style-type: none"> ◦ normal HTTP processing overhead plus XML parsing overhead • Not flexible due to standardised protocols that need to be followed
RPC (Remote Procedural Call)	<ul style="list-style-type: none"> • Server independent. • Process-oriented and thread-oriented models work well • Simple due to straightforward semantics • Development of procedures for remote 	<ul style="list-style-type: none"> • Users are required to know procedure names or specific parameters in a specific order. • Context switching increases scheduling costs • RPC has no standard

	<ul style="list-style-type: none"> calls is general. Code rewriting / re-developing is reduced Allows for applications to be used in a distributed environment 	<ul style="list-style-type: none"> Interaction based, so there is no flexibility relating to hardware Can only support TCP/IP protocol. Doesn't yet work for wide-area networks.
Framework for RESTful API		
Flask	<ul style="list-style-type: none"> Scalable Flexible and Simple Easy to Navigate Lightweight <ul style="list-style-type: none"> Doesn't need many extensions to work Supports modular programming <ul style="list-style-type: none"> Functionalities can be split into modules. Allows structure to be more flexible and testable Easy to follow documentation 	<ul style="list-style-type: none"> Doesn't have a lot of tools <ul style="list-style-type: none"> Will need to add extensions which can slow the app Hard to get used to big projects in modular form, esp. if you start half way Flask is compatible with many technologies. Thus this becomes additional maintenance costs to handle those dependencies
Django	<ul style="list-style-type: none"> Features needed to build web applications is built in Has a standardised structure. Everyone using it will be on the same page Secure 	<ul style="list-style-type: none"> Django ORM is not that great Django evolves slowly It has a large ecosystem with lots of configurations Not suitable for smaller projects Steep learning curve Can't handle multiple requests at once
Express	<ul style="list-style-type: none"> Can use JavaScript both on the back end and front end. This makes development easier 	<ul style="list-style-type: none"> Software needs to be built in JavaScript and Node.js May find it difficult to understand the

	<ul style="list-style-type: none"> • It can handle lots of requests and notifications • Large Open-source community • Easy integration of third-party services and middleware • Easy to learn 	<ul style="list-style-type: none"> • callback nature. • Need to deal the Callback hell issue • Node.js is not suited for heavy computation
FastAPI	<ul style="list-style-type: none"> • Python web frameworks • Free • Fast • Data validation much simpler and faster due the Pydantic (Data validation and settings management using python type annotations) library • The autocomplete feature allows production in less time and effort during debugging 	<ul style="list-style-type: none"> • IT community is small <ul style="list-style-type: none"> ◦ lower external educational information like books, courses, or tutorials. • Not tested as much in comparison to others like Flask, so a little less reliable • Need to tie everything together, which can cause the main file to become very long or crowded
Web Services		
Google Firebase	<ul style="list-style-type: none"> • Database capabilities: <ul style="list-style-type: none"> ◦ The NoSQL database (called Firestore) is better for storing large amounts of data ◦ Helps to achieve a high performance level due to its flexibility and scalability • Faster Development: <ul style="list-style-type: none"> ◦ Firebase offers a set of pre-requisites of backend ◦ reduces the 	<ul style="list-style-type: none"> • Limited querying capabilities: <ul style="list-style-type: none"> • not made for processing complex queries as it relies on a flat data hierarchy • Limited Data migration • Platform-dependence • Android focused • Less support for iOS

	<p>overhead costs and reduces dependency-based challenges.</p> <ul style="list-style-type: none"> • It has a large following in the IT community • Concise documentation • Quick and easy integration and setup 	
Digital Ocean	<ul style="list-style-type: none"> • Easy • Scalable • Includes virtual machine app engine, storage and database • Quick deployment. • Reliable • Droplet backups are easy to enable and affordable. • Has good Documentation and tutorials 	<ul style="list-style-type: none"> • It could be a little hard for beginners. • Setup of the server might take time. • No live support available • Doesn't offer some important pre-installed systems like Bitnami apps, ODOO, multisite WordPress, etc. • Expensive
Aws LightSail	<ul style="list-style-type: none"> • Fixed Pricing Model, relatively cheap too • Easy to use User Interface • Easy to set up Networking • Provides managed databases that you can attach to your instances. • Virtual servers. • Simplified load balancing. • Large IT community and wide range of resources 	<ul style="list-style-type: none"> • Not suitable for enterprise workloads • limited when compared to AWS EC2 • No auto-scaling available • No metrics or dashboard to know bandwidth usage or storage use • Poor tech support from AWS
Python Anywhere	<ul style="list-style-type: none"> • Offers MySQL and SQLite • PythonAnywhere is more like a VPS; you 	<ul style="list-style-type: none"> • Doesn't support NoSQL databases like Mongo and Couch • Must push to GitHub

	SSH in (SFTP for paid users), you have static file storage, file upload, etc. <ul style="list-style-type: none"> · Has many deployment options including git/svn/hg. · Simple interface · Relatively cheap 	or Bitbucket, then from a bash shell you pull it to PythonAnywhere, which is a bit tedious <ul style="list-style-type: none"> · only supports Python officially on the server side
--	---	---

Table 1: Comparison of different web services

How parameters can be passed to the module and collection of results

For the construction of the application programming interface (API) we are using the RESTful API architecture and design principles. The fundamental nature of this architecture involves pre-defining a set of HTTP methods that perform actions on particular resources.

Firstly, we are implementing a Pythonic scraper that will access the following website: <https://www.cdc.gov/outbreaks/> and navigate to specific outbreak articles. Within these articles, the scraper will parse the HTML code and collate relevant disease and publication information to store in a sqlite database. Everytime a user requests for information, the scraper runs again to ensure that the data extracted from the website is timely, relevant and concurrent with the data already existing in the database. If there are updates detected on the CDC website then the relevant fields in the database are updated before the API accesses them.

The input parameters for the API are as follows:

Input	Format	Explanation
Period of time	“yyyy-MM-ddTHH:mm:ss”	Composed of two sub inputs: <ul style="list-style-type: none"> - Start date - End date This input is mandatory.
Key terms	E.g String,String	List of key terms separated by commas. API is not case

		sensitive to these terms. This input parameter is optional.
Location	String	The location string can be a country, city, state that is contained within the disease reports. This input parameter may be optional.

Table 2: Depicting user inputs

The API first checks that the input parameters are valid .i.e dates are not in the future, correct format etc. The API extracts the relevant data from the sqlite database and converts the data into JSON objects.

Sample Interaction

1. User provides parameters to the web-applications

Search the database for reports on disease outbreaks

2021-10-13T00:00:00

2021-11-13T00:00:00

Indiana

E. coli

Search

2. Extracted Input

```
{
  start_date : "2021-10-13T00:00:00"
  end_date : "2021-11-13T00:00:00"
  key_terms : "E. coli"
  location : "Indiana"
}
```

3. API identifies a relevant article (valuable data is outlined)

E. coli Outbreak Linked to Baby Spinach



Posted January 6, 2022

This outbreak is over. Stay up to date on food recalls and outbreaks to avoid getting sick from eating contaminated food.

Fast Facts

- Illnesses: 15
- Hospitalizations: 4
- Deaths: 0
- States: 10
- Recall: No
- Investigation status: Closed



January 6, 2022

CDC, public health and regulatory officials in several states, and the U.S. Food and Drug Administration (FDA) investigated a multistate outbreak of *E. coli* O157:H7 infections.

Epidemiologic and laboratory data showed that Josie's Organics prepackaged baby spinach with a "best by" date of October 23, 2021, made people sick.

As of January 6, 2022, this outbreak is over.

Epidemiologic Data

A total of 15 people infected with the outbreak strain of *E. coli* O157:H7 were reported from 10 states (see [map](#)). The true number of sick people in this outbreak was likely much higher than the number reported, and this outbreak may not be limited to the states with known illnesses. This is because many people recover without medical care and are not tested for *E. coli*.

Illnesses started on dates ranging from October 13, 2021, to November 8, 2021, (see [timeline](#)). Sick people ranged in age from 1 to 76 years, with a median age of 26, and 80% were female. Of 15 people with information available, 4 were hospitalized and 3 developed a type of kidney failure called hemolytic uremic syndrome (HUS). No deaths were reported.

Data Table	
State of Residence	Number of Sick People
California	1
Indiana	4
Iowa	1
Michigan	1
Minnesota	2
Missouri	1
Nebraska	1
Ohio	1
Pennsylvania	1
South Dakota	2

4. Scraper Output (generated by scraping outlined texts)

```
{
  url: "https://www.cdc.gov/ecoli/2021/o157h7-11-21/index.html",
  date_of_publication: "2022-01-06T00:00:00",
  headline: "E. coli Outbreak Linked to Baby Spinach",
  main_text: "CDC, public health and regulatory officials ... On November 15, 2021, CDC advised people not to eat, sell or serve Josie's Organics prepackaged baby spinach with \"best by\" date of October 23, 2021.",
  reports: [
    diseases: ["E. coli"],
    syndromes: ["hemolytic uremic syndrome"],
    event_date: "2021-10-13T00:00:00 to 2021-11-08T00:00:00",
    locations: ["Indiana", "United States"],
    cases : 4,
    Hospitalizations: ,
    Total_deaths: ,
    total_cases: 15,
  ]
}
```



```

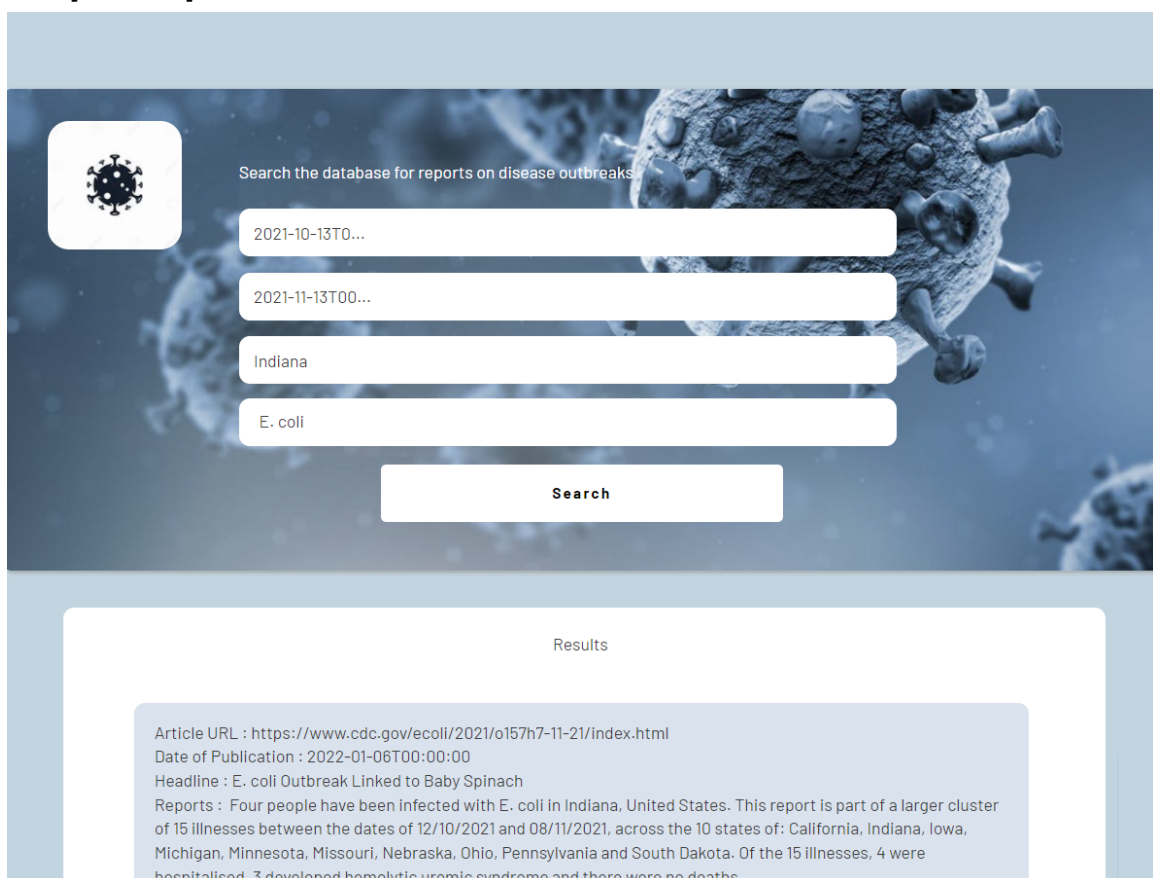
total_hospitalizations: 4,
total_deaths: 0,
Total_locations: ["California, Indiana, Iowa, Michigan,
                  Minnesota, Missouri, Nebraska, Ohio,
                  Pennsylvania, South Dakota"]
    ]
}

```

5. Sample Report (generated from highlighted data)

Four people have been infected with *E. coli* in Indiana, United States. This report is part of a larger cluster of 15 illnesses between the dates of 12/10/2021 and 08/11/2021, across the 10 states of: California, Indiana, Iowa, Michigan, Minnesota, Missouri, Nebraska, Ohio, Pennsylvania and South Dakota. Of the 15 illnesses, 4 were hospitalised, 3 developed hemolytic uremic syndrome and there were no deaths.

6. Sample Output



Search the database for reports on disease outbreaks

2021-10-13T0...

2021-11-13T00...

Indiana

E. coli

Search

Results

Article URL : <https://www.cdc.gov/ecoli/2021/o157h7-11-21/index.html>
 Date of Publication : 2022-01-06T00:00:00
 Headline : E. coli Outbreak Linked to Baby Spinach
 Reports : Four people have been infected with *E. coli* in Indiana, United States. This report is part of a larger cluster of 15 illnesses between the dates of 12/10/2021 and 08/11/2021, across the 10 states of: California, Indiana, Iowa, Michigan, Minnesota, Missouri, Nebraska, Ohio, Pennsylvania and South Dakota. Of the 15 illnesses, 4 were hospitalised, 3 developed hemolytic uremic syndrome and there were no deaths.

API Status Codes

Once the user input is processed, the application will check the returned status codes resulting from the API call. This code can provide users with feedback based on the results of the request. The sample status codes we may return are as below:

Code	Meaning	Description
200	OK	The request action was successful
400	Bad request	The request was malformed
401	Unauthorised	The requested action was not able to be performed
404	Not Found	The resource was not found
500	Internal server error	There was a server error when processing the request.

Table 3: Displaying API Status codes

API Endpoints

HTTP Method	API endpoint	Parameters	Description
GET	/report?start_date=...&end_date=...	Start date End date	Returns all reports within the date range
GET	/report?{parameters}	Start date End date Location Key Terms	Returns specific reports based on parameters

Table 4: Displaying API Endpoints

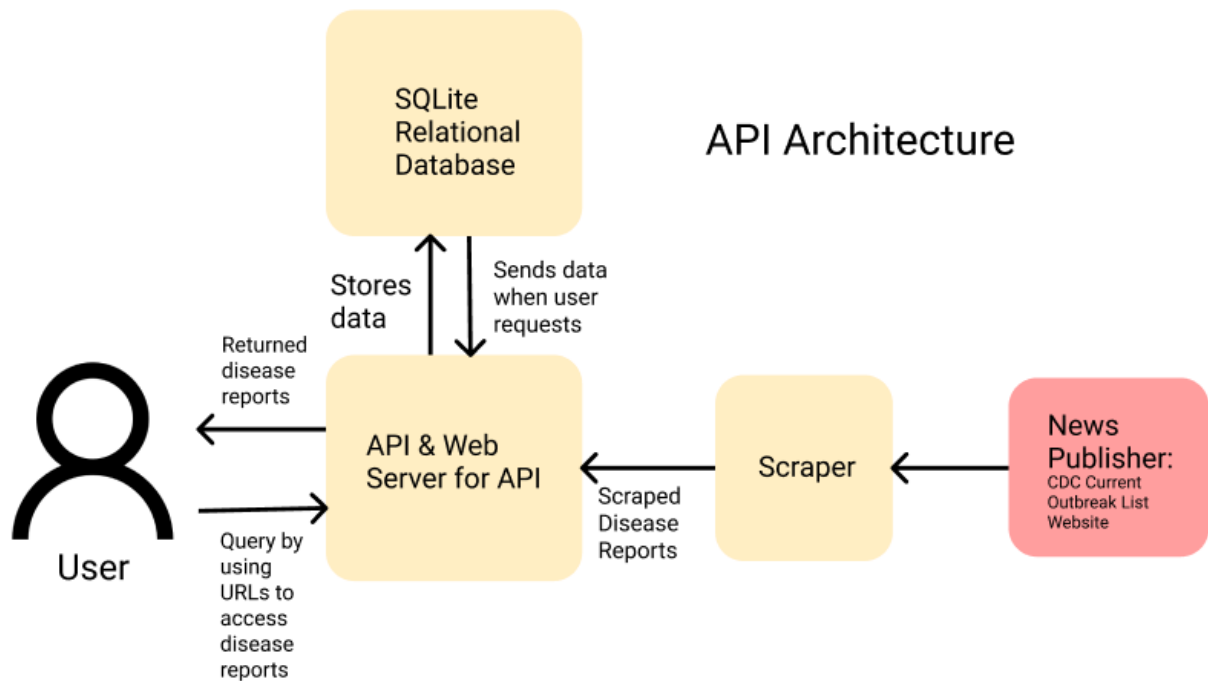


Diagram 1: Summarising API interactions

Documentation

It is imperative to apply good principles when designing and constructing APIs, including providing detailed and effective API documentation. For the documentation of our RESTful API we have decided to use Swagger. Using Swagger will be advantageous, as there is a Flask extension called Flask-RESTPlus which allows us to automate the documentation and specification process. Furthermore, Swagger can be easily maintained and adjusted, proving beneficial for updating, testing and debugging our API.

Justification

Before choosing particular technologies for our stack, we conducted thorough research and analysed the advantages and disadvantages as shown below:

Language/ Platform	Advantages	Disadvantages
-----------------------	------------	---------------

Python for Backend	<ul style="list-style-type: none"> • Rich native library support for web scraping: Scrapy, BeautifulSoup. • Python libraries automates the extraction process of web data once a day which is a significant advantage compared to those which does not. • Python web-scraping script can easily handle both scraping data and parsing into a structured format, while others can only do the first. • Concise and simple syntax • More gentler learning curve comparing to Node.js and Golang • Experiences from team members in Python 	<ul style="list-style-type: none"> • Has comparatively limited performance and requires more effort to scale • Dynamic typing nature might increase the number of errors • Limited performance in shortcoming comparing to other languages like C++
Libraries: Scrapy	<ul style="list-style-type: none"> • Efficiently extract data from websites, process them and store them in a format • “Spider” which can automate the web scraping process everyday whereas BeautifulSoup does not • End-to-End 	<ul style="list-style-type: none"> • BeautifulSoup is better at extracting specific pieces of information on a webpage.
Flask for Framework	<ul style="list-style-type: none"> • Very flexible when it comes to adding libraries or plugins, whereas other languages cannot afford too many changes. • Flask application requires much fewer 	<ul style="list-style-type: none"> • Admin features are not as prominent as in Django. • With Flask, developers have to work with different databases by using ORM systems for Python and SQLAlchemy

	<p>lines of code for a simple task, whereas Django needs more than 2 times more lines of code than Flask.</p> <ul style="list-style-type: none"> • Can build simple applications easily with less code while Django's functionalities might be outdated for simple applications. 	<p>as the SQL toolkit. SQL queries have to be written for common tasks.</p> <ul style="list-style-type: none"> • Doesn't have a ready-to-use feature to handle administration tasks, whereas Django does.
Sqlite for Databasing	<ul style="list-style-type: none"> • Easy to use, less requirements for setup • Can be easily embedded in an application since it is lightweight. • Reliable. Unlike PostgreSQL, it is simple to make a backup with SQLite. Allows writing smaller queries and hence reduces the likelihood of bugs. • Flexible. Supports all key programming languages including Python, Java, C++ and so on. 	<ul style="list-style-type: none"> • SQLite has syntax and format-related limitations when you compare it to PostgreSQL or MySQL. • Volume related restrictions. SQLite is only prepared for low to medium traffic use cases.
Javascript for Frontend	<ul style="list-style-type: none"> • Since JavaScript is able to run on a browser, it is extremely fast. • JavaScript can be inserted into any web page • The demand on servers is reduced as JavaScript is run on the client's side of the application. • Javascript is extremely powerful in terms of enriching user interactivity. Javascript has the capability to include "events" such as 	<ul style="list-style-type: none"> • Different browsers can interpret javascript differently. This makes it difficult to guarantee a uniform output. • The frontend code is visible to others. • If an error occurs in JavaScript, the webpage may stop rendering. • Debugging JavaScript is not as easy as other languages such as Python. • It only includes single inheritance so cannot

	<p>clicking and can respond to them by communicating with the application's backend.</p> <ul style="list-style-type: none"> • Webpages can also dynamically update themselves as a response to user activity without the page having to reload. • Additional functionality that cannot be achieved using HTML and CSS alone can be achieved with Javascript. 	<p>support object oriented characteristics.</p> <ul style="list-style-type: none"> • If JavaScript is disabled on the browser, the code for the whole application would not run.
HTML for Frontend	<ul style="list-style-type: none"> • HTML is simple and easy to use • HTML can be easily integrated with JavaScript • HTML is beneficial in implementing the foundational structure and content of the page. • HTML is also simple and a platform independent language that is supported by all browsers. 	<ul style="list-style-type: none"> • Using HTML alone can result in a lot of code even for very simple pages. • HTML is static in nature and implementing dynamic web pages are extremely difficult compared to PHP and JavaScript. • It is difficult to make aesthetic designs using HTML
CSS for Frontend	<ul style="list-style-type: none"> • CSS allows for greater control of the aesthetic of the webpage as it has more formatting options compared to HTML. • Using CSS means that there would be less code per web page allowing for faster download times. • CSS would also save time since the code can be written once, and reused for different 	<ul style="list-style-type: none"> • There are different levels of CSS e.g CSS 1, 2, 3 which could cause confusion on which one to choose. • CSS is not compatible with all browsers, so extra testing would be needed to ensure compatibility.

	<p>pages.</p> <ul style="list-style-type: none"> • It also allows for the ease of maintenance as a simple change in the style would allow for all elements in the page to be updated automatically. 	
Axure for frontend design	<ul style="list-style-type: none"> • Allows easy HTML and CSS code generation and exporting • The team is comfortable with the Axure interface and its offered features • No cost allowing easy access to members of the team that did not have it prior • Provides all UX features that the team has discussed in the brief initial UX design 	<ul style="list-style-type: none"> • Axure has a learning curve when starting out as certain features may be hard to find • On Axure it is difficult to make complex interactive UX design such as popup windows • Requires prior knowledge in coding (HTML,CSS) to export UX • Creates HTML that is hard to read without prior experience.
VScode for an IDE	<ul style="list-style-type: none"> • The team has easy access to VScode as it has no cost • The team is comfortable with VScode being the main IDE, as all team members have prior experience. • Quick compilation times for large projects • VScode allows easy integration with other systems • Easy to use with Git • With the Python extension VScode is simple to use and offers all features the team needs • VScode saves a lot of time with its intelligent 	<ul style="list-style-type: none"> • High memory and CPU consumption • Steep learning curve when starting out

	code completion feature	
--	-------------------------	--

Table 5: Justification of chosen technology stack