SENG3011 Final Report

Team megAPIxels

Lachlan Fraser	z5258840
Rubin Roy	z5168911
Humza Saeed	z5309373
Sam Thorley	z5257239
Austin Walsh	z5311341

25 April 2022

GitHub Repository: https://github.com/SENG3011-megAPIxels/SENG3011_megAPIxels

Deployed Analytics Application: https://seng3011-megapixels.pages.dev

API Swagger Documentation:
http://swagger-env-1.eba-zzwsivt4.ap-southeast-2.elasticbeanstalk.com/docs/#/

Table of Contents

1. Introduction

As the world adapts to life after the COVID-19 pandemic, the need for businesses and individuals to be aware of their societal context is paramount in preparation for future scenarios. Our data analytics platform, ‘Interactive Outbreak Predictor’ (IOP), provides the ability to view machine learning-based future predictions for a range of areas including COVID-19 cases and deaths, economic, jobs, real estate and stock market data. In conjunction to this, users can view historical data for those datasets since the start of the pandemic, for reference and comparison. This report delves into the development of IOP beginning with the research conducted to form our requirements as well as our design ideas, later moving onto technical architecture and project management. IOP has been developed with the community in mind, addressing the concerns of people from a range of demographics to facilitate analysis and informed decision making based on the various future scenarios presented to them, potentially improving future outcomes.

2. Use Cases and Requirements

In understanding our problem, we have conducted research and extrapolated user requirements for our project for which use cases have also been derived. The following sections contain our findings and requirements.

2.1. Research

The sections under 2.1 contain our research methods and findings of our target audience and the analytics features deemed useful for them.

2.1.1. Second-Hand Data and demographics

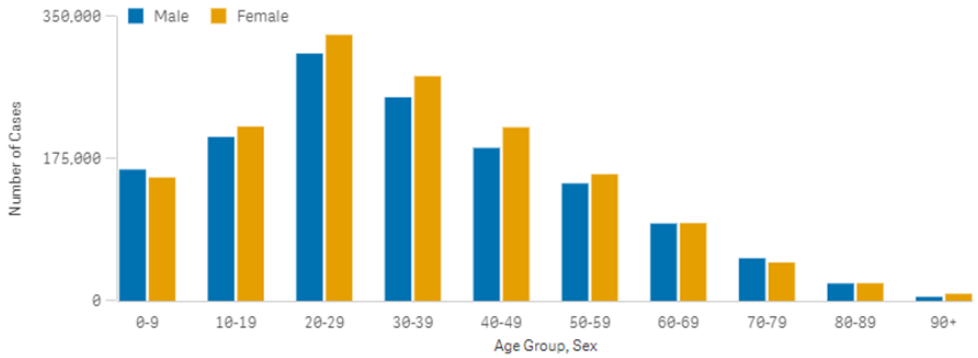
The following is a statistical enquiry designed to aid in the formulation of the problem statement. The three questions below were considered and asked to a number of members of the public to investigate the demographic of users affected by COVID-19 while simultaneously identifying our main target audience and which features should be prioritised:

- Age demographic
- Occupation
- Computer Usage and familiarity

COVID-19 cases by age group and sex

This graph shows the number of COVID-19 cases for males and females by age group since the first case was reported.

Source: NINDSS data 22/4/2022



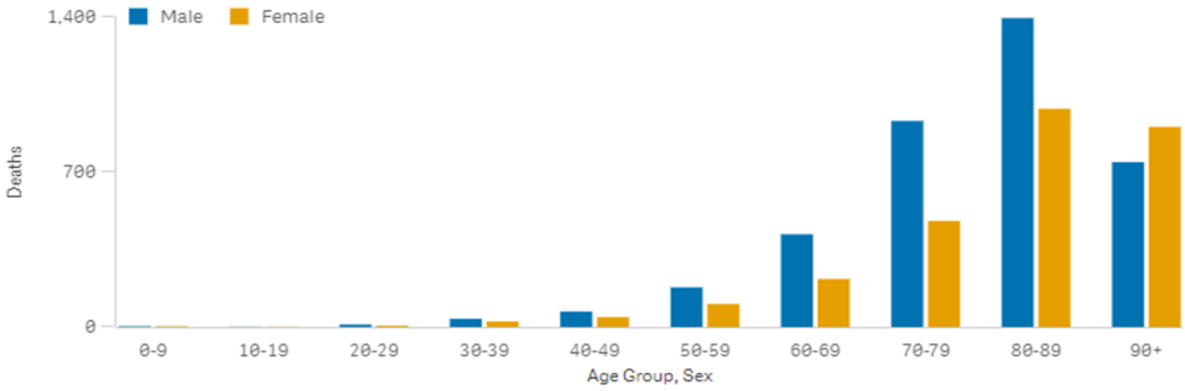
The total number of cases in this chart may be less than what is reported due to delays in notification to the National Interoperable Notifiable Disease Surveillance System (NINDSS) or where the case's age or sex are unknown.

Figure 1. COVID-19 cases by age group and sex | Source: Department of Health, Australian Gov

COVID-19 deaths by age group and sex

This graph shows the number of COVID-19 associated deaths in Australia for males and females by age group since the first case was reported.

Source: NINDSS data 22/4/2022



The total number of deaths in this chart may be less than what is reported due to delays in notification to the National Interoperable Notifiable Disease Surveillance System (NINDSS) or where the case's age or sex are unknown.

Figure 2. COVID-19 deaths by age group and sex | Source: Department of Health, Australian Gov

From the data above, it is clear that most people who catch the disease are people aged between 20-39, whereas most deaths occur from 80-90+ years.

From these observations, we were led to identifying the primary demographic trend to be of young to middle aged adults, as well as the elderly. (i.e, 20-50, 70-80).

2.1.2. First-Hand Data and Demographics

For our first hand data and surveying, our goal was to gather enough information from people in our target audience so that we could determine what users were most concerned about. We figured the best and most efficient way to do this was to create google form sheet and send it out to 25 people from a wide range of age groups, as shown in Figure 3 below:

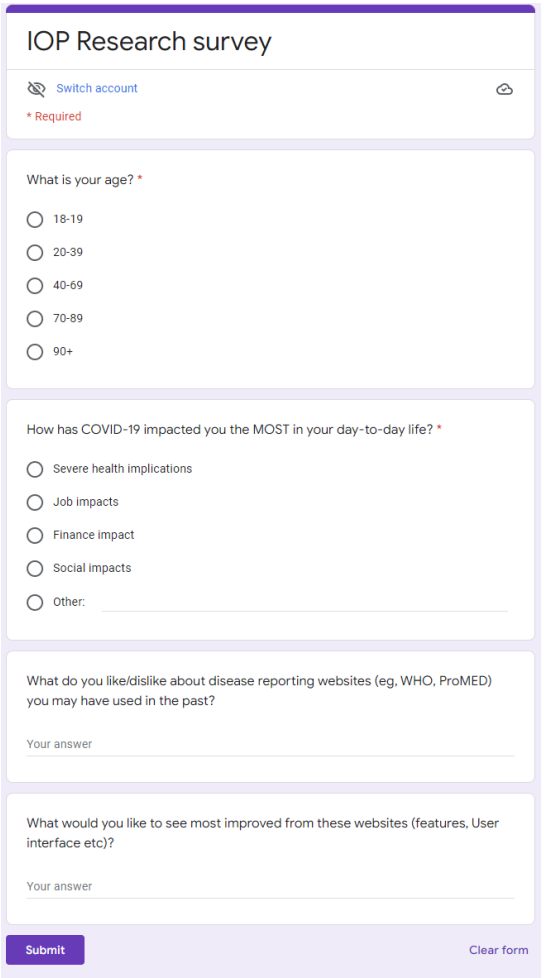


Figure 3. Google forms containing research questions

The reason why we chose google forms instead of other tools, such as jotform.com, was because it was the most simple and fastest way to create an effective survey using a google account which most people have, instead of creating a new account on other similar websites.

The following bar graph was generated from the multiple choice sections of the survey, namely, “What is your age?” and “How has COVID-19 impacted you the MOST in your day-to-day life?”:

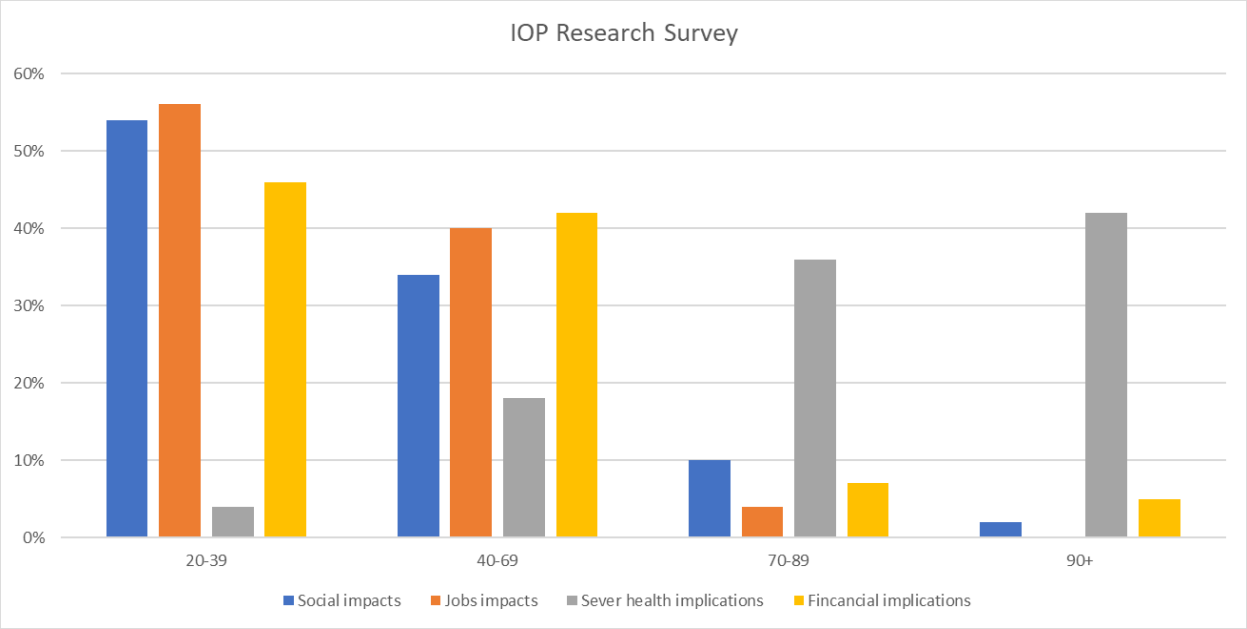


Figure 4. Bar graph showing survey results.

It is clear from the data above that most people who are aged between 20-39 are concerned about jobs and financial impacts, and most people aged 70-90+ reported to be most concerned about health implications. For this reason, we decided to prioritise the implementation of COVID-19 and other disease graphs and graphs related to the jobs market and other financial figures including stocks, exchange rates, real estate and unemployment.

A large number of people reported that they had used websites such as WHO, NSW Health and Our World In Data. Most people expressed their dislike for those websites for the following reasons:

- *Often cluttered and hard-to-read graphs and information*

To address this, we have implemented very simple, large colour-coded graphs with short lines of information conveniently located in a modal section on the right side of the screen. This will prevent cutteredness and overlapping of data to make it easy for the user to guide their eyes over the screen

- *Website is too bright and white to look at when in a dark room, even at the lowest brightness*

We decided to add a light mode and a dark mode which seamlessly transform the entire website into either a sleek, energetic white theme, or a dim, jet black theme at the user's convenience.

- *Buttons and colours not very user-friendly*

The overall colour theme will be a modern blue-red combination, combining red logos and different shades of blue across the heat map, header/footers and the modals. The buttons will be dark blue to blend into the blue background theme and will also turn red when hovered over. Users are assured to love this unique colour theme which stands outs from IOP's competing UI's

- *Too much scrolling through long lists of information*

We've implemented a no-scroll application. This means absolutely no scrolling through long pieces of important data. Instead, everything is summarised and simplified into large graphs and small, easy-to-read toolbars which enable users to comfortably switch between different graphs with ease.

- *Better layout and organisation of the page*

Each component on every page is separated evenly by white spaces and specifically placed in the selected locations to maximise ease of use and minimise wastage of time.

- *Could show other types of information too, often which are impacted by disease cases*

Even though IOP primarily provides past, current and future information about diseases, we have added graphs that provide past, current and future data on unemployment, average salaries by industry, stocks, exchange rates, real estate figures, and unemployment rates. All in one simple application.

2.2. Requirements and User Stories

Section 2.2 contains the requirements and user stories for our data analytics application as well as our API based on the findings from our research.

2.2.1. Landing page

The landing page is designed to give a simple overview of the broad function of the IOP application. It contains simple and easy to understand single-lined text as well as illustrations to give the user an idea of the purpose of the application. A begin button is located below the images to start using the API
User Story:

As a first-time user of IOP, I want to know what the purpose of IOP is without having to figure it out myself, or read through a long page of information.

User Case:

A user will hover over the first diagram on the left and read the short first step on how to use the API. The user will then hover over the next diagram on the right and do the same thing. Once the user understands the purpose of the API, they will click on 'Begin' to start using the API.

Screenshots (turn over):

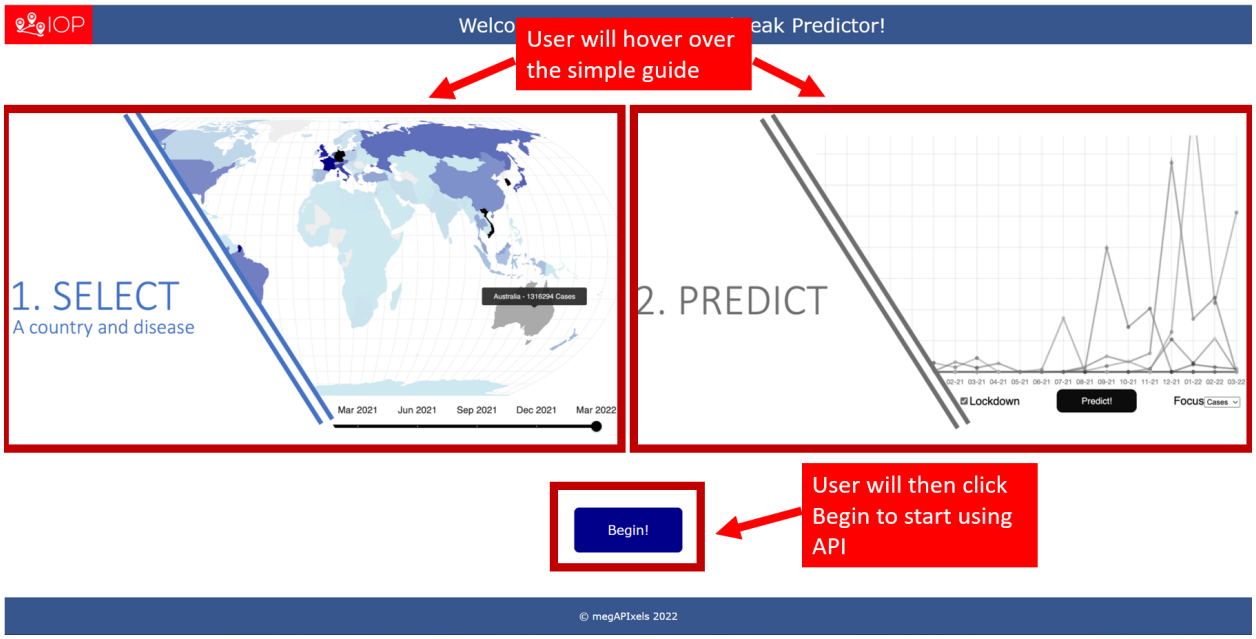


Figure 5. The landing page

2.2.2 Map Page

The map page acts as the home page of IOP. It contains a world map where the user can select the country of choice to view past, present and future information about diseases and financial figures. The countries will be colour-coded to show the amount of new cases per country. Countries with fewer cases will be lighter in colour, whereas countries with more cases will be darker. This allows the user to easily compare the number of cases in every country. It also includes the time slider at the bottom, allowing the user to select the month and year, as well as a modal on the right to display key information.

User Story:

As a user, I would like to hover over today's total number of cases of COVID-19 in my home country, Italy. I want to then see expanded details of COVID-19 numbers where I live, Australia. I also would like to travel back in time and compare the case numbers for each country

Use Case:

Consider the current month/year to be April 2022. The user must first move the slider into the 'APR 2022' position. The user can hover over 'Italy' to quickly see the number of COVID cases in their country. The user can then perform a click on Australia to reveal information such as new cases, new deaths, percentage vaccinated and total vaccinated on the right-side modal.

The user can move the slider down at the bottom to move through time and see the heat map display a visual overview of cases for each country.

Screenshots:

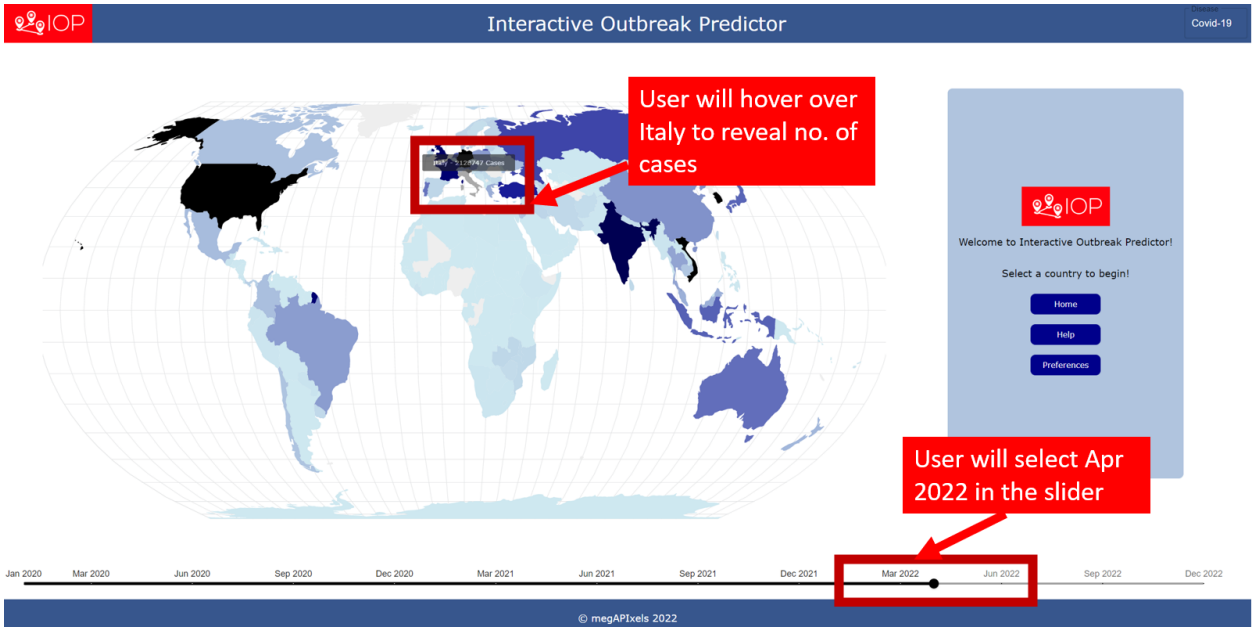


Figure 6. The slider

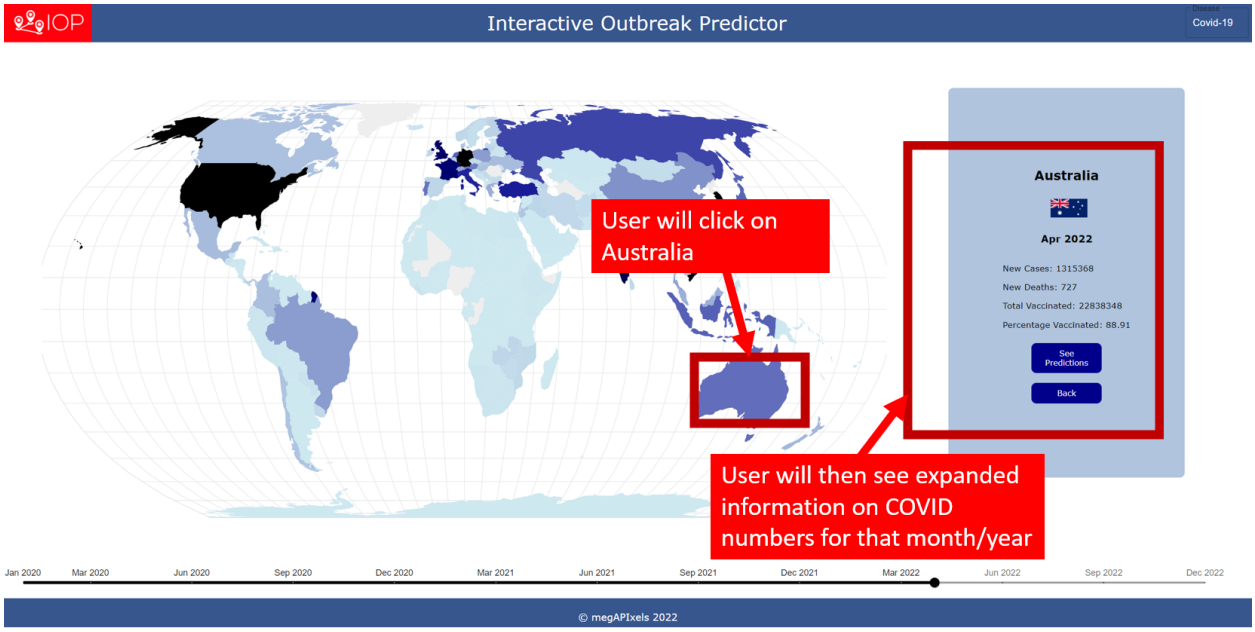


Figure 7. The modal

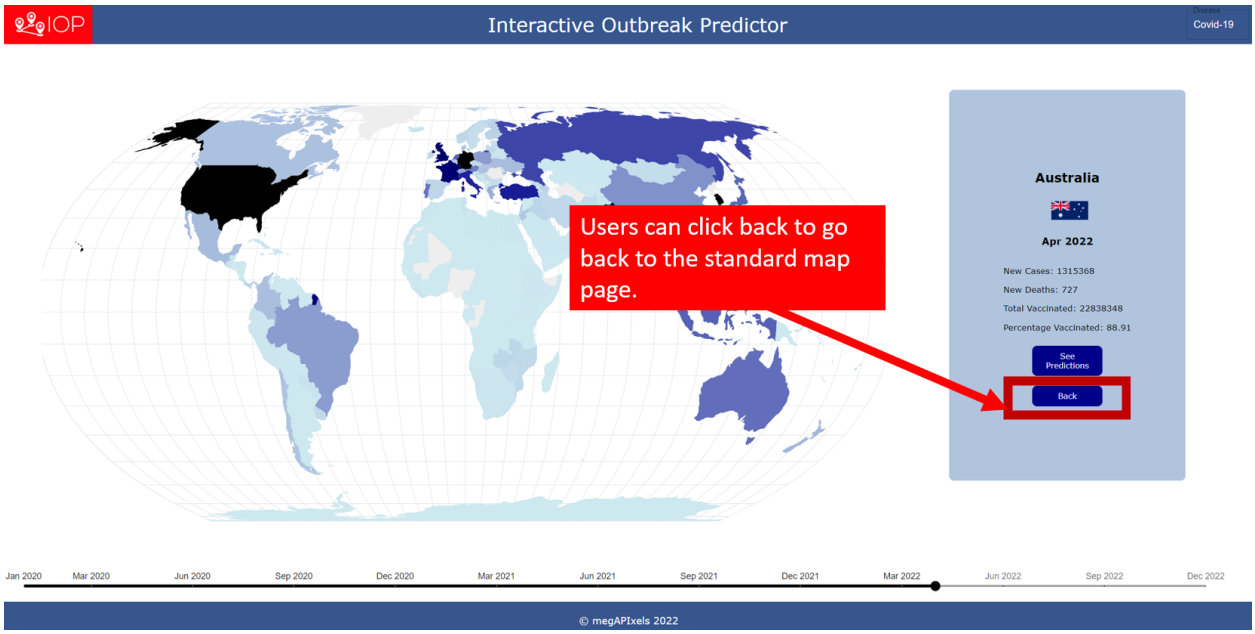


Figure 8. The back button

2.2.3. Country/Predictions page

After selecting the desired country as aforementioned, the user can access the predictions page. This page shows graphical information about various types of figures, including disease cases/deaths for both the country and subregions, average salaries by industry, stocks/exchange rates, unemployment rates and house price indices. The graphs also extend by a few months to show predicted trends in the figures. For the disease graphs, users can experiment with the checkboxes to see how factors may affect the number of cases or deaths.

User story:

As a user, I would like to see the graph of COVID-19 cases in my country, Australia, and I would like to see how mask wearing and lockdown may affect those numbers moving forward. I also want to see jobs, stocks, unemployment and real estate figures for my country.

Use case:

The user must click on ‘See predictions’ after selecting Australia as the country. After the graph loads, the user should check the ‘masks’ and ‘lockdown’ checkbox and click ‘predict’ to see how the graph data changes. The user can then click on the different tabs in the toolbar (Jobs, Financial, Unemployment, Real Estate) to view the corresponding graphs for the selected country.

Screenshots :

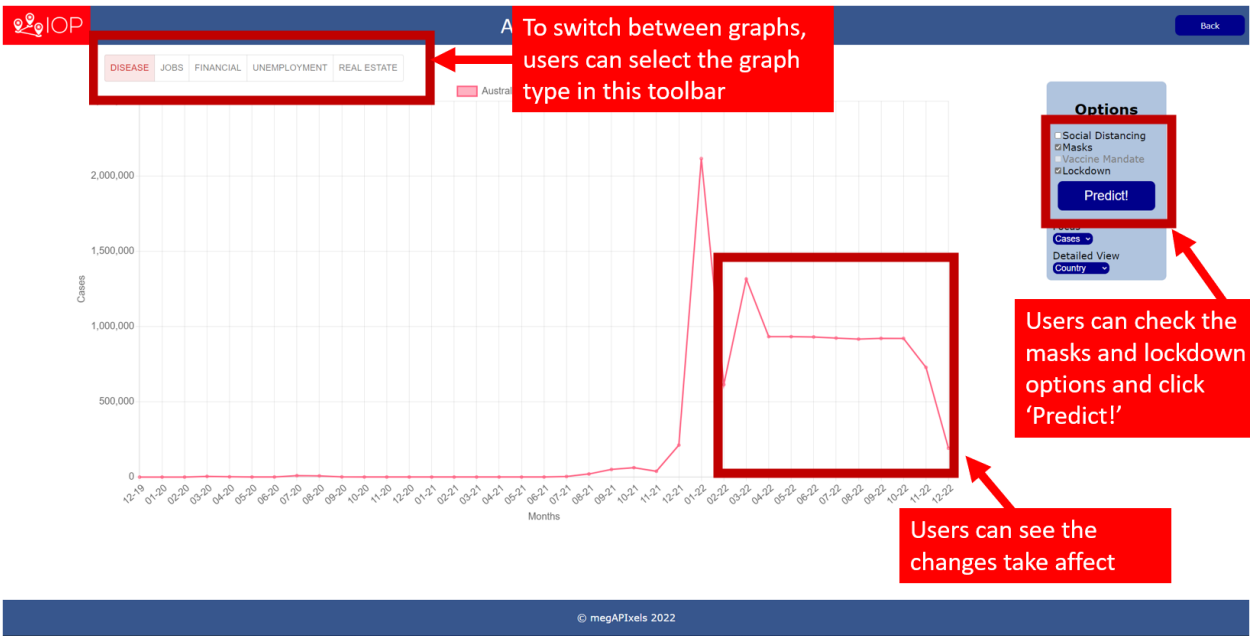


Figure 9. Country page graphs: Diseases

2.2.4. Country Unemployment Predictions

User story:

As a user, I would now like to see what the unemployment rate is expected to be in August 2022.

Use case:

The user clicks on ‘Unemployment’ in the toolbar above to view the unemployment rate for August 2022.

Screenshots:

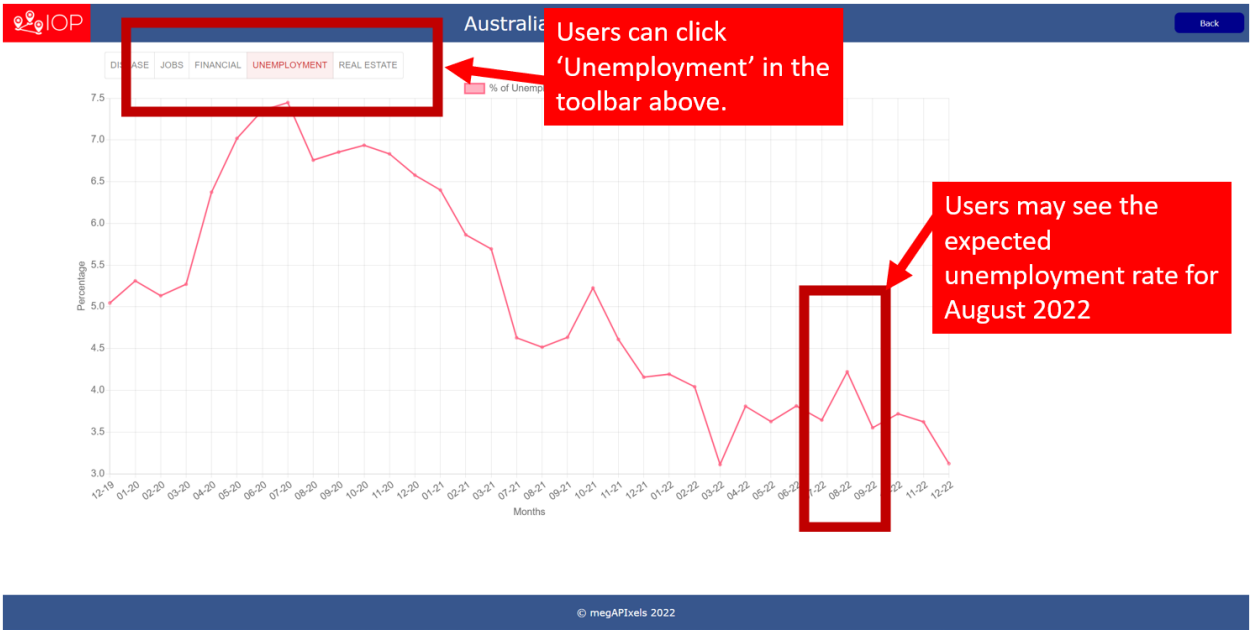


Figure 10. Switching between graphs

2.2.5. Graph Predictions

User story:

As a user, I would like to see the trend in the Average salaries in my field of work as a fast food worker.

Use case:

The user clicks on ‘Jobs’ in the toolbar above and then selects ‘Hospitality & catering jobs’ in the dropdown on the right. The salary graph for that industry will appear

Screenshots:

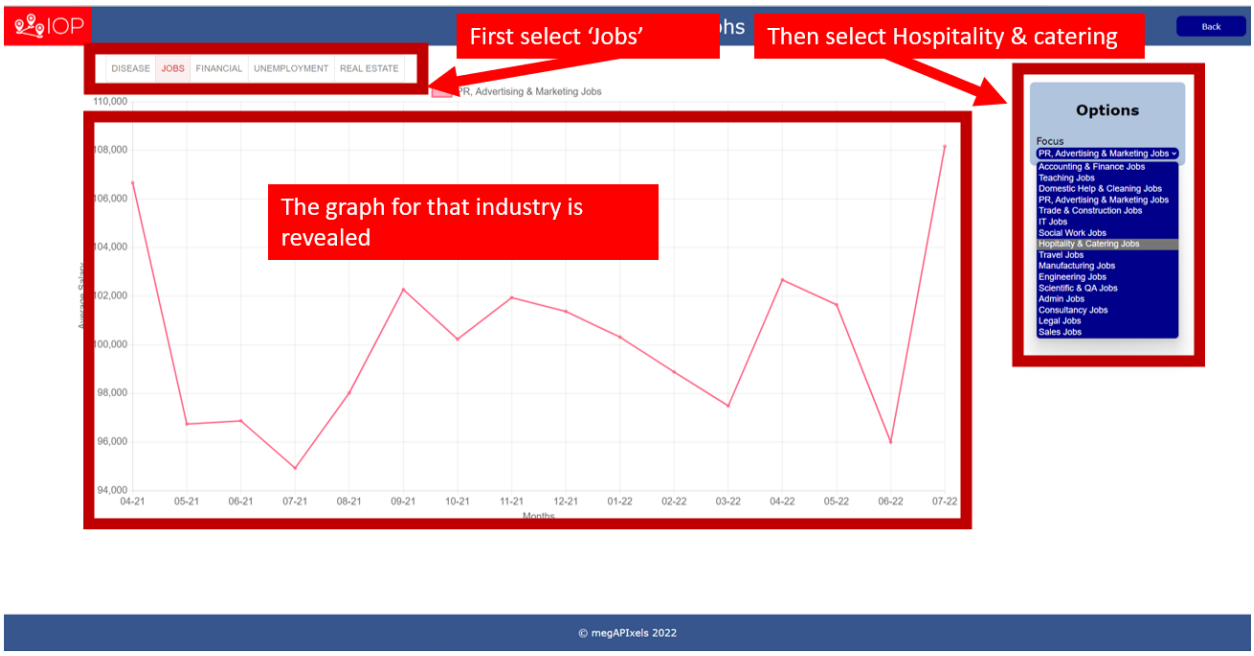


Figure 11. Jobs Graph

User Story:

Now, I would like to see stock figures and exchange rates relative to US dollars in my country.

User case:

The user selects ‘Financial’ in the toolbar above. Then, select ‘Exchange rates’ from the dropdown menu on the right to show the graph containing exchange rates relative to USD. Select ‘Stocks’ from the dropdown menu to then display the graph showing stocks figures

Screenshots (turn over):

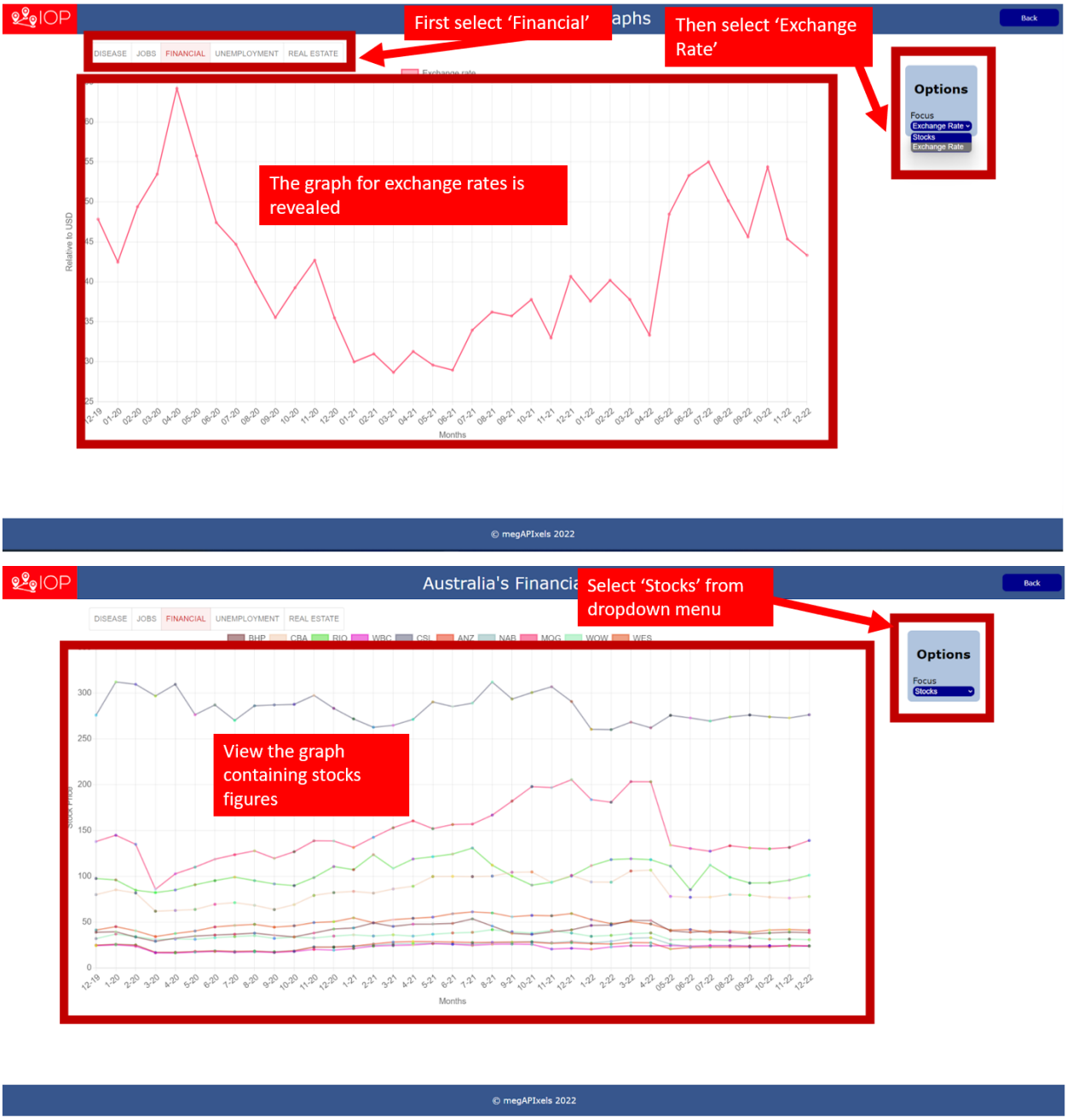


Figure 12. Finance Graphs

<u>User story:</u>

As a user, I would like to see the Real Estate trends for Greater Sydney in House Price Index (HPI).

Use Case:

The user selects ‘Real Estate’ from the toolbar atop the page. The user then follows the plot for Greater Sydney on the Real Estate graph (Coloured Fluorescent green below)

Screenshots:

Figure 13. Real estate graphs

User Story:

As a user, I would like to see how social distancing might affect future COVID deaths in my state of NSW.

User case:

The user selects ‘Disease’ in the toolbar above. Then, select ‘Deaths’ from the ‘Focus’ dropdown menu on the right, then select ‘Subregions’ from the ‘Detailed view’ dropdown and then check ‘Social distancing’, then click ‘Predict!’ to show the graph of COVID deaths and how social distancing affects future predicted numbers. The user can follow the coloured line corresponding to the key above to see numbers for the state of NSW (Coloured as Cyan below)

Screenshots (turn over):

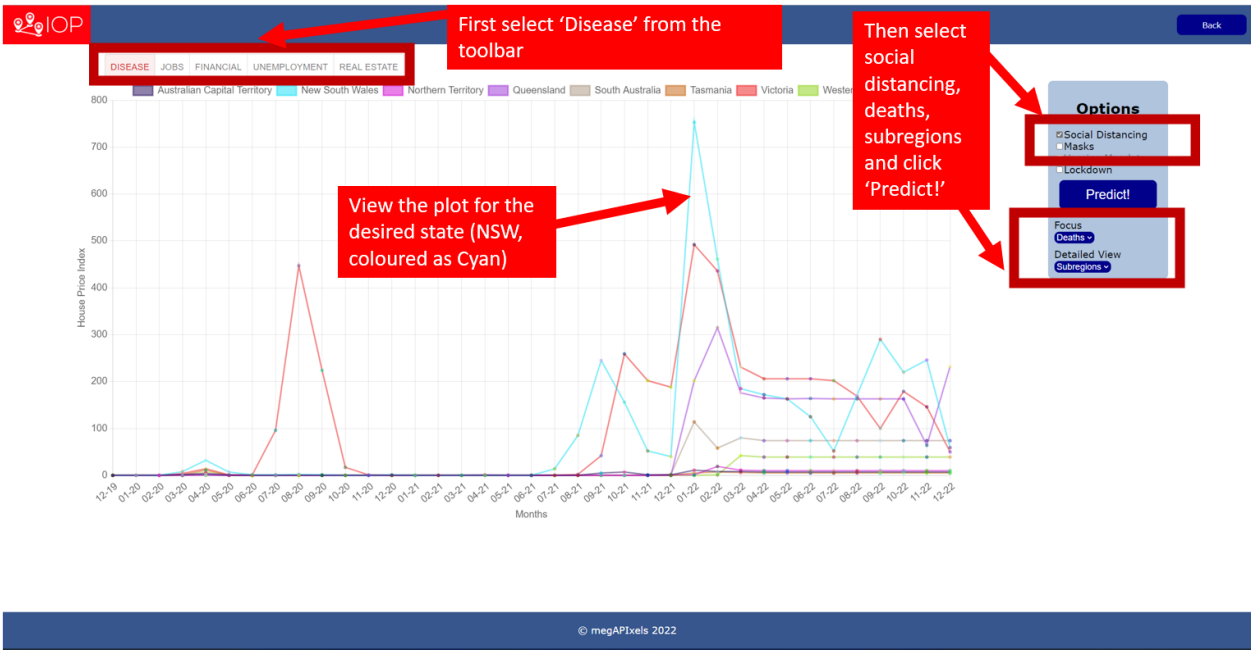


Figure 14: Disease predictions and subregions

2.2.6. Preferences page

After going back to the homepage, users may want to visit the preferences page, where they can change the theme of the UI and subscribe to email alerts.

User story:

As a user who is committed to refer to IOP regularly, I would like to subscribe with my email to receive alerts about COVID cases in my country, Australia.

Use Case:

After going back to the map/home page, the user clicks on 'Preferences' and goes to the preferences page. The user then inputs a first name, last name, valid email address, Australia as country, COVID-19 as disease.

Screenshots (turn over):

Figure 15. Preferences page: Subscribing to alerts

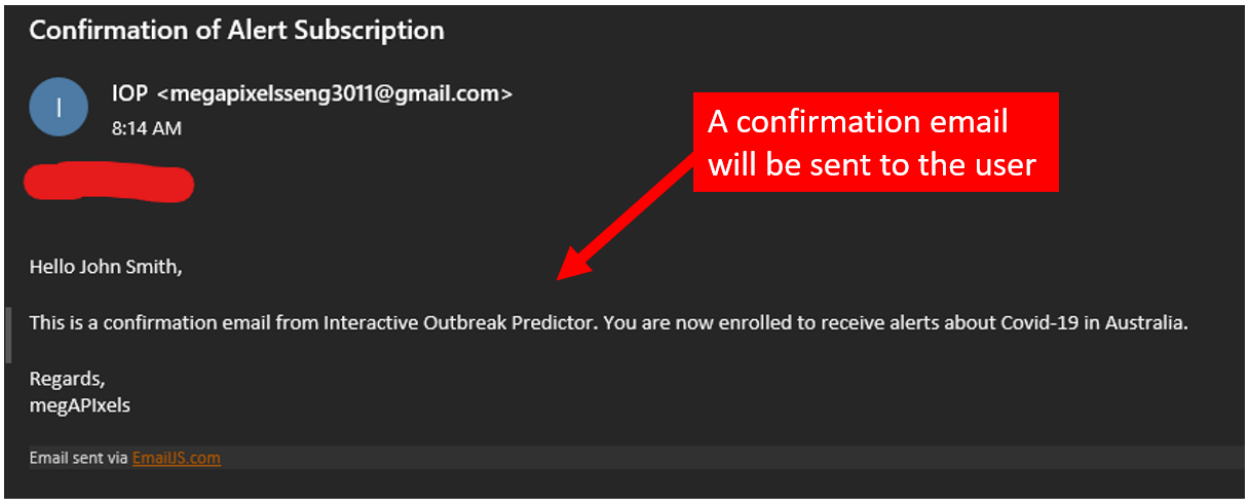


Figure 16. Confirmation email

User story:

As a user, I would like to change the API appearance to ‘dark mode’, because I think it looks cooler than white mode.

Use case:

In the preferences page, the user can check the ‘Dark mode’ option to switch the screen to dark mode. The reverse will be applied if ‘White mode’ is selected.

Screenshots:

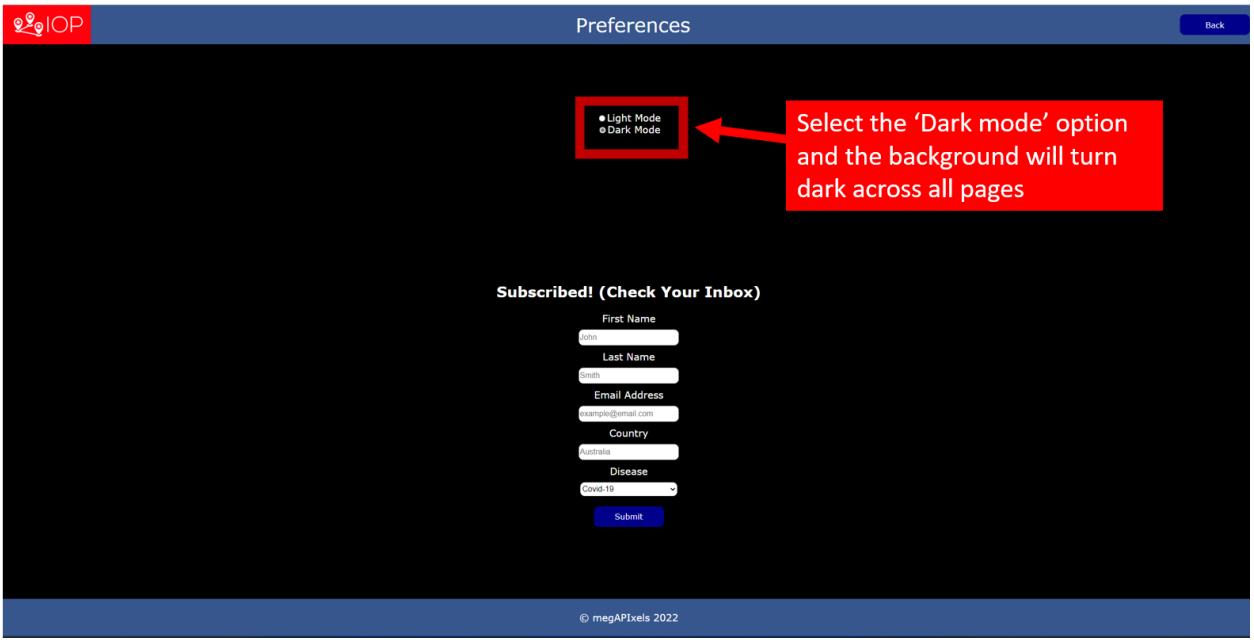


Figure 17. Dark/Light modes.

2.2.7. Help page

The help page contains a guide on how to use all of IOP’s features in detail. Each section is marked with section numbers. Images and GIFs are included to help visually demonstrate what was said in the text. The content section at the top contains anchor links to automatically transport the user to specific parts of the page.

User story (turn over):

As a user, I'm not too sure what IOP is about or how to use it, so I want to be able to read about it to get started.

Use case:

When on the help page, the user will scroll down (or use the anchor links) to section 1 and 2 to learn more about IOP and the basics of the UI and functionality.

Screenshots:

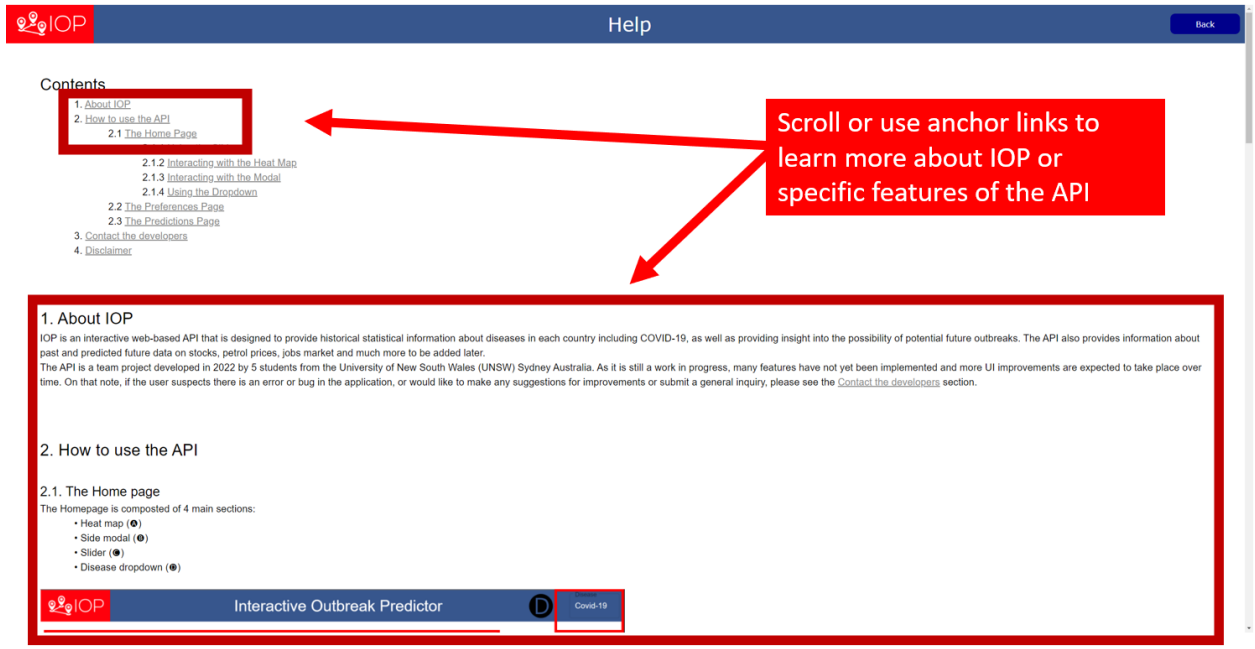


Figure 18. Help Page

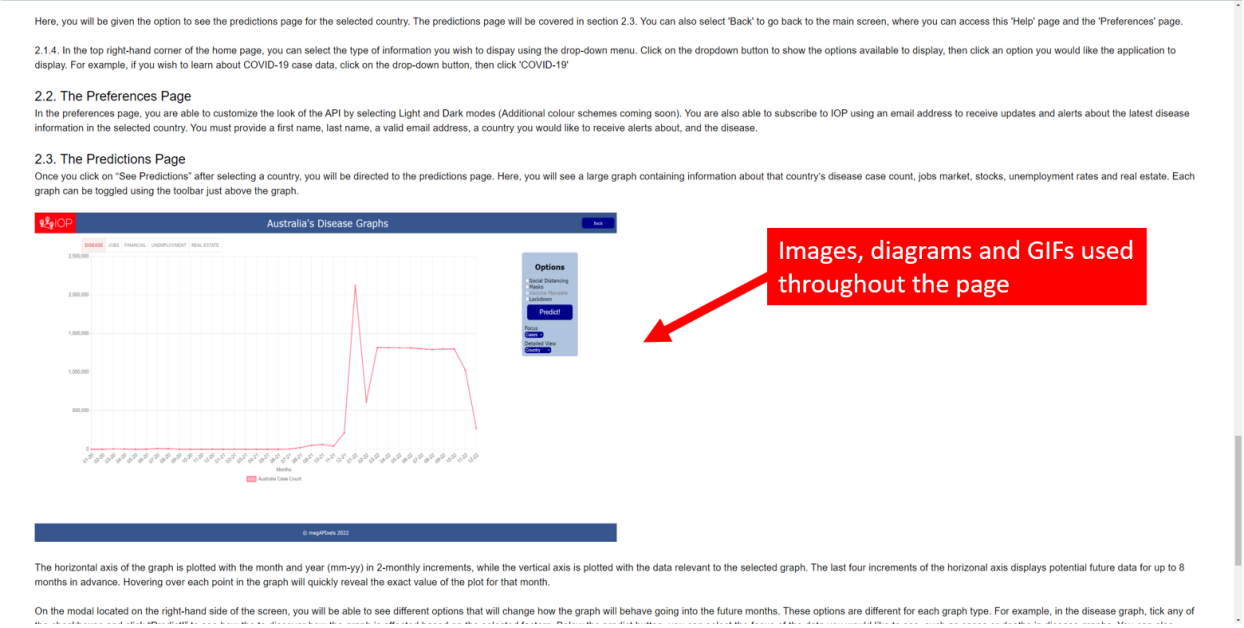


Figure 19. Help Page contd.

2.2.8. API User Requirements

Global Covid Endpoint

User story:

As a user, I want to be able to receive historical covid data for all countries since the start of the pandemic, so that I can use a country's data in my own application.

Use case:

When an API call is made to the endpoint, the user receives a JSON output of historical covid data including cases, deaths and vaccination numbers for all countries.

Country Covid Endpoint

User story:

As a user, I want to be able to receive historical covid data for a given country, so that I can use a country's subregion covid data in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of covid data for a country's subregions.

Exchange Rate Endpoint

User story:

As a user, I want to be able to receive historical exchange rate data for a given country, so that I can use a country's past exchange rate data in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of historical exchange rate data for a country.

Jobs Endpoint

User story:

As a user, I want to be able to receive historical job salary data for different industries, so that I can use a country's historical job data in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of historical job salaries for a given country.

Real Estate Endpoint

User story:

As a user, I want to be able to receive historical real estate data for a country's subregions, so that I can use the real estate data in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of historical real estate data for a given country's subregions.

Stocks Endpoint

User story:

As a user, I want to be able to receive historical stock data for a given country, so that I can use the data in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of historical stock prices for the top ten stocks of a given country.

Unemployment Endpoint

User story:

As a user, I want to be able to receive historical unemployment data for a given country, so that I can use the unemployment numbers in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of historical unemployment figures for a given country.

Future Country Covid Endpoint

User story:

As a user, I want to be able to receive future predictions on covid data within a given country, so that I can use predictions for a country's subregions in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of future covid data for a country's subregions.

Future Global Covid Endpoint

User story:

As a user, I want to be able to receive future predictions on covid data for all countries, so that I can use the covid predictions of different countries in my own application.

Use case:

When an API call is made to the endpoint, the user receives a JSON output of future covid data including cases, deaths and vaccination numbers for all countries.

Future Real Estate Endpoint

User story:

As a user, I want to be able to receive future predictions on real estate prices for a given country, so that I can use the subregion prediction data in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of future real estate data for a country's subregions.

Future Salaries Endpoint

User story:

As a user, I want to be able to receive future predictions of jobs salaries for different industries, so that I can use the prediction salaries for a given country in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of future jobs data for a given country.

Future Stocks Endpoint

User story:

As a user, I want to be able to receive future predictions on stock prices for a given country, so that I can use the stock price predictions in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of future stock prices for a country's top ten stocks.

Future Exchange Rates Endpoint

User story:

As a user, I want to be able to receive future predictions on exchange rates for a given country, so that I can use the exchange rate predictions in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of future exchange rates for a country.

Future Unemployment Endpoint

User story:

As a user, I want to be able to receive future predictions on unemployment numbers for a given country, so that I can use unemployment predictions in my own application.

Use case:

When an API call is made to the endpoint, the user provides a country query parameter and receives a JSON output of future unemployment data for a given country.

Report Endpoint

User story:

As a user, I want to be able to obtain a specific report based on its report ID, so that I can easily get a certain disease report's data.

Use case:

When an API call is made to the endpoint, the user provides a report ID query parameter and receives a JSON output of the report's disease data.

Search Endpoint

User story:

As a user, I want to be able to search for articles and reports based on a start date, end date, search terms and location, so I can receive data tailored to my requirements.

Use case:

When an API call is made to the endpoint, the user provides a start date, end date, search terms and location query parameters and receives a JSON output of the corresponding article and report data.

Article Endpoint

User story:

As a user, I want to be able to obtain a specific article based on its article ID, so that I can easily get a certain article's data.

Use case:

When an API call is made to the endpoint, the user provides an article ID query parameter and receives a JSON output of the article's data.

Live Endpoint

User story:

As a user, I want to be able to check whether the API is live and running, so that I know that it is working and usable.

Use case:

When an API call is made to the endpoint, the user receives a JSON output indicating whether the API is live or not.

2.3. Scenarios

The following personas have been constructed to comprehend the lifestyle of potential users within our demographic.

Table 1. Personas

Intro	Occupation	Difficulties	Main concern from COVID-19
Housam Age: 31	Pharmacist	-Works long hours at work	- Not keeping up to date with disease trends and communicating them with pharmacy customers.
Lia Age: 26	High school teacher	-Does not often have time use disease reporting websites, especially while teaching/marking homework	- Concerned about potential future outbreaks and the risk it will pose to her job and her students' academic and mental wellbeing

Mariana Age: 51	Office manager	-Must communicate with office employees about changing COVID-19 trends and rules	- Concerned about the impact of future outbreak on the business she manages, as well as deflation of house prices
Margaret, Grandmother Age: 76	Grandmother	-Not comfortable with using technology -Prone to asking her children and grandchildren for help with using software. - Suffers from Mesothelioma	- The effect it will have on her health as she is among the age group with the most deaths.

Housam is a 31 year old pharmacist who lives and works in Western Sydney. He works long hours at work and needs to keep up to date with the latest information regarding COVID-19 cases so that he can effectively communicate them to his customers, many of whom are from the older generation with a lack of understanding of the virus.

Given *– I access the URL and I am on the Home page*

When *– I click on 'Australia'*

And *– I click on 'See Predictions'*

Then *– I can see a graphical overview of the number of cases of COVID-19 in my country.*

When *- I click 'Subregion' in the dropdown in the modal*

Then *– I can see past, current and future COVID-19 cases for my state of NSW*

Given *– I am on the preferences page*

When *– I input my first name, last name, valid email address, country, disease.*

And *– I click on 'Submit'*

Then *– I will get a confirmation email and begin to receive notifications about COVID-19 cases in my country.*

Lia is a 26 year old Year 12 teacher who lives in Sydney and interacts with her students on a daily basis. She wants to be able to easily access COVID-19 case data on her computer at work but also keep up to date during very busy times, such as writing/marking final exams as well as during her classes.

Given *– I access the URL and I am on the Home page*

When *– I move the slider to the current month and year*

And *– I hover over 'Australia'*

Then *– I can see the exact number of cases of COVID-19 in my country.*

When *- I click on 'Australia'*

Then *– I can see new cases, new deaths, total vaccinations in my country.*

Given *– I am on the preferences page*

When *– I input my first name, last name, valid email address, country, disease.*

And *– I click on 'Submit'*

Then *– I will get a confirmation email and begin to receive notifications about COVID-19 cases in my country.*

Margaret is a grandmother who resides in Seattle, United states with a respiratory disease called mesothelioma. She is 77 years old and stays at home, but has a family that goes out regularly. She is quite concerned about her health and thus would benefit from using IOP to see current COVID-19 case numbers and future predictions. However She may have difficulties with using the API when her family isnt home

Given *– I am on the Home page*

When *– I click on 'Help' on the side modal.*

Then *– I will see a guide on how to use the API*

When *- I click on '2. How to use the API'*

Then *- I will be transported to the '2. How to use the API' section.*

Given *– I am on the Home page*

When *– I move the slider to the current month and year*

And *– I hover over 'United States'*

Then *– I can see the exact number of cases of COVID-19 in the US.*

When *- I click on 'Deaths'*

Then *– I can see new deaths by country.*

When *- I click on masks and lockdowns*

Then *- I click 'Predict!'*

Then *- I will be shown the predicted future trends of COVID-19 cases*

3. Tracking Changes

As changes are decided upon based on feedback received, we have documented them as part of this section and on our Jira Task boards.

3.1. Jira Task Boards

Below is a graphical representation of our requirements on a Jira board.

The full boards can be found here:
https://unswseng.atlassian.net/jira/software/projects/SE3Y22G8/boards/40/roadmap?timeline=MONTHS&shared=&atlOrigin=eyJpIjoiNzhlNWU3ZjEyYjMyNGU3M2I1NTE2N2JlZjk1ZWY4MGMiLCJwIjoiaiJ9

3.2 Changes based on earlier feedback

Based on feedback from previous reports and presentations, many changes were made. Below is a comprehensive list of major changes incorporated as a result. No changes to the API had to be made as a direct result of testing.

Table 2: Feedback and subsequent changes

	Feedback	**Changes**
Design Details	More detail required for API design principles	Each principle expanded upon greatly, with each given it's own subsection
	Provide a more comprehensive explanation of all error codes	Added a table with each error code used in the API and their meaning
	Report can be unreadable at times with large sections of text	Headings and subsections added to all reports
		Added tables, diagrams, and screenshots throughout
		Use of confluence instead of a *pdf* file to

		separate sections
	Missing justification as to why *SQL* databases were chosen over *NoSQL*	Added justification for *SQL* over *NoSQL*
Management Information	Justification for each management tool used missing	Added justification and alternatives for each tool
	Unclear what *Jira* is being used for	Section on *Jira* and it's use greatly expanded upon
API	Does not use the error codes outlined. All errors are treated as 400.	Added a greater variety of error codes to each *Lambda* script and its corresponding endpoint
Analytics Tool	Our main selling point of predictions and that ability to add restrictions to data is not highlighted	Added landing page telling users about how to use the tool, highlighting predictions
		Added predictions to the heat map page which is also the home page
	No customizability/personalisation for users	Added ability to switch between light and dark mode
		Added feature for users to subscribe to alerts. We decided that allowing users to create accounts would not add value to our application, hence chose to allow for subscription instead.
	UI/UX is unprofessional	Extensive overhaul of the colour scheme and layout of the tool after research into example UI/UX on *Dribble*
Repository	Structure not quite matching specifications	Missing subdirectories under Phase_1 and Phase_2 added
Swagger	Lack of information about required input format for each endpoint	Required input format added to the Swagger documentation

4. Prototyping

Our design phase involved creating lo-fi and hi-fi prototypes that would provide a basis for our outbreak predictor. The prototypes shown below demonstrate the development of our design as well as the introduction of new ideas in our user interface.

4.1 Low-Fidelity Prototypes

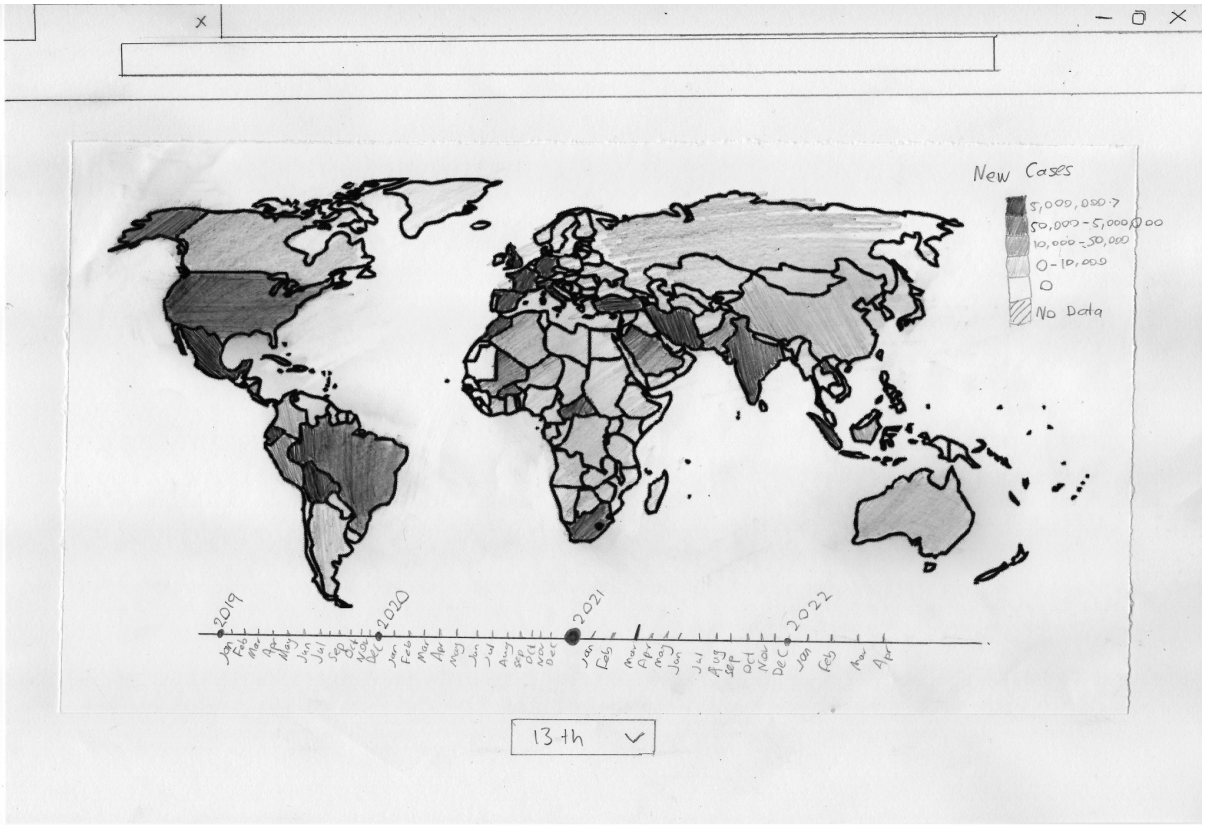


Figure 20: Lo-Fi Map Screen Prototype

In Figure 20 above, an initial design for our global map screen is portrayed, depicting a heatmap layout of countries in an interactive fashion alongside a colour-coded legend. Beneath the map is our original idea for the slider which contained every month for the years between 2019 - 2022, as well as a drop down from which the user could select a specific date. However, in our final implementation the idea to display a heatmap for specific days was converted to a monthly fashion due to missing data as well as a perceived lack of value.

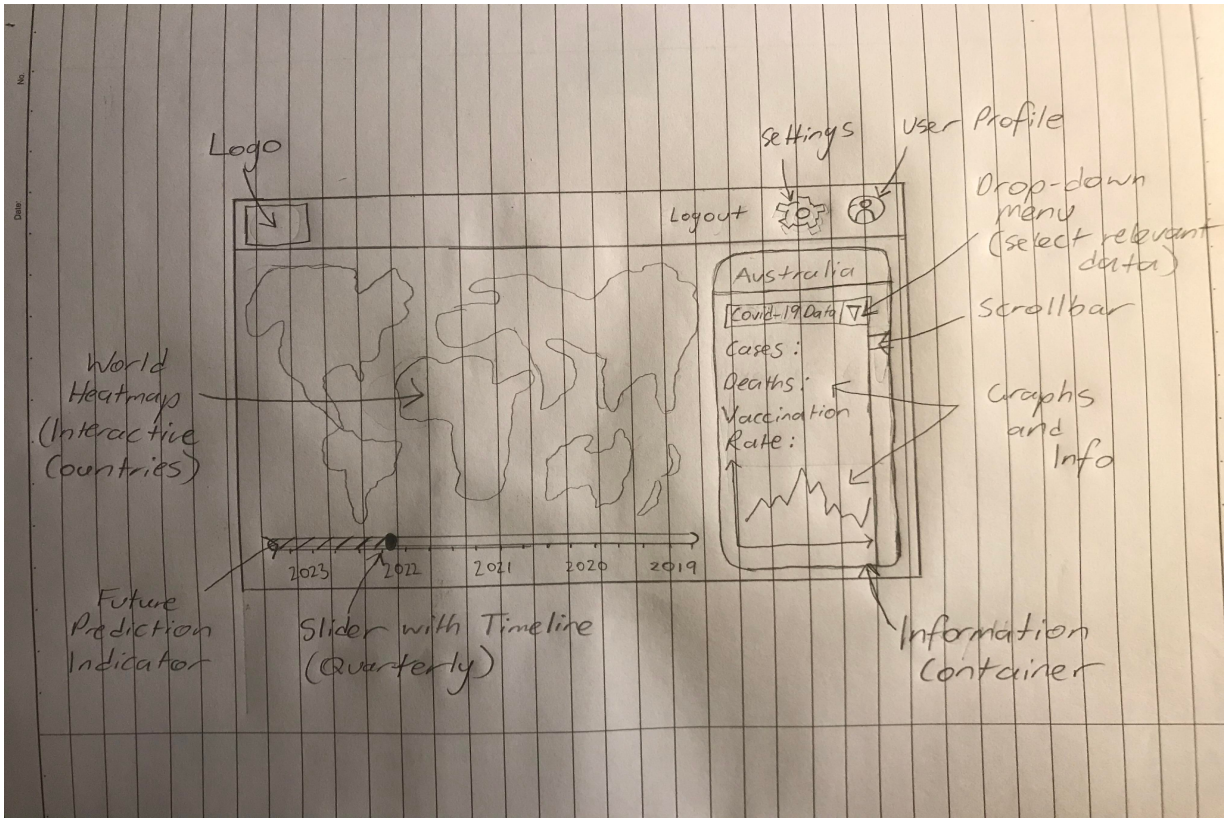


Figure 21: Lo-Fi Map Screen prototype

For our second lo-fi prototype depicted in Figure 21 above, a more developed map page closer to our final implementation is shown depicting features that would allow the user to interact with the page to view different historical and future data. In conjunction with the heatmap and slider we have an information section that displays the cases, deaths, vaccination rate and graphs for a chosen country. Initially, the user was going to be able to access a range of data from the drop down menu in the information container however to reduce visual noise, separate tabs were used in our final implementation.

4.2 High-Fidelity Prototype

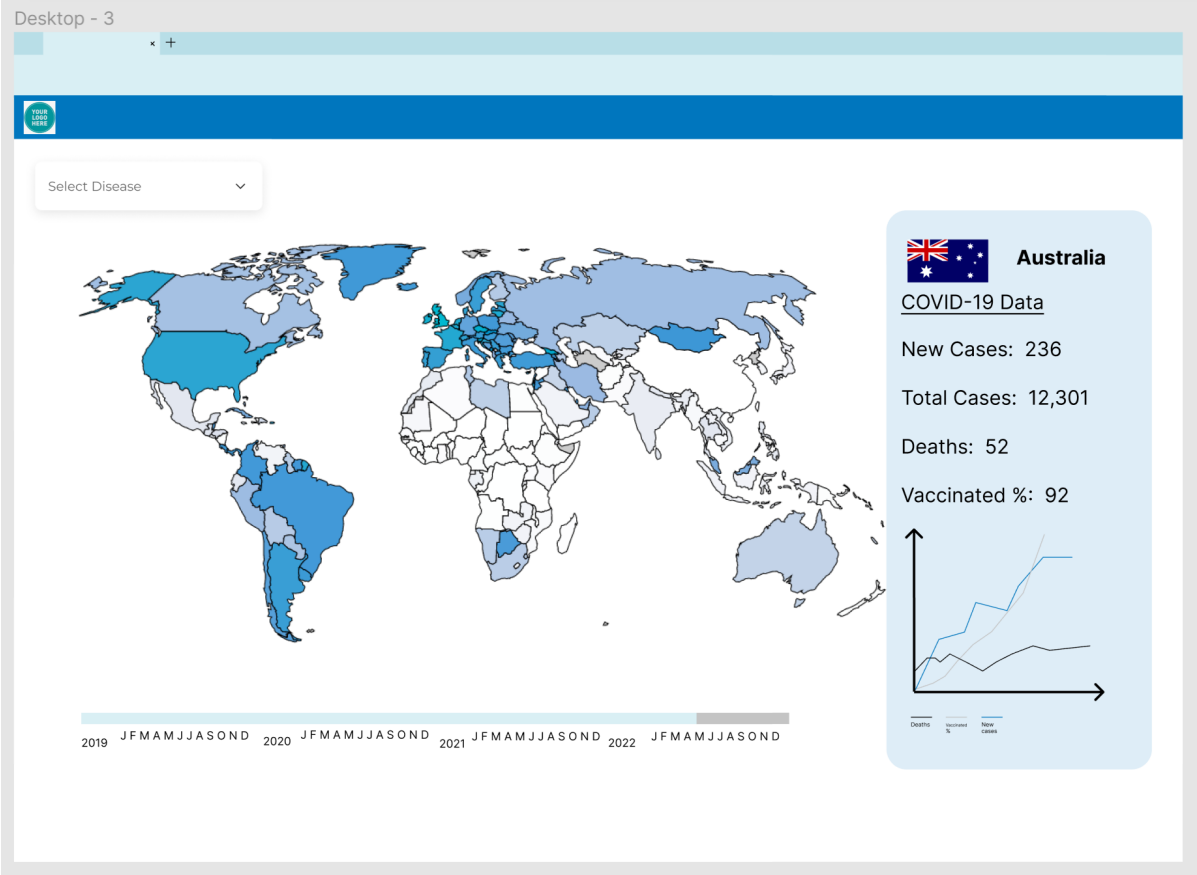


Figure 22: Hi-Fi map screen prototype

Figure 22 above displays our map screen created in Figma and depicts the features shown in Figures 20 and 21 as well as some interactive features including the drop down selection. However, the map screen does not display all of the features included in our final implementation such as different datasets and the prediction feature for graphs included in different tab pages.

5. System Design and Implementation

This section contains the features that we have implemented in our application as well as the technical architecture for the system. We have also included future features that could be developed and decided upon following our implementation phase.

5.1. Summary of Key Achievements

The following sections portray our system's key achievements in terms of the features it delivers as well as future possibilities.

5.1.1. List of Achieved Features

- Heat-map of COVID-19 cases over time
    - Slider allowing a selection for a month from the start of the pandemic (12/19) to future predictive data (12/22)
- Country snapshot
    - Ability to select a country on heat-map, displaying a snapshot at the chosen month of COVID-19 cases, deaths, vaccinations (total & percentage)
- Graphs with historical and predictive data for various data sets of a chosen country
    - COVID-19 cases and deaths for the chosen country
        - All countries
    - COVID-19 cases and deaths for each subregion of the country
        - Australia, Canada, France, Great Britain, China, Netherlands, New Zealand, Denmark
    - Average Job Salary by Industry
        - Australia, USA
    - Stock price of top 10 stocks
        - Australia, USA
    - Exchange rate relative to USD
        - All countries except USA
    - Exchange rate relative to AUD
        - USA
    - Unemployment percentage
        - All countries
    - House Price Index for each subregion + national average
        - Australia
- Help page with extensive documentation on application usage
- Landing page with graphical summary of application usage
- Customizability with light and dark mode
- Ability for users to subscribe to alerts for new predictions relating to COVID-19

5.1.2. List of Achieved Implementation

- API deployed on *AWS*
    - Postgres database instance hosted on *AWS RDS* and filled with historical and future data from external APIs and sources
    - Parsers deployed on *AWS Lambdas* for each endpoint
    - API endpoints hosted on *AWS API Gateway*
    - Log files for each *Lambda* script, the database instance, and the API logged on *AWS CloudWatch*
- Analytics tool deployed on *CloudflarePages* and integrated with *GitHub* repository

5.1.3. List of Possible Future Features/Improvements

Below is a brainstorm of ideas that we had to implement if there were more sprints to come and the project were to continue. The ideas are listed in order of priority. For country-specific features, popular countries such as USA, UK, Canada, China, India, France, Germany would be prioritised first as they would be more likely to be used than smaller countries such as Fiji.

- More comprehensive data for graphs and predictions
    - Subregion COVID-19 data for all countries
    - Average job salaries by industry for more countries
    - Stock prices of top 10 stocks for other major markets (eg. Shanghai, Hong Kong, Europe, Japan)
    - The ability to choose exchange rates to compare to instead of a standardised USD
    - House Price Index for all countries
    - Other graph datasets (eg. Greenhouse gas emissions over time, petrol prices, cryptocurrency prices)
- Extending to other major diseases (eg. Malaria, Yellow Fever, Meningococcal)

5.2. API Architecture and Design

For this project, we have developed a web API to format and return articles and disease reports from the ProMED website, as well as many other external APIs (outlined later). We use Python scrapers that extract data from each external source which is then stored in tables in a web-based SQL database. Accordingly, the database parser returns data in JSON form at the request of our API. Both the scraper and parser scripts are run from an *AWS Lambda* function. The API itself is RESTful and hosted on *AWS API Gateway*. The API is called by any external client application as well as by our own application. This architecture is displayed diagramatically in Figure 23 below.

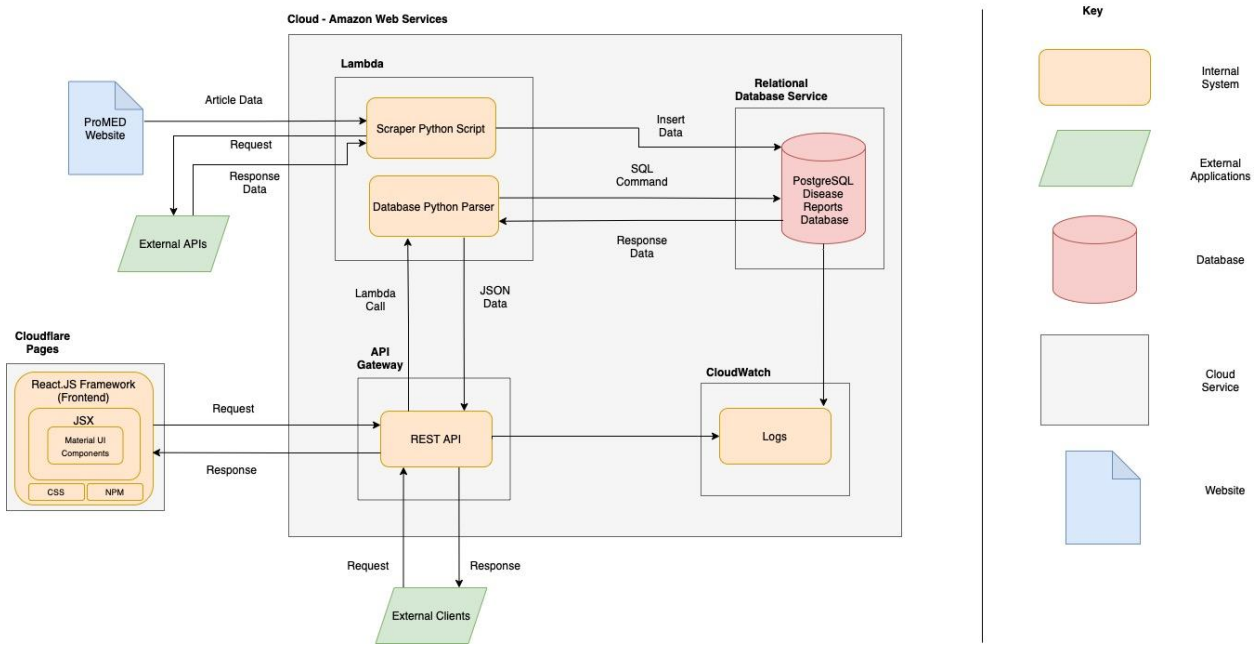


Figure 23: Architecture Diagram

5.2.1. REST API Design Principles

In order for users to interact with our API smoothly and efficiently, we wanted them to be able to pass query parameters and obtain valid results as easily as possible. For this reason, we chose to create a RESTful API that would adhere to a number of common design principles. This included the use of well-known HTTP methods, meaningful endpoint names, versioning, and providing examples.

5.2.1.1 HTTP Endpoints

Firstly, all API endpoints utilised common HTTP methods and responses. Namely, since the user/application was simply using our API to retrieve information from a database, each of our endpoints used a GET request and received URL query parameters to define the search. The response status codes returned by our API also align with the HTTP standard for codes such as 200 to indicate a successful response, 400 for a bad request or 404 for a non-existent resource.

The endpoint names themselves have also been designed to be logical and meaningful so that they simply describe their function. An example of this is the “/proMedApi/globalcovid” endpoint which the user interacts with to search for covid data for all countries globally.

5.2.1.2. Versioning

Additionally, the API’s design employed URI versioning to cater for the inevitable changes to the system. For example, this included changes in the response type or user request parameters to

be used to extract data. We used the “v[x]” syntax where the x represents an integer indicating the version of our api for example “/proMedApi/v1”.

5.2.1.3. JSON Responses

The requests handled by the API returned JSON responses containing information extracted from the backend database that holds the disease data scraped from the external sources. We used JSON responses as they are simple and widely used. Once the API has received its parameters, it performs a search and filter on the database. This database is filled with data from web scraping. On a successful call, the API returns the data as a list of JSON objects. On error, this list only contains one element: a JSON object describing the error that occurred. In either situation, an appropriate HTTP status code is also returned to help the user interpret the response. The user is able to easily collect results from the API in this JSON list format to extract for use in their own application.

5.2.1.4. Documentation and Examples

To further aid the ease of use of our API, we provided a number of sample HTTP calls and responses in our extensive *Swagger* documentation. The link to it can be found on the title page of this report. The documentation has been extended to also include endpoints implemented in phase 2. This allows external developers to also use our API for their own use. Additionally, some common errors and their responses are displayed in Table 3. This documentation outlines the structure of our API in a clear, graphical format. Additionally, *Swagger* also acts as a simple client to call our API and the user is able to view the sample input and responses in order for them to get a better idea of how to use the endpoints themselves.

Table 3: Error responses used by the API

Error	**Reason Encountered**	**HTTP Response**
Invalid input	The user has incorrectly formatted their URL query parameters or provided invalid input	400 Bad Request
Data fetching error	The server was unable to fetch data from the database but not due to input	500 Internal Server Error
N/A	Correct API usage	200 OK

5.2.2. Web Service Mode and Deployment

The API module developed was created using *AWS API Gateway*. Initially, we determined the requirements of the API and targeted it towards external consumers or developers who wanted to integrate it as part of their application. The functional requirements have been stated in the project context however in a non-functional sense it was expected that the API would have a

fast response time and reliable uptime which would be facilitated by *Amazon Web Services*. User privacy and system security is inherently dealt with by *AWS* due to the reliance on their web services.

5.3. Software Architecture and Justification

This section delves into the software architecture for our backend and frontend accompanied by a justification for our choices.

5.3.1. Python

We decided to develop our API backend in the Python programming language with its included standard libraries. We decided to use Python as the primary backend language for the API as all of our team members had some level of experience having done previous courses on Python such as COMP1531. In addition, it is relatively simple to work with to build API backends due to its readability, flexibility and versatility as well as having more portability than other open-source languages such as C. This means we can run the same code on different platforms without having to change anything.

Another reason we favoured Python over other programming languages is because of its compatibility with other tools and frameworks. Many of the popular tools and SDKs used to work with different areas of development are built for Python as the primary environment on which scripts are run. According to many sources including northeastern.edu, Python is the most popular programming language used by developers today. Because of this, there is a large active community of developers that use Python, meaning it was likely that any issue we ran into would have already been solved and highly documented.

5.3.1.1 Alternatives

However, this doesn't mean Python has no drawbacks. A disadvantage of Python is that it isn’t native to Android or IOS, so it would be unsuitable to deploy Python programs on mobile platforms. This did not affect our project however, since we were not deploying our API or application on an Android or IOS app. It is also slower than compiled languages such as C and C++ and is prone to runtime errors, making it more difficult to account for errors while developing..

Another programming language we could have chosen was Java. Although Java is also a very popular choice of language among developers, it wasn’t suitable for us because its parent company, Oracle, charges a monthly licence fee for using the Java Development Kit. It also does not have as many libraries as Python does.

5.3.2. Python Libraries

There are a number of Python libraries and SDK's we used to work with our DBMS, web server and UI front end development. To run SQL queries and functions on our backend, we used *psycopg2*. It is the go-to PostgreSQL driver for the python programming language. It has little to no alternatives and is highly used and documented. *Psycopg2* allows us to connect to the database from our python scripts and systematically extract information stored on the database in our desired format.

We looked at various web scraping tools to determine a suitable way to conveniently and efficiently extract data from the ProMed website into a collection that was easy for us to use for phase 1. We decided to go with *scrapy* as our scraping framework within Python. This was because it is the most popular python web scraping library, meaning it was highly documented, and allowed us to extract and collect data into different formats, including JSON, CSV and XML, providing us greater flexibility. Additionally, *scrapy* has built in spider features. The use of spiders allowed us to scrape the web more efficiently using a set of instructions we can define. *Selenium* is a web automation library, but has features that can allow web scraping. As it isn't primarily a scraping library, it is much more complicated to use as it is very overloaded with other features. Another key disadvantage of *Selenium* was that it only begins browsing the web page after it has fully loaded, which can significantly decrease the overall speed of the API especially if the client's internet connection is weak. *BeautifulSoup* is a very simple content parser which was also considered. It is much easier to use, however has very little functionality and cannot retrieve, parse, or extract data by itself. Hence *scrapy* was chosen as it was easy to use and provided enough functionality for our backend use case.

To extract reports from the scraped raw text, we used multiple libraries in combination. To match key terms we used the python regex library, *re*, to match the text for the provided list of key terms. This is the go-to pattern matching library that is robust and well documented, utilising regular expressions. Most members also had experience with its use. To extract dates and locations from the raw text, we used a library called *spacy*. This is a very sophisticated and overloaded library that analyses each word and phrase given in a string for a variety of languages. It could identify words such as 'London', 'New South Wales', and 'Sydney' as being locations, as well as dates in a variety of formats. This was the only library found that could identify location names and dates in a variety of types and formats. However upon testing we had found the library to be prone to occasionally overmatch dates and make mistakes, so we also utilised the *dateutil* library to parse all matched dates for verification. The parser function of this library matched provided strings in a range of formats into datetime objects. This allowed us to filter out poorly identified dates by *spacy* and also store dates in a standardised datetime format. An alternative to *dateutil* is *datetime* which is a python built-in library. However, it does not have the string matching capabilities that *dateutil* has and requires strings to be parsed in a strict format.

5.3.3. Data Prediction

For the data prediction functionality in phase 2 of our API, we used *statsmodels*. As no team member had experience using python prediction models, we ended up with *statsmodels* through trial and error over multiple online tutorials. Others trialled included *tensorflow*, and *Simpy*. Using a tutorial from *machinelearningmaster.com*, we were able to get a working prediction model for our data. We were not able to get working models using *tensorflow*, or *Simpy* as they were much more complicated to use.

5.3.3.1. Model

For the prediction model, we used an Autoregression algorithm that essentially uses machine learning to train the prediction model. Initially the differences in values over time are calculated and the transformed data is split into thirds in which 2 thirds of the data is used for training and the last third is used for testing. The training data also forms a historical array that is used in the predictions. After the Autoregression training is complete the algorithm begins to walk over time and create predictions while new observations are added to the historical array. The number of predictions formed are limited by the length of the test data as for each test observation added to the historical array a new prediction is created.

```
def createPredictions(series, growthEffect):
    # split dataset
    X = difference(series)
    size = int(len(X) * 0.66)
    train, test = X[0:size], X[size:]
    # train autoregression
    window = 6
    model = AutoReg(train, lags=6)
    model_fit = model.fit()
    coef = model_fit.params
    # walk forward over time steps in test
    history = [train[i] for i in range(len(train))]
    predictions = list()
    for t in range(len(test)):
        yhat = predict(coef, history)
        obs = test[t]
        predictions.append(int(abs(yhat + series[len(series) - 1]) * growthEffect))
        history.append(obs)

    return predictions
```

Figure 24: Prediction Algorithm

To apply restriction filters to the predictions model, we first had to find scientific research for the effect that each restriction had on the case and death numbers. Data from multiple peer-reviewed scientific papers were used, all of which were on PubMed:

- A study on the effects of social distancing and lockdowns in Brazil (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8093001/)
- A study on the effects of social distancing in USA (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7310657/)
- A study on the effects of mask mandates in USA (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8221503/)

These papers gave case growth reduction values, which were then applied to the corresponding growth in the prediction model:

Table 4. Growth reduction values of cases and deaths with different restrictions

Lockdown	Cases	0.29%
	Deaths	0.35%
Masks	Cases	0.14%
	Deaths	0.13%
Social Distancing	Cases	0.05%
	Deaths	0.07%

This method has many limitations as the studies were all carried out in Brazil and America. The growth rates would vary greatly depending on many factors such as population density, average age, average income, types of masks, and political ideologies. However, due to our limited knowledge in data predictions and the time constraints of the project, we decided that this method was sufficient.

It was decided that all data would be stored as monthly aggregates to reduce storage costs with the database, as well as maintain a reasonable level of performance for the heat-map functionality, which loads all historical and future covid case, death and vaccination data for each country.

5.3.4. External APIs

For phase 2 of the project, many external data sources were required. These correspond to the user stories in section 2.2.3. Many APIs require premium subscriptions or have rate limits. All APIs were chosen because they were free, had a high enough rate limit, and were easily accessible by our scrapers.

5.3.4.1. COVID-19 Case + Deaths

For COVID-19 case and death numbers for each country and some subregions, we used the *COVID-19-API* from *M Media Group*. It was the only free and limitless API that also provided historical data required for the model. It does however lack data for some countries such as

North Korea and Nigeria. It also only has case and death numbers for subregions of some countries, such as Australia and France. No free alternative for subregion data could be found.

5.3.4.2. COVID-19 Vaccination

Our global vaccination data comes from a GitHub repository run by *Govex* called *COVID-19*, which provides vaccination numbers for most countries. This was the most comprehensive and up-to-date source we could find. It consists of csv files which are updated hourly by automated bots. This is then scraped by our scrapers. This method is free and is also limitless. However, it is more complicated than calling APIs as done with the other data sources.

5.3.4.3. Job Salaries

Our data for average job salaries by industry was gathered from an API run by the UK job advertising website, *Adzuna*. It was the only free source found. It does however have many limitations as it only has historical data for a few countries, such as Australia, USA, UK, and Canada. Some data was also found to be inaccurate. However, as it was the only source found, it was used for the available countries.

5.3.4.4. Stock Prices

Our stock data for the top 10 stocks is obtained from *Nasdaq* for the US market, and *Investing.com* for the Australian market. A list of the top 10 stocks are manually listed as of the end of 2021, and the corresponding csv files with historical data are then downloaded and scraped. All financial APIs investigated either had very low usage limits or required a premium subscription, hence this more manual method of downloading csv files had to be used. Due to this being a more time consuming and labour intensive method, only data for the Australian market and the US market, due to being the world's largest market, were used.

5.3.4.5. Exchange Rates

Our exchange rate data relative to USD utilises the *CurrencyLayer* API to gather historical data. This API has a free usage limit of 250 requests per month. This was just enough to gather exchange rate data for each country and load it into the database. USD was used as a base for the comparisons as it is the primary currency used for trade internationally. It is also the only option that can be used under the free model. In the case of America, as USD compared to USD will always be 1, the comparison is instead with AUD and is obtained by calculating the inverse of the data for Australia.

5.3.4.6. Unemployment

Our global unemployment percentages are gathered from *WorldBank*. A csv file with historical unemployment values is downloaded, which is then scraped and loaded into the database. This was the only source found that had historical unemployment values for every country. It is free and limitless, however requires manual downloading of the csv file each month.

5.3.4.7. Real Estate Value

No accurate and free real estate global data could be found. *The Australian Bureau of Statistics* however had house price indexes for each subregion of Australia. This data was only updated quarterly and the most recent record was for the 4th quarter of 2021. As it is government run, it is free to use and limitless.

5.3.5. ReactJS

React is a powerful open-source JavaScript library maintained by Meta to create web based UI's. It is used to create popular websites such as Facebook and Netflix. It is a library providing a convenient way of building aesthetic user interfaces. Using *React* enables us to easily create dynamic web applications with more functionality. There were many advantages of using *React* that brought us to use it. Firstly, it was a library that everyone in the team had experience with. This made it easier for team members and eliminated the need for learning a new UI development library which saved crucial time over the course of the 10 week project. In addition, *React* enables reusable components which allow complex code to be written in blocks, making the application easier to develop and maintain. *React* is also significantly faster than pure JavaScript as it uses a virtual DOM which exists entirely in memory, making virtual components easier and faster to run. Only changes are applied to the original DOM, instead of re-rendering everything.

5.3.5.1. Alternatives

An alternative to *React* is *Angular* which is maintained by Google. *Angular* is a cross-platform web development framework which has better performance and is arguably considered more readable and portable than *React*. Although *Angular* is a popular choice of framework among web developers, we decided not to use it in our project for a number of reasons. Firstly, no team member was familiar with *Angular*. It would not have been time efficient for the team to learn *Angular* within the short 10-week time frame, for little advantage. For this reason, we decided to spare the time in learning *Angular* and chose React as our option.

Another alternative to *React* that we considered was *Vue*. An open-source front end JavaScript framework for designing UI's. Notable websites made using *Vue* include Gitlab and Trivago. Unlike *React*, *Vue* uses HTML instead of JSX. However, it also uses a virtual DOM just like *React*. Despite this, both *React* and *Vue* present similar performance. *Vue* enables developers to create websites that are light, fast and efficient and is usually recommended for new developers learning front end UI design due to the decoupling of HTML, CSS and JavaScript. Again, as *Vue* offered little benefit over *React*, the team decided that it was not worth spending the time learning *Vue*.

5.3.6. React Libraries

Some additional libraries used include *Node Package Manager (npm)* which is a global online open-source repository used to manage packages and dependencies for our project. It was

used to install, upgrade and configure packages. It was essential to our project as it helped prevent and resolve dependency conflicts by ensuring all versions of packages are the same to prevent bugs during development.

An alternative to *npm* is *yarn*. However, *npm* and *yarn* have few differences and since team members were more familiar with *npm*, we decided not to use *yarn* to reduce learning curves for team members in a short timeframe.

Material UI is a React library that allows us to import and use different components which formed the face of our UI application. These components included a range of features including but not limited to buttons, lines, scrolls, sliders, drop-down menus, and textboxes. It also allowed greater use of typographical features such as fonts and text wrapping. By using *Material UI*, we created a web application that provided an exceptional and satisfying digital experience to users.

An alternative we could have chosen was *Storybook*, which is an open source component tool. It provides greater functionality to *MaterialUI* with integrated testing, greater options, and the ability to review components with your team. However, it was much more complicated than *MaterialUI* with a full desktop application. No team member had also used *Storybook* before, hence to save time we used *MaterialUI*.

For our heat-map, we used *react-simple-maps*, which is much simpler to integrate than overloaded alternatives such as *Google Maps* by offering a simple static customisable map. This also allowed us to integrate a tooltip over a country when hovered with the mouse. We also used *react-chartjs-2* to create graphs as it was a much better fit for our API data compared to similar chart component libraries, such as *recharts*. *Recharts* graphs would not fit in our page layout. *EmailJS* was used to send emails to our users. It was simple to use and provided 200 free emails per month, which was more than enough for our project scope. An alternative was *node-mailer*, however it is much less used and frequently maintained than *EmailJS.* To display country flags on the map, the *Country Flags API* was used. It is both free to use and limitless in usage. It is also very simple to use taking in an iso-code, which is also used by our backend, and returning a png file. This was much simpler than the *World Flags API* which was overloaded and returned images of large sizes which would take much longer for the frontend to load. It also returned extra information for the country which was not necessary for our application.

5.3.7. Database

For our relational database management system, we used *PostgreSQL*, hosted on *AWS RDS* (discussed later). This is a system that most team members were familiar with as it is used in the core SENG stream course on database systems. This saved time learning an unfamiliar RDBMS, which was crucial for this project which had very little development time before the deadline. It is also JSON compatible, allowing for greater design freedom later on. *Postgres* also features a well documented python library for access to instances called *psycopg2*, which most team members were also familiar with. *Postgres* is also open source, meaning no financial costs were required with running database instances. It is also widely used and documented, with

frequent updates released. A notable disadvantage of *postgres* is that it can be slow and unstable for large scale businesses, which our project was not. It can also be difficult to update versions. However, since we were a short-term project there was no need to do so.

5.3.7.1. SQL vs NoSQL

We chose to use *SQL* over *NoSQL*. *SQL* as a relational model offers reduced data storage due to normalisation, maximising performance. A drawback of *SQL* however is it is very rigid once created and changes to schemas often cannot be applied without reloading data. *NoSQL* does not have this issue as it is non-relational and allows changes to be made while in use. It is however easy to make mistakes due to type and write inconsistencies which must be considered by the users. Since no member of our team had experience using distributed systems or *NoSQL*, we decided to stick with *SQL* due to time concerns, as it did not provide enough benefit to warrant the investment.

5.3.7.2. DBMS Alternatives

One of the *SQL* alternatives considered was *Amazon Aurora*. It is highly integrated with *Amazon RDS* and hence would have been much more reliable and easy to use. It also features greater performance and security than *Postgres*. However, *Aurora* is not included in the free tier of AWS, meaning costs would have been incurred. It is also a new system that no team member had used, so would have involved a time cost.

Another alternative considered was *MySQL*. It is very similar to *PostgreSQL*, so would have a relatively low learning curve, and hence time cost for the team. Just like *Postgres*, it also features extensive documentation and community support. It also has greater security features compared to *Postgres*. However, since the database instance will be hosted by *AWS RDS*, it will already be isolated from external access. *MySQL* would also be free to use as it is also open-source. Overall, since there are few additional benefits to using *MySQL* over *postgres* in the case of our project, *postgres* was used due to familiarity.

Finally, *MongoDB* was also considered. Overall, *MongoDB* has a considerable learning curve as it does not use *SQL*. It is relatively easy to set up with cloud services and is very powerful for data analysis. However, *MongoDB* is a paid service. It offers a free version that could be used for academic purposes, however its function is limited. Hence due to the learning curve and reduced functionality, *postgres* was a more attractive option for our project.

5.3.8. Cloud Services + Deployment

Our API was hosted on a cloud service. This allowed for frequent scheduled web scraping, as well as a constant, reliable offering of our API. We chose to use *Amazon Web Services*. It is the most popular cloud service, with some team members having experience using it. It has regions all over the world, including Sydney. It has hundreds of services including those for database management, API hosting, and a computing platform. It is also the recommended cloud service provider for the course. It has a free tier which offers limited use of many of its services for free.

5.3.8.1. Alternatives

One of the main competitors for *AWS* is *Microsoft Azure*. It also features a free tier with much of the same services offered. There seemed to be little difference between the limits offered by the free tiers of *Azure* and *AWS*. *Azure* however, according to their website, is mainly targeted at Windows and Microsoft software. The fact that some of our team members used MacOS paired with familiarity with *AWS*, meant *Azure* was not used.

The other main competitor is *Google Cloud*. Their free tier however had significantly lower limits than those offered by *AWS* and *Azure*. Since we would be creating an API, with significant data scraping and storage required, we decided to stick with *AWS*.

5.3.8.2. Database Service

For the database service, we hosted our *PostgreSQL* database on the *Amazon Relational Database Service (RDS)*. It is designed to be used with *postgres*, hence has significant documentation and tutorials for the project. It also allows for strict access controls defined by our team and controlled by Amazon. It offers 750 hours of database usage per month under the free tier as well as 20GB of storage. It also allows for usage of the *postgres* command line tool, *psql*. The created database schemas can be found in the `/PHASE_1/Application_Sourcecode/db/schemas.sql` file.

5.3.8.3. Computational Service

For the computing platform, we used *AWS Lambda*. Python scripts were run at set intervals with the use of cron jobs, to scrape the external data sources for the required information and store them in the *postgres* database. They were also set up to fetch from the database when requested by the API. The free tier offers 1 million requests per month, with up to 3.2 million seconds worth of compute time. This was more than enough for the purposes of our project.

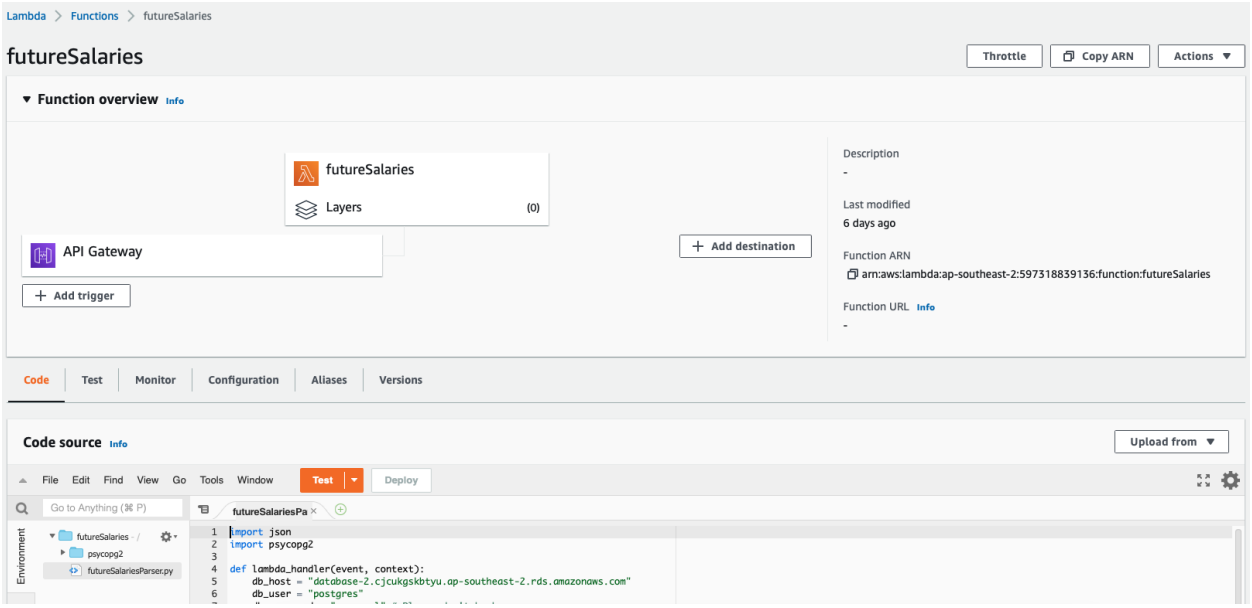


Figure 25: Example *Lambda* script deployment for futureSalaries parser

5.3.8.4. API Host

To host the API, we used *Amazon API Gateway* to host our RESTful API. This was connected to the lambda prasers which then fetched the required data from the database. The free tier offers 1 million API calls per month, which was ample for our project.

5.3.8.5. Web Application Deployment

The web application that we developed is hosted on Cloudflare Pages. This is integrated with *GitHub*, with any code pushed to ‘main’ instantly reflected on our public URL. It is compatible with *React* and has built in CI/CD which triggers on every code change. It also allows easy testing with preview URLs for every git branch and pull request. There is also a free tier limited to 500 builds per month, which was more than enough for our project.

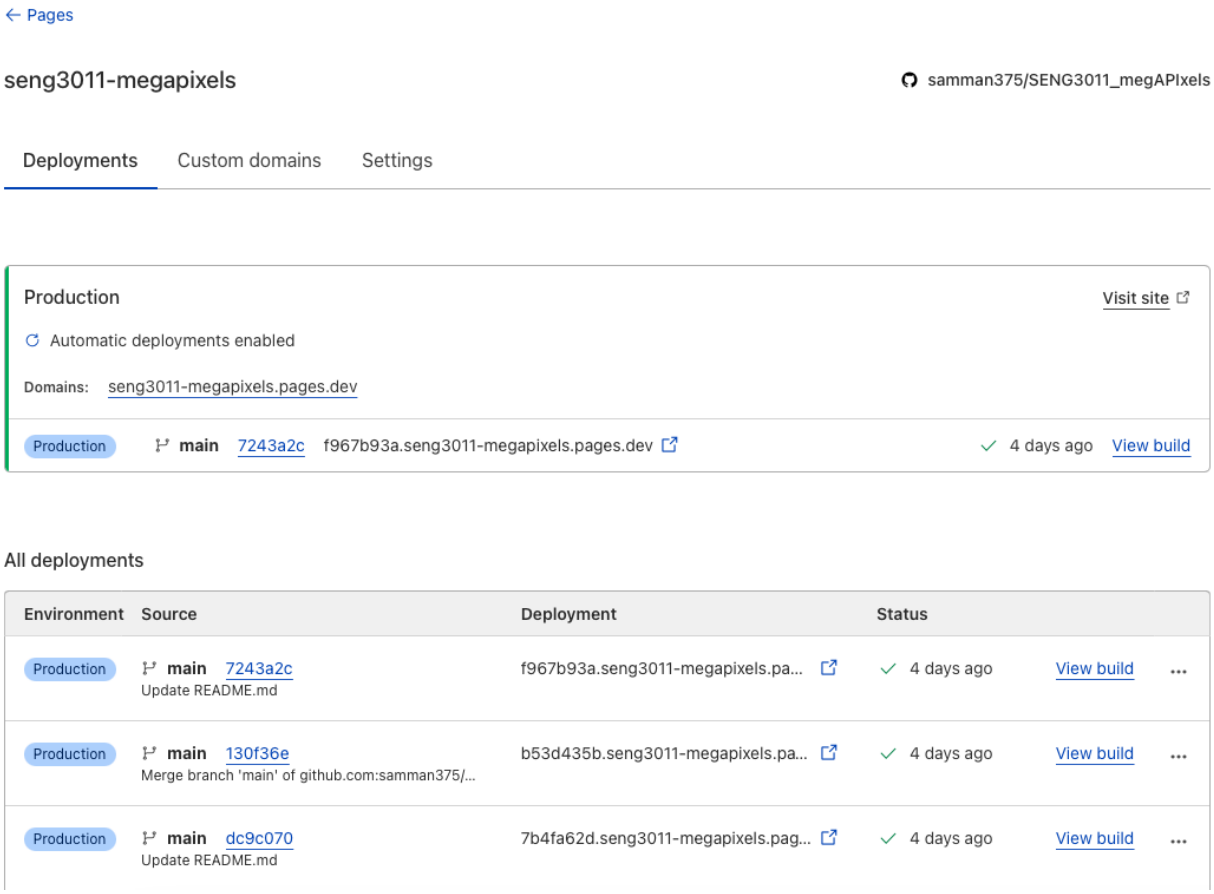


Figure: 26: Analytics tool deployment on *CloudFlare Pages*

Alternative deployment services considered were *Vercel, Github Pages*, and *AWS Amplify*. All have free tiers. *Amplify* and *GitHub Pages* were considered as they could have had better integration with the services already being used in *AWS* and *GitHub*. However, *AWS* is not used for the frontend, so would not have been any better integrated. *GitHub Pages* also did not allow for development sites. *Cloudflare* also allows for multiple users in the free tier, which *Vercel* does not. For these reasons, *Cloudflare* was used to deploy the web application.

5.3.8.6. Monitoring

We used *Amazon Cloudwatch* to monitor our API and Lambda services. This gave an auto generated, detailed, and regularly updated log which could be referred to. Since it is integrated with *AWS*, it can be activated very easily. By keeping the logs within the *AWS* environment, it also ensures they remain for internal viewing only. It also allows us to log not only the API, but also each *Lambda* script as well as the database instance. The only other alternative considered was a non-cloud based log file which would be updated every time the API or lambda functions were called. However, this is not only harder to set up, but also less detailed and more likely to fail.

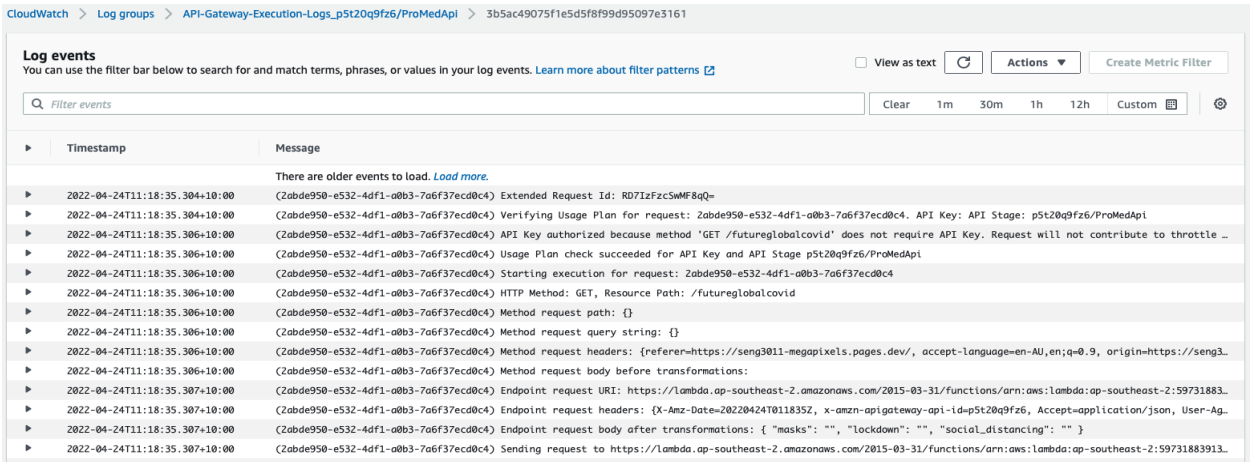


Figure 27: Example of *CloudWatch* log for our API

6. Team Management

As part of conducting our project, the means for which we managed team member tasks and communication along with team meetings have been included in this section. In addition a breakdown of team member roles throughout each deliverable has been incorporated.

6.1. Team Organisation and Work Arrangements

The team followed agile methodologies throughout the project as planned. Daily meetings occurred in which a check-up, code review and updated sprint plan to ensure the different sections of the team (front-end and back-end) were keeping up to date with each other and ensured deadlines were going to be met. Additionally, the initially created sprint schedule was refined and updated over time to accommodate updated requirements, missed deadlines and changes in member responsibilities. Communication was constant between members, and was made through the following outlets outlined within the planned management information from earlier deliverables.

6.2. Communication

This section explains the communication channels used by our team alongside examples of their use.

6.2.1. Facebook Messenger

The team's main form of communication occurred through a Facebook Messenger chat. Consistent updates, questions and important reminders were all posted to this chat for all members to see and to ensure no member was left behind. No video meetings occurred through this chat, but the planning of these meetings did. An example usage of the chat is seen in Figure 28 below.

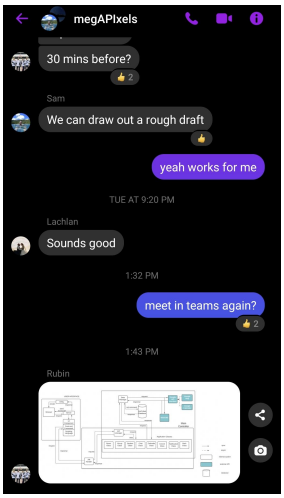


Figure 28: Example usage of team's *Messenger* chat

Alternatives to Facebook Messenger were considered, these included Discord, Slack and Teams. Discord provides a platform in which the team has a large control over the customisation, being able to create multiple text chats for different purposes and pin items to quickly be found again. It also allows people to voice/video chat in the same location. Messenger was chosen over Discord because of its ease of use, the team needed a simple chat to quickly contact each other which was better provided by Messenger.

Slack is a very similar program to Discord, allowing users to have control over most parts of their Slack. As such, for the same reasons as stated above, Messenger was chosen over Slack. Additionally, some members of the team had not used Slack before, while all members had used Messenger meaning it did not include a time cost for some members.

6.2.2. Teams Video Meetings

Whilst not initially outlined within earlier management information, Teams was used as the form of communication for video meetings such as check-ups and standups. The different sections of the team would schedule their own check-up meeting times as well as the entire team choosing to meet at a specified time multiple times a week. Additionally, with meetings being conducted on Teams, our mentor was able to see any questions that arose during the meeting outside of the scheduled mentor meeting each week.

Alternatives to Teams for video meetings were considered, namely Messenger and Discord. Messenger was not chosen as the mentor would not be able to access meetings to answer any questions, as well as the Teams chat was already created with all members before the Messenger chat and members saw no issue in continuing to use it.

Discord was not chosen for video meetings as it was deemed excessive to create a new server and channels just to talk over video. If Discord was chosen as the main communication platform, it would have most likely been chosen as the video meeting platform as well, but there was less incentive after Messenger was chosen as the communication platform. Additionally, some members would have to create and learn Discord which took away from time spent on the project.

6.3. Collaboration

To facilitate team collaboration on different tasks during our project we have included sections that explain what we used and how it has been used.

6.3.1. Google Drive

A Google Drive was used to hold collaborative documents that were required throughout the phases of development. Google Drive allows for documents to be created easily and can hold many different types of documents, such as pictures, editable diagrams and pdfs. Importantly,

Google Drive has an excellent collaborative system, allowing members to edit the same document simultaneously with no issues, comment on sections or suggest edits that can be made.

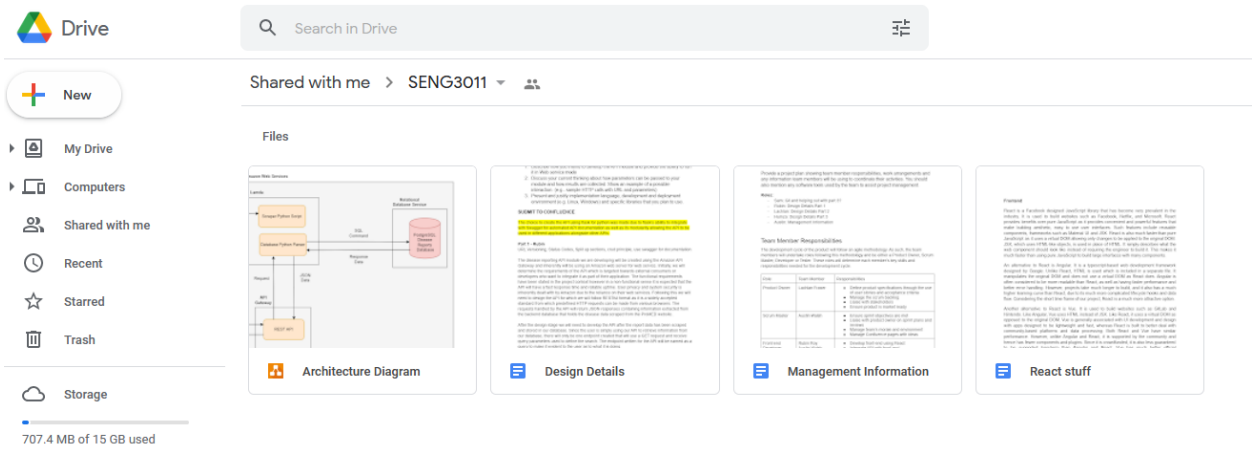


Figure 29: Team Google Drive

Confluence was another option to host documents, and while it is not used by the team to collaborate on documents, it will be used to submit final documents to the mentor. Confluence provides many similar features to that of Google Drive, but was not chosen due to the time cost as most of the members had not used Confluence before. Confluence also has a worse collaboration experience when compared to Google Drive with slower updates occurring between user's edits.

6.3.2. GitHub

To develop the product the team used a Github repository where the code from each phase of the development was worked on. Github allows for the team to easily collaborate on code during the development cycle of the product. As version control software, it means members will have no issues with working on the same code pages or losing code due to unforeseen circumstances. Github also provides settings ensuring the team will follow the agile principles. These settings include branch control, ensuring members have to pull the branch before merging, merge requests requiring review and stopping branches from being pushed to. This was used effectively in the final deployment of the product. It was deployed on Cloudflare with continuous deployment integrated with the repository, meaning any time new code was pushed to the main branch, a new version of the product was deployed to Cloudflare.

Some other version control software was considered, namely Gitlab and Azure. Gitlab is much the same as Github, providing a platform for members to collaboratively work on the development of the product. The main reason for the team's choice of Github over Gitlab was due to past experience, most of the members have used Github more recently than Gitlab, reducing the time cost associated with relearning the Gitlab layout and specifics.

Azure is a Microsoft cloud platform that provides a platform for hosting, deploying and maintaining a product. It can also integrate existing products held on other software such as Github and Jenkins. Whilst it provides many useful features for the development cycle, the time cost associated with team members learning and setting up an account would be too high to justify its usage. The service is also subscription based and developed to be used by large companies so it is somewhat overcomplicated for what the team requires from a version control software.

6.3.3. Jira

Finally, a Jira board was used by the Product Owner and Scrum Master to create and assign sprint objectives which developers completed. Jira provides the team with an easy way to see, manage and ensure members are working on their assigned sprint objectives. Specifically, this board will be updated throughout the project duration to help all team members to visualise and understand what has been completed, is being worked on, and still needs to be developed. This tool provides a wonderful visual interface to keep the team on task and provide information on project completion at a glance. Jira is also integrated with Confluence providing the team with easy access to both submit documents and view sprints within the one location.

Github and Gitlab boards were another option to manage the sprint objectives and whilst Github might have been a good option given it was located within the product repository, the team determine Jira’s additional integration with Confluence and easier management of epics would prove to be more useful for the team.

6.4. Team Member Responsibilities

The development cycle of the product will follow an agile methodology. As such, the team members will undertake roles following this methodology and be either a Product Owner, Scrum Master, Developer or Tester. These roles will determine each member's key skills and responsibilities needed for the development cycle.

6.4.1. Deliverable 1 Member Responsibilities

During deliverable 1, the team was fit into the following roles in order to complete the required tasks within the time. These roles focused on different aspects required from this deliverable, and as such, roles like developers were not required.

Table 5: Phase 1 Member Roles and Responsibilities

Role	**Team Member**	**Responsibilities**
System Designer	Sam Thorley	● Set up AWS development environment ● Connect AWS environment to database ● Choose development language and initial system architecture
Documentation Constructor	Lachlan Fraser	● Create initial documentation for API ● Generate basic endpoints and error codes ● Define required parameters and results
Report Writer	Austin Walsh Humza Saeed Rubin Roy	● Design product specifications ● Define API requirements ● Construct report detailing API development cycle

6.4.2. Deliverable 2 Member Responsibilities

For deliverable 2, the API was developed, as such, the roles changed again to reflect the new set of requirements. This phase included two areas of initial development, database integration with parsing, and development of a scraper for proMED. Additionally, testing was also required, and a couple members also undertook this testing.

Table 6: Deliverable 2 Member Roles and Responsibilities

Role	**Team Member**	**Responsibilities**
API Developer	Austin Walsh Sam Thorley	● Construct python parser scripts for handling database data ● Upload scripts to AWS and integrate with API Gateway ● Handle input requirements with different status code returns
Scraper Developer	Humza Saeed Rubin Roy	● Construct python code for parser ● Scrape data from proMED ● Parse data to fit with database schema ● Upload data to database
Development Tester	Austin Walsh Humza Saeed	● Construct testing suit to test API ● Create testing document outlining undertaken tests and further testing
Documentation Constructor	Lachlan Fraser	● Create swagger endpoints from API ● Connect swagger to API via API Gateway ● Update previous documentation to reflect new requirements

6.4.3. Deliverable 3 & 4 Team Member Responsibilities

For deliverables 3 and 4, the development of the product began. The team was thus split into two sections again, this time focusing on back-end development and the development of the front-end. Testing was required again, and in this case a back-end and front-end developer would test each of the requirements.

Table 7: Final Team Member Responsibilities of Developers

Role	**Member**	**Responsibilities**
Front-end Developer	Austin Walsh	● Choose and integrate a graph library ● Create the country pages ● Parse back-end data to display on the graphs ● Integrate prediction choices to change graph results
	Lachlan Fraser	● Create general page layout and style

		• Implement world map and slider • Code preferences page and associated elements • General creation of styled elements
	Humza Saeed	• Create help page • Large scale style fixes • Ensuring style consistency between pages • Testing and fixing front-end issues
Back-end Developer	Sam Thorley	• Research data sources to integrate with the project • Managing data parsing for the database • Find and integrate historical data for stock, exchange rates, jobs, real estate and unemployment • Filter predictions for covid data
	Rubin Roy	• Create machine learning prediction algorithm • Updating previous endpoints to work with predictions • Creating prediction based endpoints for stock, exchange rate, real estate, jobs and unemployment • Create predictions for covid data
Development Tester	Humza Saeed	• Initial testing code for endpoints • Testing front-end for errors • Reporting errors to be fixed on the front-end
	Sam Thorley	• Updating testing code for endpoints • Working with Rubin to test back-end lambda's and code • Ensuring data is parsed correctly and fixing errors with data integration

7. Reflection and Appraisal of Work

Following the completion of our project we have included a reflection on our achievements as well as our team's cohesion and involvement in each deliverable. This section also explains the challenges we faced alongside what we could have done differently.

7.1. Major Project Achievements

As we reflect on our project this term in Software Engineering Workshop 3, our team is very proud of the work we have completed overall. We have collaborated well as a team to combine our knowledge of software systems and architecture in order to develop a deployed API as well as a useful analytics and prediction software application. The process of design and development has been a valuable insight into real-world industry teamwork and we have all learned to appreciate the wide variety of skills and hard work required to develop such a project.

7.1.1. Teamwork

One major achievement we have all experienced this term is that of working in a team. The team had been constructed from a number of members with varying interests and levels of experience. We had not worked together before and were anticipating some initial friction as team dynamics and roles were established. However, we quickly found that we all shared a passion for software engineering and were able to unite in our vision for diligent work and a quality product.

This shared vision allowed us to quickly assign team roles and spread the workload efficiently. We were all able to work in an area that interested and stimulated us whilst being a part of a greater team effort. As we continued to share in the project together, we found that each person was able to put their best foot forward by contributing in a way in which they were comfortable so that the whole team could thrive.

We have progressed from strangers to colleagues to friends and, as a result, we have all learned more about what it's like to work in a team in order to excel both individually and together. These teamwork skills and achievements are sure to help bring us success in our future careers.

7.1.2. API Development

During the development of our ProMED API, we have gained a great amount of skills and knowledge regarding web scraping and API deployment. Each team member was heavily involved in this project in one way or other and we all had new skills and techniques to learn. This involved the integration of several products provided by Amazon Web Services as well as the incorporation of a number of Python code libraries to implement our scraping and parsing scripts.

Nevertheless, we were successfully able to persevere and overcome the challenges of using new technologies to develop a working API with thorough documentation. This proved to be a significant achievement and was very satisfying to watch as the many aspects of this system came to fruition. We were also very pleased to see that our efforts were rewarded with praise from our mentor and a high mark given for Deliverable 2.

7.1.3. Application Development

The culmination of the overall term project resulted in the design and development of a deployed web application. For our team, this was the Interactive Outbreak Predictor (IOP). The completion of this product is by far the greatest achievement for us as a team this term, and for many of us, in our university careers as a whole. This is the first time we have really been given the opportunity to combine skills from a number of external courses in order to create an overall application.

We began by developing a database and API, as a part of phase 1 of this project, before continuing to develop a complex frontend and improving our backend to incorporate machine learning and prediction algorithms. The combination of these skills and techniques has allowed us to develop a truly unique and accomplished application. Furthermore, we were able to incorporate feedback provided by our mentor throughout the term to continually implement new ideas and improve our product into something that we are all very proud of as a significant achievement.

7.2. Issues and Problems Encountered

Throughout the development of our API and application, we have come across a few challenges that we have had to overcome in order to produce a quality product. In particular, these included gaining familiarity with Amazon Web Services, performing successful web scraping from the ProMED website, and understanding machine learning to develop predictive software models. These shortcomings and resolutions are described in this section.

7.2.1. AWS

Firstly, we experienced some difficulty getting used to the many web services offered by Amazon. For many of us, this was the first time that we were creating an API that would be deployed on the internet for other people to use. Therefore, after taking a look at a few different deployment platforms, we settled on Amazon Web Services. This prompted us to also take a look at their other tools to help streamline our development.

However, despite the convenience offered by using services such as API Gateway, Lambda and Elastic Beanstalk, this took quite a while for us to set up and understand. Specifically, we had issues with Amazon’s permission system. As one of us was the root user and the rest of us were only given a subset of privileges, this proved to be a stumbling block throughout the

development process as we would have to constantly rely on each other to complete simple tasks.

Despite this, we eventually found ways around these issues by manually modifying and adding new privileges to other group members so that we were able to work more independently.

7.2.2. Web Scraping

An additional significant challenge faced in the completion of the API was the web scraping process. This was required to retrieve article data from the ProMED website in a way that could be easily formatted and returned by our API. Our team only had limited web scraping knowledge and past experience had been based on simple websites. Therefore, we found it quite challenging to scrape data from the ProMED website which is complex and designed largely as a single-page application.

This prompted us to explore some other options for retrieving website data including Selenium, BeautifulSoup and LXML. However, we found that these tools were unable to return data in the formats we were after. Therefore, we persisted with the use of the Python library scrapy to scrape ProMED in more of a traditional manner. In addition to this, ProMED's access limitations prevent fast scraping by returning a 403 and denying access to the website. We tried to utilise free proxies to bypass this, although it was unsuccessful. Hence, a single thread approach with request auto-throttling was used to minimise the number of 500 responses. Consequently, this significantly lengthened scraping time at around 20 pages a minute.

Due to the early difficulty that was encountered in this area, we were unable to complete all of the necessary web scraping for deliverable 2. Instead, only about one week's worth of articles was added to our database. Furthermore, we did not have time to find a way to extract valid reports from content due to the initial time it took to develop the scraper. However, we were able to extract the diseases, syndromes, event dates and locations from the raw text which we placed in a single report object linked to an article in the database.

7.2.3. Machine Learning

Another significant issue arose with the implementation of machine learning in our predictive software. This proved to be an area that required a great deal of research and was something that we spent a lot of time figuring out and debugging. Machine learning can be a challenging area of software engineering due to its complexity and it can be difficult to produce correct and meaningful predictions.

For our project, in order to simplify this task as much as possible, we decided to employ an Autoregression algorithm to train the prediction model. In this manner, historical data is split into thirds such that the first two thirds is used for training and the last third is used for testing. This allows us to have a greater level of control over our predictions. However, this model is also significantly limited by the length of the test data so further research may be necessary in order to gain true confidence in our machine learning capabilities.

7.2.4. Finding External Sources

As hinted at in the external APIs section of the system design and implementation, finding data for the required endpoints proved more difficult than initially predicted. As the data required was very specific, requiring historical data for all countries, there were very few data sources that could be found to satisfy this. As such, many countries have a couple graphs that do not display anything as no data was found. This was mainly seen in real estate data as only valid data for Australia was found, and all other countries displayed a no-data graph on the front-end.

Another problem with the data was that some required the downloading of csv files to be scraped into the database. This can prove a problem when trying to keep the program up to date with those data sources, as it means new data has to be manually downloaded and then scraped instead of scraped directly from the website.

7.2.5. Frontend Graph Libraries and UI Problems

The front-end team faced a couple issues when creating the product which resulted in varying levels of time lost resolving these. Initially, the team had decided to use *recharts* as the main library to display graphs, and whilst data was able to be displayed via it, the team was unable to format the graph to fit the existing page styles. Thus, the team had to choose a different library and rewrite the code to display graphs.

Additionally, there were also some small UI problems that took a sizable amount of time to resolve. There was an issue where no white space was shown between a title text and it required adding the html entity for whitespace to resolve the issue. Another problem faced was that some of the flags for countries display would not show, which was resolved by manually changing the country name to fit with the library for country flags.

7.3. Skills We Wish We Had

Going into the project for Software Engineering Workshop 3, each team member had a set of skills that they had learned from other university courses as well as their own research. However, not everybody on the team shared the same skill set. Therefore, we were able to successfully assign roles among the team so that each person could work in an area they were familiar with. That being said, there was still a lot of individual research that was necessary in order to complete this project and it would have been beneficial to us to have some guidance and teaching in these areas.

Specifically, none of us had ever deployed any software before. However, this project required deployment in a few different instances. Firstly, we needed to deploy our backend API, which we chose to do on AWS API Gateway. Next, our Swagger documentation also needed to be deployed which we again did through AWS using their Elastic Beanstalk framework. Finally, our frontend application was also deployed on CloudFlare Pages which we determined to be a better solution for that use case. In order to complete these deployments, a lot of external

research was needed and this took a significant amount of time and effort that could have been put towards other aspects of the project. For this reason, it would have been very useful to us as a team if we were taught a simple method of deployment prior to attempting this blindly on our own. In addition to this, skills such as GitHub continuous integration and deployment would also have been helpful to go through in class.

7.4. What We Would Have Done Differently

Overall, we are very proud of what we have accomplished this term in Software Engineering Workshop 3. We have successfully designed and developed a web API and complex application. However, as we draw to the conclusion of the project as a whole, we may reflect on the journey we have embarked on this term. We begin to ponder how we might change our processes and methods in order to produce an even better product in future.

We have been happy with most of the tools and technology employed this term. However, some aspects probably created more problems than they were worth and could have been replaced with another option. For example, in phase 1, we chose to deploy our Swagger documentation via AWS Elastic Beanstalk. This took a lot of time to set up and was quite fiddly to navigate. Therefore, for phase 2, we searched for a better alternative and decided to use CloudFlare Pages for our frontend application deployment. This proved to be significantly easier to use and a lot of time could have been saved earlier on in the project if we had decided to use this system initially for phase 1.

Furthermore, we felt as though we were under a great deal of time pressure throughout the completion of this project. This may have been improved if we were to be more intentional about our time management from the beginning. In the first few weeks of the term, we had relatively little to do as we were figuring out our design and implementation stack. However, if we had completed these tasks more efficiently, we could have moved on to work on future deliverables straight away. We already had access to the specifications for these future tasks and we would have saved ourselves from some time stress if we had already started this work earlier. Additionally, after a deliverable was due, we would often rest for a few days before meeting again to discuss the next steps of the project. In hindsight, although these periods did not seem significant at the time, a lot of work could have been occurring then and would have meant that we didn’t feel as rushed toward the end of term. If we did have more time to work on our project, we would, without a doubt, have designed and developed a better product overall. Therefore, this could be another area to grow on and change for future projects.

8. Conclusion

In conclusion, IOP serves as a useful tool for businesses and individuals in investigating future scenarios to aid in planning and decision making. Our application's interactive features account for all types of users due to its simple interactivity and graphical presentation. Our overall experience developing the application was satisfying as we were able to overcome challenges we had during earlier deliverables such as the limitations of scraping ProMED, as well as successfully deploying our API on AWS. On the other hand, IOP could be improved by filling in data gaps and refining the prediction model for restriction filters to generate predictions with greater reliability. Despite this, our final implementation of IOP includes all our major requirement features, making it an ideal analytics platform for a wide range of users to employ in their day to day planning by providing a variety of foresight in a user friendly fashion.