Visualisation of Pandemics Simulation

Final Year Project Report

Matthew Lakin
F222199

Computer Science and Mathematics
24COC255

Project Supervisor: Hossein Nevisi
October 2024 - May 2025

Department of Computer Science
Loughborough University

Contents

Chapter 1

Abstract

"We've actually invested very little in a system to stop an epidemic. We're not ready for the next epidemic."

- Bill Gates, 2015 [5]

With the recent pandemic of COVID-19, the world has become more aware of the need for data visualisation. The ability to see the spread of a virus and how it affects different countries is important for understanding the transmission of the virus and how to contain it.
This project aims to create a tool which can visualise epidemic and pandemic data in a way that is easily understood. The tool has been designed to show the number of cases and deaths per week for each country. The tool also demonstrates a progressive timeline of the pandemic and its movements, and give the user the ability to compare and contrast how different countries are affected.
The application considered in this report is a web-based tool which uses D3.js to simplify and render given transmission data. The data is in a JSON format and is sourced from the European Centre for Disease Prevention and Control (ECDC). For this dissertation, I have created an encoding script which imports data of any format and outputs the data in a standardised form. This change enables analysis of a broader range of data. By creating this script, the application has a broader spectrum for analysis. The application is able to be used analyse the data and compare how different countries are affected over a time period.

Chapter 2

Introduction

The specific problem to be addressed in this project is to support and simplify the communication of big datasets for analysis and assessment. A key feature of the application is ability to visualise the transmission of a pandemic.

Data visualisation is just one step of the solutions of containing and analysing pandemics movements. A limiting factor of any application is the quality and accuracy of the data pooled. In this case, the effect is compounded by only using one source. Other factors would factors would need considered such as geo-political transparency. As an example, in the early days of the COVID-19 pandemic, China was not forthcoming with their data [9]. This made it difficult for other countries to prepare for the virus.

2.1 Aims and Objectives

The outcome required of this project is to create a tool which can visualise epidemic data in a way that is accessible to all. The interactive tool will enable the user to show the number of cases and deaths per week for multiple countries visually in a time-series. A feature of the application has the ability to progressively illustrate a timeline of the epidemic giving a worldwide transmission overview.

To achieve this aim, I will be using a node.js backend to serve the data to the front end. The front end will be a web application which will use D3.js to render the data. The data will be in a JSON format and will be sourced from the European Centre for Disease Prevention and Control (ECDC) [4]. The data will be in a similar format to the COVID-19 data, but will be more generic so it can be used for any epidemic data.

My design methodology utilises the Waterfall model to support a structured workflow for my project. This will ensure a logically sequenced approach to coding and report architecture which allow focus in each part of the project.

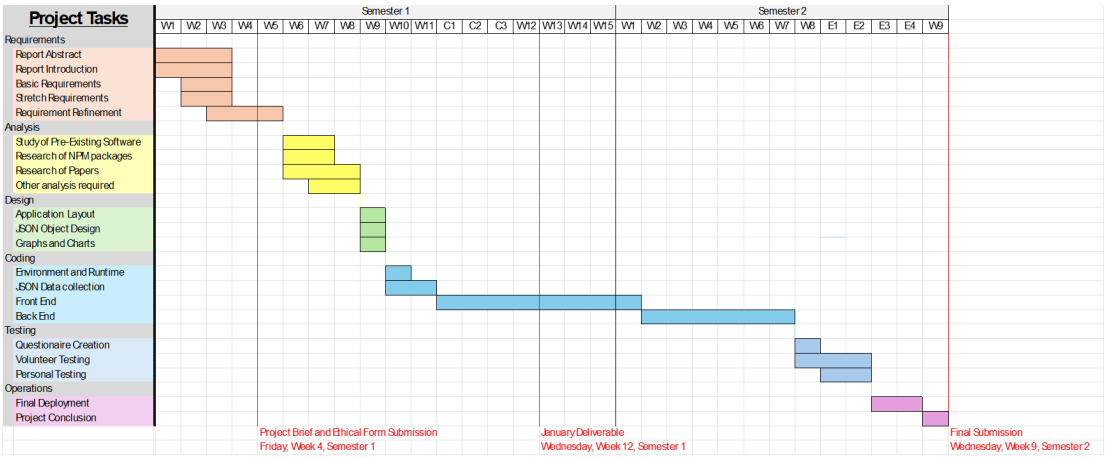

Figure 2.1: Gantt Chart

Along with using the Waterfall model, I created a Gantt chart to roughly plan my time for the project. This approach ensures the project is on track against a critical path and establish that I am not spending too much time on one section of the project. I will go into further detail about the Gantt chart in the Methodology section of this report.

Chapter 3

Problem Domain

For this project, I have elected to use the MoSCoW approach to define the requirements of the project. This specifies the requirements into layers of priority and sets the scope of the project.

- **Must Have (Minimum Viable Product)**
  - The dashboard must show a map of the Earth.
  - The dashboard must visualise the COVID-19 data and show the number of global cases and deaths per week.
  - The map will render polygon and country data from a GeoJSON file.
  - The individual countries must be clickable.
    - Clicking a country will update a panel on the right.
    - The panel will have a time-series graph showing the number of cases and deaths in that country over the time period.
    - The panel will have another time-series for use in comparison. The user will be able to choose a factor to compare against, such as population, GDP or another country's statistics.
    - Highlight the secondary country if applicable.
  - The dashboard must have a timeline slider of the pandemic.
    - The user must be able to scrub through the timeline.
    - There will be a "playback" feature which will automatically scrub through the timeline.
    - The timeline will show the current week in ISO 8601 format.
  - Search Bar
    - The user must be able to search for a country by name.
    - The search bar will have a dropdown list of countries.
    - The dropdown list will be filtered by the user's input.
    - If an unknown country is entered, the user will be prompted to enter a valid country name.
    - If a valid country is entered, it will be the same as clicking on the country and the text field will be wiped.
  - Reactive and responsive design.
    - Use of smooth transitions and animations to make the dashboard more visually appealing.
    - All graphs and functionalities should be connected to the timeline slider and country selected.

- **Should Have**
  - Zooming in on the map.
    - Adding a button to toggle a "zoom" feature on the map.
    - When clicked, the map will zoom into the selected country.
    - When clicked again, the map will zoom back out to the world view.
  - Hotspots on the map.
    - This would require analysis of the data to find out where the rate of infection is highest.
- **Could Have**
  - Analysis of data with other data sources to search for correlations.
    - I am currently unsure what data sources I would use for this, but it would be interesting to see if there are any correlations between the spread of the virus and other factors.
    - Comparing the number of cases or deaths to different countries.
    - Normalising the data to a GDP per capita basis.
  - Extensibility to other datasets.
    - The code should be generic enough to be used for any pandemic data.
    - Want to use a generic data format and a script which is able to cast the data into the correct format.
  - Frame updating to resizing the window.
  - Removal of spurious and erroneous data at runtime.
- **Would Have**
  - Predictive modelling and forecasting.
    - This is my most ambitious stretch goal. I would like to be able to predict the spread of the virus using the data I have. This would require a lot of work and research into the field of predictive modelling. This would likely be a separate project in itself.

Chapter 4

Methodology

The project incorporates a Waterfall model into my project since it will ensure my code and report as a whole is developed in a structured and ordered fashion.
The steps of Waterfall I will be following are:

1. **Requirements**
    - This section requires producing and refining a set of requirements. I plan on having 2 groups of requirements: Minimum Viable Product requirements and Stretch requirements.
    - I plan on refining all the requirements in this stage so I can stay on this path plan.
2. **Analysis**
    - In this section, I will be doing research into the problems which are discovered by my requirement refinement. I will investigate papers and other relevant published sources.
    - I also want to investigate node packages to represent my data graphically. (This would continue into the design section)
    - This section will likely take up a large section of my report, and I may return to this section midway through the project.
3. **Design**
    - I would be designing the layout and colour scheme of my application as it would appear in a browser window. It is worth noting that different size screens would render it differently, so I need to take that into account when planning.
    - I am not using a database for this project, but I will be using JSON and creating a standard design for objects in JSON will be beneficial.
    - I need to decide what values I use for graphs and what kind of graphs to use.
4. **Coding**
    - I will be making a web application with node.js. This will require me to create a valid work environment and get the node runtime working on my local machine. I will likely use Docker for this since it is a very easy to work with tool perfect for this application.
    - I will mostly be using JavaScript for this project because of previous experience and native compatibility with web development. There will also be HTML and CSS for loading the d3 canvas and aligning elements.
    - Node.js also comes with node package manager which will also be useful for loading the custom packages I plan on using in this project.

5. Testing

- For testing, I will be using a variety of normal, boundary and erroneous tests to check my software for crashes, unexpected results, and vulnerabilities.
- I also plan on having 3rd-party applicants to test my application. They would have a list of things to do in the application so I can analyse the ease of use.
- I would also ask them to fill in a questionnaire.
- I will use the feedback from the testers to improve my application.

6. Operations

- I want to deploy my application to a docker container and eventually a server.
- Node.js applications are commonly deployed onto servers and having a production repository would be a good standard for myself. Pushing to a server from a local git repository is a highly transferable skill which is widely used.

Chapter 5

Literature Review

5.1 Existing Solutions

5.1.1 COVID-19 Dashboard

A common tool for data visualisation is a software called PowerBI, a Microsoft product which allows for the creation of dashboards. The dashboard in the figure shows a real-time representation of the COVID-19 pandemic.

Figure 5.1: COVID-19 Dashboard Example [1]

This dashboard has a broad spectrum of information even at first glance. The map uses bubbles to denote hotspots of activity for the virus. The bubbles are sized by the number of cases in that country. The bubbles can also be clicked for more detailed information on that country.
The user is also able to alter the time frame of the data, and the data is updated in real time.
There is a lot here which I would like to incorporate into my own project. The map is a good way to visualise the data and the bubbles are a good way to show the number of cases in each country. I also like the big bold text which summarises the data which is easy to read.

5.1.2 Plague Inc.

Plague Inc. is a game published in 2012 by Ndemic Creations. It gives the user the ability to create and spread a virus across the world. The game was initially released on mobile devices and has since been released on PC and console. [3]
Mobiles games require accessible and easy to use interfaces, which is why Plague Inc. is a good example of a visual pandemic simulation. The game is simple to use and has a lot of information on the screen at once.

Figure 5.2: Plague Inc. Dashboard Example

Despite many systems being dramatised for the sake of gameplay, the game offers many features which could be useful for a real-world pandemic simulation.
The game uses a world map which gradually changes colour as the virus spreads, this is from many red dots appearing on the map. The user also has access to individual breakdowns of each country and can see the number of cases and deaths in each country.
The game also has a transmission system which changes from country to country.

There is certain information provided in these examples that I will not be able to provide in my project. For example, the COVID-19 dashboard has real-time data, which I will not be able to provide. The Plague Inc. game has a transmission system which shows the spread country to country, which I will not be able to provide.

5.2 Data Sources

The data for this project will be sourced from the European Centre for Disease Prevention and Control (ECDC) [4]. The data is historic and only has data from the 1st week of 2020 to the 23rd week of 2022. The data is in a JSON format and includes the number of cases and deaths per week for each country. The format of the data is as follows:

```
{
    "country": "Afghanistan",
    "country_code": "AFG",
    "continent": "Asia",
    "population": 38928341,
    "indicator": "cases",
    "weekly_count": 1368,
    "year_week": "2020-47",
    "rate_14_day": 6.5043,
    "cumulative_count": 44771,
    "source": "Epidemic intelligence
        national data"
}
```

Listing 5.1: Cases JSON Example

```
{
    "country": "Afghanistan",
    "country_code": "AFG",
    "continent": "Asia",
    "population": 38928341,
    "indicator": "deaths",
    "weekly_count": 69,
    "year_week": "2020-47",
    "rate_14_day": 3.3395,
    "cumulative_count": 1695,
    "source": "Epidemic intelligence
        national data"
}
```

Listing 5.2: Deaths JSON Example

This is the format that the ECDC provide, but I will be amending it slightly to make it easier to work with. I will be combining the cases and deaths into one object and adding a date field. This will make it easier to work with the data in the front end.

It also needs to correspond with the GeoJSON data I will be using to render the map.

It is worth noting that I don't want this code to be limited to just COVID-19 data. I want to be able to use this code for any pandemic data. This is why I am making the data more generic and easier to work with.

5.3 Compartmental Models

5.3.1 SIR Model

Compartmental models are a type of mathematical model used to represent the different populations in a system. The most common compartmental system in epidemiology is the SIR model [2]. The SIR model is a simple model which divides the population into three compartments: Susceptible, Infected and Recovered. The model is represented by the following differential equations:

$$\frac{dS}{dt} = -\alpha SI \tag{5.1}$$

$$\frac{dI}{dt} = \alpha SI - \beta I \tag{5.2}$$

$$\frac{dR}{dt} = \beta I \tag{5.3}$$

Where:

- S is the number of susceptible individuals.
- I is the number of infected individuals.
- R is the number of recovered individuals.
- α is the rate of infection.
- β is the rate of recovery.

To explain this model, it's important to understand proportional reasoning. In the first equation, 5.1, the rate of change of the susceptible population is proportional to the complement of the product of the susceptible and infected populations.
A way to think of why that is true is because the more infected people there are, the more likely it is for a susceptible person to become infected.
This model is a good starting point for understanding the spread of a virus, but it has some limitations. In this model, once you have recovered, you are no longer able to be infected. Another issue is that the model assumes the disease is non-fatal.

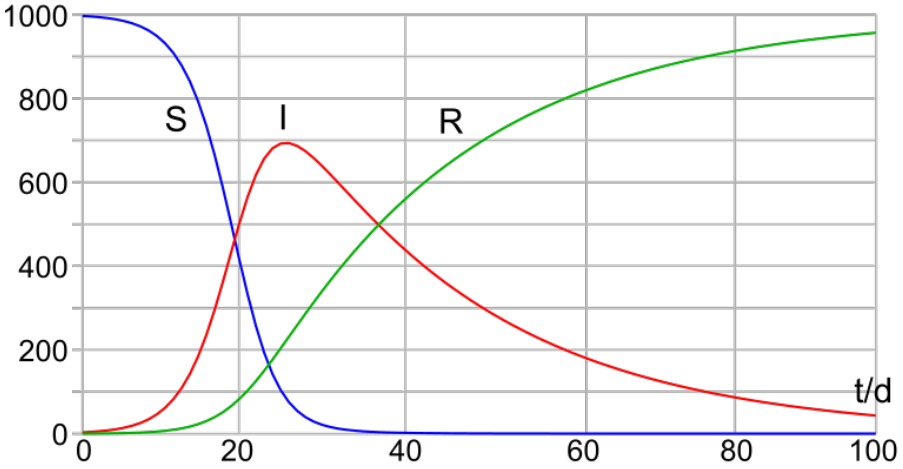

Figure 5.3: Diagram of SIR Model with start values S=997, I=3, R=0, α=0.0004, β=0.04. [6]

5.3.2 Solution to the SIR Model

The SIR model can be solved using numerical methods. The most common method is the Euler method. The Euler method is a simple method for solving ordinary differential equations. It is not the most accurate method, but it is easy to implement. The Euler method is represented by the following equations:

$$S_{n+1} = S_n - \alpha S_n I_n \Delta t \tag{5.4}$$
$$I_{n+1} = I_n + \alpha S_n I_n \Delta t - \beta I_n \Delta t \tag{5.5}$$
$$R_{n+1} = R_n + \beta I_n \Delta t \tag{5.6}$$

Where:

- S_n is the number of susceptible individuals at time n.
- I_n is the number of infected individuals at time n.
- R_n is the number of recovered individuals at time n.
- Δt is the time step.

If we look at the phase space of the SIR model, we can see that the model has a fixed point at the origin. This means that the model will always return to the origin. This is not the case in real life, as the disease will eventually die out. This is because the model does not take into account the finite population size.

5.3.3 Improvements to the SIR Model

The SIR model can be improved by adding more compartments to the model. However, this makes the model more complex and harder to solve.

SEIR Model

This model adds an exposed compartment to the SIR model. For individuals who have been infected but are not yet infectious. The model is represented by the following differential equations:

$$\frac{dS}{dt} = \mu N - \mu S - \frac{\beta I S}{N} \tag{5.7}$$
$$\frac{dE}{dt} = \frac{\beta I S}{N} - (\mu + a)E \tag{5.8}$$
$$\frac{dI}{dt} = aE - (\mu + \gamma)I \tag{5.9}$$
$$\frac{dR}{dt} = \gamma I - \mu R \tag{5.10}$$

The SEIR model is better for simulations than the SIR model since it takes into account the incubation period of the disease. This is important as it is possible for a person to be infected but not yet infectious.

It is beyond the scope of this project to go into detail about the SEIR model, but it is worth noting that the model is more complex and requires more data to solve. The model is also more accurate than the SIR model.

5.4 Statistical Analysis

Statistical analysis is a key part of this project. I will be using statistical analysis to find correlations between the data and other factors. I will be comparing the cumulative cases and deaths of two countries to see if there are any correlations between the two.

I also want to make a statistic comparing the number of cases to the gdp per capita of the country to see if there is an inverse correlation between the two. This would be interesting to see richer countries were able to contain the virus better than poorer countries.

$$\text{Infections} \propto \frac{1}{\text{GDP Per Capita}} \tag{5.11}$$

If this is the case, I would like to calculate the constant of proportionality. This would be a good indicator of how much money a country needs to spend to contain the virus.

I would also like to compare number of cases to other statistics I have available to me in my GeoJSON file. This includes:

- Population
- Sovereign status
- Type of economy
- etc

Chapter 6

Design Review

6.1 Wire-framing

I need to work within the constraints of node.js and D3.js for my designs. Below is a wire-frame of the dashboard I plan on creating.

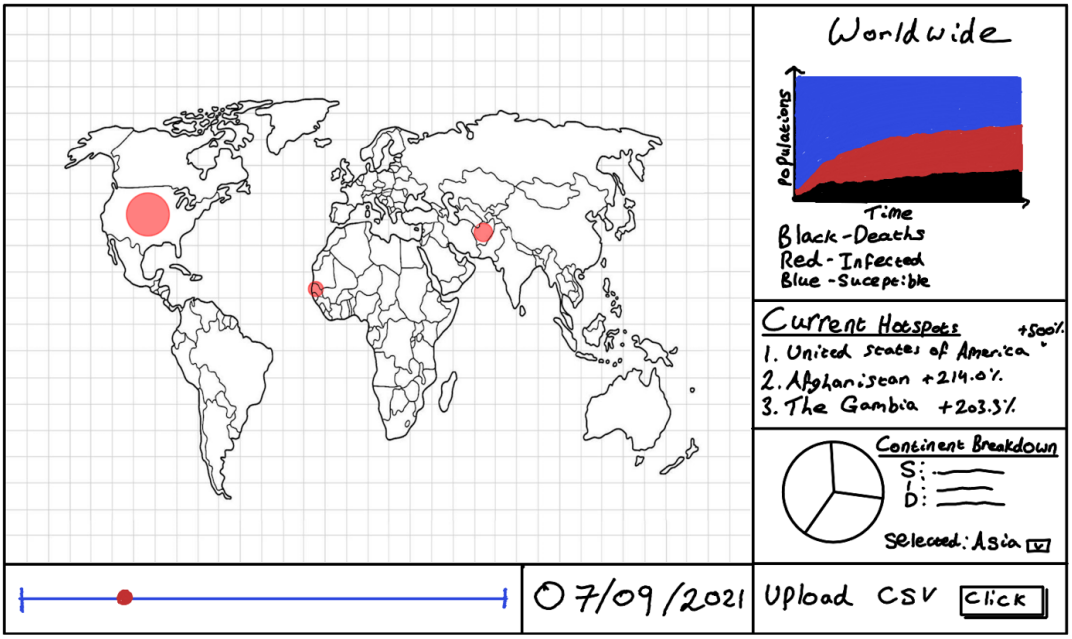

Figure 6.1: Wireframe of Dashboard

Specific features of the dashboard include:

- A map of the Earth.
- An interactive timeline of the pandemic.
- A area chart showing the number of cases, deaths and population susceptible.
- Listed hotspots and the percentage change within the countries.
- Hotspots on the map.

- A pie chart for susceptible, infected and dead populations within continents.
- A button to load different data files.
- The current date in the data.

When clicking on a country, the canvas will zoom onto the country, giving more specific information on how the pandemic is affecting that country. A button will also be available to return to the world map.

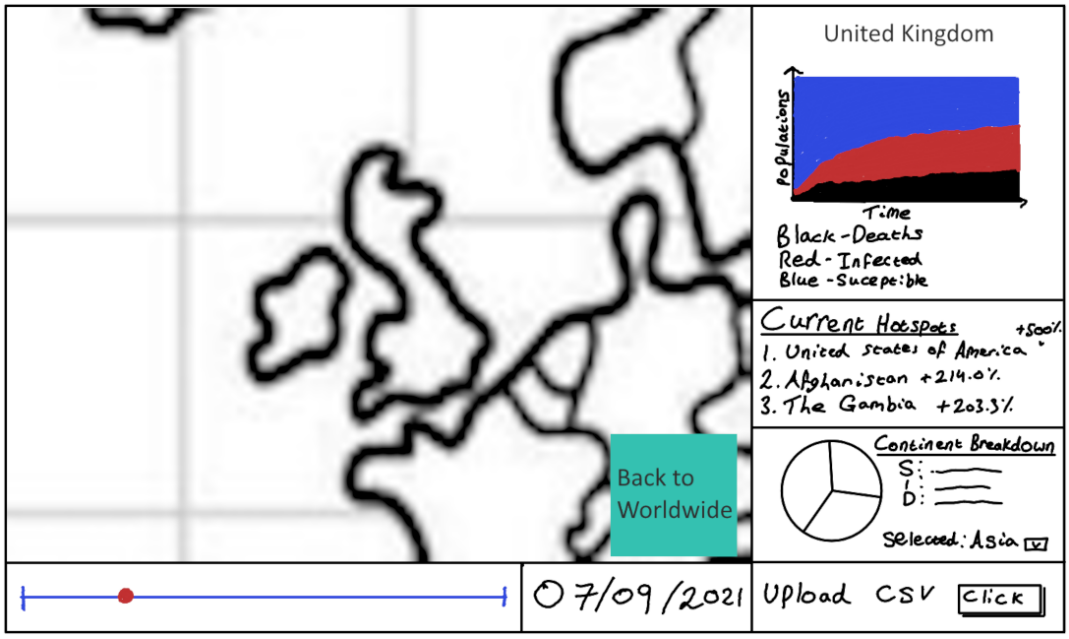

Figure 6.2: Wireframe of Country Page

A perk of using SVG to render the map is that when zooming, no resolution is lost. This is because SVG is a vector format, meaning that the image is made up of lines and shapes rather than pixels. This means that the image can be scaled up and down without losing quality.

6.2 D3.js

D3.js is a JavaScript library for producing dynamic, interactive data visualisations in web browsers. It makes use of the widely supported SVG format to render graphics. D3.js is a powerful tool for creating visualisations and is widely used in industry.

D3.js is a good choice for this project as it natively supports GeoJSON data which is the format I will be using for the map. The polygon data is open source. [8]

This is exclusively used on a web browser, so I will be using node.js to serve the data to the front end.

D3.js also has many built-in functions for creating different types of graphs and charts and interacting with them. As an example, D3.js has a built-in function for calculating the bounding box of a country polygon. I will be using this function to calculate where to zoom in on the map. Here's a JavaScript snippet of how I will be using this function:

```
// Creates a 2d array of the bounding box of the country
const bounds = path.bounds(d);
const dx = bounds[1][0] - bounds[0][0];
const dy = bounds[1][1] - bounds[0][1];
const x = (bounds[0][0] + bounds[1][0]) / 2;
const y = (bounds[0][1] + bounds[1][1]) / 2;

// Calculate the scale of the zoom
const scale = Math.max(1, Math.min(8, 0.9 / Math.max(dx / width, dy / height)));
const translate = [width / 2 - scale * x, height / 2 - scale * y];

// Smoothly transition to the new zoom level
svg.transition()
    .duration(750)
    .call(
        zoom.transform,
        d3.zoomIdentity.translate(translate[0], translate[1]).scale(scale)
    );
```

The GeoJSON data has much more information than the COVID-19 data. It includes geometry of the country, information regarding the countries economy, sovereign status, GDP, all in multiple languages. I will be cleaning the data to only include relevant information for this project: country name in English, country code (to link the data with the Covid-19 JSON file) and geometry information.

6.3 Chloropleth Maps

For this project, I referred to a book on data visualisation by Andy Kirk [7]. The book is a good introduction to maps with data visualisation as well as the use of colour in maps making it a good resource.

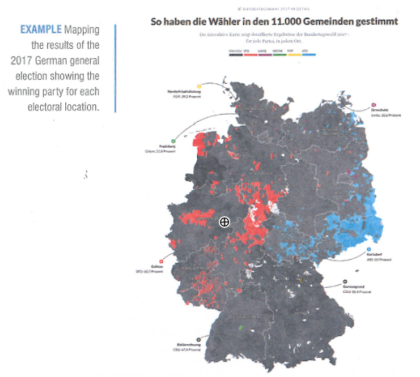

Figure 6.44 How Voters in the 11,000 Municipalities Voted, [Translated] by *Berliner Morgenpost*

Figure 6.3: Example Chloropleth Map from Andy Kirk's Book [7]

Choropleth maps are a type of map which uses colour to represent data. The map is divided into regions and each region is coloured based on the data. This is a good way to visualise data as it allows for easy comparison between regions.

The figure on the left is colouring the regions based on different voter populations in Germany. This differs to how I will be using chloropleth. My legend will be a logarithmic scale from white to red to signify the number of cases in that region. The darker the red, the more cases there are in that region.

The reason I am choosing a logarithmic scale is due to the disparity of the numbers, especially in the early days of the pandemic. The logarithmic scale allows for a better visualisation of the data as it allows for a wider range of values to be represented. It is commonly the used scale for pandemic data.

Chapter 7

Technical Solution

7.1 Node.js

Before I start working on any code, I need to set up my development environment. As I discussed in my design section, I will be using node.js to serve data in front end and D3.js to render the data.

This is my first time using node.js and web development in general, so I spent a lot of time researching how to set up a node.js server. The first step I took was to install node.js on my local machine. I intend on deploying the application to a server hosted by the university using a Docker container, but I will be developing the application on my local machine.

I used the express.js framework to create a simple server with a router to serve the data. I have a diagram of the structure of my router schema below.

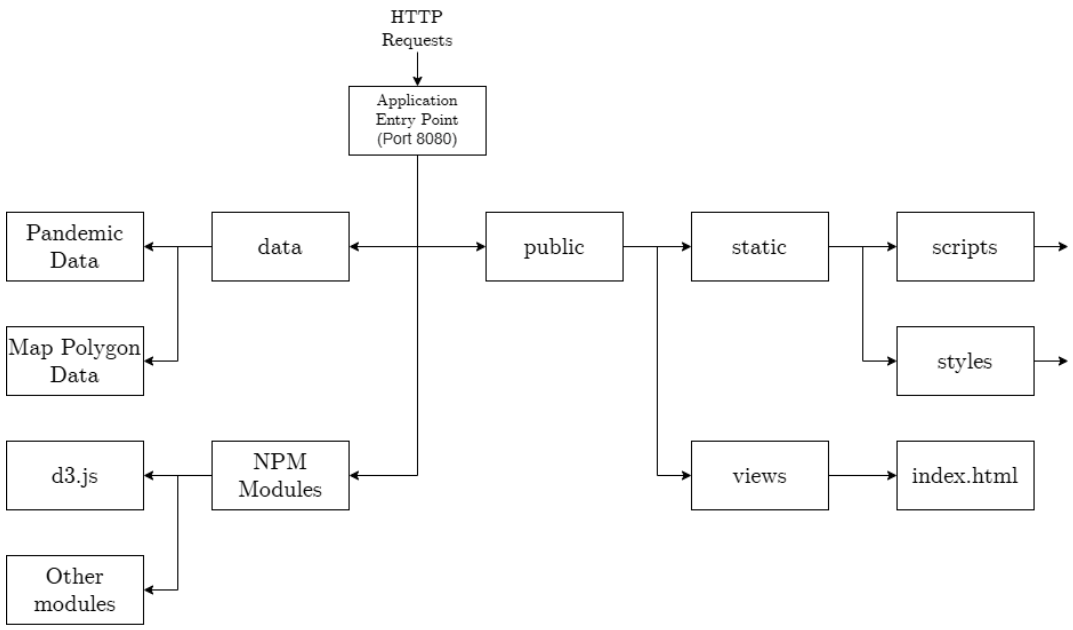

Figure 7.1: Data Routing Structure

7.2 Asynchronous Data Fetching

When dealing with data in JavaScript, and web programming as a whole, you must deal with asynchronous data fetching. This is because the data is not available immediately and the program must wait for the data to be fetched before it can be used. In my project, I used the fetch API to fetch the data. The word "server" in this context is a bit misleading, as the data is not being served from a server, but rather a local file. In node.js, the lines between server and client are blurred, as the server is also the client.

The fetch API is a promise-based API which returns a promise object. This means that the data is not available immediately and the program must wait for the data to be fetched. If the promise is resolved, it will contain the okay response of "200", and the data requested. If the promise is rejected, for example if the file is not found, it will contain the error message with an error code.

Because of the chance of the promise being rejected, I used error validation and a catch statement to handle the error. This is a good practice when dealing with asynchronous data fetching, as it allows for better error handling and debugging. Here is an example of how I used the fetch API to fetch the data:

```
1   fetch('https://api.example.com/data')
2   .then(response => {
3       if (!response.ok) throw new
4               Error('Network response was not ok'); // #404
5       return response.json();
6   })
7   .then(data => {
8       console.log('Data received:', data); // #200
9       // Process the data here
10  })
11  .catch(error => {
12      console.error('Fetch error:', error);
13  });
```

On the next page is a sequence diagram of the data fetching process. The diagram shows the flow of data from the server to the client and how the data is fetched asynchronously.

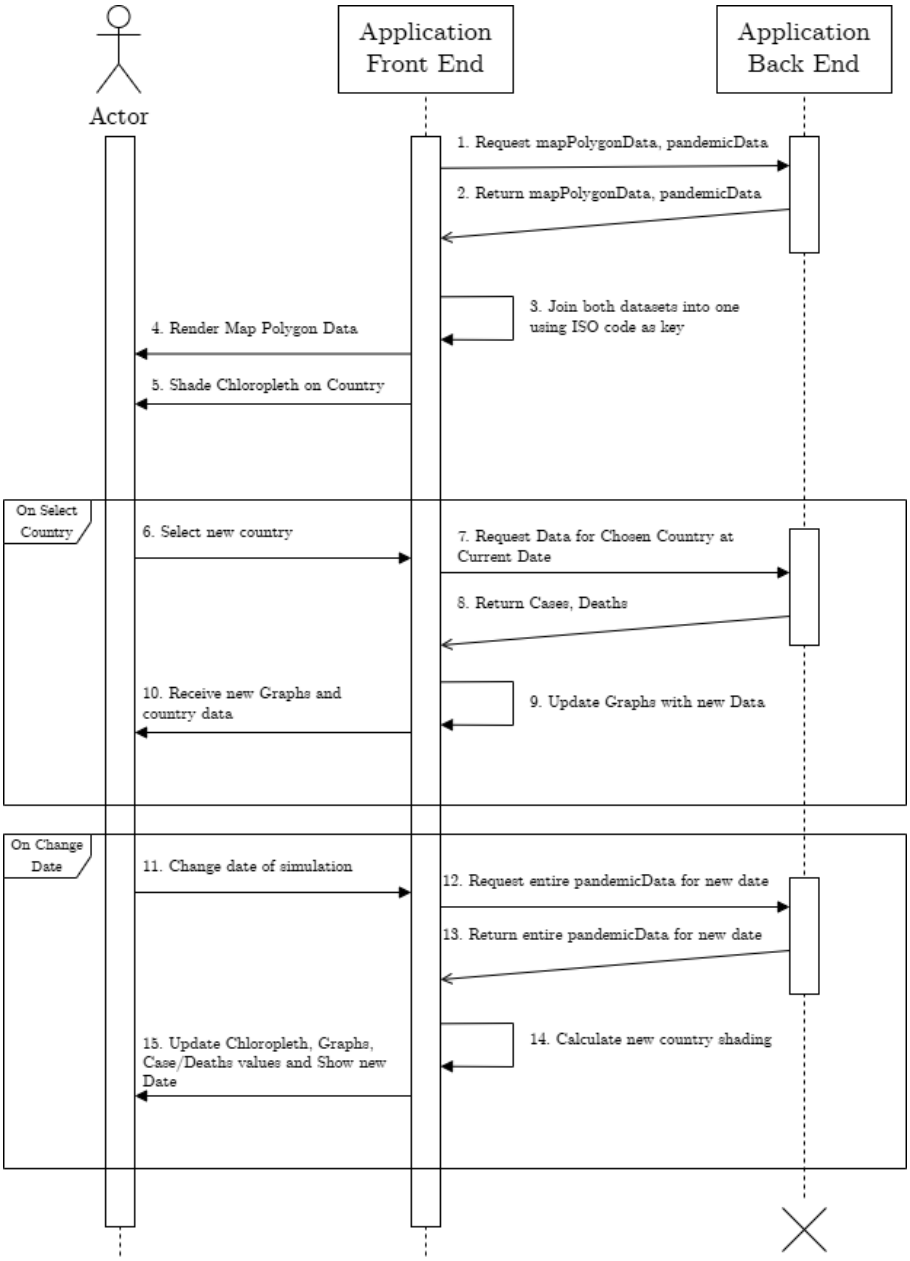

Figure 7.2: UML Sequence Diagram for Asynchronous Data Fetching

7.3 Cleaning the Data

Before I start working on the front end, I need to clean the data. The data from the ECDC is in a JSON format, but it is not in a format that is easy to work with. I need to combine the cases and deaths into one object and add a date field. This will make it easier to work with the data in the front end.

I will be using Python to clean the data, since doing this is a one-time task and Python is a good language for data manipulation. I will be using the json library to read the JSON file and the defaultdict package to manipulate the data into a standardised form. I will then write the cleaned data to a new JSON file.

I restructured the data to look like this:

```
{
    "AFG": {
        "properties": {
            "country": "Afghanistan",
            "country_code": "AFG",
            "continent": "Asia",
            "population": 38928341,
            "source": "Epidemic intelligence national data"
        },
        "data": {
            "2020-01": {
                "cases": 0,
                "deaths": 0,
                "cumulative_cases": 0,
                "cumulative_deaths": 0,
                "rate_14_day_cases": 0.0,
                "rate_14_day_deaths": 0.0
            },
            [...]
        }
    },
    [...]
}
```

The python code I used can be found in the appendix.

7.4 Country Selection

For this project, I need to decide on what list of countries to use. The GeoJSON file I will be using has a selection of countries, but a definitive list of countries doesn't exist.

There are multiple countries that are in dispute due to political or territorial reasons. For example, Taiwan is not recognised as a country by the United Nations due to China needing to agree, but it is a country in its own right. I will be including Taiwan in my list of countries.

Another consideration are countries which aren't included in the **ISO 3166 – 1 Alpha – 3** country code set which is what both the ECDC and GeoJSON file use. An example is Somaliland which is a self-declared state but is not recognised by the United Nations or the ISO-3166-1 country code, but the dataset does contain data so we will extend the country code set to include this.

There are more examples like this, but I will not mention every single one. I will be using the ISO-3166-1 country code set as a base, but I will be extending it to include countries which are not recognised by the United Nations for which I have data on.

Initially, I wanted to use UNIX timestamps to store the date, but I have since changed my mind. This is because of of how the HTML slider tag works. The slider expects consistent step values, and UNIX timestamps are not consistent due to daylight savings time.

To combat this, I am storing data in the string form of "YYYY-ww" where YYYY is the year and ww is the week number. This is a consistent format and will work with the HTML slider tag. I will have a function to convert the string to a date object when I need to use it. It is also ISO 8601 compliant, which is a common standard for this type of data.

7.5 Frontend

7.5.1 User Interface and Experience

Below is my dashboard created using D3.js and HTML.

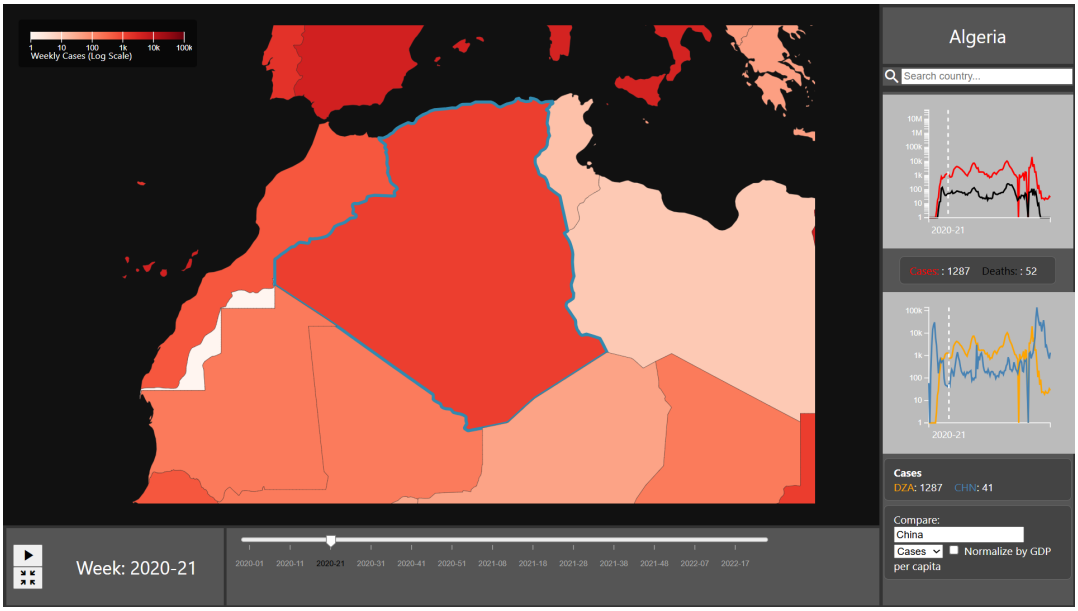

Figure 7.3: Dashboard

I will go into more detail about the features of the dashboard in the coming sections.

7.5.2 World Map

The world map in this project was a success in planning. I used the D3.js library to render the map and it worked well. The map is interactive and allows the user to zoom in on countries.

I was able to take advantage of features in D3.js like transition and zoom for a smoother feel to the map. The chloropleth map is a good way to represent the data and allows the user to see the hotspots of the virus.

I used the GeoJSON polygon data to render the map. This had many beneficial side effects, like: Choice of projection, the multipolygon feature of GeoJSON, rendering the data in SVG format, and the ability to use D3.js features like transitions and zoom.

7.5.3 Info Panel

The info panel is a div which is activated when a country is clicked. The div contains information about the country and the data. The information is displayed in a table format. The table contains the following information:

Country Graph

I initially planned to use a stacked area chart for this project, but I have since changed my mind. The reason is that the stacked area chart does not properly represent the data when used with a logarithmic scale. The populations of the susceptible and the cases/deaths are very disproportionate so a linear scale would not work. Instead, I will be using a logarithmic scale for the y-axis and have multiple lines on the same graph. This will allow me to show the data in a more meaningful way.

Comparison Graph

The comparison graph is a simple line chart with a logarithmic scale. The x-axis is the date and the y-axis is the number of cases/deaths. The country is able to be changed as well as the type of data. The options of data I chose to include are:

- Cases
- Deaths
- Cases per GDP per capita
- Deaths per GDP per capita

I intended on having more options, however I was unsure that a line graph would be the best way to represent the data with a binary part like Sovereign status.

7.5.4 Search Bar

The search bar is a simple input field which is designed to behave the same as just clicking a country. The user can type in the name of the country, and the field will show a list of countries which match the input. The user can then click on the country and the map will zoom in on the country. This is a good way to allow the user to find a country quickly without having to scroll through the map.

7.5.5 Zoom

I used a "zoom mode" in my application. It has functionality when a country is selected and sets the bounding box of the country to the view. This code also allows the user to freely zoom in and out of the map, and pan around the map. This is a d3.js feature which is used extensively.

7.5.6 Live Playback

The application has a live playback feature which allows the user to see how the data progresses over time. The user can click on the play button and the application will automatically update the data every week. When the button is clicked, it becomes a toggle button which allows the user to pause the playback. The playback automatically stops when the end of the data is reached.

7.5.7 Week Slider

The slider is a d3.js slider which allows the user to select a week. It required some logic to cast the position on the slider to a week. The slider is a good way to to allow the user to select a week without having to type in the date.

7.6 Backend

For the data in my application, I am using the file "interface.js" to load the data from the JSON file. This file is an async module which exports a function to load the data. The function can fetch all data or data specified in the parameters. The function will return a promise which resolves to the data. This is a simple way to load data from a file and is a common pattern in JavaScript.

7.7 Deployment

In this project, I used Docker to deploy my application. Docker is a containerisation platform which allows you to package your application and all its dependencies into a single container. The only prerequisite is that the host machine has the Docker engine installed. This is a good way to deploy applications as it allows for easy deployment and scaling.

I only used Docker in the production environment, as I was developing the application on my local machine. In the README file of my project, I have included instructions on how to run the application in both the node.js runtime and the Docker container.

I decided against using the Docker image to run the data cleaning script for debugging purposes. But Docker would be a good solution because of the dockerfile's capability for "layers". If the data has been cleaned already, the dockerfile will not need to run the data cleaning script again.

In my Dockerfile, I used the node.js runtime as the base image, and then copied the files into the container. I then installed the dependencies using npm and exposed the port 8080. You can see the Dockerfile below:

```
1   FROM node:20
2
3   WORKDIR /app
4
5   COPY package*.json ./
6   RUN npm install
7
8   COPY . .
9
10  EXPOSE 8080
11
12  CMD ["npm", "start"]
```

Figure 7.4: Dockerfile for the Application

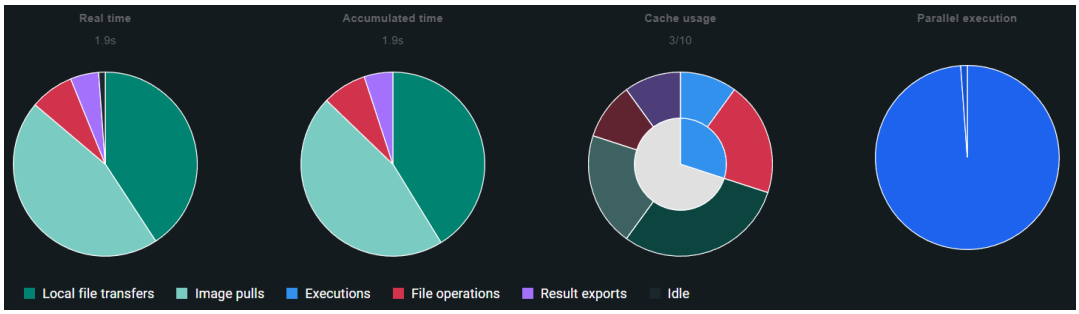

Figure 7.5: Building the Docker Image

Chapter 8

User Analysis

For this section, I conducted user testing on my application. I lead the testers through an activity where they had to use the application and extract information from it. I then asked them what they thought of the application and if they had any suggestions for improvement.

I followed the LEON ethics guidelines for this project, producing an informed consent form, participant information sheet, risk assessment and an approved questionnaire. I also made sure to anonymise the data and not use any personal information.

8.1 User Testing

The first set of questions was in regards to the UI and UX of the application and how easy it was to use. These were the questions asked:

Question 1: By clicking the country, select Egypt
Question 2: Using the zoom button, zoom into Egypt.

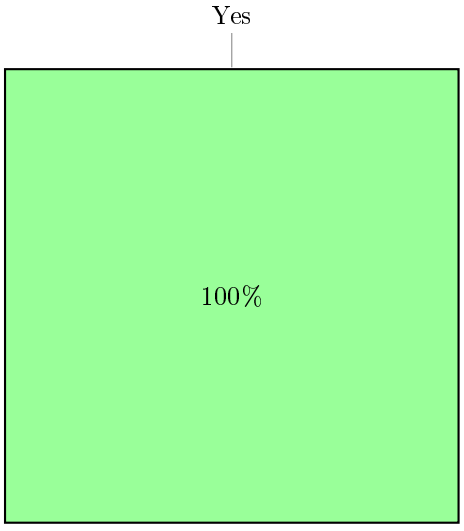

Figure 8.1: Question 1 Results

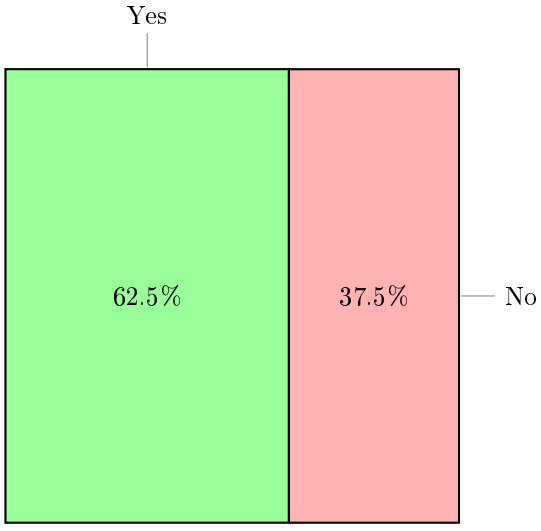

Figure 8.2: Question 2 Results

There was no issue with the first question, all users were able to select a country and see the corresponding data. However, the second question caused some issues. The main issue with the zoom feature was that the button to activate the zoom mode was not clear. The users were confused about how to activate the zoom mode and some of them did not know that they could click

on the map to zoom in.

The next set of questions was in regards to getting data from the application. I had 4 questions in this section:

Question 3: How many cases of Covid-19 was recorded in the 20th week of 2020 in Egypt?
Question 4: Using the comparison graph, did Egypt or South Africa have more deaths on the 45th week of 2020?
Question 5: On what week did Sweden have it's peak case count?
Question 6: Compare the cases per GDP between United Kingdom and Germany on the week 2021-48, which country had more cases per GDP?

The results of the questions are below:

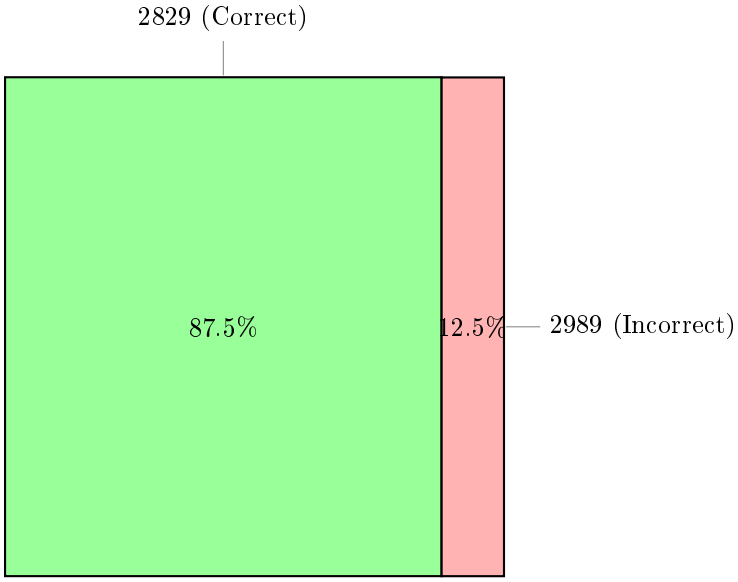

Figure 8.3: Question 3 Results

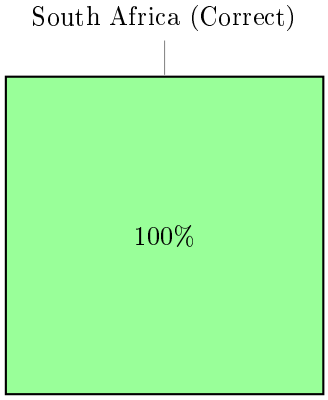

Figure 8.4: Question 4 Results

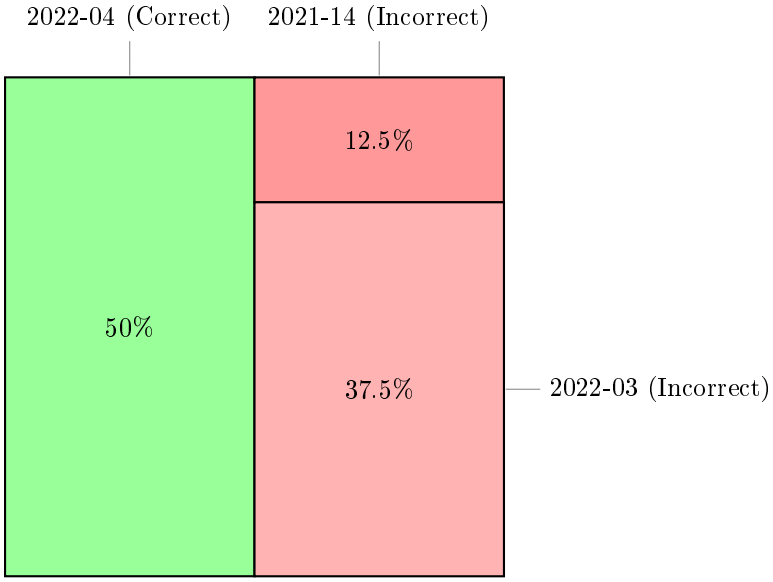

Figure 8.5: Question 5 Results

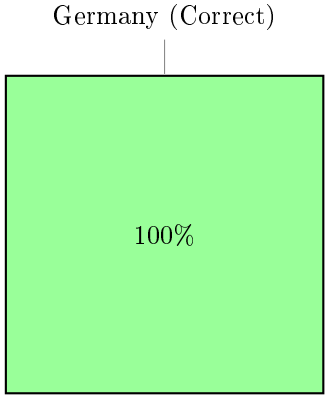

Figure 8.6: Question 6 Results

Overall, I am happy with the results of the user testing. My worst answered question was question 5, where only 50% of users got the question correct. This was somewhat expected as the question required the user to use the program in a way that was not initially intended. The question was asking the user to find the peak case count of a country, which is not a feature of the program. The users were able to find the data, but they were not able to find the peak case count. This is something I will be looking to improve in the future.

8.2 User Feedback

The users were overall happy with the application. They liked the design and the way the data was presented. The users also liked the interactivity of the map and how they could zoom in on countries.

However, I received many suggestions for improvement, some of which I will be implementing. The main suggestions were:

- The zoom button was not clear and the users did not know how to activate it.
  - The button was hard to identify and it was not clear what it did. I will be changing the button to a more recognisable icon and adding a tooltip to explain what it does.
  - The current behaviour of the button shows a globe icon when the map is not in zoom mode, and a zoom icon when the map is in zoom mode. This is not clear and I will be inverting the icons.
- The search bar was not clear and the users did not know how to use it.
  - The behaviour of the search bar and the comparison search bar were slightly different and one required the enter key to be pressed. I will be changing the search bar to behave the same as the comparison search bar.
  - The search icon was not a button like the users expected. I will be changing the search icon to a button and adding a tooltip to explain what it does.
- The users wanted the graphs to be viewable on a separate, larger window.
  - This is a good idea, however the code base is not set up for this. I will be looking into this in the future, but it is not a priority at the moment.
- Functionality to right click a country to add country to comparison graph.
  - I will not be implementing this feature as it does not follow good UX design. Adding a country implicitly is a better solution.
- Searching for a country did not highlight the country on the map.
  - This is actually an intended feature, and was a bug in the code. I added a fix to highlight the country when it is selected.

Chapter 9

Conclusion

In undertaking this project, I have met the objectives set. I have learned how to use multiple new technologies and how to use them together. I have also learned how to use Docker to deploy my application and how to use the fetch API to fetch data.

The web application was well received by the testers and they were able to use it to extract information from the data. The application is interactive and allows the user to zoom in on countries. The application is also able to display the data in a meaningful way.

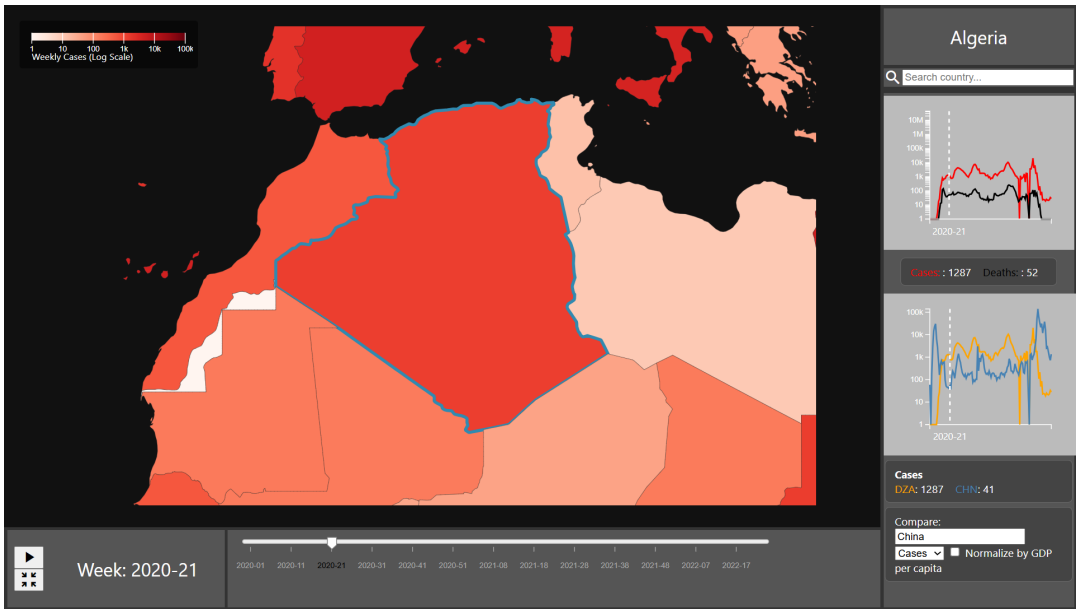

Figure 9.1: Application Screenshot

I have created a tool which can be used to visualise and analyse an arbitrary pandemic dataset. The tool is able to display the data in a meaningful way and can be used to find trends in the data. It can also be used to compare countries and find out which countries are doing better or worse than others.

Chapter 10

Glossary

- **Choropleth Map** - A map in which areas are shaded or patterned in proportion to the value of a variable being represented.
- **D3.js** - A JavaScript library, Data-Driven-Documents, used for producing dynamic, interactive data visualisations in web browsers.
- **JSON** - JavaScript Object Notation, a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate.
- **GeoJSON** - A format for encoding a variety of geographic data structures using JSON.
- **Node.js** - An open-source, cross-platform, back-end JavaScript runtime environment that executes JavaScript code outside a web browser.
- **SIR Model** - A simple compartmental model used in epidemiology to simulate the spread of infectious diseases.
- **Epidemic** - A sudden increase in the number of cases of a disease above what is normally expected in a population in a given area.
- **Pandemic** - An epidemic that has spread across a large region, for instance multiple continents or worldwide.
- **Docker** - A platfirm for developing, shipping, and running applications in containers.
- **ISO-3166-1** - The International Organization for Standardization (ISO) standard for country codes.
- **SVG** - Scalable Vector Graphics, an XML-based vector image format for two-dimensional graphics with support for interactivity and animation.

Chapter 11

Bibliography

[1] Hamada S. Badr, Benjamin F. Zaitchik, Gaige H. Kerr, and et al. Unified real-time environmental-epidemiological data for multiscale modeling of the covid-19 pandemic. *Scientific Data*, 10(1):367, June 2023.

[2] Ross Beckley, Cametria Weatherspoon, Michael Alexander, Marissa Chandler, Anthony Johnson, and Ghan S Bhatt. Modeling epidemics with differential equations. *Tennessee State University Internal Report*, 2013.

[3] Ndemic Creations. Plague inc. `https://www.ndemiccreations.com/en/22-plague-inc`, May 2012. Accessed: May 2025.

[4] European Centre for Disease Prevention and Control. European centre for disease prevention and control. `https://www.ecdc.europa.eu/`. Accessed: January 2025.

[5] Bill Gates. The next outbreak? we're not ready, March 2015.

[6] Klaus-Dieter Keller. Sir model, March 2019.

[7] Andy Kirk. *Data Visualization: A Handbook for Data Driven Design*. SAGE Publications Ltd, 2nd edition, 2019.

[8] Ash Kyd. Geojson vector maps. `https://geojson-maps.kyd.au/`. Accessed: January 2025.

[9] BBC News. "covid-19 pandemic: China 'refused to give data' to who team". `https://www.bbc.co.uk/news/world-asia-china-56054468`, February 2021. Accessed: January 2025.

Chapter 12

Appendix

12.1 restructure-json.py

```
import json
from collections import defaultdict

#########################################################
#ENTER UNCLEANED JSON FILE, e.g. "data.json"
oldJSONFile = "data.json"
#########################################################

# Load JSON data from file
with open(oldJSONFile, "r", encoding="utf-8") as infile:
    json_data = json.load(infile)

# Data structure to hold per-country data
countries = defaultdict(lambda: {
    "properties": {},
    "data": defaultdict(lambda: {
        "cases": 0,
        "deaths": 0,
        "cumulative_cases": 0,
        "cumulative_deaths": 0,
        "rate_14_day_cases": 0.0,
        "rate_14_day_deaths": 0.0
    })
})

# Initialize global totals
world_totals = defaultdict(lambda: {
    "cases": 0,
    "deaths": 0,
    "cumulative_cases": 0,
    "cumulative_deaths": 0
})
world_population = 0  # We will sum all populations

for entry in json_data:
    country = entry.get("country", "Unknown")
    country_code = entry.get("country_code", "N/A")

    # Populate country properties once
    if not countries[country_code]["properties"]:
        countries[country_code]["properties"] = {
            "country": country,
            "country_code": country_code,
            "continent": entry.get("continent", "Unknown"),
            "population": entry.get("population", 0),
            "source": entry.get("source", "Unknown")
```

```
            }
            world_population += entry.get("population", 0)  # Sum global population

        # Use the original year_week format (YYYY-WW)
        year_week = entry.get("year_week", "Unknown")

        # Extract indicator and values
        indicator = entry.get("indicator", "").lower()
        weekly_count = entry.get("weekly_count", 0)
        cumulative_count = entry.get("cumulative_count", 0)
        rate_14_day = entry.get("rate_14_day", 0.0)

        ##################################################################
        # AMMEND THIS AREA TO FIX TO YOUR OWN JSON FILE

        # Store values based on indicator type
        if indicator == "cases":
            countries[country_code]["data"][year_week]["cases"] = weekly_count
            countries[country_code]["data"][year_week]["cumulative_cases"] =
                cumulative_count
            countries[country_code]["data"][year_week]["rate_14_day_cases"] =
                rate_14_day

            # Add to world totals
            world_totals[year_week]["cases"] += weekly_count
            world_totals[year_week]["cumulative_cases"] += cumulative_count

        elif indicator == "deaths":
            countries[country_code]["data"][year_week]["deaths"] = weekly_count
            countries[country_code]["data"][year_week]["cumulative_deaths"] =
                cumulative_count
            countries[country_code]["data"][year_week]["rate_14_day_deaths"] =
                rate_14_day

            # Add to world totals
            world_totals[year_week]["deaths"] += weekly_count
            world_totals[year_week]["cumulative_deaths"] += cumulative_count


        ##################################################################

    # Add global totals as a "World" entry
    countries["WORLD"] = {
        "properties": {
            "country": "World",
            "country_code": "WORLD",
            "continent": "Earth",
            "population": world_population,
            "source": "Aggregated Data"
        },
        "data": world_totals
    }

    # Convert defaultdict to normal dict for JSON output
    final_result = {country_code: {"properties": data["properties"], "data": dict(
        data["data"])}
                    for country_code, data in countries.items()}

    # Write output to a new JSON file
    with open("pandemicData.json", "w", encoding="utf-8") as outfile:
        json.dump(final_result, outfile, indent=4)

    print("Data successfully transformed and saved to 'pandemicData.json'")
```

12.2 Gantt Chart