London COVID-19 TRACKER

A web application to inspect, visualise and predict infection counts.

By Abu Bakar Naseer, BSc Computing Project Report

Birkbeck College, UOL

13th of May 2021

Student ID:	13139991
Supervisor:	Carsten Fuhs
Number of words:	9,243

Statement

This report is the result of my own work except where explicitly stated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged

Abstract

COVID-19, the recent virus type of coronavirus, has quickly taken over the world and made humanity rethink how they travel, socially interact, and function in their day-to-day lives. London, having been the economic hub of the world, has particularly been hit the hardest compared to other major cities around the world.

With these rapidly changing times where lockdowns are put into place with little to no warnings, there is now a need for a new breed of software applications to monitor and analyse the spread of the infection without being glued to the grim news reports. While many such applications have already been developed and deployed for use by the public, a lot of these attempt to cater to a large number of countries, which does not allow them to provide more in-depth analyses of individual cities, and of course they only provide historical data which might not be of use to the user considering the rate at which this virus spreads. Additionally, these applications assume knowledge of complex epidemiological metrics and require the users to interpret these and draw conclusions themselves.

COVIDTracker.london is a web application that aims to provide a simple interactive map of London where users can find COVID-related information local to their residential area. This includes a simple straightforward danger metric that is representative of the chances of getting infected. It not only can be used for viewing historical data but also gives the user the option to select a date in the future to predict the infection counts for. This is helpful in the fact that users can safely plan future social events by utilizing the power of Artificial Intelligence. As complex as the application is, it can be accessed over both mobile and desktop ensuring a multitude of users can access it and help contribute towards a safe road back towards “normalization”.

Link to the demo of the application [1]

Table of Contents

1. Introduction

1.1 Description

In December 2019, Chinese authorities received reports of dozens of pneumonia cases from Wuhan where the cause was still unknown. The cause was days later identified by researchers to be a new virus from the family of coronaviruses, SARS-CoV-2. [2]

Figure 1.1 First group of coronavirus patients [3]

In the weeks following - the face of the earth rapidly started to cover by people experiencing persistent dry coughing caused by this highly contagious virus, effects of which were still majorly unknown. Through the power of the internet that connects all of us between continents and countries - people quickly realized the gravity of the situation and became vigilant regarding its spread. In March 2020, the World Health Organization officially characterized COVID-19 as a pandemic and admitted that this was the first virus from the family of coronaviruses to have been determined as one.

There was now stigma attached to the most normal of interactions. For example, an innocent cough in public became a cause for concern where everyone would visibly be uncomfortable and raise brows as it would imply that the cougher could be infected.

1.2 **Background / Existing Applications**

In such a dire time, there was a newborn need by the public to monitor the spread of this virus in their local areas and take caution as needed. From that time up until now (Mar 2021 as of writing), a variety of tools to visualise the spread of the disease have sprung up. A few examples follow.

The tech giant Google has started to integrate graphs into their search engine which automatically displays when you search for anything related to the virus. These tools either primarily showcase graphs or simple interactive maps which are easier to digest for the user.

Figure 1.2 Google Graph when searching "COVID".

Another such example is the UK government's website, which in addition to certain analysis graphs also offers an interactive map.

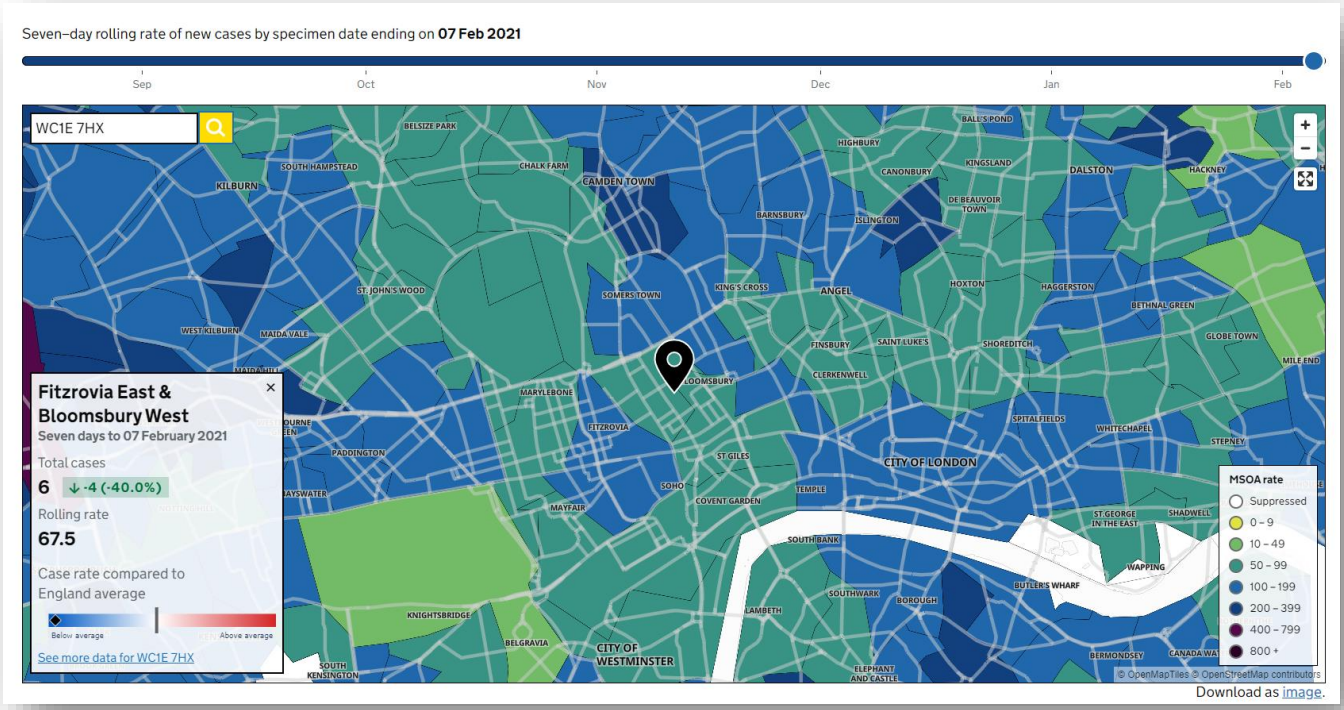


Figure 1.3 UK GOV Interactive Map [4]

It is important to note that this overwhelms the user with data and there is no single metric that the user can compare areas against. This report describes and details the development of another tool – **COVIDTracker.london** which aims to take the interactive maps a step further. It can be used to monitor London's current cases, danger levels of different boroughs as well as forecast future scenarios for the same data which currently a limited number of other applications offer.

2. Aims and Objectives

2.1 Context

As of March 2021, London had suffered through a total of three lockdowns and approximately 19 thousand deaths of citizens where the cited reason on the death certificate was COVID-19. [5]

Life, as people knew it, had changed such that they had to protect themselves with surgical masks before interacting with others outside their households and required frequent sanitization to keep themselves uninfected. Lockdowns were being enforced around the world rapidly with little warning.

It came as no surprise that there was now a need for newer smarter software tools to help warn the public of the situation and ease their worries.

2.2 Requirements

Keeping the above context in mind, this project was developed so that it could bring more awareness among London citizens of the threat to their lifestyle and health and provide them with a tool that they could use to take preventive measures against this virus.

2.2.1 Functional Requirements

The tool in question was scoped out with below “should-have” features:

1) Use of a simple well-defined metric “danger level” which could be used to compare the boroughs and areas.
2) Display and comparison of different danger levels of London’s boroughs which would help users decide if they wish to travel to these areas and would keep them up to date with the spread of the virus as the pandemic progressed.
3) Having been given a future date, the application should have been able to extrapolate the data and predict what the danger rate is likely to be for that date.
4) The application should encourage users to contribute to the dataset, and its prediction accuracy, by answering a quick survey detailing their health

concerning any COVID-related symptoms. Responses to these surveys should have been stored in a database for future use.
5) Availability of programmatic querying of the backend API directly if the users did not wish to use the web app provided.

2.2.2 Non-Functional Requirements

Further scoping of the features indicated the below as non-functional features:

1) The different danger levels would be provided with the help of an interactive tooltip map where the user could also enter their London postcode to find information relating to their Borough.
2) Instead of the COVID survey's questions being part of the application code, the questions would be stored and fetched from a database that allows new questions, to be added or updated, without the need for the redeployment of the application.
3) There would be an automatically generated API documentation page on the website which details its endpoints and parameters.
4) The tool would be provided in the form of a responsive web-application that can run on any modern smartphone device.

Development of such a tool will also come with the added benefit of providing the author with the experience of using and processing external data and then creating an API based on it. That resulting API would then be used in a user-friendly frontend. This is believed to be the future of development and Software Engineering as it demonstrated that you do not need to be responsible for all parts of a product to make it meaningful. (i.e, this project hinges on the data provided by the UK GOV's API. The backend processes this data and predicts it for use by the frontend. There is no need for the application to have its own data source.)

2.3 Use Cases

These requirements when expressed formally in the form of a diagram would appear as below.

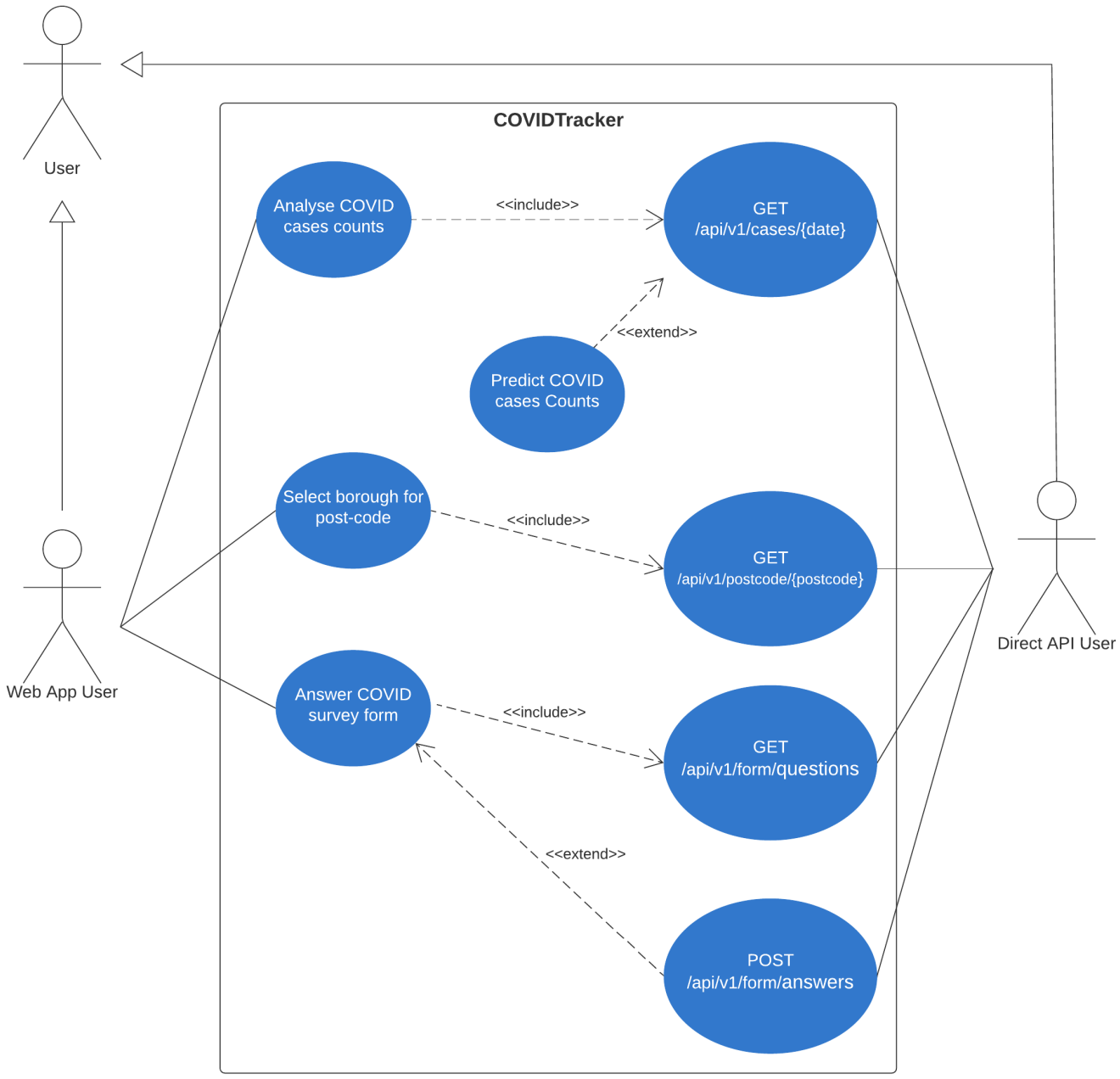


Figure 2.1 Use case diagram

3. Resulting Website

3.1 Home Page

The resulting website **COVIDTracker.london** is a mobile-friendly web application that can be accessed from any device after which users are shown the following landing page.

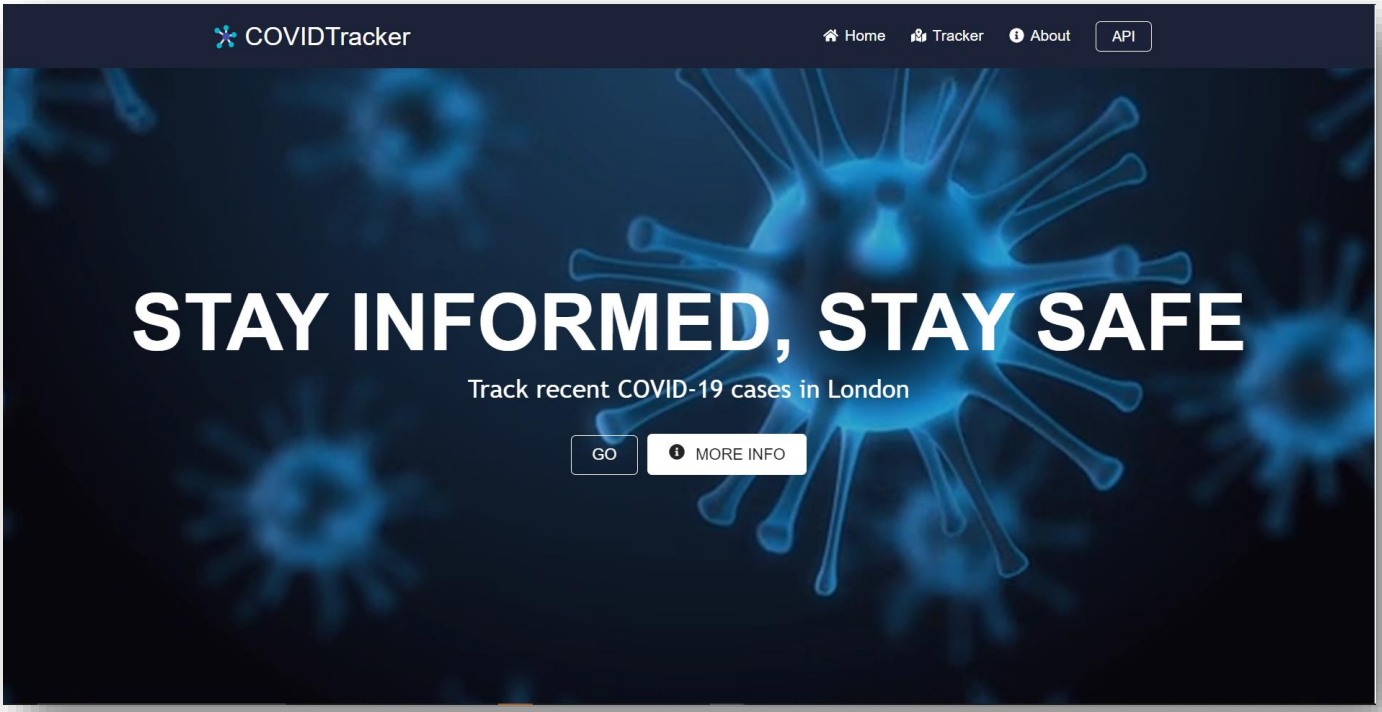


Figure 3.1 Landing Page

The users can then scroll down to see more information sections (see next two figures) representing the use cases of the application.

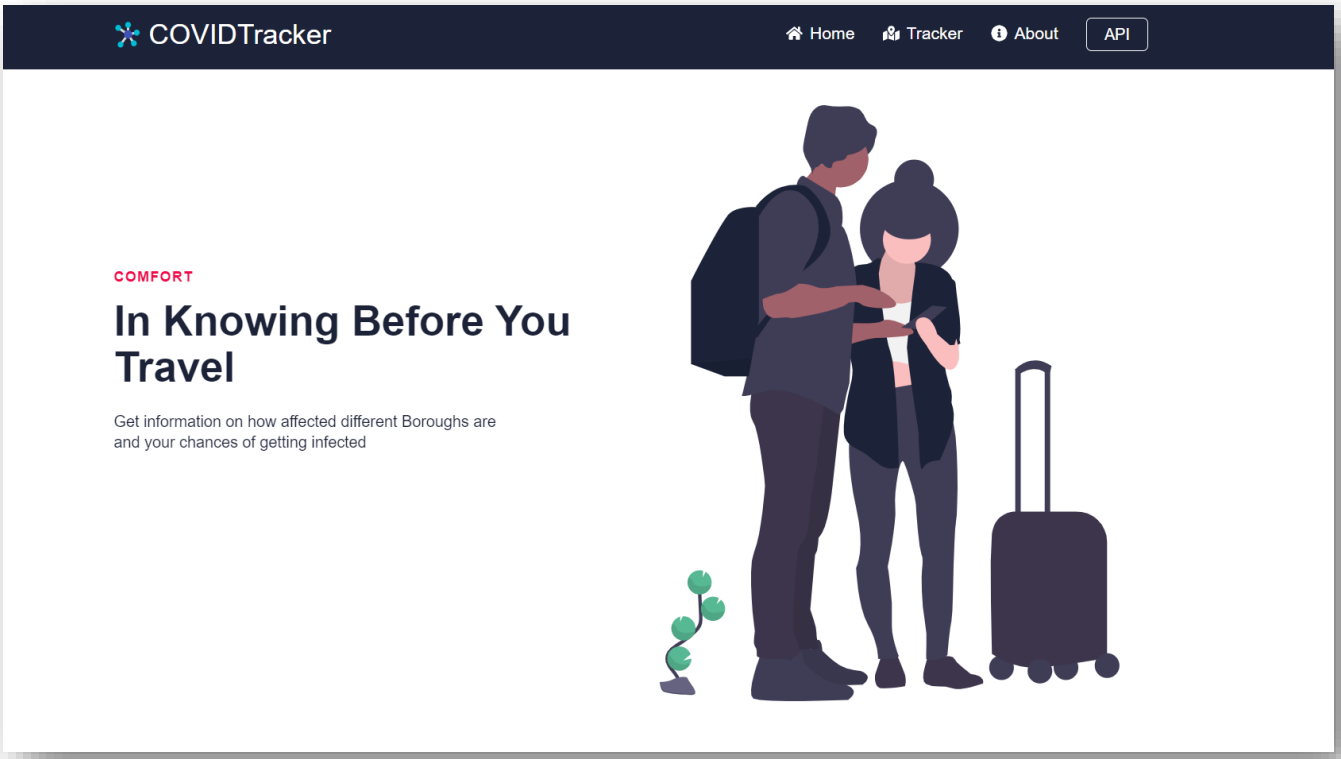


Figure 3.2 First information tile

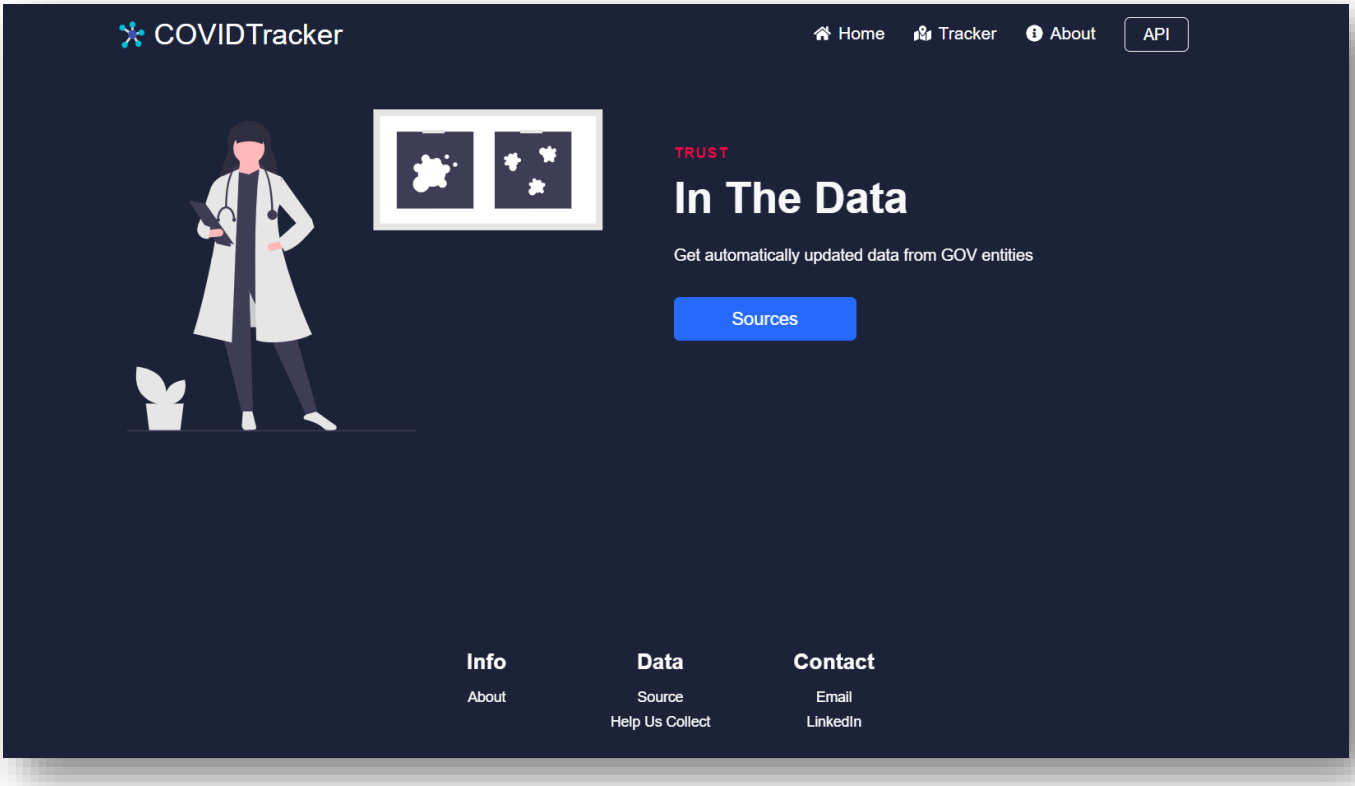


Figure 3.3 Third information tile

Alternatively, they can click on the **GO** button in the first section to navigate to the tracker page.

3.2 **Tracker Page**

This page showcases to the user an interactive map and contains most of the functionality of the application.

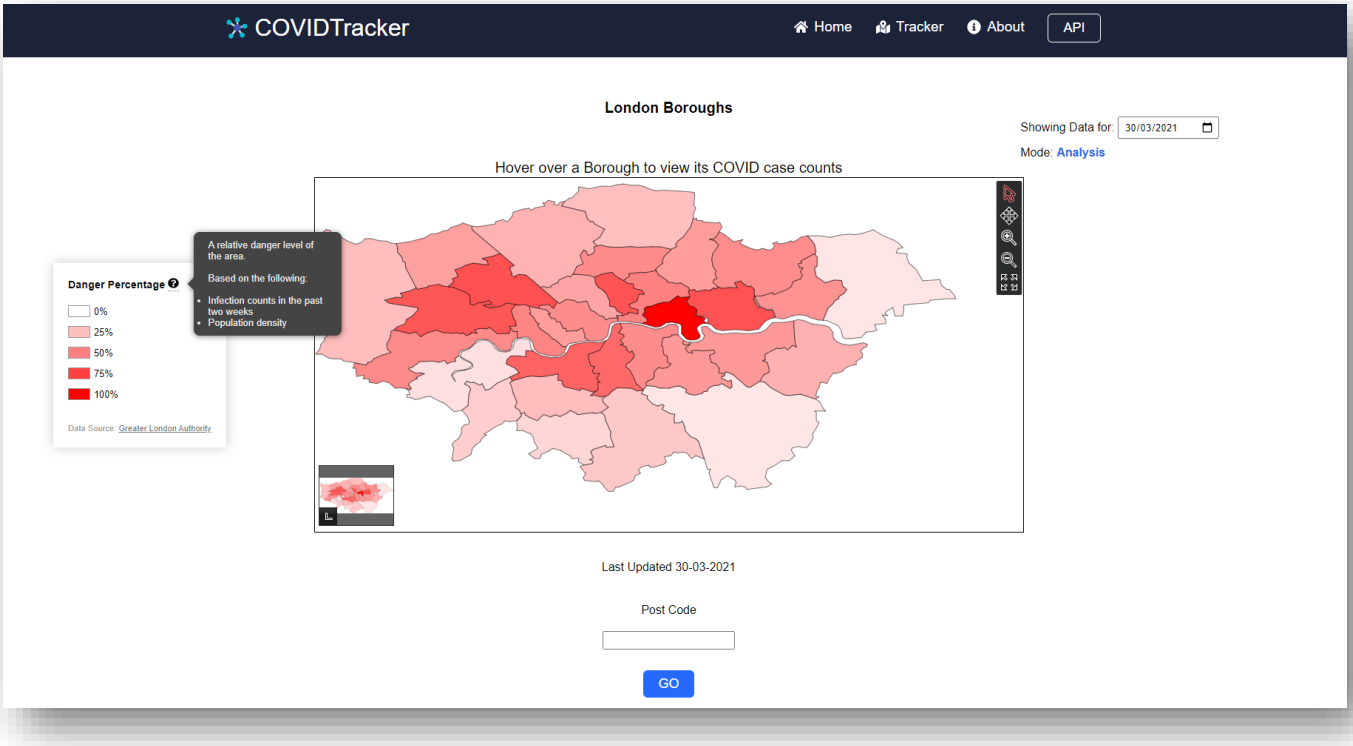


Figure 3.4 Main Tracker Page

The users can either hover over a borough to view its statistics and relative danger level or they can enter a postcode and it will automatically detect the borough as shown in the next few figures.

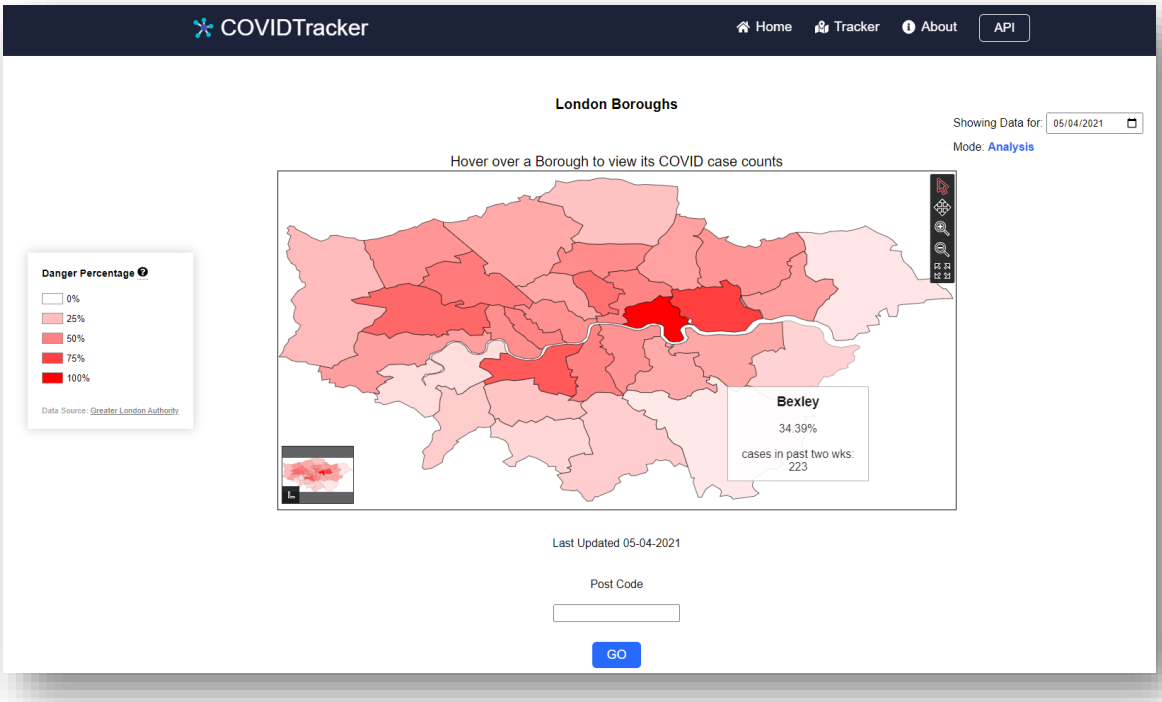


Figure 3.5 Borough hovering tooltip

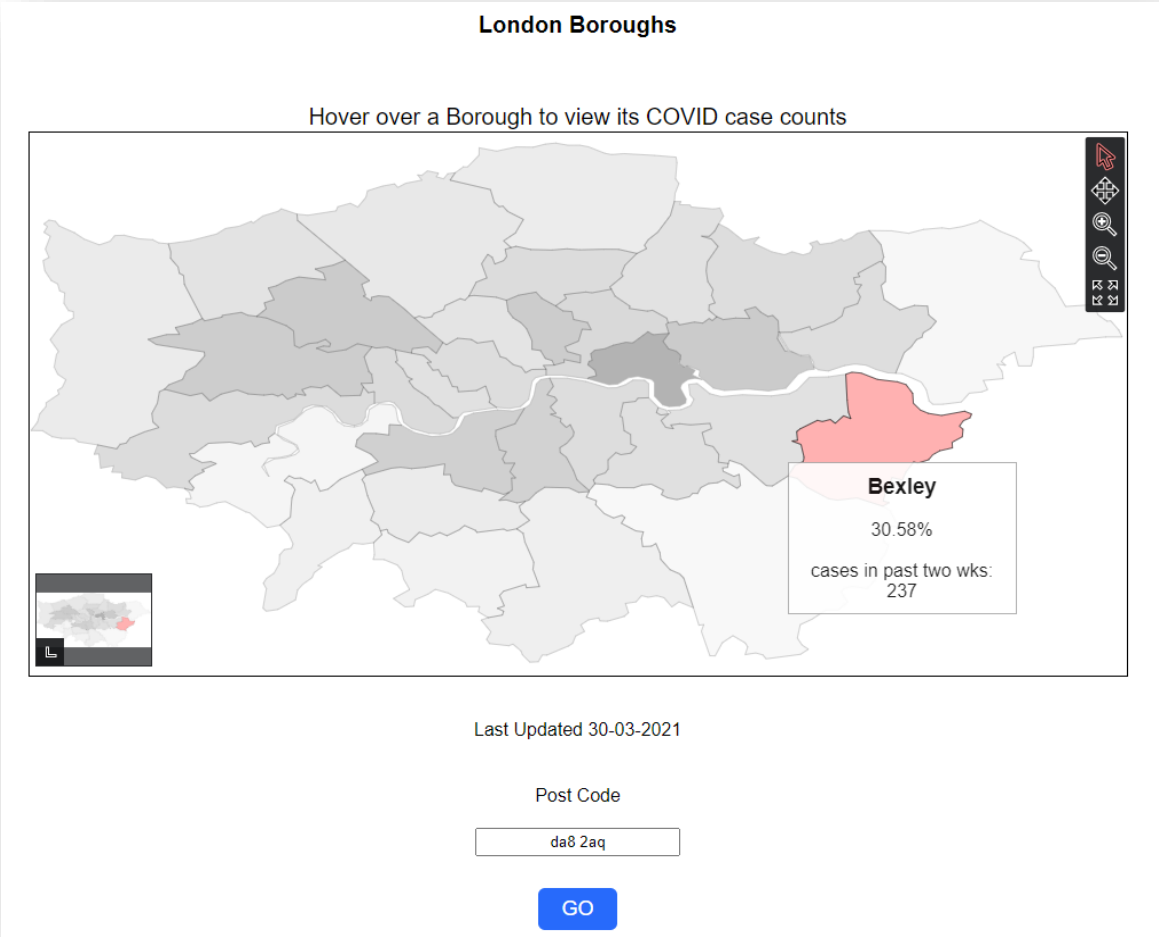


Figure 3.6 Automatic selection of borough through postcode entry

3.3 **Forecasting of COVID case counts using machine learning**

In addition to this, there is a date picker on the top right of the map which informs the user if they are in **Analysis** mode (the default) or **Prediction** mode which is activated when the user changes the date to one that is past the "Last Updated" date (the date the data is present for) and hence requires training a model on existing data and using it to forecast the infection counts for the selected future date.

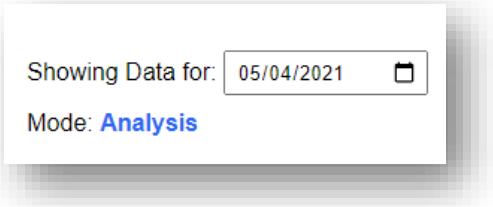


Figure 3.7 Date selector and application mode display

Using this feature, predicted infection counts and rates can be seen based on all the data that has been acquired up until that point. This forecasting has proven to be fairly accurate. This can be seen in the fact that in early January it reported close to zero new COVID cases for the weeks of June which is in line with the last milestone that the UK's four-step plan to come out of lockdown.

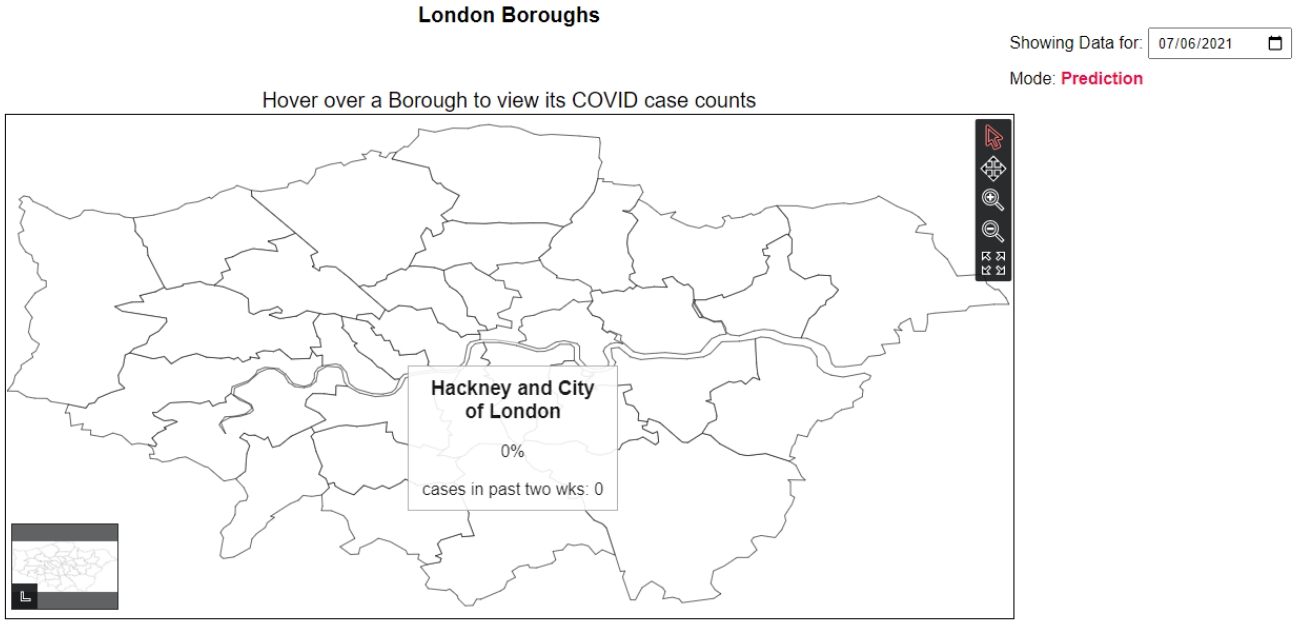


Figure 3.8 Zero danger % shown for June

3.4 COVID Self-Assessment Page

After spending a few minutes on the tracker page, the user will receive a pop-up in their browser window inviting them to answer a survey if they are finding the application useful.

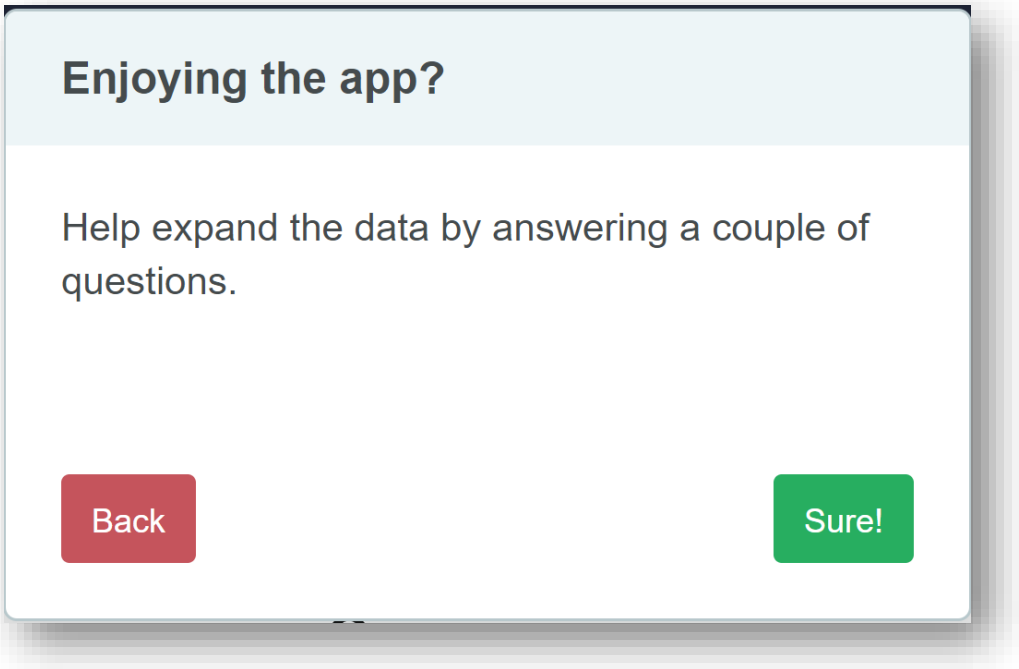


Figure 3.9 Popup on the tracker page

They can click out of this to return to the tracker page, or they can agree to it to be redirected to a survey page where questions appear on their screen one by one, which when answered can help in assessing how likely it is that the user is infected with COVID.

This data is currently not being processed and is only being collected and being stored in a database in hopes that once it is large enough it will be integrated into the app so that the model is also trained on this data in addition to the data provided by the UK GOV API.

3.5 About Page

Lastly, an about page exists that can be navigated to using the header or footer of the application where background information for the project can be found.

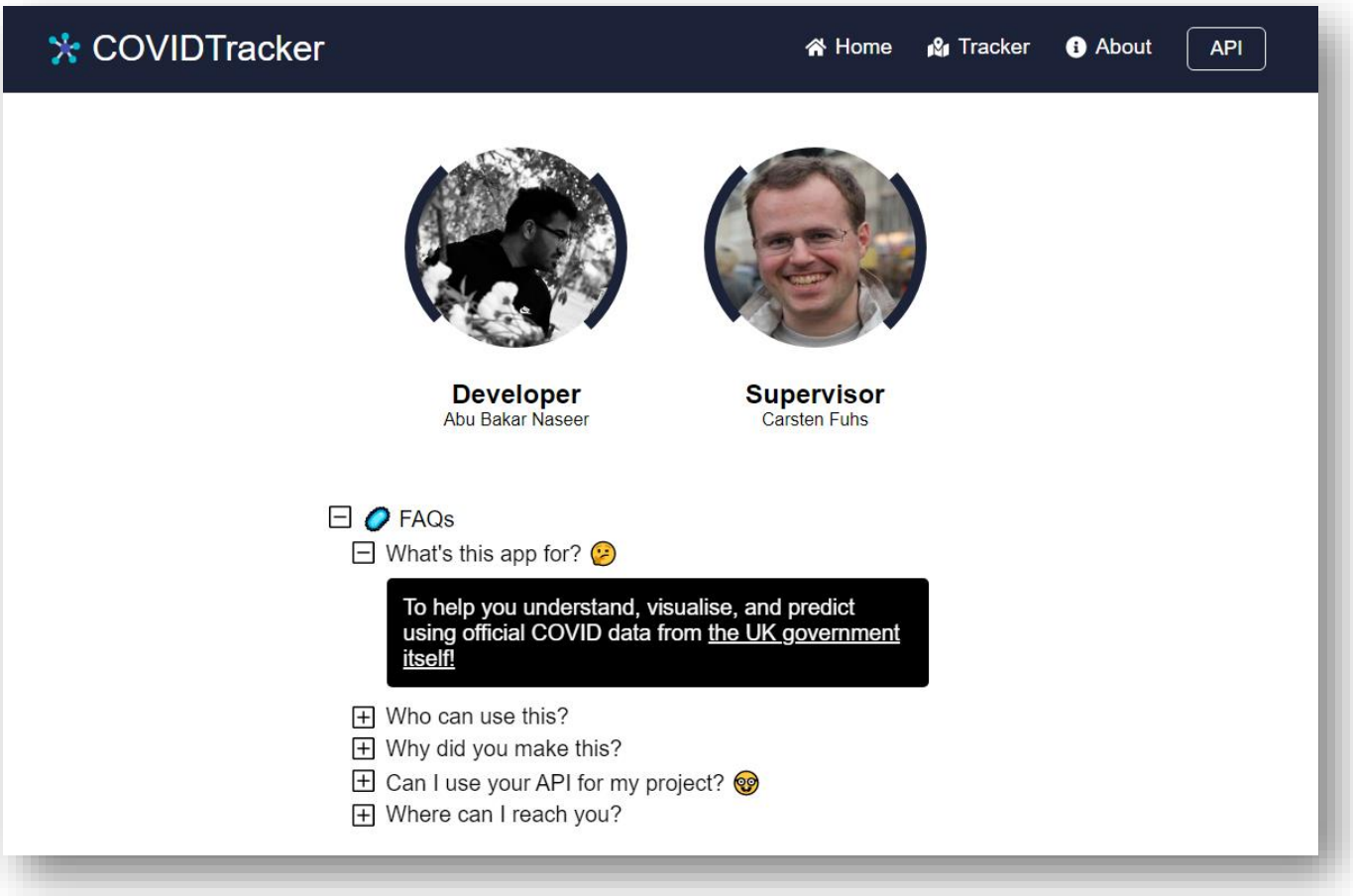


Figure 3.10 About page on the web application

Please see the appendix for the full list of features and the screenshots of the mobile version of the website.

4. Research and Design

This section will reflect the pre-implementation research that was carried out and aims to describe the context behind design choices that were made for the application.

4.1 Website feel and look

While the author was aware that there needed to be an interactive map as the main feature of the web app like the one shown in Figure 4.1, there were still some design questions that needed to be addressed. Such as how many pages the application would have, what each page would pertain to, and what the website would need to look like to maintain a serious demeanor considering the nature of the problem it was addressing i.e. a life-threatening virus.

4.1.1 Theme colour

The color **navy blue** was chosen as the official theme color of the web application after some research was done to conclude that it is seen as a “sign of stability and reliability” [6] – something that was much needed in these uncertain times.

4.1.2 Tracker Page

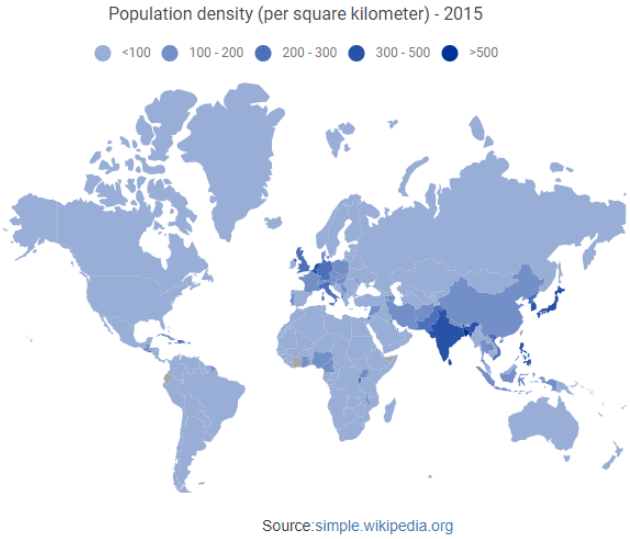


Figure 4.1 Example of a statistical map

The initial wireframe mockup of the website consisted of the below which highlighted the type of information that needed to be shown for each London Borough in the form of a tooltip. This also showed the three forms of inputs that the application needed to accept – date, post-code, and user mouse hover events to show the tooltip at the appropriate location of the map.

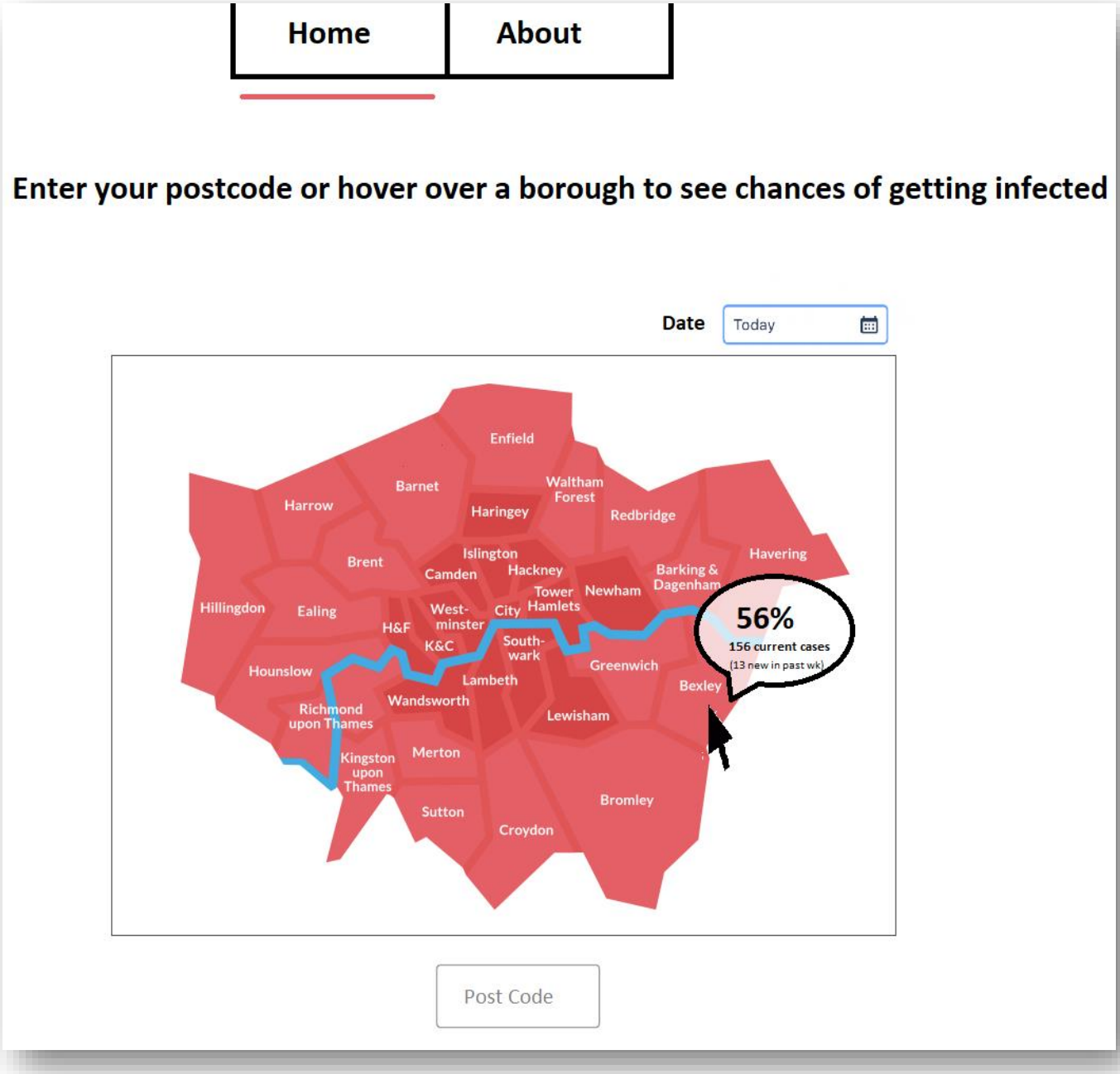


Figure 4.2 Initial wireframe

Among other adjustments and features additions that were made during development, the color of the boroughs was changed so that its colour would scale according to their danger percentages and the names of the boroughs were removed from the map's initial display to reduce crowdedness of the map such that they would only show in the tooltip.

4.1.3 Covid Self-Assessment Form

The COVID self-assessment survey part of the application needed to resemble Typeform's design. Typeform is a Software as a Service providing interactive well-designed user forms but charges a monthly fee for integrating with third-party websites which is why it was decided that **COVIDTracker** would not depend upon it and would rather try to use the design as inspiration.

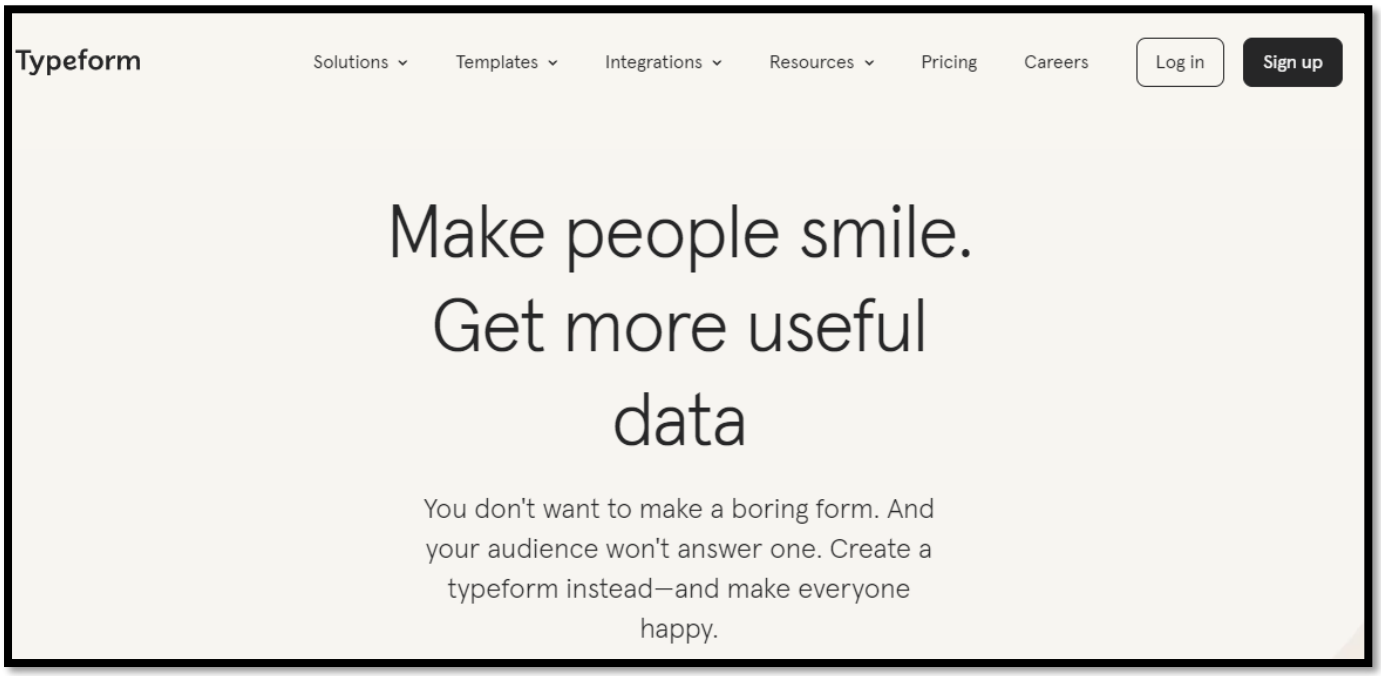


Figure 4.3 Typeform's details on their website [7]

In Typeform style forms, the questions appear to slide into and out the view pane and have high usability ratings which the author was quite impressed by and wanted to integrate into this web application at some level. Therefore, the first draft of the survey was first created in Typeform which would then be used as design inspiration to create the survey in the **COVIDTracker** web app.

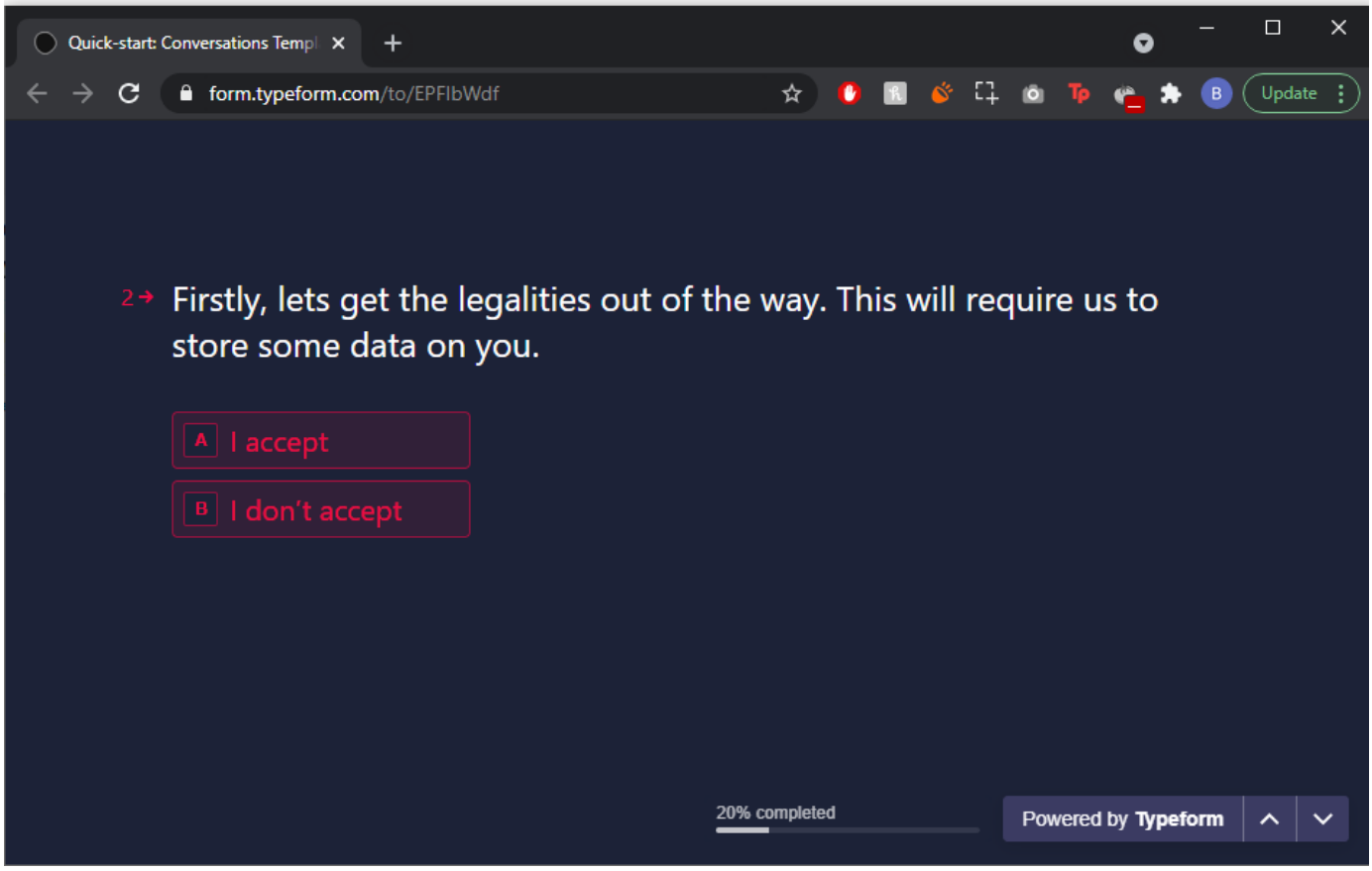


Figure 4.4 Design inspiration for the survey

4.2 Danger % Calculation

There was a need for a single representative metric that the users could rely upon for comparing different London Boroughs and one that was simple enough that the users could glance over it without needing to try to understand complex statistical terms. This metric had been so far called the “Danger Percentage” in mockups, but there had been no formula decided.

To find out the basis for the formula, some statistic studies were considered and one that particularly stood out was one where the authors had analysed populations of cities in Algeria against their COVID case counts [8].

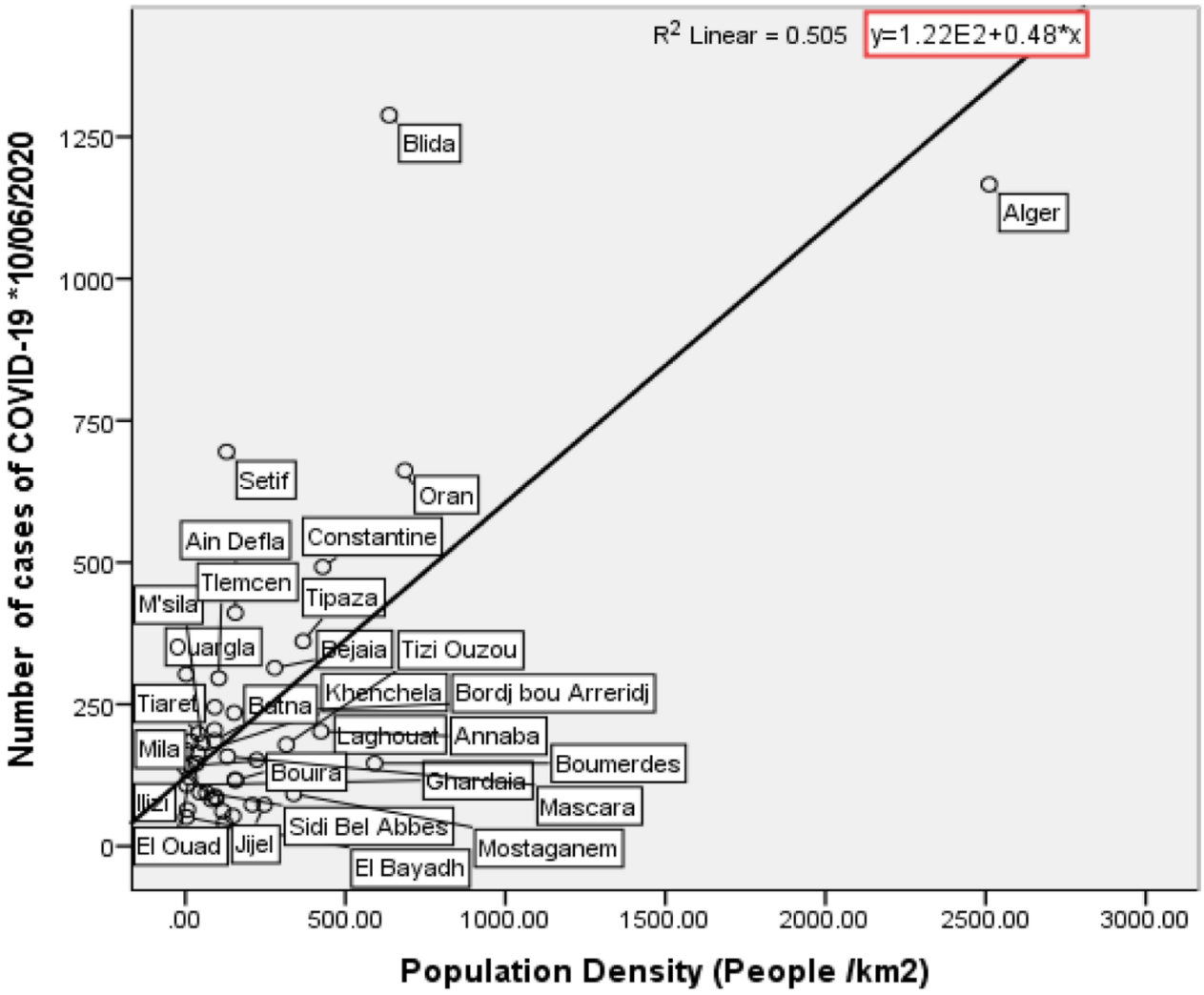


Figure 4.5 One of the graphs used for analysis in [8]

The authors of this report concluded that the population density of an area had a high correlation with the number of COVID cases such that the case counts increased as the population density increased. Keeping the result of this study in mind, the following formula was designed for a single borough.

$$\textit{Absolute Danger Value} = \textit{Total COVID Case Count} * \textit{Population Density}$$

In layman's terms - the danger value of a borough would increase if either of the factors, case count or population density, increased (where danger value is related to and proportional to the chances of getting infected in a borough).

A second study that also stood out, during the research phase, indicated that most people recovered from COVID-19 within two weeks [9] which meant that the formula could be further optimized so that only COVID case counts for the last two weeks for a borough were considered and allow for people, who had already recovered from COVID, to be excluded.

$$\begin{aligned} &\boldsymbol{Absolute\ Danger\ Value} \\ &\quad = \boldsymbol{COVID\ Case\ Counts\ for\ last\ 2\ weeks} \\ &\quad * \boldsymbol{Population\ Density} \end{aligned}$$

This would bring down the metrics down to a smaller magnitude and make them more manageable for calculation of any other derived metrics.

Finally, to convert a single borough's danger value into a relative percentage that would be compared against different boroughs, the below standard formula was used for converting values into percentages.

$$\boldsymbol{Danger}\ \% = \frac{\boldsymbol{Danger\ Value\ of\ Borough}}{\boldsymbol{Max\ Danger\ Value\ from\ all\ Boroughs}} * \mathbf{100}$$

This scaled the values to be between 1 to 100 where 0% would indicate that the borough was the least dangerous from all the 32 London boroughs and 100% would indicate it was the most dangerous of all boroughs and therefore had high COVID case counts and/or population density.

This relative scaling meant you could not compare the danger percentage of a borough on different days or time periods because it only served to represent a relative figure at a single point in time. For example, a 0% danger percentage case borough could possibly have the same danger percentage with 100 covid case counts on a different day if its ratio in the above formula remained the same but this was considered not to be a problem because the application could then be easily made to show the absolute danger value which is comparable for a borough on different times.

4.3 Data Sources

After having formulated the danger metric, the final part of the design and research phase revolved around finding the correct data sources for the population density and the COVID case counts for each borough. To ensure high reliability and data quality it was decided that it was best to utilize official government data only for the application.

Grid | Graph | 13952 records | « | 1 – 100 | »

area_name	area_code	*date*	new_cases	total_cases
Westminster	E09000033	2021-04-21	1	14139
Wandsworth	E09000032	2021-04-21	3	22128
Waltham Forest	E09000031	2021-04-21	3	24887
Tower Hamlets	E09000030	2021-04-21	7	28792
Sutton	E09000029	2021-04-21	2	15899
Southwark	E09000028	2021-04-21	5	22399
Richmond upon Tha…	E09000027	2021-04-21	3	10825
Redbridge	E09000026	2021-04-21	2	32853
Newham	E09000025	2021-04-21	4	35518
Merton	E09000024	2021-04-21	2	16215
Lewisham	E09000023	2021-04-21	0	21631
Lambeth	E09000022	2021-04-21	5	24205
Kingston upon Thames	E09000021	2021-04-21	3	12234
Kensington and Chels…	E09000020	2021-04-21	2	8983

Figure 4.6 Data for COVID Case Counts for each borough

Therefore, for the COVID case counts for each borough the above data source was chosen which is provided and maintained by the official government entity *Greater London Authority* as part of its **Coronavirus (COVID-19) Cases and Vaccinations** dataset [10]. It was also confirmed, before the final decision, that this data was available in the form of an API which the COVIDTracker application could query.

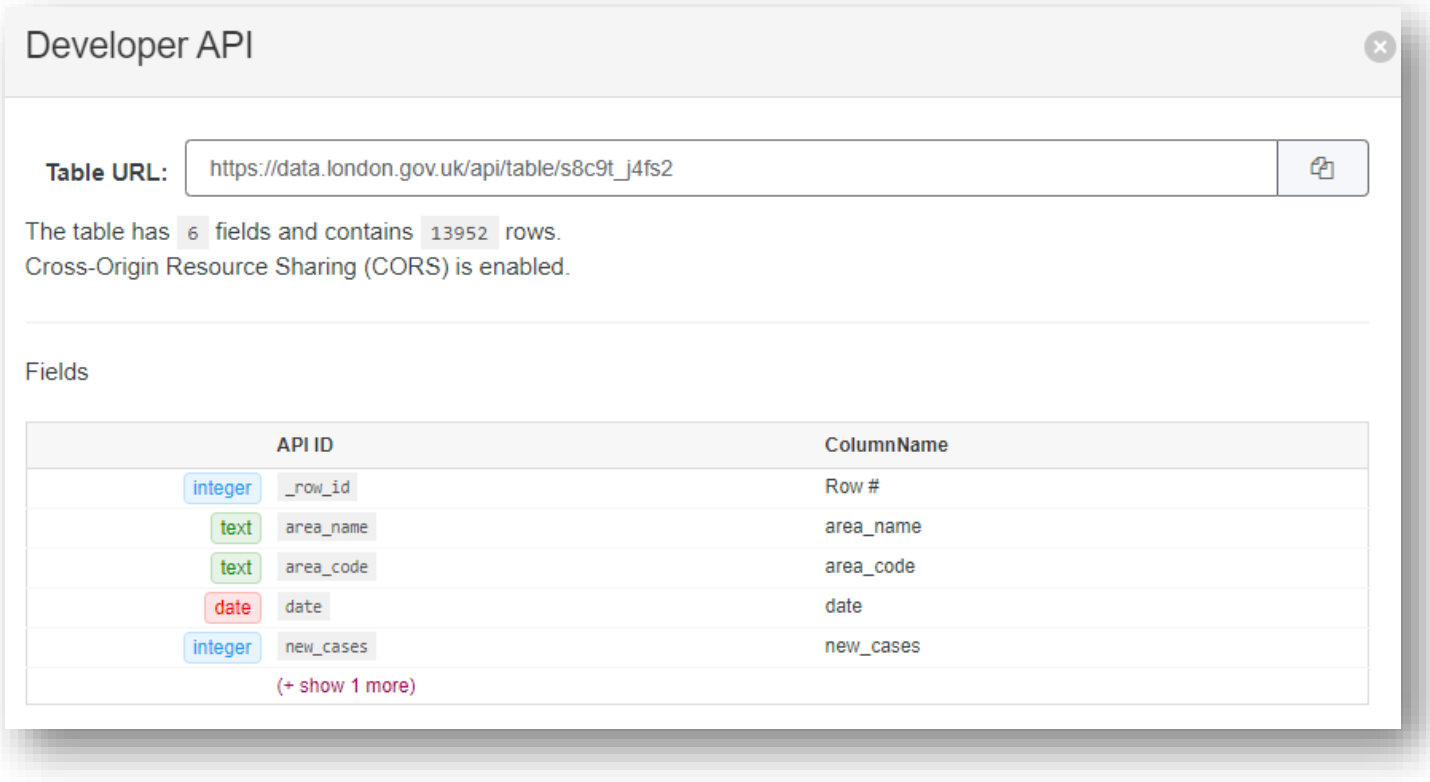


Figure 4.7 API information

For the population density, there were two official datasets found where one considered the daytime[1] population densities only and the other considered the total population density. The latter was chosen over the daytime population since the daytime population was thought to be diluted by the great number of tourists that London usually sees but is not expected to be present in the same quantity now due to the COVID pandemic.

API data sources are important for rapidly changing data such as for the COVID case counts data, but population density data is only ever collected once every few years and the data is then projected and estimated for future years. This data was therefore only available in a CSV file format that could be downloaded from the **Land Area and Population Density, Ward and Borough** dataset [11] which is also maintained by the *Greater London Authority*.

[1] Defined in the dataset as "The estimated number of people in a borough in the daytime during an average day".

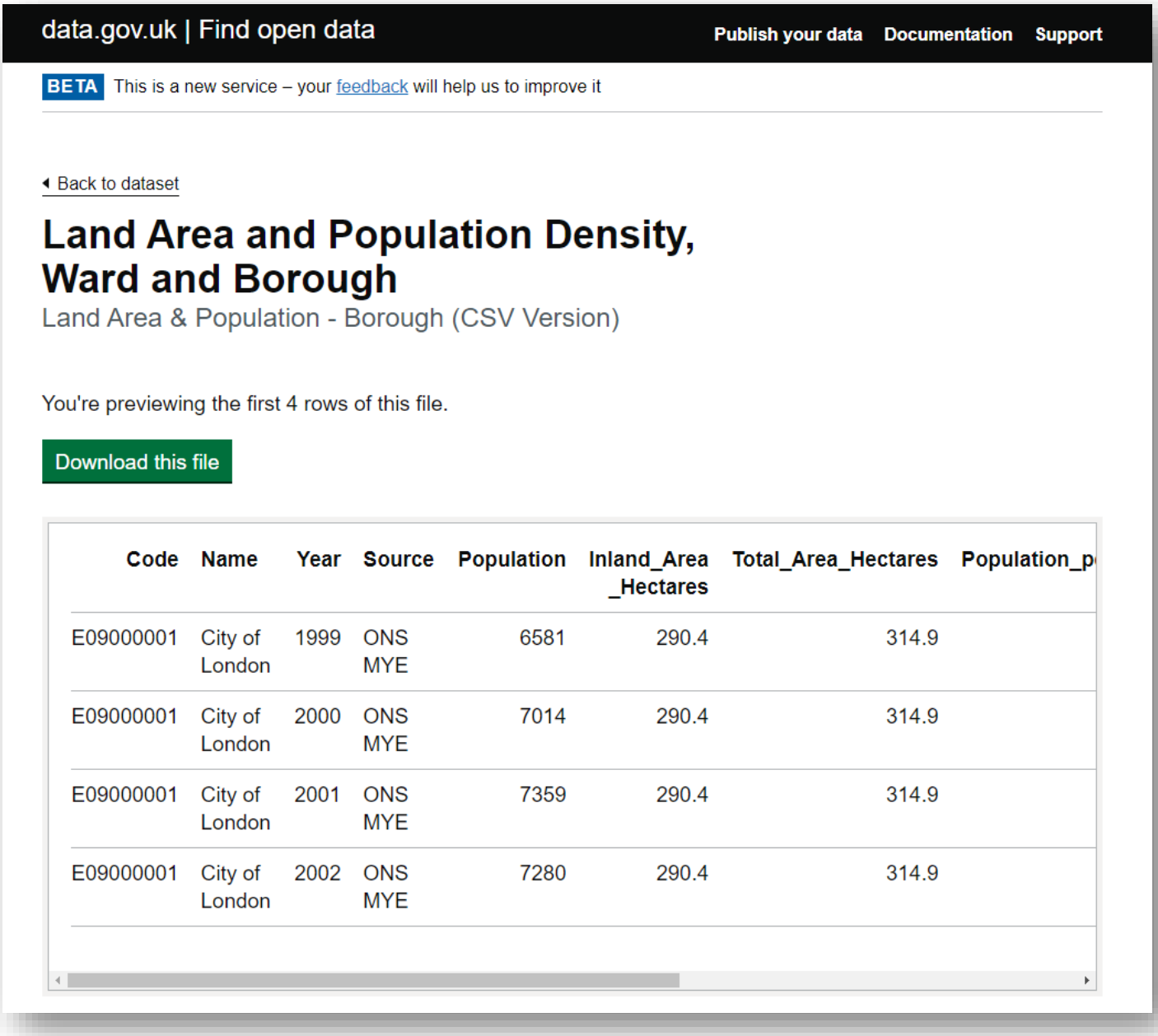


Figure 4.8 Dataset used for population densities of boroughs

5. Implementation

An important part of an application, second to only its features, is its technical implementation as it decides how reliably the application operates and how reliably it can deliver its value to the users. This section looks at the different implementation details that brought **COVIDTracker.london** to life. 5.1 Language, Tools, and Technologies reviews the tech stack that was used and reasoning behind each chosen component, then 5.2 Application Architecture describes the architecture in a more strategic sense, and 5.3 Development process will finish off by dictating the development process end to end.

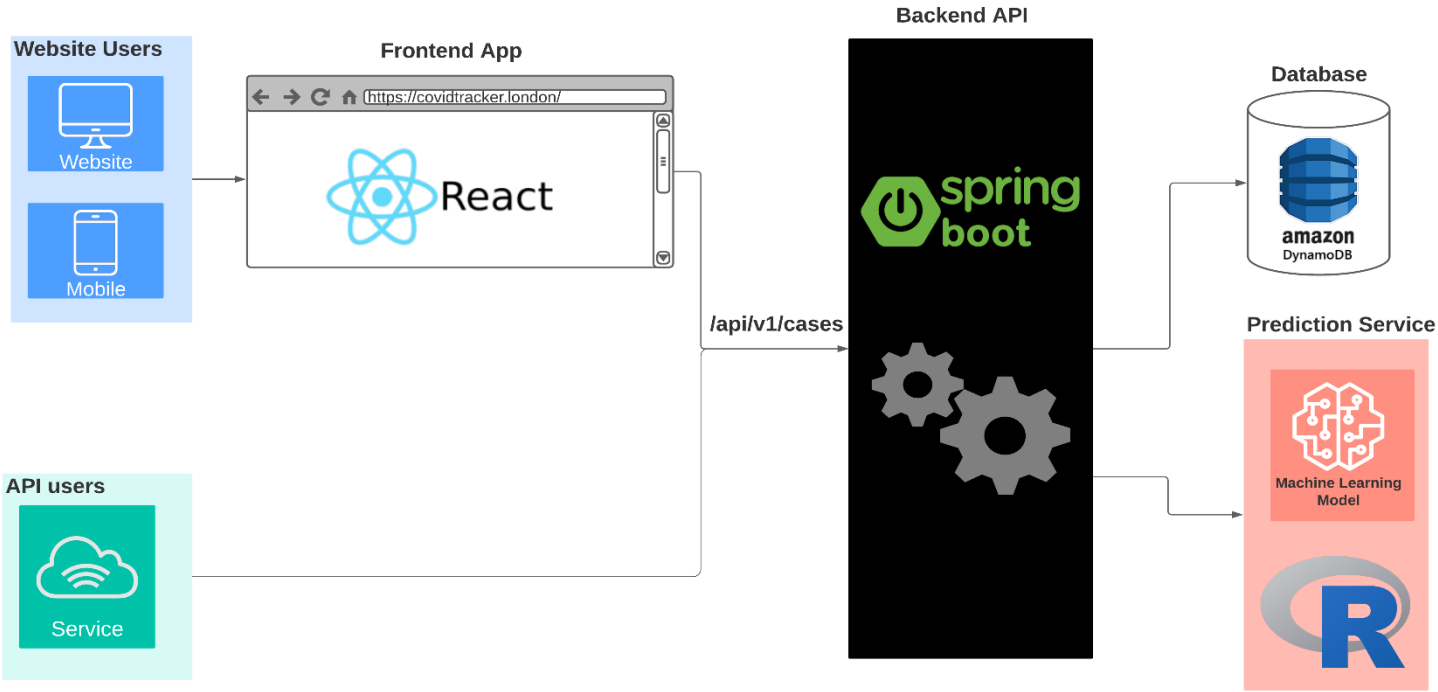


Figure 5.1 System diagram

5.1 Language, Tools, and Technologies

Links for any libraries or frameworks mentioned in this sub-section can be found in Appendix B: Tools and technologies.

5.1.1 Frontend Application

The frontend application is built using React.JS as it is a matured framework that allows for reusability and fast-paced development as many of its components are already available from third-party libraries and can be reused or modified, as necessary.

5.1.2 Backend API

The backend API is written using the Spring Boot framework due to its popularity among Java developers. It is a convenient method of using the industrial-strength Spring framework without having to deal with the complex boilerplate XML configurations which are instead done with Spring Boot's new annotations which enhance developer productivity.

Spring framework manages the instances of objects using what is called an IoC[1] container where the lifecycle (creation, assembly, destruction, etc.) of the objects is left up to the framework and the programmer only specifies what objects it needs by specifying @Bean's. The framework can then inject them where they are referenced automatically. This allows for simpler code where two different areas of the application requiring an object of the same class can use the same instance instead of creating a new one. Programmers can then avoid having to repeat how and from where to get object instances or how they will be created.

Spring also comes with a built-in web server so that the resulting packaged JAR file can be deployed directly without the need for additional setup.

5.1.3 Database

The database of choice is the NoSQL key-value document store DynamoDB which is provided by Amazon. This was chosen as it offers horizontal scalability and allows schema-less objects to be saved quickly efficiently which would help

[1] Inversion of Control is a programming principle where the calling method is made responsible for providing the required object instances instead of the called method having to create its own.

save responses from the applications COVID survey flexibly where the changes to the questions would not require changes to the databases schema.

5.1.4 Prediction Service

The prediction aspect of the application is powered by the statistical analysis language R which the author had some previous experience with and was also preferred over python since it is innately built as a statistical language and therefore is more capable and feature-rich for machine learning.

5.1.5 Cloud Provider

Amazon Web Services was chosen due to its dominating status in the industry and it having the largest market share out of any other cloud providers like Google Cloud Platform and Microsoft Azure.

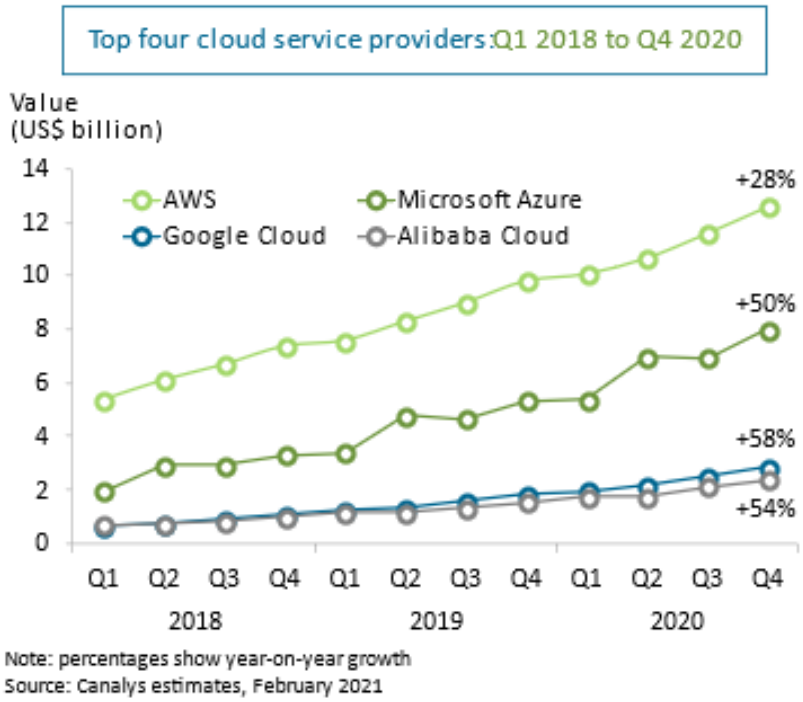


Figure 5.2 Cloud providers market value over time [12]

Fortunately, AWS also provided a year of free services to new joiners which was apt to this project's timeline.

5.2 Application Architecture

In this section, we consider the structural design of the application and the high-level components and modules that complement it.

5.2.1 Multitier Architecture

Since the inception of the idea for this project, it had been decided that this web application would utilize a multitier architecture which is more extensible and would allow the application to be expanded for a variety of new use cases.

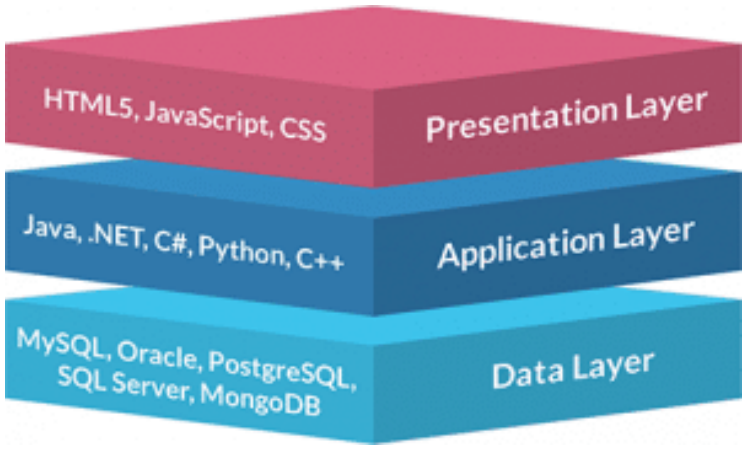


Figure 5.3 3-tier architecture [13]

Some other benefits of a multitier architecture include:

1) Decrease in complexity by being able to develop the three tiers independently.
2) Modularity between the presentation layer and application layer meaning that the presentation layer could either be removed and the application be used directly (direct access to the backend API), or it could be directly replaced by a different frontend framework entirely.
3) Improving scalability by allowing the hosting of each layer on different web servers, resources for which can be expanded individually.

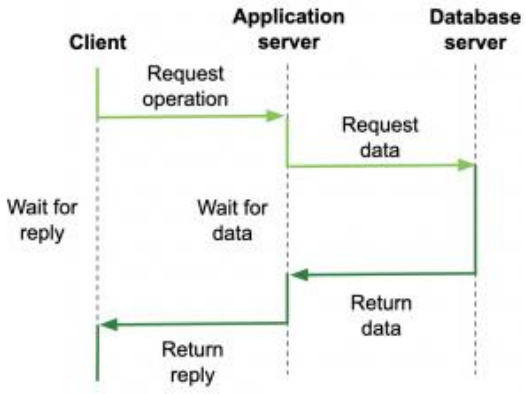


Figure 5.4 Interactions between tiers triggered by a single request from user [14]

Note that this is different from a single-tier monolith which when hosted on the same server can cause dependencies and strong coupling between different layers.

It is important to clarify that for the **COVIDTracker** application, even though it was designed and created as three tiers, the deployment of the **Presentation Layer** (React App) and the **Application Layer** (Backend API) is done on the same server since the frontend utilizes a minuscule amount of resources and hence can share the server resources with the backend code to save computing costs but can easily be split into different webservers to allow independent scaling in the future.

5.2.2 Rendering the Web Page (CSR vs SSR)

The choice of where exactly to render the web page that is being served to the user is application dependent and can sometimes be confusing but is an important one as it directly influences the page's usability and the general user experience. Client-Side Rendering downloads all the code needed for the application and executes it on the user's browser to render the page, this may involve a series of API calls to the webserver to fetch the user-specific information needed, which in turn means that different parts of the web page may load incrementally.

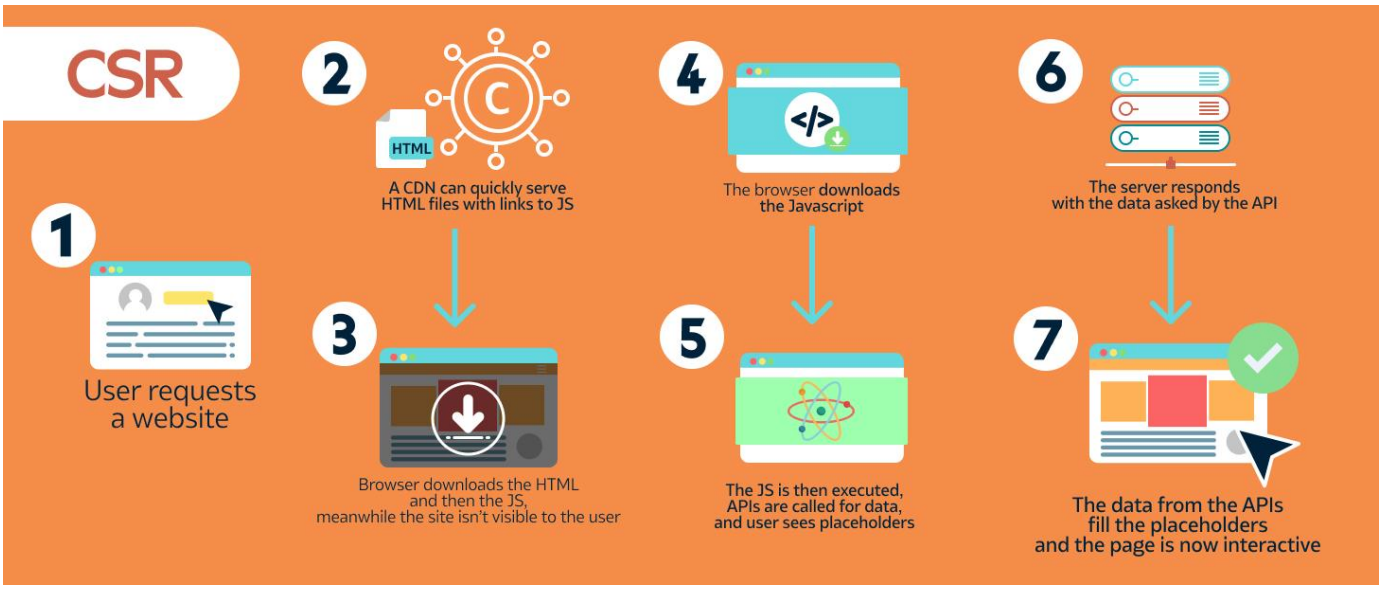


Figure 5.5 How Client-Side Rendering works [15]

Whereas in Server-Side Rendering the page is rendered on the webserver, personalized according to the user's request, and then downloaded by the user's browser. One advantage that this method has over the aforementioned is its better indexing and ranking by searching platforms (See: Search Engine Optimization).

Figure 5.6 Incremental loading in CSR
(source: Google Images)

For **COVIDTracker**, Client-Side Rendering (CSR) was chosen over Server-Side Rendering (SRS) since it allowed for richer interactions and encouraged the use of widely available libraries to design and bring the idea to life.

The only downside to this was the increased initial load time for the website. For example, currently, if a user switches to the tracker page of the application- even though the page will load instantly, the interactive map itself might take some time since it requires additional processing time but this an acceptable sacrifice for the enhanced UI features.

Due to having used CSR, there are essentially two independent applications – the frontend and the backend that communicate with each other through a REST API.

5.3 Development process

5.3.1 Phase 1 – Backend API setup

The backend API for the application naturally needed to be the starting point as it was going to be doing most of the processing and was the most complex to develop of the whole application without which no other parts could be implemented.

It had already been established in **5.1.2** Backend API that this part was going to be done using Spring Boot. Spring Boot provides an online interface where you can select your dependency manager of choice and a few package dependencies to start your application off. This is formally known as "bootstrapping an application".

For bootstrapping the **COVIDTracker** backend, the below options were selected in the web client.

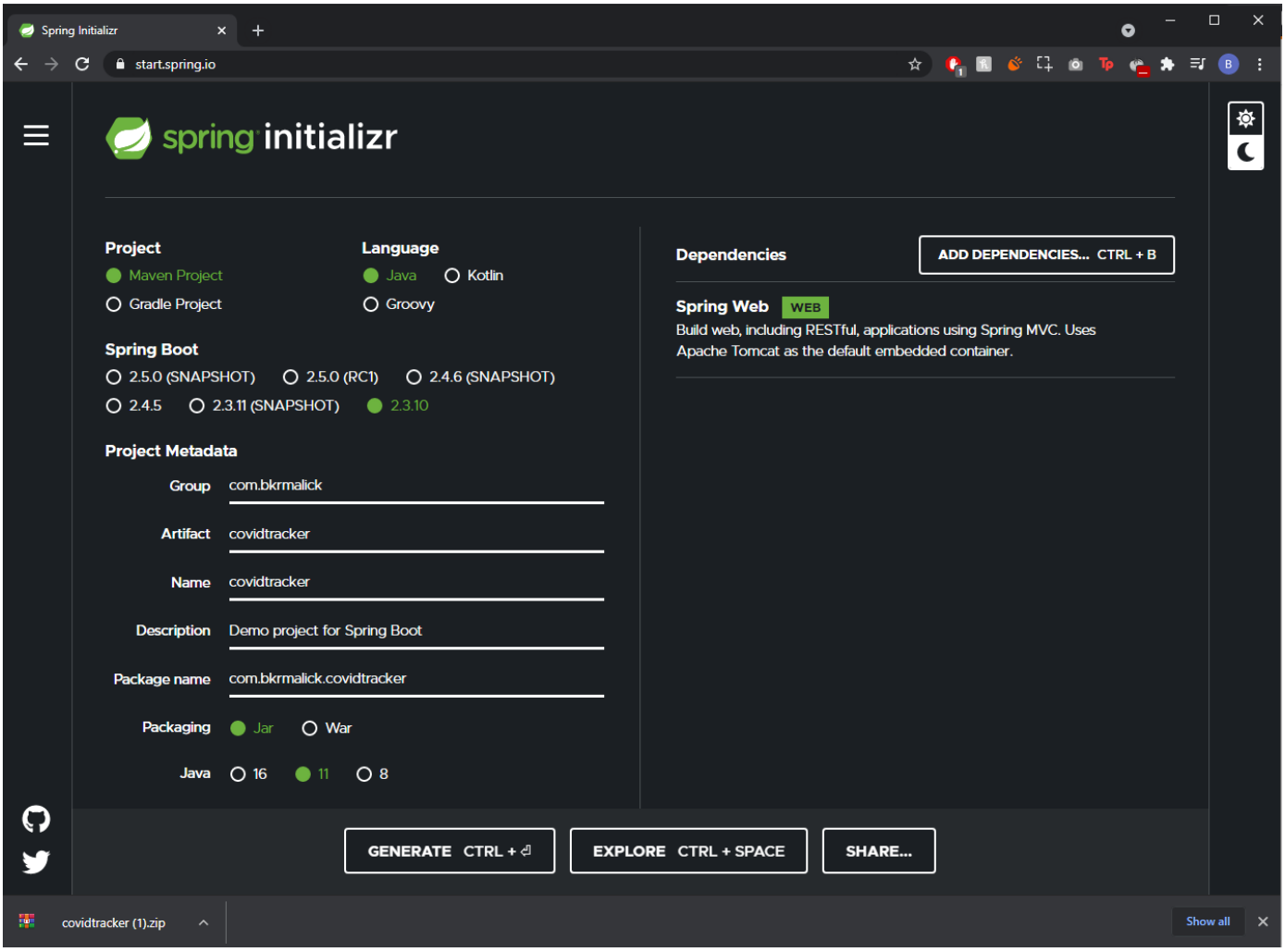


Figure 5.7 Bootstrapping COVIDTracker Backend API

The initial **Spring Web** dependency was essential for creating RESTful APIs using a Model-View-Architecture architecture which is discussed more later in this section. Many more dependencies were added later in the development cycle. After confirming, the bootstrapping tool generated a ZIP file containing the initial source files needed.

The ZIP files were extracted, loaded into IntelliJ[1] and the barebone project was then built and run to ensure everything was working and was confirmed as such by the below message shown in the terminal output of the IDE.

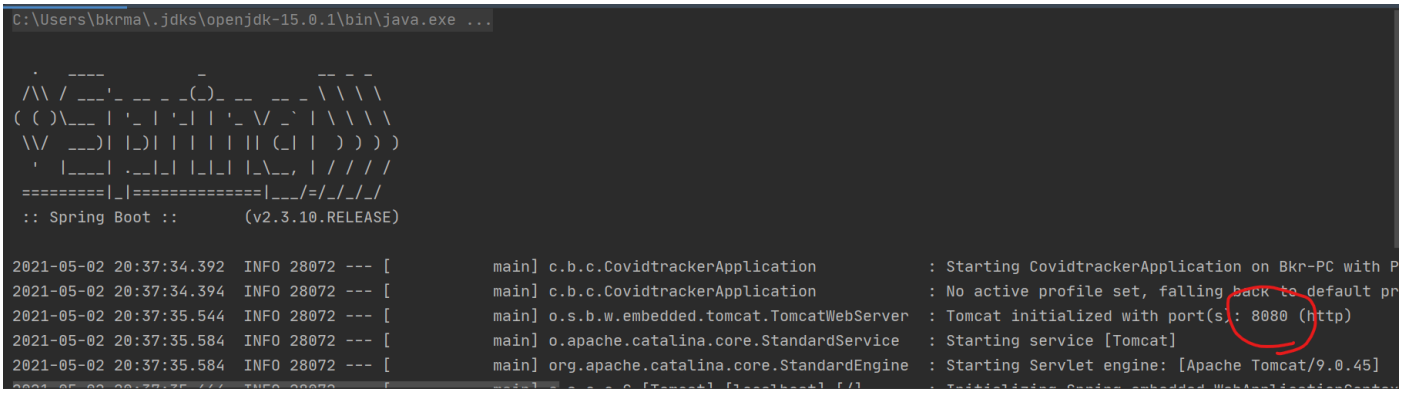


Figure 5.8 API's first run on localhost:8080

There was also an error displayed when browsing onto **localhost:8080** indicating that the server was alive and running but had no data to display. The port number was later changed to 5000 from the default 8080 as that is the port number assumed by Elastic Beanstalk[2] and helped in the deployment of the application.

Now that the environment was ready, it was time to implement the API endpoints and return the required data such that the frontend could operate on it.

The Model-View-Architecture mentioned earlier is a well-founded architecture style that has been preferred in recent times for developing rigid and mature web applications. This was used as the poison of choice for the project's

[1] IntelliJ is an Integrated Development Environment used for programming in Java.
[2] Elastic Beanstalk is a orchestration device provided by Amazon Web Services onto which the COVIDTRACKER application is deployed.

architecture. As evident by its name, it is based on the MVC design pattern and separates the application into different levels where **Models** are used to store information about data objects, **Views** are formats of representation of data to the user, and **Controllers** accept events from users and acts upon them appropriately.

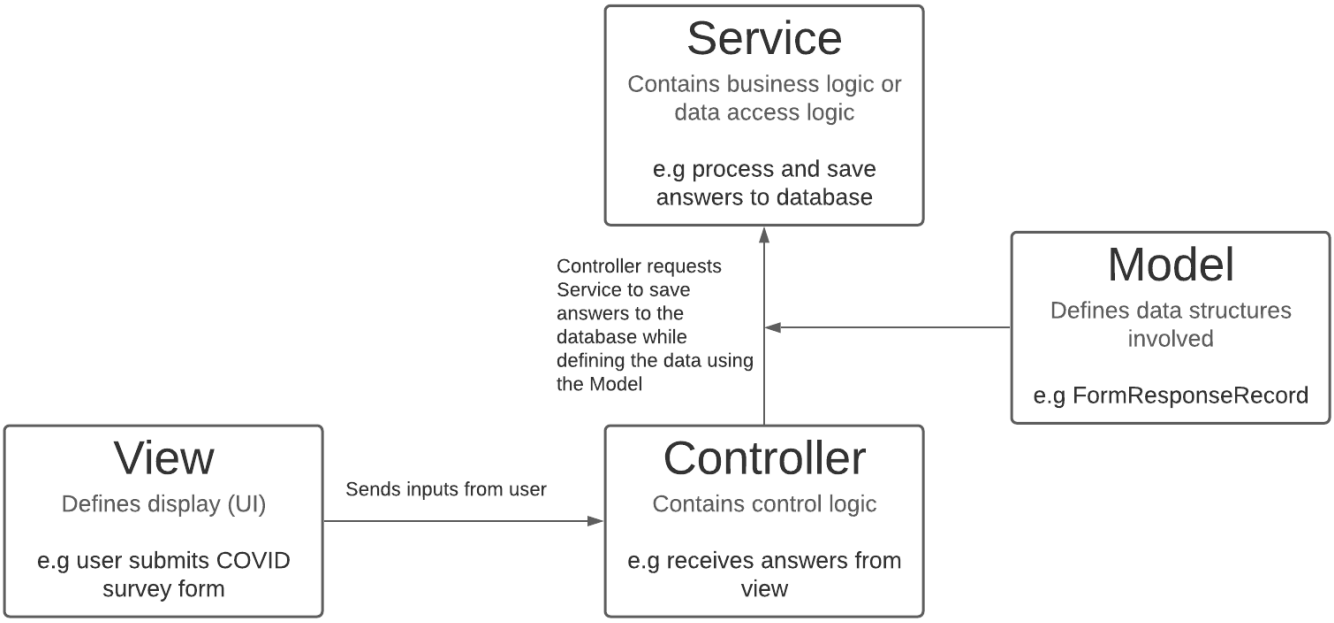


Figure 5.9 MVC components [16]

In the case of COVIDTracker backend API – the view part was going to be managed by the independent frontend application and the backend was only going to be returning a standard unformatted JSON which the frontend application was going to ingest.

The overall structure for the backend API was finalized as shown in Figure 1.1 which is also how it currently stands.

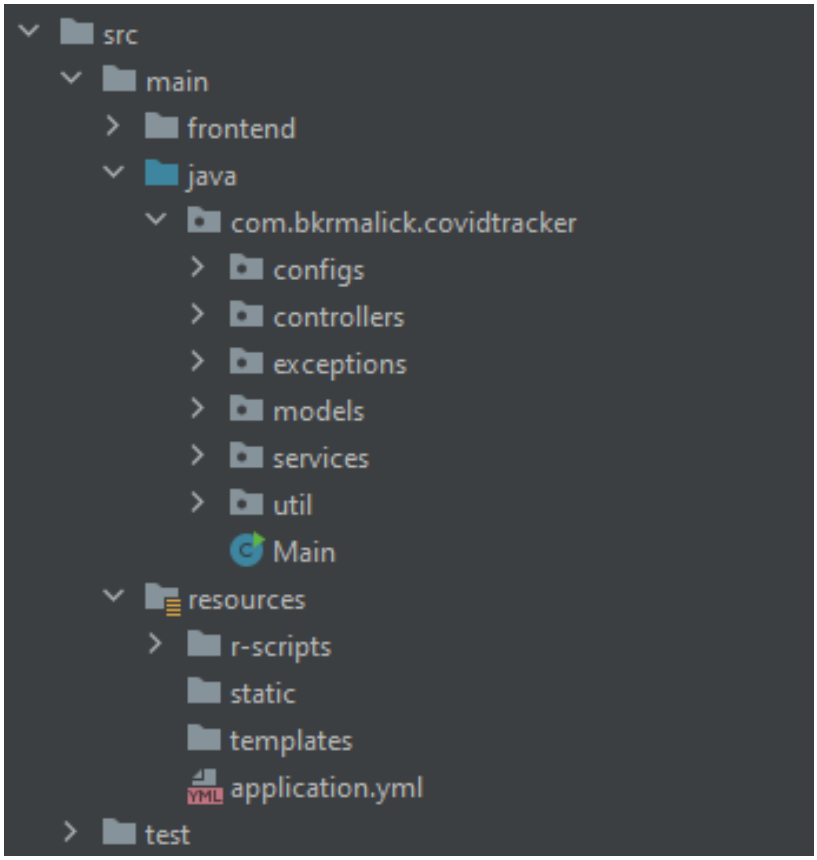


Figure 5.10 Project directory structure

The **resources** folder contains the backend server configuration file `application.yml` and the Main class shown is the entry point for the API and needs to be run to start the HTTP server.

Other than the above, there are three important packages here that are well-worth expanding on and describing what kind of classes they contain. Note that the labels specified in @ symbols are class annotations that the Spring Boot framework provides and are used to give the classes special meaning that the Spring framework can then infer from.

1) The **controllers** package stores three @Controller classes. One for each of the endpoints of the API.

Figure 5.11 Controller to endpoint mapping

Each controller is responsible for responding to GET requests or processing POST-ed data from the user for that endpoint. On receiving a user request, the controllers trigger and communicate with other classes of the application as required to complete the request.

```
@GetMapping("/{date}")
public CasesApiOutput getCasesApiOutputDate(@PathVariable("date")
                                            @DateTimeFormat(pattern = "dd-MM-yyyy")
                                                    LocalDate date)
{
    logger.info("INCOMING REQUEST FOR DATE [" + date + "]");
    return casesProcessingService.produceOutputResponse(date);
}
```

Figure 5.12 A method in CasesController.java

In the method shown in Figure 5.12, the controller method receives the date for which the user wishes to see the COVID case counts, and then this controller method communicates with the other service classes to carry out this request.

2) There are two external APIs that the COVIDTracker backend API uses. One for fetching the up-to-date COVID case counts and one for the post-code to borough translation. To communicate with these two external APIs, it needs to store model classes that describe what format to expect the data in. As such, the **models** package contains classes that model any form of external data objects that are used by the backend server. This includes the forms of data returned by the external APIS as well as corresponding data structures that the backend API itself needs to output at the endpoints.
   In addition to this, the models package also stores @DynamoDBTable classes which describe the structure of the database tables and the data stored in them. Allowing for fetching and storing of records to the appropriate database tables.

3) The main processing and computational work is done in the **services** package which stores @Service classes. These classes are responsible for carrying out the requests they receive from the controller classes by making use of the data structures defined in the models package.

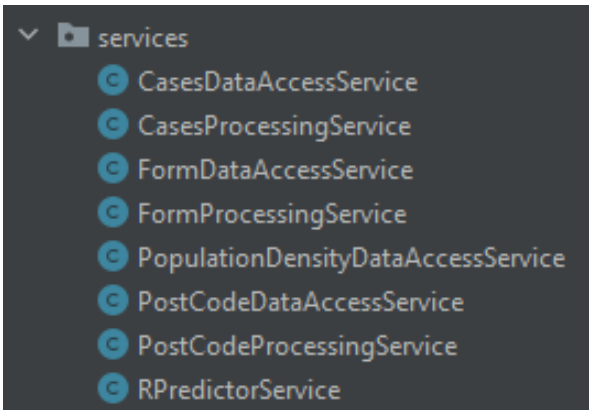


Figure 5.13 The different types of services

As can be seen in Figure 5.13, there are different types of services each with its own set of distinct responsibilities. The data access services are the ones responsible for communicating with external sources of data such as the external APIs and the NoSQL database. The processing services contain the true business rules and logic of what the application is trying to achieve. This separation between services and their purpose allows for easy extensibility and possible future modification.

```
public FormResponseRecord saveAnswers(Map<String, String> answers)
{
    FormResponseRecord item = new FormResponseRecord(answers);

    try
    {
        dynamoDBMapper.save(item);
        logger.info("saved to db response object [" + item + "]");
    } catch (Exception e)
    {
        logger.error("Failed to store response object [" + item + "]", e);
        throw new GeneralUserVisibleException("There was an error storing the response, " +
                "please try again later or contact admin! Response Reference: " +
                item.getId(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
    return item;
}
```

Figure 5.14 Example of a data access service method

Combining all these different types of classes, a call to the backend API for the COVID cases for a future date is roughly equivalent to the below

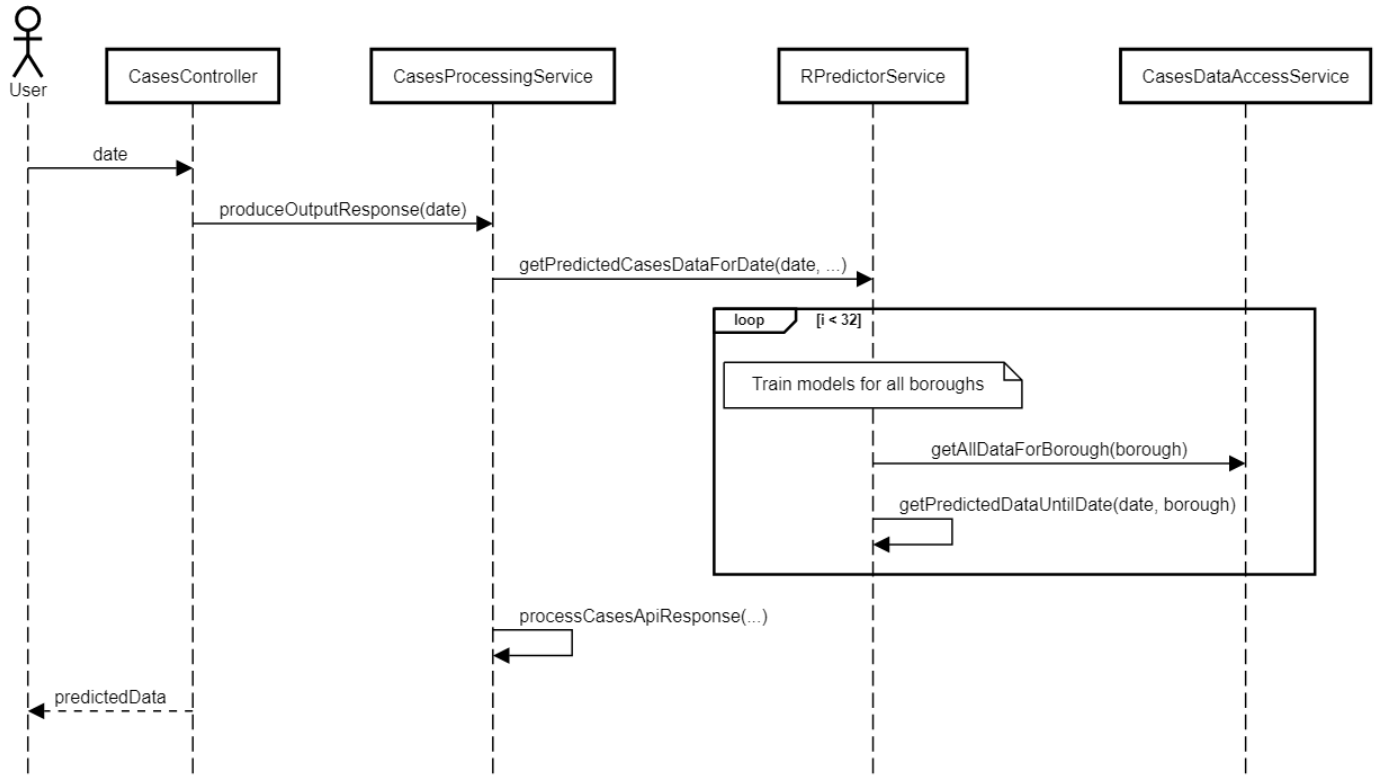


Figure 5.15 Sequence diagram for a call to the API with a future date

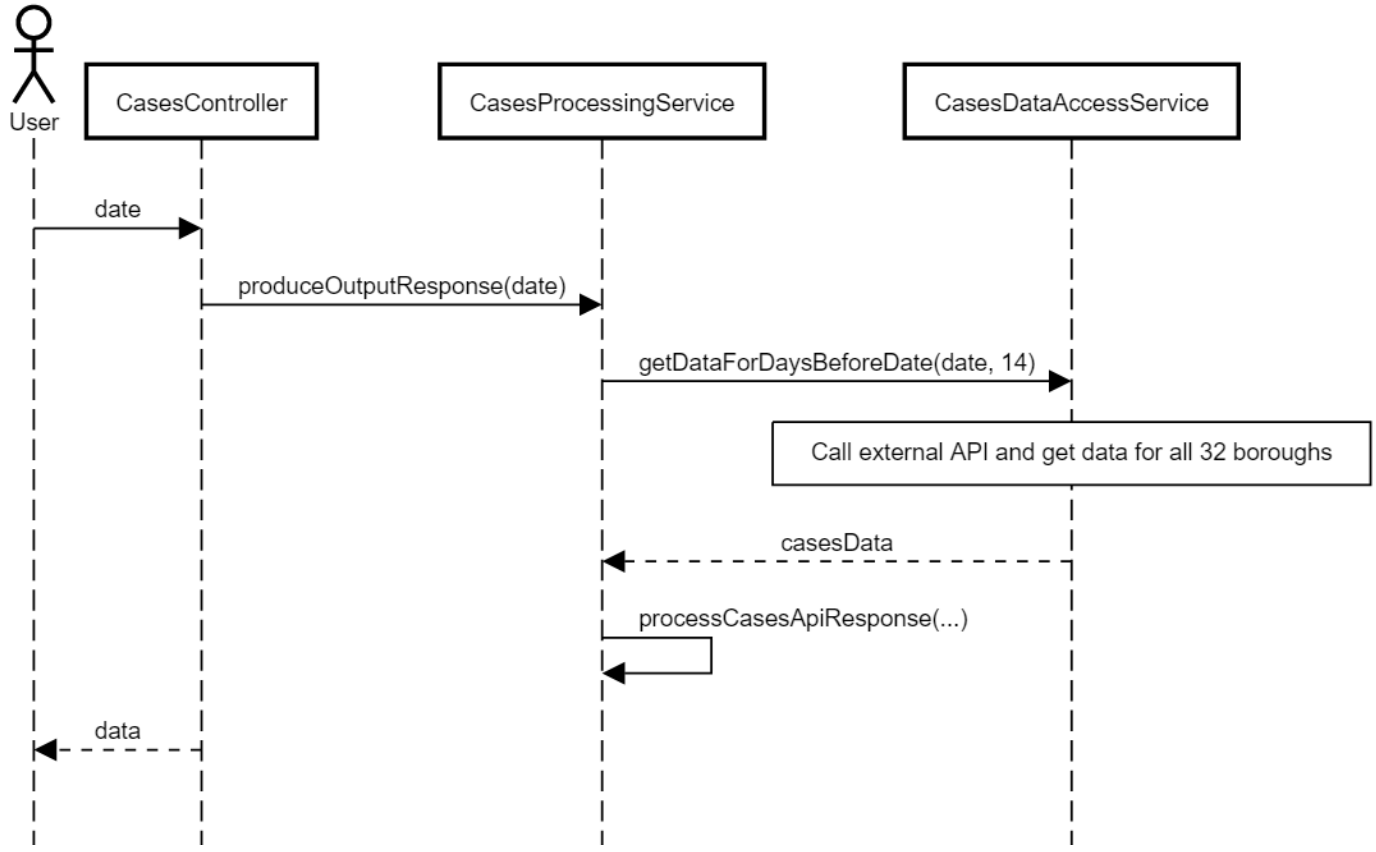


Figure 5.16 Sequence diagram for a call to the API with a historical date

At the end of these interactions, the API returns a JSON response showing each borough's data along with a “mode” attribute representing if this is predicted data or existing data.

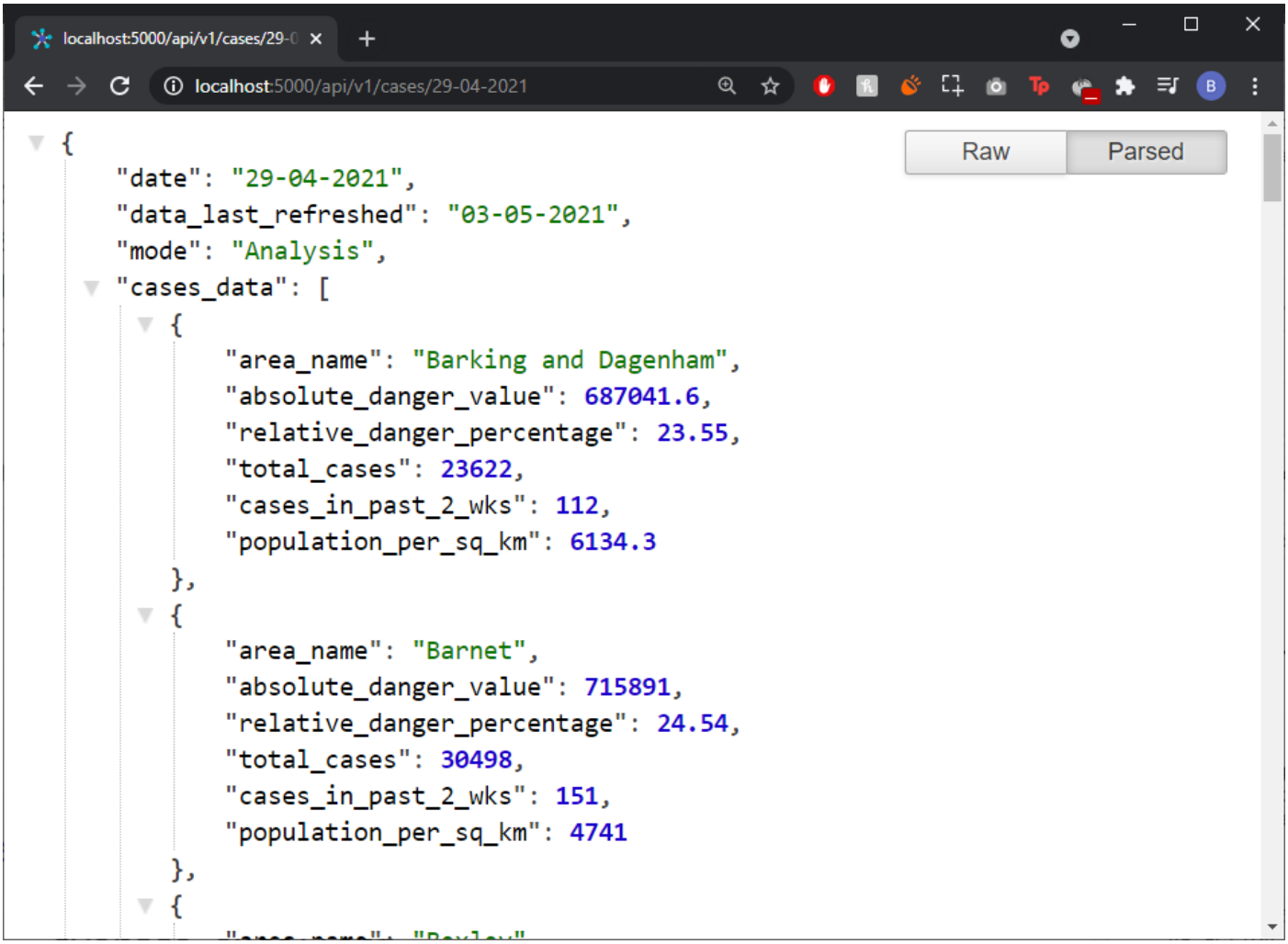


Figure 5.17 API output for a future date

There are other requests that are also possible for the backend API for example- a postcode to borough translation or retrieving the COVID survey questions and saving their responses.

5.3.3 Phase 2 – Prediction service

A major portion of development time was spent in defining and creating the machine learning **RPredictionService** shown in Figure 5.15 and Figure 5.13. This service is responsible for integrating the Java codebase with the R Programming Language environment that is used to carry out the forecasting.

The first step to implementing the service was creating the R script that would take as input existing COVID cases data from the backend API for a specific borough, train a machine learning model[3] on it, and return the predicted case counts. For example, for the Bexley borough, the data was as below.

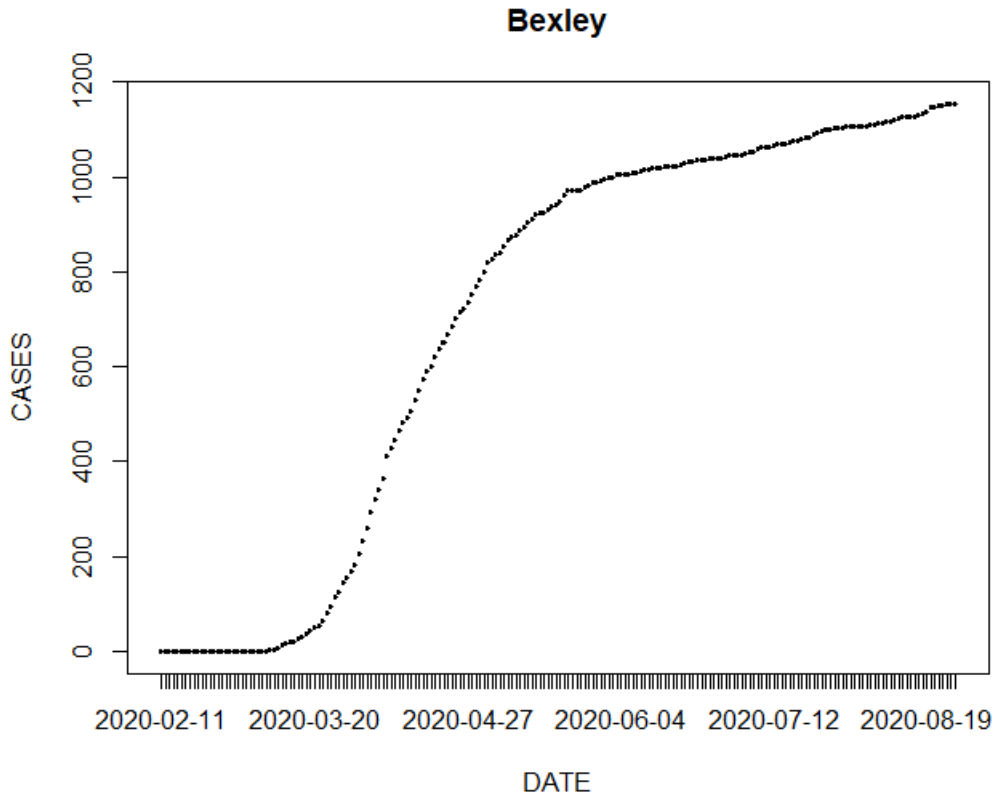


Figure 5.18 Total COVID cases for Bexley borough against date

The script needed to be able to accept data of this format containing a particular borough's number of cases against the dates the data were recorded and be able to interpolate it to forecast what the total case would be on a future date. This would be a multi-step process in the script where it would

1) Accept historical COVID data for a borough and a future date to forecast until.
2) Train a model on the data using some algorithm.
3) Predict and return a new dataset using the trained model containing future dates.

[3] "A **machine learning model** is a file that has been trained to recognize certain types of patterns" [19]. After a model has been generated it can then be used to predict future data.

For step 1, there was some analysis of the data done to check whether it would be worthwhile to train a model using data for all boroughs instead of creating a separate model for each and then also to take each borough's population density as a parameter to account for this.

$$CASES_{borough} \sim DATE + POP.DENSITY_{borough}$$

Here the ~ symbol signifies that the case counts of a borough are dependent on the date and the population density of the borough.

This consideration was dismissed since the data for each borough varies greatly (possibly for reasons beyond the population density, reasons that were not being modeled). Training a model on a dataset for all 32 boroughs together would also have taken a long time which makes for a negative user experience. Therefore, it was decided that a separate model would be trained for each borough and each would only take one input parameter – the date.

$$CASES_{borough} \sim DATE$$

This was then further simplified to the below where the day index is the number of days passed from the minimum date that there was data for (10th of February 2020).

$$CASES_{borough} \sim DAY\_INDEX$$

For step 2, a variety of machine learning algorithms were considered. These standard learning algorithms aim to train and produce a model, but the performance of the resulting model varies greatly depending on the quality of data and the algorithm used so this was an important decision.

1) **Linear Regression** – this is the simplest model possible. It produces a model which extrapolates the best straight line you can draw through the data.

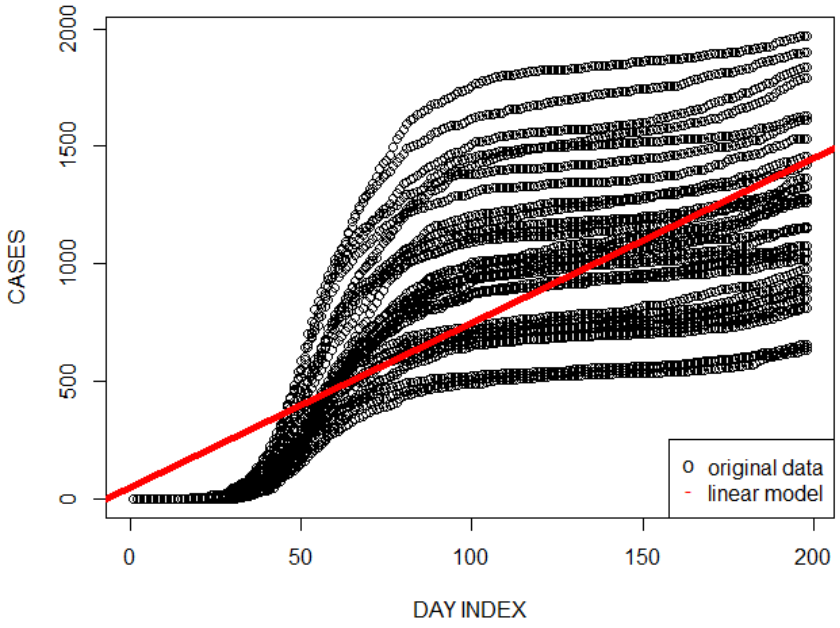


Figure 5.19 Linear model

Each black line in the graph represents a borough and it is evident that this linear model underestimates the case counts for some boroughs and over-estimates for others and is naturally not a good fit.

2) **Support Vector Regression** – this is another learning algorithm but is more flexible and can fit the data better. It first maps the polynomial data points into higher-order space where they become linear.

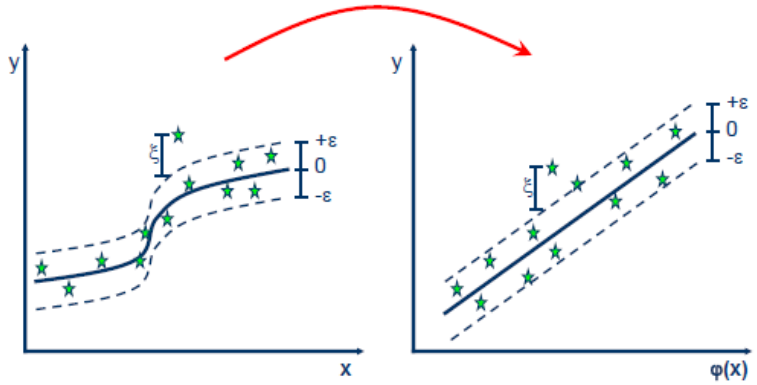


Figure 5.20 SVR algorithm mapping to higher dimension [17]

In the higher-order dimension, the algorithm tries to fit a regression line for the, now 2D, data points in such a way that the errors are within a certain threshold ±ε (shown by the dotted lines in Figure 5.20).

The regression line created in higher order dimensional space can then be mapped back to the original dimension to reveal a more flexible fit to the data points.

To analyse whether this algorithm would be a good fit for the COVIDTRACKER applications purpose, the existing data for a single borough was split into two parts – 70% training and 30% test data.

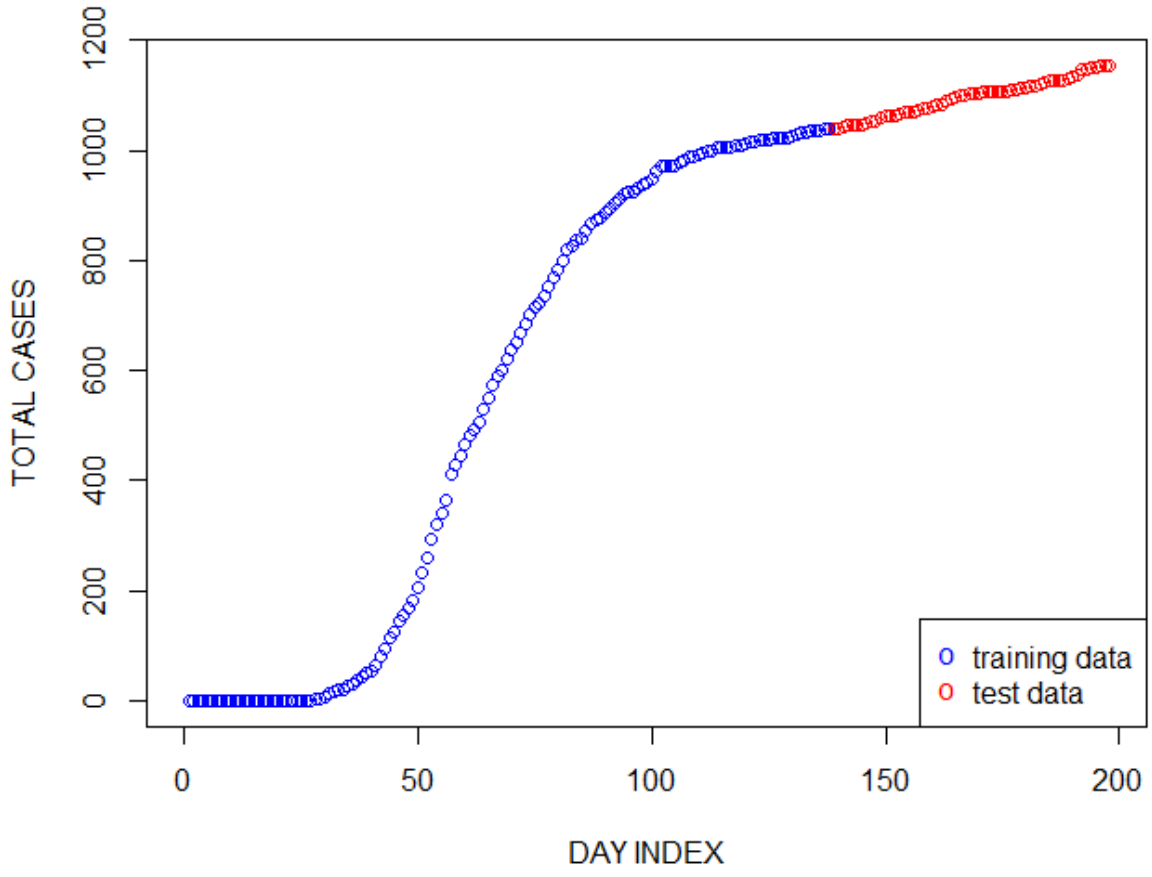


Figure 5.21 Bexley borough data

The SVR algorithm was then used to produce a model on the training data and the model's predictions were checked against the test data which it had not seen before.

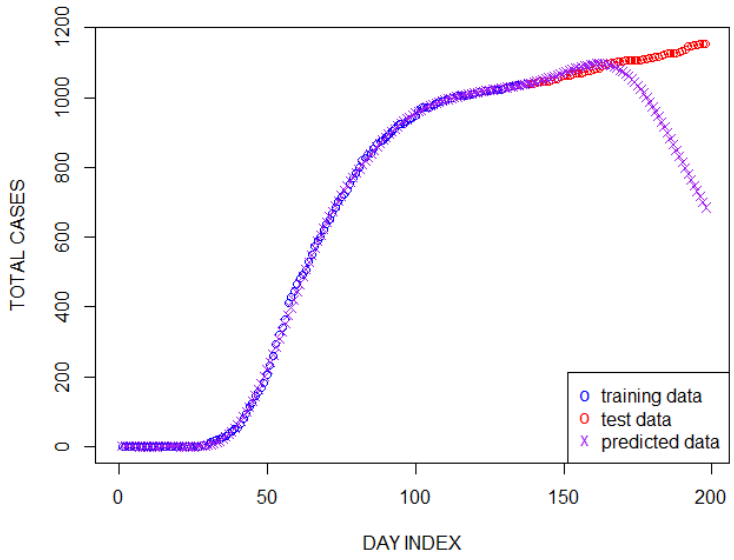


Figure 5.22 SVR model predictions for Bexley

It was found that although SVR performed and fit well for the training data, it failed to model the monotonic (always increasing) nature of the COVID case counts and overreacted by the variations in unseen test data. The former issue could have been circumvented by using an R function “isoreg” but this would only have leveled out the downward slopes to a straight line which is not useful for the SVR curve as it would essentially mean the application predicts the total number of covid cases remains the same for the future.

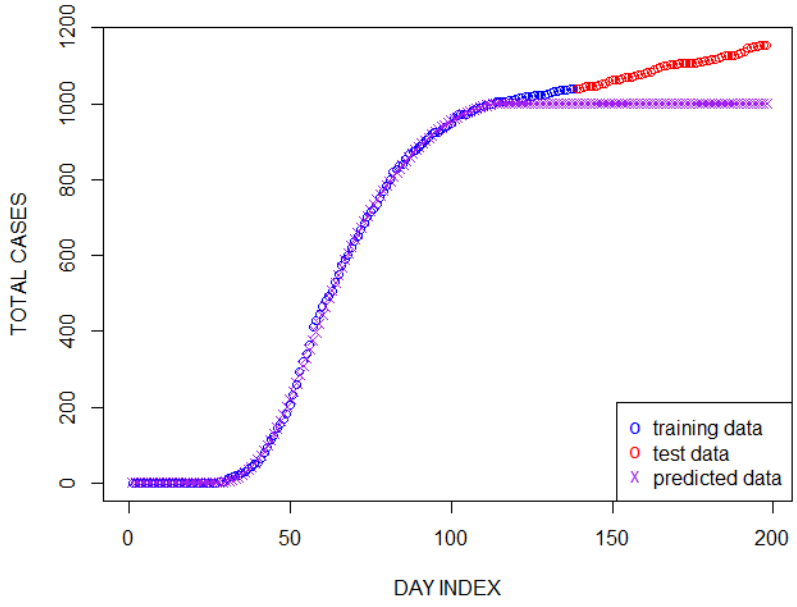


Figure 5.23 isoreg-ed predicted data

The real issue lied with the fact that for this scenario, SVR was produced a highly flexible overfit model which are known to be inaccurate in the machine learning community as they fit too closely to the training data and have poor performance with test data.

3) **Polynomial Regression** – this is a simpler model where a curve is fit to the training data. This was considered as an alternative to the previous algorithms, and it stroked a fair balance between the flexibility and simplicity that the earlier two offered. Polynomial regression is simple but powerful and ensures that the model does not overreact to unseen data.

   To find the correct degree of the polynomial, different polynomial models with varying degrees were created and their performance was assessed.

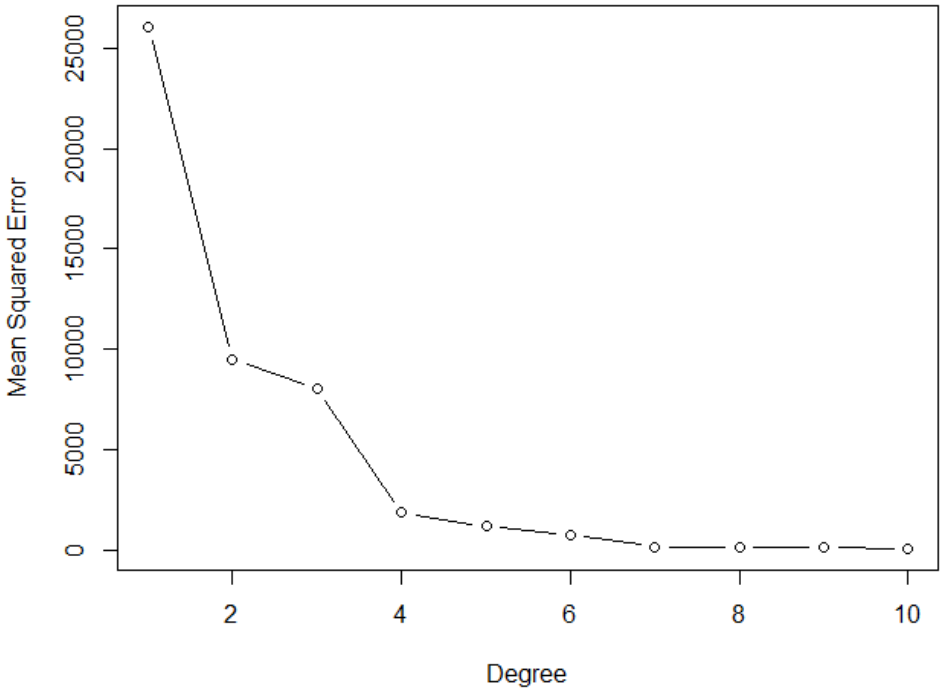


Figure 5.24 Model error against the polynomial degree

The elbow method was then used to identify degree=4 as the most optimal which produced the below model.

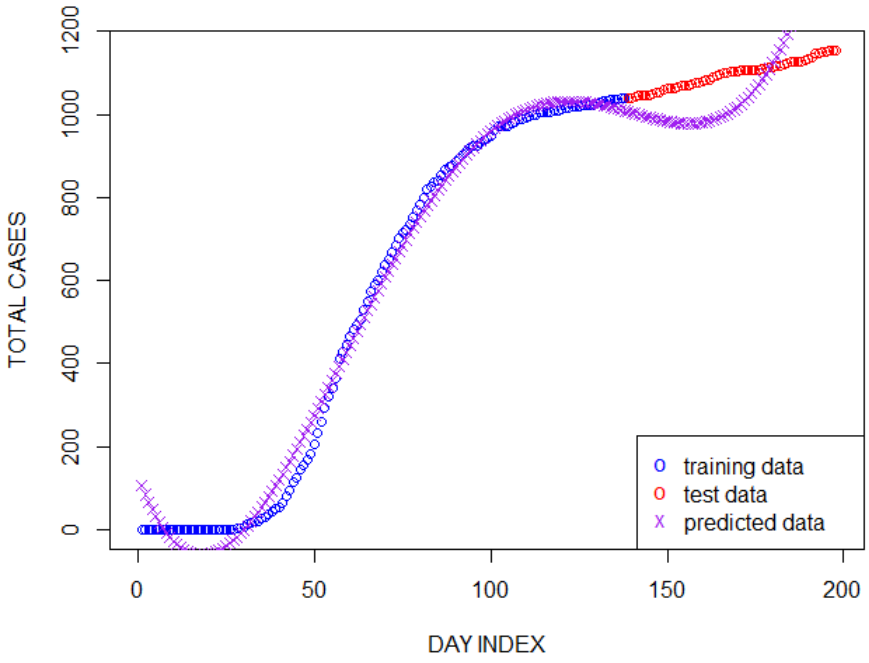


Figure 5.25 Raw polynomial model

And then *isoreg*-ing it again as before.

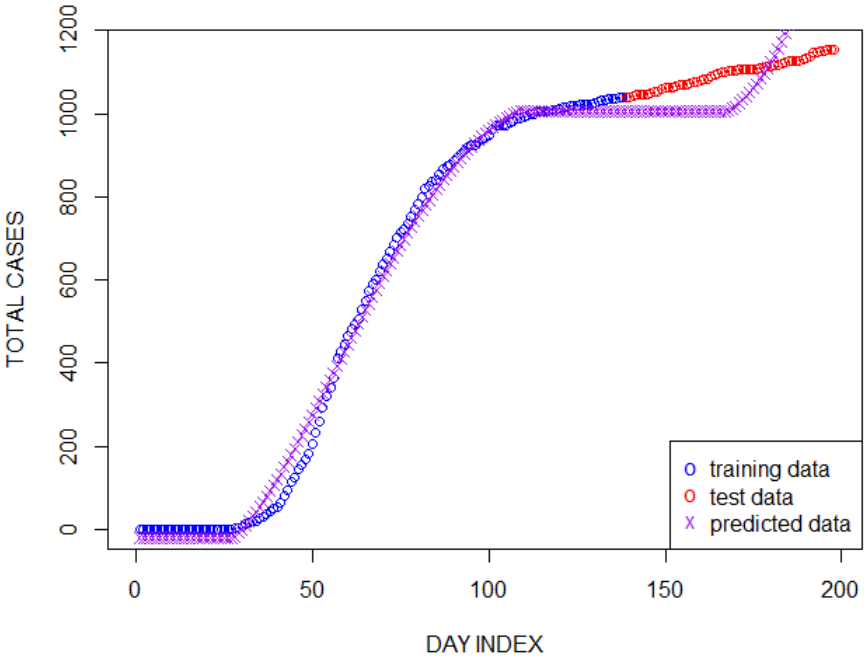


Figure 5.26 Polynomial regression isoreg-ed

As can be seen, this model was the one that showed the most promise. After removing the dips with isoreg, the predicted curve most closely resembled the original one depicted by the data and it did not show signs of overfitting.

Therefore, this polynomial regression learning method with degree=4 as a parameter was chosen as the machine learning model that would be trained and used for predictions in the application. A script named `cases-predictor.R` was created with the required functions to train the model and return its predictions.

After having created the script, the last step was to establish a method to feed it data and launch it from the Java codebase. Multiple Java libraries were available for this purpose, each with its pros and cons but the most widely adopted and easy-to-use out of the box was Renjin.

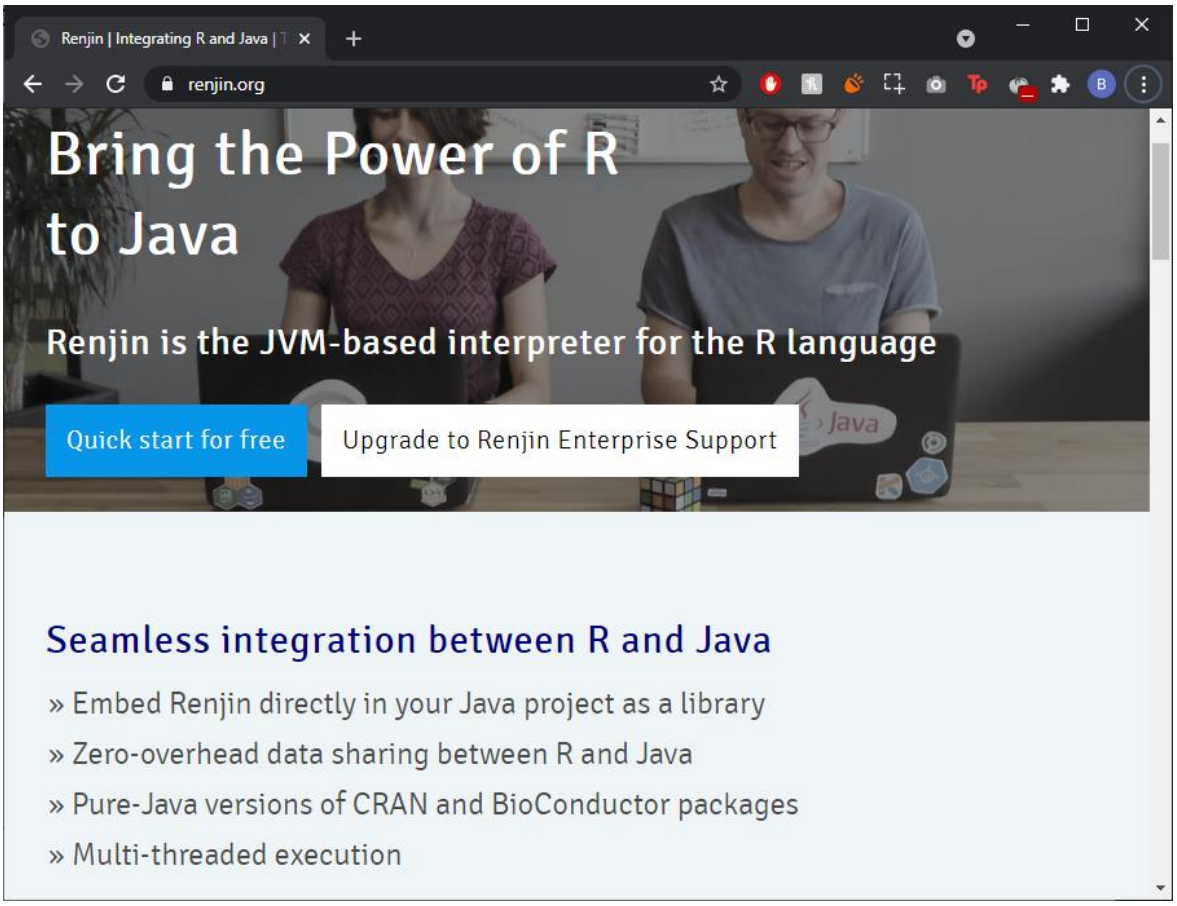


Figure 5.27 Renjin website specifying what it does

Fortunately, it also came bundled with the JVM-based interpreter for the R environment. This meant that it did not require separate R installation on the server conforming to the restriction that Elastic Beanstalk, the deployment service to be used, does not provide direct access for installation of third-party applications therefore Renjin was quickly deemed the most appropriate choice.

It was installed and set up such that the Java class **RPredictorService** would load an R script and feed it data to train it and extract predictions from it and some additional caching logic was put in place on the Java side to avoid having to re-train the model as much as possible and instead use the caches.

An example for this is if currently, a user requests a prediction for a future date say 8th of July – the **RPredictorService** will train a model, fetch the predictions, and cache them in memory. Now any further requests coming in from any other user requesting a prediction for a date before or on the 8th of July, it will use the cached predictions instead of retraining the model. This allows for significantly faster response times for the API and hence betters the application's UX.

5.3.2 Phase 3 – Frontend application setup

The ReactJS frontend code was bootstrapped from a tool provided by Facebook called `create-react-app`. It was executed in the same folder as the backend code to generate a basic React project in a newly created **frontend** directory (as seen in Figure 5.10).

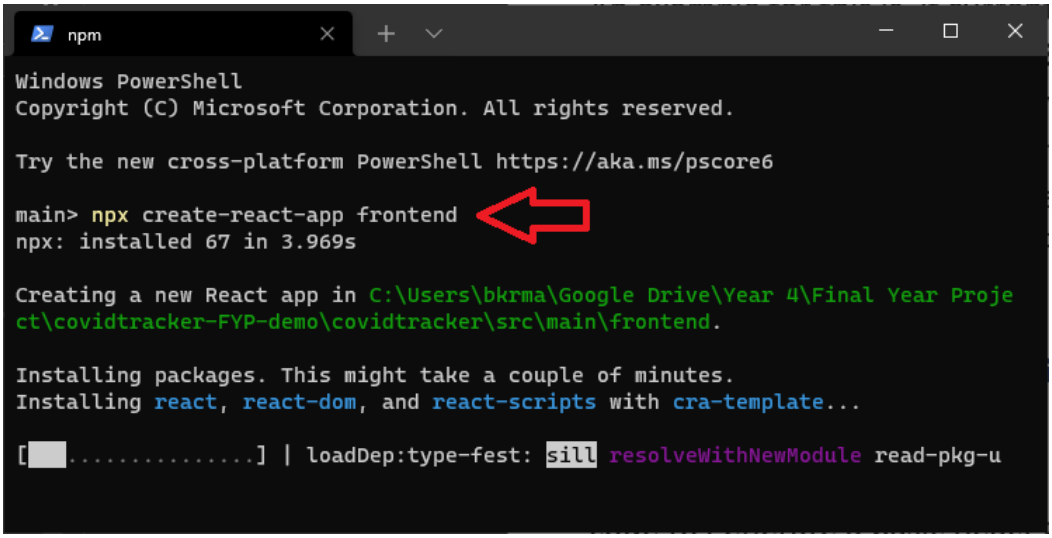


Figure 5.28 Bootstrapping frontend

After the setup finished, the project could then be run locally with the usual `npm run start` node command which opened the application in the browser with hot reloading[4].

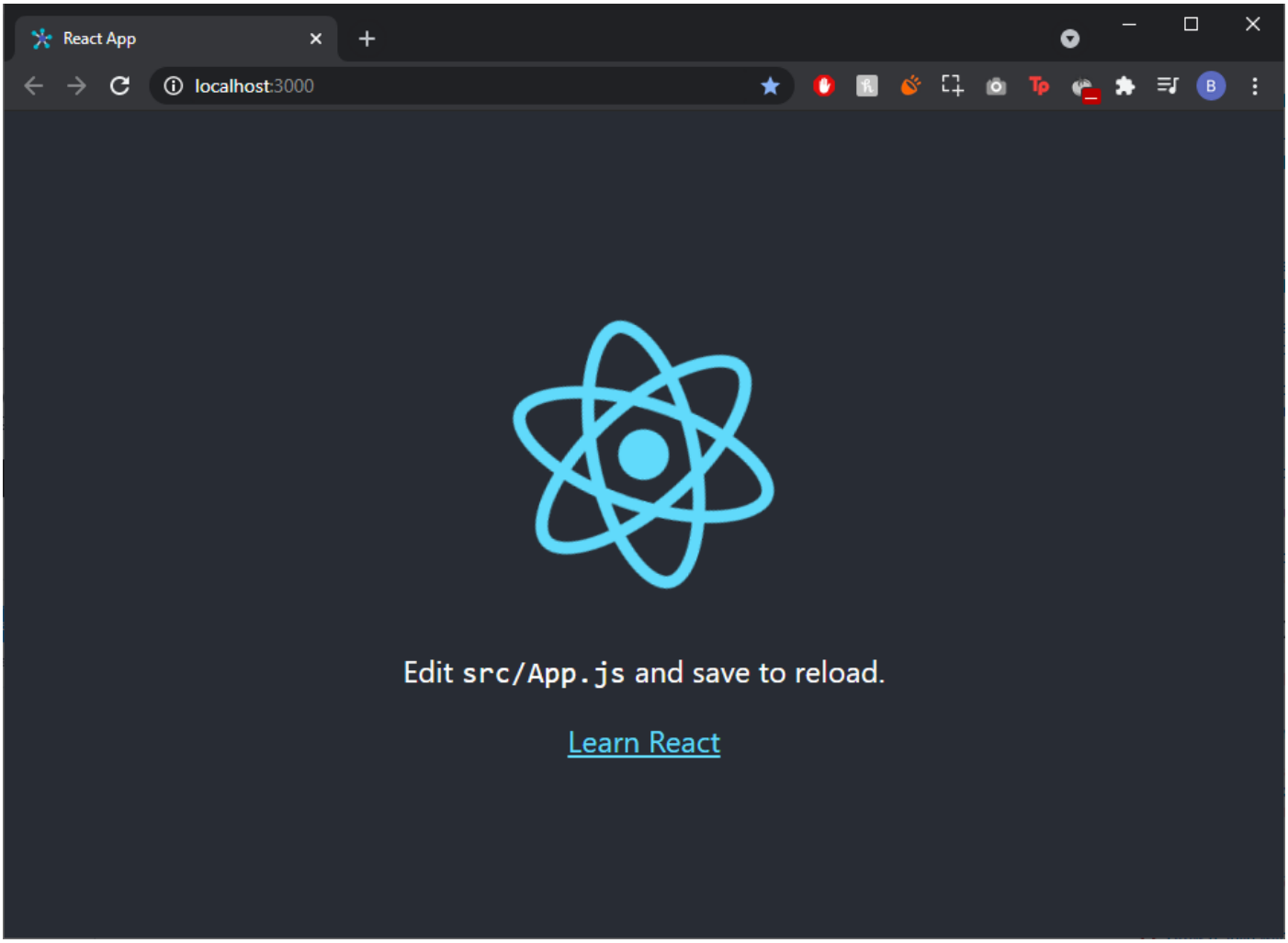


Figure 5.29 Running locally

This simplified the whole process of setting up and configuring a React project into a single command.

After this, the frontend's components were iteratively developed to first produce the tracker page with the appropriate features that were designed in 4.1.2 Tracker Page, and then the surrounding web application was developed to encompass it.

[4] Hot reloading is a feature offered by certain dev environments where the application automatically restarts showing the new changes without having to restart it manually every time allowing for a more streamlined development workflow.

The most notable feature of the tracker page is its usage of a custom implemented Support Vector Graph of London and its boroughs. This was chosen over integrating a Google maps API since Google API was not able to split London based on boroughs and did not support the definition of custom tooltips that were needed, not to mention the service fee Google charges.

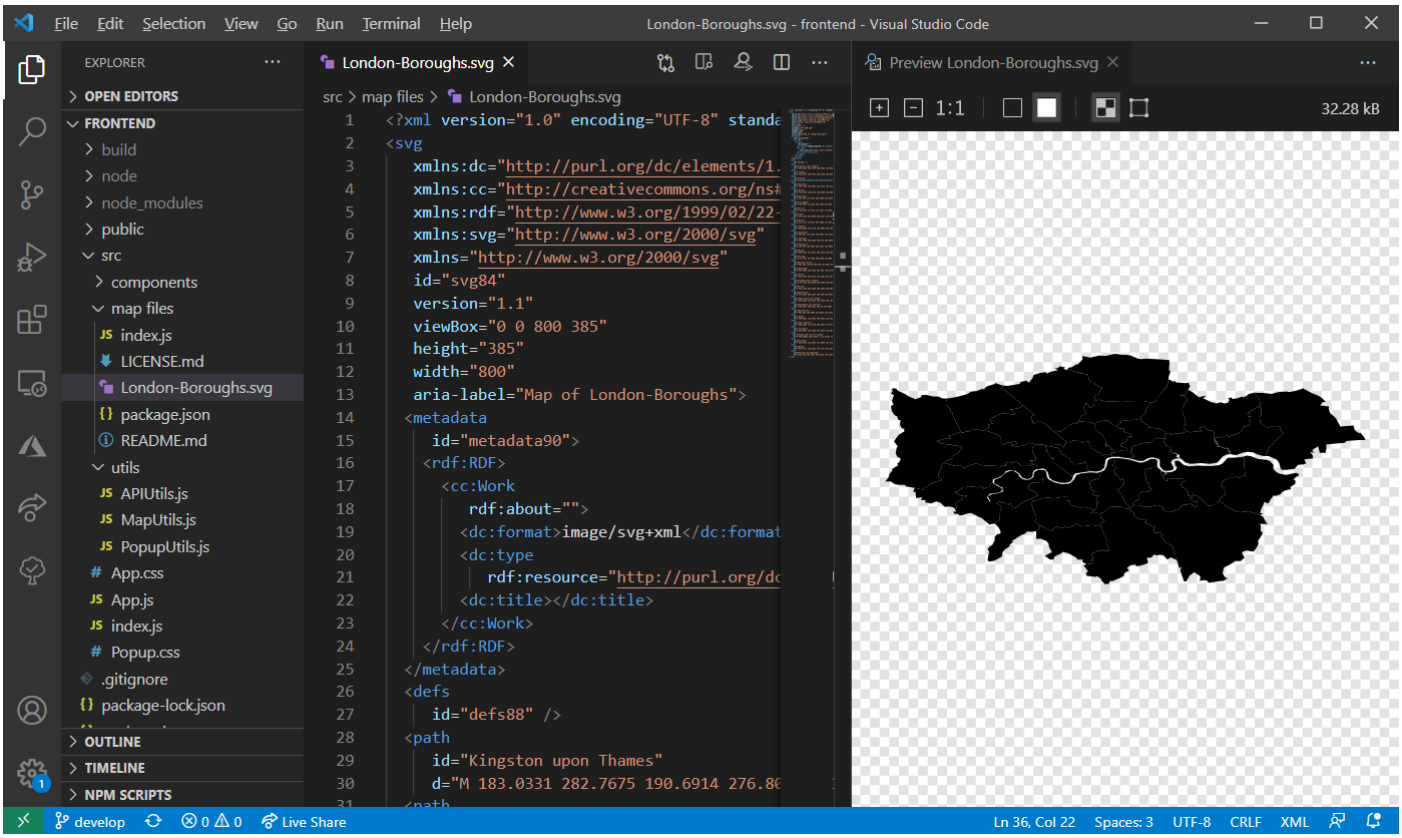


Figure 5.30 SVG HTML definitions vs preview

Furthermore, an SVG map makes use of raw HTML elements to display images which allowed for infinite zoom into the map without loss of any display quality that one would get from a picture.

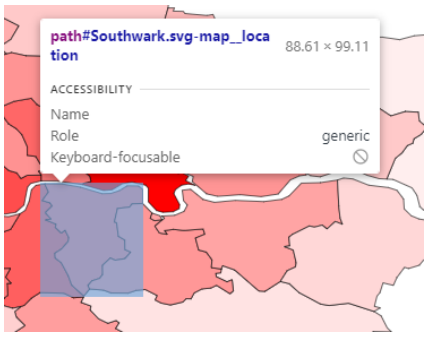


Figure 5.31 Each borough as an individual HTML element

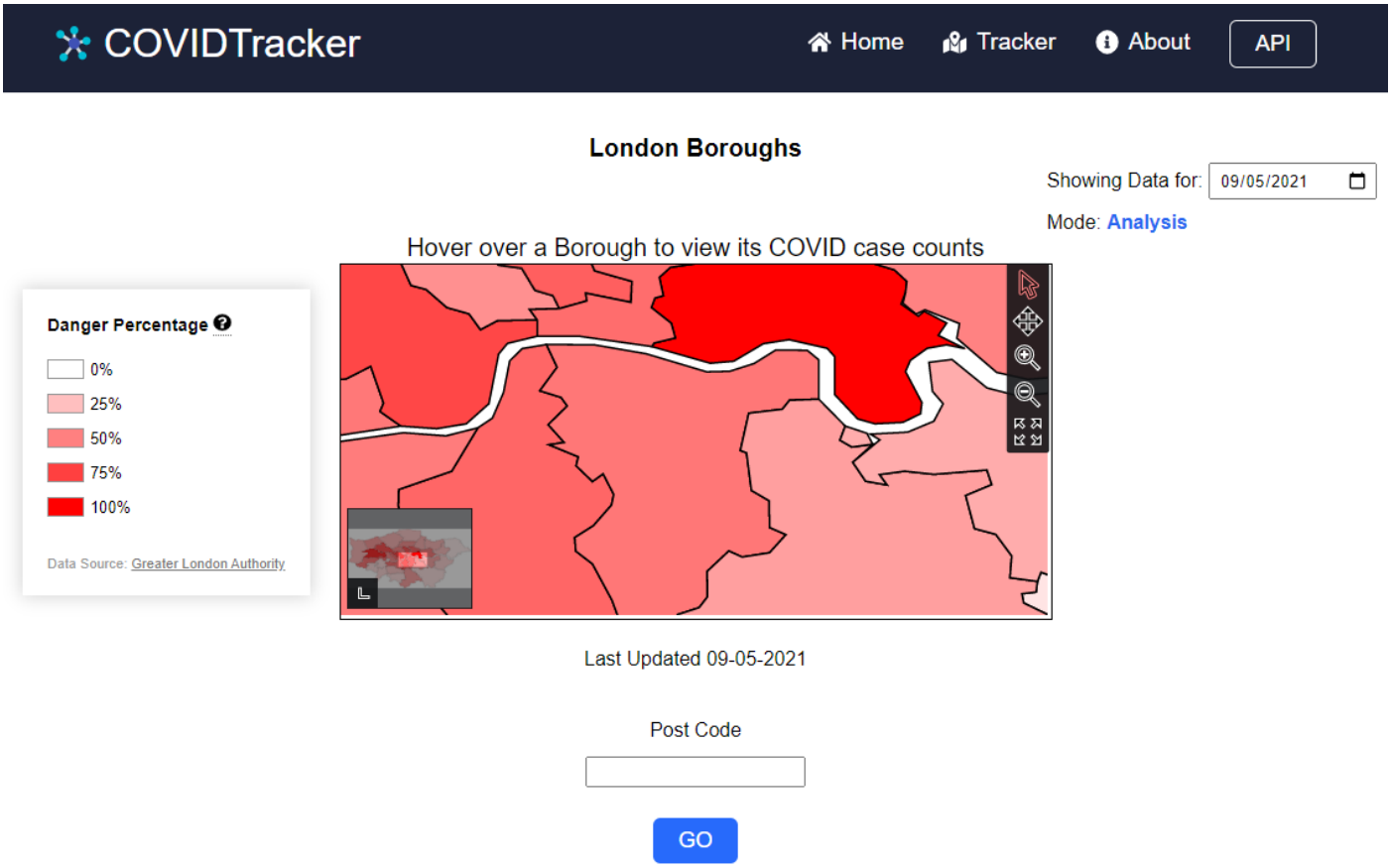


Figure 5.32 Zoomed into the borough of Southwark

5.3.4 Phase 4 – Deployment and Hosting

The full web application was packaged using maven[5] as a single JAR file and is hosted on Elastic Beanstalk which is one of the services Amazon provides as part of its cloud stack.

Maven configuration is defined (in `pom.xml`) such that it also runs a few Node commands to minify and build the frontend JavaScript code as part of its build process. On running, this produced a single fat JAR file including all the dependencies and the frontend application.

Elastic Beanstalk, the tool used for deployment, creates an "environment" for each application which acts as a simplified façade to the other complex cloud services that it orchestrates automatically. It manages multiple services underneath like EC2 (infrastructure), S3 (storage), and CloudWatch (logs).

[5] Maven is a Java build automation tool that is used for managing dependencies for a project and packaging it for deployment.

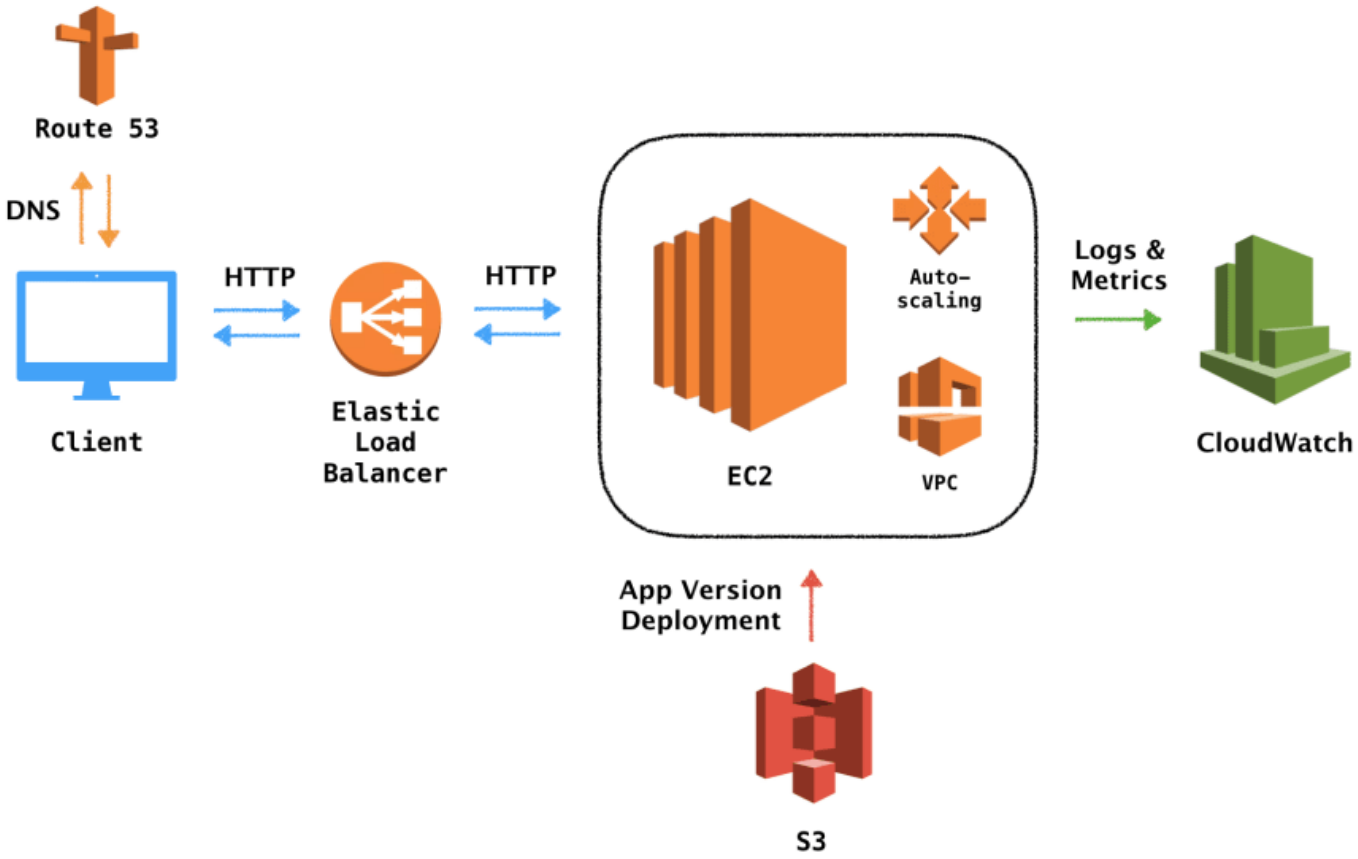


Figure 5.33 ElasticBeanstalk under the “hood” [18]

ELB also adds the ability to add horizontal scaling capability to your program using load balancers as shown in the figure.

Other AWS services being used by the application such as DynamoDB are independent and managed within the application code itself.

To deploy, the JAR produced from the maven build was uploaded onto ElasticBeanstalk which created the applications environment with the required resources and granted it a publicly accessible URL.

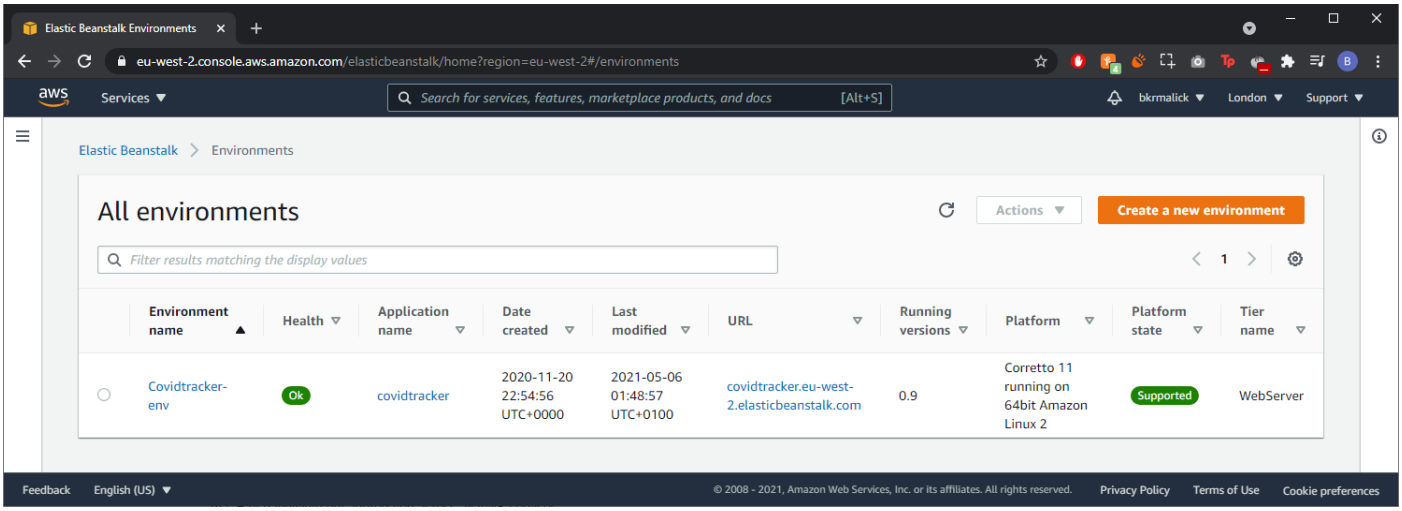


Figure 5.34 COVIDTracker running on ELB

Amazon-provided DNS services under Route53 were then used to reroute the custom-bought domain **covidtracker.london** to the ELB application.
Finally, a redirect was added to the server's listeners to reroute any HTTP requests to HTTPS.

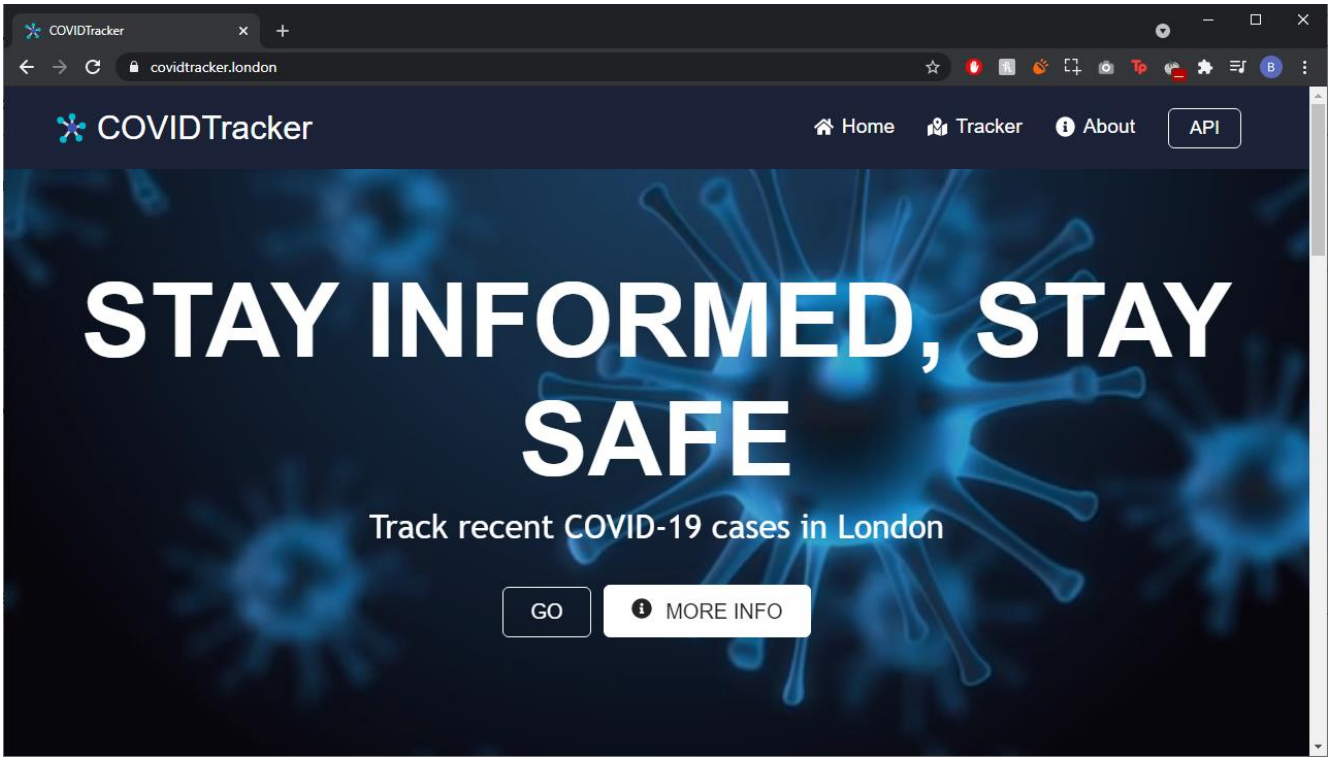


Figure 5.35 Application deployed on https://covidtracker.london

6. Testing

Due to **COVIDTracker** being an interactive and visual web app, most of the testing was manual where the application was run locally and tested end-to-end manually each time a new code change was made to ensure the change had the desired effect. With that being said, automated testing is an essential step in delivering a software product and since this was a single developer project, there needed to be appropriate tests in place to prevent the author from succumbing to common pitfalls and to have the end goals clearly defined.

6.1 Frontend

The frontend code was **Snapshot tested**. In this method of testing, a snapshot of the frontend components is generated and stored which represents their state at a particular point in time. From then onwards when any further code changes are made, newly generated snapshots are compared with the old ones and if they do not match, the tests fail. The developer can then either revert the code change (if the change was unexpected) or can update the snapshots to reflect the new changes.

For COVIDTracker, these tests are stored in the `src/tests` folder

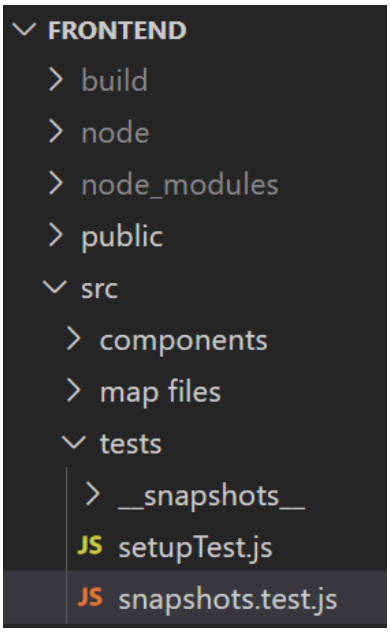


Figure 6.1 Tests directory

These can be run with the `npm test` command.

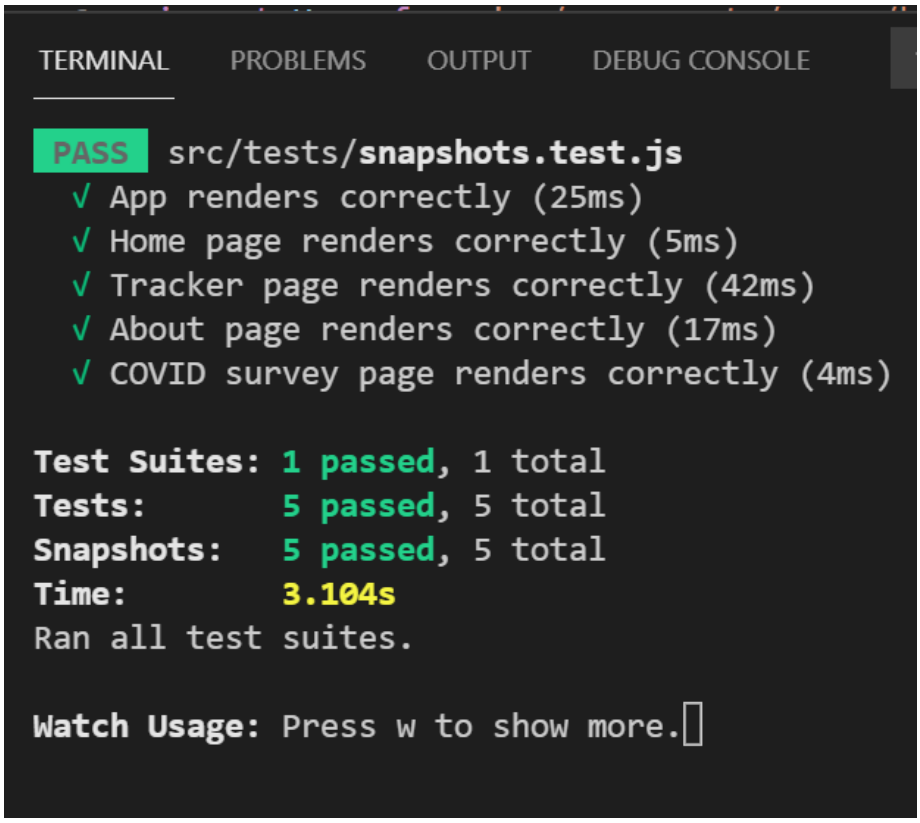


Figure 6.2 All test results passing

If there is a failure, it shows the exact line where the component has unexpectedly been changed.

```
- Snapshot
+ Received

@@ -1,7 +1,7 @@
  <div
-   id="about-page"
+   id="about-page-typo"
  >
    <h1
      className="center title"
    >
      About
```

Figure 6.3 Typo in the div id

```
it("Tracker page renders correctly", () =>
{
  const componentRender = renderer
    .create(<Tracker />)
    .toJSON();
  expect(componentRender).toMatchSnapshot();
});
```

Figure 6.4 Testing Tracker component in one of the snapshot tests

6.2 **Backend**

The backend Java code was unit tested using Junit5. The bootstrapped Spring Boot project already included with it a test class with the special annotation @SpringBootTest which meant that it would load the full server and its context for testing. This allowed for full testing of all the layers.

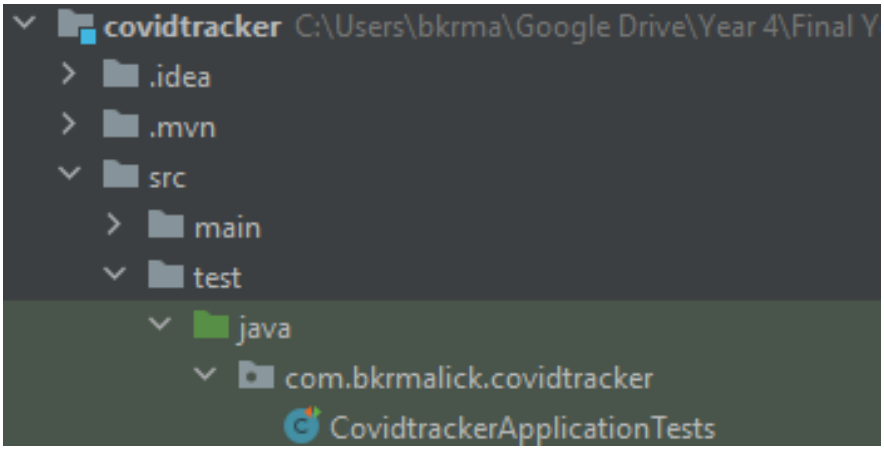


Figure 6.5 Test class

Multiple unit tests were defined in this class to test each of the functionalities. Some actions such as saving responses to the database were deliberately left out as these actions would require complex mocking and the investment it required was rather spent on developing other features. These actions with side effects were rather manually tested to ensure they operated as intended.

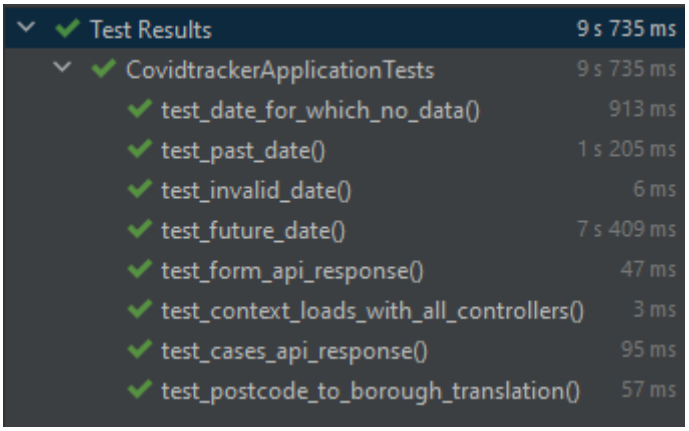


Figure 6.6 All tests passing

```
@Test
public void test_postcode_to_borough_translation() throws Exception
{
    JSONObject response = getBoroughForPostCode("WC1E 7HX"); // Birkbeck's post code
    String borough = response.getString( s: "borough");

    assertThat(borough).isEqualTo("Camden");
}
```

Figure 6.7 Testing postcode-to-borough API

The full list of what was tested for the **backend** and **frontend** is indicated in Appendix A: Feature list and the coverage report can be found in Appendix E: Backend Testing Coverage Report.

7. Discussions and Conclusions

7.1 Reflection

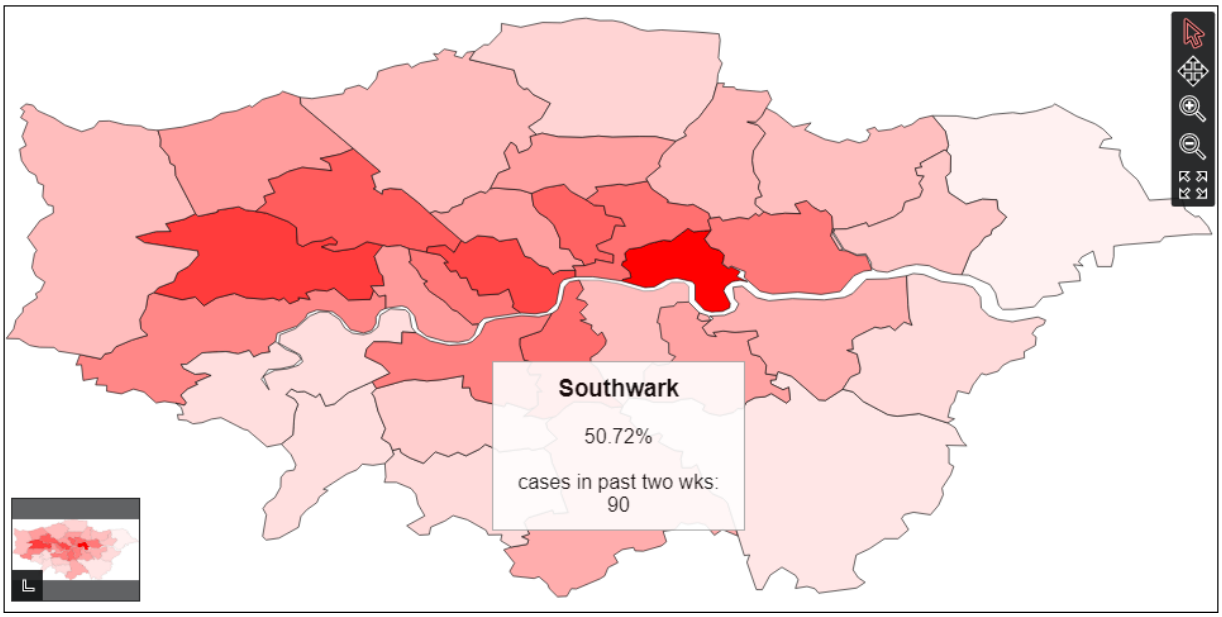


Figure 7.1 Applications main feature

The main purpose of COVIDTracker was to provide users with a method of examining the current COVID scenario and the infection spread across London. It has sought out to address the confused residents which were victims of the split position of government reports on the virus such that a few weeks into the pandemic the residents were informed by the Prime Minister that the “peak was behind us” (30th of April 2020) but then a few weeks later a nation-wide lockdown was imposed due to the worsening situation. This back and forth then happened multiple times including during the 2020 Christmas period where citizens were promised a relaxation period given a stricter lockdown, but certain areas of London were still forced to stay home.

Since it is becoming clear that this virus is here to stay among us for a long time, COVIDTracker aims to allow London residents to become more independent and remain up to date with the always-changing statistics themselves and perhaps even give them more justification and insight on the government’s decisions. It would give them mobility to travel depending on the provided danger metric.

The danger metric is a single straightforward statistic objective of which was to avoid complicating or overburdening the user and presumes no existing knowledge of epidemiology. The users can glance over the metric to gain an estimate of the situation.

The resulting web application is pleasing to the eye and offers a simple and intuitive user interface that flows naturally on a mobile device (see Appendix D: Mobile version screenshots).

The added feature of being able to forecast the COVID case counts, and the danger metrics will also allow users to plan for future social events that no other application has been seen to offer. Therefore, the application is said to have met all its original stated objectives except for one – the API documentation. There is an API button on the header of the website which was supposed to link to the documentation but has been made to reroute to the about page for the time being. There are many tools out there that can generate this documentation automatically, but this piece of work has been left as a suggestion for future improvement.

7.2 Critical evaluation and future work

At many points in the development of the application, an additional nice-to-have feature was brought up, either by a user or the author, but was put aside considering the project's original scope and timeline. This section lists these as suggested future work and underlines any areas where the application can improve.

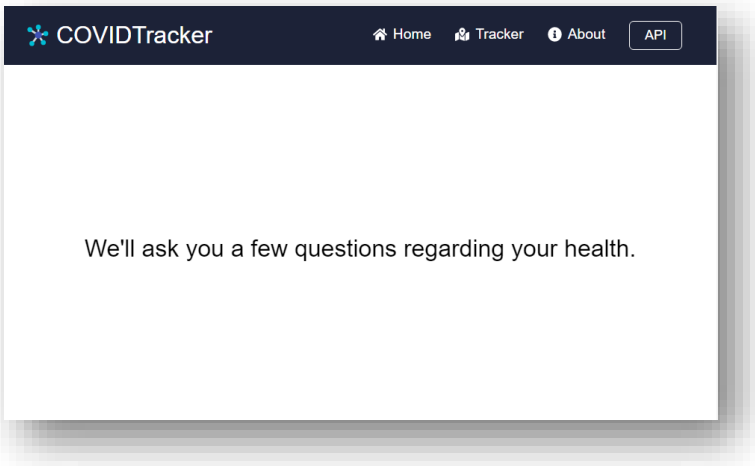


Figure 7.2 COVID survey

The application currently is only collecting and storing the COVID survey responses from the users which asks them for their borough and a few other questions to assess whether the user is actively affected by the virus. A future piece of work could process and integrate this data into the prediction model that is being used for forecasting. This would require some design considerations to account for the overlapping of the two data sources but once the users to the website grow large enough that the application has a strong stream of responses – the application can shift entirely to using this information instead of the external API.

The prediction model in use is not perfect and does not respond well to the flattening of the curve (shown in blue). Its prediction accuracy and performance remained high during the COVID case rises in Summer 2020 and up until early 2021 but has decreased since the flattening.

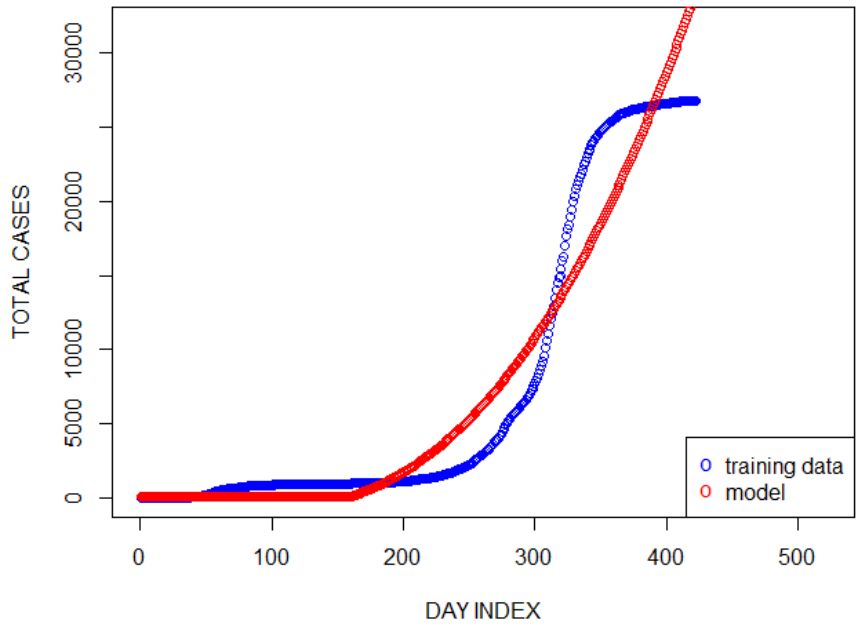


Figure 7.3 Current model in use

This causes an initial large jump in the number of cases when switching from **Analysis** mode to the **Prediction** mode. Therefore, the model needs to be reiterated upon. Its performance and accuracy can be increased by including other factors such as average age of the borough, percentage of vaccinated population, and the government announcements of lifting or imposing restrictions.

To grant it more flexibility, the predictor service could be extracted into a microservice of its own where it can be deployed separately from the backend and frontend where different variations can be tested.

The data shown by the application for each borough is limited to only infection counts. This is because when the application was developed in summer 2020, the timeline for a vaccine was unclear. Nevertheless, the application can be easily expanded to show vaccine rates for each borough or similarly, the number of deaths reported were the cause stated was coronavirus.

References

[1] A. B. Naseer, "Demo of COVIDTracker," [Online]. Available: https://drive.google.com/drive/folders/16QSIIRYD73sGxeWnhUl4iOJ8gCGFDVC7?usp=sharing.

[2] D. B. Taylor, "A Timeline of the Coronavirus Pandemic," 17 March 2021. [Online]. Available: https://www.nytimes.com/article/coronavirus-timeline.html. [Accessed 4 April 2021].

[3] E. Feng, "In Quarantined Wuhan, Hospital Beds For Coronavirus Patients Are Scarce," NPR, 5 February 2020. [Online]. Available: https://www.npr.org/sections/goatsandsoda/2020/02/05/802896668/in-quarantined-wuhan-hospital-beds-for-coronavirus-patients-are-scarce?t=1613192433403&t=1618093055656. [Accessed 13 February 2021].

[4] UK GOV, "Interactive Map | Coronavirus (COVID-19) in the UK," [Online]. Available: https://coronavirus.data.gov.uk/details/interactive-map.

[5] UK GOV, "Deaths | Coronavirus in the UK," [Online]. Available: https://coronavirus.data.gov.uk/details/deaths?areaType=region&areaName=London. [Accessed 15 February 2021].

[6] verywellmind, "The Color Psychology of Blue," [Online]. Available: https://www.verywellmind.com/the-color-psychology-of-blue-2795815#:~:text=Because%20blue%20is%20favored%20by,sign%20of%20stability%20and%20reliability.. [Accessed 18 April 2021].

[7] TypeForm, 11 May 2021. [Online]. Available: https://www.typeform.com/.

[8] N. K. a. M. Khelfaoui, "Population density, a factor in the spread of COVID-19 in Algeria: statistic study," 2020. [Online]. Available: https://doi.org/10.1186/s42269-020-00393-x. [Accessed 1 May 2020].

[9] COVID Symptom Study, "How long does COVID last?," 6 June 2020. [Online]. Available: https://covid.joinzoe.com/post/covid-long-term#:~:text=It's%20commonly%20believed%20that%20COVID,within%20two%20weeks%20or%20so.. [Accessed 25 August 2020].

[10] "Coronavirus (COVID-19) Cases and Vaccinations Dataset," Greater London Authority, [Online]. Available: https://data.london.gov.uk/dataset/coronavirus--covid-19--cases. [Accessed 15 June 2020].

[11] "Land Area and Population Density, Ward and Borough," Greater London Authority, [Online]. Available: https://data.gov.uk/dataset/a76f46f9-c10b-4fe7-82f6-aa928471fcd1/land-area-and-population-density-ward-and-borough. [Accessed 15 June 2020].

[12] ParkMyCloud, "AWS vs Azure vs Google Cloud Market Share," [Online]. Available: https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/. [Accessed 7 May 2021].

[13] "3-Tier Architecture: A Complete Overview," [Online]. Available: https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/. [Accessed 11 April 2021].

[14] "Understanding Software and System Architecture," [Online]. Available: https://thenewstack.io/primer-understanding-software-and-system-architecture/. [Accessed 16 April 2021].

[15] "SSR vs CSR: Which method works best?," [Online]. Available: https://www.growth-rocket.com/blog/a-closer-look-at-client-side-server-side-rendering/. [Accessed 16 April 2021].

[16] Mozilla, "MVC," [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/MVC. [Accessed 2 May 2021].

[17] D. S. Sayad, "Support Vector Regression," [Online]. Available: https://www.saedsayad.com/support_vector_machine_reg.htm. [Accessed 5 May 2021].

[18] Dev.to, "Deploying an HTTP API on AWS using Elastic Beanstalk," [Online]. Available: https://dev.to/frosnerd/deploying-an-http-api-on-aws-using-elastic-beanstalk-5dh7. [Accessed 7 May 2021].

[19] Microsoft, "What is a machine learning model?," [Online]. Available: https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model. [Accessed 11 May 2021].

Appendixes

Appendix A: **Feature list**

Feature	Achieved?	Unit Tested?
Interactive London Borough Map	✓	✓
User able to enter a postcode to view statistics for their borough	✓	✓
Each borough shows “danger level”	✓	✓
Forecasting of future COVID case counts	✓	✓
A responsive web application able to run on any modern smartphone	✓	✓
COVID survey form in the style of a “TypeForm” allowing users to contribute to the dataset	✓	✓
Saving user responses to a NoSQL database	✓	✘ (Manually tested)
Direct API available to use for other developers	✓	✘ (External developer responsible for their own testing)
API’s Swagger documentation	✘	✘

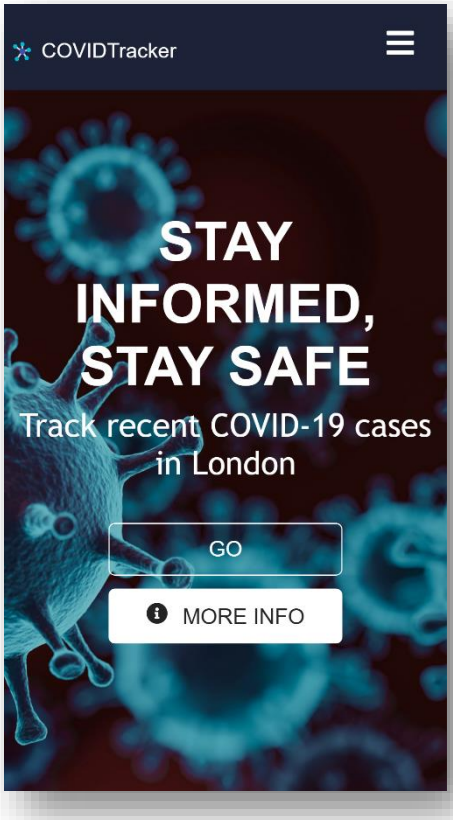
Appendix B: **Tools and technologies**

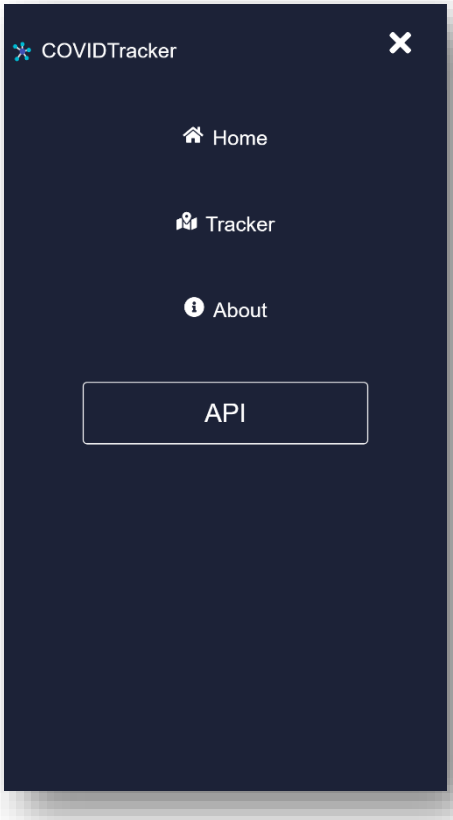
Clickable Link	Type	Used For
ReactJS	Framework	Underlying JavaScript framework for the frontend
Spring Boot	Framework	Underlying Java framework for the backend
DynamoDB	NoSQL Database	Storing survey questions, survey responses, and borough population densities
Renjin	Library	R integration to Java
Amazon Web Services	Cloud Provider	Hosting, DNS, Database, etc.

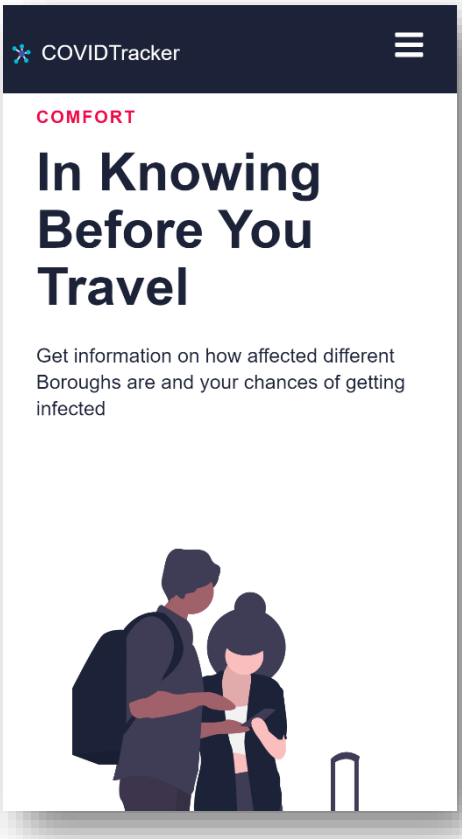
Appendix C: **Sources and libraries used for frontend**

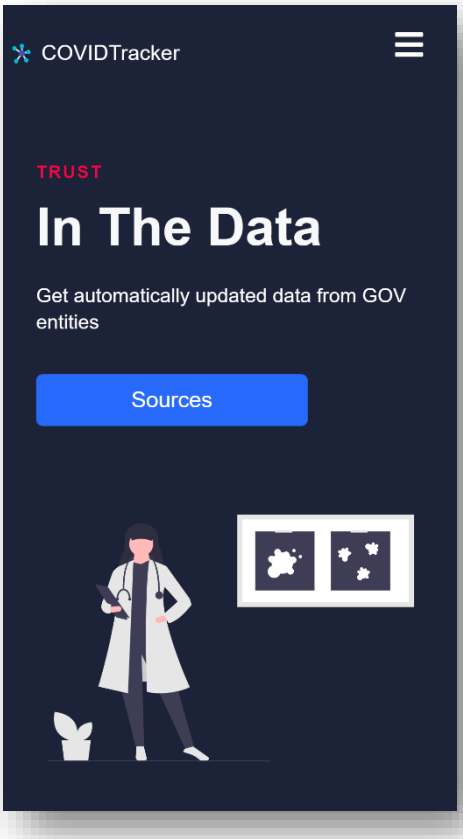
Source / Library Link	Type	Used For
VEGA Datasets	File	London boroughs raw SVG map
VictorCazanave svg-maps	Library	Displaying SVG map
react-svg-pan-zoom	Library	Interactive tools for the map viewer
Briancodex react-website-v2	Code Snippets	Starter code for components design
Brian Design	Tutorial	Designing and creating components
TileMill Advanced Legends	Code Snippet	Starter code for the map legend

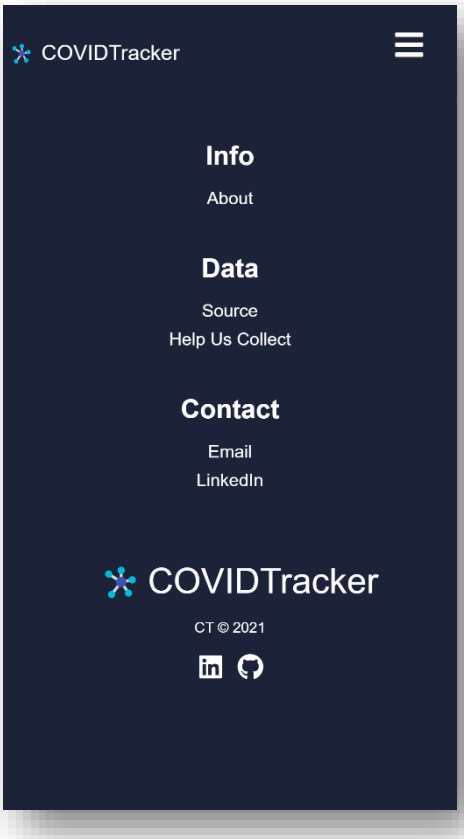
Appendix D: **Mobile version screenshots**

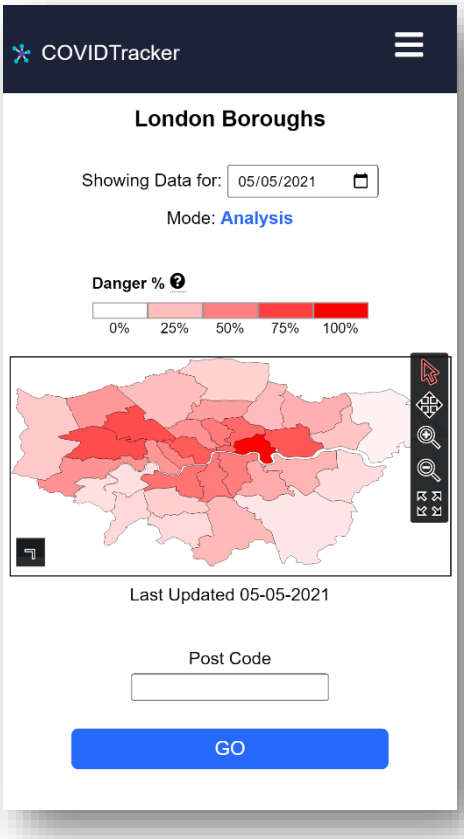


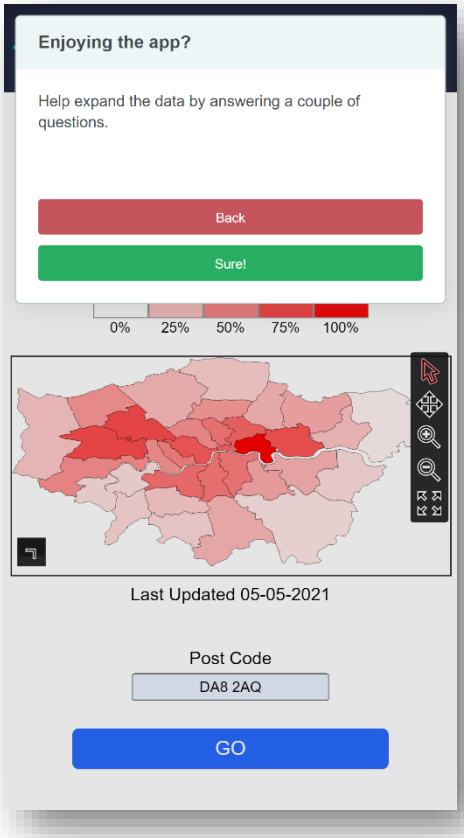


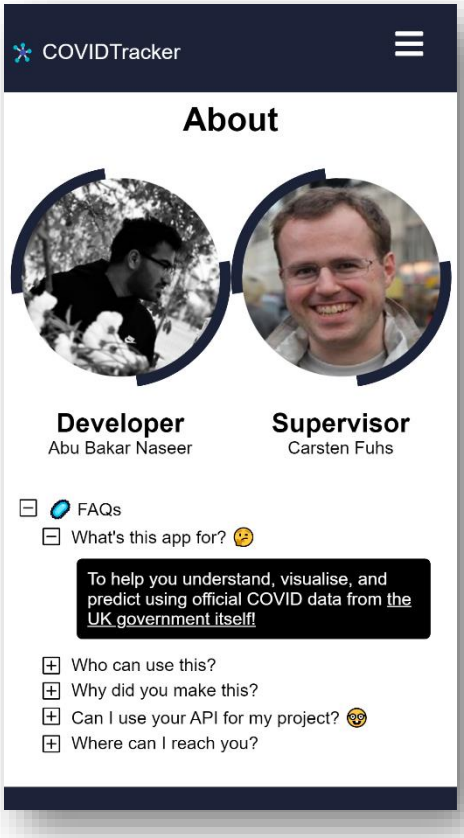


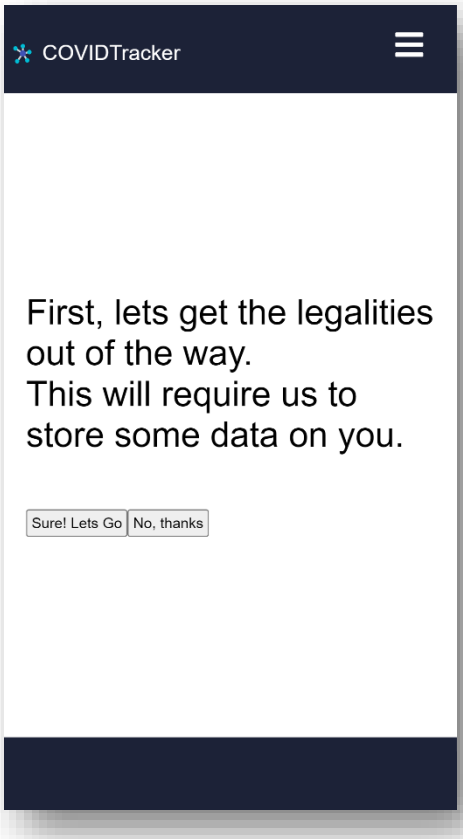


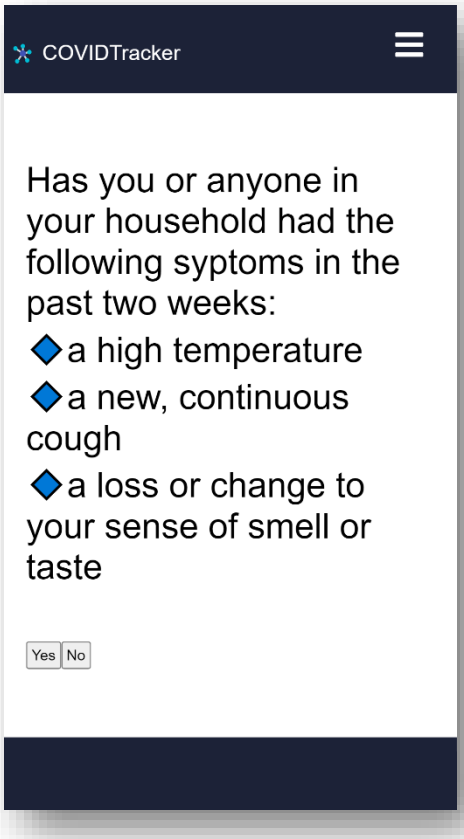

COVIDTracker
Info
About
Data
Source
Help Us Collect
Contact
Email
LinkedIn
COVIDTracker
CT © 2021


COVIDTracker
London Boroughs
Showing Data for: 05/05/2021
Mode: Analysis
Danger %
0%
25%
50%
75%
100%
Last Updated 05-05-2021
Post Code
GO


Enjoying the app?
Help expand the data by answering a couple of questions.
Back
Sure!
0%
25%
50%
75%
100%
Last Updated 05-05-2021
Post Code
DA8 2AQ
GO


COVIDTracker
About
Developer
Abu Bakar Naseer
Supervisor
Carsten Fuhs
FAQs
What's this app for?
To help you understand, visualise, and predict using official COVID data from the UK government itself!
Who can use this?
Why did you make this?
Can I use your API for my project?
Where can I reach you?





Appendix E: **Backend Testing Coverage Report**

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	96.9% (31/ 32)	74.3% (139/ 187)	75.8% (400/ 528)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.bkrmalick.covidtracker	100% (1/ 1)	50% (1/ 2)	33.3% (1/ 3)
com.bkrmalick.covidtracker.configs	100% (6/ 6)	88.9% (16/ 18)	85.7% (24/ 28)
com.bkrmalick.covidtracker.controllers	100% (3/ 3)	80% (8/ 10)	88.9% (16/ 18)
com.bkrmalick.covidtracker.exceptions	100% (3/ 3)	80% (8/ 10)	87% (20/ 23)
com.bkrmalick.covidtracker.models.cases_api.input	100% (2/ 2)	85% (17/ 20)	92.5% (37/ 40)
com.bkrmalick.covidtracker.models.cases_api.output	100% (2/ 2)	60.9% (14/ 23)	60% (27/ 45)
com.bkrmalick.covidtracker.models.dynamo_db	66.7% (2/ 3)	59.2% (29/ 49)	49.5% (48/ 97)
com.bkrmalick.covidtracker.models.form_api	100% (1/ 1)	50% (2/ 4)	50% (4/ 8)
com.bkrmalick.covidtracker.models.postcode_api.input	100% (1/ 1)	66.7% (4/ 6)	58.3% (7/ 12)
com.bkrmalick.covidtracker.models.postcode_api.output	100% (1/ 1)	50% (2/ 4)	50% (4/ 8)
com.bkrmalick.covidtracker.services	100% (8/ 8)	94.9% (37/ 39)	86.5% (211/ 244)
com.bkrmalick.covidtracker.util	100% (1/ 1)	50% (1/ 2)	50% (1/ 2)

generated on 2021-05-09 02:13

Appendix F: **Ethics Department Approval for User Data Collection**

School of Business, Economics and Informatics

Department of Computer Science and Information Systems

Ethical Review Form

Name(s) of applicant	Abu Bakar Naseer
Job title	BSc Student
Funding source	N/A
Project Title	London COVID Infection Risk Calculator and Spread Visualiser (Web Application)

Attachments:
Indicate the attachments enclosed with this form (please tick boxes):

Information sheet: ☐ Consent Form: ☐ Questionnaire: ☐ Data Management Plan ☐

Description and rationale or proposed project

This web application will analyse number of reported infections (obtained from NHS APIs) in London Boroughs and advise if it is safe to go to that area depending on parameters such as population density, and number of reported infections over past two weeks. The calculation will produce a single value representing the relative danger level among boroughs

There will be a date selector above the map for forecasting cases on a future date using Machine Learning linear regression. More advanced models are optional.

Web App will allow users to optionally fill in a form and will record rough estimated location (non-identifiable) and gather data for developing own database over time (which will then allow the visualiser to collector more granular data) This processing of the data is left out of scope of the project and is suggested as a future improvement.

Ethical issues:

- Storing and processing user (non-identifiable) data

I confirm that the proposed project conforms with College and professional ethical guidelines, as indicated: *(please delete as appropriate)*

1. Access to participants: YES
2. Informed consent: YES
3. Anonymity and Confidentiality: YES
4. Potential Harm to Participants: YES
5. Potential Harm to Researcher(s) YES
6. Potential Harm to the College: YES
7. Participants' right to decline to take part: YES
8. Uses of the information (including publication): YES
9. Conflicts of Interest: YES
10. Other relevant ethical concerns, including those arising from internet research (please specify): NO

Classification of project *(please delete as appropriate)*: ROUTINE

The applicant: Abu Bakar Naseer Date: 30/09/2020

I confirm the proposal classification as:

ROUTINE

Decision (please delete as appropriate):

Acceptance

Departmental Research Ethics Officer:  Date: 13/10/2020