In this report:

- **Design Details for the Sympt-API (using Promed-mail)**
- **Design Details for the Sympt-App**

Design Details Sympt-API

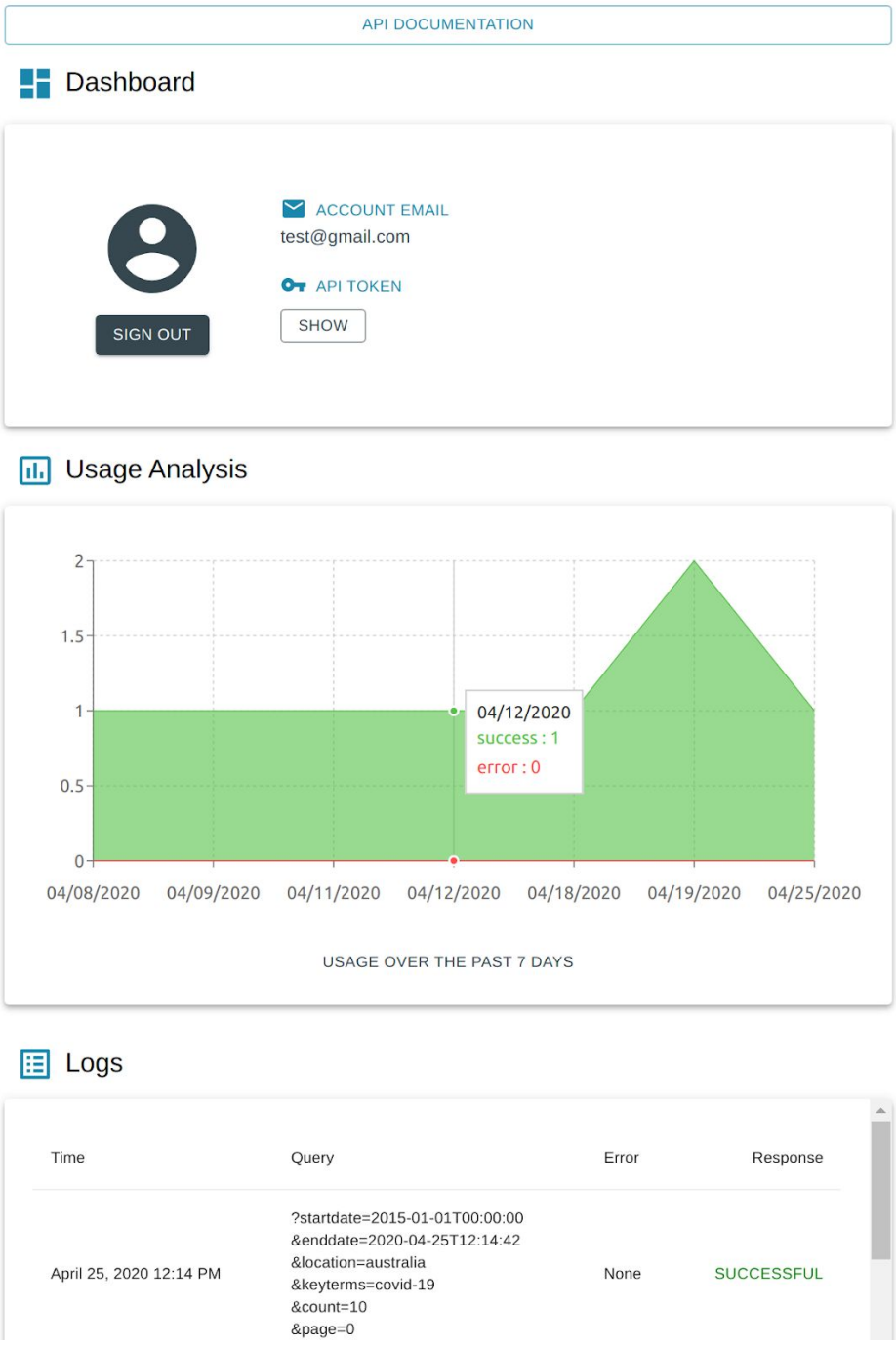
Design decisions were made on three considerations: the developer audience, resources and constraints and the business drivers. The goal for the API was to collate and retrieve disease reports which fit under the same search criteria. This was done so that searching for articles was easier and the user is able to filter out content which is irrelevant to them.

Sympt API contains one GET endpoint which returns disease articles based on the date of publication, location and optional key terms. Example requests are discussed below. For developer information visit: http://sympt-swagger.herokuapp.com/docs/

API Usage

In order to use the Sympt API, a developer must register online at https://symptdev.netlify.com/
Once registered, they can sign in and will be issued an API authorisation token which they can use to make requests by passing it in the header.

On the dashboard, a developer can expect to see their information as well as a log of their requests. Shown below is the dashboard of a user with email test@gmail.com.



API Module - Technology Stack

Beams utilised the following technologies to develop the API module:

- NodeJS
- ExpressJS
- Typescript
- Firestore
- Puppeteer

The backend was constructed in Express with NodeJS. Express is useful in creating server applications and is considerably simple, flexible and easy to use. Given the short time frame of the project, Beams decided to utilise NodeJS + Express to aid in creating the base of the backend in a shorter time frame. This allowed for more time to be dedicated to other opportunities such as improving scraping technologies and future web-app features.

Moreover, Beams developed in Typescript, a superset of the Javascript language which enforces strict data types. We utilised ts-node to compile our TS code into ES2020 compatible JS on-the-fly, thus allowing us to take advantage of the latest JS features and produce more compact, readable code. Strong typing also provided further rigidity in our code which helped us detect issues before run time, and, together with strict ESlint integration, allowed us to keep our codebase consistent with potential to scale.

Firebase's Firestore database tool was used in conjunction with our web scraper. Results from the web scraper were formatted and stored in the Firestore database for fast retrieval by user API requests. Firestore's online interface opened opportunities to test data structures online, have multiple tenants interacting with the same data, and provided real-time updates to data changes both in-app and through our servers, bringing us opportunities to experiment and produce results rapidly.

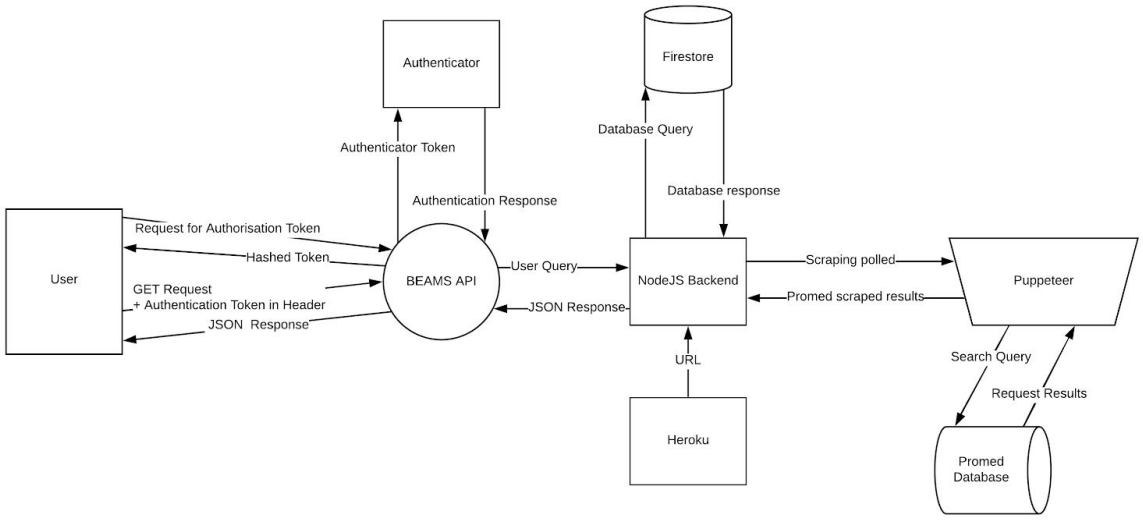
Beams used Puppeteer, a Google Developer Tool, which instantiates a Headless Chrome component that allows developers to interact with the browser DOM without the GUI of Chrome. This allowed us to generate server-side rendered content to interact with the ProMed site, including entering terms into ProMed's search system and scraping relevant articles and post

results. We can further process and parse this raw data, along with other data received from other sources, into JSON format and structure it as an accessible API endpoint.

Additionally, Beams integrated CircleCI into our GitHub repository to ensure our commits and pull requests abide by preset linting standards, and pass unit tests which will be developed overtime. CD methods are implemented through the heroku hosting platform.

In conjunction with CI, Beams used commit hooks with husky to ensure code is neat and passed tests before it enters the CI pipeline. We utilised Airbnb's ESLint specification (https://github.com/airbnb/javascript) since it provides a rigorous guideline to follow and because it allowed us to adapt to industry-level code consistency and style. This ensured mistakes were caught before they were pushed into the master branch and saved time when there were several branches being tested and built through our CI pipelines.

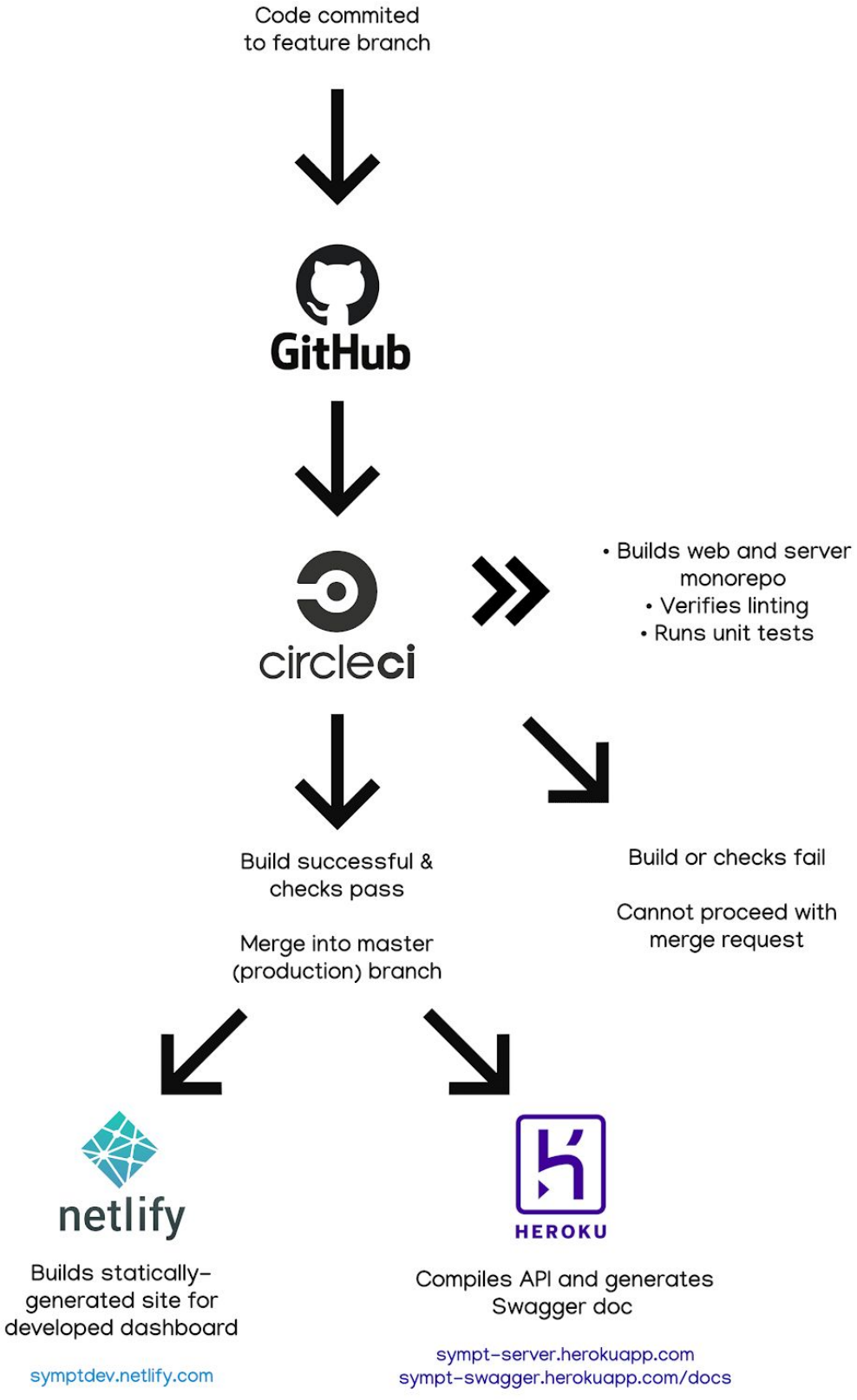
API System Design Diagram



API Deployment & DevOps

Using Heroku, the API is deployed at http://sympt-server.herokuapp.com/. The developer dashboard is hosted separately as a statically-generated site on Netlify. By utilising GitHub branch checks with CircleCI, commits are only merged into the master (i.e. production) branch if tests are passed to assure errors do not slip into the hosted site/server. These 3 platforms together create a strong DevOps ecosystem that automates vital processes, therefore speeding up development and code confidence significantly.

DevOps Cycle



API Request Design

The API serves responses to users using the standard URL query method. The API collates and formats data scraped from https://promedmail.org/ using a web-scraper.

API Request Format:

http://sympt-server.herokuapp.com/articles/?startdate=2018-01-01T08:45:00&enddate=2018-01-01T08:45:00&location=somelocation&keyterms=firstterm*[,secondterm,thirdterm…]*&count=requests-per-page&page=page-no

An example request for searching for 5 articles on coronavirus and SARS during January, 2020 in China with an existing authorisation token passed in the header:

http://sympt-server.herokuapp.com/articles/?startdate=2020-01-01T00:00:00&enddate=2020-02-01T00:00:00&location=china&keyterms=coronavirus,sars&count=5&page=0

The API does not enforce strict URL parameters so that usage is flexible. However, the following conditions must be met:

- The start date and end date meet the format `yyyy-mm-dd` where:
    - yyyy is the year
    - `mm` is the month
    - `dd` is the day
- Start date is before end date

Any deviation from the above conditions will result in a malformed request response from the API. We provide dynamic error checking that can inform the API users of bad endpoint format, missing queries or unauthenticated access in the future.

The parameters are discussed below:

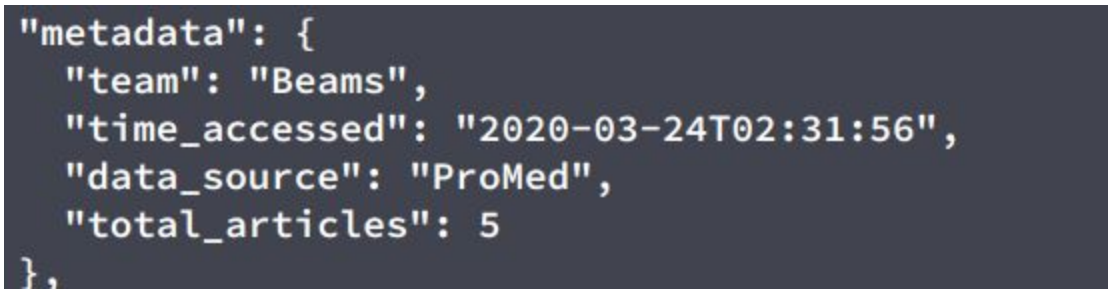
- The start date and end date also accept the format `yyyy-mm-ddThh:mm:ss` where:
    - `hh` is the hour
    - `mm` is the minutes
    - `ss` is the seconds
- Location is a string and is case insensitive

- Keyterms are a list of comma separated terms which must come from the disease_list.json file otherwise results returned are too vague
- Count is a number between 0 and 10. If set to 0 or not set the API will return all articles matching other search criteria
- Page is a positive number or 0. Page 0 displays the most recent articles with each page after going further back into the database. If the page is not set it becomes 0. Each page contains 10 articles.

API Response Design

The API returns a JSON object with two items: metadata and articles.

- Metadata is an object, shown below.



```
"metadata": {
  "team": "Beams",
  "time_accessed": "2020-03-24T02:31:56",
  "data_source": "ProMed",
  "total_articles": 5
},
```

- Articles is an array of Article objects. Shown below.

```
Article ⌄ {
  url*                  string
                        example: https://promedmail.org/promed-post/?id=6943858
  date_of_publication*  string
                        example: 2020-02-01 23:02:03
  headline*             string
                        example: Novel coronavirus (28): China (HU) animal reservoir
  main_text*            string
                        example: On 26 Jan 2020, the China Centers for Disease Control and Prevention announced that the new coronavirus was detected in environmental samples from the
                        South China Seafood Market in Wuhan [see item 2]. The virus originated from wild animals sold in the seafood market. For this reason, the State General
                        Administration of Market Supervision, the Ministry of Agriculture and Rural Affairs, and the National Forestry and Grassland Bureau jointly issued a public
                        announcement on the 26th that from now on until the end of the national epidemic, wild animal trading activities are prohibited to cut off the source and
                        transmission of the virus.
  reports*              ⌄ [Report ⌄ {
    event_date*           string
                          example: 2020-01-03 xx:xx:xx to 2020-01-15
    locations*            Location ⌄ {
      country*              string
                            example: China
      location*             string
                            example: Wuhan
                        }
    diseases*           ⌄ [string
                        example: Coronavirus]
    syndromes*          ⌄ [string
                        example: Fever]
                      }]
}
```

Challenges faced during API development

These will be discussed below:

- Retrieval from ProMed and text processing for the user
- Limited resources
- Security

Usability Considerations

Scraping of the ProMed website was initially difficult as the website itself does not change URL when a new article is clicked, choosing to instead change a component on the page. This meant that accessing articles and their html source was difficult. To combat this, Beams used Puppeteer (as described above) and was able to successfully scrape the ProMed website based on the key terms and diseases which were given to us and upload them to our database.

In terms of processing the text received from ProMed, Beams found the articles to be lengthy and contain references to other articles and random chunks of text which did not provide value to the user. As such, we have processed the text and retrieved what we deem is the most important part of the article. This ensures the user is given nicely formatted text. Processing was done through regex and stripped the article of any bare bones html as well as random hyperlinks that were placed throughout and at the end of the article.

In the cases of many search results, the parameters count and page have been introduced. This ensures that users can access articles periodically and specify how many articles they wish to receive. This is done so that one request does not throttle the network or take too long to return a response.

Count: This parameter is optional and must be between 0 and 10. When specified, it will limit the amount of returned articles to its value. If count is larger than 10, a 403: bad request error will be returned.

Page: This parameter is optional. Pages start at 0 and contain 10 articles per page. If the requested page contains no articles, the user will receive a 500 error notifying them that the current page exceeds the amount of articles remaining.

The introduction of these parameters makes our API flexible and allows the developer to easily dictate which articles and how many they want.

Manageability Considerations

In order to use the API, a user must register on the API website (symptdev.netlify.com). This ensures that the API is being used by registered users, but also allows for users to manage and monitor the requests they are making. A developer dashboard is provided so that the user can see when and what request they made along with other information such as the response code and what errors were returned. Users can retrieve their API access tokens from here, and are also given the option to refresh their tokens in case their current one has expired or been leaked.

Security Considerations

As stated before, a user must register to use the API. This is done to monitor API usage and prevent DDoS attacks. Once registered, the user will be given an authentication token that they are required to pass in the header through with each request. This allows us to identify each user and maintain a record of the types of queries our users are making. In case our server or firebase service is attacked, we are able to disable firestore's read/write functionality to prevent data loss, and, after identifying the attacker, disable their account from access to the server.

Furthermore, we are able to utilise firebase's built-in authentication functionality to provide industry-level account management and security practices throughout the platform. Firebase can validate user IDs and generate JWTs that can be passed between client and server; the server can then validate these JWTs, verify who the request has come from, and perform actions securely.

The Beams team also distributes vital API keys in a private setting without external access. Keys are stored in environment variables and are ignored by git, so they are not unwillingly uploaded. Important actions are performed only through client-server interactions, so even if the client is compromised, the attacker will not have access to admin-level tasks.

Design Details Sympt-App

The Sympt App was designed to be a clean, intuitive and informative interface that NSW residents could open on the go to get stats and information on infectious diseases. With the current global climate, there was also a focus on COVID-19 and the impact that social distancing will have on the spread of the disease.

Final Product

The final product is a mobile application which displays disease statistics in a visually pleasing way for many infectious diseases (list can be found in Appendix below) prevalent in Australia. Furthermore, a predictions model was developed allowing users to see how, specifically COVID-19 would spread around Australia and, more in detail, in regions around NSW. A key feature of the final product is the ability of the user to toggle whether predictions take into account social distancing or not. This allows them to visualise the impact that social distancing measures have on the spread of contagious diseases such as COVID-19.

App Use Cases

This app, as stated above is designed for NSW residents. The reason for this is the fact that all developers are NSW residents and there were not enough reliable and accurate resources that we could access during the scope of the project to gather data sources from Australia-wide, let alone world-wide.

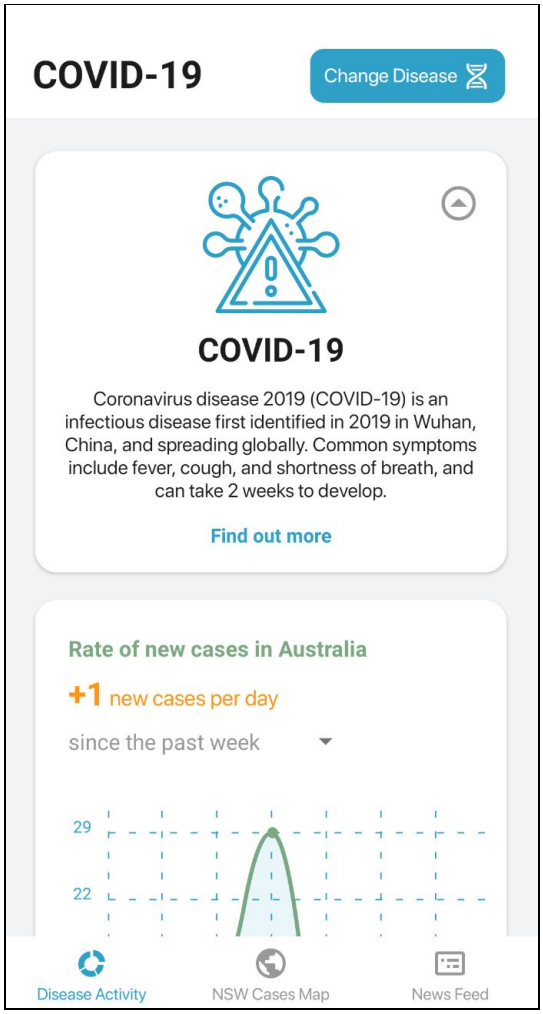
Since the aim of the app was to demonstrate, visually, the impact of social distancing as well as having an analytical focus, we believe the app would benefit someone who has an interest in how coronavirus is progressing as well as potentially an interest in other infectious diseases in Australia. The statistics which are displayed to the user for all infectious diseases include: cases in the past decade, year, month and week, the number of cases per state and the number of cumulative cases since the start of the year. The extra statistics which are displayed to the user who is interested in COVID-19 include: predictions for how the disease will spread statewise until Dec 2020 and a map of NSW divided into regions demonstrating how the disease will spread throughout regions a month from the current date.

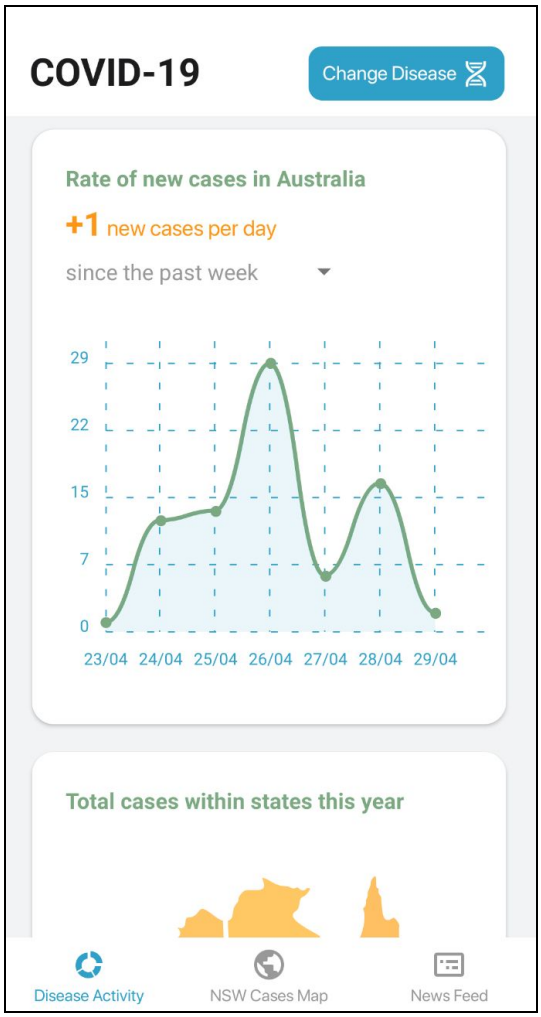
Final Design

Below are the app's various screens:

Disease Activity Screen

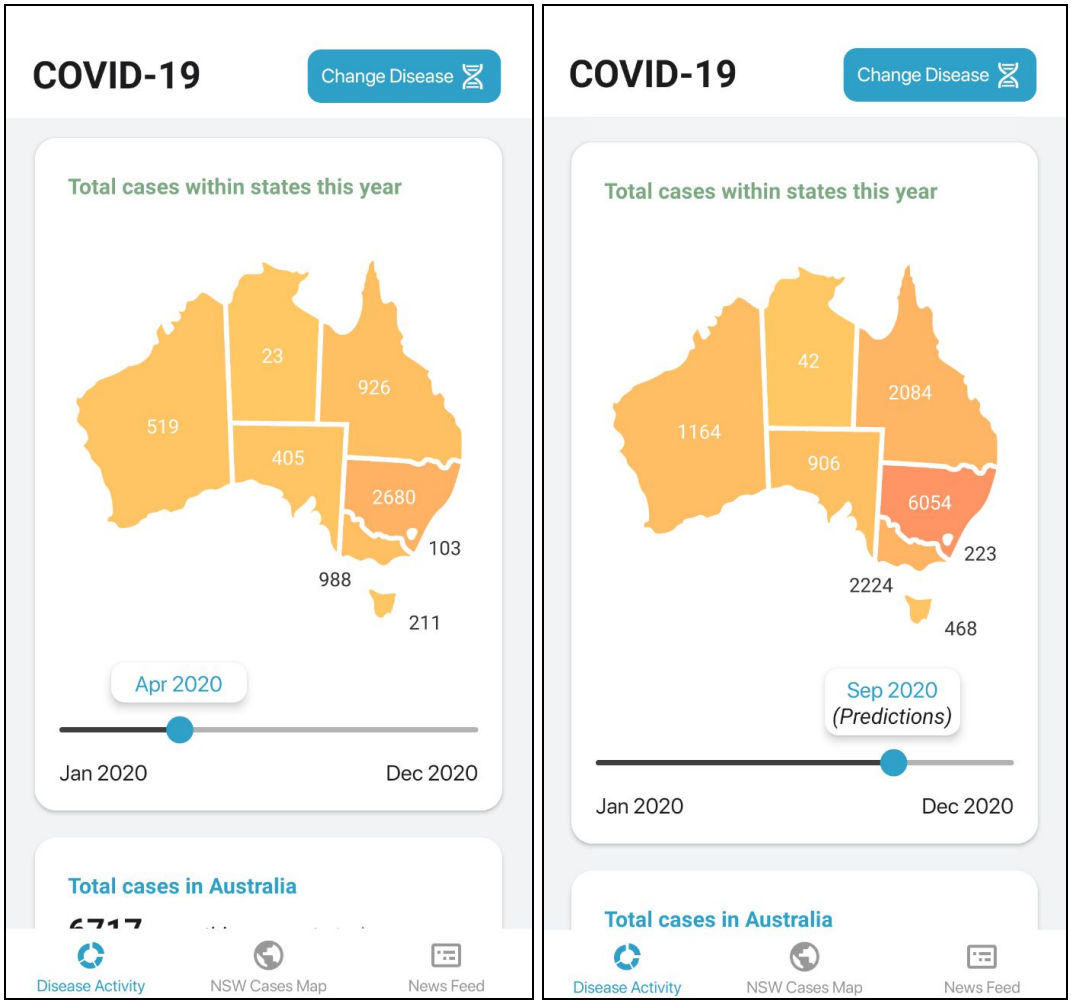
Shows info about the selected disease, as well as graphical statistics.





Disease Activity Screen

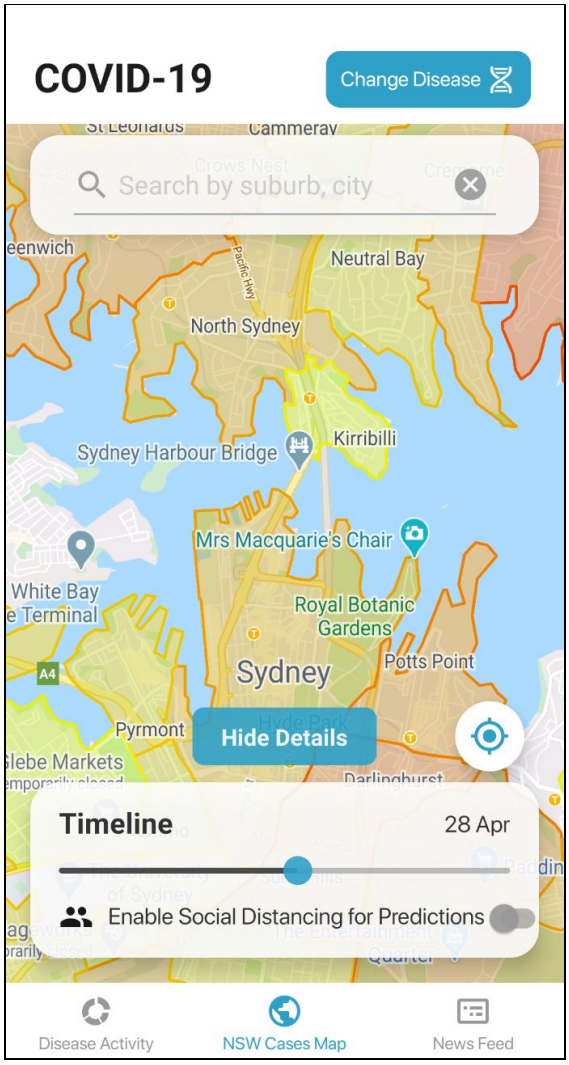
Showing total cases of COVID-19 Australia-wide from Jan 2020 until Present as reported by NNDSS and Predictions made by our SIR Model from Present to Dec 2020.



Disease Activity Screen and NSW Cases Map

Disease Activity Screen shows cumulative number of cases up to the date picked by user.

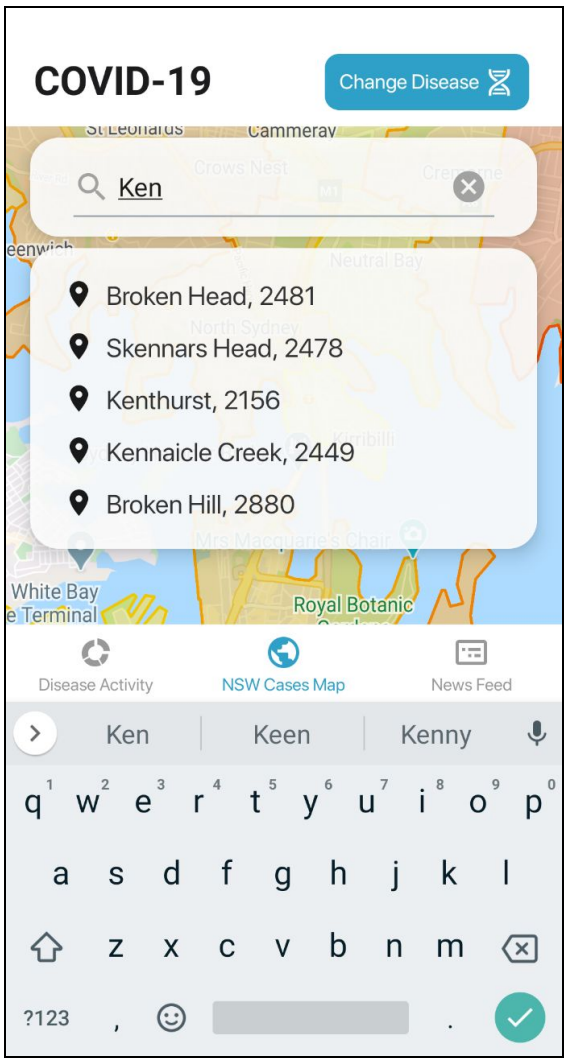
NSW Cases Map, default view, shows the number of cases within regions in NSW.

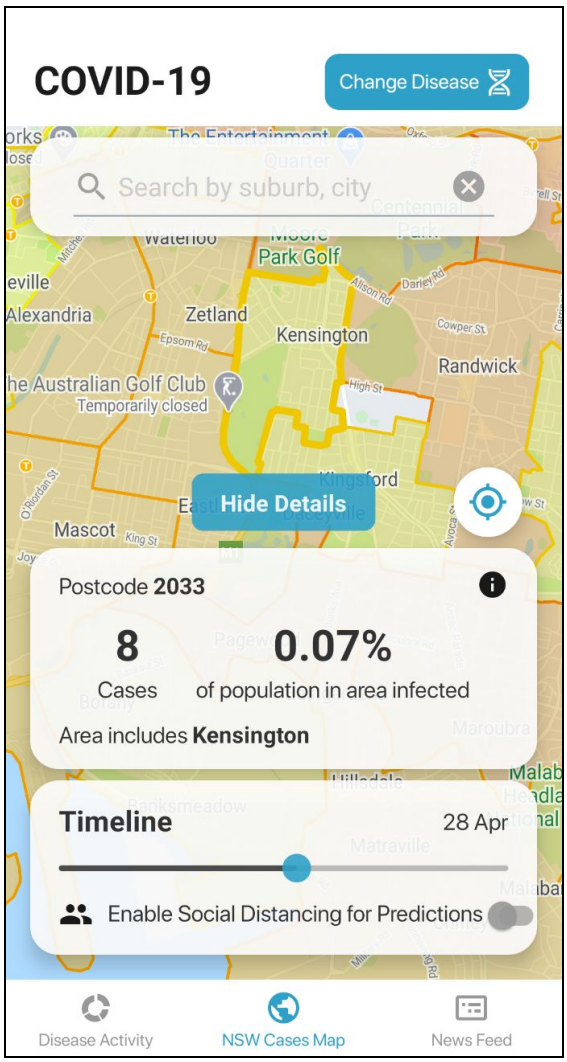


NSW Cases Map

Search Functionality allows the user to search for their suburb. Upon selection, the user is given more details about the region including the number of cases and the population in the area which is infected. The user can also click on the Button to the right of the ‘Hide Details’ button to be brought to the region they are currently in.

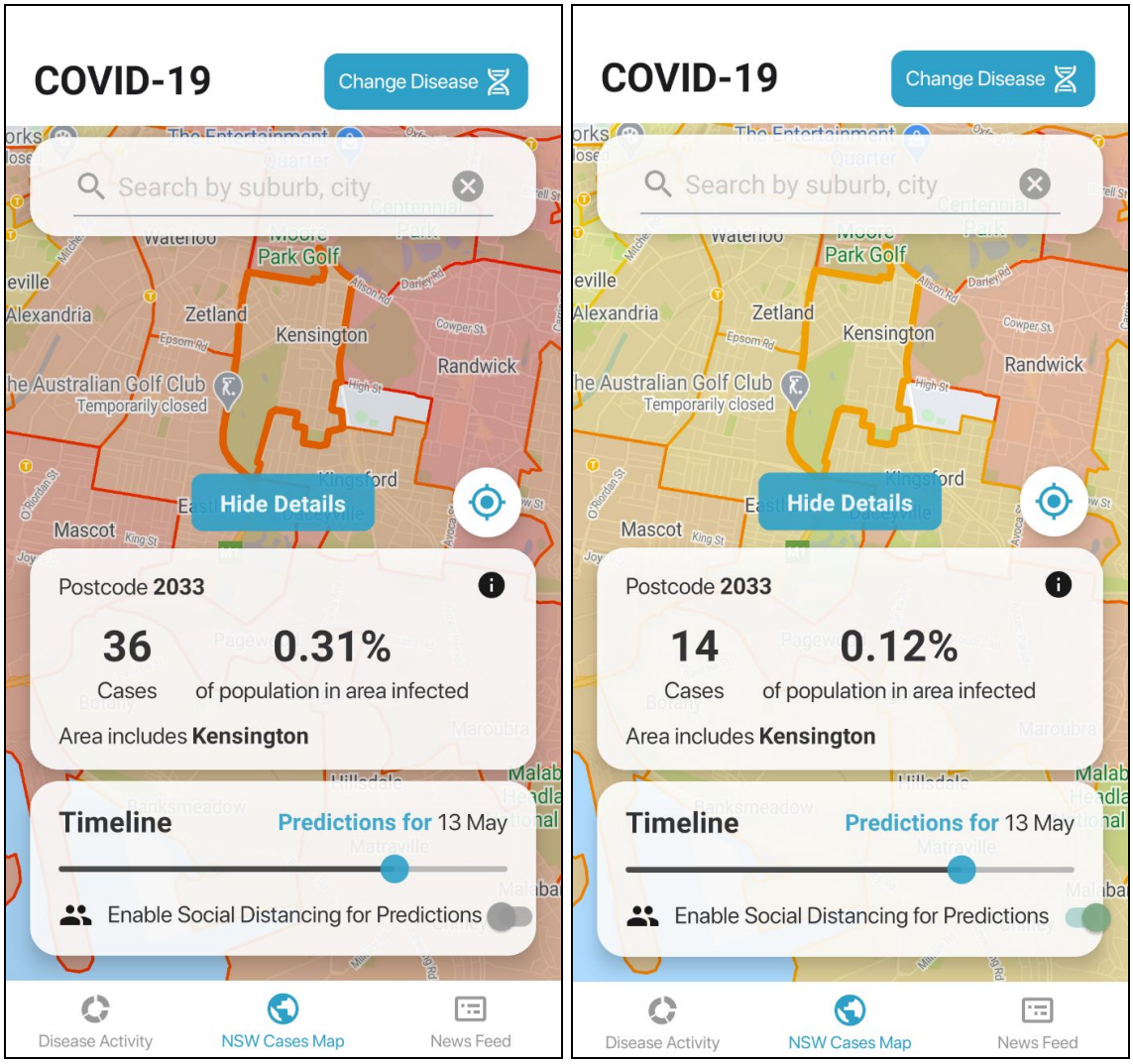
Underneath the details, the user is able to scroll through the timeline to see statistics for the past from March 25th to Present.





NSW Cases Map

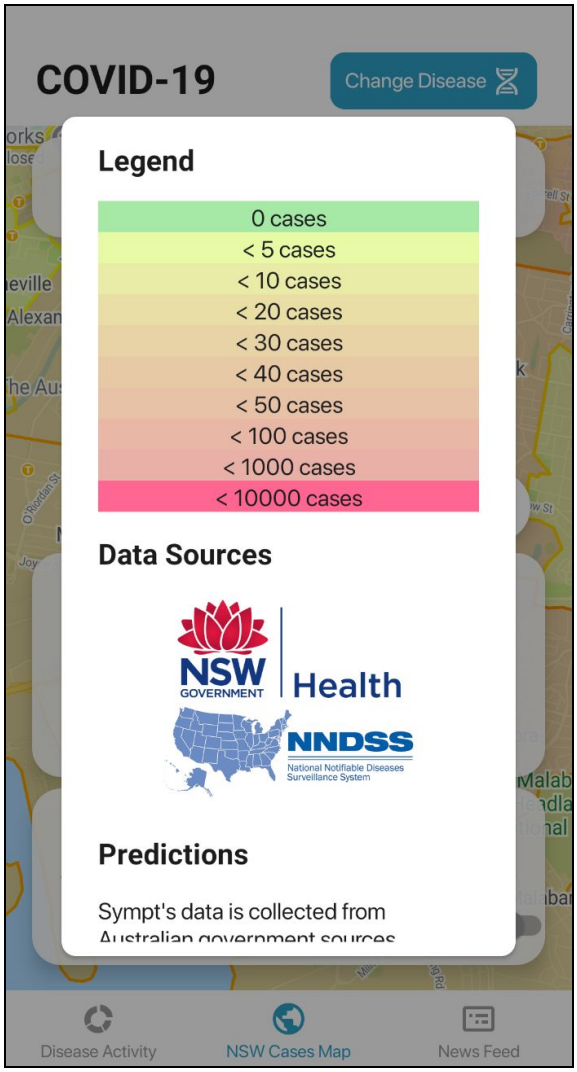
The timeline can also go into the future, allowing the user to view predictions up to a month in advance for their specific region, with or without social distancing.

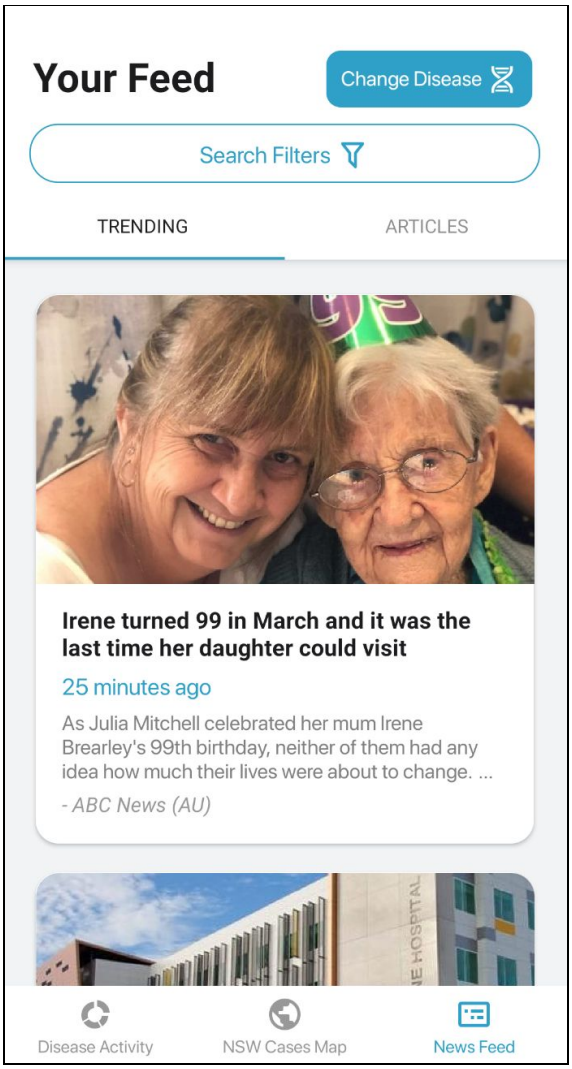


NSW Cases Map and News Feed

The legend can be opened by pressing the information button in the details section. Shows what each colour on the map means as well as the data sources for the information and a brief explanation of the SIR model used to calculate the predictions.

The News Feed shows articles related to Australia and the Disease you are searching for. If articles relevant to NSW exist, they will be displayed.



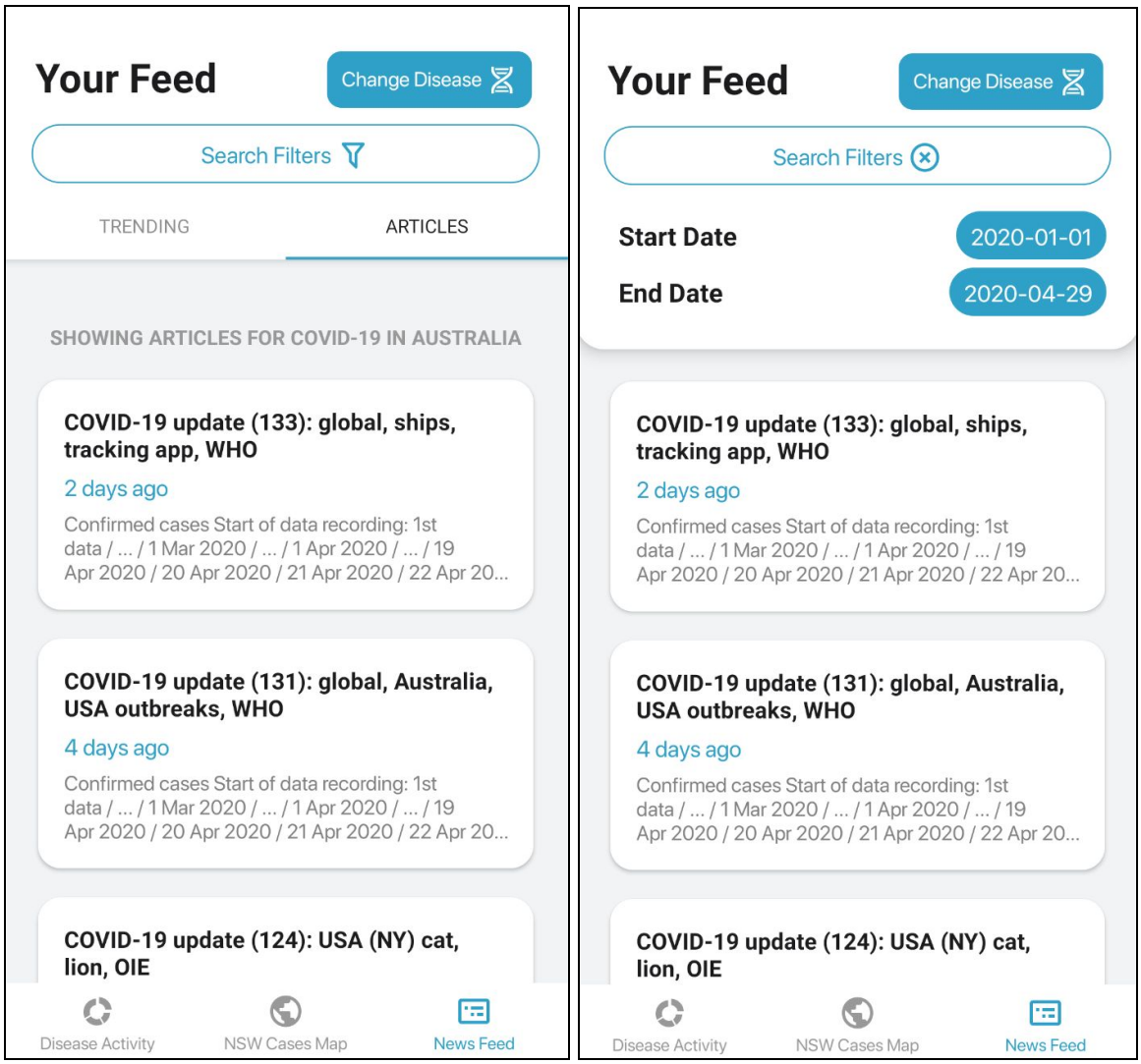


News Feed

This articles section is where we showcase the Sympt-API. It displays articles relevant to Australia in relation to the Disease the user has selected within the current year.

The user has the option to change the start and end date allowing them to search for articles within a specific time range.

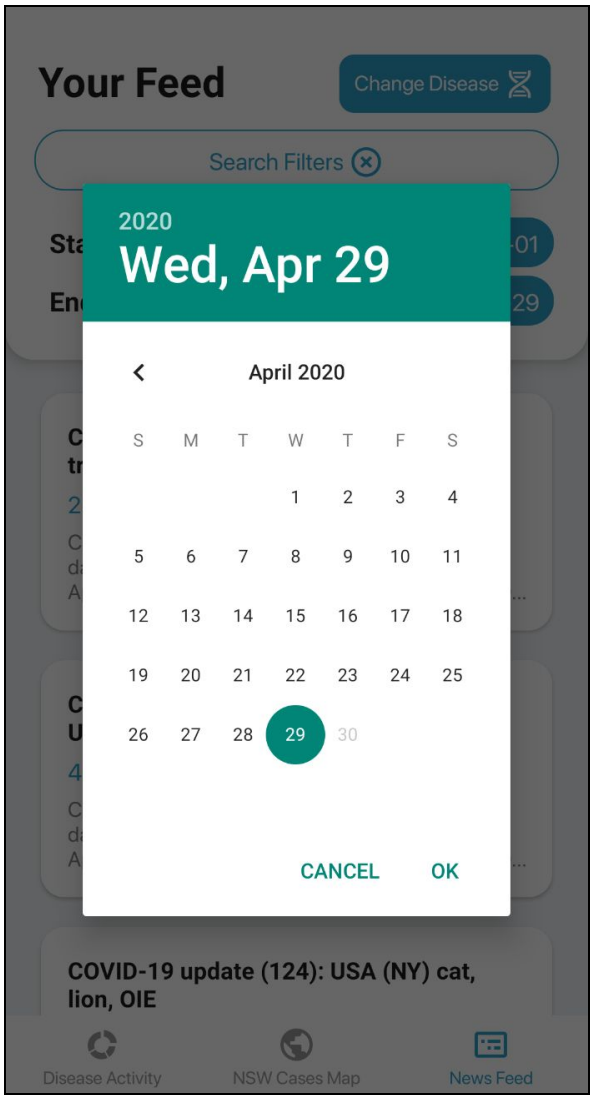
This feed also has infinite scroll implemented. The user will get 10 articles at a time so as to prevent from overwhelming their phone with many article results if the search time frame is large.



News Feed

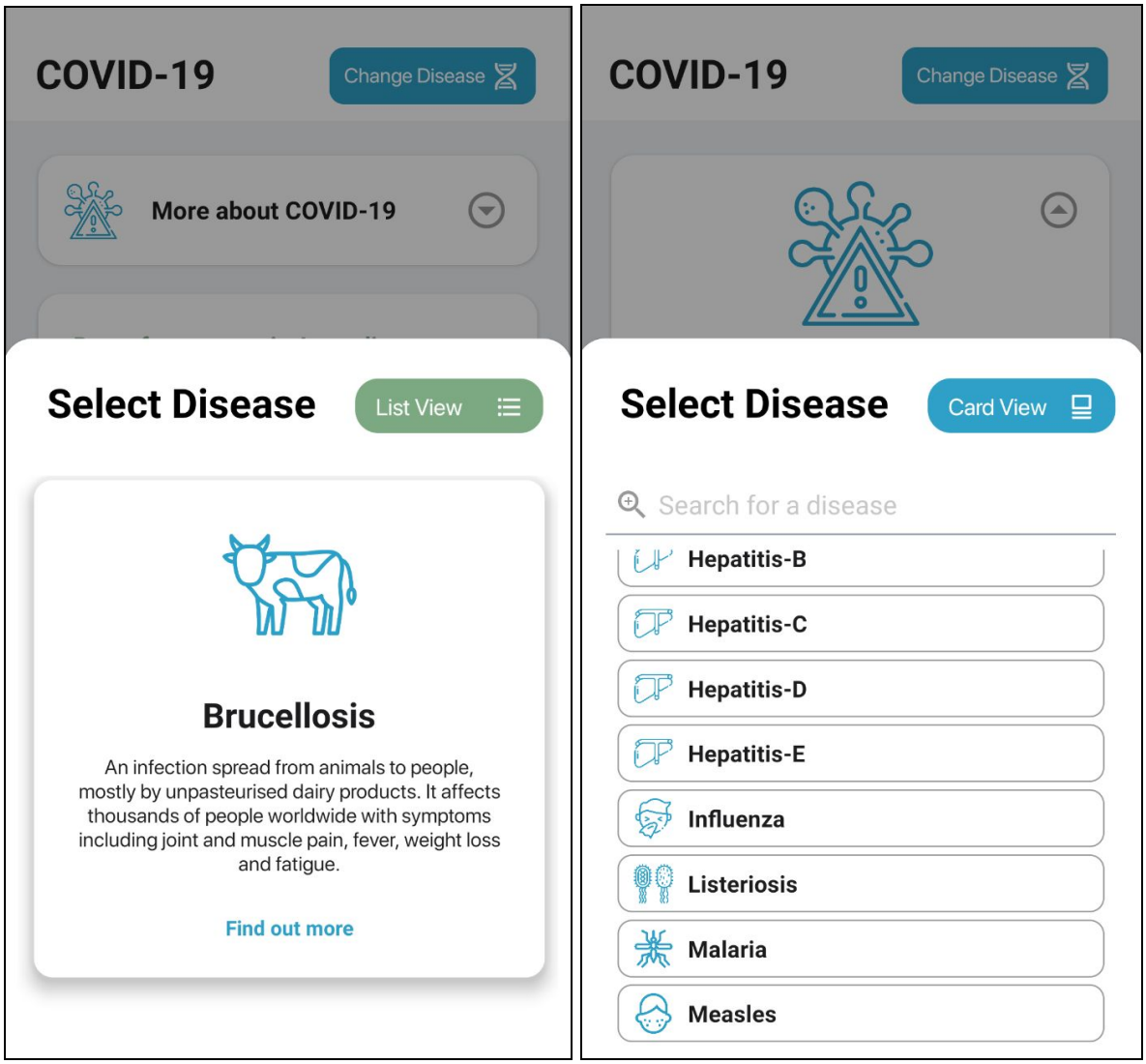
This is the modal that appears when users are choosing to set start and end date.

On the right is the preview of the Promed article that appears when the user clicks on an article card. The link at the bottom redirects to the Promed-Mail website.



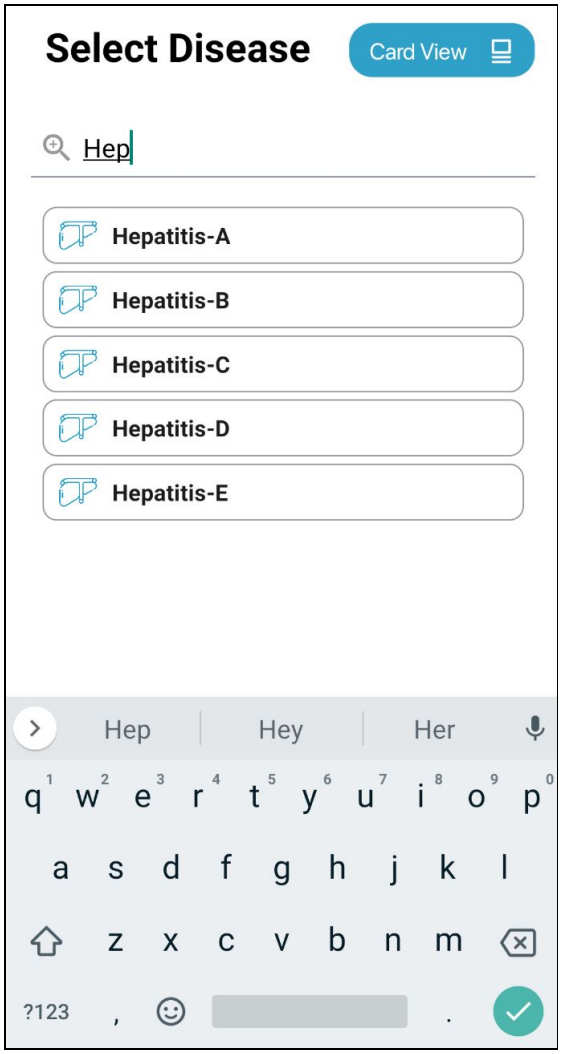
Select Disease

When the user clicks the change disease button at the top right hand corner, they are presented with a card view of all the infectious diseases which we have data on, by swiping left, the user is able to scroll through all the diseases. Alternatively, the user can choose to view the diseases in list view.



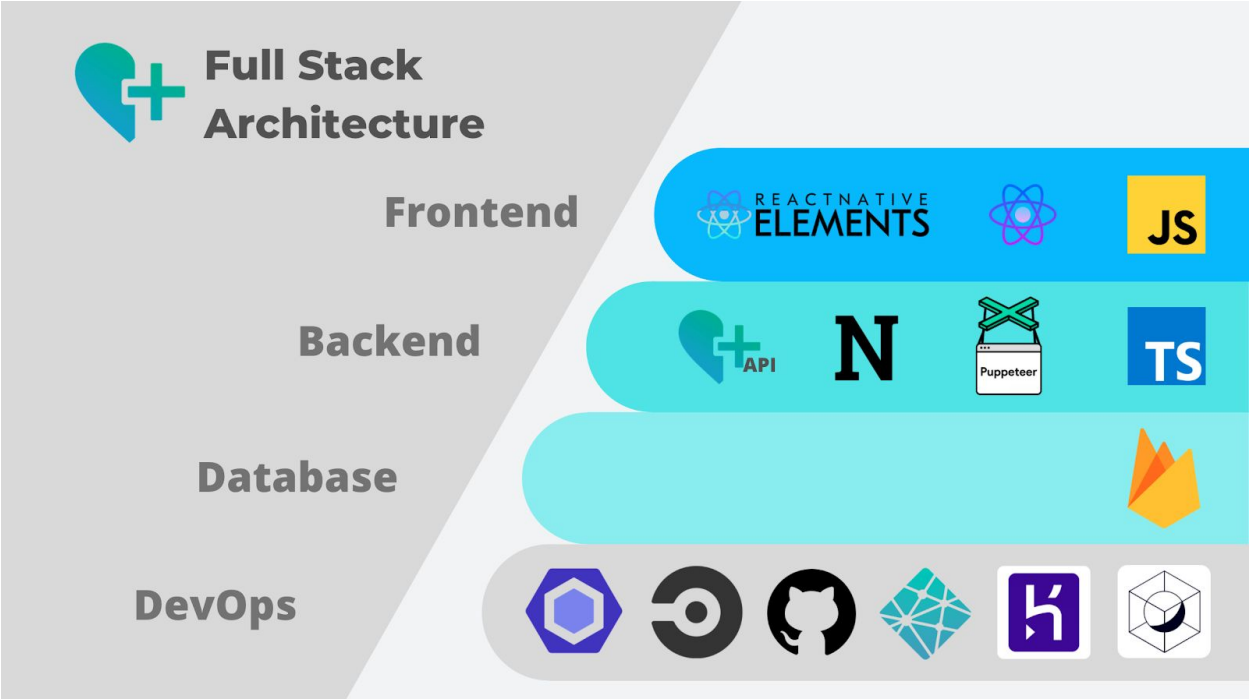
Select Disease

The search option allows a user to type in the disease which they are interested in to speed up the process of changing diseases.



App Technology Stack

The team used various technologies in the development of the app, listed in the image below. See following sections for reasoning and benefits of certain technologies.



Tech Stack Reasoning

The team built a React Native app using the Expo and Expo-cli tool in conjunction with Javascript as well as different backend endpoints and APIs developed in Typescript.

Expo is an open-source platform for making universal native apps for Android, iOS, and the web with JavaScript and React. It is quick and easy to use and learn allowing the team to focus on developing the final product. With the Expo client, we are able to send our app for review by each other which is extremely useful when both testing and debugging and for Quality Assurance purposes. This is all without having to build or create IPA or APK files, also supporting hot-updates during development to see incremental live changes to the frontend design and as well as viewing a live demo of the application on any android device that supports and can install the Expo Application for an accurate live demonstration of the application usability and performance without the use of an emulator.

The team chose to develop a mobile app using the open-source mobile application framework React Native. React Native combines native development with React and Javascript libraries to allow developers to build intuitive user interfaces. Given the team's proficiency in React and Javascript, React Native was a tool that was easily learned by most team members and allowed for fast and clean development of the final product. React Native provides a simple UI which was enhanced with React Native Elements and serves the user their desired pages fast, also providing developers an easy to implement and reference library supported by online official and unofficial documentation.

Frontend developers developed all application features in Javascript and Backend developers developed all server, API and web scraping features in Typescript. The choice of Javascript was a simple one due to its immediate allowance of the use of the React Native and React Native Elements libraries. All developers were well-versed in Javascript having prior exposure to Frontend development which also made it an easy choice. A further pro of the use of Javascript was its vast online support of not only the language but also supporting libraries, namely the React Native family. Backend development was completed in Typescript to overcome Javascript's most clear flaw which is its lack of static typing whilst inheriting all the pros of Javascript. This, combined with a linting service ESLint made development, debugging and

testing much more coherent and streamlined rather than being separate development stages, also improving basic readability of code for other developers.

Google Maps API & Google Places API were chosen for their simplicity and abundance of documentation. The Google Maps API was used in conjunction with the react-native-maps npm package which allows developers to easily display a map in their mobile application. This package also allowed for easily implementing geocode regional outlines and nice highlighting and interactive features which added to the user experience. Google Places API was used to resolve the users postcode into their region. This was the API we chose as we had already been using Google Maps API and this API served the purpose we needed.

Our Sympt-API was developed as a majority of our backend and hosted by Heroku, allowing other developers or interested parties to formulate their own queries for our API and to receive results accordingly. Being developed as a Backend feature, the API itself was developed in Typescript for reasons mentioned above. The API was a combination of a database connection as well as a supporting web scraper, this allowed the majority of the requests to be served with both relevant and accurate information. On an initial call to our API service, we try to get all information from the database, which has been previously populated by automated scraping of the PromedMail website articles, that matches the callers search query terms. In the event that there aren't sufficient articles stored in the database, we then trigger our web scraper to scrape the PromedMail website along with the given search queries, the results of which are both stored in the database and served to the caller.

The NewsAPI was set up as an additional feature for the users of our application intended to complement the data presented to them by our application, listing news that is most relevant to Australian NSW Citizens given the target demographic of our application users, filtering on all received news sources accordingly. We initially planned to include a list of Tweets from the Twitter API however we decided against in part combination of the feedback received from our first presentation being that before we added more features we needed to reduce the scope of the problem we were solving and that we were solving one specific problem. Additionally, given the tight time constraints we decided to cut the feature as it was non-essential.

Firebase is a suite of online services utilised as a backend-as-a-service to provide backend functionality such as authentication and a real-time database without needing a live server, and can be scaled for a small-sized app for free. We decided to utilise authentication through Firebase for our API login and signup due to its quick setup and industry-standard encryption methods. This allowed us to focus on developing the app and website rather than building a whole authentication system from scratch. Firebase authentication also provides API features including refreshing tokens and managing users from the Firebase console giving us fine-grained control over users of our API.

Cloud Firestore is a flexible, scalable, NoSQL database for mobile, web, and server development from Firebase and Google Cloud Platform. We chose this platform as our database due to its cloud capabilities as well as its automatic data handling and load balancing capabilities. On top of this, it had a large amount of reads and writes included in its free package, and paid extension is relatively inexpensive if required.

APIs and Data Sources

The team utilised 4 main APIs:

- Google Maps API and Google Places API
- News API
- Sympt-API

The team also utilised these data sources:

- National Notifiable Disease Surveillance System (NNDSS) from the Australian Government Department of Health
- NSW Government Health
- SIR Model for Spread of Disease from the Mathematical Association of America

How the APIs Were Used

Google Maps and Places API

Google Maps API was used in order to generate the NSW regions map allowing our users to view their own region and regions around the NSW state.

Google Places API was incorporated in this screen as well, allowing users to pinpoint their own location. The Places API took in the users coordinates and was used to find their suburb and region from these coords, allowing us to highlight the region relevant to the user.

News API

The News API was used to generate a news feed relevant to Australians and NSW residents with a focus on the disease that the user was searching for. The articles were filtered on Australian news sources as well as key words such as state names and suburb names. This ensured that irrelevant news articles were removed from the feed.

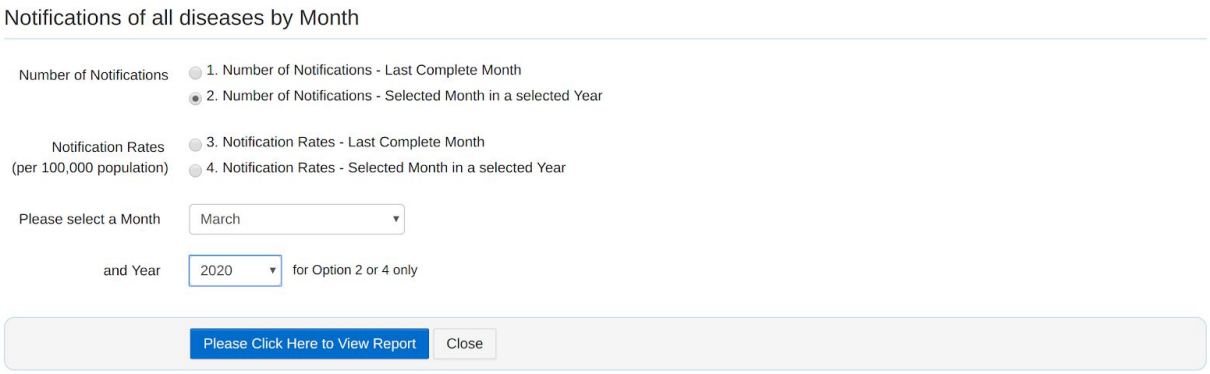
Sympt API

The Sympt API was used similarly to the News API and generated a feed of articles pulled from Promed-Mail. The articles were filtered on Australia as the location and the specific disease that the user had preselected. We also allowed the user to edit start date and end date allowing them to narrow their search to what they deemed relevant. The default values for start and end date limited the articles to just articles published this year.

How the Data Sources Were Used

NNDSS

The National Notifiable Disease Surveillance System coordinates the national surveillance on an agreed list of communicable diseases or disease groups in Australia. This data was displayed in tabular form on their website with results separated by months as shown below.



Home / All diseases by month / View Report

Number of notifications of diseases received from State and Territory health authorities in the month 1 March to 31 March 2020										
	ACT	NSW	NT	Qld	SA	Tas	Vic	WA	Aust	Aust YTD
Bloodborne diseases										
Hepatitis (NEC)	0	0	0	0	0	0	0	0	0	0
Hepatitis B (newly acquired)	0	0	0	4	0	2	0	3	9	27
Hepatitis B (unspecified)	9	171	0	72	19	10	52	43	376	1292
Hepatitis C (newly acquired)	0	0	0	37	0	0	1	7	45	166
Hepatitis C (unspecified)	14	239	15	163	12	17	169	82	711	2020
Hepatitis D	0	0	0	0	0	0	0	0	0	10
Gastrointestinal diseases										
Botulism	0	0	0	0	0	0	0	0	0	0
Campylobacteriosis	58	788	41	733	150	106	504	202	2582	9314
Cryptosporidiosis	4	80	9	227	10	1	43	149	523	1473
Haemolytic uraemic syndrome (HUS)	0	0	0	1	0	0	0	0	1	3
Hepatitis A	0	2	0	1	2	0	18	2	25	63
Hepatitis E	0	3	0	3	0	1	0	0	7	25
Listeriosis	0	2	0	0	0	0	1	1	4	11
Paratyphoid	0	4	0	2	0	0	3	0	9	35
STEC	0	12	0	3	18	0	12	16	61	228
Salmonellosis	13	330	39	484	78	28	188	163	1323	5618
Shigellosis	3	61	53	39	15	0	37	31	239	1028
Typhoid Fever	0	3	0	4	1	1	9	0	18	66
Other bacterial infections										
Legionellosis	0	19	1	9	4	1	18	7	59	142
Leprosy	0	0	0	0	0	0	0	0	0	0

We were able to scrape the website using Puppeteer to get infectious disease data from Jan 2000 until the present month (April 2020). This information was stored in our firestore database. Following this, a private endpoint was created on our heroku hosted server (found at http://sympt-server.herokuapp.com/_cases/?diseases=disease_name&location=state_name) that retrieved the number of cases by state. This is an example request getting results for Diphtheria in NSW
https://sympt-server.herokuapp.com/_cases/?disease=diphtheria&location=NSW
And the sample response:

```
// 20200429121528
// https://sympt-server.herokuapp.com/_cases/?disease=diphtheria&location=NSW

{
  "2007-01-01": 0,
  "2008-06-01": 0,
  "2020-03-01": 0,
  "2002-01-01": 0,
  "2019-12-01": 0,
  "2019-01-01": 1,
  "2016-04-01": 0,
  "2008-02-01": 0,
  "2004-08-01": 0,
  "2004-07-01": 0,
  "2009-08-01": 0,
  "2000-11-01": 0,
  "2005-11-01": 0,
  "2020-04-29": 0,
  "2003-10-01": 0,
  "2011-10-01": 0,
  "2020-04-24": 0,
  "2009-05-01": 0,
  "2017-11-01": 0,
  "2004-03-01": 0,
  "2012-06-01": 0,
  "2015-03-01": 0,
  "2020-04-19": 0,
  "2013-08-01": 0,
  "2020-04-01": 0,
  "2015-04-01": 0,
  "2014-09-01": 0,
  "2015-08-01": 0,
```


The response contains the number of cases in that month and year for the selected disease and location. This data was then used to generate the Australia wide map which displayed the total number of cases per state per disease.

NSW Government Health

This resource was used to get the number of cases of COVID-19 per region in NSW (for the NSW Map Cases Screen). The data was pulled from this website https://nswdac-np-covid-19-postcode-heatmap.azurewebsites.net/datafiles/data_cases.json which was updated daily. Below is an example of how the data is organised.

```
// 20200429123452
// https://nswdac-np-covid-19-postcode-heatmap.azurewebsites.net/datafiles/data_cases.json

{
  "data": [
    {
      "POA_NAME16": "2290",
      "Number": 5,
      "Date": "25-Mar"
    },
    {
      "POA_NAME16": "2566",
      "Number": 3,
      "Date": "25-Mar"
    },
    {
      "POA_NAME16": "2300",
      "Number": 2,
      "Date": "25-Mar"
    },
    {
      "POA_NAME16": "2540",
      "Number": 2,
      "Date": "25-Mar"
    },
    {
      "POA_NAME16": "2196",
      "Number": 3,
      "Date": "25-Mar"
    },
    {
      "POA_NAME16": "2219",
      "Number": 4,
      "Date": "25-Mar"
    },
```


As you can see, the data contains objects which are organised by Postal Area Code and contain the number of cases as well as the date that this number was recorded.

This data was used in conjunction with ABS which provided postcodes which were included in the postal area codes and each suburb population.

SIR Model

The SIR Model for spread of Disease, specifically the differential equation model, was used to compute the predictive data to reflect future disease spread and projection. The predictions model plots three functions that extend over time, allowing the application to display day-specific predictive data. Further, the SIR model stands for Susceptible, Infected, Recovered, and hence the first function mapped by this model $S(t)$ is one to represent the number of susceptible individuals in a population, at a given day. This function's derivative is calculated by multiplying the chance of a transmittable contact between an infected person and a susceptible person at any given day, b, by the infected population $I(t)$, and again by the susceptible fraction of the population $s(t)$.

$$\frac{dS}{dt} = -b\, s(t)\, I(t)$$

To calculate the function $s(t)$, simply substitute the infected population $I(t)$, by the fraction of the infected population instead $i(t)$.

$$\frac{ds}{dt} = -b\, s(t)\, i(t)$$

The second function mapped by this model, $R(t)$, is one to represent the number of recovered individuals in a population, at a given day. This function's derivative is calculated by multiplying the chance of recovery from a particular disease at any given day, k, by the infected fraction of the population $i(t)$.

$$\frac{dr}{dt} = k\, i(t)$$

The third and last function mapped by this model $I(t)$, is one to represent the number of infected individuals in a population, at a given day. This function's derivative uses both previous differential equations $s(t)$ and $r(t)$, to compute the differential equation for the fraction of the infected population.

$$\frac{di}{dt} = b\, s(t)\, i(t) - k\, i(t)$$

Hence, with all the function's derivatives calculated, we can write a simple loop to calculate each differential equation for a large number of days, and push those into an array to represent them onto the application as region specific predictive data.

Furthermore, the constant b used in this implementation of the SIR model was calculated with respect to health data, provided by nsw government sources. However, this implementation used two different constants for the chance of valid contacts for disease transmission, one to reflect the rate of spread with social distancing measures, and one without. Using COVID-19 as an example, the calculation of these constants carefully observed periods of time when the spread of the disease was at its highests (end of March to the beginning of April), and again when the spread of the disease was at its lowest, as a result of social distancing (mid April to present time). The use of these two constants allowed for the application to represent this predictive data in a way that can inform our users of the importance of social distancing restrictions, as well as give users a higher and more informed understanding of the spread of the disease over time.

Lastly, the last constant used k, reflects the period of recovery for a disease. Again, using COVID-19 as the example, we used the average recovery time for the disease to calculate the rate of recovery of patients in our predictions model (2-3 weeks).

Key Benefits of our Design

The key benefits of our project in terms of implementation and design were:

- Reusable components
- Implementing APIs through backend
- Storing data in database efficiently and in a way that made data retrieval easy

React Native allowed for the development of components which were easy to use and could be reused. Furthermore, splitting parts of the screens into components meant that the code base follows the Single Responsibility Principle. Code was also easy to read and understand which meant that if another person had to edit or work on a component they did not develop, it was easy to pick up and modify.

The team implemented API calls through the backend making the front end rendering much quicker and efficient. To further comment on this, since we were using our own API we were able to modify and reduce the number of calls we had to make by structuring the database in a way which was efficient and able to serve us the data we needed fast.

Appendix

List of Infectious Diseases that Sympt App displays

- COVID-19
- Malaria
- Measles
- Influenza
- Chickenpox
- Syphilis
- Brucellosis
- Chikungunya
- Cryptosporidiosis
- Dengue
- Diphtheria
- Hepatitis-A
- Hepatitis-B
- Hepatitis-C
- Hepatitis-D
- Hepatitis-E
- Mumps
- Pertussis
- Q Fever
- Rotavirus
- Salmonellosis
- Tuberculosis
- Tularaemia
- Listeriosis
- Rubella