# COVID-19 Prediction and Mitigation

## TI3806 - Bachelorproject

M.R. Meester
I. Görkey
O.J. Braakman
T.T.G. Rood
N. Bauman

# COVID-19 Prediction and Mitigation

## Bachelor Thesis

by

M.R. Meester
I. Görkey
O.J. Braakman
T.T.G. Rood
N. Bauman

at the Delft University of Technology,
to be defended digitally on Thursday July 2, 2020 at 11:00 AM.

Supervisor: Dr. C. Hauff TU Delft
Client: Dr. H. Wang TU Delft

# Preface

This thesis describes the process from research to end-product of the COVID-19 Prediction and Mitigation project as part of the Bachelor End Project (BEP). The BEP is an obligatory project at the end of the third year in order to obtain the Computer Science and Engineering Bachelor's Degree at the Delft University of Technology.

We would like to thank our client Dr. Huijuan Wang for allowing us to work on such a socially relevant, responsible and interesting project. She was always encouraging and enthusiastic to hear about our progress which really motivated us. We would also like to thank Dr. Claudia Hauff, our coach, who guided us through the whole process and always provided valuable feedback.

*Matthias Meester*
*Isitan Görkey*
*Olaf Braakman*
*Thomas Rood*
*Niels Bauman*
*Delft, June 2020*

# Summary

With the world in grasp of the COVID-19 pandemic, models predicting the spread of the virus can give indications to what extent a country is controlling the pandemic. Policymakers can decide to install so-called mitigation strategies to limit the spread of the virus. To aid the decision-making process, this report describes how a web application was created that is capable of visualising predictions on the future course of the virus spread in the Netherlands. Furthermore, the application allows for changing the spread rate of the virus to simulate both mitigation and exit strategies.

Research has been conducted on how we can combine predictions and simulations of mitigation strategies in a single visual solution, in order to aid policymakers. Existing products were analysed in order to get a better understanding of the users' wishes. Design goals were established which have been taken into account when designing and building the software. Furthermore, suitable languages and frameworks for the implementation were chosen.

We have created a tool which both implements a prediction algorithm and visualises the outcomes of this algorithm in a web application. First, a visual design of the product was created after which an accompanying software architecture was established. This design and architecture were then implemented and tested accordingly. Most of the conducted tests were unit tests, but also user tests were performed. During the implementation phase, potential ethical consequences were considered and handled accordingly.

# Contents

# 1

# Introduction

As of 11th March 2020, COVID-19 has been declared to be a pandemic by the World Health Organization (WHO) [1]. The virus disrupts lives, economies and societies [2]. This disruption obliges governments to take action against the virus in order to stop the spread as soon as possible. A possible way to halt the spread of the virus is by putting an entire country on lock-down. However, such a lock-down has far-reaching consequences on, among other things, the economy. To find the right balance between stopping the virus from spreading and minimising other impacts on society, policymakers try to make the best decisions. Countries can install so-called mitigation strategies to limit the spread of the virus. Examples of such strategies are the closing of schools, cancelling large-scale events and limiting travel between regions.

To aid the decision-making process, the client from the Artificial Intelligence and Networking Team of the Delft University of Technology requested the development of a web application. This application should be capable of visualising predictions on the future course of the virus spread in the Netherlands. Furthermore, it should allow for changing the spread rate of the virus to simulate both mitigation and exit strategies. This thesis describes how this tool was created and presents the end product.

The thesis starts by elaborating on the conducted research in chapter 2. The design choices that were made can be found in chapter 3. Chapter 4 goes into detail on how the product was implemented, after which chapter 5 covers how this implementation is tested. Chapter 6 evaluates several processes of the project. Chapter 7 discusses the ethical implications of the product. Finally, in chapter 8 the conclusion can be found and discussion points and recommendations are given in chapter 9.

# 2
# Research

This chapter defines the problem and covers the research that was conducted before the start of the implementation phase. The goal of this research phase was to discover and analyse the needs and wishes of the client and the end-users. Furthermore, design possibilities and tools that are suitable to achieve the set goals are explored.

## 2.1. Problem definition and analysis

With the world in grasp of the COVID-19 pandemic, models predicting the spread of the virus can give indications to what extent a country is controlling the pandemic. If the number of infected individuals surpasses the healthcare capacity of a country, the consequences are deeply undesirable. Thus, a country under the influence of a health crisis must ensure to stay below their healthcare capacity by spreading out the number of infected individuals over a longer period of time. This spreading out can be achieved by putting mitigation strategies in place. Insight into the current and future spread of the COVID-19 virus can help tell to what extent a country is controlling the pandemic. Prediction algorithms can help to display the possible effect of a chosen mitigation strategy, thereby allowing policymakers to evaluate the potential benefits and consequences of their proposed strategies. However, displaying data produced by the prediction algorithm in, for example, a static table makes it hard to observe trends or relations. A more effective way of presenting this data is via visual tools such as maps, graphs or other techniques.

The research phase had to tackle the following problem: “How can we combine predictions with simulations of mitigation strategies in a single visual solution, in order to aid policymakers?”.

## 2.2. Prediction algorithms for epidemics

Multiple studies have been conducted that attempt to model and predict the spread of epidemics, using different modelling methods. One of the methods used is data assimilation or filtering [3]. Here, available data is used to recursively train a simple model. Others use an ensemble of forecasting systems to make their predictions [4, 5]. The idea of using an ensemble of forecasting models, is that they "combine multiple models to obtain a single prediction that leverages the strengths of each model" [5].

This project makes use of the Network Inference Prediction Algorithm (NIPA) [6]. What makes this algorithm different from the ones mentioned above, is its focus on the flow of a population and the impact on the spread of the virus thereof. Several other studies have already attempted to do the same [7–10]. These papers use data on, for example, the number of flights and available seats in the airplane as input to their prediction algorithm [9]. What makes NIPA different is that, unlike the aforementioned studies, NIPA requires none of this input but instead estimates the interactions between cities without prior knowledge.

As input, NIPA requires just the number of new infections per day per region as well as the population size of the corresponding region, as shown in figure 2.1. The algorithm models the spread using the SIR-model [11]: at any discrete time $k$, every individual is in either one of the compartments susceptible (S), infectious (I) or removed (R), modeled as a fraction of the entire population. Next to these fractions, the iterations of this model depend on the curing probability $\delta_i$, which quantifies the fraction of individuals in region $i$ to cure of the virus, as well as the infection probability $\beta_{ij}$ from region $j$ to region $i$.

The infection probability $\beta_i$ for a province is estimated by trying multiple curing probabilities in a predefined search space. With each $\delta_i$ candidate, a system of SIR equations is constructed. The optimal solution to this system of equations also results in the optimal infection probability vector $\beta_i$. The curing probability associated with this best fitting infection probability vector is then selected as the best fitting curing probability. We have then obtained the optimal values for $\delta$ and $\beta$. Once these variables are estimated, predictions can be made for any region $i$ at any time $k$. The details of NIPA's working principles are described in section 4.2.

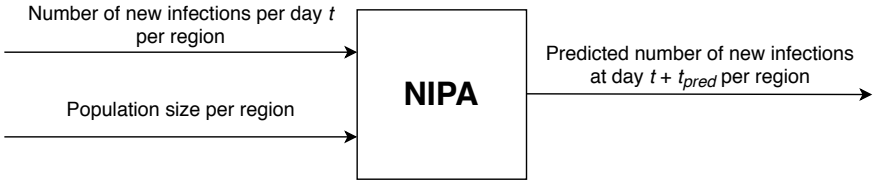



Figure 2.1: NIPA inputs and outputs

## 2.3. Requirement analysis

The first meeting with the client resulted in a list of functional requirements and will be discussed in this section. An overview of this list can be found in appendix C.1.

The product must have a working NIPA implementation as its basis. NIPA is used to calculate a set of infection probabilities ($\beta$ values) between the provinces of a country. The implementation must allow user input of a relative change to the computed $\beta$ values in order to simulate mitigation and exit strategies. It should be noted that such strategies have to be translated by experts into a corresponding increase or decrease of infection probability: the input is not an actual strategy but the numeric change in infection probability that this strategy is estimated to cause.

The start date on which this change of infection probability takes place, as well as the end date on which $\beta$ goes back to the previously estimated amount, can be given as input by the user. Based on these variables, the NIPA results and effects of different $\beta$ values must be visualised through a web application. With these functional requirements, the decision-makers that are not computer scientists nor mathematicians are able to observe the impact of different $\beta$ values visually.

The website of the RIVM is updated with new data on a daily basis. Therefore, the product should have a way of automatically updating the data on which the predictions are based.

If there was time left from the planned ten-week period of the project, with the agreement of the group members and supervisors, could-have features could be implemented. The first feature is the usage of data from other countries to estimate the curing probability. Especially the neighbouring countries, Germany and Belgium, could be part of the input. Secondly, the users could download the product's collected data in a standard format. Lastly, an API could be built to serve a standard central point where researchers worldwide could retrieve the raw data from, as well as the NIPA results.

The product will not include maps or other visualisations for countries other than the Netherlands.

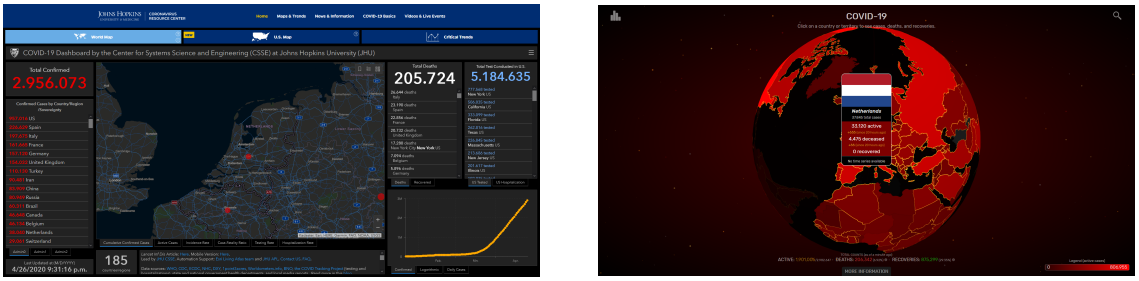
## 2.4. Product analysis

We conducted systematic interviews with the target audience and the client during the research phase. This section describes the analysis of existing products and formalises the opinions of potential end-users on these products. Finally, the provided opinions are combined to shape a general idea of what features the application should consist of.

### 2.4.1. Existing products

We composed a list of online resources which display COVID-19 data over time in a web application. More products were released during the development process, which will be mentioned in section 9.2. The most known example at time of conducting the research is the Johns Hopkins University & Medicine COVID-19 dashboard[1], as seen in figure 2.2a. Another example is the COVID-19 Visualizer[2], by Mamoon N. and Rasskin G., as seen in figure 2.2b. The main difference between these two dashboards is that the Johns Hopkins dashboard focuses on providing as much numerical data as possible, while the COVID-19 Visualizer focuses on making the application visually appealing, only displaying basic numerical data when a country is clicked on.

[1] https://coronavirus.jhu.edu/map.html
[2] https://www.covidvisualizer.com/

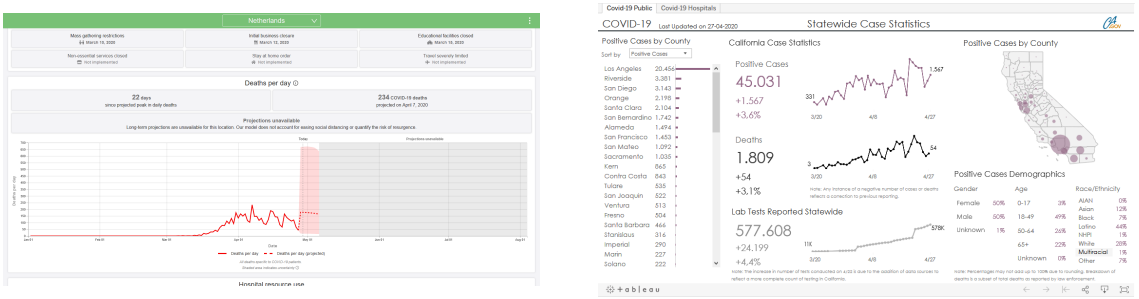



(a) Johns Hopkins University COVID-19 Dashboard

(b) COVID-19 Visualizer

Figure 2.2: Two dashboards taking different approaches visualising the same data

The dashboard from the Institute for Health Metrics and Evaluation (IHME, an independent global health research center at the University of Washington)[3] plots the number of deaths per day and hospital occupation per day over time as seen in figure 2.3a. At the same time, it also projects predictions multiple days ahead. They specify that their predictions are based on a mixed effects non-linear regression framework [12]. Another interesting dashboard is the COVID-19 Public Dashboard for the state of California.[4] This dashboard, as listed in figure 2.3b, is one of the more detailed dashboards we were able to find, as it shows data per county.




(a) IHME COVID-19 Dashboard

(b) COVID-19 Public Dashboard California

Figure 2.3: Two dashboards with each a unique feature: predictions and statistics per ethnicity

There are other dashboards out there, however, they employ similar visual designs and display the same data in the same way as the dashboards mentioned before. In general, these dashboard have shown that data can be displayed in different ways. Some put more effort into making the dashboard visually appealing, while others preferred providing as much information as possible. Both options have to be considered in order to find the most suitable solution for this project.

[3] https://covid19.healthdata.org/netherlands
[4] https://public.tableau.com/views/COVID-19PublicDashboard/Covid-19Public

### 2.4.2. Target audience questionnaire

We composed a list of seven questions, both open and closed, each dedicated to a specific dashboard. To come up with the questions, a closer look was taken at the selected dashboards and features that stood out were written down. Based on the selected features, questions were formulated that asked for the users' opinion. The questions were sent via e-mail. The answers to these questions helped to solidify the goals of the project.

1. Johns Hopkins University COVID-19 dashboard (figure 2.2a):

   (a) Do you think that the amount of information that is displayed at once is too much? If yes, which elements should be left out?

   (b) Would you prefer circles on countries denoting the amount of cases or something else (like a colour based approach)?

   **Motivation:** To establish a product according to our client's and end-user's needs, it is important to know the preferred amount of information that should be displayed on the application. The Johns Hopkins University COVID-19 dashboard displays a large amount of information, such as the number of confirmed infected, dead and recovered per country displayed through tables and maps, and is therefore an ideal example to get to know the user's preferences. Next to that, since the product is likely to contain an interactive map, it needs to be clear how data points are visualised in such a map. The Johns Hopkins example uses circles, which is one of the possibilities to get the user's opinion on.

2. COVID-19 Visualizer (figure 2.2b):

   (a) Do you prefer such a visualisation over the Johns Hopkins one? Meaning, do you prefer a modest user-interface with only the most basic data, over displaying the most possible data.

   **Motivation:** The visual approach of this dashboard is modest compared to the Johns Hopkins one. It uses a globe to project data and allows the user to interact with it. The data display is very minimalistic, meaning they only display basic data, but clicking on a county reveals more details. We were keen to know if users prefer such a minimalistic approach over a cluttered dashboard.

3. IHME COVID-19 dashboard (figure 2.3a):

   (a) Do you like the way the predictions are displayed?

   (b) Do you think the distinction between lines for factual data and the predicted data is clear? If not, how would you like to see this distinction?

   **Motivation:** All other selected dashboards are based on currently available data and do not make any predictions. The IHME dashboard is the only selected dashboard containing such predictions and since visualising predictions is the main focus of this project, the preferred prediction visualisation method needs to be established. This also includes being able to differentiate between factual data and predicted data, for which the final question is included.

4. COVID-19 Public Dashboard California (figure 2.3b):

   (a) Do you like the in-depth visualisation per county or is it too small?

   (b) Do you want current/future tabular data, next to visualisations in maps and graphs?

   **Motivation:** Instead of a zoomed-out visualisation, this dashboard visualises data over many small counties. The displayed data is very specialised per county. Multiple graphs on the side display data which summarise the zoomed-in data. These questions attempt to research the level of depth the visualisation should have.

### 2.4.3. Client's vision - questionnaire outcome

The client's vision is essential since, in the end, the client is the one that should be satisfied with the final product. Besides that, the client also considered herself a target user, so the client's input should be considered as such as well.

- For the first dashboard (figure 2.2a), the client states that there is too much information displayed at once. She suggests the use of a single column containing the total confirmed case and/or total deaths. The client stresses the importance of normalising data when plotting the total number of confirmed cases per province. She suggests that normalisation per province can be used to more easily compare the numbers against the normalised country average. Next to that, the client was fond of the colour scheme used on the RIVM website, consisting of colours ranging between creme and navy blue.
- For the second dashboard (figure 2.2b), the client prefers the use of a static map instead of the more dynamic approach the COVID-19 Visualizer dashboard took. She prefers the more tabular layout with a smaller column of accumulative confirmed cases per province and a map.
- The third dashboard (figure 2.3a) differentiates between the existing data and predicted data by using different line styles and windows. The client agrees with this approach and suggests to not only show the number of confirmed deaths but also include the daily newly infected cases over time.
- For the final dashboard (figure 2.3b), the client stated that the displayed amount of data was excessive. The addition of data display in tabular form is too much, but she agrees on the level of detail present on the map: just province-based data instead of county-based data.

### 2.4.4. End-user's vision - questionnaire outcome

While the satisfaction of the client should be prioritised, the client is in this case not the sole end-user of the product. Therefore, the vision of other potential end-users should be included in the process of decision-making. One possible end-user that was willing to answer the listed questions is a general practitioner active in Delft with interest in epidemiology. He is currently collaborating with the Network Architecture and Services group (NAS)[5] in accumulating and evaluating exit-strategies. His answers to the questions provided a different view on the product.

- The user states that the large amount of information displayed on the Johns Hopkins dashboard is not overwhelming. Next to that, he suggests that a colour based for a map visualisation is preferred over using circles corresponding to the number of infections, as circles are not practical; they can overlap.
- For the second selected dashboard, the user notes that the figures displaying the number of deaths are incorrect: rather than displaying the number of daily reported deaths, the corrected death figures per day should be displayed. Furthermore, since this is the only dashboard that visualises predictions, he puts emphasis on the fact that it needs to be clear how the predictions are made.
- The third dashboard, which consists of a clear user-interface with limited amounts of information, according to the user lacks some information. He does, however, state that, while he prefers more information, another user might not.
- The final dashboard is said to be a bit difficult to understand at first. He suggests to enlarge the map in order for a better user experience. Albeit having to study the displayed information closely in order to understand it, the user does mention that he prefers such detailed information over very little information. Lastly, he mentions that while tabular data is not necessary, it could be a nice feature to add.

[5] https://www.nas.ewi.tudelft.nl/

### 2.4.5. A scientific approach to visualising data

For an effective visualisation, we should look at four questions [13]. These are:

1. What type of knowledge needs to be visualised?
2. Who is being addressed?
3. Why should knowledge be visualised?
4. Which is the best method to visualise this knowledge?

Since question 1 to 3 have already been answered in previous sections, the focus of this section is on question 4. To answer question 4, we have to discover the most optimal way to visualise the predictions over the course of the pandemic. A successful visualisation with the goal of transferring knowledge has to adhere to several properties [14]:

1. It captures and depicts objective knowledge
2. It offers insights and relations between those
3. It is a visual representation
4. It encourages people to converse about the data depicted
5. It is flexible, such that changes in insights can be incorporated
6. It is communicable; knowledge can be transferred to others who were not involved in the makings
7. It leads to new discoveries and is useful to viewers

Having incorporated these aspects, as well as looking at the existing products, the following type of visualisations have been included in the product.

- **Line diagrams** with information on the number of (predicted) cases. It should be noted that these need to have several properties to be effective [15]:
  - A fitting caption
  - Axes
  - Scales
  - Symbols
  - Data field
  - Good contrast between foreground and background

  To successfully differentiate between predicted data and known data, we should display a good contrast between the two types of data. This can be achieved, for example, by using different colours and a different line type for each of the data types.
- **A map of the Netherlands** with colours representing the amount of (predicted) cases. Three out of the four products analysed in section 2.4.1 contain a map and a map visualisation also adheres to all of the properties previously mentioned.

To adhere to the principles listed above, we took some extra points into account. Of the earlier mentioned properties, 5 and 6 were important points to take into account when making the user interface. If these two are not clear and concise, users might not understand how parameters influence the outcome of the figures.

### 2.4.6. Final product vision

As part of the research phase, the questions that were asked to the client have shown that the client expects a prediction visualisation tool with a simple interface that focuses on the indicators of SIR, namely the fractions of susceptible, infected and, removed individuals. When the same questions were asked to a possible end-user, who is a medical doctor, he indicated that he prefers seeing as many details as possible instead of a to-the-point interface that shows fewer details.

As can be seen from the answers of the client and the possible end-user, there is a conflict between the two. One expects a simple interface while the other would like to see as much information as possible. It is, however, important to keep in mind that the end-user that has been interviewed is a medical doctor interested in epidemiology. Due to time limitations and the policymakers being busy in the current circumstances, it was not possible to interview an actual policymaker as another type of end-user.

Considering the requests of the clients and the wishes of the end-users, a strategy to follow could be building a customisable interface. The initial view of the product shall, in this case, look like a minimalist tool with a visualisation on the map, slider for the $\beta$ factor representing the spread rate, and another slider for dates. On this minimalist design, there could be an option-menu where the interested users could view a selection of a broader set of data that could be useful for different purposes.

With such a solution, not only policymakers that are focused on the consequences of their actions have a simple interface, but scientists can reach more detailed information to use in their research. This customisable interface, however, was listed as a "nice to have" feature as the project prioritised the wishes of policymakers.

The product requirements in appendix C.1 lists a "nice to have" feature of an API that could be built to serve a standard central point where researchers worldwide could retrieve predicted data and existing infection data.

## 2.5. Design goals

This section describes the design goals of the product. These are the goals that have mainly been taken into account when designing and building the software. During the development process, the focus has been on these design goals. Without taking these into account, the application would not have ended up as the client originally intended.

### 2.5.1. Performance

Optimised performance, meaning that the algorithm runs quickly, is strongly needed because of two reasons.

Firstly, users need to be able to tweak the $\beta$ parameter of the algorithm, meaning that a part of the algorithm has to run on the spot, whilst the user has to wait. The goal is to minimise this waiting time as much as possible.

Secondly, as the product to develop is a web application, multiple users could access the application at the same time. The server hosting the application needs to be able to handle all of the incoming requests at a high speed.

### 2.5.2. Maintainability

As a goal of the project is to make an open-source product which can be modified by different researchers, the product needs to be easily adaptable to become useful to someone with different needs. An example would be a researcher using a different type of data or a different prediction algorithm. This is achieved in two ways. Firstly, the code is written in well-known languages, which will be further discussed in section 2.6. Next to that, the code is well documented. Lastly, the program is split up into different replaceable modules, with separate modules containing the prediction algorithm and the front-end implementation.

### 2.5.3. Usability

The program should be easy to use. This is achieved by having an intuitive user interface, as well as a clear and concise explanation. Additionally, documentation on how to use the program is provided.

### 2.5.4. Visual appeal

While the main focus of the project is on the visualisation of the computed predictions, it is of essential value that the web application is visually appealing. Previously, different possible ways of visualising the predicted data were discussed. A selection of visualisation methods was made as a result of the users' wishes and the

conducted research. It is now one of the main goals to take the visual appeal into account when implementing the selected methods, to ensure satisfactory completion of the project.

## 2.6. Development tools

So far, the users' wishes have been established and formalised. Now, a closer look needs to be taken into how these requirements are going to be implemented. This section motivates the choice of programming languages as well as the choice of front-end and back-end frameworks.

### 2.6.1. Back-end language/framework

In 2020 there are a lot of back-end frameworks to choose from. Some of the back-end framework we considered are:

- PHP - Laravel[6]
- Python - Django[7]
- Java - Spring[8]
- Ruby - Ruby on Rails[9]
- NodeJS - Express[10]

The project requirements do not contain exceptional features which are framework specific. Building an API is a could-have feature, but all frameworks allow the creation of an API. The remaining criteria, which we considered most important when choosing the back-end programming language, were:

- Ease of implementation of NIPA
- Personal experience
- Popularity (large open-source community)

Furthermore, we are aware of the fact that any of the aforementioned languages can implement the algorithm, but we aim to stick with the most accessible language for data science and the open-source community. Some of the team members already had experience with both the Express and Laravel framework. Because of that, we wanted to challenge ourselves by choosing a new and unknown framework. Therefore, we chose the Django framework for Python. Python offers mathematical libraries such as Pandas[11], Numpy[12], and Sklearn[13] which ease the efforts of implementing NIPA.

### 2.6.2. Front-end language/framework

There is a vast number of front-end frameworks available. For front-end framework selection the considered frameworks are:

- ReactJS[14]
- Angular[15]
- VueJS[16]

All of the above front-end frameworks provide the same basic functionality: a framework that makes it easy for the user to create a web application. We chose Angular, as that is the only one that natively comes with TypeScript. TypeScript enforces data typing, which in turn will allow for catching errors earlier on in the debugging process.

[6] https://laravel.com/
[7] https://www.djangoproject.com/
[8] https://spring.io/
[9] https://rubyonrails.org/
[10] https://expressjs.com/
[11] https://pandas.pydata.org/
[12] https://numpy.org/
[13] https://scikit-learn.org/
[14] https://reactjs.org/
[15] https://angular.io/
[16] https://vuejs.org/

**Visualisation**
The application visualises maps and graphs to the user. Implementing these visualisations from scratch would be like reinventing the wheel. Therefore, we considered multiple libraries that offer these types of visualisations:

- JavaScript - Leaflet[17]
- JavaScript - AnyMap[18]
- JavaScript - D3.js[19]

As none of us had ever worked with any of these libraries or even tried to visualise a map in some form, the decision was made to just try and play around with three of the above.

The first impression on Leaflet was not too bad. However, some difficulties were encountered trying to customise certain things, for example, only showing the Netherlands instead of the entire world. Next to that, Leaflet itself does not provide any functionality for visualising graphs, which would mean yet another library would have to be found for that as well. For these reasons the decision was made to first check out the other libraries, hoping that they would provide more functionalities. The second library that was tested was AnyChart's AnyMap. This looked even more promising at first, as a map of just the Netherlands was displayed pretty quickly. Next to that, its parent product AnyChart provides a lot of functionality for displaying graphs. Sadly, however, it was noticed that a trial version was used with a watermark in the map itself, meaning their product had to be paid for. The client made clear at the very beginning of the project that she had some funding available for this project, so this would not necessarily pose a problem. The decision was made, however, to keep on searching for another library, as one of the goals of the project is to be open-source, which means not using any paid products. Finally, we implemented the D3.js script, and after being able to visualise the map of the Netherlands pretty quickly, we started toying with adding other visualisations to the map. Immediately it was agreed that this library was the best one so far. Just like the previous library, D3.js also provides extensive support for displaying graphs, which saves us from finding an additional library.

After searching the internet for alternative visualisation libraries, the decision was made to use D3.js for the project, as it offered a great amount of customisation while still taking a lot of work out of our hands.

## 2.7. Discussion

As part of the research into the users' needs and vision, we created a list of questions to find out what is essential to the client and existing dashboard solutions. The interviews with the client and an end-users have been useful for deriving the requirements. We would have preferred to interview more potential end-users, but the availability of these potential end-users was limited during the interesting times dominated by COVID-19. After review, the questions are biased towards our perception and could therefore have been more objective.

## 2.8. Conclusion

To conclude, this research chapter looked into the problem of effectively predicting and visualising the spread of a pandemic, in order to assist policymakers and scientists. To answer the question stated in section 2.1, we first researched several existing dashboard solutions and based on those, formulated a number of questions regarding each of them. We then interviewed two potential end-users and compared their visions and opinions. In the end, the questioned users had a conflict of interest. From this conflict of interest, we concluded that dashboard flexibility, in a sense that a user can define the dashboard layout, could also be added to the list of "nice to have" requirements. Besides these interviews, we also conducted literature research into the scientific most optimal way of visualising data. This research resulted in some general rules of thumb that we should try to stick to when developing the dashboard. We decided to use Django, written in Python, as the back-end framework and decided to use Angular, written in TypeScript, as the main front-end framework. Furthermore, we use the D3.js library as the main tool for visualisation.

[17] https://leafletjs.com/
[18] https://www.anychart.com/products/anymap/overview/
[19] https://d3js.org/

# 3

# Product design

The problem, as described in section 2.1, made clear that several challenges had to be thought through before implementation of the product could begin. This chapter establishes these challenges and elaborates on the solutions found. First, the challenges related to visual design are discussed in section 3.1, after which challenges related to software architecture will be discussed in section 3.2.

## 3.1. Visual Design

The first point of attention was the importance of visualisation. Since the primary goal of the project was to aid policymakers, the dashboard should be both structured and user-friendly. For these requirements to be realised, a visual design should be created and iterated upon. The procedure of selecting visual elements will be explained in section 3.1.1. Then, the user input section is reviewed in section 3.1.2 after which the selected visual elements are discussed and explained in section 3.1.3 and section 3.1.4. Finally, the additional pages are examined in section 3.1.5.

### 3.1.1. Visual element selection

The currently available infection data combined with the predicted infection data can be displayed in numerous ways, each highlighting different aspects of the virus' spread. The interviews with the client as well as potential end-users, described in section 2.4, showed that there is a clear preference for two of these aspects: the long-term spread of the virus and the present-day situation combined with the expected short-term spread. To fulfill their needs, the conducted research resulted in the inclusion of a graph, to display long-term data, and a map, to display current and short-term data. After taking these aspects to the drawing board, the design as can be seen in figure 3.1 was created.

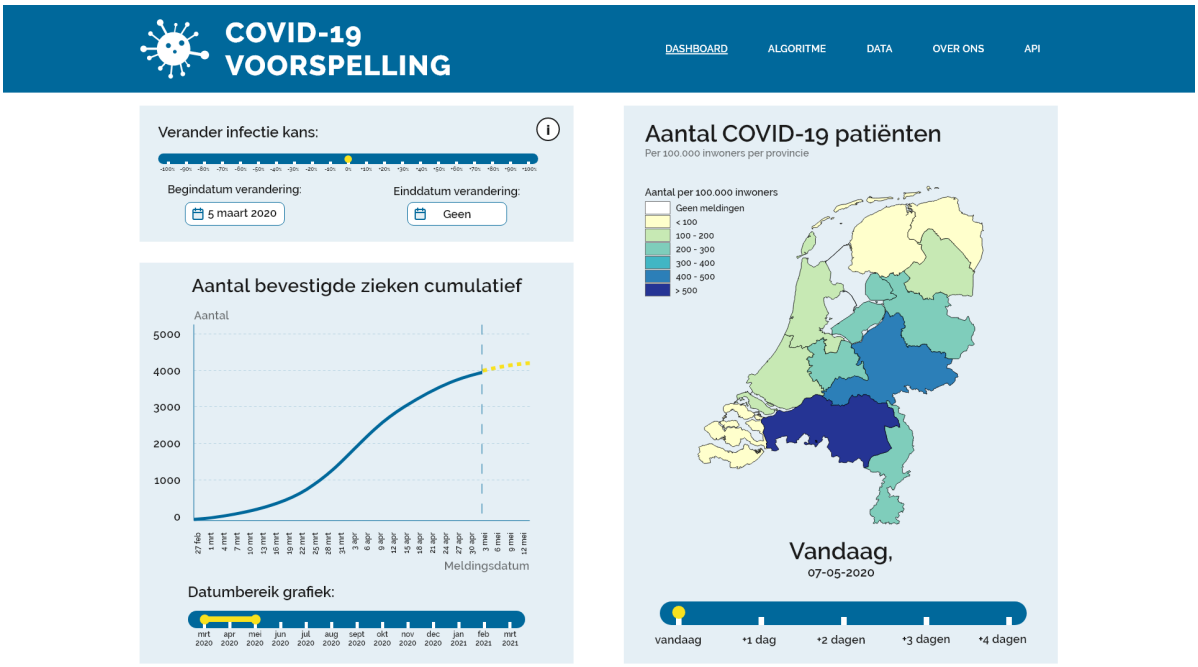



Figure 3.1: The initial visual design of the dashboard. On the top left the user input component (section 3.1.2), below it the graph component (section 3.1.3) and on the right the map component (section 3.1.4)

### 3.1.2. User input component
The upper left component of the design in figure 3.1 is the user input section of the dashboard. This component is required such that a user can alter some of the parameters of the prediction algorithm. The slider allows for changing the infection probability relative to the infection probability estimated by the prediction algorithm. The two date pickers can be used to control the start and end date of this change in infection probability. Furthermore, to explain the complex principle of 'changing the infection probability', a tooltip is added. This tooltip displays a dialog when clicked on, explaining how the input component works.

### 3.1.3. Graph component - long-term visualisation
The bottom left section of figure 3.1 contains the graph, which displays the cumulative confirmed number of infected individuals in the entire country at a given date. The idea behind the incorporated slider is that it allows a user to change the date range of the x-axis from a single month up to a year, this way allowing for a long-term overview. To clearly indicate which data is factual and which is predicted, a distinction is made within the colour and type of line inside the graph: factual data is represented by a solid blue line, while predicted data is represented by a dashed yellow line.

### 3.1.4. Map component - short-term visualisation
The right compartment of the design in figure 3.1 is the map component. This map displays the current situation per province combined with a short-term prediction. Contrary to the graph, which shows absolute accumulated data, the map shows normalised data per province. It does so by presenting the number of infected inhabitants per province per 100,000 inhabitants on a colour based approach: the darker the colour, the more normalised newly infected individuals per day. The slider allows the user to see the predicted situation on a specific day. This, in turn, allows the user to see what kind of state each province is in and thus observe in what way the virus is predicted to spread to different provinces.

### 3.1.5. Information pages
Additionally, the header included in the design contains navigation elements to several information pages. One of these pages summarises the prediction algorithm used, including a reference to the paper used. More explicit data should be displayed on a separate 'data' page, for users that prefer numbers over visualisations. Furthermore, a general introduction to the creators of the website is provided on an 'about us' page, which should also include references to the client, coach, and general purpose of the end-project. Lastly, an API documentation page should be provided, explaining the available data endpoints. Since these pages consist of mostly static content and the design of the dashboard has priority over the rest, it was decided not to create visual designs for any of these pages beforehand.

## 3.2. Software architecture
The software architecture of the application was split up in a back- and front-end component. While designing the architecture of these components, the visual design was taken into account. In section 3.2.1 a general overview of the back-end structure is given, after which section 3.2.2 describes the front-end architecture of the application.

### 3.2.1. Back-end
First, several decisions had to be made regarding the back-end architecture. In order for the estimated infection probabilities to be the most accurate, the prediction algorithm relies on the most up-to-date infection data. The RIVM publishes new data on a daily basis, which will then serve as the updated input for the prediction algorithm.

There are multiple ways to allow the front-end of the application to access the predicted data. The two options that were considered are client-side data collection and prediction versus server-side data collection and prediction. Prediction, in this context, means the estimation of infection and curing probabilities. The final iteration that uses these estimated values to calculate the infected fraction of inhabitants is performed in the front-end of the application, as will be discussed in section 3.2.2. The advantages and disadvantages of both options were evaluated, after which a decision was made.

- **Client-side**
  The first possibility is to run both the data collection and the prediction algorithm client-side. The advantage of this option would be that no back-end server is required, thereby improving maintainability since only static JavaScript and CSS files have to be served. The disadvantage, however, is that the web application could become slow since the data collection and prediction calculations would have to be run every time the page is requested.

- **Server-side**
  The second option is to run the data collection and the prediction algorithm server-side. The collected data, as well as the output of the prediction algorithm, should be stored such that these do not have to be collected and computed every time a user enters the website. This option will provide the fastest loading time to the user, as a request to the back-end only means returning the stored output of the algorithm. The disadvantage of this option is that setting up a back-end server is more complex than hosting static files.

If client-side computation was chosen, the performance of the application would be dependent on the run-time of the data collection and the prediction algorithm. Since the efficiency of the provided algorithm was unknown at the time and long loading times were undesirable, this option seemed less suitable for the project. The second option, despite requiring more time to set up, seemed more appropriate. Since some members of the group had experience with setting up such servers, this would not pose a problem. In the end, it was therefore decided that both the data collection and prediction algorithm would be run server-side.

Now that the decision was made to run the data collection and prediction algorithm server-side, the best method for storing the results had to be established. The two methods of storage that were considered are file storage and database storage. The advantages and disadvantages of using either will be listed below.

- **Database storage**
  One of the advantages of using a database is data filtering. A database is optimised to retrieve a selection of data based on certain criteria. However, a disadvantage of this is that it introduces overhead and complexity: such optimisation does not come for free.

- **File storage**
  An advantage of file storage is that it is simple. Unlike database storage, file storage does not require any setup. A disadvantage of file storage is that files are not suitable for frequent look-ups, since unlike database storage, no meta-data is available to aid the look-up efficiency.

Since the application will always require all the available data in sequential order, there is no advantage in optimised retrieval of a selection of data as provided by database storage. Database storage would therefore only introduce unnecessary overhead. Although files are not suitable for look-ups, this will not pose a problem as no look-ups have to be performed by the application. Therefore, the decision was made to go with the simplicity of file storage. Additionally, the files should be managed by a separate entity, a file manager.

The predicted data should then be provided to the front-end through the means of an API. The API also has to be accessible to anyone that wants to use the computed data for their own purposes. Finally, it was agreed upon to split the back-end design into the following four distinct parts:

- Data collector
- File manager
- NIPA
- API

Firstly, raw data is fed into a data collector, which collects and formats this data into usable infection data for the NIPA algorithm. Consequently, the formatted data is fed into a file manager, which manages the storage and retrieval. The stored data is then fed into the NIPA algorithm, after which the algorithm's output is stored by the file manager. Finally, this output is made available to the front-end via an API. This API also functions as a point of reference that other researchers can use to retrieve our data. The final back-end structure can be seen in figure 3.2.

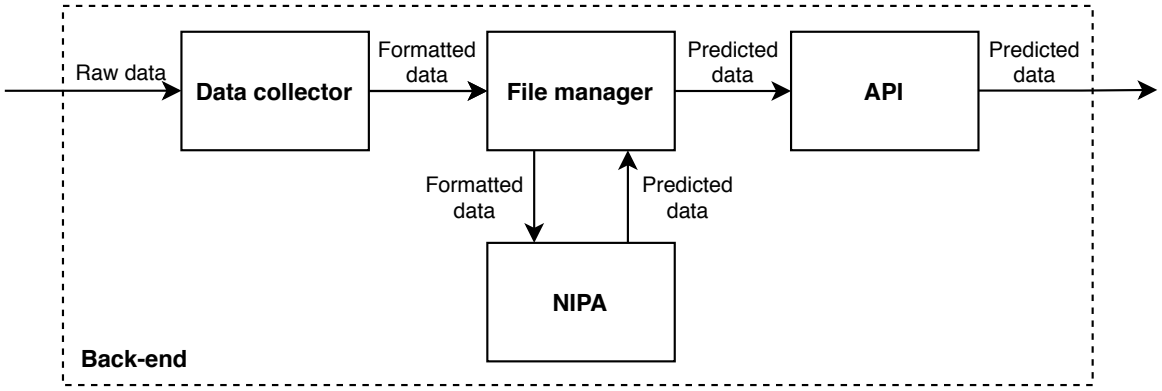



Figure 3.2: Overview of the back-end structure.

### 3.2.2. Front-end

Together with the back-end architecture, the front-end architecture was designed. It was realised that in order to provide the best user experience, the final iteration to calculate the infected fraction of inhabitants had to be performed client-side. The reason for this is that user input of parameters required reiteration over the predicted data as retrieved from the back-end.

An alternative could be to perform all computations server-side. However, this was not deemed a viable solution as it would either require precomputing infection values using all possible combinations of infection probability and date ranges, or making a request to the back-end each time a user changes any of the parameters. The first is not viable, since an infinitely large amount of date ranges can be picked, leading to an infinite amount of combinations of infection probability and date ranges. Although the maximum end date could be capped at one year from now, this would still lead to greatly increased computation time and file sizes. The second is not preferred because we assumed that having to make a back-end call for each change in parameter would be slower than performing the final iteration in the front-end.

Furthermore, the different visualisation components largely require similar data. To avoid duplicate computations from being made, there should be a general point of access that always does each computation only once. Every time a user inputs different infection probabilities, the data is recalculated and automatically shared amongst all components. Taking all of these requirements into account, the front-end structure, as seen in figure 3.3, was created.

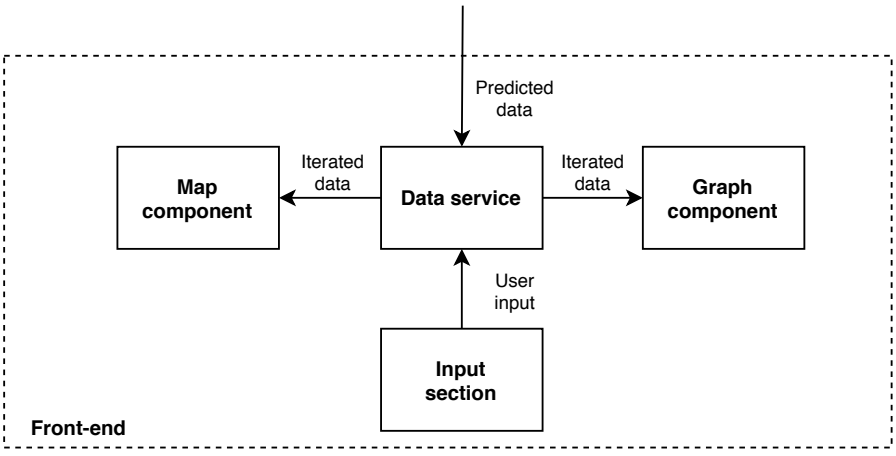



Figure 3.3: Overview of the front-end structure.

# 4
# Implementation

This chapter covers the implementation of the software architecture and visual design as described in chapter 3. Section 4.1 covers the implementation of each of the back-end's components. Since NIPA occupies such a large part of the back-end, it was decided to elaborate on it in a separate section, namely section 4.2. At last section 4.3 covers how the front-end structure and visualisations are realised according to the design.

## 4.1. Back-end

In section 3.2.1 four individual back-end parts are mentioned. The following subsections will cover how each of these parts works and elaborate on how they are connected to each other.

### 4.1.1. Data collector

As NIPA requires infection data, there is a need for a data source. NIPA requires the fraction of newly infected people per province per day. This means that two types of data are needed: the number of inhabitants per province, and the amount of newly infected people per province. If we divide the number of newly infected people per province by the number of inhabitants in that province, we get the required fraction for NIPA.

There are two types of data-sources which we have considered using: static data and daily updated data. The first being saving and formatting the data manually once and then using that data for all the predictions. The second being building a component which collects and processes infection data, both automatically. We have decided to go with the second approach, as new infection data is registered daily. Thus the application is kept up-to-date with new data every day, which is necessary to make the most up-to-date predictions. We came up with the following requirements for the data source:

- Must be updated daily
- Must at least have infection data per province
- Must be accessible through an API
- Must match RIVM data, as listed on their website [1]

It was hard to find a data source that complies with all of these requirements, as most data sources only had data for the country of the Netherlands as a whole instead of listing infection data per province.

However, there is one source that does suit our needs. This data source is called "NL COVID-19 Hub" and is provided by Esri[2] on the platform ArcGIS[3]. The ArcGIS endpoint contains two different metrics: the total number of inhabitants per municipality and the total number of reported COVID-19 cases per municipality. We have confirmed that this data source is the same as used on the RIVM website[4] in their own maps. This was done by downloading the infection data listed on the RIVM page about COVID-19 and downloading the data they used for the map of infections per day per municipality[5]. This data exactly matched the data which the data collector received from ArcGIS, and thus the data collected from ArcGIS is correct.

The data contains infections per municipality per day, which later can be grouped into province data based on the municipalities' respective provinces. However, there have been two major problems with this data-source.

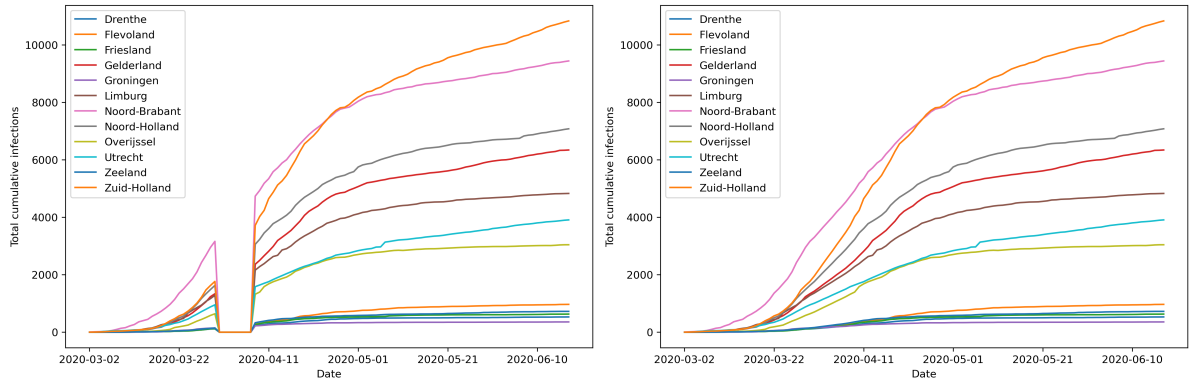
The first problem is that there is a whole week of missing infection data in this data-set. We have tried to find a source which has these missing data points but have not been successful in doing this. We have thus decided to interpolate missing data points per municipality. The method used to interpolate is called linear

[1] https://www.rivm.nl/coronavirus-covid-19/actueel
[2] https://www.esri.nl
[3] https://www.arcgis.com/
[4] https://www.rivm.nl
[5] https://www.rivm.nl/coronavirus-covid-19/actueel#kaart

interpolation, which is the most basic form of interpolation and takes a weighted average of the closest $y$ values. The reason why this method is used is that this method seemed to fit the curve best. Other methods like cubic or quadratic interpolation resulted in a fairly linear result as well, making us decide to just stick with the simplest method. To get the $y$ value at point $x$, one would take the left $(x_l, y_l)$ and right $(x_r, y_r)$ closest known points, and then solve the following equation:

$$y = y_l(1 - \frac{x - x_l}{x_r - x_l}) + y_r \frac{x - x_l}{x_r - x_l}$$

The result of applying this interpolation to the missing data points is shown in figure 4.1.




(a) Non-interpolated infection data

(b) Interpolated infection data

Figure 4.1: Result of interpolating missing data points

We assume these data points will not be given by the RIVM or ArcGIS anymore. At the time of writing this, the data from this week has been missing for almost three months. In the web application, there is an information box which notifies that this specific week's data is interpolated. The data collector ensures that potential missing data in the future is interpolated too. However, there is one problem with this, as there has not been enough time to implement a flag for this data. This flag could be used so that the information box in the front-end notes that these days are interpolated too, instead of only the one specific week. Researchers then also know for sure which dates to trust. This could potentially be a feature for the future.

The second problem arose when ArcGIS changed its output format. It, for example, changed the format of the data and added/removed certain fields. The data collector now tries to account for a change in data types as much as possible. Now, if a return type is altered, the data collector is less likely to crash. This is done by having checks such as, if a date is returned with a timestamp appended, this timestamp is removed. However, there were also good aspects of ArcGIS changing their return format. At first, we had to collect inhabitant data from the Centraal Bureau voor de Statistiek (CBS)[6]. But later on in the project, ArcGIS started returning inhabitant data as well, making the CBS data collection redundant. We have thus removed this part from the data collector.

After interpolating the municipality data, municipalities are grouped by their respective provinces. Then a column denoting new infections per day is appended, by taking the difference of cumulative infections of each day with the previous day. A snippet of input and output of the data collector can be seen in figure 4.2. This figure illustrates what kind of grouping and formatting is done.

[6] https://www.cbs.nl

| | Datum | Provincie | Bevolkingsaantal | Meldingen | Gemeentenaam |
|---|---|---|---|---|---|
| 0 | 2020-03-02 | Drenthe | 19460 | NaN | Westerveld |
| 1 | 2020-03-02 | Drenthe | 24327 | NaN | De Wolden |
| 2 | 2020-03-02 | Drenthe | 25453 | NaN | Aa en Hunze |
| 3 | 2020-03-02 | Drenthe | 25558 | NaN | Borger-Odoorn |
| 4 | 2020-03-02 | Drenthe | 31242 | NaN | Noordenveld |
| ... | ... | ... | ... | ... | ... |
| 38335 | 2020-06-17 | Zuid-Holland | 119300 | 528.0 | Dordrecht |
| 38336 | 2020-06-17 | Zuid-Holland | 125174 | 168.0 | Leiden |
| 38337 | 2020-06-17 | Zuid-Holland | 125283 | 215.0 | Zoetermeer |
| 38338 | 2020-06-17 | Zuid-Holland | 545163 | 1692.0 | 's-Gravenhage |
| 38339 | 2020-06-17 | Zuid-Holland | 651376 | 2594.0 | Rotterdam |

38340 rows × 5 columns

(a) Input data collected from ArcGIS

| | province | date | number_of_inhabitants | total_infected | newly_infected |
|---|---|---|---|---|---|
| 0 | Drenthe | 2020-03-02 | 493657 | 1 | 1 |
| 1 | Drenthe | 2020-03-03 | 493657 | 2 | 1 |
| 2 | Drenthe | 2020-03-04 | 493657 | 2 | 0 |
| 3 | Drenthe | 2020-03-05 | 493657 | 3 | 1 |
| 4 | Drenthe | 2020-03-06 | 493657 | 3 | 0 |
| ... | ... | ... | ... | ... | ... |
| 1291 | Zuid-Holland | 2020-06-13 | 3708585 | 10664 | 53 |
| 1292 | Zuid-Holland | 2020-06-14 | 3708585 | 10705 | 41 |
| 1293 | Zuid-Holland | 2020-06-15 | 3708585 | 10749 | 44 |
| 1294 | Zuid-Holland | 2020-06-16 | 3708585 | 10797 | 48 |
| 1295 | Zuid-Holland | 2020-06-17 | 3708585 | 10837 | 40 |

1296 rows × 5 columns

(b) Return data of the data collector

Figure 4.2: Input and output of the data collector

### 4.1.2. File management

To serve as a bridge between all the back-end components, a file manager is implemented. This manager exists such that output from the data collector and the NIPA algorithm can be saved and retrieved from a single point. Their output is saved and instantly available to the front-end, instead of having to be calculated on the spot, which takes on average about 15 seconds. All relevant variables produced by NIPA for the front-end are saved in a JavaScript Object Notation (JSON) file. In this manner, a final front-end calculation can be done solely based on this file. The output of the data collector, which can be seen in figure 4.2b, is directly saved into a Comma-Separated Values (CSV) file.

### 4.1.3. API

One of the requirements of the client was that the product could serve as an endpoint for other researchers to get data from, both prediction-wise as well as raw data used for these predictions. To account for this need, an Application Programming Interface (API) is integrated into the product. This API serves both as a bridge between the front- and back-end and as a reference point for researchers. The API can be used to get data of both NIPA and infection data from the back-end, to be used elsewhere. The API complies with the Representational State Transfer (REST) architecture [16]. There is also a dedicated web page that documents all of the available API endpoints. The use of these endpoints is explained below:

- **NIPA Data**: Returns the NIPA results in JSON format, containing all the relevant variables needed for the front-end, as mentioned in 4.1.2.
- **Infection Data**: Returns the formatted infection data in CSV format, as mentioned in 4.1.2.
- **NIPA I Hat**: Returns the NIPA î-matrix, which contains the predicted fraction of new infections per province per day for the coming 70 days, in CSV format.
- **Infections per day per province**: Returns the infections in a table format instead of a list, can be set to return cumulative infections or to return new infections per day, in CSV format.
- **Number of inhabitants per province**: Returns in table format the number of inhabitants per province per the last noted date, in CSV format.

### 4.1.4. Cron jobs

As referenced in section 3.2.1, infection data should be collected on a daily basis. To realise this, so-called *cronjobs* are scheduled. Cronjobs are UNIX commands that are scheduled to be ran at certain times or intervals.

For this project, a cronjob is scheduled that first collects the most up-to-date infection data from ArcGIS, after which it runs the prediction algorithm on the collected data. Since the time on which ArcGIS publishes new data varies, the job is run every hour. This way, new data is always collected within an hour of its release, resulting in predictions that are as up-to-date as possible.

## 4.2. NIPA

The project description (Appendix B) requests the NIPA [6] to be used to do the predictions. This section is dedicated to introducing the reader to the NIPA. It presents the key insights for understanding the algorithm and why it works. It then compares the implementation to the MATLAB implementation provided by the authors. Finally, it also mentions implementation-wise obstacles we came across.

### 4.2.1. Algorithm description

The NIPA paper splits the algorithm up into three distinct parts:

1. Data pre-processing
2. Network inference
3. SIR iteration

Each part was originally implemented independently. The first and third step were completed without much problem. The second part, the network inference, consumed lots of hours of debugging and understanding the principles behind the NIPA. The data pre-processing part in the NIPA paper performs interpolation over missing data and smoothing of the data. We did have to perform some data interpolation, but this is all done in the data collector (4.1.1). We do not consider the smoothing of the data to be necessary since the data obtained does not contain any strange outliers. Furthermore, the paper does not motivate the reasoning for the smoothing and neither specifies a window size for the rolling average.

### 4.2.2. SIR decomposition

Assume an optimal curing probability $\delta_i$, ultimately the goal is to obtain an infection probability vector $\beta_i$ per province $i$. For a while we struggled to understand how this vector was obtained. The key insight for us was the decomposition of the SIR Epidemic Model (D.1, D.2, D.3). The paper mentions the construction of matrix $F_i$ and vector $V_i$ as described in equation D.4. Initially the derivation of this matrix and vector was vague to us, until the link with the SIR equations was made. As an example to show why it is possible, the first row of the $F_i$ matrix and $V_i$ vector is obtained as follows. From the original equation D.1, the first term can be moved to the left side of the equation, resulting in:

$$I_i[k+1] - (1-\delta_i)I_i[k] = (1 - I_i[k] - R_i[k]) \sum_{j=1}^{N} \beta_{i,j} I_j[k] \tag{4.1}$$

Then, by substituting equation D.3 in the previous equation the following equation is obtained:

$$I_i[k+1] - (1-\delta_i)I_i[k] = S_1[k] \sum_{j=1}^{N} \beta_{i,j} I_j[k] \tag{4.2}$$

It is immediately clear that the left side of this equation is equal to the $V_i$ from equation D.4 for all $k$ from $1 \dots n-1$. What remains is rewriting the summation on the right side of the equation to:

$$S_1[k] \sum_{j=1}^{N} \beta_{i,j} I_j[k] = S_i[1](I_i[1]\beta_{i,1} + \cdots + I_N[1]\beta_{i,N}) = S_i[1]I_i[1]\beta_{i,1} + \cdots + S_i[1]I_N[1]\beta_{i,N} \tag{4.3}$$

When taking the general form for all $k$ from $1 \dots n-1$ of this equation, the whole equation can be written in a matrix vector multiplication of $F_i$ and $\beta_i$ which results in equation D.5. This concludes why the rewrite of the original SIR traces to the $F_i$ matrix and $V_i$ vector works. It thus describes how a row in the $F_i$ matrix is related to the same row in the $V_i$ vector. This insight helped us better understand the inner workings of the algorithm and was helpful during the implementation of the k-fold cross-validation (4.2.4) later on.

### 4.2.3. Estimating the beta vector

The key to solving the set of linear equations, as described in equation D.5, is the usage of the Least Absolute Shrinkage and Selection Operator (LASSO) [17]. The authors of the NIPA paper propose to obtain the $\beta_i$ vector by solving the LASSO problem as described in equation D.6. Solving a LASSO problem in the general

form with Python is not a problem since the *sklearn* package provides a LASSO solver[7]. However, the NIPA penalisation term differs from the general form of the regularisation term. The standard form of the regularisation term ($\rho_i \sum_{j=1}^{N} \beta_{i,j}$) sums all elements in the solution vector, whereas the NIPA regularisation term ($\rho_i \sum_{j=1, j\neq i}^{N} \beta_{i,j}$) excludes the self-infection probability $\beta_{i,i}$. Therefore solving the LASSO was not straightforward. Looking for a solution, we resorted to the MATLAB implementation for inspiration. The MATLAB code included a custom written LASSO solver which rewrites the LASSO problem to a constrained linear least-squares problem. We took inspiration from this implementation and translated the small piece of MATLAB code to our Python implementation. To solve the constrained linear least-squares problem, the MATLAB code made use of the *quadprog*[8] function. This function allows enforcing lower and upper bound constraints on the vector solution as function parameters. This is especially useful for constraining $\beta_i$ as follows: $0 \leq \beta_i \leq 1$. Finding a *quadprog* Python equivalent was initially not as straightforward as was hoped. After researching, we came across the *qpsolvers*[9] library by Stephane Caron. This is a package which acts as a wrapper for a multitude of quadratic programming solvers in Python. However, none of these suitable solvers had the ability to specify lower and upper bound constraints on the solution vector. Thus we had to come up with our own solution to the problem.

The standard form which quadratic programs solve is shown in equation 4.4:

$$\begin{aligned} \underset{\beta_i}{\text{minimise}} \quad & \tfrac{1}{2}\beta_i^T P \beta_i + q^T \beta_i \\ \text{subject to} \quad & G\beta_i \leq h \\ & A\beta_i = b \end{aligned} \tag{4.4}$$

With help from a detailed blog post [18] from again Staphene Caron, we managed to encode the lower and upper bound constraints in the $G$ matrix and $h$ vector. For sake of the example, imagine the problem is applied to three provinces ($\|\beta_i\| = 3$), then the constraints $0 \leq \beta_i \leq 1$ can be encoded as follows:

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \qquad h = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad G \begin{bmatrix} \beta_{i,1} \\ \beta_{i,2} \\ \beta_{i,3} \end{bmatrix} \leq h$$

Since the solver makes sure the equality $G\beta_i \leq h$ will always hold, the boundary conditions can thus be enforced on the $\beta_i$ vector. Approximately a week after completing the implementation of the algorithm, one of us noticed by chance that the *qpsolvers* library was updated to a newer version (May 16, 2020 version 1.3), which does allow to define lower and upper bound constraints without having to encode those manually. We decided to scrap our own lower and upper bound implementation and use the new functionality which was added to the library mainly due to better performance. It also reduced unnecessary complexity in the algorithm, making it easier to understand for anyone looking at the code.

### 4.2.4. k-fold cross-validation

Overfitting is a common mistake in training an algorithm. The authors of the NIPA paper therefore propose 3-fold cross-validation to be the suitable approach to obtain the optimal $\beta$ matrix. The MATLAB implementation uses the *cvpartition* function. This function generates $k$ validation and training logic arrays which are used in order to obtain validation indices for the given fold for the $F_i$ and $V_i$ matrices. The logic arrays are constructed by setting two-thirds of the array with ones (true) and the others with zeros (false) randomly. A simple k-fold cross-validator for Python is the *KFold* class from the *sklearn* library. When implemented for the first time, the results showed big differences between the validation Mean Squared Error (MSE) and the training MSE, suggesting a potential overfit. We soon realised that the *KFold* library does not shuffle indices when partitioning the folds. The indices were selected as can be seen in figure 4.3.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
[8] https://www.mathworks.com/help/optim/ug/quadprog.html
[9] https://github.com/stephane-caron/qpsolvers

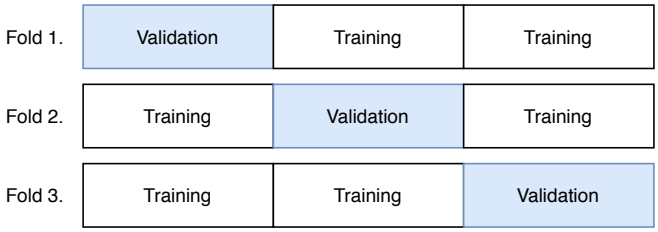



Figure 4.3: Default k-fold validation and training indices selection using the *KFold* class

This selection of indices happened on a non-shuffled $F_i$ matrix and $V_i$ vector. Since the entries in both constructs are in chronological+-* order, the training $\beta_i$ might differ drastically from the actual $\beta_i$ since a whole range of history is missing. Instead, the indices had to be picked randomly just like the logical array that was produced by the *cvpartition* MATLAB function. However, the way random indices selection works with the *KFold* package is that it shuffles the data points in random order and then applies the previously mentioned method of generating the validation and training indices as in figure 4.3. We were worried that scrambling the indices would interfere with obtaining the optimal solution, since the MATLAB implementation does maintain the chronological order of the indices. However, we derived from section 4.2.2 that the order of the indices does not matter as long as original row pairs in matrix $F_i$ and vector $V_i$ are kept the same. Thus we proceeded to use the shuffled indices for the k-fold cross-validation.

| province | validation MSE | training MSE | absolute difference |
|---|---|---|---|
| Drenthe | 0.001147 | 0.000947 | 0.000200 |
| Flevoland | 0.008410 | 0.006966 | 0.001444 |
| Friesland | 0.001763 | 0.001222 | 0.000540 |
| Gelderland | 0.002080 | 0.001878 | 0.000201 |
| Groningen | 0.001203 | 0.001013 | 0.000190 |
| Limburg | 0.015107 | 0.012335 | 0.002772 |
| Noord-Brabant | 0.005291 | 0.004551 | 0.000740 |
| Noord-Holland | 0.002992 | 0.002336 | 0.000656 |
| Overijssel | 0.007388 | 0.005132 | 0.002255 |
| Utrecht | 0.007734 | 0.006473 | 0.001262 |
| Zeeland | 0.004603 | 0.003701 | 0.000901 |
| Zuid-Holland | 0.003318 | 0.002364 | 0.000954 |

(a) Average performance using non shuffled training and validation indices during k-fold cross-validation over ten runs.

| province | validation MSE | training MSE | absolute difference |
|---|---|---|---|
| Drenthe | 0.001091 | 0.000982 | 0.000108 |
| Flevoland | 0.006114 | 0.005642 | 0.000472 |
| Friesland | 0.001416 | 0.001300 | 0.000116 |
| Gelderland | 0.001958 | 0.001754 | 0.000203 |
| Groningen | 0.001008 | 0.000970 | 0.000038 |
| Limburg | 0.012705 | 0.010489 | 0.002215 |
| Noord-Brabant | 0.003661 | 0.003343 | 0.000319 |
| Noord-Holland | 0.002235 | 0.002009 | 0.000225 |
| Overijssel | 0.005815 | 0.004488 | 0.001327 |
| Utrecht | 0.006830 | 0.006037 | 0.000794 |
| Zeeland | 0.003644 | 0.003170 | 0.000473 |
| Zuid-Holland | 0.002718 | 0.002203 | 0.000515 |

(b) Average performance using shuffled training and validation indices during k-fold cross-validation over ten runs.

Figure 4.4: Comparison of the shuffled and non shuffled indices during k-fold cross-validation averaged over ten runs.

Furthermore, the shuffling of indices yielded better performance compared to the non-shuffled performance. The performance was better in the sense that the difference between the validation MSE and the training MSE was significantly reduced. Therefore, the predictions yielded with the shuffled indices should be less prone to overfitting. Results of this can be seen in figure 4.4.

### 4.2.5. Accuracy

The only measure of performance of the algorithm which is described in the paper is the Mean Absolute Percentage Error (MAPE). MAPE is a measurement to see what the absolute percentage difference is between a predicted value and the actual value. However, since the MAPE can only be computed after the predicted days have actually passed, a prediction can only be judged once the data for the corresponding predicted date is collected. We contacted the authors of the NIPA paper with regards to a possible confidence interval. However, there was no confidence interval available as NIPA only produces a point forecast. Despite this, we still implemented the MAPE measure to see the prediction performance for ourselves. However, although the functionality exists to evaluate the algorithm, due to time constraints, this performance data is not made available in the front-end.

### 4.2.6. Structure

One of the design goals from section 2.5 was maintainability. Not only did we want to reflect this programming-wise, but also structure-wise. Although the NIPA paper provides pseudocode of the algorithm, we experienced issues grasping the MATLAB implementation. We could not always relate lines in the pseudocode to corresponding lines of code in the MATLAB implementation. Therefore, we tried to implement the algorithm as similar as possible to the pseudo code. We saw value in doing this in order for others to more easily understand how the algorithm works. In addition to this, we also included comments and function documentation explaining what the purpose is of certain parts of the code.

## 4.3. Front-end

This section explains how the biggest front-end implementation challenges were tackled. First, some of the most thought-through components are discussed, before the visual end-result is presented.

### 4.3.1. Data service

The data service is the heart of the front-end implementation. As explained in section 3.2.2, all visual components of figure 3.1 rely on the same data. To share this data amongst multiple components and avoid duplicate computations, Angular's *services* are used. These *services* are ideal for the challenges listed before.

An Angular *service* is defined as "a class with a narrow, well-defined purpose" [19]. The difference between an Angular *component* and a *service* is that a *component* merely serves as a view to the user, while a *service* provides processing functionalities. As a result, component classes remain "lean and efficient" [19]. This way, a product's modularity and reusability are increased.

The data service functions as follows. When a user first enters the dashboard, the data service makes a request to the back-end to retrieve the estimated infection probability data. Once this data is retrieved, the iteration which predicts the fraction of infected per day per province starts. Upon completion, both the factual and predicted data, per day per province, is shared through multiple *observables*. An observable is an object that implements the observer design pattern [20]. The data service internally only has one observable subject, but provides several other observable subjects to external components. Each of these subjects maps the data coming from the root observable. This way the data service only has to update a single observable subject each time a parameter changes. The root subject holds the data in the format shown in figure 4.5. Here, per province per day, the number of infected individuals divided by the number of inhabitants of that province is stored in JSON format.

```
{
    "Groningen": {
        "2020-01-01": 0.15,
        "2020-01-02": 0.20
    },
    "Utrecht": {
        "2020-01-01": 0.05,
        "2020-01-02": 0.15
    }
}
```

Figure 4.5: Output format of the data service.

### 4.3.2. Slider component

As shown in figure 3.1, the visual design includes several sliders. To avoid large amounts of code duplication, a separate slider component was implemented from scratch in a generic and reusable way. Beforehand, we experimented with Angular material sliders[10] as well as Bootstrap sliders[11]. Neither provided the required customisation options to get the slider to match the design. Although it did not have to look identical, both options lacked features, such as custom tick labelling, that we deemed necessary and we thus decided to implement a slider component from scratch.

This component takes two initial inputs. The first input is an array of initial values. If the array contains two elements, the component will represent a slider with a range, as required below the graph. When only one element is provided, the slider will function as a regular slider, as required by both the input section and below the map. This way, all the components on the dashboard page can use the same slider component and therefore only a single component needs to be maintained. The second input is the list of values the slider should allow, which are also displayed below the slider as tick labels.

Whenever a user changes the values of a slider, the slider emits the selected values. The emitted values are in the same format as the input: an array of values, one value for a simple slider, two values for a range slider. The parent component should then handle the processing of the emitted values. With only two inputs and a single output, the slider component provides an easy interface and hides a lot of complexity with respect to its users.

[10] https://material.angular.io/components/slider/overview

[11] https://mdbootstrap.com/docs/angular/forms/slider/

### 4.3.3. Dashboard page

The dashboard is provided as the landing page of the product, containing the most important features as required by the project description. These features are split into three components:

- **Map component**
  The map component uses D3.JS to load the map of the Netherlands into an Scalable Vector Graphics (SVG) element. Next, it subscribes to the root observable subject from the data service and listens for data updates. Every time infection data arrives, it calculates which province should be filled with which colour and updates the SVG accordingly. The colour of the province is based on the currently active date, set by the slider, and its corresponding normalised infection values. The slider can be used to change this active date. Furthermore, a province can be selected by clicking on it. The currently selected province, if any, is stored in the data service to allow other components to also use this property.

- **Graph component**
  The graph component also uses D3.JS to create its visualisations. This component subscribes to four observables of the data service. Two of those provide the absolute and cumulative numbers of all the provinces summed together. The other two are the absolute and cumulative numbers grouped per province. The former two observables are used to plot the default line which is visible in figure 3.1. The latter two are used to plot the line which represents a single province when one is selected. On top of the original design, a switch was added to switch between plotting cumulative or absolute data. The slider can be used to adjust the visible date range of the graph.

- **Input component**
  The input component does not rely on any of the data from the data service. The slider can be used to adjust the infection probability, while the date pickers allow the user to customise the start and end date of the change in infection probability. When a user adjusts any of these parameters, the input component informs the data service about this change. This way, the data service is aware and can start recomputing the predicted infection values. After this computation is finished, the data of both the map and graph component will be updated accordingly.

As described, the dashboard component is created in a modular way. All its separate components are linked in a way that prioritises maintainability and reusability. The final structure can seen in figure 4.6.

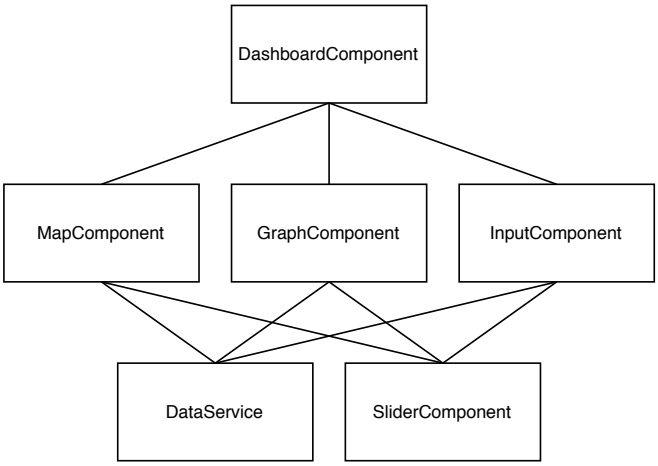



Figure 4.6: Overview of the dashboard architecture.

### 4.3.4. Information pages

As described in section 3.1.5, the application also required the implementation of information pages. These static pages only required simple HTML and CSS. Most of the time spent on these components was invested in writing the actual content. The algorithm page shortly describes the prediction algorithm used and references the original paper as well as its authors. Furthermore, the data page displays a table containing raw collected infection data, as well as the number of inhabitants per province. This is all the data required and used as input to the prediction algorithm. Next to that, an 'about us' page was created, that briefly introduces the project goals, tasks, creators as well as the client and coach. Finally, a link to the API documentation page is provided where several endpoints are listed and documented.

### 4.3.5. End result

After connecting all of the components above, the primary feature of the project, the finished dashboard, can be seen in figure 4.7. Several additional features where added which were not initially incorporated in the design. A reset button was added to the input section, allowing the user to quickly reset both the infection probability slider as well as the start and end date. Furthermore, a disclaimer has been added to the map, indicating that the currently active date and its corresponding colours are predicted. As previously mentioned, a switch was added to the graph component, which allows switching between cumulative and absolute data. Finally, a legend was added to the graph in order to distinguish between the total and province-specific plots.

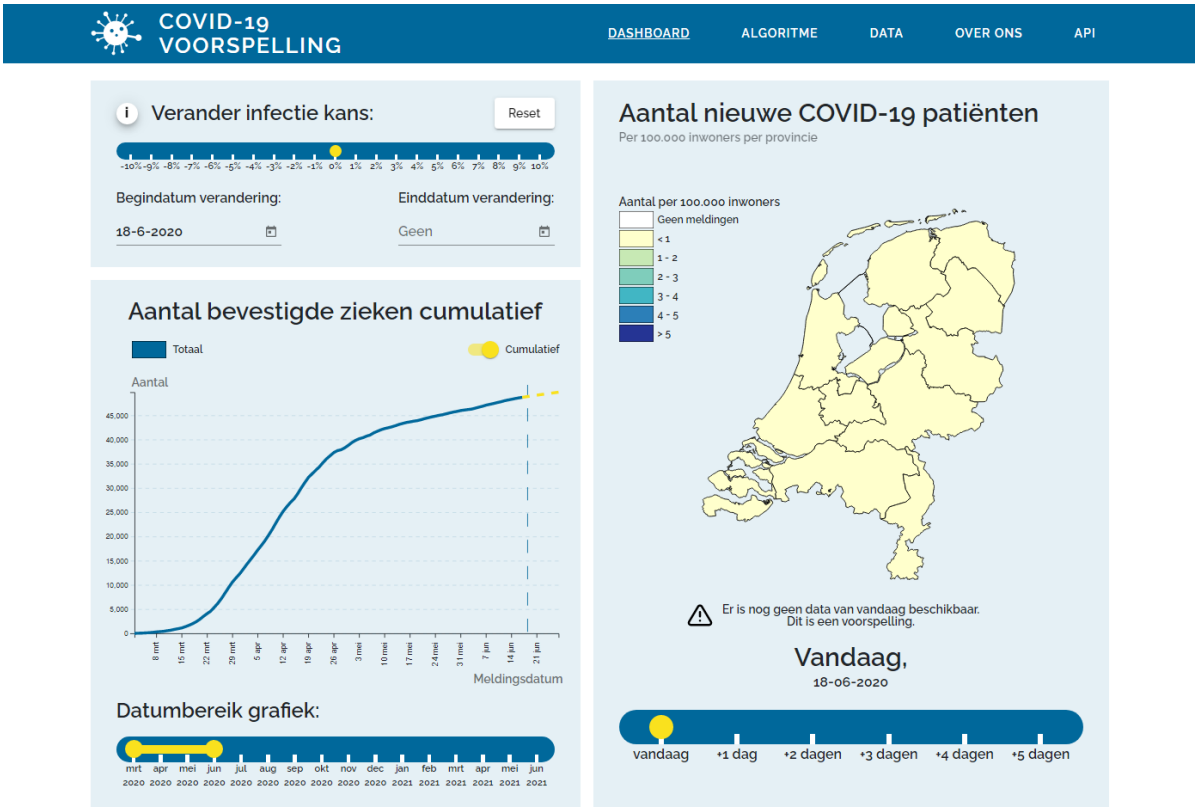



Figure 4.7: The final dashboard.

# 5
# Testing

The following chapter will describe the approaches that were taken to test the code. First, the utilities used to perform the software tests are mentioned in section 5.1. Section 5.2 describes the types of testing performed, after which section 5.3 displays the achieved test coverage. Finally, section 5.4 explains the measures taken to ensure the validity of our prediction algorithm implementation.

## 5.1. Testing utilities

Testing of the application was performed separately in the front-end and back-end. The front-end framework, Angular, provides the Jasmine[1] testing framework and the Karma[2] test runner, which were both used to implement front-end tests. The back-end was tested using the testing framework of Django.

## 5.2. Test types

Several test types were used to validate the application. We will first touch upon the use of unit testing in section 5.2.1, then discuss the system tests performed in section 5.2.2 and finally describe how user tests were carried out in section 5.2.3.

### 5.2.1. Unit testing

The primary testing type focused on during development was unit testing. The goal of unit testing is to test the functionality of individual units, such as a single function. In order to test only a single function of the application, we mocked any required dependencies.

### 5.2.2. System testing

Next to unit testing, system tests were performed. System tests are tests that assert the functionality of a complete system. This way, it can be assured that all individual components work together according to our expectations of the system. Whenever a change within a component results in incorrect system behaviour, these tests will raise awareness of the problem.

### 5.2.3. User testing

During the implementation phase, several user tests were conducted. Since the final product, as described in section 3.1, should be structured and user-friendly, both end-users and experts in the field of design were regularly asked for feedback. Below are the conducted experiments, their corresponding user as well as received feedback.

- **Client**
  The wishes of the project's client were of essential value. Multiple meetings were scheduled to discuss the progress over several weeks. These meetings provided valuable insight into whether or not we were heading in the right direction. Along the way, various suggestions were made for additional features which were gladly incorporated.
- **Coach**
  During weekly meetings with our coach, demonstrations of the progress on the product were given. Detailed feedback was given on both additional features that could be valuable, as well as existing features that required changes. Furthermore, the coach's knowledge on the topic of design led to various adjustments across the product.

[1] https://jasmine.github.io/

[2] https://karma-runner.github.io/latest/index.html

- **User**
  One of the authors of the NIPA paper was willing to provide feedback as someone that had not yet been involved in the project. This way, information could be collected on how a potential user perceived our product without any previous knowledge on the product's goal.

## 5.3. Test coverage

Line coverage was chosen as a measure of test coverage. To establish proper test quality, the line coverage was expected to be at least 80%. This number was high enough to ensure enough effort was put in, but low enough to avoid resorting to treating the metric.

The average back-end line coverage was 84%, as can be seen in table 5.1. The testing of several files was deemed unnecessary. Firstly, '`wsgi.py`' and '`asgi.py`' are files generated by the Django framework and thus we assume them to be properly set-up. Secondly, the '`commands`' folder has not been tested, as it only serves the purpose of calling the implemented algorithms from the command line. Lastly, the '`mape.py`' is not tested as it is not currently in use, but could be used in future versions of the product.

| Module | Line | |
|---|---|---|
| apps/algorithms/apps.py | 100% | 4/4 |
| apps/algorithms/management/commands/nipa.py | 0% | 0/8 |
| apps/algorithms/nipa/mape.py | 0% | 0/18 |
| apps/algorithms/nipa/nipa.py | 97% | 113/117 |
| apps/algorithms/nipa/nipa_data_interface.py | 94% | 30/32 |
| apps/algorithms/nipa/sir_compartments.py | 100% | 11/11 |
| apps/algorithms/nipa/sir_helpers.py | 86% | 43/50 |
| apps/data_collection/apps.py | 100% | 4/4 |
| apps/data_collection/arcgis_collector.py | 100% | 24/24 |
| apps/data_collection/data_collector.py | 91% | 10/11 |
| apps/data_collection/management/commands/refreshdata.py | 0% | 0/8 |
| apps/util/file_manager.py | 82% | 65/79 |
| covid_19_dashboard/asgi.py | 0% | 0/4 |
| covid_19_dashboard/settings.py | 100% | 23/23 |
| covid_19_dashboard/urls.py | 100% | 5/5 |
| covid_19_dashboard/views.py | 98% | 54/55 |
| covid_19_dashboard/wsgi.py | 0% | 0/4 |
| Total | 84% | 386/457 |

Table 5.1: Overview of back-end test coverage.

The average front-end line coverage was 93%, as can be seen in table 5.2. Most of the front-end code has been covered by the tests, as reflected by the high line coverage. Private methods could and have not been tested directly since these can not be accessed directly. They have, however, been assessed through testing the complete functionality of the components they belong to.

| Module | Line | |
|---|---|---|
| src/app/about-us | 100% | 1/1 |
| src/app/algorithm | 100% | 1/1 |
| src/app/dashboard | 100% | 1/1 |
| src/app/dashboard/graph | 98% | 165/169 |
| src/app/dashboard/input | 97% | 29/30 |
| src/app/dashboard/input/dialog | 100% | 2/2 |
| src/app/dashboard/map | 91% | 64/70 |
| src/app/dashboard/slider | 90% | 94/105 |
| src/app/data | 100% | 59/59 |
| src/app/services | 82% | 92/112 |
| Total | 93% | 508/550 |

Table 5.2: Overview of front-end test coverage.

## 5.4. Prediction algorithm validation

An important aspect of the application was validating the results of our implementation of the prediction algorithm. Although predictions can never be validated beforehand, the results of our implementation could be compared to the results of the provided MATLAB code.

The MATLAB code provided by the client was the original implementation used in the NIPA paper [6]. We wanted to run the MATLAB code on the data of the Netherlands and compare the results with our own implementation. The first step, therefore, was to convert the infection data of the Netherlands into a readable MATLAB *.mat* format. We wrote a converter script which uses Python and MATLAB to convert the RIVM data to the proper format.

After the first successful attempt of running the MATLAB code, we observed that the results produced by our product and the MATLAB implementation differed slightly. These first sets of tests made us catch a bug where the whole $F_i$ matrix and $V_i$ vector were passed on to the LASSO machine learning algorithm instead of just the training indices. This resulted in the product's results flattening faster than the output of the MATLAB implementation. After this bug was resolved, the product produced similar results to MATLAB, as can be seen in figure 5.1.

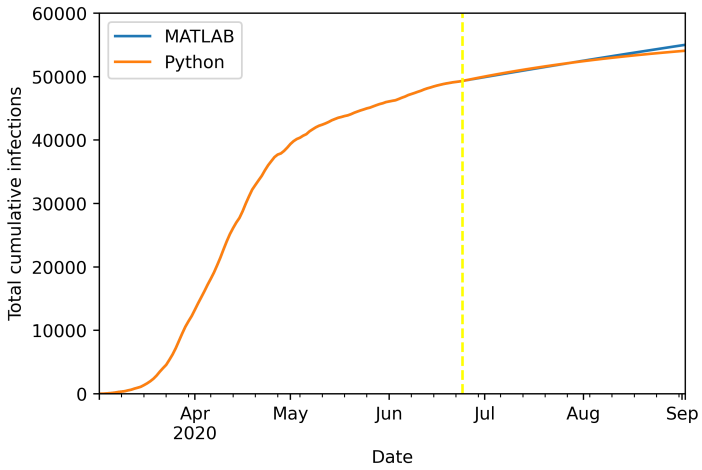



Figure 5.1: Comparison of the cumulative MATLAB and Python results. The yellow line indicates the start of predicted days.

The next step was to verify the similarity of the cumulative results. An initial attempt was made to compare the optimal curing probability $\delta_i$ and infection probability $\beta_i$ vector of the product and the MATLAB code per province. The tests have shown that the produced values of $\delta_i$ and $\beta_i$ differed significantly and rather randomly, yet the end results of cumulative SIR iteration done with predicted $\delta$ vector and $\beta$ matrix were similar. After these tests, we found out that the $\delta$ vector and $\beta$ matrix are similar when the same indices for testing and training were used. However, the selection of the data points used in training and testing indices is made randomly. Hence, it does not make sense to compare the similarity of $\delta$ and $\beta$ as long as the overall prediction results are similar. It is important to keep in mind that $\beta$ estimates are unstable: they change drastically on small changes of data despite producing similar prediction results. Therefore we proceeded to compare the predicted data point similarity of the MATLAB code and the product. It was found that given 80 days of data, and a prediction of the following 5 days, the prediction results consistently produced above 99% similarity. Thus to keep a margin of error we decided to test if our implementation has above 95% similarity with the MATLAB results.

# 6
# Process Evaluation

This chapter evaluates several processes within the project. In section 6.1, insight is given into the management of the project using the Scrum methodology. Next, section 6.2 describes the way in which work was distributed amongst team members. Section 6.3 reflects on the communication process of both internal and external meetings.

## 6.1. Scrum

To manage the project, an agile management framework called Scrum [21] was used. The Scrum methodology relies on dividing work into short iterations, called sprints, where each sprint has its own planning and review. This project made use of seven-day-long sprints, starting and ending on Monday. Each new sprint started with a reflection on the previous week's sprint. Notes were taken on whether or not a task was completed and the time it took to do so. In case someone did not manage to finish a task, a note was taken on how this could have been prevented. Once the sprint review was completed, a new sprint planning was created. The reason for choosing seven-day-long sprints was to allow team members to work during the weekends as well.

Documenting each of these sprints was done using GitLab's wiki pages. This allowed us to keep the project's code, issues and documentation all in one place. Tables were used to keep the planning and reviews structured and clear.

Overall, this structure of management worked well. Everyone was aware of their tasks for the week early on, allowing each member to create their own week planning. Weekly reflection with the entire team enabled us to locate any problems before it was too late. One of the things we learned along the way was that estimating the required time for a specific task is rather difficult. This led to some tasks taking up to twice as long as projected. However, due to the strict planning made beforehand, which would have left us with time for additional features at the end, this did not pose a problem. Within the field of software engineering, effort estimation is a known problem. So much, that many studies have been conducted that attempt to create effort estimation techniques [22]. We agree that a closer look should be taken at some of these estimation techniques in order to avoid similar issues in future projects.

## 6.2. Distribution of work

The project saw a clear division of tasks. During the research phase, personal preferences were taken into account when dividing the different fields of research to be conducted. Some had a preference for design and were thus assigned to look into different visualisation options, while others preferred looking into the most suitable programming languages and frameworks. Looking back, this definitely helped the quality of the product: highly motivated members each researched something of their interest.

In the first four weeks of the implementation phase, three members were dedicated to the back-end implementation, whilst the other two were dedicated to the front-end implementation. This decision was taken deliberately, since we were working with unfamiliar technologies. By choosing this approach, we hoped that each of us would get a more specific understanding of one technology instead of getting just a general level understanding of multiple technologies. Towards the end of the implementation phase, the focus shifted more towards implementing additional features to the front-end. Eventually, all of us contributed to the front-end. Here, the team members who were already familiar with front-end could use their gained knowledge to aid the other members, as well as maintain the quality of the front-end code. In the end, we are sure that splitting the team in two increased the efficiency of implementing new features. Once every member was specialised in their respective area, the right approach to implementing such features was quickly found.

## 6.3. Communication

Due to the COVID-19 pandemic, the TU Delft prohibited all on-campus activities. This led to the unique situation of having to do the entire bachelor end-project digitally. As a result, several challenges arose regarding the structure of the project. First, section 6.3.1 discusses the approach taken to internal meetings after which section 6.3.2 explains how external meetings were approached.

### 6.3.1. Internal meetings

One of these challenges was communication, which instead of daily on-campus meetings turned into online calls. This challenge was solved by the team through the use of a Discord[1] channel. We agreed to have a daily meeting at 10:30 in the morning to discuss what everyone was working on and how new features were coming along.

This approach was followed for the first two weeks, after which it was realised that the working environment was not ideal. Most members had a hard time focusing on their work due to not in any way being connected to their peers. As a result, not all work was finished on time or work was implemented wrongly due to a lack of communication. Together it was decided that a change was required and thus a meeting was organised.

During this meeting, a new schedule was created. From that week on, each member had to be available in the Discord channel between 9:30 and 12:00, and 13:30 and 14:30. This way, a change in plan or implementation could be quickly discussed and communicated. Next to that, it encouraged members to work together in small groups to help each other when they got stuck. The new schedule did not mean that no work should be done outside of these hours, but merely functioned as a guideline. Other hours could be managed individually. This approach solved the problems we were having in the starting phase of the project and allowed us to develop the application much more efficiently.

### 6.3.2. External meetings

Throughout the project, the team had a weekly meeting with the project coach. During each meeting, the progress was discussed and questions were asked from both sides. Furthermore, the coach provided several existing COVID-19 dashboards that she discovered throughout the development phase. These dashboards were used as a source of inspiration and included in the report to provide an overview of existing dashboards.

Next to those meetings, a meeting between the team and the client was held roughly every two weeks. These meetings also served the purpose of showing the progress up till that point and asking questions. Furthermore, the client would make suggestions regarding additional features or features to be changed. This way, we could make sure that the end-product was according to the client's wishes.

Looking back, we valued the meetings, as the coach and the client provided different perspectives. This led to extensive feedback on a wide variety of topics, that possibly we had not yet considered.

[1] https://discord.com/

# 7
# Ethics

The COVID-19 pandemic raises a lot of dilemmas between being able to sustain a functioning society and large scale healthcare concerns [23]. With an ethically complicated topic such as the COVID-19 pandemic [24], any relevant ethical questions should be elaborated on. Therefore, this chapter discusses the potential impact this application might have and what resulting consequences it might bring with it, in an attempt to answer whether it is morally permissible to provide individuals with potentially faulty predictions of the spread of the COVID-19 virus. Finally, the sections discuss how we attempted to suppress any negative consequences.

## 7.1. Algorithm credibility and relevance

Determining the credibility of the prediction algorithm is a rather tough task, since the quality of the prediction can only be assessed once the predicted dates passed and the actual data was collected to compare it to. As mentioned in section 4.2.5, the authors of the NIPA method do not specify a confidence interval but only a point forecast. This implies that there is no concrete percentage of confidence which we can assign to our prediction. A missing confidence interval might conceal a low precision and then any prediction might lose its value. Even with a confidence interval, is the algorithm trustworthy enough to be taken into consideration when making policies? This is an important question to keep asking ourselves, but most importantly, this is a question the end-users should also be asking themselves. When using output produced by the algorithm, the user should be reflecting critically on the credibility and the relevance of the results. In the end, the prediction is only based on historic data and does not take into consideration changes in mitigation strategies by the government. Furthermore, the data on which predictions are based, is only a lower bound of the actual number of infected individuals in the Netherlands. Thus, the algorithm most likely also does not predict the actual number of infected individuals. Therefore, we stress the fact that policies should not be solely based on the results produced by this algorithm. To aid the end-user in being critical towards the predictions, we inserted the following features:

- **Disclaimers**
  These show whether some value is a predicted value or not.
- **Information tooltips**
  They aid the user in getting a better understanding of how to interact with the application in the intended way. Tooltips are also used to notify the user of which data is interpolated.
- **Additional information pages**
  These give the user a general explanation of how the algorithm and the data collection work.

## 7.2. Social transparency

Whenever the application is running, anyone has access to the data we have collected, interpolated and predicted. Is it ethically responsible to provide anyone with potentially wrong predictions about a healthcare crisis? Because, in the end, there is no guarantee that the algorithm produces results which can reflect future development of the virus in the Netherlands. A way to provide transparency towards the public is by creating a social impact statement [25]. This statement outlines five guiding principles to implement and maintain an algorithm in a publicly accountable way.

1. **Responsibility**:
   The first step towards public accountability is to define the responsible party who is to resolve any issues with the algorithm. Since all the used knowledge and resources are in the public domain, it is not straightforward to define a single responsible person or party. Therefore, we conclude that the public domain itself is responsible for the quality of the algorithm.

2. **Explainability**:
   The inner workings of the algorithms should be explainable to the public. In addition to the original paper, we dedicated a separate page in our application to the workings of the algorithm itself.

3. **Accuracy**:
   Judging the performance is, as already stated earlier in the section above, rather difficult. The authors of the NIPA paper propose the MAPE to be a good measure of how well the algorithm is performing. This error indicates how well an older trained version of the algorithm performed on newly gathered data.

4. **Auditability**:
   The application is completely open-source and therefore, any third party is able to examine the codebase and judge the quality of the implementation of the algorithm.

5. **Fairness**:
   Only the number of newly infected individuals and populations of the provinces are used. Thus there is no attribute in the data present which can be discriminated on.

By filling in this social impact statement we hope to create transparency towards the general public. Not necessarily for the algorithm to be completely trusted, but to seem trustworthy enough to be considered potentially relevant.

## 7.3. Negative consequences

Identifying scenarios inferring potentially negative consequences early on can help prevent or suppress any negative backlash. We attempted to identify such scenarios beforehand and take the necessary measures.

- **Predictions misalign with government policies**
  There is a chance that predictions produced by the algorithm are not inline with the policies enforced by the government. A non-critical user or misinformed user can potentially be fed misinformation about the development of the virus in the Netherlands. We inserted a clear explanation of who we are and thus preventing anyone from suggesting that the dashboard might be from governmental origins.

- **Data is incomplete**
  The collected data might not be complete or the data might differ from other data sources. The collection of data is thoroughly discussed in section 4.1.1. The source of data collection is motivated on one of the additional pages and the trade-offs are listed. These also include reasons why the currently used source might not have been the best option. This measure was taken to prevent any remaining questions about the validity of the data.

## 7.4. Conclusion

So far we have discussed the credibility of the algorithm and determined its relevance in potential policy-making. Again, we stress the fact that human judgement should be the number one priority and that the application must only occupy an assisting role. Therefore we also discussed the transparency of the implementation and maintainability. At last, we identified problematic scenarios which may lead to a public backlash. The necessary actions to prevent these issues have already been taken. We therefore conclude that it can be morally permissible to feed individuals with potentially wrong predictions, given the taken actions to prevent non-criticality in a user.

# 8

# Conclusion

The goal of this project, as described in appendix B, was to implement a given prediction algorithm, called NIPA, after which the algorithm's output should be visualised. For the project to be successful, all must-have requirements as listed in appendix C.1 must be implemented. Furthermore, all non-functional requirements in appendix C.2 should be taken into account.

This project is unique in that it aims to solve a problem that was introduced only recently. With the first confirmed infected Dutch inhabitant on the 27th of February [26], the project was quickly set up and listed as one of the possible end-projects. Making this project a success seemed like a way of giving back to society and we were therefore very happy to contribute.

In seven weeks of development, a product was created that meets all the must-have requirements. Additionally, infection data is collected on a daily basis, fulfilling the single should-have requirement as well. Finally, two could-have features have been implemented, namely an API was created serving both the collected factual infection data as well as the infection data predicted by the prediction algorithm. Although feedback showed that several additions to the product would be beneficial for the usability of the product, a lack of time required for decisions to be made on which features to implement and which to skip.

Next to the functional requirements, some non-functional requirements were set. One of the ways in which these were taken into account can be seen in section 4.3. While implementing the front-end of the application, we made sure to split up functionality and with that keep the software understandable and modifiable. Furthermore, it was agreed upon that every function should be provided with proper documentation. In general, during code implementation and review the non-functional requirements were incorporated.

When taking a look at the project description, it states that we were expected to implement the provided prediction algorithm and use it to simulate the effect of both mitigation and exit strategies by allowing the user to input the relative percentage change in infection probability that this strategy would cause. Finally, the findings had to be visualised through a web application. If we look at the final product, we can conclude with certainty that all of these requirements have been fulfilled and therewith the project can be seen as successful.

# 9
# Discussion and recommendations

This chapter discusses potential improvements that can be made in the future. In section 9.1, multiple of these improvements are listed and elaborated upon. Furthermore, section 9.2 discusses and compares other dashboards released both prior to as well as during the implementation phase.

## 9.1. Future improvements

Despite the end-product fitting the client's needs, it is essential to elaborate on the improvements that can be made with future work and research. This section lists recommendations and several potential improvements on the product.

### 9.1.1. Manipulation of delta values

The client has agreed that it is not a priority to manipulate the curing probability $\delta$ in the web app. The product can be improved by adding a slider, which can manipulate $\delta$, similar to the slider of the $\beta$.

### 9.1.2. Manipulation of individual beta values

The product currently allows manipulation of the infection probabilities with a single slider that impacts the entirety of the Netherlands. As a future improvement, separate sliders could be added to alter just the infection probability between two provinces. This could help decision-makers to easily observe outcomes of closing down certain locations while opening up others.

### 9.1.3. Municipal calculations

With the request of the client, the app has been built to calculate NIPA and results per province. Yet, if it is thought that it would benefit the accuracy, an addition could be done to calculate results per municipality. This would require changes on the map component of the front-end, as well as the back-end data collection.

### 9.1.4. Multi-threading of NIPA

NIPA attempts to find the best fitting $\beta$ and $\delta$ values. To achieve an optimal solution, it runs the LASSO machine learning algorithm for every region, $\delta$ value and $\rho$ value it has been instructed to try. This part of the algorithm has a time complexity of $O(|\text{region}| \cdot |\text{delta\_candidates}| \cdot |\text{rho\_candidates}|)$ and can take long amounts of time for a larger set of regions or for an increase in the delta and rho search spaces. At an earlier stage of the implementation, we implemented multi-threading to see the increase of performance. The duration of NIPA running with multi-threading can be seen in table 9.1. However, in the end we did not include multi-threading, since a rework of the algorithm forced us to rewrite multi-threading. The product currently works by trying 20 rho and delta candidates. Running NIPA with multi-threading on 20 delta candidates only improved the run time by 8.64 seconds, which we found not enough to ignore the complexity multi-threading adds to the production code. Furthermore, multi-threading could get in our way when debugging certain aspects of the algorithm. If future developers decide to modify the product to work with a larger set of regions, or an increased $\delta$ and $\rho$ search space, then they should look into integrating multi-threading into the product.

| Number of delta candidates | Total execution time of NIPA on all provinces of the Netherlands. | |
|---|---|---|
| | With multi-threading (6 threads) | Without multi-threading |
| 20 | 5.76 seconds | 14.4 seconds |
| 100 | 31.32 seconds | 85.92 seconds |

Table 9.1: Influence of multi-threading on the execution time with varying numbers of delta candidates.

### 9.1.5. Security

Every time the website is loaded, data is requested from the back-end. This puts the website at risk of Denial of Service (DoS) attacks [27], where many requests could be sent to the website to make it inaccessible. This is a threat to consider improving upon for future researchers working with this app.

Finally, a website security certificate could be requested if it is decided that the contents of the web app will be publicised to another domain name.

## 9.2. Similar products

In section 2.4 we mentioned some dashboards that were released before we started the development phase of the product. These products were mainly dashboards that displayed confirmed infection data, without predictions. During the implementation phase, several new dashboards were released and discovered that make predictions about COVID-19's future course. Other dashboards can be used as a source of inspiration for future work, such as additional features or improvements. In section 9.2.1, the products discovered prior to the implementation phase will be compared, after which section 9.2.2 will compare later discovered products that all visualise predictions.

### 9.2.1. Prior discovered products

In table 9.2, a comparison is made between our product and the dashboards that we encountered during the research phase. Each product has been judged on a number of criteria.

- **Infection demographics**
  From the prior discovered products, COVID-19 Public Dashboard of California[1] is the only dashboard we have encountered that offers demographics of the infected people.
- **Intended region**
  Three of the five dashboards provide worldwide infection data. One dashboard focuses on the U.S.A. while our dashboard focuses on the Netherlands.
- **Tabular view**
  All the dashboards have a tabular view and a map visualisation, except for COVID-19 Visualizer[2], which does not have a tabular view but instead shows data over a world map.
- **Display predictions**
  The IHME COVID-19 Projections[3] website is the only dashboard we have encountered in the research phase that did predictions over the course of the virus. They provide a confidence interval for their predictions, which is not offered in our product.
- **Data download option**
  The IMHE and California dashboard are the only two dashboard that allow for directly downloading the displayed data.
- **API**
  Our product is the only product amongst the others that offers an API.

| Product name | Infection demographics | Intended region | Tabular view | Predictions | Downloadable data | API |
|---|---|---|---|---|---|---|
| COVID-19 Public Dashboard of California[1] | Yes | California U.S.A | Yes | No | Yes | No |
| COVID-19 Visualizer[2] | No | Worldwide | No | No | No | No |
| IHME COVID-19 Projections[3] | No | Worldwide | Yes | Yes | Yes | No |
| Johns Hopkins University COVID-19 dashboard[4] | No | Worldwide | Yes | No | No | No |
| COVID-19 Voorspelling (Our product) | No | Netherlands | Yes | Yes | No | Yes |

Table 9.2: Product's comparison with dashboards we have encountered during the research phase.

[1] https://public.tableau.com/views/COVID-19PublicDashboard/Covid-19Public
[2] https://www.covidvisualizer.com
[3] https://covid19.healthdata.org/netherlands
[4] https://coronavirus.jhu.edu/map

### 9.2.2. Later discovered products

In table 9.3, a comparison is made between our product and other products that do predictions. These products are the ones that have been discovered after our project's research phase.

- **Built-in mitigation and exit strategies**
  An interesting product is Policy Impact Predictor for COVID-19 from Van der Schaar Lab[5]. This product works with the data of the United Kingdom and has three built-in mitigation strategies, namely: maintaining U.K.'s current policies, following the government's plan and ending lockdown in August.

  Unlike this product, ours does not offer built in strategies. It does, however, offer the ability of manipulating the spread rate of COVID-19. The impact on spread rate of exit strategies could be calculated by scientists, after which decision-makers can visually see the outcome of the computed change in spread rate.

  The other two products, COVID-19-projections[6] and Windfall AI[7], both don't offer different exit strategies nor manipulation of parameters.

- **Intended region**
  Two of the dashboards are made solely for the Netherlands: Windfall AI and our product. COVID-19-projections is intended for the entire world, while the Van der Schaar Lab product is intended for the U.K.

- **Confidence interval**
  Van der Schaar Lab's product and COVID-19-projections both display a confidence interval in their predictions. This is not offered in our product nor in Windfall AI. The reason our product does not have a similar feature is due to the limitations of the NIPA. We have consulted the writers of the NIPA paper and were told that NIPA does not provide a confidence interval.

- **Manipulation of parameters** Our product offers a slider for changing the infection probability, which is not offered in any other products. Although, there is no indicator offered to show a confidence interval for the prediction range, again, due to NIPA not offering a confidence interval.

- **API**
  Lastly, our product offers an API. The other websites allow downloading their data, however, they don't offer an API to access their data. We have built an API to allow researchers to access our data easier.

We have been shown another COVID-19 dashboard from Rijksoverheid[8] by our coach. However, this dashboard was showing different types of data than we have seen in all other dashboards, such as intensive-care and hospital records per day, the number of people who may infect others and the reproduction number, which stands for how fast the virus spreads itself. This product is not included in the comparison tables since its features and purpose is different than the rest of the dashboards.

| Product name | Built-in exit and mitigation strategies | Intended region | Prediction algorithm | Confidence interval | Manipulation of parameters | API |
|---|---|---|---|---|---|---|
| Van der Schaar Lab, Policy Impact Predictor for COVID-19[5] | Yes | U.K. | Two-layer Gaussian process with SEIR [28] | Yes | No | No |
| COVID-19-projections[6] | No | Worldwide | SEIR model [28] | Yes | No | No |
| Windfall AI[7] | No | Netherlands | Linear and Ridge regressions [29], Random forest regression [30] | No | No | No |
| COVID-19 Voorspelling (Our product) | No | Netherlands | NIPA | No | Yes | Yes |

Table 9.3: Product's comparison with similar products that do predictions and have been released after our research phase.

[5] https://www.vanderschaar-lab.com/policy-impact-predictor-for-covid-19
[6] https://covid19-projections.com
[7] https://windfall.ai/covid
[8] https://coronadashboard.rijksoverheid.nl

# Bibliography


[1] T. Adhanom Ghebreyesus, "WHO director-general's opening remarks at the media briefing on COVID-19 - 11 march 2020," Mar 2020, accessed on 2020-04-20. [Online]. Available: https://www.who.int/dg/speeches/detail/who-director-general-s-opening-remarks-at-the-media-briefing-on-covid-19---11-march-2020

[2] Anonymous, "Monthly briefing: COVID-19 disrupting lives, economies and societies | department of economic and social affairs," Apr 2020, accessed on 2020-04-20. [Online]. Available: https://www.un.org/development/desa/dpad/publication/world-economic-situation-and-prospects-april-2020-briefing-no-136/

[3] W. Yang, A. Karspeck, and J. Shaman, "Comparison of Filtering Methods for the Modeling and Retrospective Forecasting of Influenza Epidemics," *PLoS Computational Biology*, vol. 10, no. 4, p. e1003583, Apr. 2014, accessed on 2020-04-26. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1003583

[4] T. K. Yamana, S. Kandula, and J. Shaman, "Individual versus superensemble forecasts of seasonal influenza outbreaks in the United States," *PLOS Computational Biology*, vol. 13, no. 11, p. e1005801, Nov. 2017, accessed on 2020-04-26. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1005801

[5] E. L. Ray and N. G. Reich, "Prediction of infectious disease epidemics via weighted density ensembles," *PLOS Computational Biology*, vol. 14, no. 2, p. e1005910, Feb. 2018, accessed on 2020-04-26. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1005910

[6] B. Prasse, M. A. Achterberg, L. Ma, and P. Van Mieghem, "Network-based prediction of the 2019-NCOV epidemic outbreak in the Chinese province Hubei," *arXiv preprint arXiv:2002.04482*, 2020.

[7] V. Belik, T. Geisel, and D. Brockmann, "Natural human mobility patterns and spatial spread of infectious diseases," *Physical Review X*, vol. 1, no. 1, p. 011001, Aug. 2011, arXiv: 1103.6224, accessed on 2020-06-21. [Online]. Available: http://arxiv.org/abs/1103.6224

[8] D. Brockmann and D. Helbing, "The Hidden Geometry of Complex, Network-Driven Contagion Phenomena," *Science*, vol. 342, no. 6164, pp. 1337–1342, Dec. 2013, accessed on 2020-06-23. [Online]. Available: https://science.sciencemag.org/content/342/6164/1337

[9] V. Colizza, A. Barrat, M. Barthélemy, and A. Vespignani, "The role of the airline transportation network in the prediction and predictability of global epidemics," *Proceedings of the National Academy of Sciences*, vol. 103, no. 7, pp. 2015–2020, Feb. 2006, accessed on 2020-06-23. [Online]. Available: https://www.pnas.org/content/103/7/2015

[10] N. A. o. Sciences, "In This Issue," *Proceedings of the National Academy of Sciences*, vol. 106, no. 51, pp. 21 459–21 460, Dec. 2009, accessed on 2020-06-23. [Online]. Available: https://www.pnas.org/content/106/51/21459

[11] D. Smith and L. Moore, "The SIR model for spread of disease: The differential equation model," *Loci.(originally Convergence.)*, 2004, accessed on 2020-06-20. [Online]. Available: https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model

[12] "Forecasting the impact of the first wave of the COVID-19 pandemic on hospital demand and deaths for the USA and European Economic Area countries," Apr. 2020, accessed on 2020-06-23. [Online]. Available: http://www.healthdata.org/research-article/forecasting-impact-first-wave-covid-19-pandemic-hospital-demand-and-deaths-usa-and

[13] Burkhard and R. Aslak, "Towards a framework and a model for knowledge visualization: Synergies between information and knowledge visualization," in *Knowledge and information visualization*. Springer, 2005, pp. 238–255.

[14] M. J. Eppler, "What is an effective knowledge visualization? insights from a review of seminal concepts," in *Knowledge Visualization Currents*. Springer, 2013, pp. 3–12.

[15] Slutsky and D. J, "The effective use of graphs," *Journal of wrist surgery*, vol. 3, no. 02, pp. 067–068, 2014.

[16] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine Irvine, 2000, vol. 7.

[17] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[18] S. Caron, "Quadratic programming in python," accessed on 2020-06-15. [Online]. Available: https://scaron.info/blog/quadratic-programming-in-python.html

[19] "Angular," accessed on 2020-06-18. [Online]. Available: https://angular.io/guide/architecture-services

[20] "Observer design pattern," accessed on 2020-06-18. [Online]. Available: https://sourcemaking.com/design_patterns/observer

[21] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Prentice Hall Upper Saddle River, 2002, vol. 1.

[22] M. Usman, E. Mendes, F. Weidt, and R. Britto, "Effort estimation in agile software development: A systematic literature review," in *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 82–91, accessed on 2020-06-23. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/2639490.2639503

[23] I. Chakraborty and P. Maity, "COVID-19 outbreak: Migration, effects on society, global environment and prevention," *Science of The Total Environment*, vol. 728, p. 138882, 2020, accessed on 2020-06-18. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0048969720323998

[24] E. J. Kompanje, "Ethics in the time of corona," *European Respiratory Society*, 2020, accessed on 2020-06-16. [Online]. Available: https://www.ersnet.org/covid-19-blog/ethics-in-the-time-of-corona

[25] N. Diakopoulos, S. Friedler, M. Arenas, M. Hay, B. Howe, H. V. Jagadish, K. Unsworth, A. Sahuguet, S. Venkatasubramanian, C. Wilson, C. Yu, and B. Zevenbergen, "Principles for accountable algorithms and a social impact statement for algorithms," accessed on 2020-06-14. [Online]. Available: https://www.fatml.org/resources/principles-for-accountable-algorithms

[26] "Ontwikkeling COVID-19 in grafieken | RIVM," accessed on 2020-06-20. [Online]. Available: https://www.rivm.nl/coronavirus-covid-19/grafieken

[27] Anonymous, "Security tip (st04-015) - understanding denial-of-service attacks," Nov 2009, accessed on 2020-06-24. [Online]. Available: https://www.us-cert.gov/ncas/tips/ST04-015

[28] J. L. Aron and I. B. Schwartz, "Seasonality and period-doubling bifurcations in an epidemic model," *Journal of theoretical biology*, vol. 110, no. 4, pp. 665–679, 1984.

[29] S. Bhattacharyya, "Ridge and lasso regression: L1 and l2 regularization," Sep 2018, accessed on 2020-06-08. [Online]. Available: https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

[30] A. Chakure, "Random forest and its implementation," Jun 2019, accessed on 2020-06-20. [Online]. Available: https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f

# A

# SIG

This appendix discusses the feedback received by Software Improvement Group (SIG).

## A.1. First submission

The first submission to SIG scored a 4.6 out of 5.5 on overall maintainability. Although this was already a high score, the tool also showed some elements of the code base which could still be improved.

Several code characteristics required improvement according to SIG standards. One of them was unit size. Unit size is measured as the number of lines of code in a unit. For this metric, an initial score of 2.4 out of 5.5 was achieved. This low score was mainly caused by the functions implementing the final prediction iteration, both in the front-end and back-end. According to SIG, these functions had moderate risk with between 16 and 30 lines of code.

Furthermore, unit complexity could be improved. Unit complexity is measured as the McCabe complexity of a unit. Initially, a score of 4.3 out of 5.5 was achieved. This metric went hand in hand with the unit size, as the large unit size caused higher McGabe complexity.

Another metric that required attention was unit interfacing. Unit interfacing is measured as the number of parameters a unit expects. This metric initially scored 4.0 out of 5.5. Multiple functions in the back-end part of the code base had more than four parameters. A large part of this was accountable to multiple functions working with the SIR format. The SIR format consists of the three vectors $S$, $I$ and $R$, which were initially passed separately to each function. Next to that, some of those functions also required additional parameters. With all those parameters added together, some functions reached a total of 5 or 6 parameters, which SIG deemed as a high risk.

On several other metrics, the maximum score of 5.5 out of 5.5 was achieved. According to SIG, code volume, code duplication and module coupling all achieved the maximum score. Module coupling scored a 4.4 and component balance a 5.1.

## A.2. Actions taken

The first point of attention that we worked on was the unit size. Indeed, several functions were more than 15 lines long and therefore, according to SIG, had an impact on the code maintainability. The functions that were too large were split into separate functions, each with their own task. An example of such a function was the one that performed the front-end iteration. Although most of this function was dedicated to this iteration, the computed results were formatted and shared with other components. This functionality was initially also implemented in the same function, but now extracted and put into a separate function. As a result, both functions had a single responsibility, and thus the code quality improved. Similar actions were taken for other functions.

After improving the unit size of several functions, unit complexity was automatically improved. As mentioned in the previous section, the two go hand in hand. Splitting up functions to reduce unit size also led to reduced McGabe complexity of the refactored functions.

Finally, unit interfacing was improved. The primary problem, as mentioned before, was that the SIR vectors were passed separately to multiple functions. These were already three arguments, while SIG only deems less than three arguments as low risk. Often, these functions would require additional parameters. To improve the unit interfacing, a separate class called *SIR* was created that simply contains three variables. This way, functions using SIR data required two less parameters, as only an instance of that class was required. Furthermore, the reduction of unit size also resulted in reduced unit interfacing.

## A.3. Second submission

The second submission to the Software Improvement Group scored a 4.7 out of 5.5 on overall maintainability. This improvement is mostly thanks to the large reduction in unit size, which was increased in score from 2.4 to 3.6. The only remaining refactoring candidates with large unit sizes are a multitude of *scss* files. However, the file type *scss* is specifically made for nesting *css* styles. Therefore, we do not agree that these files actually require refactoring.

Together with the unit size, the unit complexity was improved. The changes made resulted in an improvement from 4.3 to 5.3.

After incorporating the unit interfacing suggestions, the back-end score improved, but by very little: from 4.0 to 4.3. The reason for this is that SIG sees a function with three parameters as a code smell. Due to the algorithmic complexity of NIPA, it became nearly impossible to get the number of parameters below three. To get the number of parameters below two, the overall structure had to be so drastically changed that this would be at the cost of code maintainability. Since, after all, the goal of SIG feedback is to improve code maintainability, we chose not to refactor methods with three parameters.

This second feedback also showed a metric that decreased in score: module coupling decreased by 0.6, from 5.4 to 4.8. The cause SIG provides for this lower score, is the file manager in the back-end. However, we do not agree that it is a bad thing that this module is being used as much as it is. After all, the purpose of the file manager is to abstract the handling of files away from the other components of the application.

# B

# Project Description

This BEP project aims to address the following urgent and challenging questions:

(A) Based on the number of newly infected and the number of death per day per city in e.g. the Netherlands that have been observed in the past period, predict the number of newly infected/deceased per day per city in the coming days.

(B) Effect of mitigation strategies such as social distancing on e.g. the prevalence of COVID-19, so called "flattening the curve".

- **Method**
  The team is supposed to:
  1. Implement the algorithm NIPA [6] which solves question A.
  2. Apply NIPA to evaluate the effect of diverse mitigation strategies
  3. Build a webpage/app so that the public could visualise the prediction, evolution of the prevalence over time and the effect of an mitigation strategy that a user chooses.
- **Client**
  Dr. Huijuan Wang, PI of the AI-Networking Lab, Multimedia Computing Group, Department of Intelligent Systems. This project is in collaboration with the Network Architecture and Services Group of TU Delft, which has been requested by the RIVM (National Institute for Public Health and the Environment) to tackle the aforementioned challenges.

This project is very challenging. Beyond the software part, the understanding and implementation of the algorithm NIPA require solid knowledge in linear algebra, probability theory, machine learning algorithm Lasso and possibly real-time data collection from RIVM.

# C

# Requirements

## C.1. Functional requirements

Functional requirements are ranked based on the MoSCoW method. Functional requirements in the must-have group form the core of the project and form the minimal viable product.

- **Must have**
  - A working NIPA implementation
  - Allow user input of beta values to simulate mitigation/exit strategies. This should come with an adjustable time-frame (start date, end date)
  - Visualisation of NIPA results
  - Visualisation of the effects of different betas
  - The system shall be accessible through a web application
- **Should have**
  - Automatic data collection from RIVM
- **Could have**
  - Use data from other countries to estimate recovery rate (delta)
  - Research equality between countries
  - Others could/should be able to download our collected data (in one standard format)
  - Include neighbouring countries as part of the input
  - Data API
- **Won't have**
  - Include maps/visualisation for the entire world

## C.2. Non-functional requirements

- Open-source
- Understandable and modifiable software
- In-depth documentation
- The system must be scalable

# D

# NIPA formulae

This appendix lists all the relevant formulae from the NIPA paper:

- SIR epidemic model:

$$I_i[k+1] = (1-\delta_i)I_i[k] + (1 - I_i[k] - R_i[k]) \sum_{j=1}^{N} \beta_{i,j} I_j[k] \tag{D.1}$$

$$R_i[k+1] = R_i[k] + \delta_i I_i[k] \tag{D.2}$$

$$S_i[k] = 1 - I_i[k] - R_i[k] \tag{D.3}$$

- The $F_i$ matrix and $V_i$ vector are defined as:

$$F_i = \begin{bmatrix} S_i[1]I_1[1] & \dots & S_i[1]I_N[1] \\ \vdots & \ddots & \vdots \\ S_i[n-1]I_1[n-1] & \dots & S_i[n-1]I_N[n-1] \end{bmatrix} \qquad V_i = \begin{bmatrix} I_i[2] - (1-\delta_i)I_i[1] \\ \vdots \\ I_i[n] - (1-\delta_i)I_i[n-1] \end{bmatrix} \tag{D.4}$$

- The SIR Epidemic Model can then be rewritten in linear form which extracts the $\beta_i$ vector:

$$F_i \begin{bmatrix} \beta_{i,1} \\ \vdots \\ \beta_{i,N} \end{bmatrix} = V_i \tag{D.5}$$

- Reconstruction of the network by the LASSO:

$$\min_{\beta_i,\dots,\beta_{iN}} \left\| V_i - F_i \begin{pmatrix} \beta_{i1} \\ \vdots \\ \beta_{iN} \end{pmatrix} \right\|_2^N + \rho_i \sum_{j=1, j\neq i}^{N} \beta_{ij} \tag{D.6}$$

# E

# Info sheet

**Title of the project:** COVID-19 Prediction and Mitigation
**Name of the client organization:** TU Delft Artificial Intelligence and Networking Team
**Date of the final presentation:** 2 July 2020

## Description

**Challenge:** The goal of this project was to build an application which implements an algorithm that predicts the future course of the COVID-19 pandemic in the Netherlands and visualises the outcomes of these predictions. This project was done for the client of the TU Delft Artificial Intelligence and Networking Team.

**Research:** The research phase taught us how to combine predictions with simulations of mitigation strategies in a single visual solution. This knowledge was then applied on the design of the product, both visual and software-architecture wise, to finally end up with a product that fulfills all must-have and should-have requirements.

**Process:** We used Scrum to manage the project. Because of the ongoing pandemic, the group work was done via internet in daily voice-calls. Two members were mostly responsible for the front-end, while the others mostly worked on the back-end. The online setting initially led to inefficiency of work for all members. A meeting was scheduled to resolve this issue: all members were expected to be available in a Discord channel at strict times. This resolved the issue.

**Product:** The created product is a web application. We tested the application with results from the existing source code of NIPA written in MATLAB as well as via unit tests. We implemented all core features. The product also serves as an API endpoint for researchers who want to download data on the pandemic or want to integrate NIPA predictions in an application of their own.

**Outlook:** The program has been written to be as open-source and modular as possible, so other researchers can use the code for their own purposes. We proposed future improvements and recommendations for improved security features. The web application has been deployed and will remain available for the unforeseeable future.

## Members of the project team

**Matthias R. Meester**
Interests: Machine learning, mathematical finance
Contributions: Back-end and front-end development

**Isitan Görkey**
Interests: Cybersecurity and data science.
Contributions: Back-end development

**Olaf J. Braakman**
Interests: Data science
Contributions: Back-end and front-end development

**Thomas T. G. Rood**
Interests: Visual design and data science
Contributions: Product design and front-end development

**Niels Bauman**
Interests: Software design, algorithm optimisation.
Contributions: Front-end development and deployment.

All team members contributed to preparing the presentation and necessary deliverables. The final report for this project can be found at: http://repository.tudelft.nl.

## Info table

| Name | Role | Email |
|---|---|---|
| Dr. Huijuan Wang | Client | h.wang@tudelft.nl |
| Dr. Claudia Hauff | Coach | c.hauff@tudelft.nl |
| Matthias R. Meester | Team Member | matthiasmeester@gmail.com |
| Isitan Görkey | Team Member | isitan.gorkey@gmail.com |
| Olaf J. Braakman | Team Member | olafbraakman@hotmail.com |
| Thomas T. G. Rood | Team Member | thomasrood99@gmail.com |
| Niels Bauman | Team Member | nielsb1999@hotmail.nl |