Contents

[Introduction 1](#_Toc116770405)

[Proposed Webservice 1](#_Toc116770406)

[Data Considerations 1](#_Toc116770407)

[Note on data processing 3](#_Toc116770408)

[A note about data 3](#_Toc116770409)

[Web service format 4](#_Toc116770410)

[Common Queries 4](#_Toc116770411)

[Daily Queries 5](#_Toc116770412)

[Monthly Queries 6](#_Toc116770413)

[All Data Queries 6](#_Toc116770414)

[Proposed front end 7](#_Toc116770415)

[Web Map 7](#_Toc116770416)

[Basic Structure and governance 8](#_Toc116770417)

[Home Page (Landing Page) 8](#_Toc116770418)

[About Page 9](#_Toc116770419)

[Historic Data 10](#_Toc116770420)

[Historic View – Popup Overlay 11](#_Toc116770421)

[Predictive view 12](#_Toc116770422)

# Introduction

Student ID: **19000170**

This report explains the process of selecting and creating a new web service as required for the UHI Web Services Module.

# Proposed Webservice

The city of New York is one of the most populated cities in the world, with over 8.5 million residents. As such, traffic accidents are more common than in other cities and road safety is a concern for many residents. Many factors can lead to the likelihood of an accident, including weather.

The data for this web service is available as part of Google’s Big Query offering for free. This data was used, in addition to NOAA data, to establish correlations between weather and accidents. The resulting data set showed accidents per day (and their location) and the number of deaths and injuries to motorists, cyclists, and pedestrians. As this data was for the entire of New York City (and conditions can vary from area to area) the data was then collated by borough in New York.

Therefore, the new web service will:

1. Allow a user to view the historic accidents and any weather conditions which have occurred in New York. The result will also contain details of any injuries and deaths.
2. The service could be modified to bring in additional data to show the historical accidents at specific points in the city.
3. Finally, the Machine Learning model could be made available to the web service to allow a user to plug in values and to get predictions for accidents, injuries, and deaths.

The audience for this web service will vary greatly depending on which part of the service is being used.

Professional users could be those who are involved in planning the levels of resourcing for emergency vehicles and finding if any specific times and conditions would require additional resources to be allocated. Non-professional users could be homeowners, or those looking to purchase in an area, who would like to see, historically, just how “safe” the area is. These users would also benefit from the prediction feature, but not to the same extent as professional users.

To develop the web service, only data for 2019 and 2020 will be loaded in.

## Data Considerations

As previously noted, accident data is freely available from Google as part of their BigQuery offering. This data is part of the NYPD dataset so, in theory, should be reliable. The data is part of the “**new\_york\_mv\_collisions.nypd\_mv\_collisions**” data set.

The data provides the following information:

* borough
* contributing\_factor\_vehicle\_1
* contributing\_factor\_vehicle\_2
* contributing\_factor\_vehicle\_3
* contributing\_factor\_vehicle\_4
* contributing\_factor\_vehicle\_5
* cross\_street\_name
* timestamp
* latitude
* longitude
* location
* number\_of\_cyclist\_injured
* number\_of\_cyclist\_killed
* number\_of\_motorist\_injured
* number\_of\_motorist\_killed
* number\_of\_pedestrians\_injured
* number\_of\_pedestrians\_killed
* number\_of\_persons\_injured
* number\_of\_persons\_killed
* off\_street\_name
* on\_street\_name
* unique\_key
* vehicle\_type\_code1
* vehicle\_type\_code2
* vehicle\_type\_code\_3
* vehicle\_type\_code\_4
* vehicle\_type\_code\_5
* zip\_code

It should be noted that the data is not always complete. Sometimes, only the time of the accident has been recorded, sometimes the location is missing – this will be covered later.

The proposed service won’t care about a number of these fields, specifically, it **will** care about:

* Borough
* Timestamp
* Latitude
* Longitude
* number\_of\_cyclist\_injured
* number\_of\_cyclist\_killed
* number\_of\_motorist\_injured
* number\_of\_motorist\_killed
* number\_of\_pedestrians\_injured
* number\_of\_pedestrians\_killed
* number\_of\_persons\_injured
* number\_of\_persons\_killed

It will also care about the weather conditions. Again, this is available from Google and is part of the NOAA dataset.

Putting the data and the weather together identifies a problem. Some collision data has a borough, some data has only latitude and longitude and some have neither of these. Using some SQL with another dataset, the borough can be found based on the latitude and longitude. Those which have no location data simply have to be discarded. This is approximately 10% of all the data (though this web service is using only 2 years of data, the figure will be smaller.

We can create a data source for the web service with this data. The data source will have 5 records for each day of the 2 years. Why five records? New York City has five boroughs. This means our web service will be able to provide borough-specific data. The JSON files used (2019.json and 2020.json) are included in the “data” directory.

This data would be enough to create a basic MVP which would be consumed by a front end, the MVP would cover point 1 of the requirements.

### Note on data processing

It will be noted in the sample JSON files that there are two elements for the date. One of these is simply called “date” and the other is called “collision\_date.” To aid searching, the “date” field will be kept as a string, however, the “collision\_date” field will be used for sorting and will need to be a date. JSON, however, has no concept of date. So to fix this, this data will be imported into MongoDB and then a script ran over the data to modify the field type. The script used is shown below:

db.SampleData.updateMany(  
  {},  
  [{ "$set": { "collision\_date": { "$toDate": "$collision\_date" } }}]  
);

This script will loop over every document in the SampleData collection and take the collision\_date field, change this to a date and repopulate the new field with the value of the old field.

### A note about data

The data which is being used is, as noted, available from Google. This data has been used as part of another module for the course. The git repo (and zip) contains some custom python which is used to create the JSON which will be imported. The python script will do 99% of the work (with a simple search-replace operation needed to insert a comma between elements, which makes the file ready for import to MongoDB and then the collision\_date conversion as noted above.

## Web service format

With only 2 years of data in this web service, there are still over 3,700 records. And with 28 fields that could be returned, GraphQL is the technology to use. This is because GraphQL will return only what is requested, unlike with a REST API where the code on the front end would have to disregard information at the point of consumption.

All historic data would be stored in a Mongo Database.

Proposed queries (subject to change during development)

### Common Queries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Proposed Call** | **Description** | **Notes** |
| getBoroughNames | Query | {  getBoroughNames {  borough  }  } | This query will return a distinct list of all boroughs in the dataset. | This should always return 5 elements. |
| getMinDateEntry | Query | {  getMinDateEntry(borough:"QUEENS"){  date  }  } | This query will return the earliest entry of a recorded incident in the database | Will require the borough name as a parameter |
| getMaxDateEntry | Query | {  getMaxDateEntry(borough:"QUEENS"){  date  }  } | This query will return the latest entry of a recorded incident in the database | Will require the borough name as a parameter |

### Daily Queries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Proposed Call** | **Description** | **Notes** |
| getBoroughSingleDayDeaths | Query | {  getBoroughSingelDayDeaths(borough:"QUEENS", date:"2019-01-25"){  date  deaths  }  } | This query will return the deaths in a specific borough on a specific date. | Will require both the borough name and the date as parameters. |
| getBoroughSingleDayInjuries | Query | {  getBoroughSingleDayInjuries(borough:"QUEENS", date:"2019-01-15"){  date  injuries  } | This query will return the injuries in a specific borough on a specific date. | Will require both the borough name and the date as parameters. |
| getSingleDayWeather | Query | getSingleDayWeather(borough:"QUEENS", date:"2019-01-21") {  date,  temp,  fog,  windspeed,  rainfall  }  } | This query will return any weather conditions recorded for the given date. | Will require both the borough name and date as parameters. |
| getAccidentData | Query | {  allAccidentData(borough:"QUEENS", date:"2019-01-01"){  id  date  borough  weekday  year  month  day  collision\_date  temp  dewp  slp  visib  wdsp  mxpsd  gust  max  min  prcp  sndp  fog  cyc\_kill  cyc\_injd  moto\_kill  moto\_injd  peds\_kill  peds\_injd  pers\_kill  pers\_injd  num\_cols  }  } | This query will return all accidents (injuries, deaths, and times) for a given date and borough. | This will require both borough and date as parameters. |

## Monthly Queries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Proposed Call** | **Description** | **Notes** |
| getDeathsDeathsMonth | Query | {  getBroroughDeathsMonth(borough:"QUEENS", year:2019, month:1) {  deaths  }  } | This query will return the total number of deaths in a specific borough for a specific month and year. | Will require both the borough name and the month and year as parameters. |
| getBroroughInjuryMonth | Query | {  getBroroughInjuryMonth(borough:"QUEENS", year:2019, month:1) {  totalInjuries  }  } | This query will return the total number of injuries in a specific borough for a specific month and year. | Will require both the borough name and the month and year as parameters. |

## All Data Queries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Proposed Call** | **Description** | **Notes** |
| getBoroughDeaths | Query | {  getBroroughDeaths(borough:"QUEENS"){  date  deaths  }  } | Returns an array of the dates and number of deaths per day | Will require the borough name as a parameter. |
| getBoroughInjuries | Query | {  getBoroughInjuries(borough:"QUEENS"){  date  injuries  }  } | Returns an array of the dates and number of injuries per day | Will require the borough name as a parameter. |

# Proposed front end

To present data to the user a basic react application could be constructed.

The following screens illustrate a **proposed** data flow, with proposed ways to visualise the data.

## Web Map

The proposal is that the front end is constructed using react. This should enable faster prototyping and allow handling of the required data and connections to Mongo (and other external sources as needed) and is performant (at least with the data which is being proposed.)

The map shown on the left shows 4 “pages” however the data would be shown on a popup on those pages.

A stretch goal could that the predictive view be placed behind some form of password protection. This is not in the scope of this module and would be added should sufficient time be available.

### Basic Structure and governance

At this stage, only a basic idea of the structure of the project is possible. It would be proposed to create the application in such a way as common items are kept as components. Immediately this would mean the navbar.

Pages would be stored in their directory also as would images. Setting the correct structure would enable other developers to get up to speed much more quickly.

As per other projects, the creation of a basic Kanban board and storing code in Git would be good practice.

### Home Page (Landing Page)

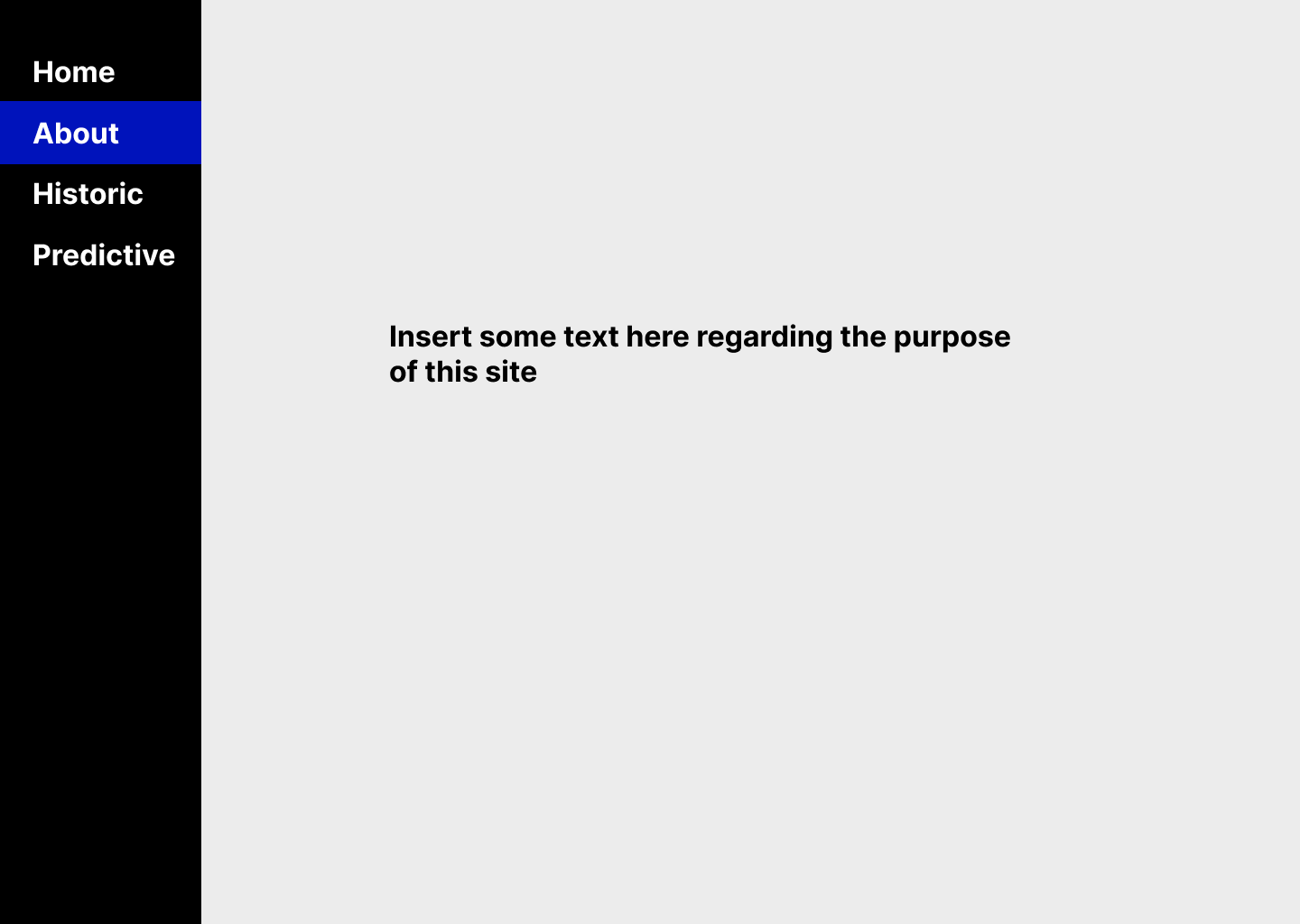
The landing page for users coming to the site could look something like below:

Map

Description automatically generated

The welcome page serves a very simple, but very important function. It would welcome the user to the site and be easy enough to navigate. This basic proposed layout would show the map of the 5 boroughs of New York, have the title of this service, and give links further in. It should be noted that the “Home” and “Landing Page” are the same thing, at any time if the user clicks the “Home” link they would be redirected to this Landing Page.

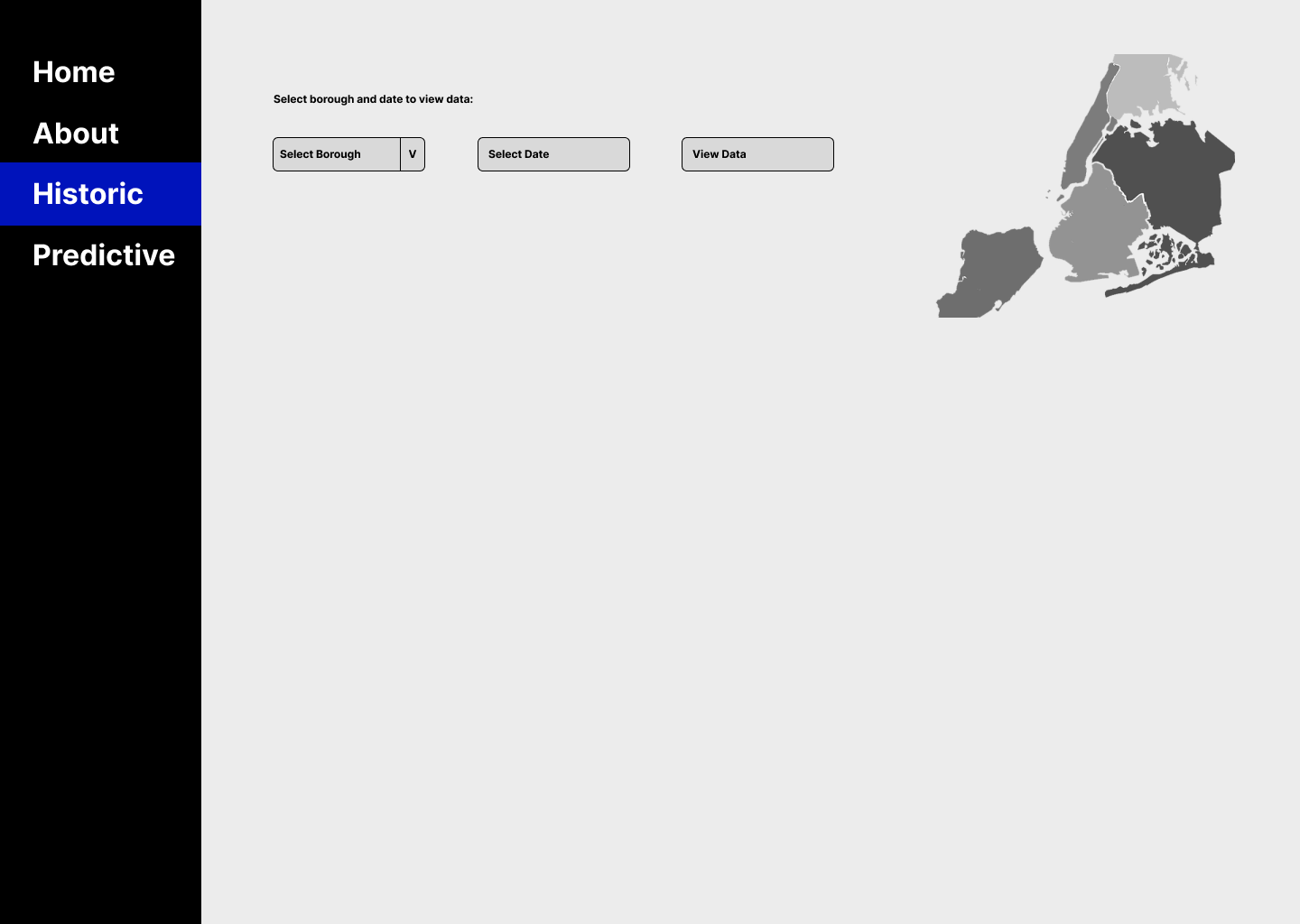
### About Page



On a commercial application, this page would display licences, data sources, and some help text. At the time of creating this document, this help text is not available, so the screenshot simply shows “insert some text here.”

For this proof of concept, this page could also contain links to the git repos used in this project.

### Historic Data



This is where the proof of concept will start to consume the data which was created and served via the web service.

The user will need to select both a borough and a date before the “View Data” button is enabled.

The borough dropdown list will be populated from the web service with the **getAllBoroughs** query (as noted above.) The user will only be allowed to select a single borough at a time.

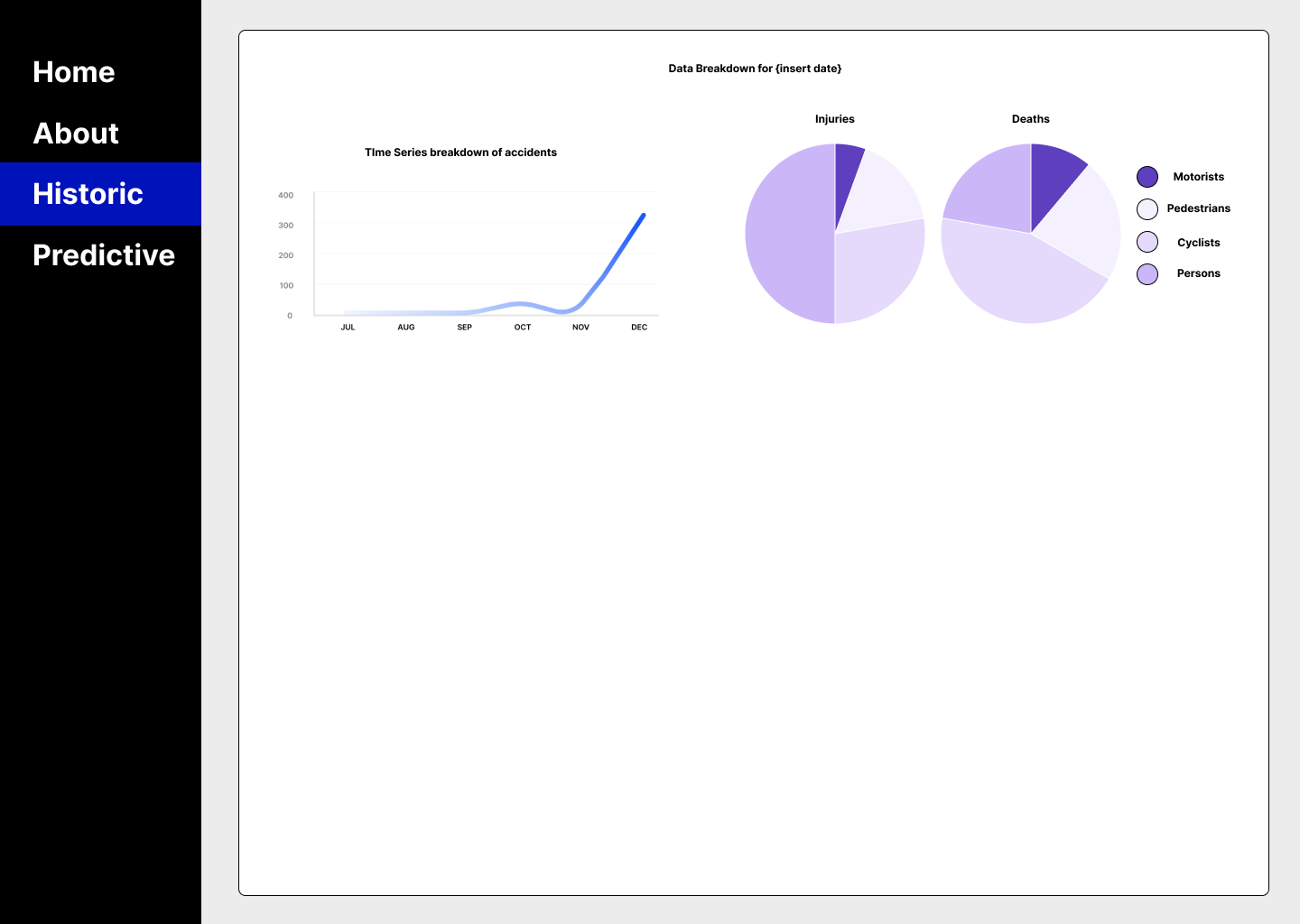
The Select date will create a calendar of some description with the earliest and latest date populated via the **getMinDateEntry** and **getMaxDateEntry** queries. Exactly how this calendar will work is unknown at this time. It will require a spike and some investigation on if a calendar component exists in the react ecosystem which could be used.

As noted, the View Data button will be disabled until both a borough and a date have been selected.

An optional goal, time permitting, would be to take the PNG of the five boroughs of New York and to create geometry to allow highlighting of the map to take place when the user interacts with the Select borough dropdown.

Once the user clicks the View Data then a popup overlay will appear.

### Historic View – Popup Overlay



Here is where the data we are consuming will be shown to the user in visual form.

Starting with the pie charts would be populated from the either getBoroughDeathsMonth or the getBoroughInjuriesMonth queries.

There will be a line chart showing the times of accidents against the number. While no two accidents will happen at the same time, the points will form a trend. In addition, depending on the graphing library used, a popup would be available on hovering over the data points which could show, for that time, the number of injuries and deaths which occurred. This data would be fetched using the allAccidentData query.

On this page will also be some icons to denote the weather conditions during the day. As yet how this will show is unclear.

#### Update

On further consideration, the historic page could be modified to not have a popup with data but a tabbed view. This view could look something like the below:

Graphical user interface, application

Description automatically generated

The Daily Stats tab would have the same graphics as the first iteration, the monthly would have the trend and the pie charts, and new endpoints will be needed which will take borough and month/year combination.

## Predictive view

This is very much a “stretch goal” and will require some investigation as well as experimentation. The aim here would be to host, somewhere, a tensor flow model and for the user to be able to input a variety of factors and to return the number of accidents (predicted) – there would be no need for graphs, etc – this is very much just text on the page.

How this would work is very much unknown at this time. There will be no database, as the values would be populated because of user input.