Comprehensive Monkeypox Dashboard

Software Design Document

version 1.0

Prepared by
Saketh Dendi, Felix Huang, and Sakin Kirti (Dendi Huang Kirti Group)
CSDS 393
4 October 2022

Table of Contents

Introduction

As stated in our SRS, the purpose of this project is to provide a comprehensive analysis and visualization of the Monkeypox virus spread throughout the United States on a national and a state by state level. This is so that everyday people can quickly access any relevant statistical information they need about the spread of the virus in the United States. This website will stay online for the duration of the Monkeypox epidemic or until it no longer is a concern to the public's health.

1.1 Purpose

In order to keep our project on schedule and keep our development process flowing smoothly we have developed this document to illustrate the components that will make up our website and their functions. The following table shows our thought process behind the development.

Table 1:

Section	**Content**
Section 2	The 7 principle classes and components that make up our project
Section 3	Illustrate and explain the main interactions between the classes and database
Section 4	Explains the contents and methods of the classes and components interfaces

We have divided each of our 7 principle classes and components into front end and back end classifications. These classifications are shown below:

Table 2:

Front End	**Back End**
MapView	DatasetCleaner
ChartView	DatabaseUpdater
PredictionFetch	DatabaseSyncher
	MonkeypoxPredictor

The third section covers all the interactions between our classes that we need in order to make our website function autonomously.

In the fourth section we describe all of the functions that we so far have intended to include in our final design. This includes functions to update the database, a function to create UI elements, and functions that interact with different classes and a few others.

1.2 Document Conventions

The formatting of this document is done the following way to stay consistent and allow any viewer to quickly orient themselves.

- Titles: Google Docs' *Title* Format (26pt Calibri, black)
- Section Headers: Google Docs *Header 1* Format (20pt Calibri, black)
- Subsection Headers: Google Docs' *Header 3* Format (14pt Calibri, dark gray 4)
- Body: 12pt Calibri, black

1.3 Introduction Notes

Please note that this is a shortened version of our Introduction. To access the full version, please see section 5.1.

Principal Classes and Components

2.1 MapView

The front-end design contains a MapView component, which is responsible for the following:

- Rendering the interactive U.S. map on the main page of the website
- Populating the tables for nationwide, statewide, and statistics changes with the latest cleaned Monkeypox data

The MapView component will provide the interface for the PF-1 [Daily U.S. Map of Monkeypox Statistics] product function and the functional requirements specified in section 4.1.3 of the Software Requirements Specification document and will be written in JavaScript.

2.2 ChartView

The front-end design contains a ChartView component, which is responsible for the following:

- Rendering an interactive time series chart of confirmed cases of Monkeypox for a given state
- Populating the tables for statewide and statistics changes with the latest cleaned Monkeypox data

The ChartView component will provide the interface for the PF-2 [Monkeypox Statistics and Trends Charts] product function and will be written in JavaScript as specified in the Software Requirements Specification document.

2.3 PredictionFetch

The front-end design contains a PredictionFetch component, which is responsible for the following:

- Fetching from the backend database the cleaned predictive analytics data: aggregate prevalence rate, incidence rate, and case-fatality ratio predictions for the following week
- Fetching the 14-day confirmed cases progression for a given state

The PredictionFetch class component will serve as the main load to the front-end of the website the necessary data needed to be displayed and will be written in JavaScript.

2.4 DatasetCleaner

Upon fetching the new dataset from Global.Health, the messy dataset has unneeded and dirty data. This class takes responsibility for the following:

- removing the unnecessary data
- cleaning the data into a format that is easy to produce analytics from

The DatasetCleaner object will be written in python and will get the Global.Health dataset as a pandas dataframe and will produce a cleaned pandas dataframe as output.

2.5 DatabaseUpdater

The DatabaseUpdater class is responsible for the following:

- taking the pandas dataframe produced by DatasetCleaner and producing different aggregations of the data and storing them as different relations in our backend database.

This will make updating the front-end easier. This class will be written in python and will produce relations in the database.

2.6 MonkeypoxPredictor

The MonkeypoxPredictor class is responsible for the following:

- generating predictions for the 14-day confirmed case progression
- Predictions for the 14-day death count progression
- Predictions for aggregate prevalence rate, incidence rate, and cast-fatality ratio for the following week.

It will also produce updates to our backend database as outputs. This class will be written in python.

2.7 DatabaseSyncher

The Database Syncher is a python class that is designed to copy an online global health monkeypox database and continually update our own local database everyday in order to keep the local database up to date. It will run on a cron timer that updates daily in order to get the most up to date monkeypox statistics. It will also account for when our database and the online database stop synching for whatever reason.

Interactions Between Components

3.1 DatabaseSyncher and DatabaseUpdater

Description: The DatabaseSyncher runs continuously on a cron timer. When 24 hours have gone by, it calls DatabaseUpdater to update our backend database. Before fully updating the database, the DatabaseUpdater calls on the DatasetCleaner and Monkeypox Predictor to clean the data, and create predictions for the next two weeks to come.

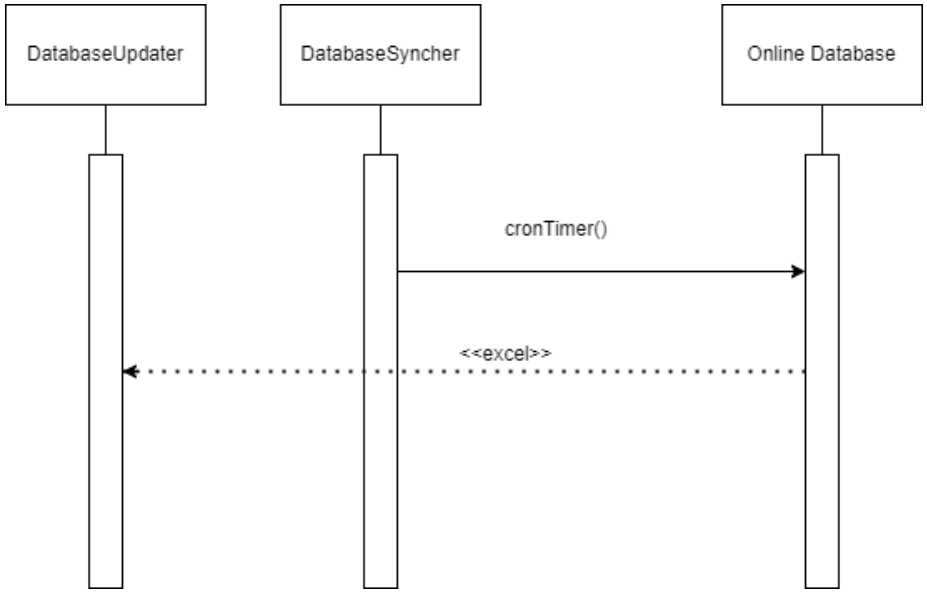

Figure 1: Sequence diagram for updating the local database

3.2 DatabaseUpdater and DatasetCleaner

Description: When the DatabaseUpdater is called from the DatabaseSyncher, it retrieves the raw data from the global.health API. Afterwards, it passes this data to the DatasetCleaner which performs necessary changes to this dataset and returns the cleaned data back to the DatabaseUpdater.

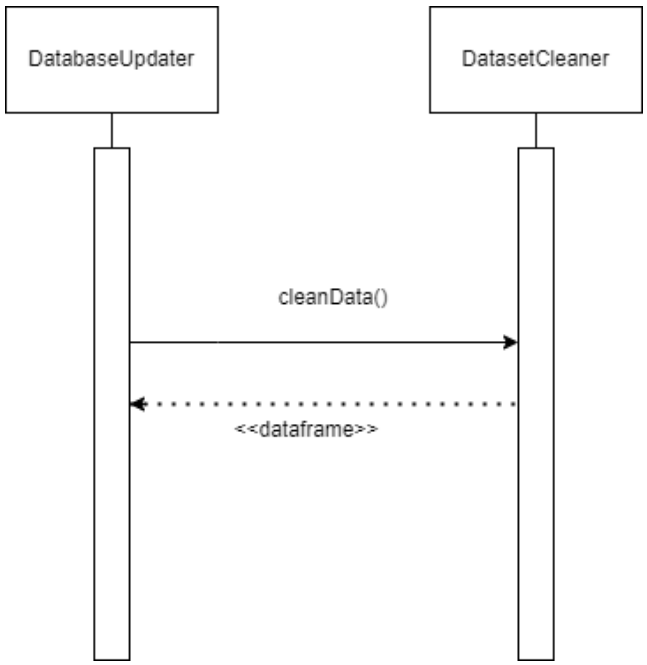

Figure 2: Sequence diagram for cleaning Monkeypox data

3.3 DatabaseUpdater and MonkeypoxPredictor

Description: After the cleaned data is returned to the DatabaseUpdater, the MonkeypoxPredictor is called on to provide data extrapolations and predictions for the coming weeks. This data is also returned to the DatabaseUpdater in the form of an extended set of rows in the dataset with a new column that denotes whether the row is a confirmed data point or predicted data point.

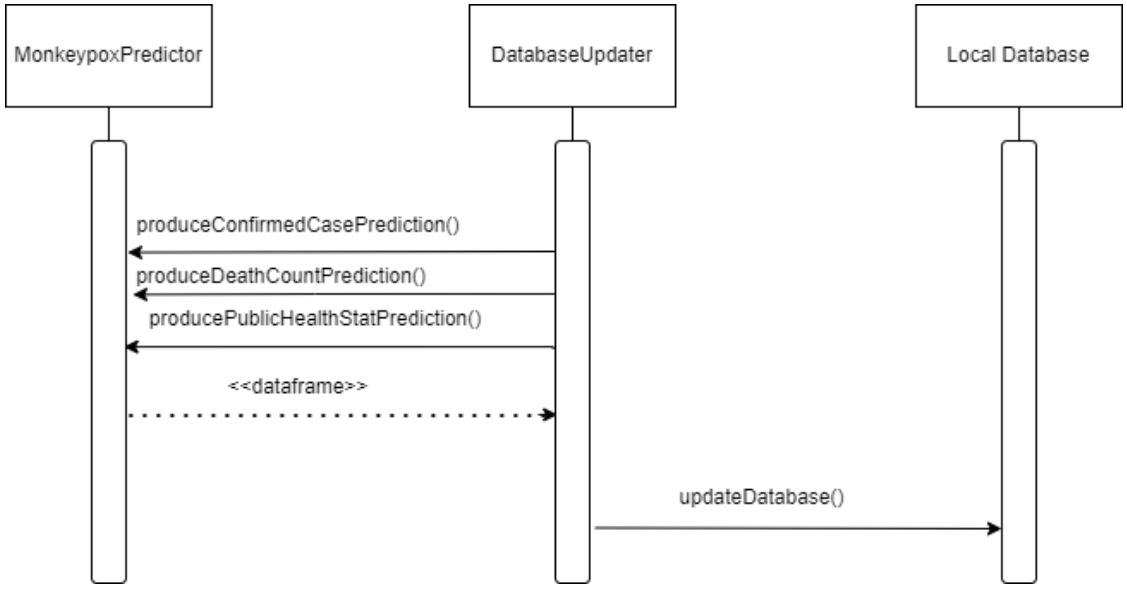

Figure 3: Sequence diagram for producing predictions from monkeypox data

3.4 DatabaseUpdater and Database

Description: After cleaning and performing predictions, the DatabaseUpdater then replaces the previous data stored in the database with the new updated data that includes the clean data as well as the predictions from the data.

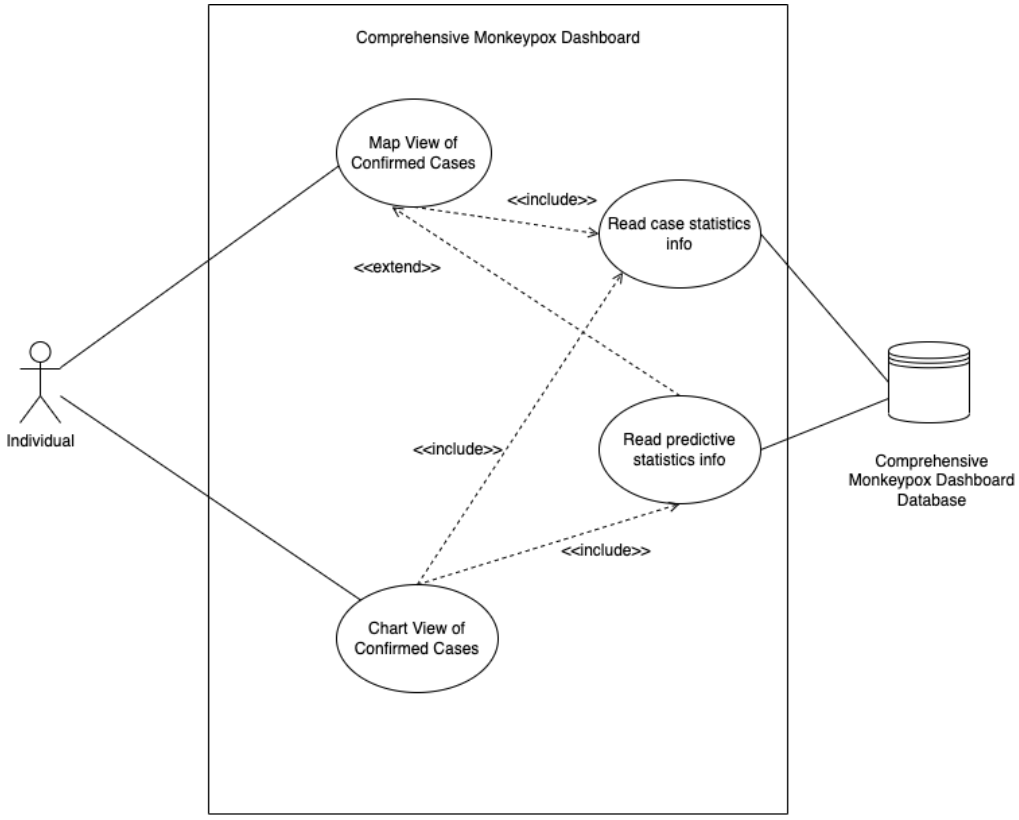

Figure 4: High-level overview of system use case diagram for website

3.5 MapView and PredictionFetch

Description: In the interaction between MapView and PredictionFetch, the PredictionFetch component is used to provide to the MapView component the relevant predictive Monkeypox statistics that PredictionFetch fetches from the backend database. MapView will call the getStatsChanges(String state) method of PredictionFetch to retrieve the relevant data to display.

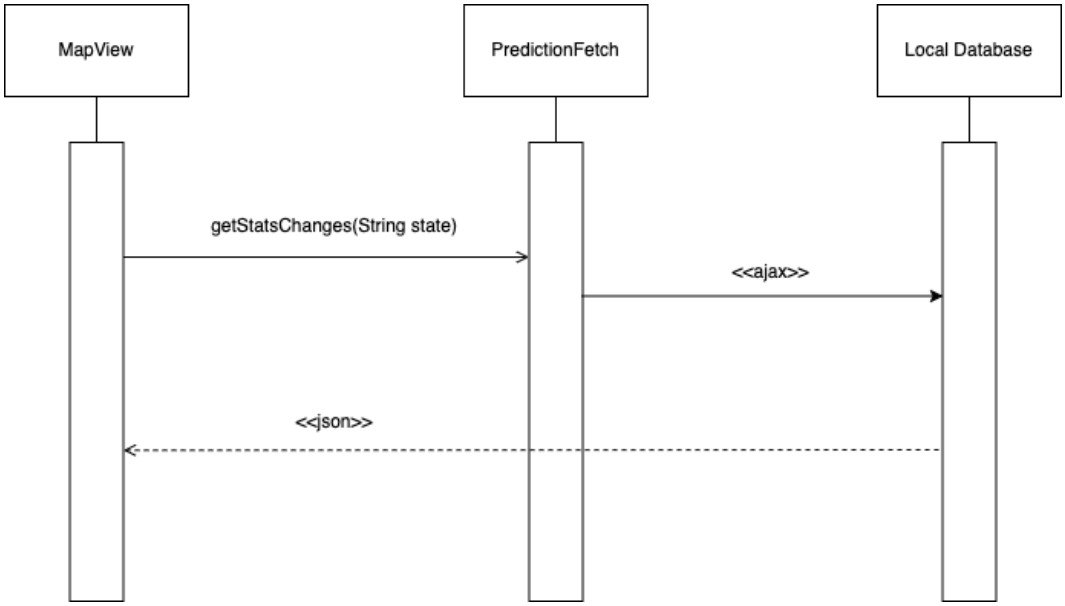

Figure 5: Sequence diagram for viewing statistics changes on the map page

3.6 MapView and Database

Description: The MapView component will retrieve the cleaned and updated Monkeypox confirmed cases data from the backend database using a GET request.

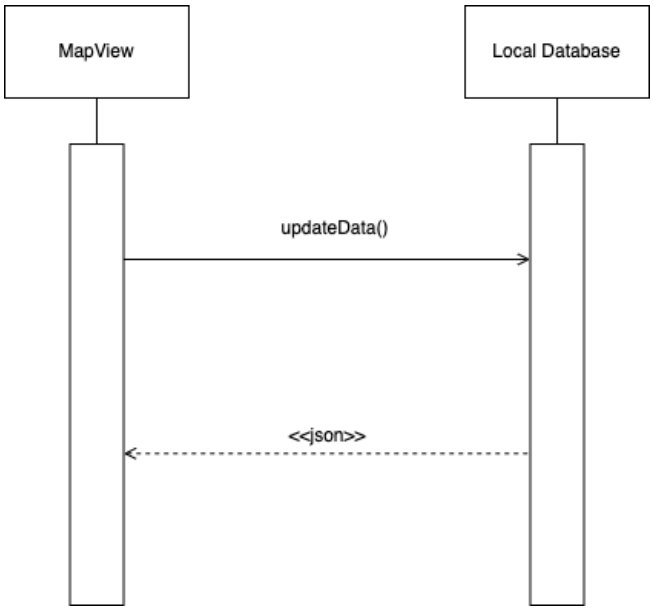

Figure 6: Sequence diagram for viewing confirmed cases data on the map

3.7 ChartView and PredictionFetch

Description: In the interaction between ChartView and PredictionFetch, the PredictionFetch component is used in a similar manner to how it is used for MapView. The ChartView component will retrieve the relevant predictive Monkeypox statistics that PredictionFetch fetches from the backend database. MapView will call both the getProgression(String state) and getStatsChanges(String state) methods of PredictionFetch to retrieve the progression data for the chart and statistics changes.

3.8 ChartView and Database

Description: The ChartView component will retrieve the cleaned and updated Monkeypox confirmed cases data from the backend database using a GET request.

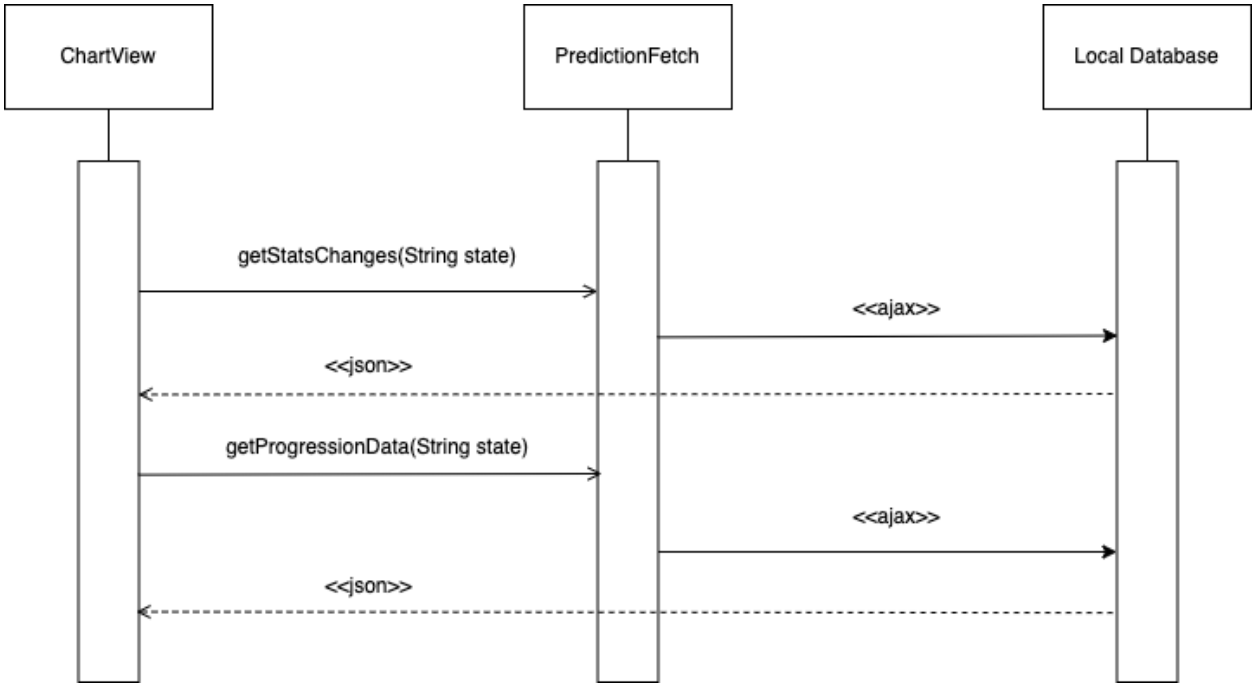

Figure 7: Sequence diagram for viewing chart progression and predictive statistics

3.9 PredictionFetch and Database

Description: The PredictionFetch component will retrieve the cleaned and updated predictive Monkeypox statistics from the backend database using a GET request.

Class and Component Interfaces

4.1 MapView

The MapView is one of the primary functions of the site where the user will be able to view an interactive map of the United States that can pull up the state-by-state Monkeypox statistics and graphs.

4.1.1 boolean updateData()

This method will load the latest parsed and formatted nationwide and statewide Monkeypox data from the local database of the website. The method will return true if the latest data was able to be retrieved and false otherwise.

4.1.2 React.Element displayMap()

This method will render a U.S. map with a legend of confirmed cases of Monkeypox.
for the main page of the website and color code each state according to the legend.

4.1.3 React.Element handleUserAction(String action)

This method will change and render a new display of the map according to a given interactive map action (e.g. zoom in, zoom out, drag, hover, etc.) as it pertains to the functional requirements INT-1 [Interact.Hover], INT-2 [Interact.Move], INT-3 [Interact.Zoom.In], and INT-4 [Interact.Zoom.Out].

4.1.4 React.Element displayWindow(String state)

This method will render a small window of the timeseries chart of the confirmed cases of monkeypox for a given state since the first confirmed case of monkeypox. The method will pertain to INT-1 [Interact.Hover] and STAT-3 [Stat.View].

4.1.5 React.Element displayData()

This method will compile and display tables for total nationwide confirmed cases, states by confirmed cases, and national statistics changes in prevalence rate, incidence rate, and case-fatality ratio for the previous week and next week.

4.2 ChartView

4.2.1 boolean updateData()

This method will load the latest parsed and formatted nationwide and statewide Monkeypox data from the local database of the website. The method will return true if the latest data was able to be retrieved and false otherwise.

4.2.2 React.Element displayChart(String state)

This method will display a timeseries chart of the confirmed cases of monkeypox for a given state since the first confirmed case of monkeypox. Projected progression data will be converted into graphable data points to serve as an extension of the chart data.

4.2.3 React.Element handleHover(String state, String date)

This method will fetch the statewide confirmed case count for a specified date and state and display a small window containing the date formatted as “MMM d, yyyy” and the case count for that date. This method pertains to STAT-2 [Stat.Hover].

4.2.4 React.Element displayData(String state)

This method will display tables for a specified state: the total nationwide confirmed cases and deaths to date, states by confirmed cases, and national statistics changes in prevalence rate, incidence rate, and case-fatality ratio for the next week.

4.3 PredictionFetch

4.3.1 JSON getProgression(String state)

This method will fetch from the backend the confirmed case count data for a given state for each day in a 14-day projected progression and return the results. The retrieved data will be returned as a JSON object.

4.3.2 JSON getStatsChanges(String state)

This method will fetch from the backend the aggregate prevalence rate, incidence rate, and case-fatality ratio nationwide data for the previous week for a given state as well as the predictive analytics data for the following week. The retrieved data will be returned as a JSON object.

4.4 DatasetCleaner

4.4.1 void cleanData(pandas.DataFrame)

This method takes the global.health dataframe and performs several cleaning functions including:

1. Isolating United States-only data
2. Generating a new column for the state where the case originates
3. Cleaning and reformatting any yes/no columns
4. Removing unnecessary columns to reduce time/space for database
5. Performing data checks to make sure that each column is of the right data type

4.4.2 void isolateUSData(pandas.DataFrame)

This is a helper method that performs the data isolation for point 1 in the *void cleanData(pandas.DataFrame)* section.

4.4.3 void cleanColumns(pandas.DataFrame)

This is a health method that performs the cleaning and reformatting columns, removing unnecessary columns, and performing data checks to make sure that each column is of the right data type as a part of the *void cleanData(pandas.DataFrame)* method.

4.5 DatabaseUpdater

4.5.1 void updateDatabase()

After the DatasetCleaner cleans the dataset, this method from this class updates the previous database with the new values updated after the date change. Note that this is its own object because a series of helper methods will be imperative to ensure that data is not replaced, but rather the database is updated in a way that preserves time and space complexities.

4.5.2 pandas.DataFrame produceConfirmedCaseRelation()

This method produces a pandas DataFrame that contains the relation describing date and confirmed case counts in order to update the database.

4.5.3 pandas.DataFrame producePublicHealthStatistics()

This method calculates the relevant public health statistics such as incidence rate, prevalence rate, and case-fatality ratio. Additionally, this will be updated in the database.

4.6 MonkeypoxPredictor

4.6.1 pandas.DataFrame produceConfirmedCasePrediction()

This method will produce the predictions for confirmed cases for the next two weeks based on an extrapolation of data provided in the previous two weeks.

4.6.2 pandas.DataFrame produceDeathCountPrediction()

This method produces the predictions for death counts for the next two weeks based on an extrapolation of data provided in the previous two weeks.

4.6.3 pandas.DataFrame producePublicHealthStatPredictions()

This method produces predictions for what the incidence rate, prevalence rate, and case-fatality ratio will look like in the next week.

4.7 DatabaseSyncher

4.7.1 void pullDatabase()

This method will pull a copy of the online global health database for the Dataset Cleaner to use. It will run every day at the same time based on when the database is updated.

4.7.2 void cronTimer()

This method will be in charge of running a cron timer in the background and execute the sync function once every day. The method will also contain the conditions to run the sync function as well.

4.7.3 void backupCron()

This method runs if the cronTimer goes offline or the backend goes offline. It is a failsafe so that the backend database stays functioning even if the online database we synch from goes offline.

Appendix

This section contains our supplementary introduction which contains additional details of our introduction and progress log of which individual worked on which sections of the document along with a glossary of necessary terms.

5.1 Supplementary Introduction

This section contains our full-length introduction as feedback from our submitted SRS suggested.

As stated in our SRS, the purpose of this project is to provide a comprehensive analysis and visualization of the Monkeypox virus spread throughout the United States on a national and a state by state level. This is so that everyday people can quickly access any relevant statistical information they need about the spread of the virus in the United States. This website will stay online for the duration of the Monkeypox epidemic or until it no longer is a concern to the public's health.

5.1.1 Purpose

In order to keep our project on schedule and keep our development process flowing smoothly we have developed this document to illustrate the components that will make up our website and their functions. The following table shows our thought process behind the development.

Table 1:

Section	**Content**
Section 2	The 7 principle classes and components that make up our project
Section 3	Illustrate and explain the main interactions between the classes and database
Section 4	Explains the contents and methods of the classes and components interfaces

As stated in table one, section 2 provides explanations of all the different principle classes we will be using in this project which includes, MapView, ChartView, PredictionFetch, DatasetCleaner, DatabaseUpdater, MonkeypoxPredictor, and DatabaseSyncher. We have divided these classes into front end and back end classifications. The classes that provide more interface facing functions will be regarded as front end classes whereas the classes that work with the database in the background will be the back end classes. For example the Mapview will be a frontend class because its objective is to output a map interface for the User to view. In the same breath the DatabaseSyncher will be a backend class because all its functions work with the backend database.

Front End	**Back End**

MapView	DatasetCleaner
ChartView	DatabaseUpdater
PredictionFetch	DatabaseSyncher
	MonkeypoxPredictor

The third section covers all the interactions between our classes that we need in order to make our website function autonomously. There are several interactions which include DatabaseSyncher to DatabaseUpdater, DatabaseUpdater to MonkeypoxPredictor, and MonkeypoxPredictor to PredictionFetch among others. Each interaction is illustrated using a sequence diagram which shows the process of how the database interacts with a class and then that class interacts with another class.

In the 4th section we are describing all of the functions that we so far have intended to include in our final design. This includes functions to update the database, a function to create UI elements , and functions that interact with different classes and a few others. Each function has their own parameters and some interact with other classes. This ensures that our front and back end will correspond no matter the situation.

5.1.2 Document Conventions

The main body of this document is written in 12pt Calibri font and all titles and section headers are created using Google Docs' heading text formatting. These correspond to 26pt font for titles and 20pt font for section headers. Any other additional subheaders are sized relative to their specificity (more specific subtopics are in smaller font). While all classes and components listed here are high priority, any bolding is done to draw the attention of the reader and provide additional emphasis.

5.2 Inspection Report

This section of the appendix contains a progress log detailing the parts that each individual worked on along with a short description of the overall breakdown of how much of the document that each team member had a hand in producing.

5.2.1 Progress Log

FH = Felix Huang, SD = Saketh Dendi, SK = Sakin Kirti

Date	Person	Item
09/27/2022	FH, SD, SK	Project meeting to break up sections to work on
09/27/2022	FH	Outlined table of contents
09/28/2022	FH	Added front-end components (MapView, ChartView, and PredictionFetch) to corresponding descriptions in principal components and component interfaces
09/28/2022	SK	Added template for inspection log and overall inspection report
09/28/2022	SK	Wrote analytics-related classes (DatasetCleaner, DatabaseUpdater, MonkeypoxPredictor) to the Principal Classes and Component Interfaces
10/01/2022	SD	Wrote class component description and methods for DatabaseSyncher
10/02/2022	FH, SD, SK	Project meeting to discuss class and component interactions
10/02/2022	SK	Created interaction descriptions for backend interactions (i.e. DatabaseUpdater and Database)
10/02/2022	FH	Created interaction descriptions for frontend interactions (e.g. MapView and ChartView with PredictionFetch and Database)
10/02/2022	SD	Wrote introduction
10/3/2022	SD	Created System Diagrams for interactions
10/3/2022	FH	Created overall system use case diagram for website
10/3/2022	FH	Created sequence diagrams for front-end component interactions
10/4/2022	FH, SD, SK	Project meeting to finalize report
10/4/2022	SK	Based on feedback from SRS, moved majority of introduction to Appendix and reduced the Introduction to bullet point format
10/4/2022	SK	Wrote the work breakdown (5.1.2)
10/4/2022	FH, SD	Reviewed inspection log and work breakdown (5.1.1, 5.1.2) to ensure accuracy

5.2.2 Work Breakdown

This document was worked on very evenly. Saketh performed 35% of the work, writing the introduction and discussing the backend of the program. Sakin performed 35% of the work, including the inspection report work breakdown and analytics script sections. Felix worked on 30% of the document, writing the main front-end components section of the document. Overall, the project was split evenly between group members.