Team Project Report

Ryan Connolly-Kelley; Gabriel Valenzuela; Carlos Mina Lopez; Paul Carmichael; Natasha Krisa

CSE360 Section #23427

Project Management

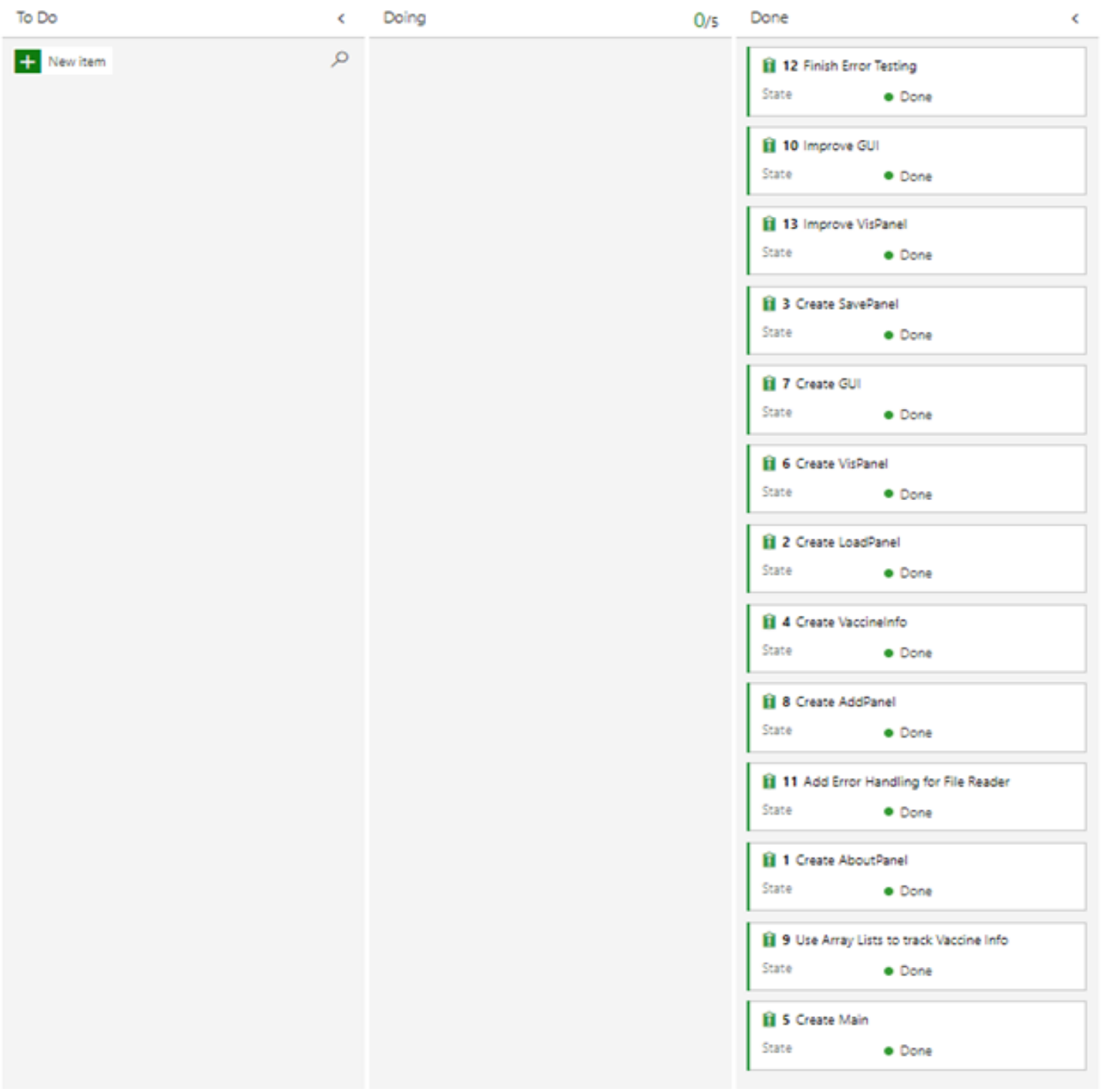
We utilized Agile development methods while completing this project. We chose to use the Agile methodology because of the nature of the project – relatively small with limited development time. By using the Agile model and taking an iterative approach to development, our team was able to quickly implement necessary functionality, while still preventing new errors from emerging or prior functionality from being lost.

Because this project had to be completed quickly, the ability to work without producing large volumes of documentation was necessary. Furthermore, as the availability of our developers varied, it was necessary that our process model enable developers to work on their own sections of the code on their own time. Thus, a more rigid process model would not have suited our needs.

We used the Model-View-Controller Architecture Model for the application. This architecture model consists of three components: the Model, which stores and manages data; the View, which is the GUI, and the Controller, which maps user actions to model updates. In our application, an ArrayList is used to store vaccine information, so it is our Model. The GUI is our application's GUI. The classes that are responsible for loading, adding, saving, and visualizing data act as our Controllers.

The components of this project were split among different team members. Gabriel Valenzuela worked on development. He also created and managed our group Discord, which we used to communicate more effectively while working on the project. Paul Carmichael worked on development. He also created our group Github repository and handled access control for it. Natasha Krisa worked on development. Carlos Mina Lopez worked on development. Ryan Connolly-Kelley worked on the report.

Progress was monitored with a Kanban chart. We chose to use a Kanban chart to monitor progress due to its simplicity. It worked well with our Agile process model, as it allowed team members to easily see what components of the project needed to be completed, and does not depend on components being completed in a specific order. Below is a screenshot of the Kanban chart as of 4/20/21, when the project was completed. A hard deadline of 04/20/2021 was set to allow a day to fully test the program before submission.


To Do
New item
Doing
0/5
Done
12 Finish Error Testing
State
Done
10 Improve GUI
State
Done
13 Improve VisPanel
State
Done
3 Create SavePanel
State
Done
7 Create GUI
State
Done
6 Create VisPanel
State
Done
2 Create LoadPanel
State
Done
4 Create VaccineInfo
State
Done
8 Create AddPanel
State
Done
11 Add Error Handling for File Reader
State
Done
1 Create AboutPanel
State
Done
9 Use Array Lists to track Vaccine Info
State
Done
5 Create Main
State
Done

Requirements

The goal of this project is to build a GUI application that will get and display information about COVID-19 vaccinations. The vaccine information is provided in the form of .csv files. This application must provide 5 buttons; Load Data, Add Data, Save Data, Visualize Data, and About.

Load Data must ask the user for the location of a .csv file, then read the file and load the data into the main table. File is expected to be in proper format.

Add Data must provide fields for the user to enter vaccine information, which is then added to the main table. IDs need to be 5 numbers in length and cannot include special characters or letters. Data cannot be added if fields are left blank. If IDs match, the new addition overwrites the old.

Save Data must save all information in the table, including user-entered information, to a .csv file.

Visualize Data must display 2 charts. One must be a bar chart that displays the number of vaccines of each type administered. The other must be any chart that can display how many vaccinations take place at each location.

About must display a page with information about the team members who worked on the project.

User Stories:

As a healthcare worker, I want to load new patient vaccination data into the application so I can access it later.

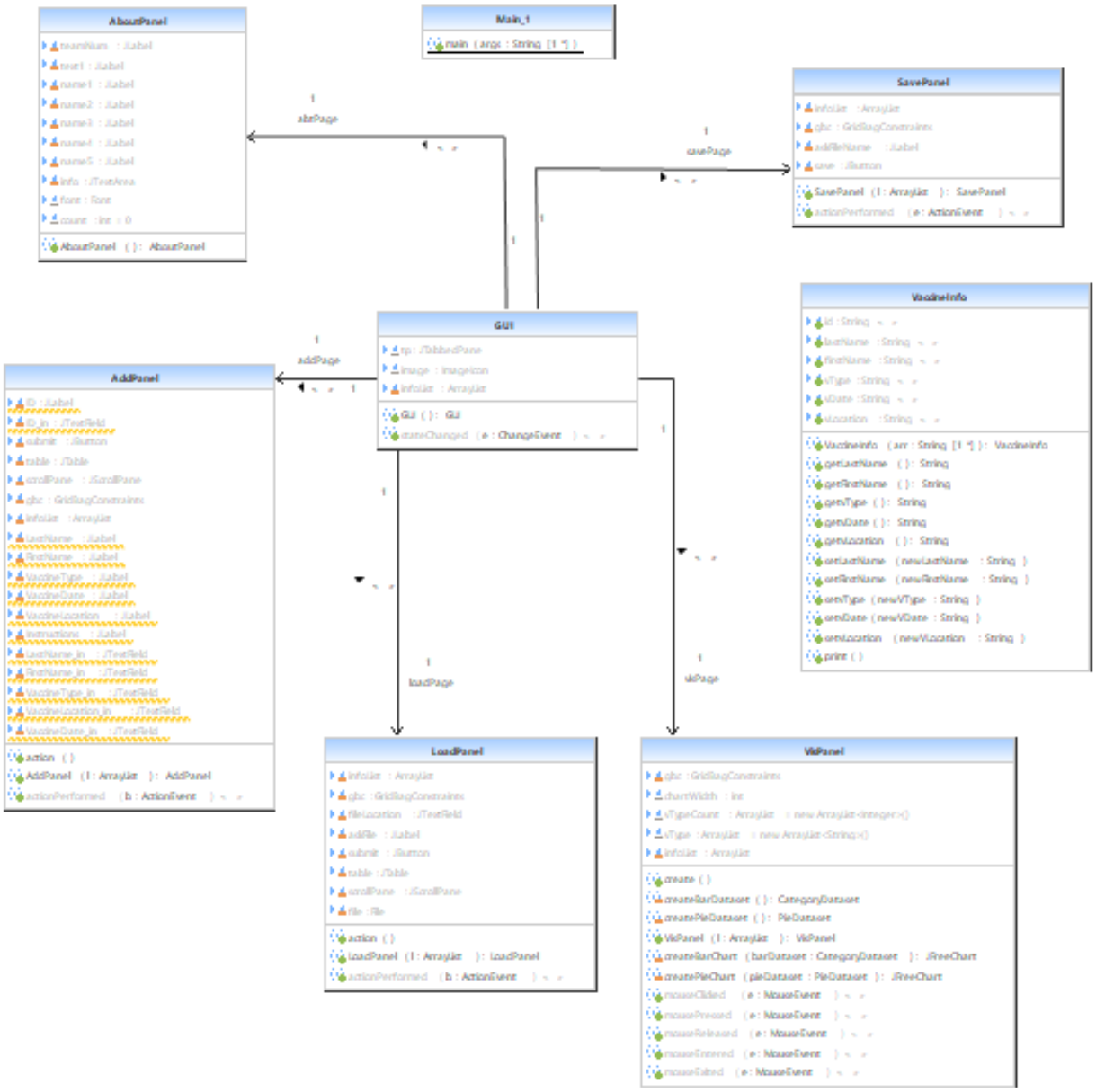
As a healthcare worker, I want to add new patient vaccination data to the table directly, so I can efficiently update patient information.

As a healthcare worker, I want to save all the vaccination data to a .csv file so that I can export and share vaccination records.

As a statistician, I want to visualize the data, so that I can see graphically where the vaccines are being administered and in what quantities.

As a user, I want to view information about the developers of this application so I can better understand how and why it was built.

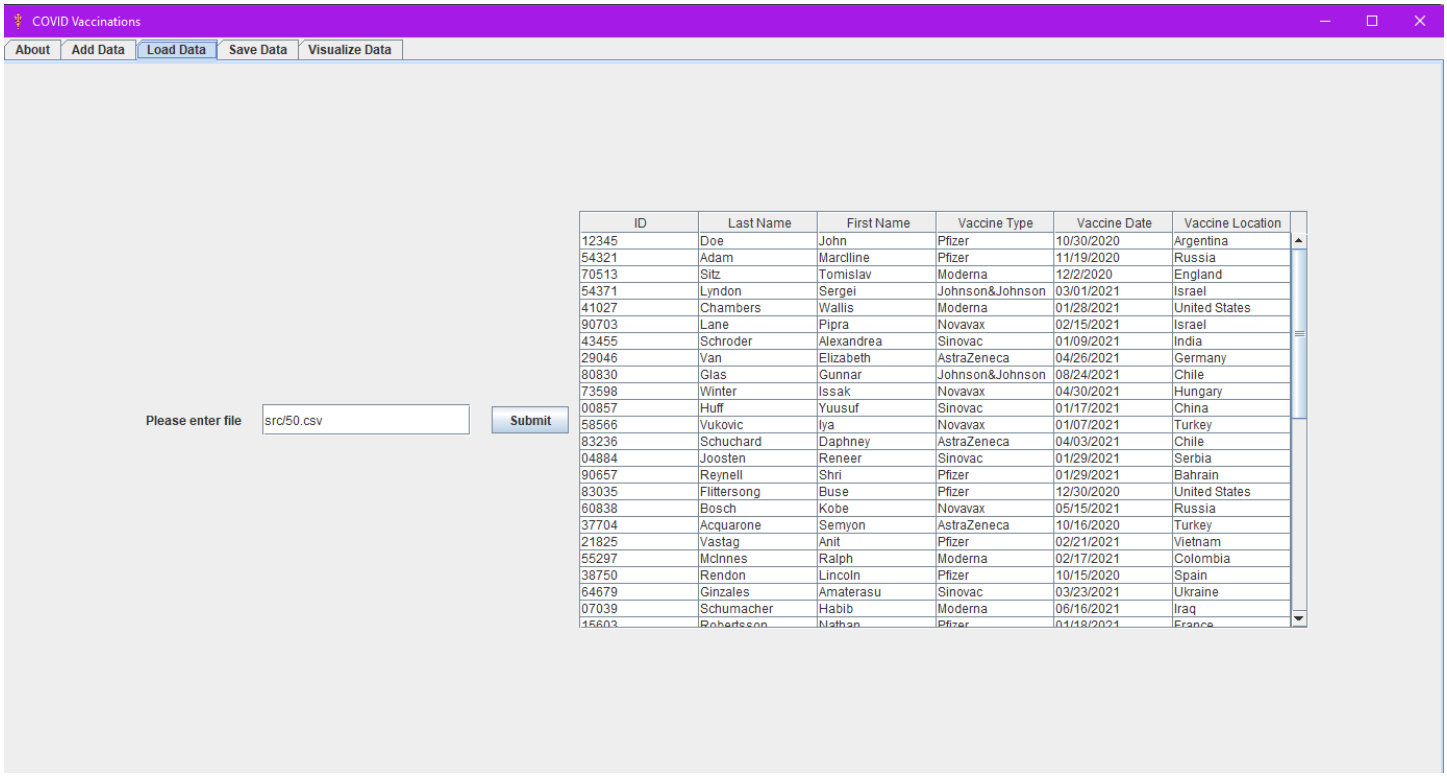
Class Diagram



Program Execution

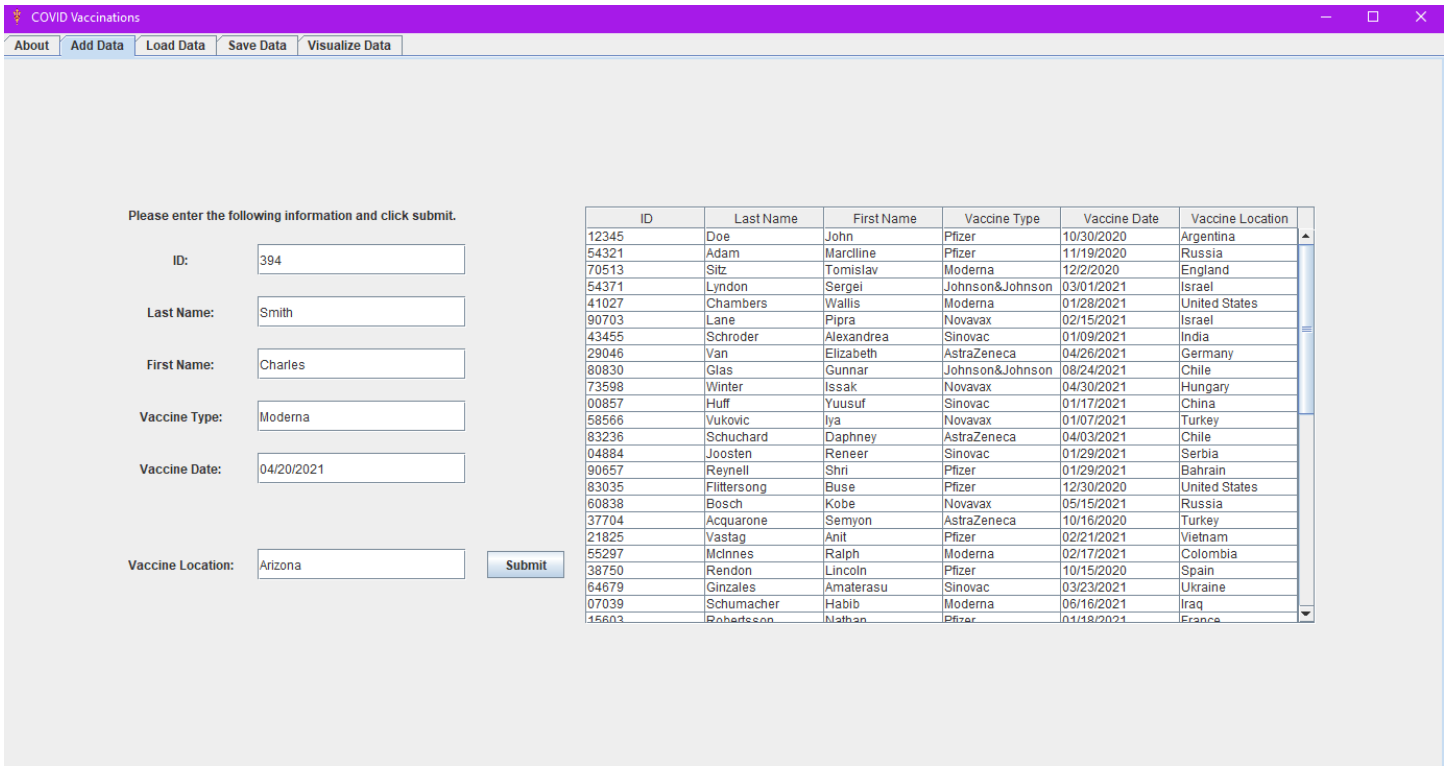
Included are screenshots of the program's execution. These screenshots demonstrate the program's functionality for each of the 5 major requirements; Load Data, Add Data, Save Data, Visualize Data, and About.

Load Data:

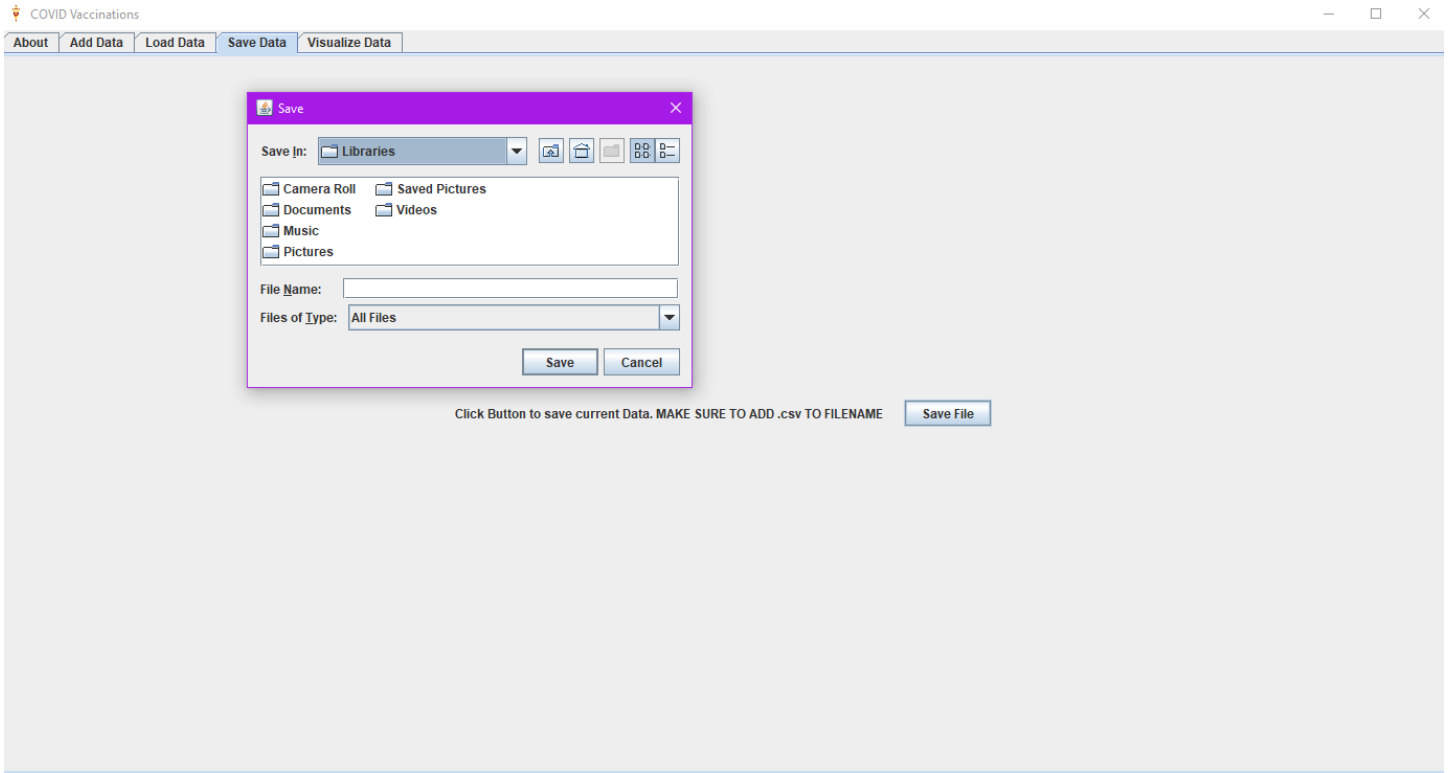


ID	Last Name	First Name	Vaccine Type	Vaccine Date	Vaccine Location
12345	Doe	John	Pfizer	10/30/2020	Argentina
54321	Adam	Marclline	Pfizer	11/19/2020	Russia
70513	Sitz	Tomislav	Moderna	12/2/2020	England
54371	Lyndon	Sergei	Johnson&Johnson	03/01/2021	Israel
41027	Chambers	Wallis	Moderna	01/28/2021	United States
90703	Lane	Pipra	Novavax	02/15/2021	Israel
43455	Schroder	Alexandrea	Sinovac	01/09/2021	India
29046	Van	Elizabeth	AstraZeneca	04/26/2021	Germany
80830	Glas	Gunnar	Johnson&Johnson	08/24/2021	Chile
73598	Winter	Issak	Novavax	04/30/2021	Hungary
00857	Huff	Yuusuf	Sinovac	01/17/2021	China
58566	Vukovic	Iya	Novavax	01/07/2021	Turkey
83236	Schuchard	Daphney	AstraZeneca	04/03/2021	Chile
04884	Joosten	Reneer	Sinovac	01/29/2021	Serbia
90657	Reynell	Shri	Pfizer	01/29/2021	Bahrain
83035	Flittersong	Buse	Pfizer	12/30/2020	United States
60838	Bosch	Kobe	Novavax	05/15/2021	Russia
37704	Acquarone	Semyon	AstraZeneca	10/16/2020	Turkey
21825	Vastag	Anit	Pfizer	02/21/2021	Vietnam
55297	McInnes	Ralph	Moderna	02/17/2021	Colombia
38750	Rendon	Lincoln	Pfizer	10/15/2020	Spain
64679	Ginzales	Amaterasu	Sinovac	03/23/2021	Ukraine
07039	Schumacher	Habib	Moderna	06/16/2021	Iraq
15603	Robertsson	Nathan	Pfizer	01/18/2021	France

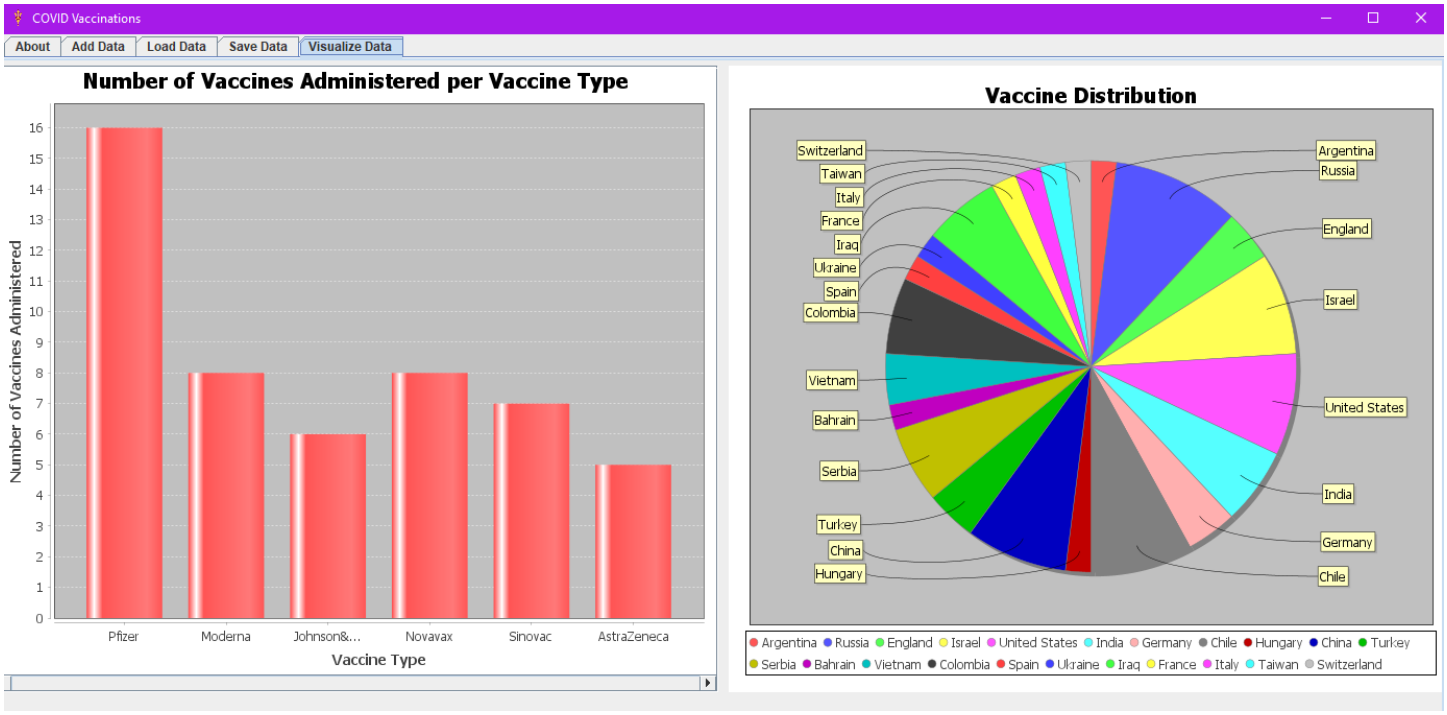
Add Data:



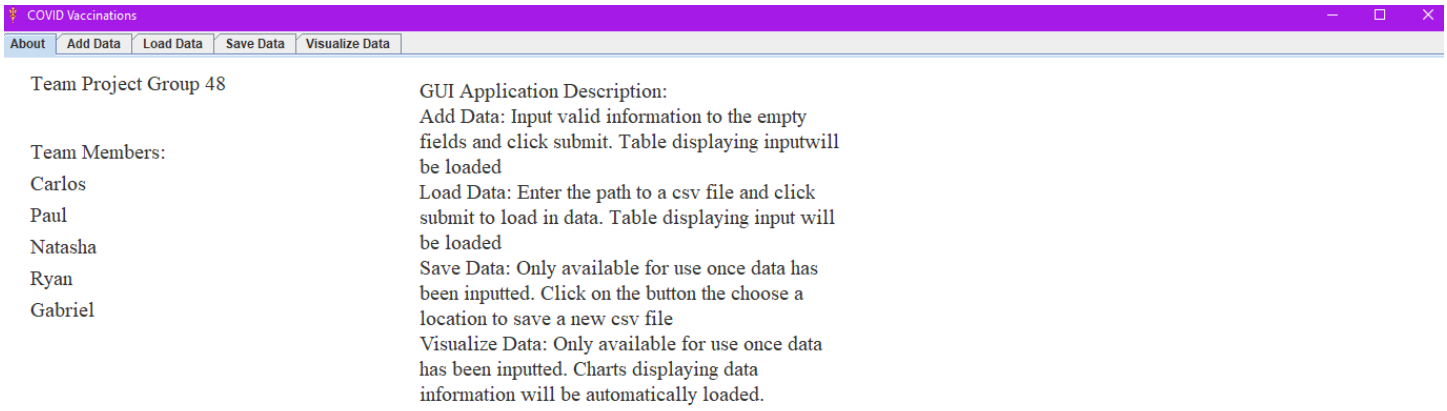
Save Data:



Visualize Data:



About:



Testing

Our team verified the program works by testing the code as it was updated. Members of the team tested code before and after pushing changes to ensure that there were no errors or loss of functionality. Each team member tested code independently on their own machines. Testing was used throughout the development process to determine if all requirements were being met for each component of the application.

The .csv files provided by the instructor were used to test the application, to ensure it could perform all necessary functions with a 10-element data set and a 50-element data set. Each .csv file was loaded into the application with LoadData, then all other features of the GUI were tested with the data.

Unit Testing:

Prior to building the GUI for components, unit testing was done on the different methods and constructors. Testing consisted of random input and output to the console to see if the results were as expected. Methods in each component had all possible exits tested. For example, saving a file has five different outcomes. “Error: Nothing to be saved”, “Successfully saved data!”, “User cancelled operation”, “An error occurred”, and “Unknown operation occurred” were each branch tested to assure proper functionality of the save method. Prior to adding the GUI interface for LoadPanel, the method to open a file, properly read data from it and store it in an ArrayList was tested. We ensured the data was properly processed for the different methods in different classes that involve generating a table, chart, save file, etc.

Component Testing:

After thoroughly testing the different methods and constructors, class and GUI testing began. Each tab was tested individually to see possible errors from beginning to use the program from there. For example, an error found in AddPanel involved adding information and the resulting JTable shifting all the GUI components. This was fixed by switching to a GridBagLayout to fix the size and position of each GUI component. Similar testing was done in LoadPanel as both deal with a JTable. VisPanel relies completely on AddPanel and LoadPanel since it is displaying the data in chart form, so this was not tested until later.

System Testing:

As the project was updated, different members made sure it ran properly on different operating systems as well as IDE environments. Instead of implementing an observer or ChangeListener to keep the GUI and ArrayList with data updated across all classes, we found a different solution. An ArrayList was instantiated in the GUI.java class which builds the entire interface. The ArrayList object was passed as an argument to each class. ArrayLists were instantiated and pointed to the main ArrayList so it can be accessed from anywhere. Testing involved adding data manually, then loading from, adding manually to ensure it was properly appended. To refresh the GUI, a changeListener was attached to the JTabbedPane that executed on switching tabs. The method calls methods from each class to update the GUI in case of any changes. Testing was straightforward, use the GUI as intended and ensure that it meets expectations. An issue with the bar chart generated was that it would not extend horizontally, causing the bars to be grouped closer as more vaccine types were added. To solve this, the bar chart is reloaded after new data additions and the horizontal size of the chart is scaled off of the number of different vaccine types.

Final Testing Phase:

Before submission, the program was turned into an executable jar file and tested by members of the group. The following checklist was completed before project submission.

Add Data:
- [x] Rejects ID with more than 5 characters (digits)
- [x] Rejects ID with letters or special characters
- [x] Asks for all fields to be filled before adding data
- [x] Fills ID less than 5 with leading zeroes
- [x] Rejects dates with letters or special characters
- [x] Replaces info if IDs match
- [x] Correctly appends data to the end
- [x] Updates automatically
- [x] Empties fields after successful data addition
- [x] Adding table does not significantly shift other components

Load Data:
- [x] Can load from specified file location
- [x] Appends to the end
- [x] Updates automatically
- [x] Replaces info if IDs match
- [x] Shows data that has been appended to end from Add Data
- [x] Rejects unspecified file locations i.e. "D:\Downloads"
- [x] Displays success message
- [x] Displays empty field message
- [x] Displays error with opening file message

Save Data:
- [x] Saves to file location
- [x] Correct format and data
- [x] Displays cancellation or error messages
- [x] Displays success message

Visualize Data:
- [x] Does not show anything if data has not been added
- [x] Gives message to add data
- [x] Data properly displayed
- [x] Bar chart can scroll once it meets a certain size
- [x] Updates automatically