

Vaccine Distribution Software Requirements Specification

Alex Anderson, Hans Prieto, Vince Qiu, Colton Trebbien, Michael Wiltshire
April 25th, 2021

Table of Contents

- 1. SRS Revision History**
- 2. The Concept of Operations (ConOps)**
 - 2.1. Current System or Situation**
 - 2.2. Justification for a New System**
 - 2.3. Operational Features of the Proposed System**
 - 2.4. User Classes**
 - 2.5. Modes of Operation**
 - 2.6. Operational Scenarios**
- 3. Specific Requirements**
 - 3.1. External Interfaces (Inputs and Outputs)**
 - 3.2. Functions**
 - 3.3. Performance Requirements**
 - 3.4. Software System Attributes**
- 4. References**

*Note that the term vaccinee used through the document refers to a person who wishes to be, or will be, vaccinated.

1. SRS Revision History

Date	Author	Description
4-9-2021	All	Created the initial software requirement specification
4-9-2021	All	Completed sections 2.1
4-9-2021	Cole	Added sections 2.2 & 2.3
4-11-2021	Cole	Added sections 2.4 & 2.5
4-11-2021	Alex	Added diagrams to section 2.4
4-11-2021	Alex	Completed section 3
4-22-2021	Alex	Update concept of operations and requirements sections
4-25-2021	Alex	Update sections 3.2, 3.3, and 3.4
4-25-2021	Colton	Update section 2.4 and 2.5

2. The Concept of Operations (ConOps)

2.1. Current System or Situation

The world has entered into a national pandemic, which has resulted in a race to create a vaccine. That race was finished by several manufacturers. Now, we are faced with the challenge to distribute these vaccines to the people.

2.2. Justification for a New System

A system needs to be created to ‘fairly’ distribute the vaccine to all people. In this sense, ‘fairly’ means the people who need it most should receive it first.

2.3. Operational Features of the Proposed System

The system has an easy-to-use web interface for users to navigate. Internally, the system prioritizes high-risk users but allows for a more fluid distribution of available vaccines to all registered users. Many people have had struggles with vaccine sign-ups, and many areas use cobbled together systems with difficult user interfaces^[1].

Our system streamlines vaccine registration into an easy to follow process. Users simply have to register, indicating that they want to be vaccinated soon. The registration GUI is always available and is easy to use, as to not discourage or prevent people from registering. The system assigns them a score and will prioritize the people most in need first. The score design

feature is important as high-risk people must be vaccinated first.

2.4. User Classes

Vaccinee:

Description: A vaccinee is the main user of our system. They can come from a wide variety of people and they can have many different levels of skill. The main objective for the vaccinee is to become registered for a vaccination appointment.

Mode of operation: The vaccinee will go to the registration website, where they will be asked to fill out information about themselves. After submitting their information the system will notify them their registration was completed and they will be contacted when an appointment becomes available.

Dosage Administrators:

Brief description: The dosage administrators are users who will interact with the appointment assignment module to receive contact information about the vaccines. The types of people who would want to be dosage administrators would include people who run or work for vaccination clinics who have knowledge about the availability of the vaccinee.

Mode of Operation: The Administrator will enter in a command to add data about the number and availability of vaccines at their current distribution location. They interact with the appointment assignment module and receive a spreadsheet/calling list with registered vaccinees contact information.

Communications technician:

Brief description: The communications technician is a more detached part of our system. They receive information from the administrator and inform the vaccinee of their upcoming vaccination date.

Mode of Operation: They can be given a list of vaccinees and their contact information from the administrator. They can then call to inform the vaccinee of their appointment.

2.5. Operational Scenarios (Also Known as “Use Cases”)

Use Case: Register for a vaccination.

Brief description: This use case describes how a vaccinee would register for a vaccination

appointment through the website.

Actors: A Vaccinee.

Preconditions:

1. The Vaccinee wishes to schedule a vaccination appointment in the near future.
2. The User has access to the website via a computer with internet access.
3. They have adequate knowledge of their personal health.
4. They are able to be contacted using a phone and email.

Steps to Complete the Task:

1. The Vaccinee will navigate to the website's address.
2. They are shown a friendly webpage with a big register now button.
 - a. The user would then click the button and the website will redirect them to a registration page.
 - b. The user fills out the required fields such as their height, age, email and other medical factors.
3. The user clicks submit and the website will:
 - a. Show that the registration process was complemented and confirmed.
 - b. Show that an error had occurred, either by not filling in all the required fields or that their email and phone number had already been registered into the database.
4. Once their appointment date arrives, a technician will email or call them about their appointment.

Postconditions:

The user is registered. They are now able to be contacted and informed of an appointment in the near future.

Use Case: Appointment assignment

Brief description: This use case describes how an administrator would assign appointments.

Actors: Dosage Administrator

Preconditions:

1. The Administer has recently received a shipment of vaccines.
2. The User has access to the system, most importantly the appointment assignment portion.
3. The User has space on their computer to generate a CSV file.

Steps to Complete the Task:

1. The User will navigate to and open up either `appointment_assignment.py` or `appointment_assignment.bat` (double click this file to open/run it)
2. The User then enters in the number of vaccines/ the number of Vaccinees the administrator would like to schedule a vaccination appointment with.
3. A new file will be created, which can be found in the same file directory.
 - a. By default the file will be named “`call_list.csv`”.
 - b. The administrator can open this file up and use the contact information to inform Vaccinees about an appointment or give this file to another technician.

Postconditions:

The Administrator now has a list of Vaccinee contact information which they can use to schedule appointments with Vaccinees.

3. Specific Requirements

Our system meets the following requirements:

- Has an interface and web server for users to register for vaccines.
- Has the functionality for internal dosage administration workers to be able to create spreadsheets or calling lists of users who need to be contacted for their vaccination appointments. This process is known as the appointment assignment process. Note that our system does not contact users or confirm appointments and relies on the internal service workers using the spreadsheet to reach out to vaccinees.
- Prioritizes users who are at higher risk by assigning appointments first.
- A database that persists user information.
- Secure storage of user information. All sensitive data is stripped once the score is generated.
- Data sanitization and protection from common attacks such as SQL injections. A user putting odd characters, like double quotes, in the different registration boxes shouldn't create an error.
- A comprehensive test suite that ensures that all features of the system operate as intended which also clearly demonstrates the intended functionality of all code that it tests.
- Documentation that clearly describes how to install and use the system unambiguously.
- 95% of users can register for a vaccination appointment in less than ten minutes.
- An intuitive and minimal user interface. There is exactly one way to interact with the registration web interface.
- Users are never blocked from registering. No user is told to check back later when

more inventory is available.

- A script to automatically insert sample data into the program to make integration testing and product demonstration easier.

3.1. External Interfaces (Inputs and Outputs)

- Input: Web registration interface.
 - The web registration interface is how users interact with the system. They register for an appointment via this module and they are entered into the queue.
 - Inputs are collected from a HTTP request made by the registration web interface sent to the web server. Users are told that they are registered via a response from the registration web server to the web interface. If there was a failure, users are alerted that their registration was unsuccessful.
 - Inputs contain the following information, based on COVID risk factors^[2]: A person's age, a person's weight and height (for BMI), their full name, their email address, their phone number, if they have heart problems, diabetes, lung problems, liver problems, a compromised immune system, a previous positive COVID-19 test, recently close contact with someone who has COVID-19, and if they're experiencing COVID-19 symptoms.
 - The person's age, weight, height in feet, and their height in inches are sent as integers. Their email address, name, and phone number are sent and stored as strings. The yes/no questions are sent as booleans. Their age, weight, height, and the health questions are not stored in the database, only the score which is calculated using this information is stored.
 - The data will be formatted as an HTML form and transferred in the body of an HTTP request.
- Output: vaccinee appointment spreadsheet.
 - The vaccinee appointment spreadsheet is an exported spreadsheet, created from the internal appointment assignment module, that lists people who are going to be vaccinated with the current batch of vaccines. The people on this list need to be called and have their appointments confirmed.
 - Outputs are in the CSV format generated from a query run on the database.
 - The format will be a well-formed CSV spreadsheet, consisting of names, phone numbers, email addresses, and scores (in order to verify their score during their appointment).
 - Each row in the CSV file must have a name, phone number, email address, and a score. The first three of which are string types and the score is a floating point number.
 - The CSV format consists of values separated by commas, one for each cell, with a row on each line.

3.2. Functions

The only part of our system that people with malicious intentions can interact with is the vaccine

registration web interface. The backend of our system is simply not accessible by outside entities. However, it is vital that the data we get from our front end is validated and sanitized. The following must be done when receiving an HTTP request in terms of data validation:

- Confirm that all strings are escaped. If SQL queries can be executed by potentially unknown users, users could retrieve information that they should not have access to or delete massive amounts of information instantaneously. If a system is unprotected from these attacks, using quotes (“”) will cause an error.
- Confirm that the HTML form body is well formed. If any data is missing or the body is malformed, which would only happen by ill-intent, they will be told their registration is unsuccessful.

In order to test these features, there are sample data that has the potential of causing SQL injections and tests which attempt to send the web server bad requests.

In terms of functionality, the following features are provided by the modules we have built:

- Calculate score, provided by the score module.
- Assign users with appointments and create CSV spreadsheets, provided by the appointment assignment module.
- Serve HTML webpage upon request, provided by the vaccinee registration web server module.
- Accept registration requests, provided by the vaccinee registration web server module.
- Send data to the server, provided by the vaccinee registration interface.
- Create database, provided by a combination of the database module.

3.3. Performance Requirements

Requests to register for a vaccination are processed by the web server, within 1 second on standard consumer hardware (~3.0GHz Quad Core CPU). The process of actually storing a user in the database can be done approximately 50 times a second (number obtained by using 100,000 insertions).

Requests to generate a CSV spreadsheet process quicker than 1 minute for 10,000 people. The time it takes to generate a file scales linearly, that is, if generating a CSV file for 10,000 people takes one minute, generating a spreadsheet for 100,000 people takes no more than ten minutes.

3.4. Software System Attributes

- Ease of use is the most important attribute of this system. A system that fails to be easy to use will be ineffective due to users preferring other resources.
 - Users only need to register once and don't have to frequently check back. They also do not need to register on multiple different vaccine distribution websites.
- Reliability is paramount. If the system is not able to return users in the correct order,

peoples' lives could be at stake.

- SQL tends to be extremely reliable and can order the data as quickly as possible without making any errors.
 - The journaling built into SQL is difficult to test. However, it supposedly should ensure that no data gets corrupted even during an unexpected shutdown. We found that even when tasks that are actively using the database are forcibly killed, no data is lost or corrupted.
- Performance is important, but not a top priority. Generating a spreadsheet should be an almost instantaneous process, but if it takes a reasonable amount of time, that is acceptable.
 - This impacts internal service workers the most. However, even if a spreadsheet of 100,000 vaccinees took around an hour to generate, it would only take 40 hours of waiting on the system to vaccinate the entire population of Oregon (at about 4 million people).
 - A spreadsheet of 100,000 vaccinees took our system a little bit less than one minute.
 - It should be noted that inserting this sample data (generating random data and writing it to our database) took considerably longer than generating call lists. To insert 100,000 records took around half an hour. However, no resource on the machine was being fully utilized, so perhaps there is an issue with that particular testing script.

4. References

^[1]M. Harding McGill and K. Hart, "Why it's so hard to sign up for vaccinations online", *Axios*, 2021. [Online]. Available: <https://www.axios.com/coronavirus-vaccine-appointments-online-system-bad-f626abd6-3811-4db3-9b02-4da0479fa53d.html>. [Accessed: 25- Apr- 2021].

^[2]"COVID-19: Who's at higher risk of serious symptoms?", *Mayo Clinic*, 2021. [Online]. Available: <https://www.mayoclinic.org/diseases-conditions/coronavirus/in-depth/coronavirus-who-is-at-risk/art-20483301>. [Accessed: 12- Apr- 2021].

Vaccine Distribution System

Software Design Specification

Alex Anderson, Hans Prieto, Vince Qiu, Colton Trebbien, Michael Wiltshire
April 25th, 2021

Table of Contents

- 1. SDS Revision History**
- 2. System Overview**
- 3. Software Architecture**
- 4. Software Modules**
 - 4.1. Vaccinee Registration Interface**
 - 4.2. Vaccinee Registration Web Server**
 - 4.3. Health Risk Scoring Algorithm**
 - 4.4. Vaccinee Database Table**
 - 4.5. Appointment Assignment Module**
- 5. Dynamic Models of Operational Scenarios (Use Cases)**
- 6. References**
- 7. Acknowledgements**

*Note that the term vaccinee used through the document refers to a person who wishes to be, or will be, vaccinated.

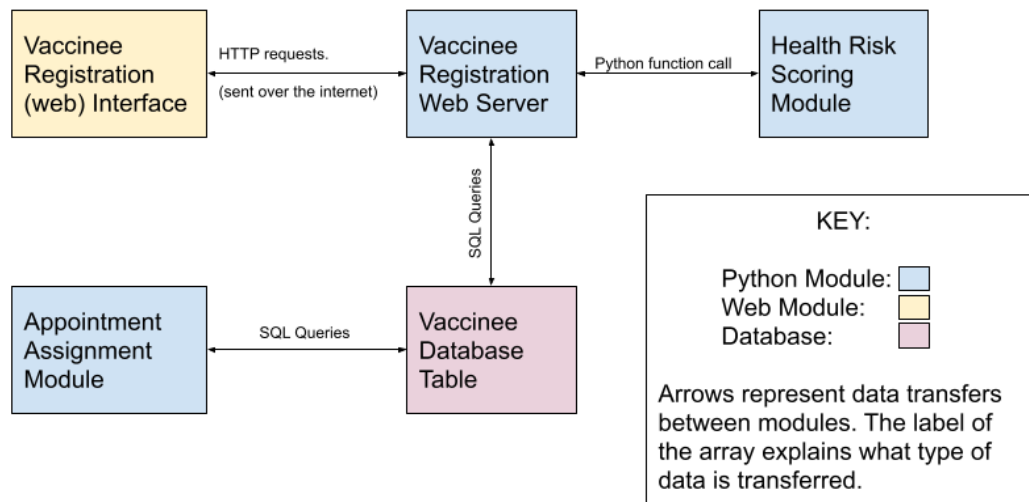
1. SDS Revision History

<u>Date</u>	<u>Author</u>	<u>Description</u>
4-8-21	All	Create initial software design specification.
4-9-21	MW	Implement initial software module descriptions.
4-9-21	HP	Describe the primary function of the registration module.
4-9-21	AA	Complete Software Architecture section.
4-11-21	AA	Complete Software Modules section.
4-22-21	AA	Add diagrams for sections 3 and 4.
4-24-21	AA	Update sections to reflect what we actually built.

2. System Overview

The system is organized in a way that makes it externally easy to use and internally effective at distributing all available vaccines. Vaccinees can register themselves and are then automatically placed in a queue to receive a vaccine. An internal worker can then input a number of available vaccines and generate a spreadsheet-- or calling list-- of people to reach out to.

3. Software Architecture



The required components include a vaccinee registration interface, vaccinee registration web server, a health risk scoring algorithm, a vaccinee database table, and a vaccine assigner module.

The vaccinee registration interface interacts directly with users and runs on the users' machines (client-side). The interface connects to the vaccinee registration web server by sending packets through the internet. This module is built in HTML and CSS, which are the foundation of every web interface.

The vaccinee registration web server provides the persistent functionality for the vaccinee registration interface and is not run on the end user's hardware, but on a central server (server-side). The web server receives and responds to requests from the vaccinee registration interface. The web server stores user profiles provided from the vaccinee database table and calculates scores with the health scoring module. This module is built in Flask, a Python framework.

The health risk scoring module generates a number that corresponds to a user's risk level based on different information by their health. The vaccinee registration web server calls on this functionality to generate scores before the data is persisted in the vaccinee database table.

The vaccinee database table persists data about users, in particular, their name, phone number, email address, a score of their health for prioritization, the date that they first registered, and if they've been vaccinated yet. The database receives queries from the vaccine registration web server and the vaccine assigner module and responds by creating, updating, or retrieving records from its storage. This module is built in SQLite, an SQL engine that utilizes files rather than separate machines to store data.

The appointment assignment module pairs up users with incoming vaccines. It queries the vaccinee database table for users who have the highest 'score,' that is, the users that should be vaccinated first, and creates a list of people who need to be reached out to to be vaccinated. It is

run separately as vaccines are received by the system. This module will be built in Python using Flask's SQL alchemy module to access information in the database.

4. Software Modules

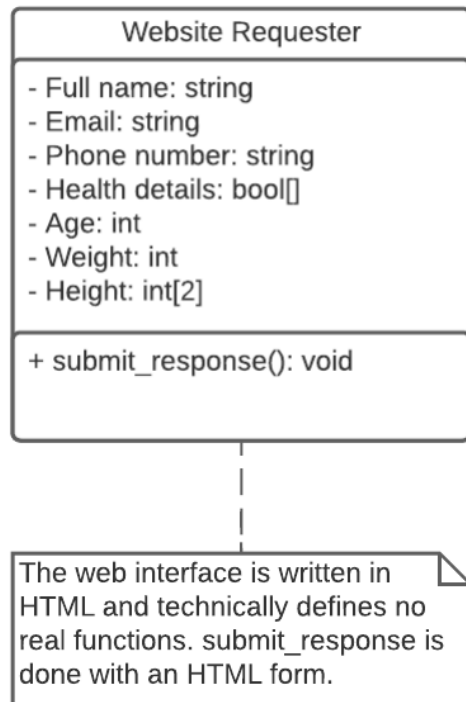
Each module is described in the following format:

- A. A description and the main function of the module.
- B. Why the module is important.
- C. A static model which describes the interface to other components.
- D. A dynamic model.
- E. A design rationale and alternative designs.
- F. Technical implementation decisions.

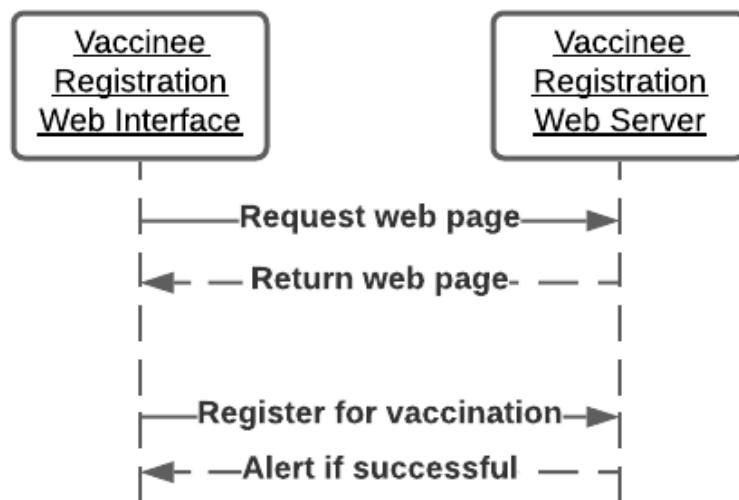
Note that in most cases, the classes are the modules themselves. They may not be defined as a class in the Python code (with the exception of the Vaccinee database class), but they exhibit object oriented behavior. That is, they hold some internal state and have some externally callable methods.

4.1. Vaccinee Registration Interface

- A. The vaccine registration interface's primary function is asking for user input, and sending the information to the vaccinee registration web server. This module is used as a way to collect information about an individual's contact information and health attributes.
- B. This is the module that users interact with. Without a user interface, very few users have the capability of interacting with our system, and even fewer want to.
- C.



D.

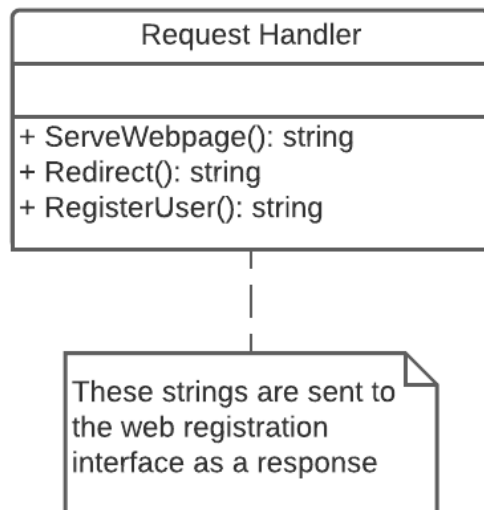


- E. This module was implemented as a website rather than a downloadable graphical user interface in order to be more accessible to the general public. People can navigate and use this website on a phone, laptop, desktop, or a variety of other devices.
- F. We decided to use HTML and CSS for this module as they are the building blocks of every website. We decided to not go with a template builder or framework, like bootstrap, as it would give us less control. However, these frameworks could make things a lot

easier. JavaScript was not required for any of the functionality we needed.

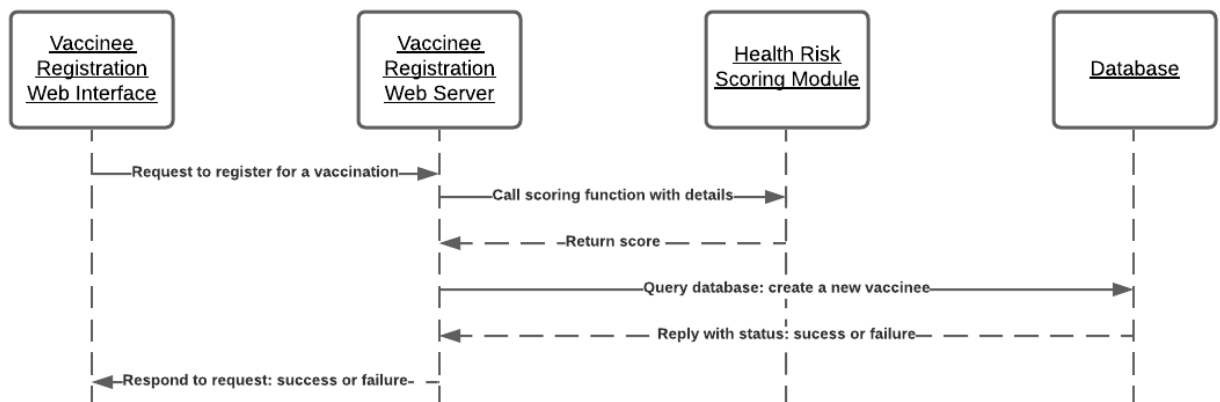
4.2. Vaccinee Registration Web Server

- A. The vaccinee registration web server serves web pages and dynamic content via HTTP requests and interacts directly with the web interface, scoring module, and database.
- B. This allows interaction between the vaccinee registration web interface and the database. Without this, users couldn't register. The requests to register from the registration web interface would be ignored.
- C.



Note that the functions generate their own input by inspecting the incoming response, which is built into Flask rather than the function itself. In some sense, each of the functions described above take a http request as an implicit parameter.

D.

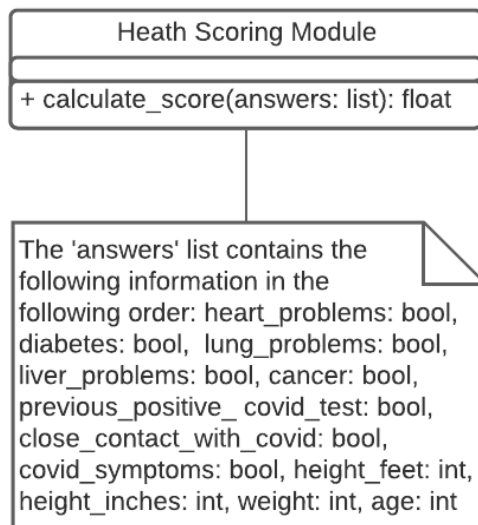


- E. We decided to go with a single web server module rather than several web servers for different interfaces due to time constraints. We decided to offload the other web server modules proposed in our project proposal to the appointment assignment module which is command-line based.

- F. We decided to use Flask, a Python framework, to implement this module. Flask is both easy to use and easy to use safely. As compared to PHP, there are far fewer exploits. Additionally, the use of Flask allows us to better modularize our code. Furthermore, all of our team members are comfortable with Python.

4.3. Health Risk Scoring Algorithm

- A. The health risk scoring algorithm takes in some information about a vaccinee's health and outputs a score. It is intended to be used as a black box and has the additional benefit of not storing sensitive data.
- B. In order to prioritize users who need the vaccine the most, their health information is parsed through this module and turned into an integer score. Those with conditions that put them at risk for COVID-19^[1] are put into calling lists earlier than users who are at less risk (so long as they are registered).
- C.



- D.



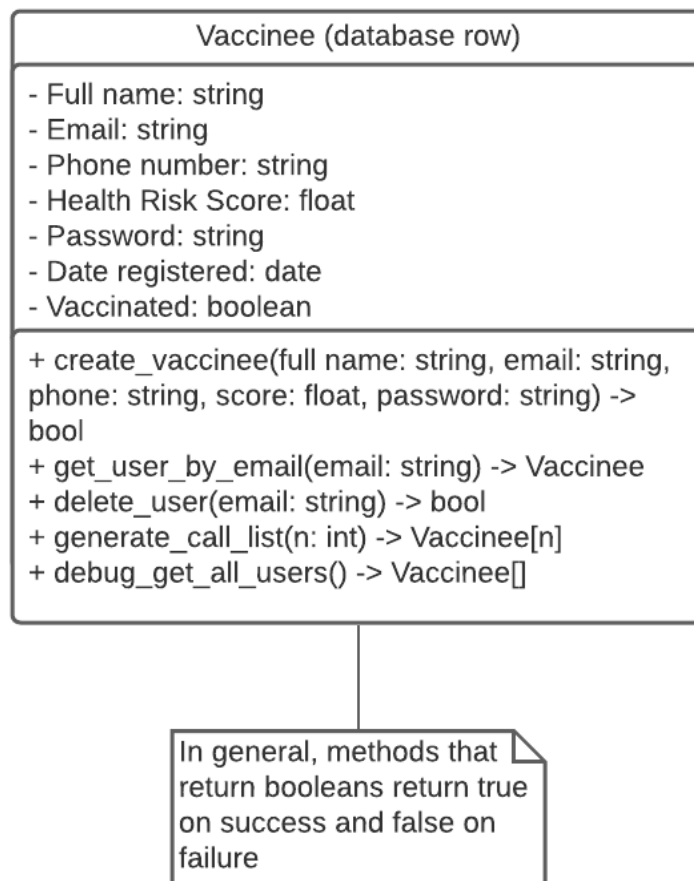
- E. The module could have very well been combined with the web server, and in some sense,

it is contained by the web server. We choose to move the code outside of the web server to make it possible to adjust the scoring method without working with web server code.

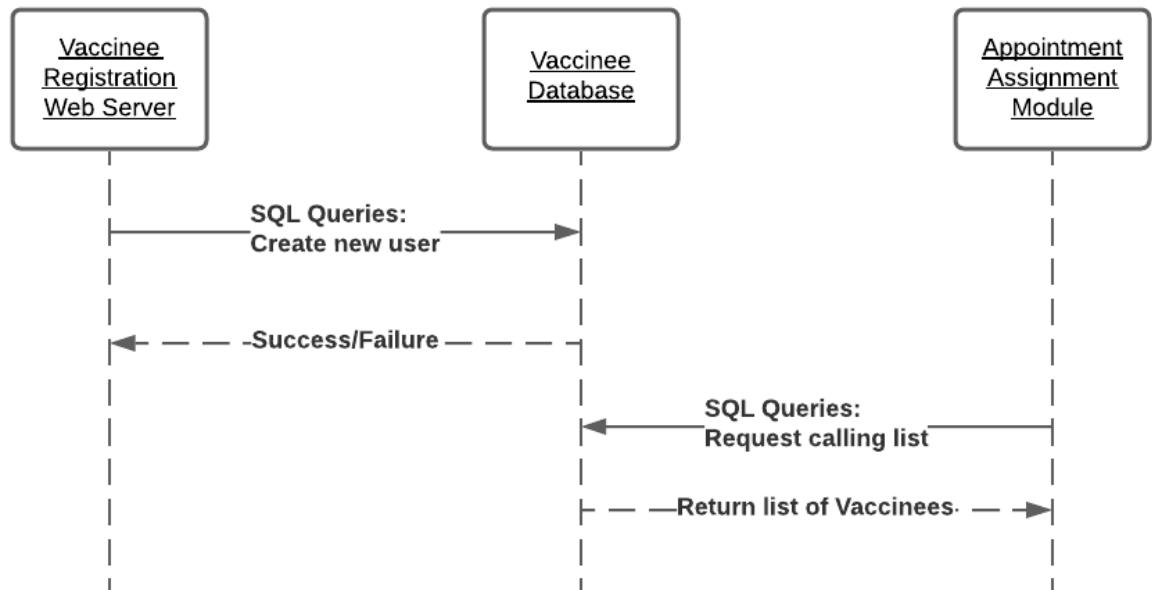
F. For this module, we choose to implement it in Python, as it's easy to integrate with flask.

4.4. Vaccinee Database Table

- A. The vaccinee database table stores information about users.
- B. Without a database, our system would have to manually search through the data line by line or store everything in memory. A database is simply required for a project of this scale to be practical and efficient.
- C.



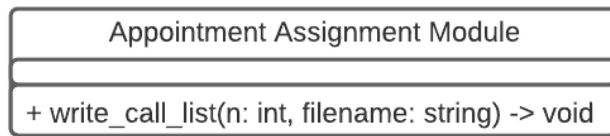
D.



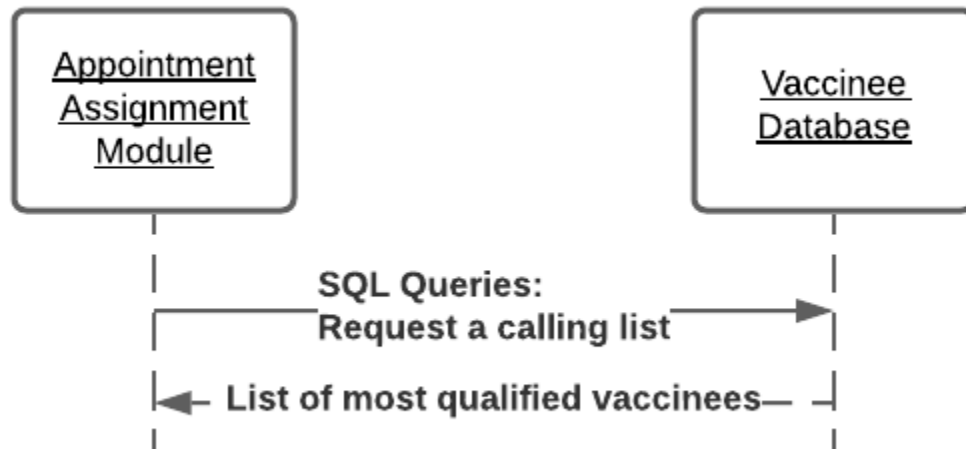
- E. We decided to separate out the database from any other module as it allowed us to build functionality involving the database much more quickly. This was a perfect use case for a database, as all records were one row of data.
- F. SQLite was a hard decision to make. Other database engines scale better, while SQLite stores its data in a flat-file. However, for the scope of this project, it is a perfect tool. It allows us to build easy mock-ups and is changeable in place with any other SQL drivers, like PostgreSQL. It may be less scalable than other SQL engines, but if that scalability is needed it is far easier to switch to them than it would be if we were using a flat-file.

4.5. Appointment Assignment Module

- A. The appointment assignment module pairs up vaccinees and vaccines. As vaccines come in, an internal service worker will type in the number of vaccines they wish to administer into a terminal window and it will generate a spreadsheet listing what users are next on the list to be vaccinated.
- B. This module allows people to schedule their appointments to be vaccinated. The information-- a calling list in spreadsheet form-- that this generates should be sent to a call center or communication technician to reach out and confirm each person's appointment.
- C.



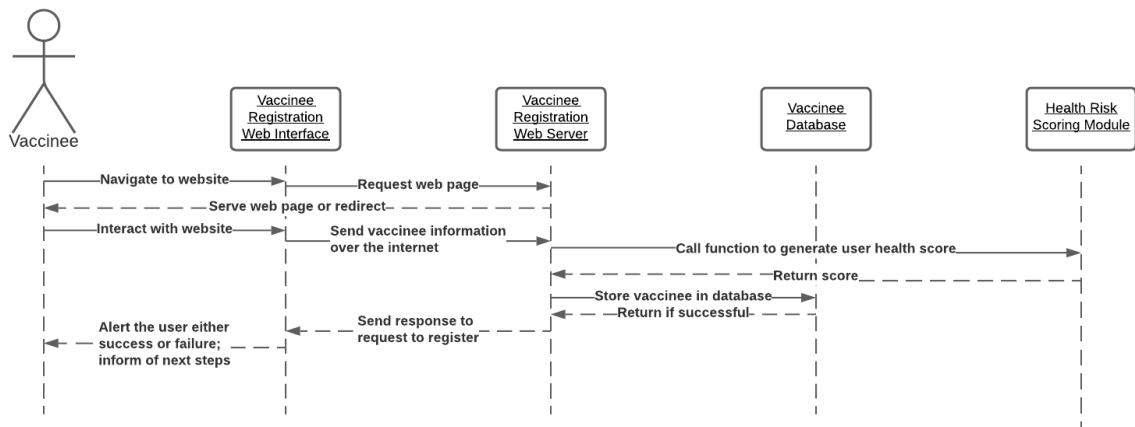
D.



- E. In the proposal, the overall design for this part was much more complex and full-fledged. No one would need to open a terminal and appointments could be automatically assigned through a web interface. However, this was too ambitious for the time constraints. It would massively increase the complexity of the system and create far more surface area for errors and security exploits to occur. The original system had a module that would automatically match information from three databases: a vaccine database, a vaccinee database, and a building availability database. These were cut, and their functionality was reduced into what is now the reduced version of their functionality.
- F. By the time that this module was fully realized, the architecture of the existing system strongly influenced it to be written in Python.

5. Dynamic Models of Operational Scenarios (Use Cases)

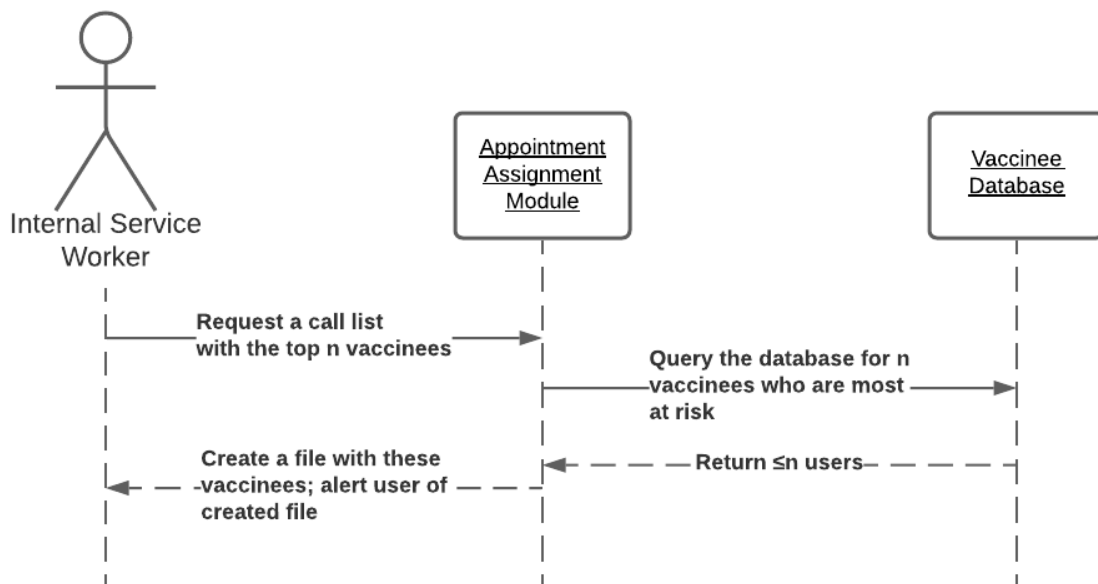
The following diagram represents the sequence which takes place when a user registers for a vaccine, which is the most common interaction with our system and may happen several thousand times each day.



When the user registers, a chain of events is set in motion. These events are performed by the vaccination registration web server. It receives a request, and with the information in that request, calculates a score. It then stores the data in the database and replies back. The user then sees a webpage indicating if the registration was a success or a failure.

The end result of this interaction is the user being stored in the vaccinee database or a user being told that their registration failed.

Once this interaction takes place multiple times and vaccines are received by the internal service worker(s), the following interaction takes place.



An internal service worker is alerted of a vaccination shipment from an outside source. They would then open up the appointment assignment module and type in the number of vaccines that they received. A file would be generated of the most at risk users, sorted based on score (for those who have the same score, the ones who register first will be earlier on the call list). The file generated is in a CSV format and contains the user's name, phone number, email address, and score. The first three allow easy contact of the user, while the score is there to verify the information they provided during their appointment, that is, to prevent fraud.

Note that the system does not contact vaccinees. That job is meant to be given to a call center or the internal service workers themselves.

6. References

^[1]"COVID-19: Who's at higher risk of serious symptoms?", *Mayo Clinic*, 2021. [Online]. Available: <https://www.mayoclinic.org/diseases-conditions/coronavirus/in-depth/coronavirus-who-is-at-risk/art-20483301>. [Accessed: 12- Apr- 2021].

Vaccine Distribution Project Plan

Alex Anderson, Hans Prieto, Vince Qiu, Colton Trebbien, Michael Wiltshire
April 25th, 2021

Table of Contents

- 1. Revision History**
- 2. Timeline and Milestones**
 - 2.1. Module Completion Dates and Minutes**
- 3. Management Plan**
- 4. Reporting**
- 5. Discussions and Builds**
- 6. Ground Rules**
- 7. Reflection**
- 8. References**

1. Revision History

<u>Date</u>	<u>Author</u>	<u>Description</u>
4/9/21	All	Created initial project plan
4/11/21	AA	Completed project plan
4/23/21	AA	Update to reflect division of work; update section 6
4/25/21	AA	Update Sections 3, 4, and 5; Add section 2.1
4/26/21	AA	Add reflection section

2. Timeline and Milestones

Originally, our project plan was as follows:

Vaccine Project Plan Timeline		Last Revised: April 9th, 2021							
		4/13	4/14	4/15	4/16	4/18	4/21	4/23	4/25
Environment set up		AA							
Git set up		AA							
Study Flask			HP/MW						
Study HTML/CSS			VQ						
Flask skeleton code			CT						
Complete build #1			All						
Database migrations				AA					
Being documenting installation process				AA					
Create skeleton webpage				VQ					
Create skeleton tests				MW					
Create skeleton code for algorithm				HP					
Complete build #2					All				
Complete crude registration interface						VQ			
Complete registration endpoint						CT			
Complete first tests						MW			
Create sample test data						MW			
Complete crude scoring						HP			
Connect scoring and registration web server						HP/CT			
Adjust project timeline						AA			
Touch up registration interface (CSS)							VQ		
Connect registration interface to registration server							VQ/AA/CT		
Finalize comprehensive tests							MW		
Finalize installation guide							AA		
Complete build #3									All
Key (FL = First_name Last_name):									
Not started:	FL								
In progress:	FL								
Done:	FL								
Late:	FL								
Late obstruction:	FL								

Two milestones were unrealized at the start of the project. Namely, creating the appointment assignment module and completing the appointment assignment module. The final project plan with those two additional milestones and the real completion dates is as follows:

Vaccine Project Plan Timeline		Last Revised: April 25th, 2021							
		4/13	4/14	4/15	4/16	4/18	4/21	4/23	4/25
Environment set up		AA							
Git set up		AA							
Study Flask			HP/MW						
Study HTML/CSS			VQ						
Flask skeleton code			CT						
Complete build #1			All						
Database migrations				AA					
Being documenting installation process				AA					
Create skeleton webpage				VQ	CT				
Create skeleton tests				MW					
Create skeleton code for algorithm				HP					
Complete build #2					All				
Complete crude registration interface						VQ			
Complete registration endpoint						CT	CT		
Complete first tests						MW			
Begin Appointment assignment module						AA			
Create sample test data						MW		AA	
Complete crude scoring						HP			
Connect scoring and registration web server						HP/CT	CT		
Adjust project timeline						AA	AA		
Touch up registration interface (CSS)							VQ		
Complete Appointment Assignment Module							AA/CT		
Connect registration interface to registration server							VQ/AA/CT		
Finalize comprehensive tests							MW		MW
Finalize installation guide							AA		AA
Complete build #3									All
Key (FL = First_name Last_name):									
Not started:		FL							
In progress:		FL							
Done:		FL							
Late:		FL							
Responsibility transfered:		FL							

2.1. Module Completion Dates and Minutes

Vaccinee Registration Web Interface

The vaccinee registration web interface was created as skeleton code on 4/18 by Colton (approximately 10 minutes). A stylistic version was then created on 4/21 by Vince (approximately 2 hours). Finally, that version was linked together with the web server by Colton on 4/23 (approximately 3 hours).

A total of around 5 hours and 10 minutes were spent on the vaccinee web registration interface, not including time spent reviewing code and testing.

Vaccinee Registration Web Server

The vaccinee registration web server was created as skeleton code on 4/14 by Colton (approximately 10 minutes). The vaccinee registration web server was then linked to the database on 4/16 by Alex (approximately 1 hour). The vaccinee registration web server was then linked to the scoring module on 4/23 by Colton (approximately 1 hour). The vaccinee registration web server was completed on 4/23 by Colton (approximately 4 hours).

A total of around 5 hours and 10 minutes were spent on the vaccinee registration web server, not including time spent reviewing code and testing.

Health Risk Scoring Module

The crude version of the health risk scoring module was completed on 4/16 by Hans (approximately 1 hour and 30 minutes). The scoring module was then redone after doing more research on COVID risk factors on 4/21 by Hans (approximately 1 hour and 30 minutes).

A total of around 3 hours were spent on the health risk scoring module, not including time spent reviewing code and testing.

Vaccinee Database

The first iteration of the vaccinee database was completed on 4/16 by Alex (approximately 2 hours). The database was then reworked during build 2 on 4/18 by Alex (approximately 2 hours). The database was reworked once again while working on the appointment assignment module on 4/23 by Alex (approximately 1 hour).

A total of around 5 hours were spent on the database module, not including time spent reviewing code and testing.

Appointment Assignment Module

The appointment assignment module was started on 4/21 by Alex (approximately 1 hour). It was then updated to pull data from the database on 4/23 by Alex (approximately 1 hour). It was completed on 4/23 by Colton (approximately 2 hours).

A total of around 4 hours were spent on the appointment assignment module, not including time spent reviewing code and testing.

Testing

Testing the database was completed on 4/21 by Mihcal (approximately 2 hours).

Miscellaneous Tasks

The script to insert sample data was done on 4/25 by Alex (approximately 1 hour). Git environment setup was done on 4/13 by Alex (approximately 30 minutes). Studying was done on 4/14 by Michael, Hans, and Vince (approximately 1 hour each). Adjusting the project plan was done throughout the project by Alex (approximately 1 and a half hours).

Miscellaneous tasks took a total of 4 hours.

3. Management Plan

Our group has five people and each person in response for different parts of the project.

- Alex was the lead engineer and database administrator. He was responsible for handling any issues regarding communication between teammates, communication between modules, and the vaccinee database module. He is also responsible for adjusting the project plan, creating the installation documentation, and creating most of the project documentation. He completed all the tasks under the miscellaneous category, including creating a script to insert sample data and testing the performance of different components.
- Colton was responsible for the majority of the Python code, especially the vaccinee registration web server, which he completed single handedly. He was also the main contributor to the appointment assignment module and vaccinee registration interface.
- Vince was responsible for the frontend design. His responsibilities included building a clean and easy-to-use interface in HTML and CSS interface that sends data and receives responses from the vaccine registration web server. He created the sample interface which Colton connected to the Flask server.
- Michael was the testing coordinator. He was responsible for creating test cases for all components.
- Hans was responsible for creating the scoring module. He used online resources to adjust different criteria to their importance.

4. Reporting

Knowing what milestones were completed and by whom was extremely important. Our team Discord-- which is an IRC chat based service-- served as a great location for communicating milestone achievements and deadlines. Keeping everyone in the loop was crucial, and Discord was a perfect medium for those frequent updates.

5. Discussions and Builds

There were three scheduled meeting times each week: 2PM on Wednesdays, 4:15 on Fridays, and 3PM on Sundays. The meetings will be done on an IRC-based service known as Discord, using the auditory chat feature. Video was not shown. Alex led most of the meetings by walking through the due milestones on the project timeline and making quick adjustments as needed. Oftentimes, someone was blocked and needed help from another team member, which was always resolved during the meetings. The frequency of meetings allowed us to work far more quickly, and very little of the meeting time was spent in idle chatter.

There were three scheduled builds: April 15th, April 18th, and April 25th.

- The first build was done outside of a scheduled meeting time and primarily ensured that the skeleton code was able to be built. Most components weren't connected, or even existent, It only assured that everyone could push code to the repository.
- The second build connected a few components, namely the database and the web server. It was also a good time to check in with everyone and confirm that everyone was on track for the final build.
- The third build tied together all loose ends. In some sense, version 1.0 of our system was released and no modules would be changed in ways that have the potential of breaking other components at this stage. We had a comprehensive testing suite, a way to insert demo data, a working website, a working database, and a working command line interface application.

The builds were not as vital as is implied above. We used an extreme programming (XP) approach and attempted to keep code working throughout the development process. They served

as times where bugs were revealed, blockers were resolved, and individual work was adapted to work with other components.

The project timeline was adjusted semi-frequently to keep up with unrealized milestones (namely, the appointment assignment module), and late work. Nothing on the critical path was delayed, and so the project was delivered on time with the features that were expected. These expected features being the must have and should have of the original project requirements section.

Along with the extreme programming (XP) approach of keeping code functional, we also did some pair (or quintuple) programming during the meetings. Someone often shared their screen during the meetings, demonstrating their problems, progress, or updates for the team.

There was one in-person meeting between Colton and Alex on Friday, April 23rd, where the appointment assignment module was fully realized.

6. Ground Rules

Initially, we had some guidelines for Git commit messages^[1], code reviews, and collaboration. We didn't follow these rules, as they were not clearly communicated and were cumbersome. Our pair programming-based approach sufficed for code reviews and the exact text of Git commit messages were deemed unimportant.

7. Reflection

Overall, our project did not have any major hiccups. There were tasks that we did not realize their full breadth, and there were several things that were delivered late, but the project was never majorly held up by anyone. No group member was prevented from working by another. The most worrying moment was realizing that we had forgotten milestones for the appointment assignment module. Thankfully, that module was fairly straightforward to build and did not have many moving parts.

We're pleased with the product that we created. We feel that it has everything a system needs and nothing more. It's easy to install, is modular, and could scale to accomplish its task in the real world. It processes thousands of requests a second and can produce call lists with a hundred thousand users in less than a minute.

There were times where getting a hold of some team members was difficult. If we were to do this project again, we would definitely figure out a way to get in contact with each other sooner. Teammates generally don't have alerts on their school email or Discord, only on their

8. References

^[1]C. Beams, "How to Write a Git Commit Message", *Chris Beams Blog*, 2021. [Online]. Available: <https://chris.beams.io/posts/git-commit/>. [Accessed: 10- Apr- 2021].

Vaccine Distribution

Programmer Documentation

Alex Anderson, Hans Prieto, Vince Qiu, Colton Trebbien, Michael Wiltshire
April 25th, 2021

Table of Contents

- 1. Revision History**
- 2. Vaccinee Registration Web Server Endpoints**
- 3. Vaccinee Database**
 - 3.1. Database fields**
 - 3.2. Exported Methods**
- 4. Health Risk Score Module Exported Methods**

1. Revision History

<u>Date</u>	<u>Author</u>	<u>Description</u>
-------------	---------------	--------------------

4/25/21	AA	Create Programmer Documentation
---------	----	---------------------------------

2. Dependencies

Python3, Flask, and Flask-sqlalchemy are required to run this project.

3. Vaccinee Registration Web Server Endpoints

The vaccinee registration web server (webserver/Flaskserver.py) has three main endpoints:

/

- GET requests: serve Index1.html (home page)
- POST requests redirect client to /register endpoint

/register

- If GET request: serve Index2.html (registration page).
- If POST request:
 - Read registration health and contact data from user. Read out of an HTML form
 - Fields: heart_prblms, diabetes, lung_prblms, liver_prblms, cancer, pos_test, close_contact, symptoms, heightft, heightin, weight, age, fname, email, phone.
 - If an error is caught by try-catch, redirect to /notfilled.
 - If duplicate detected in database insert function (create_vaccinee() returns false), redirect to /failure.
 - If everything goes successfully, redirect to /success.

/success

- Respond to all requests by serving Index3.html (page showing successful registration).

/failure

- Respond to all requests by serving Index5.html (page showing that you've already registered).

/notfilled

- Respond to all requests by serving Index6.html (page showing that you've left something blank).

3. Vaccinee Database

3.1. Database Fields

The Vaccinee class defines the database fields:

- id: Integer, auto incrementing, primacy key
- full name: char[100], non-unique, cannot be null
- email: char[120], unique, cannot be null
- phone: char[30], unique, cannot be null
- score: float, non-unique, cannot be null
- password: char[100], non-unique, cannot be null
- has_been_vaccinated: bool, initially false
- date_created: date, initially current date

3.2. Exported Methods

The vaccinee database module (webserver/database/db.py) has the following exported methods:

Setting up the database

- `init_app(app)`

Takes a Flask application and attaches the database to that application. The app can be configured beforehand to set a location for the database.

- `setup_headless()`

Creates a flask application and attaches the database to that application. Defaults to using database.db as the database file. Particularly useful for loading db.py in a REPL shell.

- `create_all()`

Initialise the database if not already initialized. Creates the file at the specified location, runs any needed commands to create tables, etc.

Using the database

- `create_vaccinee(fullname: str, email: str, phone: str, score: float, password: str) -> bool`

Writes a new row to the database with the columns given: fullname, email, phone, score, and password. Returns true if successfully created the row and false if there was an error. If everything was set up correctly, false means that one or more of the fields was duplicated.

- `get_user_by_email(email: str) -> Optional[Vaccinee]`

Returns a vaccinee object by finding the record with the same email. If none exists, returns None. Currently only used in testing.

- `delete_user(email: str) -> bool`

Attempts to delete the user with the specified email. If there is no user who has that email, or some other error occurs, it will return False. Otherwise, it will return True.

- `generate_call_list(n: int) -> Optional[List[Vaccinee]]`

Creates a 'call list' with up to n number of users. Essentially, it will return up to n vaccinees who are not marked as being vaccinated, sorted by score and then by creation date. It also marks them as being vaccinated, and therefore once a user is produced by this function they'll never be encountered again. If it runs into an error, it will return None.

- `debug_get_all_users() -> List[Vaccinee]`

Returns all vaccinees in the system. Not to be used in production as it loads the entire database into memory. Very useful for testing, as you can ensure vaccinees actually got deleted/created/etc.

4. Health Risk Score Module Exported Methods

The health scoring module has the following exported methods:

- `calculate_bmi(heightft: int, heightin: int, weight: float) -> float`

Computes a person's BMI from their height and weight.

- `calculate_score(answers: list) -> float`

Calculates the score for a user. Answers is a list formatted as follows: [heart_prblms, diabetes, lung_prblms, liver_prblms, cancer, positive_test, close_contact, symptoms, feet, inches, weight, age]. Feet, inches, weight, and age are cast to integers while the rest are cast to integers (should be 0 or 1).

Vaccine Distribution

User Documentation

Alex Anderson, Hans Prieto, Vince Qiu, Colton Trebbien, Michael Wiltshire
April 25th, 2021

Table of Contents

- 1. Revision History**
- 2. Installation and Basic Operation**
- 3. Inserting Sample Data**
- 4. Assigning Appointments**

1. Revision History

Date

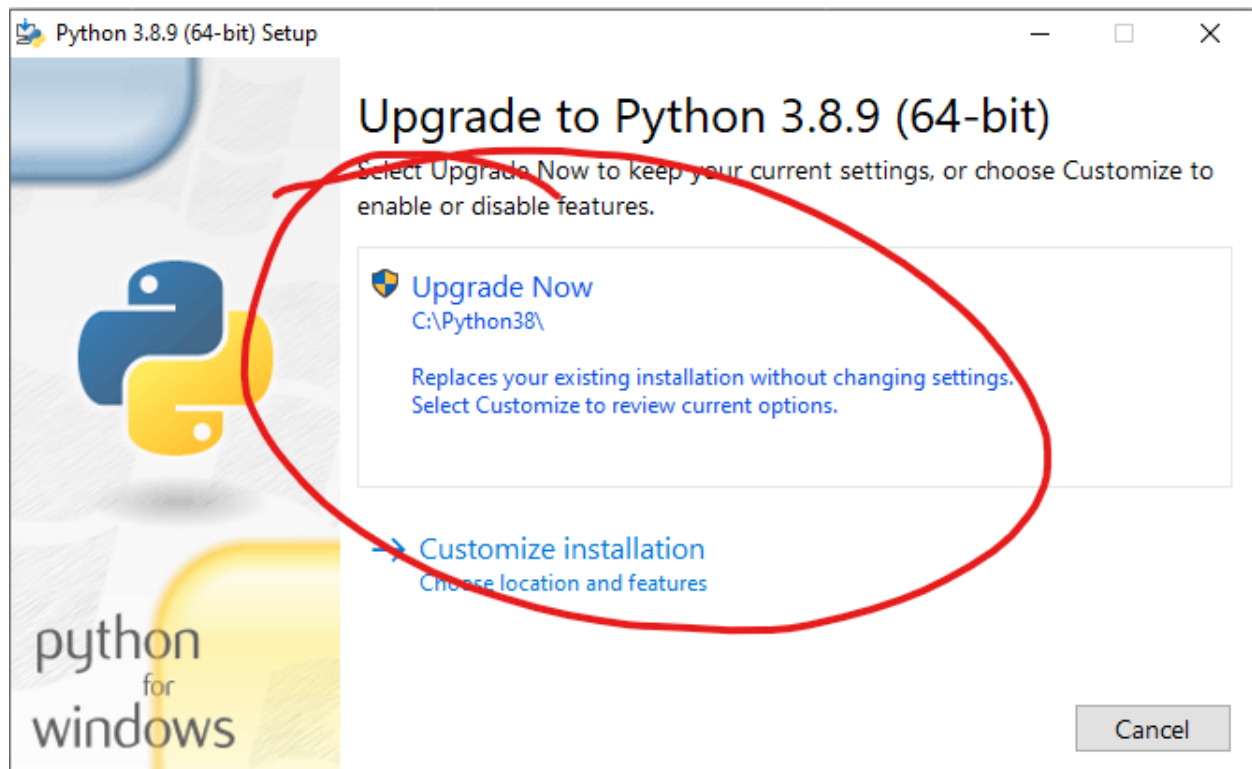
Author

Description

4/25/21 AA Create User Documentation

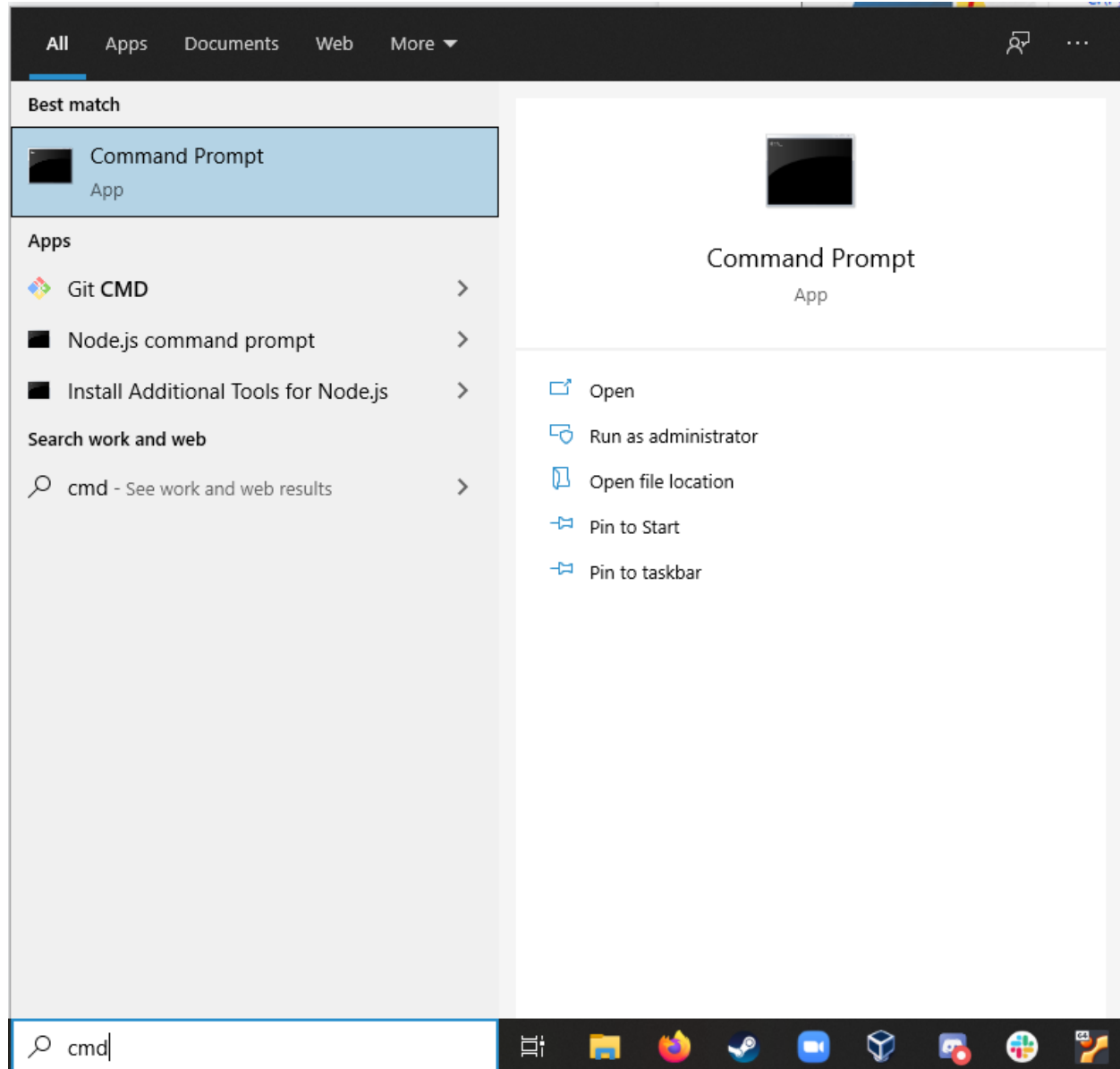
2. Installation and Basic Operation

First, install Python3 from <https://www.python.org/downloads/>.



(Note that this may say 'install now' instead. Click it if it says either upgrade or install).

Then, open a terminal or command prompt.



Then, install Flask and Flask-sqlalchemy. This can be done by typing the following in a terminal or command prompt:

```
python3 -m pip install flask flask-sqlalchemy
```

Windows users only: If you see this message about the Microsoft store,

```
Command Prompt
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

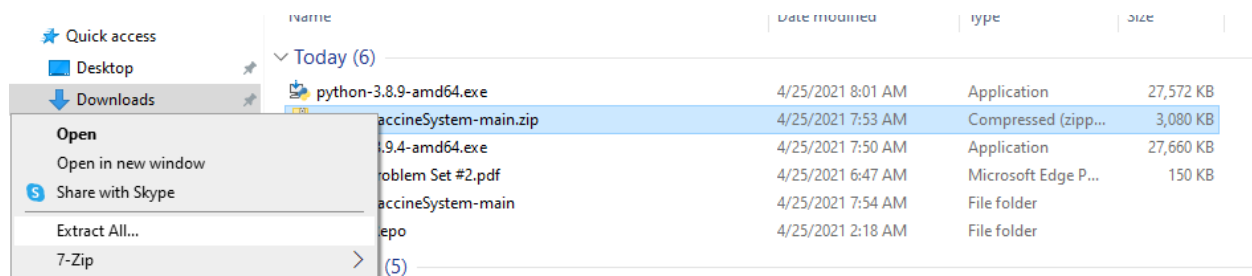
C:\Users\orutw>python3 -m pip install flask flask-sqlalchemy
Python was not found; run without arguments to install from the Microsoft Store, or disable this shortcut from Settings
> Manage App Execution Aliases.

C:\Users\orutw>
```

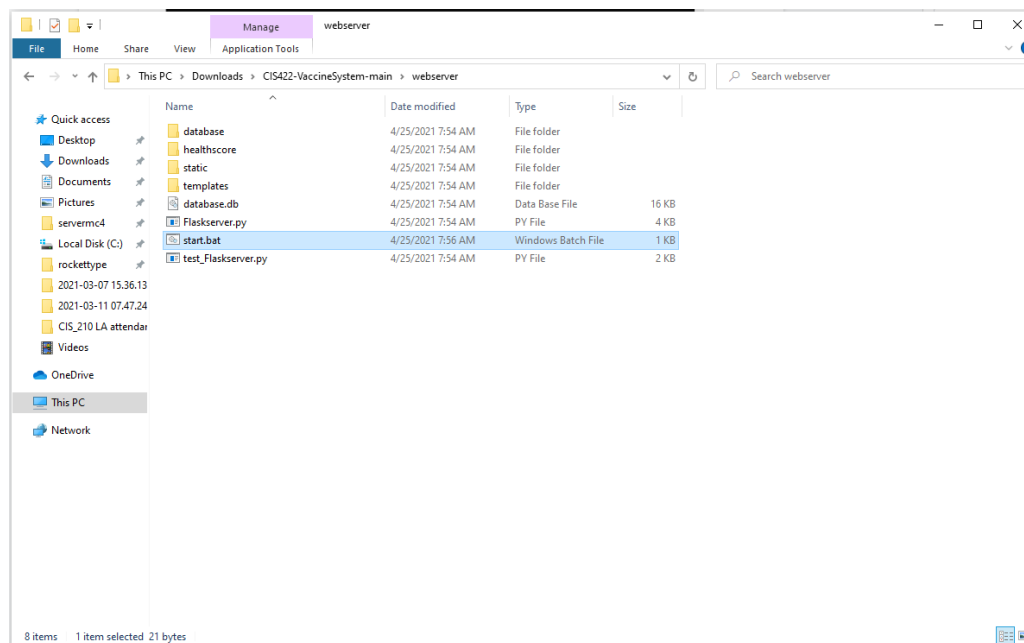
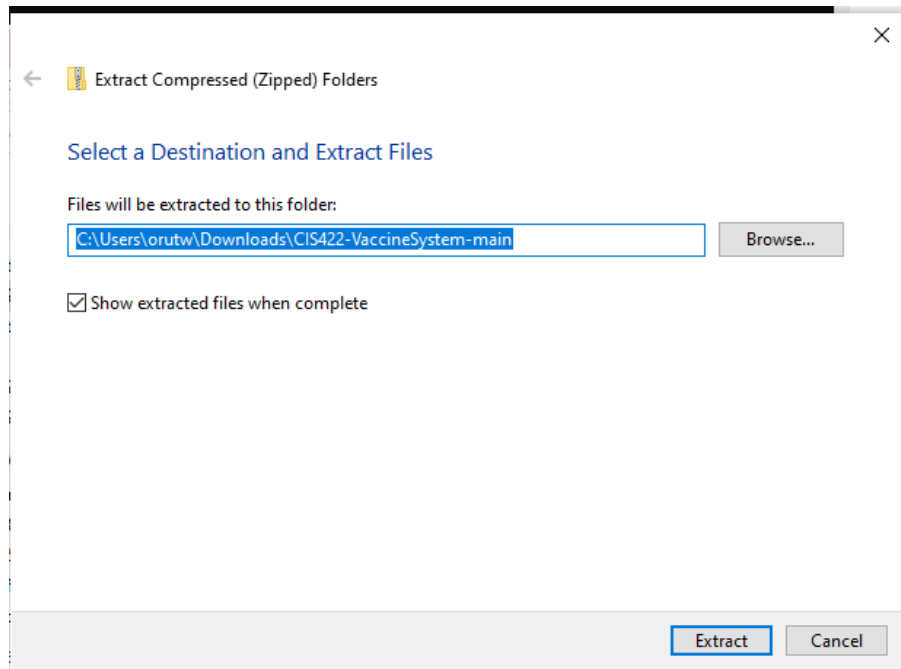
use instead:

```
python -m pip install flask flask-sqlalchemy
```






Download and extract the code (it should be included with this document). Then, navigate to the location where you extracted it and open the webserver folder. Note that if you received this guide with the project code, you will have to extract the zip file contained in the same directory as this file.



Name	Date modified	Type	Size
Today (6)			
python-3.8.9-amd64.exe	4/25/2021 8:01 AM	Application	27,572 KB
vaccineSystem-main.zip	4/25/2021 7:53 AM	Compressed (zipp...	3,080 KB
vaccineSystem-main	4/25/2021 7:50 AM	Application	27,660 KB
problem Set #2.pdf	4/25/2021 6:47 AM	Microsoft Edge P...	150 KB
vaccineSystem-main	4/25/2021 7:54 AM	File folder	
epo	4/25/2021 2:18 AM	File folder	

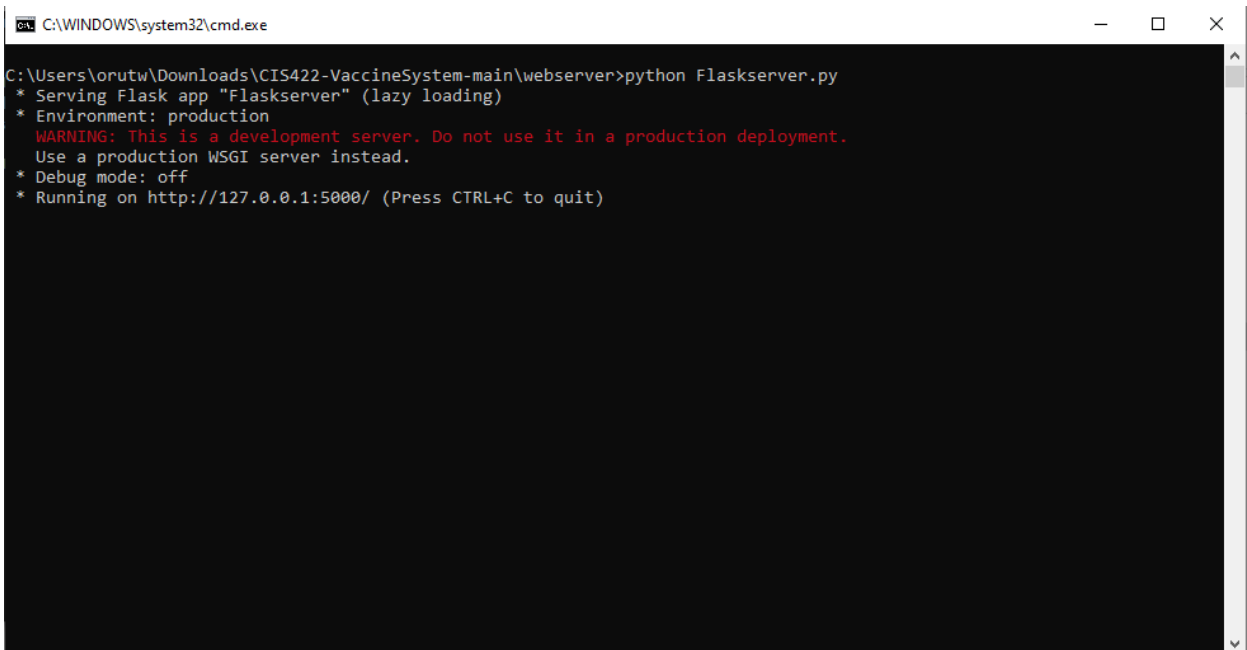


Open Flaskserver.py with Python. If you're unable to right click > open with Python, use the included start.bat on Windows (may show up as start depending on settings). On MacOS / Linux, right click in this folder and click the open a terminal here option. Then, type `python3 Flaskserver.py`.

	templates	4/25/2021 7:54 AM	File folder	
	database.db	4/25/2021 7:54 AM	Data Base File	16 KB
	Flaskserver.py	4/25/2021 7:54 AM	PY File	4 KB
	start.bat	4/25/2021 7:56 AM	Windows Batch File	1 KB
	test_Flaskserver.py	4/25/2021 7:54 AM	PY File	2 KB

Type: Windows Batch File
Size: 21 bytes
Date modified: 4/25/2021 7:56 AM

Double click this file, it should open a command prompt window with the server running inside of it.



```

C:\WINDOWS\system32\cmd.exe
C:\Users\orutw\Downloads\CIS422-VaccineSystem-main\webserver>python Flaskserver.py
* Serving Flask app "Flaskserver" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

```

To see it working, open a web browser and type `127.0.0.1:5000` in the address bar. You should see a page like this:



NOTE: Other users can view this page by navigating to your IP address. To set up this website to be accessible through the world wide web, you need to port forward TCP/IP port 5000. This is outside the scope of this guide. Please consult your internet service provider or network engineer for more information.

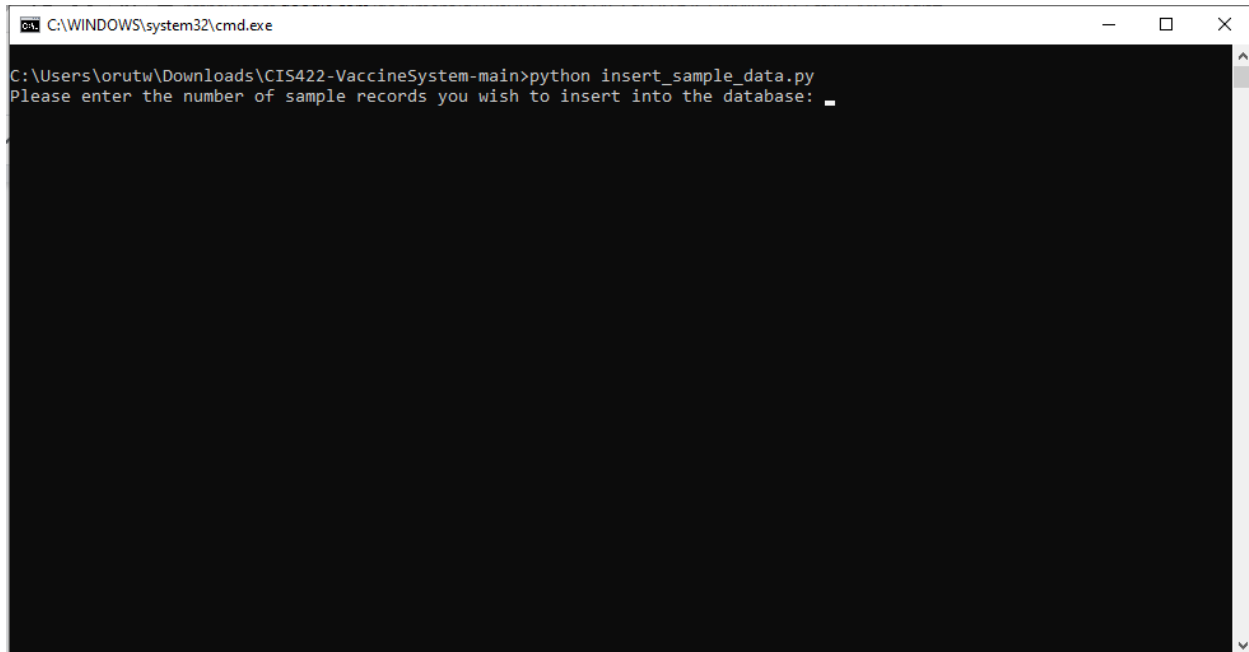
3. Inserting Sample Data

Inserting sample data allows for a quick demonstration of the project's functionality without needing anyone to register on the website.

To insert sample data, open the folder that you extracted earlier and navigate to the root directory (that is, the project folder). On Windows, double click the `insert_sample_data.bat` file. On Linux or MacOS, right click, click open a terminal here, and then type "`python insert_sample_data.py`".

Next, decide on how many sample records you'd like. A good sample set that would allow you to find yourself in the data would be around 10 records.

Finally, type whatever number you decide and hit enter. It will take around 1 second per 50 records you insert.

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text: "C:\Users\orutw\Downloads\CIS422-VaccineSystem-main>python insert_sample_data.py". Below this, a prompt message says "Please enter the number of sample records you wish to insert into the database:" followed by a cursor. The rest of the window is black, indicating that the user has not yet entered any input.

```
C:\WINDOWS\system32\cmd.exe
C:\Users\orutw\Downloads\CIS422-VaccineSystem-main>python insert_sample_data.py
Please enter the number of sample records you wish to insert into the database: _
```

4. Assigning Appointments

Once you get a shipment of vaccines, you can assign appointments. To do this, open up the `appointment_assignment.py` or `appointment_assignment.bat` file the exact same way you opened the `insert_sample_data` script.

Open the folder that you extracted earlier and navigate to the root directory (the project folder). On Windows, double click the `appointment_assignment.bat` file. On Linux or MacOS, right click, click open a terminal here, and then type “`python insert_sample_data.py`.”

You’ll be asked how many vaccines you received today. Enter the number of vaccines that you have the capacity to distribute and hit enter.

In the same folder, there will now be a file called “`call_list.csv`.” Open that with a spreadsheet viewer or distribute it to a call center to schedule appointments with those people.

```
C:\WINDOWS\system32\cmd.exe

C:\Users\orutw\Downloads\CIS422-VaccineSystem-main>python appointment_assignment.py
NOTE: Writting to call_list.csv, this can be overwritten by
specifying an argument to
NOTE: Running this file will overwrite the previous CSV sheets
How many vaccines did we recieve today?
```

WARNING: Running this program again will overwrite the call_list.csv file that was generated. Make sure to move that outside of the project folder before running it multiple times.

	A	B	C	D	E	F	G	H	I	J	K
1	Full name	Phone	Email	Score							
2											
3	Drwkh Mowlvxx	981-201-8512	fdyxam@agzowzsnd.com	49.4							
4											
5	Pdnffg Scimdyywr	319-098-8483	zejafd@lpjfwjdbzh.com	49.26							
6											
7	Bmvd Aqonk	686-930-6680	joekjxbatgdom@uolkqvcvcs.com	48.99							
8											
9	Jakd Mvistsn	833-372-2338	txapdwmk@cdk.com	48.99							
10											
11	Idymub Gnninyf	519-781-6585	wuwgnwoshihmeu@cvkcwcydoh.com	48.74							
12											
13	Hubq Frilhwmg	026-458-7102	nemtoez@aaopi.com	48.61							
14											
15	Yplouv Buimjtl	572-399-7492	iawmiduyurimtt@cccon.com	48.42							
16											
17	Txehbd Lhhfdt	425-206-7182	ggfzaqua@vcjpifuocy.com	48.36							
18											
19	Tyhftg Dptwjxt	852-723-1896	cytwxfwuykkai@ktagym.com	47.97							
20											
21	Hewmas Jrbjctrw	299-418-5323	uevpz@ydy.com	47.29							
22											
23	Tvlfex Hefunp	686-229-2358	comuudhufcivul@cihc.com	46.6							