Design Document

Pill.io

System Overview

Abstract

Pill.io is a machine learning empowered pill dispenser that will allow both caregivers and patients to have peace of mind as they partake in what for most of them is a daily requirement for their health and safety. This pill dispensing software will allow for medications to always be dispensed with only the needed medications and dosages based on the facial recognition of the patient attempting to get their medication. This will allow patients to get their medication without the struggle of opening or sorting through pill bottles and help prevent the possibility of an overdose on medications by no longer having patients access to all their medicine supplies at the same time. Moreover, caregivers will be able to access data regarding the frequency and timing of when/if their patients take their medications which will help inform their care.

Conceptual Design

Conceptually, Pill.io is a usable and accessible pill dispenser designed to help users take their medication in a timely manner. The hardware used to make this happen is Jetson Nano which will be the specific pill dispensing machine. Using React Native for the front-end work, this will be the web app and what the user sees when they set up their pill reminders initially. Expanding upon this later to include a more inclusive option array will be a priority and this will work natively on all operating systems and mobile devices. Through the backend we plan to implement the actual code of the pill dispensing, reminders, timers, etc. with Python and associated libraries.

Background

At the most basic level, the product being proposed is software for an automated pill dispenser. There exist a few examples of these types of products already, such as MedaCube (https://www.medacube.com/) and Hero (https://herohealth.com/). Both MedaCube and Hero have the included benefit of not requiring a person to sort medication into individual dosages, however, both products are meant to be used by a single person and do not include the need or ability for facial recognition. The product being proposed in this document would serve well at homes or care facilities with multiple patients that take medications, helping each of the patients get the medication they need without accidentally taking those that are not meant for them.

In addition to this, commercial-scale pill dispensers, such as those used in nursing homes, often do not provide an easy-to-use web interface that is compatible with all devices and can be seamlessly set up for a multitude of users, after which it can be easily authenticated by anyone, regardless of their technological knowledge. Furthermore, our pill sorter/dispenser provides a child-proof and stranger-proof medication management system that allows for a worry-free experience, as it prevents all theft of the user's medication. In fact, more than 750,000 prescriptions are stolen each year, most of which are family members or friends, where our product would entirely prevent such occurrences from happening (Carman & Adhopia, 2018). This is also makes for an amazing advancement in overdose prevention, a risk that most elderly users face due to their inability to properly manage their medication intake, where our product would cut all risk of overdoses or accidental ingestion of extra or improper medication from happening.

System Block Diagrams

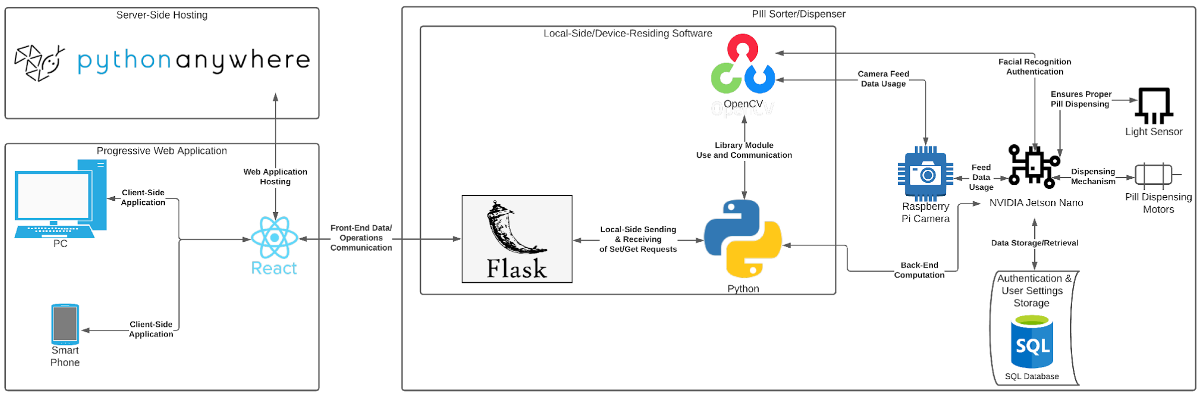


Figure 1

As demonstrated in Figure 1 above, each line contains a description of the interaction between the two components which it connects. Each group of components is contained within a boundary to represent their general category within the application itself.

On the hardware side of things, the NVIDIA Jetson Nano will be the foundation of the pill dispenser, where it will be connected to several pill dispensing motors, a Raspberry Pi camera, and will have access to internal storage where it will utilize the OpenCV library and a custom local database used for user and authentication setting storage. The Jetson Nano will communicate with the OpenCV library in order to allow permissions and begin the process of facial recognition and authentication, where the library itself will access the camera feed. The camera will also communicate with the Jetson Nano itself by allowing camera feed usage for storage of faces to be used for authentication. These faces will then be categorized within the aforementioned local database, where all of the sensitive data will be stored securely at the local hardware level. In terms of proper pill dispensing, an IR light breaker bar will be used in order to count the pills that come down the chute. If a pill passes past the light breaker bar, it will notify the software, after which the software will increment the pill count. All of the Python, its libraries, and Flask will reside locally on the SD card on the Jetson Nano as well as its internal storage.

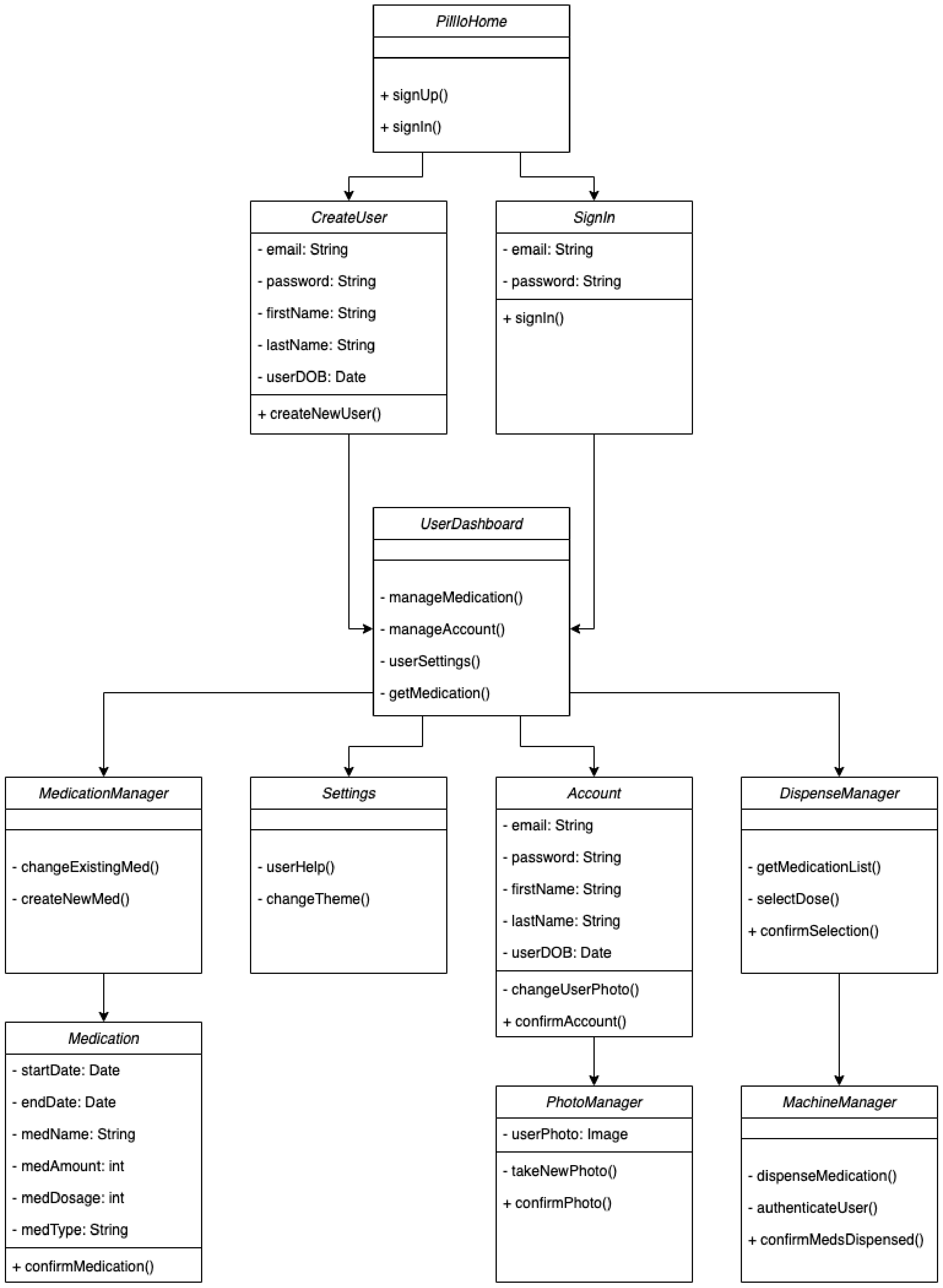
The Jetson Nano will utilize Python in order to conduct any and all data processing or computation, where Python will communicate with Flask in order to prepare all data for front-end display. Python, Python's OpenCV library, and the Flask API will all reside locally on the Jetson Nano. Flask will be used as an intermediary layer between the Python back-end and React front-end parts of the application and device. Flask will take in GET requests from the user's actions in the React app and will send those requests to Python, where the needed information will be obtained and sent back to Flask. Then this is where Flask will send this information in order for the React app to update any given fields or components. Flask will also take in SET requests from the user's actions in the React app and send those requests to Python, where the needed information will be changed or updated, and this is where Python will send the confirmation back to Flask, where in turn Flask will send it to the React app. Flask will conduct its operations and send these requests over LAN.

On the front-end side of things, we will be using React to develop a progressive web application that will be usable by anyone with a web browser, including desktop, iOS, and Android users. The web application will be hosted on a free server that allows for the hosting of web applications by students, called PythonAnywhere. The application will, thus, be accessible by authenticated users without the use of a locally hosted application. The application directory and application project files will be stored locally on each team member's computers and constantly saved/updated to GitHub by each individual member.

General Requirements

- Google Chrome, Mozilla Firefox, or some other web Browser
- Graphics Card (and a Machine to run it to train the Machine Learning models)

Class Diagram & Class Fields/Methods



PillIoHome – The class responsible for containing the home page of the application, where the user will be greeted with a screen that prompts them to either sign in as an existing user or sign up as a new user.

+ signUp() – Method used to sign up a new user in the database. This method will be called when the user clicks the "Sign Up" button on the home page. The method will take the user to the CreateUser screen and its corresponding class.

+ signIn() – Method used to sign in an existing user and crosscheck their information against the existing user database. The method will be called when the user clicks the "Sign In" button on the home page. The method will take the user to the SignIn screen and its corresponding class.

CreateUser – The class responsible for containing the new user signup page of the application, where the user will be greeted with a form that they must fill out in order to successfully register as a new user.

- email: String – The email field; will take in the email of the user signing up.

- password: String – The password field; will take in the password of the user signing up. This field will be required have at least 8 characters and at least one digit, which will be checked when user clicks the button to call createNewUser().

- firstName: String – The first name field; will take in the first name of the new user.

- lastName: String – The last name field; will take in the last name of the new user.

- userDOB: Date – The Date of Birth field, will take the date of birth of the new user, which will be formatted as a Date object internally. The user will be able to utilize a formatted entry box in order to input their date of birth in MM/DD/YYYY format.

+ createNewUser() – Method used to confirm the data entered to be used for the creation of a new user. This method will check that all of the fields are filled out, that there are no illegal characters in any of the fields, and that each field is formatted or filled out correctly, as previously mentioned. If this method finds that the user has successfully registered, the method will return success and take the user to the UserDashboard screen and its corresponding class. If the method finds that any of the conditions to register failed, it will return failure and display the failed condition to the user visually on the CreateUser screen.

SignIn – The class responsible for containing the existing user sign in screen, where the user will be greeted with a form that they must fill out in order to successfully register as a new user.

- email: String – The email field; will take in the email of the user signing in.

- password: String – The password field; will take in the password of the user signing in.

+ signIn() – Method used to confirm the data entered to be used for the signing in of the existing user. The method will crosscheck the login information against the existing user credentials database, where it will return success if said credentials are found and subsequently take the user to the UserDashboard screen and its corresponding class. If the credentials are not found or if one of the fields if empty, the method will return failure and prompt the user with the corresponding error, telling them to try again.

UserDashboard – A super class that consists of multiple subclasses about view and customize the user medication data and personal information

+ manageMedication() - This function will open and direct the users to the MedicationManager screen. It allows the users to modify and manage their medications / medicines.

+ manageAccount() - This function will open and direct the users to the account function / the account settings. It allows users to view and modify their authentication photos, login account information, and personal information such as email, password, first name, last name, dob, etc. Any changes will be stored and saved to the database.

+ userSettings() - This function will open and direct the users to the setting function where users able to get help about the instruction how to use pill.io. The setting function also provides other useful information to help users.

+ getMedication() - This function will open and direct the users to the dispense manager. It is the process of dispensing pills where users start with selecting the medicines, the authentication, dispense pill, and finally the confirmation pill taken.

MedicationManager- A class that is responsible for modifying and managing the medication on the existing user

+ changeExistingMed() - An option to change or modify the existing medication and saved the changes to the database where it will dispense the new medication.

+ createNewMed() - An option to add new medication for the user to the database. The new medication will be added to the medication list in the database where user have the option to select the new medication.

Settings –

+ userHelp() - Displays general FAQs about some of the processes of Pill.io. This includes information such as how the machine works - the process of how pill dispensing takes place through the machine. This also includes information such as how to add another user to the database of the machine for record keeping, how to add new medication to the database for record keeping, and basic information such as how to retrieve the specific medication once you placed your prescription

+ changeTheme() - Changes the general color output on the machine. This can vary in light to dark colors based on our color wheel options.

Account – The class responsible for containing the user's account settings screen, where the user is able to change any of their account settings, including their authentication photo, and store those changes to the database.

- email: String – The email field; will take in the email of the user signing up.

- password: String – The password field; will take in the password of the user signing up. This field will be required have at least 8 characters and at least one digit, which will be checked when user clicks the button to call confirmAccount().

- firstName: String – The first name field; will take in the first name of the new user.

- lastName: String – The last name field; will take in the last name of the new user.

- userDOB: Date – The Date of Birth field, will take the date of birth of the new user, which will be formatted as a Date object internally. The user will be able to utilize a formatted entry box in order to input their date of birth in MM/DD/YYYY format.

- changeUserPhoto() – Method to change the authentication photo of the existing user, where said photo will be used by the authentication algorithm in order to verify the identity of the user attempting to dispense their medication. This method will be called by the user clicking on a "Change Photo" button on the screen, where it will subsequently take them to the PhotoManager screen and its corresponding class.

+ confirmAccount() – Method to save and store all of the changes the user has made on their account, these changes will be crosschecked to make sure there are no illegal characters in any of the fields and that each field is formatted or filled out correctly. If there are no errors, the method will return success and the information will be stored into the database, taking the user back to the UserDashboard screen. If there are any aforementioned errors, the method will return failure and prompt the user with the corresponding error, telling them to try again.

DispenseManager – The class responsible for starting the pill dispensing process by determining which pills should be dispensed and checking that each dispensation ,was successful, where a user's medication list is looked through to find the needed medication(s) to dispense, and each dispensation is then verified

- getMedicationList(): Method that finds the medication list of the current user and returns its contents.

- selectDose(): Method that allows the user to select which dose of their medication they are looking to take, i.e. morning, afternoon, evening etc, that will indicate which and how many of the medication on the user's medication list need to be dispensed at the time

+ confirmSelection(): Method that checks whether or not the light sensors processed a pill passing through to ensure a successful dispensation of a single pill

Medication – The class responsible for containing the user's account settings screen, where the user is able to change any of their account settings, including their authentication photo, and store those changes to the database.

- startDate: Date – A Date object that indicates the starting date of the prescription that the user is taking, where the user will be able to utilize a formatted entry box in order to input said date in MM/DD/YYYY format.

- endDate: Date - A Date object that indicates the ending date of the prescription that the user is taking, where the user will be able to utilize a formatted entry box in order to input said date in MM/DD/YYYY format.

- medName: String – The name of the medication that the user is taking.

- medAmount: int – The quantity of pills that a user has in their prescription.

- medDosage: int – The total dosage of medication that the user has to take each time, in milligrams.

- medType: String – The classification or type of medication that the user is taking, for example: SSRIs, Anticonvulsants, etc.

+ ConfirmMedication() - Method to save and store all of the changes the user has made on their medication/prescription information, these changes will be crosschecked to make sure there are no illegal characters in any of the fields and that each field is formatted or filled out correctly. If there are no errors, the method will return success and the information will be stored into the database, taking the user back to the MedicationManager screen. If there are any aforementioned errors, the method will return failure and prompt the user with the corresponding error, telling them to try again.

PhotoManager – This class is responsible for taking pictures and recognizing the pictures taken

- userPhoto: Image

- takeNewPhoto() - Function to start the picture taking process using the raspberry PI camera.

+ confirmPhoto() - Function to confirm the picture that is taken using the "takeNewPhoto" function.

MachineManager-

- dispenseMedication() - Starts the process of dispensing specific medication based on the information the user provides for their prescription.

- authenticateUser() - Verifies the user's credentials to make sure the proper medications are assorted and delivered to the right person.

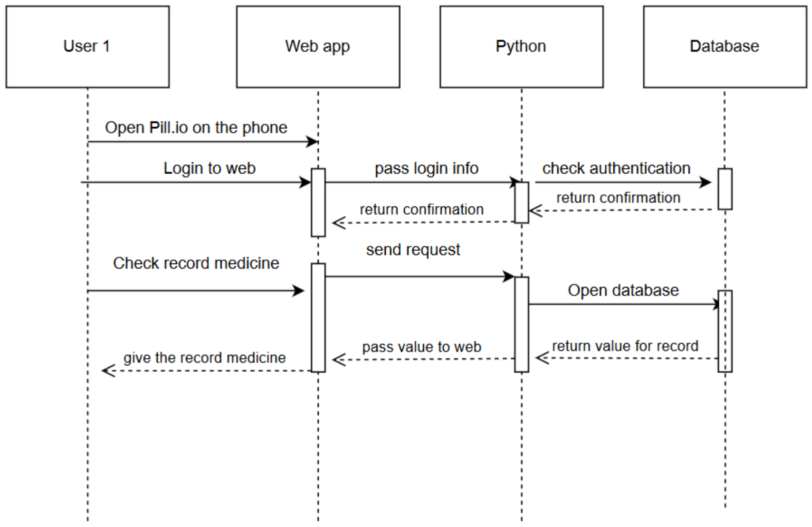
+ confirmMedsDispensed() Establishes confirmation that the pills the user requests have been delivered through the dispenser and the user has taken them from the dispenser tray.

Use Cases

Use Case 1

An elderly female patient wants to take her medicine pills

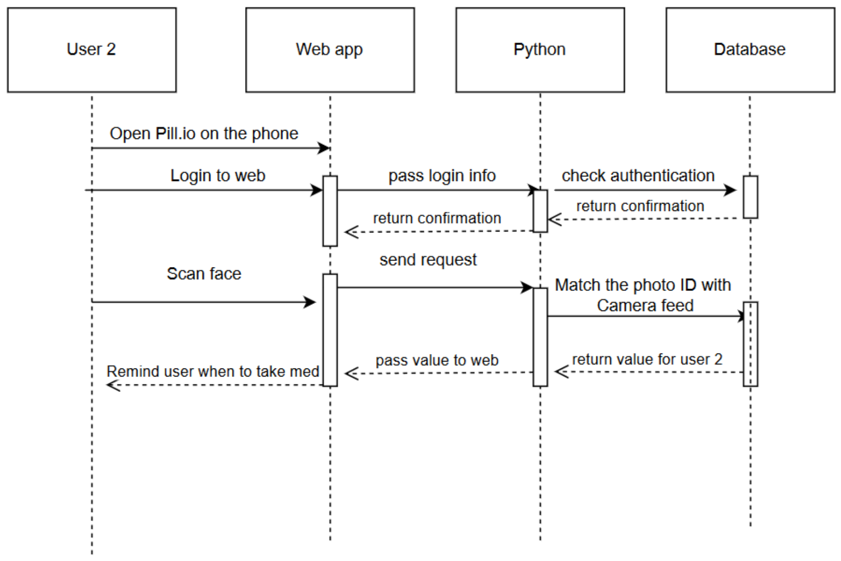
- She uses Pill.io to help her take the right medicine
- Because she needs to take the right pills at an allotted time, she takes full advantage of Pill.io's features.
- She uses the reminder feature through Pill.io's website to make sure she's taking the right pill at the right time.
- Shes able to scan her face through the Pill.io app so that the application records her specific information into the database, so a record is kept of when she needs to take her medication at the right time



Use Case 2

A newly heart transplanted patient needs assistance taking his medicines on a regular basis.

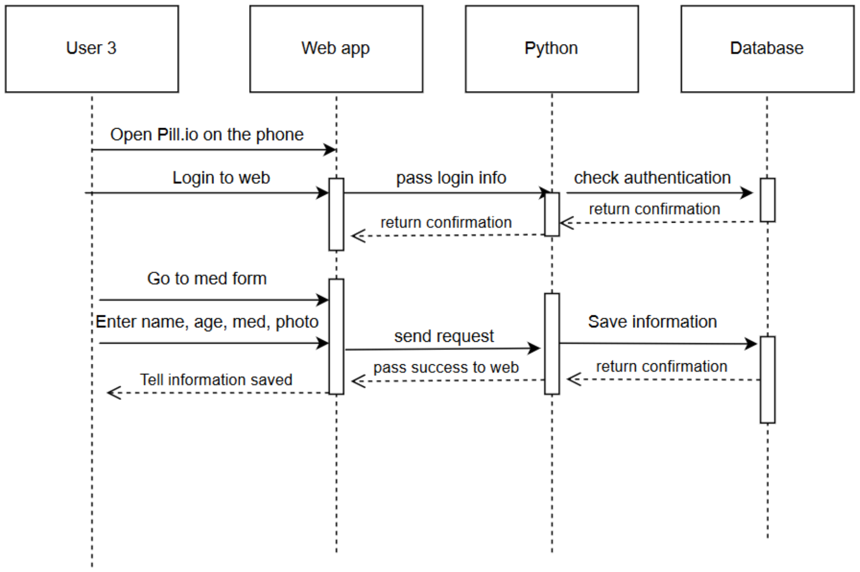
- His transplant team suggests he start using Pill.io
- Sets up Pill.io effortlessly
- Pill.io helps take medication promptly with its ease of use
- Fully utilizes Pill.io's camera feature that checks and matches the user's face with their info on the database.
- Information kept securely in the database.
- Reminder will be set for him to take his medication



Use Case 3

A user wants to save their information on the website

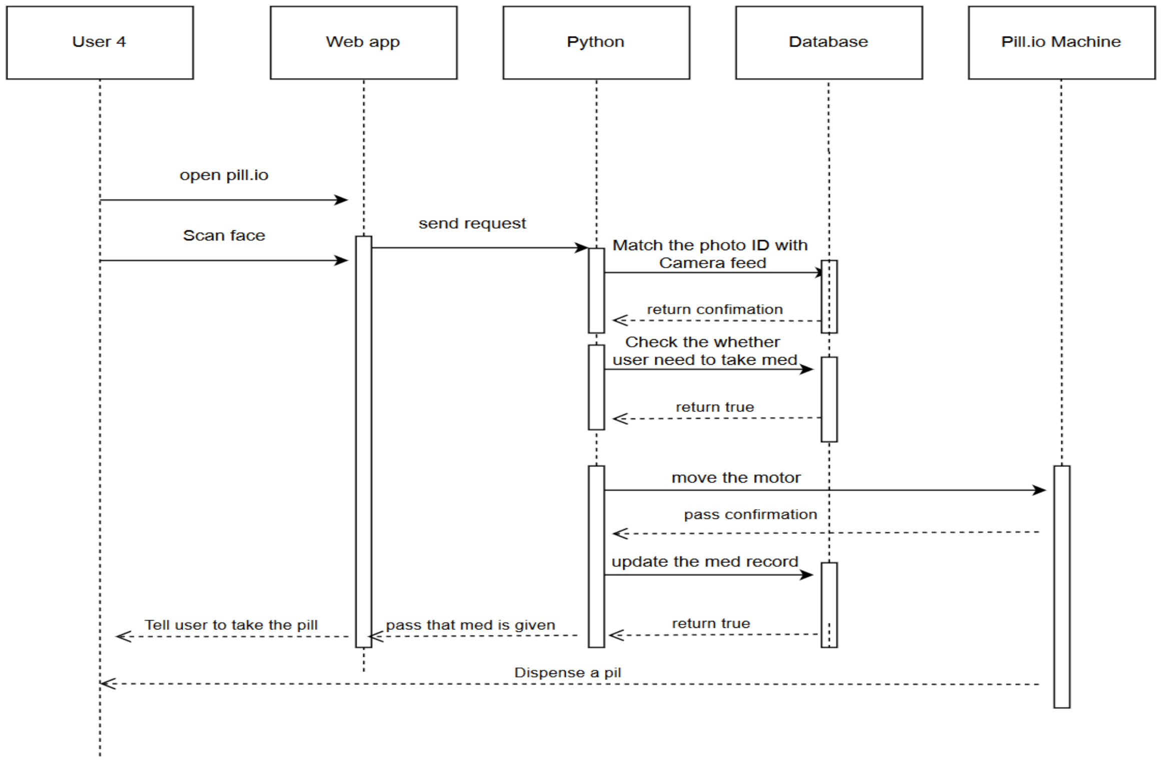
- A new Pill.io user wants to set up their device
- User connects the pill dispenser to Wi-Fi
- They input their data on the registration page
    - Enters name~~;~~
    - Enters their ag~~e,~~
    - Enters all the medicines that they are taking
    - Sets up face id by scanning their face and having it saved in the database,
- Then clicks submit to send the data to be saved
    - The data will be saved in the database which will be used later



Use Case 4

A user wants to reduce hospital visits and reduce the untaken medicine

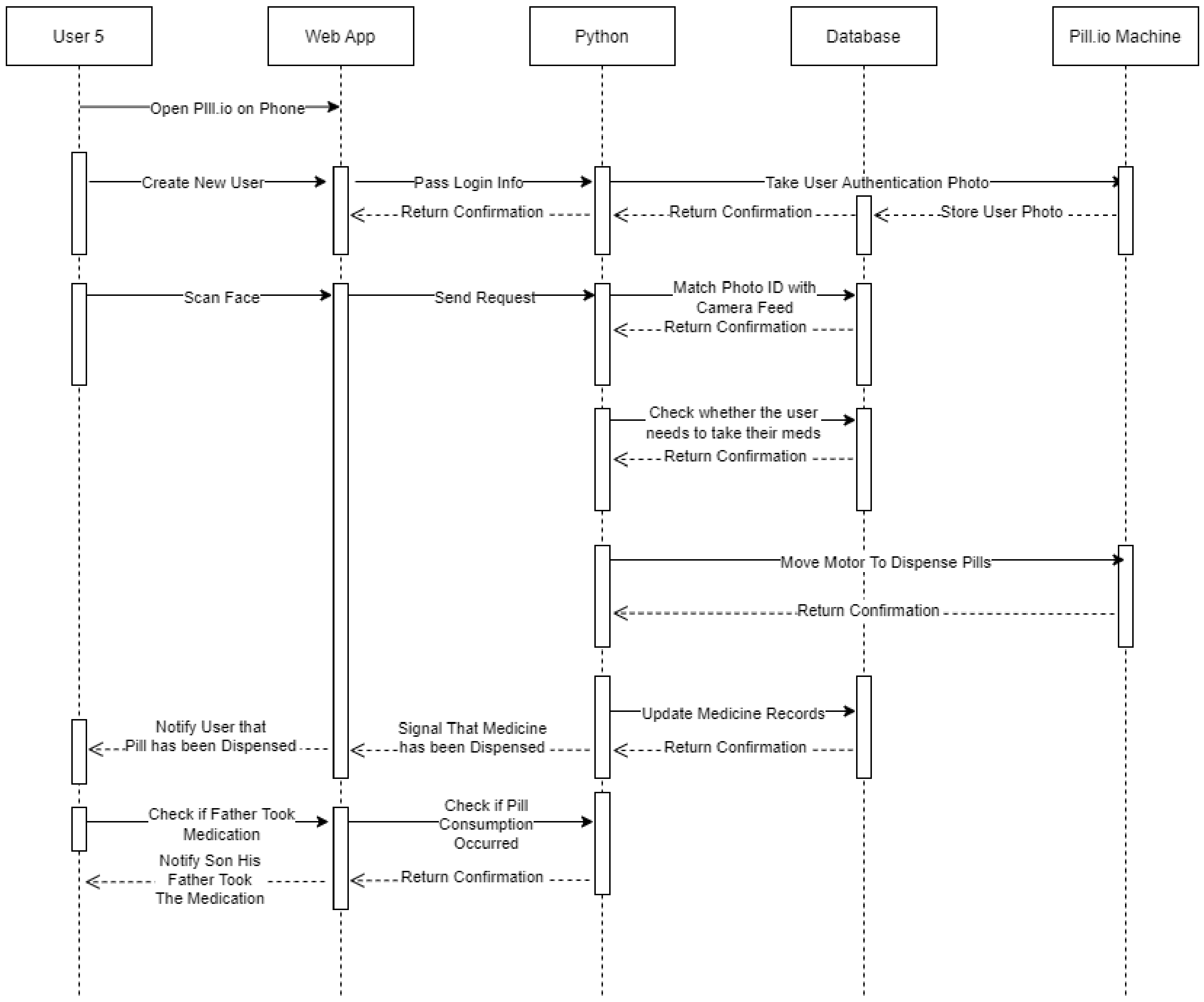
- User buys Pill.io to utilize its features for taking their medication at the allotted time slot their prescribed
- Because the user has clients that need to take prescribed medication, he needs to record their data in the database
- Once the data is recorded, the appropriate medication can be distributed when the allotted time has arrived to take the medication in order to take the medication first, the User scans with face recognition so the database recognizes who exactly it is who needs the medication along with checking if their allocated time slot is right for taking their medicine
- The appropriate medication is then retrieved and given to the client if their information in right regarding their appropriate time they need to take their medication and if the face recognition proves they are who they say they are



Use Case 5

A Son wants to make it easier for his dad to take his medicines

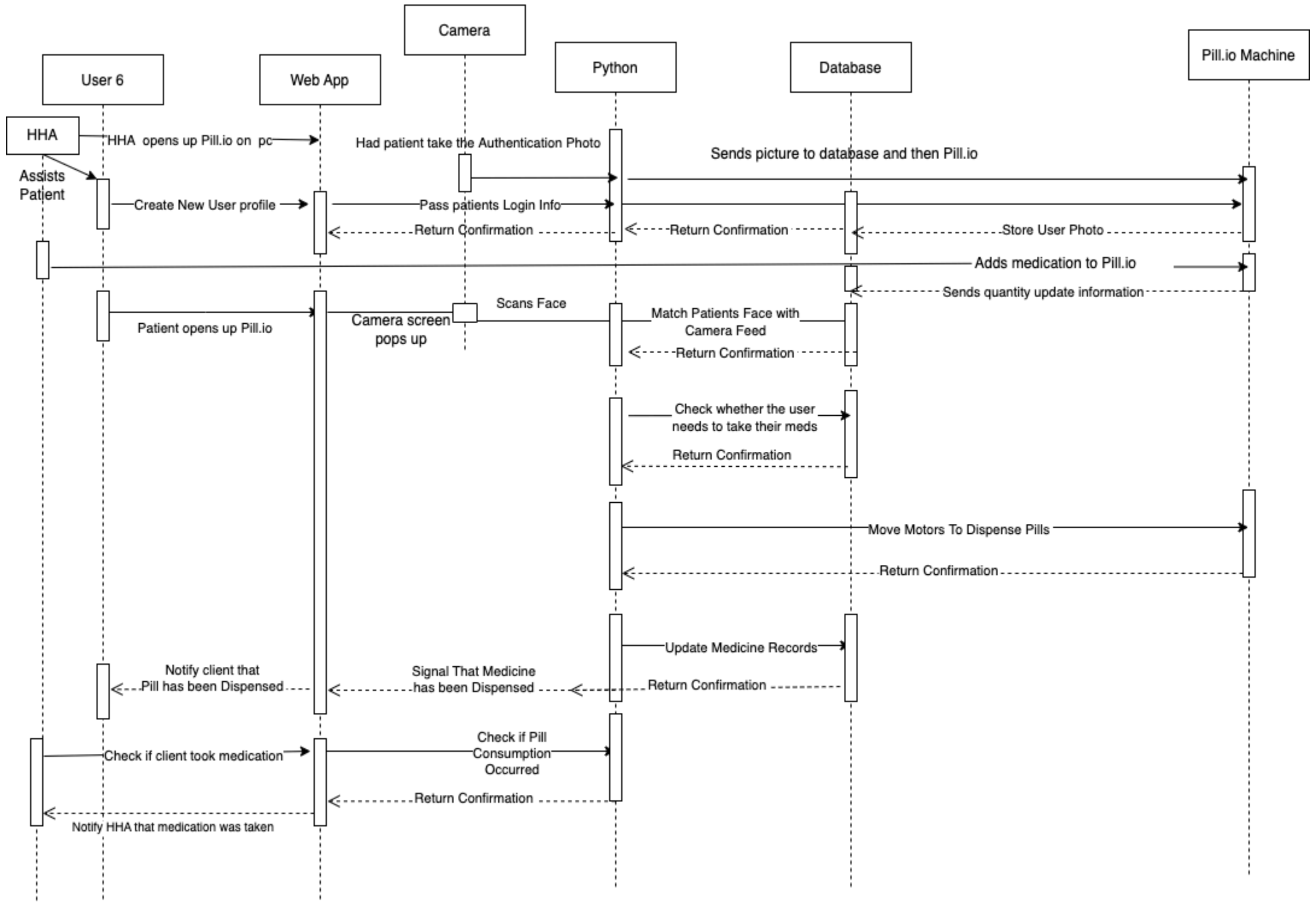
- Son buys Pill.io for his father
- Sets up facial recognition and other required details
- Fills up Pill.io with medicines
- Dad scans his face and gets his medicine
- Both father and son can check to see if the medications were taken
- Using Pill.io is much faster than acquiring specific medication manually.
- Pill.io's timetable of when specific medication would be ready made it more accessible to retrieve his dad's medication


User 5
Web App
Python
Database
Pill.io Machine
Open Pill.io on Phone
Create New User
Pass Login Info
Take User Authentication Photo
Return Confirmation
Return Confirmation
Store User Photo
Scan Face
Send Request
Match Photo ID with Camera Feed
Return Confirmation
Check whether the user needs to take their meds
Return Confirmation
Move Motor To Dispense Pills
Return Confirmation
Update Medicine Records
Notify User that Pill has been Dispensed
Signal That Medicine has been Dispensed
Return Confirmation
Check if Father Took Medication
Check if Pill Consumption Occurred
Notify Son His Father Took The Medication
Return Confirmation

Use Case 6

A home health aide wishes to accelerate and automate her client's medicine consumption

- A HHA or CNA purchases a Pill.io for her client through subsidized state government tax funds OR insurance funds (after proper approval of device)
- The HHA assists the client with the setup of the device and its features
- The HHA will transfer the medications from their bottles to the Pill.io
- The client will scan their face in order to authenticate themselves to the machine
- The HHA can utilize any Wi-Fi-enabled device to access the web application in order to ensure the proper consumption of medication by their client
- The HHA is able to perform their other duties without the hassle and worry of their patient needing assistance
- The HHA can set up notifications to inform her if the patient has not taken their medication while the HHA is not on duty, allowing them to be informed of the issue quickly and take proper measures


Camera
User 6
Web App
Python
Database
Pill.io Machine
HHA
HHA opens up Pill.io on pc
Had patient take the Authentication Photo
Sends picture to database and then Pill.io
Assists Patient
Create New User profile
Pass patients Login Info
Return Confirmation
Return Confirmation
Store User Photo
Adds medication to Pill.io
Sends quantity update information
Scans Face
Patient opens up Pill.io
Camera screen pops up
Match Patients Face with Camera Feed
Return Confirmation
Check whether the user needs to take their meds
Return Confirmation
Move Motors To Dispense Pills
Return Confirmation
Update Medicine Records
Notify client that Pill has been Dispensed
Signal That Medicine has been Dispensed
Return Confirmation
Check if client took medication
Check if Pill Consumption Occurred
Return Confirmation
Notify HHA that medication was taken

Entity Relational Diagram & Table Design

Database

For the storage of personal and medical information of the patients connected to each individual Pill.io machine, we will be using an SQL Database. This provides us with a simple yet efficient way to store the information that we know we will need, such as medicines and their dosages, patient pictures, etc. Each SQL Database will be stored on the Jetson Nano itself, allowing the sensitive data to remain local to each machine to avoid compromising any individual's private information.

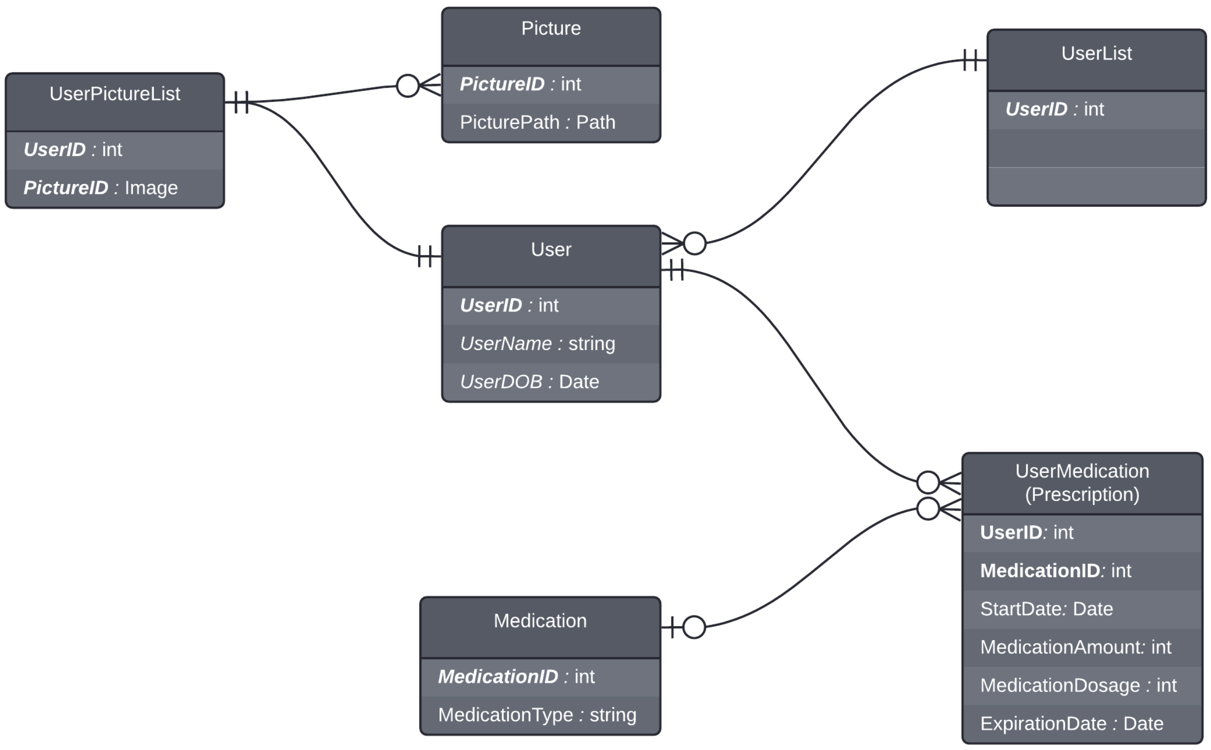


Figure 2

Database Objects

- User
    - UserID – an Integer that is a unique identifier for a single user, for internal use
    - UserName – a String containing the name that will be used to refer to a single user
    - UserDOB – a Date that is that user's date of birth
- UserList
    - UserID - an Integer that is a unique identifier for a single user, for internal use
- Picture
    - PictureID – an Integer used to refer to a single image, for internal use

  - PicturePath – a Path used to indicate where a specific picture is located in the database storage
- UserPictureList
  - UserID – an Integer that is a unique identifier for a single user, for internal use
  - PictureID – an Image of a single user's face that will be use for face recognition / user authentication
- Medication
  - MedicationID – an Integer that is a unique identifier for a single medication variation, for internal use
  - MedicationType – a String containing the type of medication (name of medication, pill form: tablet, capsule, etc), and dosage per pill)
- UserMedication (Prescription)
  - UserID – an Integer that is a unique identifier for a single user, for internal use
  - MedicationID – an Integer that is a unique identifier for a single medication variation, for internal use
  - StartDate – a Date indicating when the associated medicine was prescribed
  - MedicationAmount – an Integer designating the amount of medication provided in a single prescription
  - MedicationDosage – an Integer indicating how many units of the medication should be taken
  - ExpirationDate – a Date indicating when the associated medicine will expire

Repositories

- **Final Release**
  - https://github.com/Capstone-Projects-2022-Spring/project-pill-io/releases/tag/v4.0.0
- **Prior Releases**
  - https://github.com/Capstone-Projects-2022-Spring/project-pill-io/releases

References

Carman, T., & Adhopia, V. (2018, June 7). *CBC News*. Retrieved from CBC News: https://www.cbc.ca/news/canada/missing-drugs-pharmacies-part1-1.4708041

Hero Pill Dispenser, Medication Manager & Pill Organizer. (2022, February). Retrieved from HeroHealth: https://herohealth.com/

MedaCube. (2022, February). Retrieved from PharmaAdva: https://www.medacube.com/