CSC 400 Final Report

Med-Minder:
A Web-Based Medication Management Application

William Maroney
CSC 400
SCSU
8/26/2019

1. Introduction

Med-Minder is a web-based application which manages a user's medication information including dosages, refills, and reminders of when to take the medication. The application's primary objective is to provide users with a scheduling and optimizing prescription management web-based application using an automatically populated calendar. Med-Minder interfaces with Google Calendar API and using python library called Pytesseract and OCR to scan a prescription label and the calendar API to populate a user's calendar with events or reminders. Med-Minder has a high impact in all branches of modern society because of widespread usage of maintenance medicines and the general increase in medications being prescribed and available. Med-Minder will require server/client architecture with the server holding the medication database and the client side holding the specific medication data and calendar events which will be stored locally.

2. Architecture

Med-Minder is a web-based application for use on any device that can access the internet and has a built-in camera. Med-Minder will work without a camera but then would require the user to enter the information manually. Med-Minder is based on client/server architecture with the user being the client and the server holding the medication comparison database and user database. Medication information specific to the user will be saved locally in a small user medication database.

Major Inputs and Outputs of the Application

The major inputs and outputs of the system are outlined in this section and subsequent diagram.

Inputs to the Application from the User

1. Inputs to the application include user login information such as username and password with both accepting alpha-numeric combinations.
2. Another input into the system would be medications and their attributes which would also be alpha-numeric.

Outputs from the Application to the User

1. The application will output an email to the user's account when a new prescription is added.
2. Another output of the system is when the user chooses to scan a prescription the application automatically redirects them to the add Rx Form with a text field above with the scanned text.

Input / Output and Architecture Diagrams

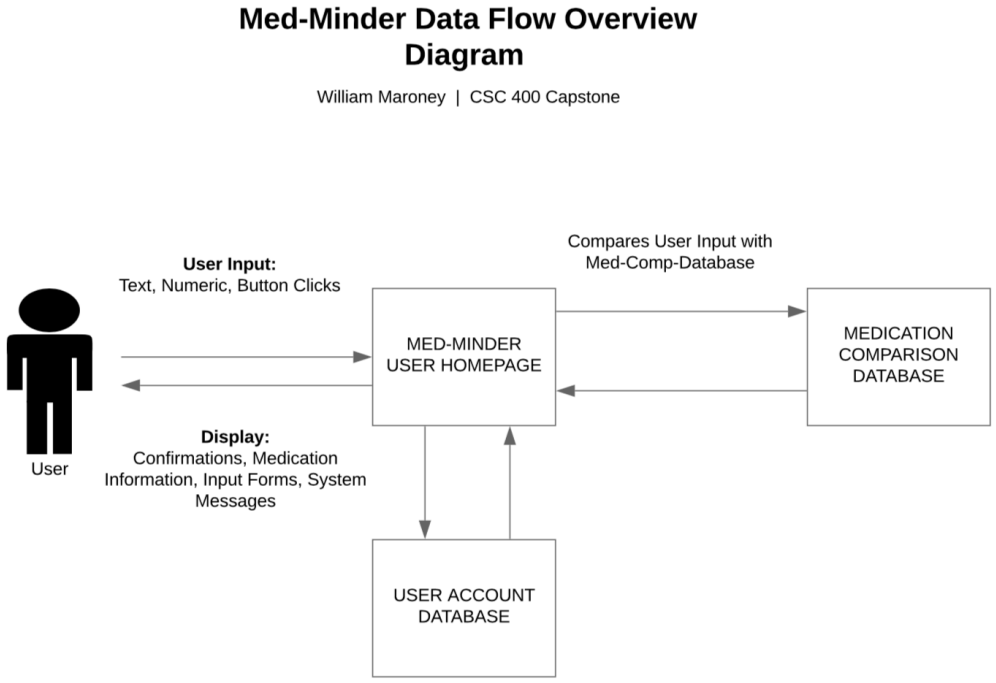

Figure 1: Original Data Flow Diagram
(Comparison Database not implemented in Version1)

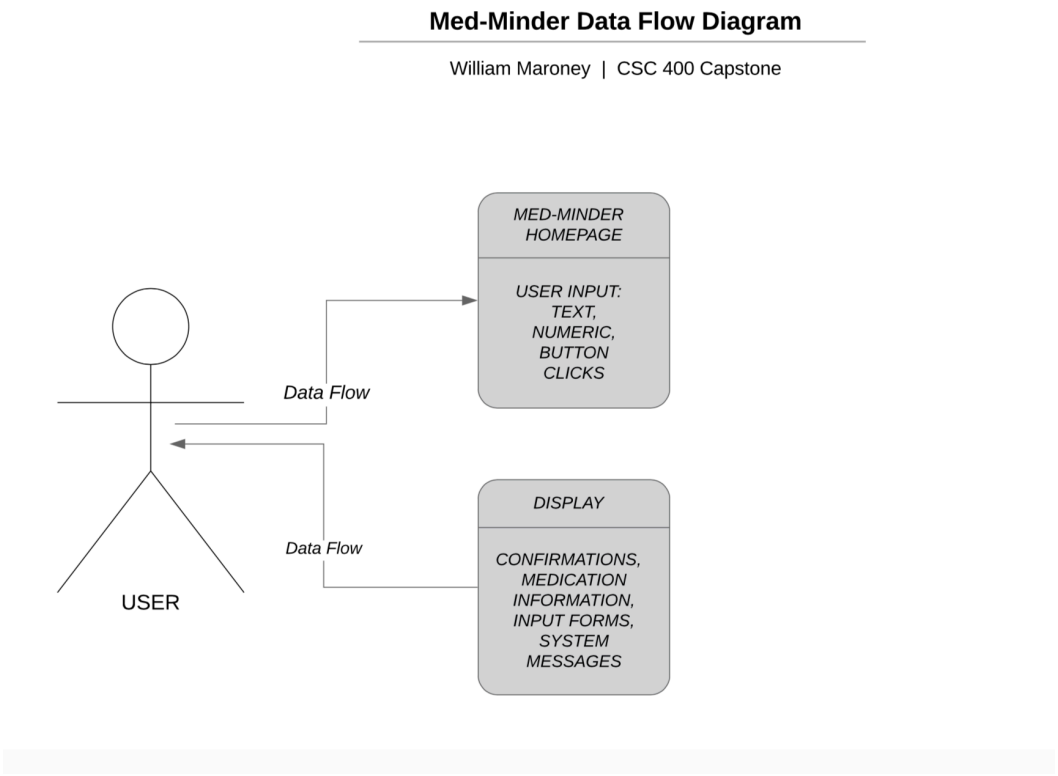

Figure 1.1: Current Data Flow Diagram

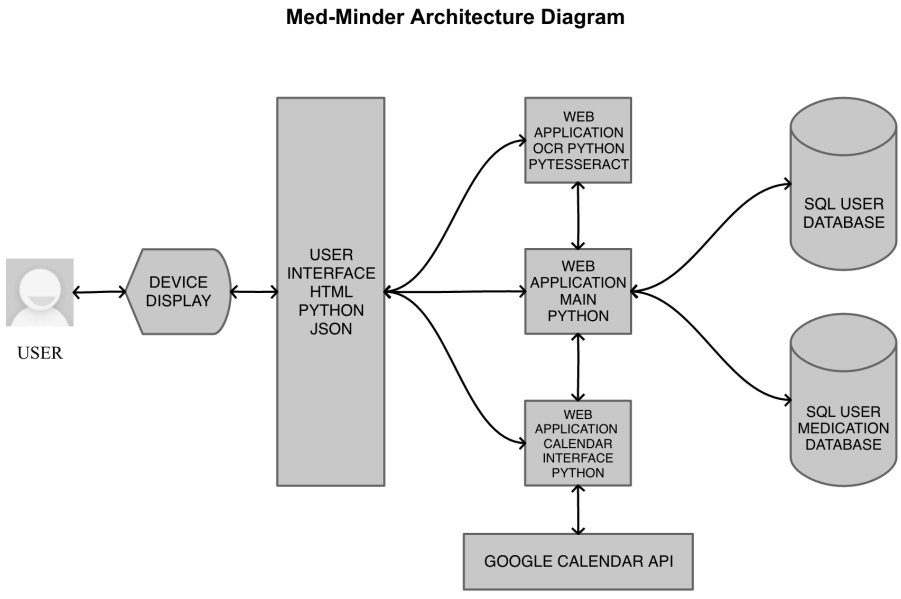

Figure 2: Architecture Diagram

3. Use Cases

Use Cases are described via use case diagrams followed by Use Case descriptions with scenarios.

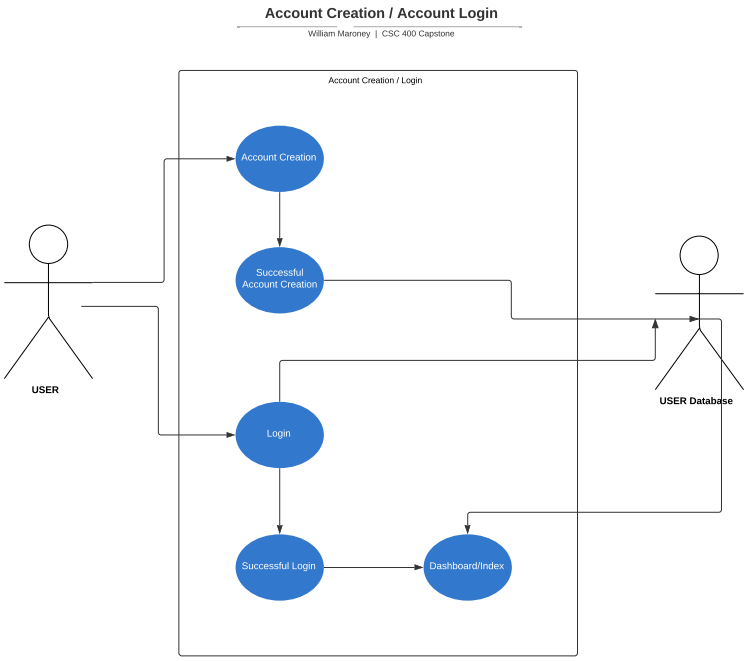

Figure 3: Account Creation/Login

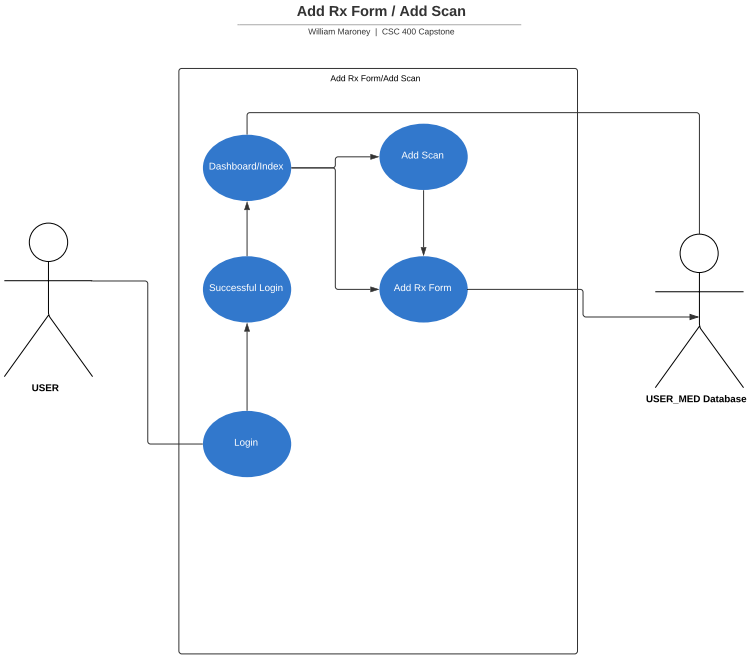

Figure 4: Add Rx Form / Add Scan

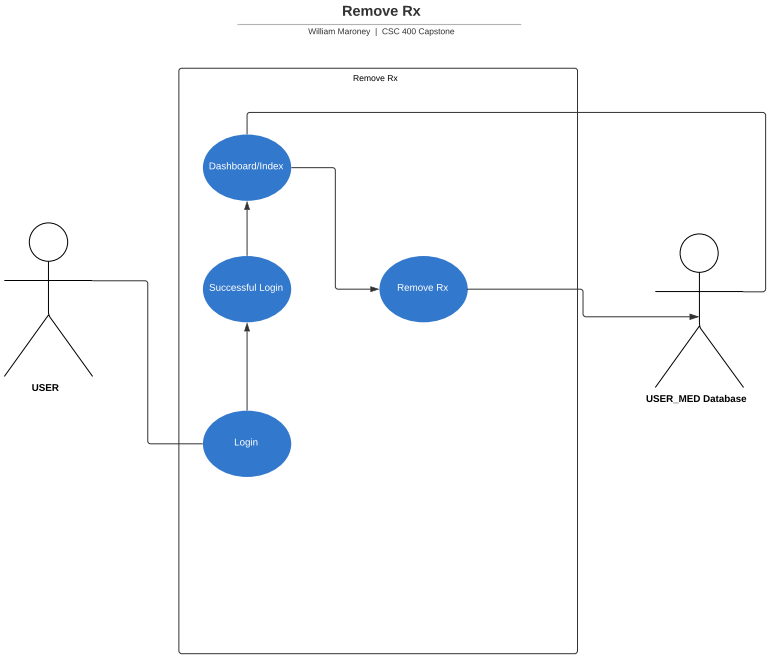

Figure 5: Remove Rx

Use Cases and Scenarios

1. **Create an account and Login.**

A potential user is required to create an account with username (email) and password.

Steps common to all scenarios:

User opens main web page and clicks sign up (Login for existing accounts)

User enters data into sign up form including username (Email) and password

After a User successfully logs into account, they will be brought to a dashboard page automatically which displays a calendar and their current medications

Scenario 1: Successful Login

Account successfully created with username (email)

Application displays dashboard (index.html)

Scenario 2: Unsuccessful Login

System notifies User of unsuccessful account creation with the following messages dependent on user circumstances.

User is redirected to login page

“Account Already Exists” and “Please Try Again” messages displayed to user

OR

“Unable to Create Account” and “Please Try Again” messages displayed to user

2. **Add Scan**

By default, users will arrive at the dashboard after logging in or creating an account. From the dashboard a user will select “Add Scan” to move to the add a scan page which prompts the user to allow the application to access the device camera. The user will select “capture” then “upload” to capture a screenshot of the prescription label then upload the picture for OCR and redirects the user the “Add Rx Form” page where the translated text is in a text field above the form.

Steps common to all scenarios:

User is brought to “Add Rx Form”

Once a user adds a new prescription an automatic email is sent reminding the user with the medication and frequency in the body of the message

Scenario 1: User selects “Add Rx Form”

User enters prescription information manually into form which includes Medication Name, Dosage, Frequency, Refills, Date Filled, Physician, and Pharmacy

Scenario 1a: Successful data entry

User enters information into form and clicks “submit”

Application adds prescription information to user’s database account

Application sends email reminder with prescription information in body of message

Scenario 2: User selects “Add Scan”

The user is brought to “Add Scan” page where they are prompted to allow the application to connect to their device camera. Once the user accepts the application opens a window camera frame with capture and upload buttons. The user will hold up the prescription information print out from the pharmacy and hit capture when the data is in the frame. Then the user will select upload which allows OCR to translate the image to text. The user is then redirected the “Add Rx Form” with an additional text field above the form with the translated text. The User can then copy and paste the information into the form.

Scenario 2a: Successful Scan

User allows application to connect to device camera

User captures image of prescription information

User uploads image for OCR translation

User is brought back to “Add Rx Form” page

Text Field is generated above the form with the translated text from the image

3. **Delete Prescription/Reminder**

Steps common to all scenarios:

User Logs into account

User selects “Remove Rx” from the dashboard

User selects the desired prescription from dropdown and clicks “Submit”

System removes prescription from calendar and user’s medication database

4. Structural Design

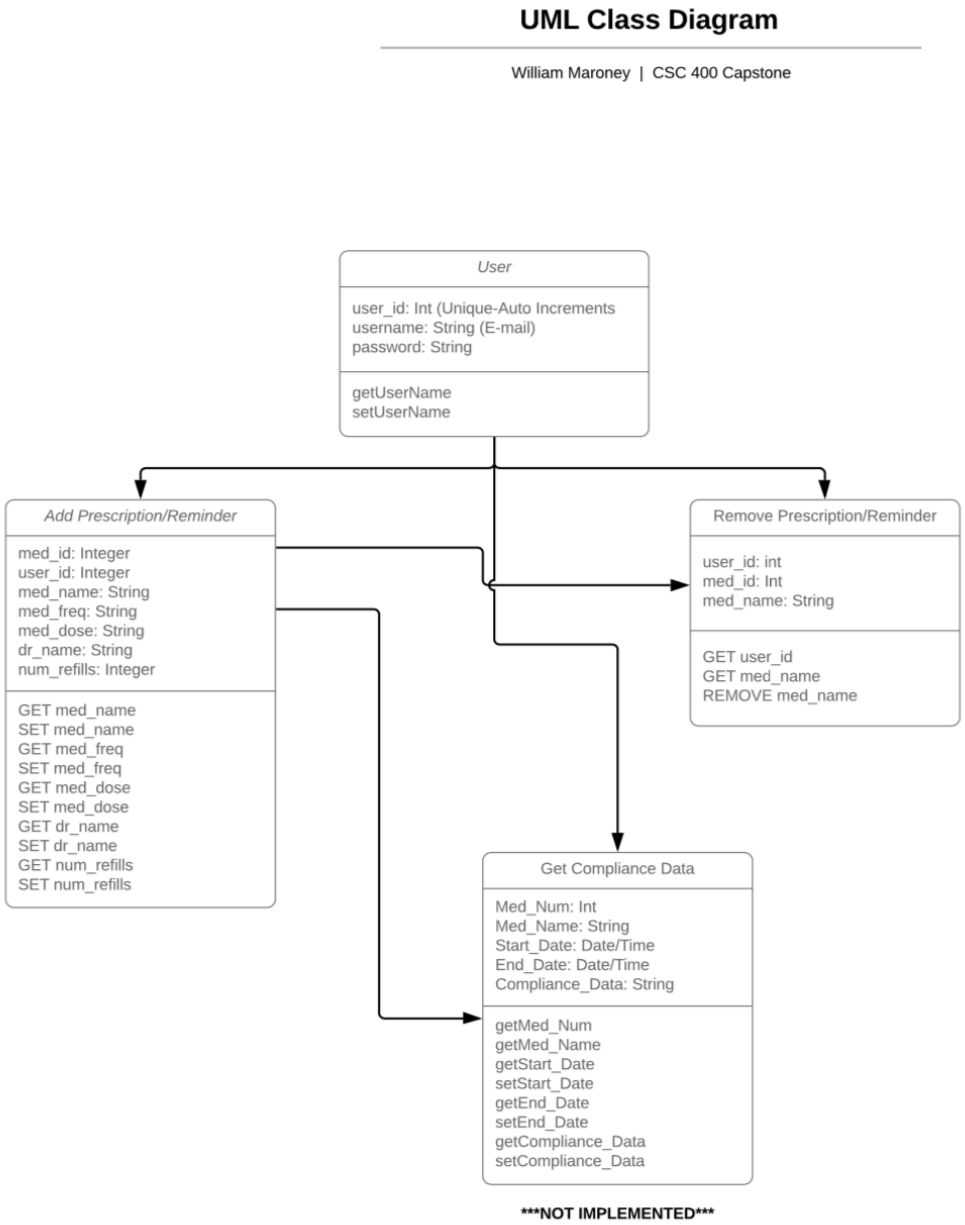

Figure 6: UML Class Diagram

5. Data

Med-Minder currently uses one database currently hosted and connected to the instance on google. The user database has two tables, one to hold usernames and passwords and the other to hold the medication information connected to each user. The user table stores the user information only including the user id, user email, and password. This table does not hold PHI or Protected Health Information and would not need encryption to maintain HIPPA compliance. The User's medication information is stored on the SQL instance connected to the application's Google VM instance. This table does hold PHI and needs encryption to maintain information security because data will be stored and reduce the data availability to data breaches or hacks. Currently, this is the only data Med-Minder uses although it was planned to include an additional database to compare medications for warnings and additional information.

Schema:

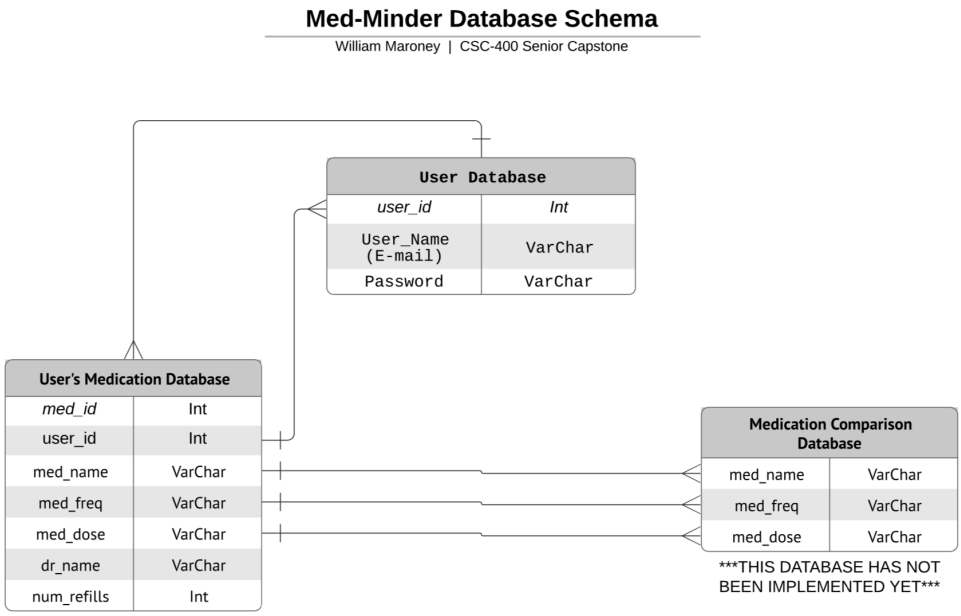

Figure 7: Database Schema

(Med Comparison Database not Implemented)

6. Implementation

The core algorithms in this application involve capturing the prescription text image and translating it to a string of text to populate the Add Rx Form automatically. Other algorithms involve querying the database, adding or removing items to/from the database, and sending the reminder email to the user. Lastly, other functions revolve around managing forms through flask forms and managing various features via their respective application routes.

1. Capturing the image of prescription text

When a user clicks the "Add Scan" button on the dashboard they are redirected to the addscan.html page. This page prompts the user to allow the application to access the camera on their device. Once the user accepts, the application opens a camera vision frame and the user can then hold up a prescription to capture and upload. The algorithms here involve some json code embedded into the html template. The captured image is saved and object character recognition (OCR) is called on the saved text jpeg file. The user is redirected to the "Add Rx Form" page and the translated text is displayed in a box above the form for the user to copy and paste.

```
{% extends "base.html" %}
{% import "bootstrap/wtf.html" as wtf %}

{% block title %}AddScan{% endblock %}

{% block page_content %}

<video id="player" controls autoplay></video>
<button id="capture">Capture</button>
<canvas id="canvas" width=320 height=240></canvas>

<form method="post" action="{{url_for('imgprocess')}}" name="imgform">
    <input id="inp_img" name="img" type="hidden" value="">
    <input id="bt_upload" type="submit" value="Upload">
</form>
```

Figure 8: JSON within AddScan HTML Template

ocr_simple.py

```
    # Read image path from command line
    imPath = img_path

    # Uncomment the line below to provide path to tesseract manually
    # pytesseract.pytesseract.tesseract_cmd = '/usr/bin/tesseract'

    # Define config parameters.
    # '-l eng'  for using the English language
    # '--oem 1' sets the OCR Engine Mode to LSTM only.
    #
    #  There are four OCR Engine Mode (oem) available
    #  0    Legacy engine only.
    #  1    Neural nets LSTM engine only.
    #  2    Legacy + LSTM engines.
    #  3    Default, based on what is available.
    #
    #  '--psm 3' sets the Page Segmentation Mode (psm) to auto.
    #  Other important psm modes will be discussed in a future post.
    # just in case ---NEEDED
    # tesseract image.jpg stdout -l eng --oem 1 --psm 3
    # output to terminal
    # tesseract image.jpg output -l eng --oem 1 --psm 3
    # output to text file "output.txt"

    config = ('-l eng --oem 1 --psm 3')

    # Read image from disk
    im = cv2.imread(imPath, cv2.IMREAD_COLOR)

    # Run tesseract OCR on image
    text = pytesseract.image_to_string(im, config=config)

    # Print recognized text
    print(text)
    return (text)
```

Figure 9: Object Character Recognition Engine (Pytesseract Library)

Object Character Recognition is implemented through pytesseract, a python library that uses artificial intelligence algorithms to detect text characters within an image. In the above figure there are some configuration parameters seen like what language the text is in and which OCR engine to run. Further information can be obtained by reviewing the pytesseract and tesseract library documentation available on the internet through a simple search.

2. Database Querying

Anytime a user adds a username to request access to the application or a user adds and removes prescriptions there are calls to the database. Also, within the main application code there are calls and queries to the database to display attributes. The database is coded in SQL and the main application in Python which requires some imported python libraries (pymysql and SQLite) to interface with the database and SQL code. The database has two tables each with primary keys (user_id and med_id) which are auto incremented to maintain database consistency and accuracy. Usernames are limited to emails specifically Gmail accounts to interface with Google Calendar API. The usermeds table references a foreign key (user_id) to connect the attributes of a user's medication to the specific user.

```
SET foreign_key_checks = 0;
drop table if exists users;
create table users (
    user_id integer auto_increment,
    user_email varchar(255)unique key not null,
    user_password varchar(255),
    PRIMARY KEY (user_id)
);

drop table if exists usermeds;
create table usermeds (
    med_id integer auto_increment,
    user_id integer,
    med_name varchar(255),
    med_freq varchar(255),
    med_dose varchar(255),
    dr_name varchar(255),
    refill_date varchar(255),
    num_refills integer,
    PRIMARY KEY (med-id),
    FOREIGN KEY fk_user(user_id) REFERENCES users(user_id)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
);
SET foreign_key_checks = 1;
```

Figure 10: Database SQL Code

3. Sending the User an Automatic Reminder Email

Whenever a user adds a new prescription to their account the application automatically sends out an email reminding the user to take the medication. This email includes the medication name and frequency which involves some queries to the database to capture the current user's medication and frequency which was just added to the database via the form. This feature was added to maintain integrity of the design and proof of concept. The original design feature was to populate the user's Google calendar with reminders and given times, however complications with calendar implementation led to a work-around process.

```
144
145 def reminderEmail(user_id, medname, freq):
146     fromaddr = "medmindercsc400@gmail.com"
147     toaddr = user_id
148     Medication.id = medname
149     Medication.frequency = freq
150     body = "Please Remember to Take Your Medication " + str(Medication.id) + " " + str(Medication.frequency) + " per
            day"
151     msg = MIMEText(body, 'html')
152     msg['From'] = fromaddr
153     msg['To'] = toaddr
154     msg['Subject'] = "Medication Reminder "
155     server = smtplib.SMTP('smtp.gmail.com:587')
156     server.starttls()
157     server.login("medmindercsc400@gmail.com", "Atmose@123")
158     text = msg.as_string()
159     server.sendmail(fromaddr,[toaddr],text)
160     server.quit()
161     print("EMail sent!")
162
```

Figure 11: defining the Reminder Email Function

4. Form Management using Flask Forms

All forms including login, account creation, and adding prescriptions involve Flask for form management. This was used for ease of application development and capturing user entered data. Flask forms libraries were imported into the application and called during form creations as seen in Figure 12 below.

```
49 # Add Rx Manual Form
50 class AddManualForm(FlaskForm):
51     med_name = StringField('Medication', validators=[DataRequired()])
52     dosage = IntegerField('Dosage', validators=[DataRequired()])
53     frequency = StringField('Frequency', validators=[DataRequired()])
54     refills = IntegerField('Refills', validators=[DataRequired()])
55     dateFill = DateField('Date Filled', validators=[])
56     doc = StringField('Physician', validators=[DataRequired()])
57     pharm = StringField('Pharmacy', validators=[DataRequired()])
58     submit =  SubmitField('Submit: ')
59
```

Figure 12: Flask Form Implementation

5. Application Route Management

Various application routes are managed via the main python code at the bottom section of main.py. Application routes are defined and then various functions and features within the route are implemented. Application routes reference respective HTML templates kept within the templates file folder in the project repository. Routes include login, signup, index, add scan, add rx manual, remove rx, and about pages. Figure 13 shows the application routes for the index and login pages of the application.

```
@app.route('/', methods=['GET', 'POST'])
@app.route('/index', methods=['GET', 'POST'])
@login_required
def index():
    form = ViewRx()
    load_meds()
    db = pymysql.connect(host='35.229.79.169', user='root', password='password', db='med_minder')
    c = db.cursor()
    c.execute('SELECT user_id FROM users WHERE user_email="{}"'.format(current_user.id))
    user_id = c.fetchall()[0][0]
    c.execute('SELECT med_id, med_name from usermeds WHERE user_id = {}'.format(user_id))
    meds=c.fetchall()
    form.med_name.choices = meds
    return render_template ('index.html', form=form, meds=med_db.values())

@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    # display the login form
    form = LoginForm()
    if form.validate_on_submit():
        user = form.username.data
        # validate user
        valid_password = user_db[form.username.data]
        if user is None or not valid_password:
            print('Invalid username or password', file=sys.stderr)
            redirect(url_for('index'))
        else:
            # print(user)
            login_user(User (user, valid_password))
            # print(user.is_authenticated)
            flash('Login Successful', category='success')
            return redirect(url_for('index'))

    return render_template('login.html', title='Sign In', form=form)
```

Figure 13: Application Routes for Index and Login

7. User Interface Design & User Guide

User Interface Design

The following User Interface (UI) designs are purposely kept simple with focus on functionality over design. Design improvements and finalizations took place towards the end of the project during the last phase or sprint. Interface-metaphors, calls-to-action, affordances and other design heuristics were added after basic functionality was achieved.

User Guide:

Figures 14 – 19 show a basic user guide to the Med-Minder application in its current stage of development. Each figure has some basic descriptive instructions following the image.

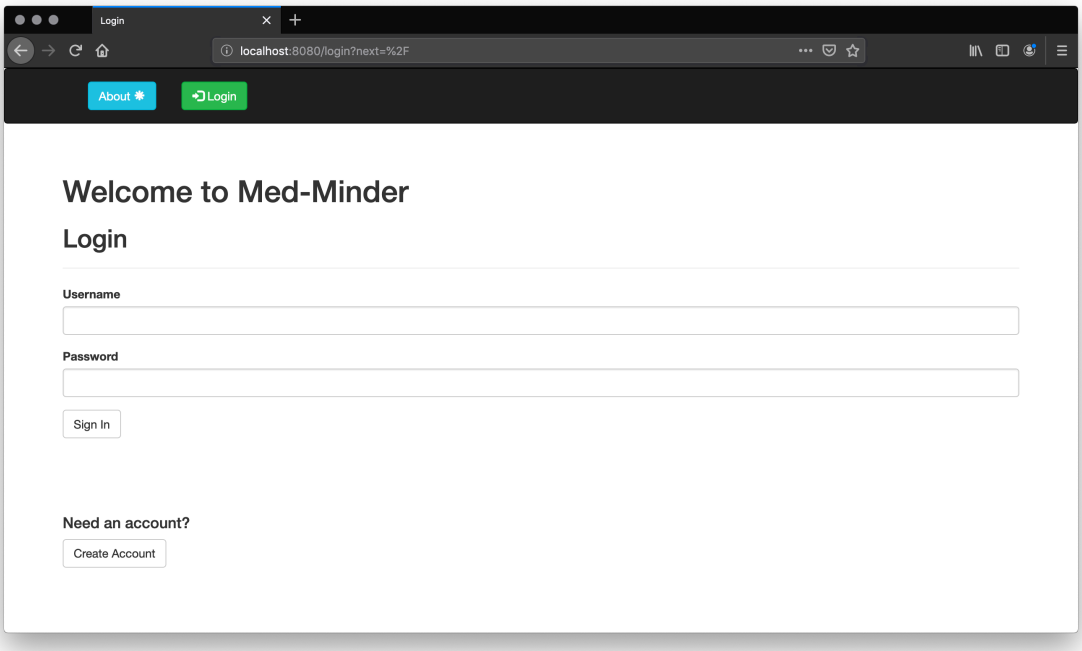

Figure 14: Login and Create Account

The user is brought to the Login page automatically upon accessing the IP address or local host address depending on if the project is currently hosted or not. If the project is hosted the user will access http://35.237.157.161:8080/ in their browser and if the project is not hosted access

http://localhost:8080/ to access the project. If the project is not currently hosted the original source code will need to be downloaded from the Github repository and then run main.py through a terminal window. Further instructions are documented within the README file.

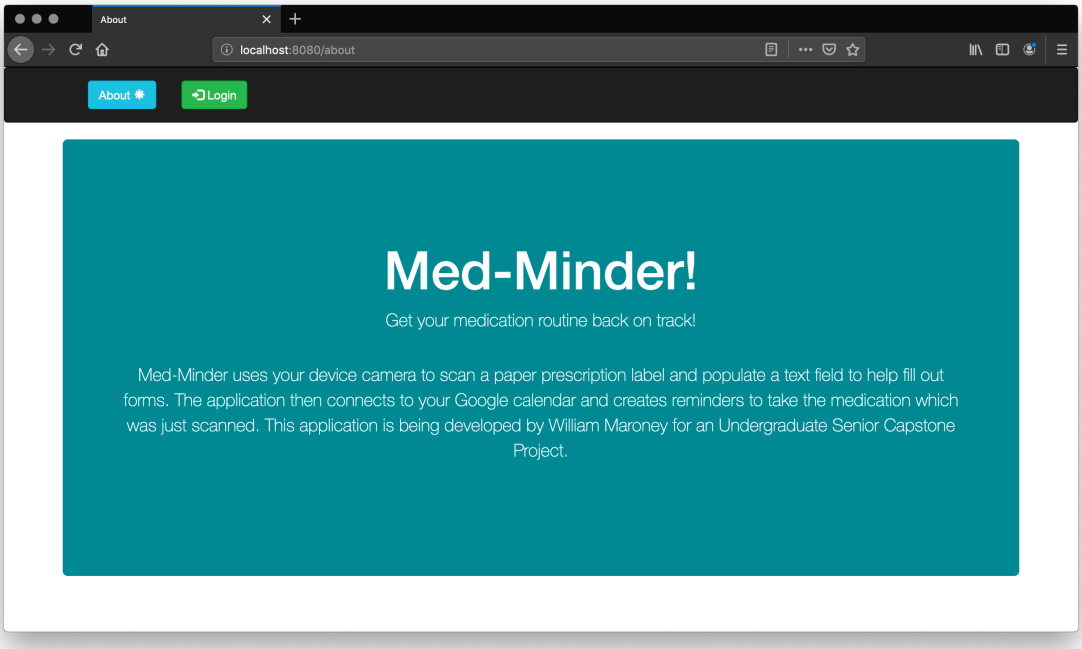

Figure 15: About Page

The about page features some basic information about the application and its development and myself, the developer. This page would continually get updated to reflect the current stage of development and design.

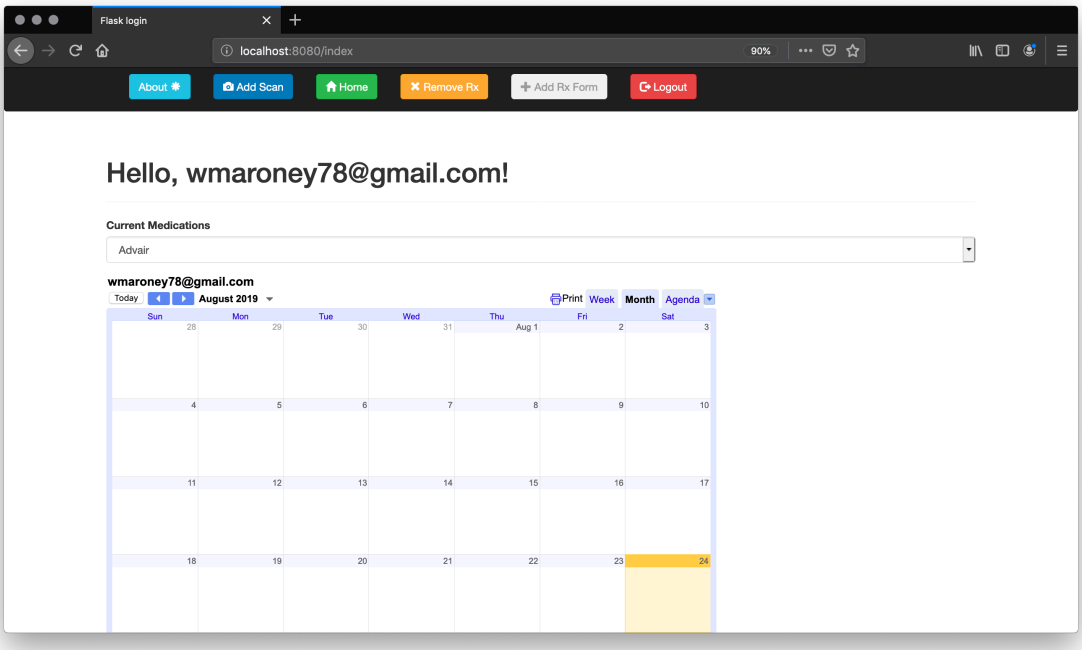

Figure 16: Main Dashboard Index Page

The main index page of the application is a dashboard which features buttons to direct the user depending on their desired action. The main dashboard features buttons for About, Add Scan, Home, Remove Rx, Add Rx Form, and Logout all with respective functions and web route redirects.

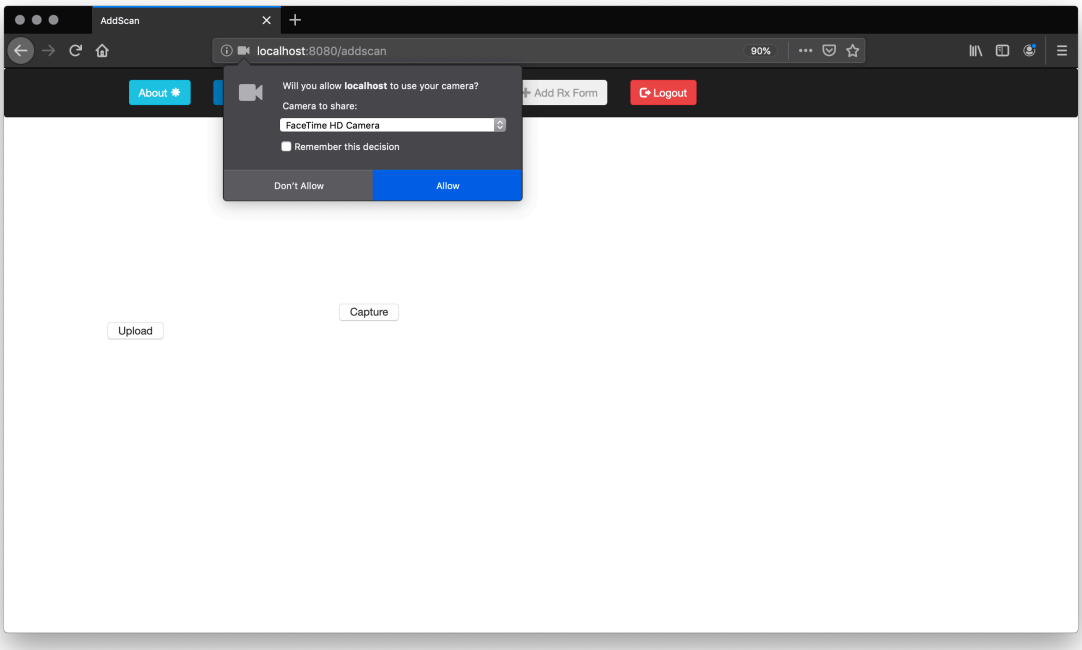

Figure 17: Adding a Scan

When the user selects “Add Scan” from the dashboard navigation bar they are prompted to allow the application to access their device camera. The user then selects capture and upload when the prescription paperwork is lined up for the camera to capture. The application then redirects the user to the “Add Rx Form” page and the translated text is located in a text box above the form.

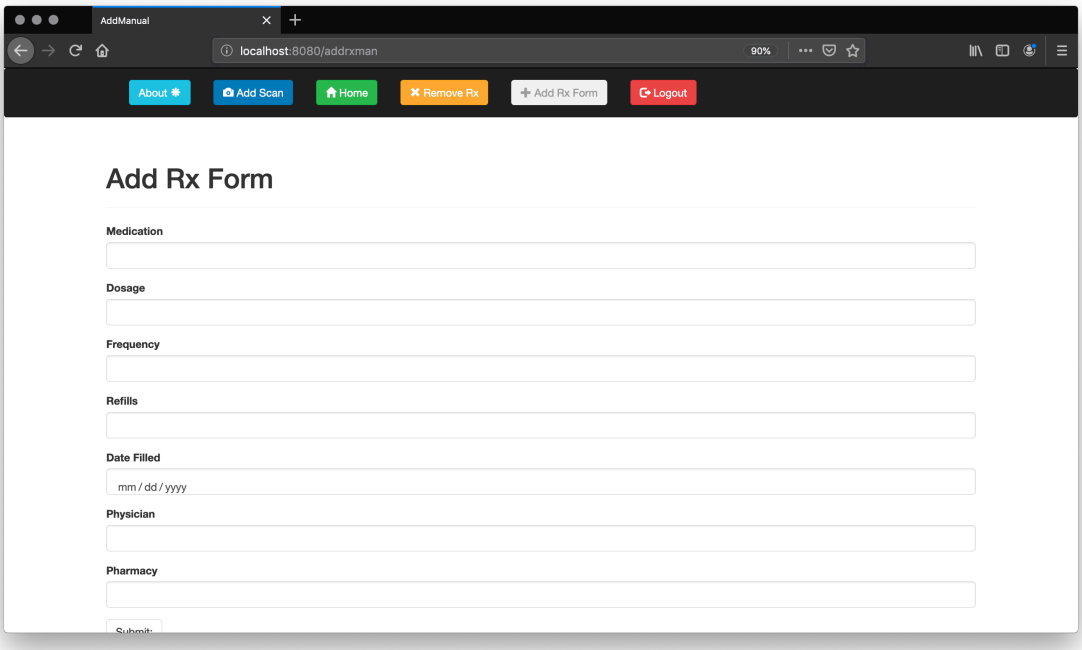

Figure 18: Add Rx Form

The user can either select the “Add Rx Form” via the button or they are brought there automatically after scanning a new prescription image into the application. From this page, the user can enter the medication information manually or copy and paste the data if they scanned an image. This feature would be updated for version 2 to include automatically populating the fields with the correct data given a better OCR accuracy.

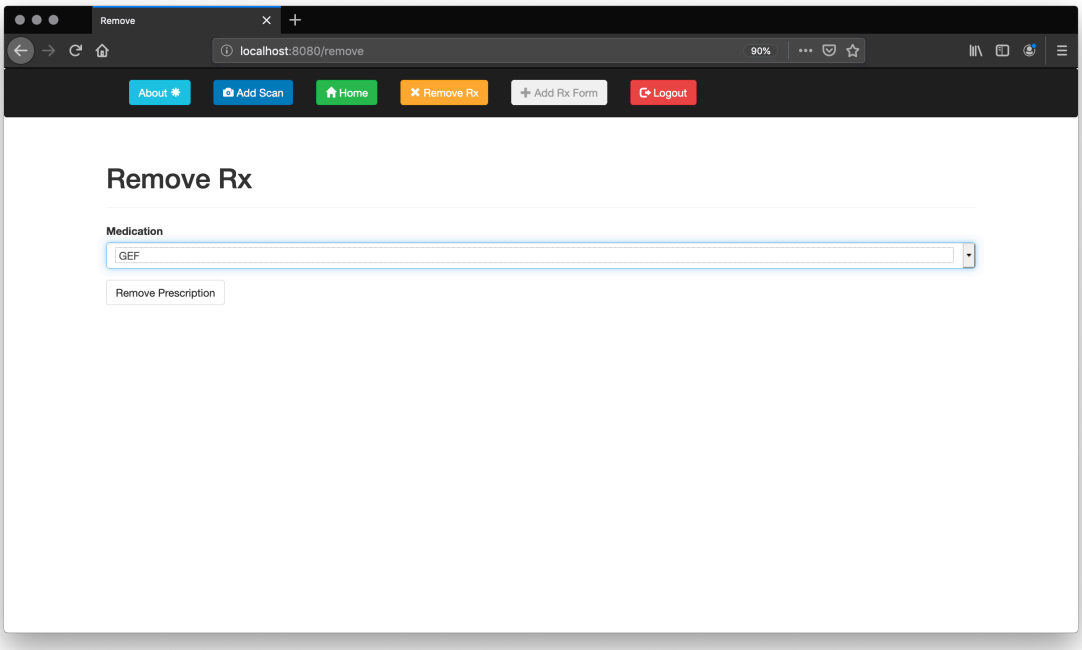

Figure 19: Remove Rx

The user can select to remove a prescription from their account via a dropdown of their current medications. After the medication is selected the user clicks remove prescription and the selection is deleted from the database and the user is redirected to the home page of the application.

8. State of the Implementation

Feature 1: Account Creation

This feature is fully operational, and a user can create an account (gmail account) and password to access the application. Version 2 will include user verification and authentication with encryption added to passwords to maintain security while storing sensitive data.

Feature 2: Login

This feature like account creation is fully operational but would need some improvements to security like password encryption.

Feature 3: Add Prescription Manually

This feature is fully functional, and the user can add the prescription information to their account via the Flask Form. Once a user adds the data and clicks submit the information is added to the database under the user's id. The user is then redirected to the home page and simultaneously an email is sent to the user's email address with a reminder to take the medication and the frequency.

Feature 4: Add Prescription via Scanned Image

This feature is fully functional with the scanned image being saved with OCR run on the image immediately upon the user uploading the image. The user is redirected to the "Add Rx Form" with the translated text in a text field above the form. The user can copy and paste the data into the form. Version 2 would include the data being automatically populated into the fields but this would require better accuracy from the OCR and several if/else statements within a for loop to iterate over the words and select the appropriate data.

Feature 5: Automatic Reminder Email

When a user adds a prescription either via scanned image or entering the data manually, both redirect the user the "Add Rx Form" page. When the user clicks submit and the data is added to the database, the user is redirected to the home page. Simultaneously the user is sent an email from Med-Minder with a reminder to take their medication with the appropriate frequency. This feature was implemented as a work-around to maintain scope of design and concept.

Feature 6: Automatic Calendar Reminder

This feature is not functioning although the actual create reminder function is built, it has not been tested or debugged due to continual errors implementing and interfacing with Google Calendar API. The API documentation describes the API in a way that appears to be simple and easy to implement creating events (reminders). Implementation is not as simple as was expected and requires additional time to be fully functional in a subsequent version of the application.

Feature 7: Remove Prescription

This feature is fully complete. Once a user logs into the application they can select “Remove Rx” from the button navigation bar and they’re redirected the Remove Rx page. From this page, the user can select the medication from a dropdown of their current medications. Once the user selects the desired medication, then they click remove medication and the med is removed from their account in the database. Version 2 would feature a confirmation and would prompt the user to confirm the deletion before actually deleting the medication.

Feature 8: Get Compliance Information

This feature was not implemented and was difficult to logically derive without the Calendar portion functioning. The feature would be implemented in a subsequent version of the application after the Google Calendar and automatic reminders were implemented, tested, and debugged.

Feature 9: Medication Comparison Database

This feature was also not implemented and was tabled to version 2 early on in the project implementation due to unforeseen issues with the calendar implementation. This feature was considered optional or additional from the projects proposal and was the first feature to be pushed into version 2.

9. Testing and Evaluation

The project was tested throughout development and was tested as well as tested on the instance. Once features were operational the application was tested on the Google Cloud VM instance. Each time a new feature was implemented, or a change was made to the source code, a test was run to verify and validate functionality. During the entire project development, the CodeRunner application was used to write and modify various code and SQLWorkbench was used to verify and validate the database entries. Google Cloud VM was used to host the project and SQL instance and these were used during the testing and evaluation of the project. A Trello board was created to manage various project tasks and set tasks to a due date to maintain project goals. A Github repository was created to hold and host the code and template files.

10. Reflection

The project development process went well and generally the original design proposal was followed with most goals being met. Some of the project's initial features were a bit difficult to implement and could've used some more time and more research before full functionality was achieved. This project would typically be developed by a team of people each working on different aspects and parts of the application. As with any development project some features need reworking and re-thinking and this project was no exception. Interface design was kept simple favoring functionality over elaborately designed webpages. File storage was kept simple with improvements anticipated and planned for version 2. Form creation and management was relatively simple with Flask and Bootstrap. The Calendar portion was originally expected to be the easier portion of implementation with OCR being more difficult. However, the opposite of this was observed during project development meaning the Calendar API was extremely difficult to work with and the OCR portion was insanely easy to implement.

11. Version 2 Feature & Improvements

There are a few version 2 feature improvements to implement including fixing the Google Calendar API interface, properly authenticating users and encrypting passwords, and auto populating the add prescription form with the correct prescription data.

1. Google Calendar & Reminder Generation

Currently the calendar portion is not implemented with a work-around process implemented where the user gets a reminder email when the add a new prescription to their account.

2. User Authentication & Password Encryption

Version 2 would include proper user authentication and password encryption to maintain security of user sensitive data. This improvement would also include improvements to storage like PHI or protected health information which is essential in maintaining HIPPA compliance.

3. Auto Populating Add Rx Form with Scan Data

Improvements can be made to the OCR and scanning portion of the project by having the scanned text automatically populate the “Add Rx Form”. Current implementation has the translated text in a free text box above the form allowing the user to copy and paste. This feature would be the first feature to improve for version 2 after the calendar reminder portion is implemented.

12. Professional Development & Lifelong Learning Requirement

Before starting this project, I had some experience in web and database design but not to the level where I developed them fully without help or direction from professors or references. This project helped me solidify some of my coding skills in SQL and HTML as well with Python and JSON. I’ve worked with some Object Character Recognition in previous coursework but had not used the pytesseract library and algorithms to achieve OCR on text. Additional experience included using GitHub and Trello to organize the code and tasks respectively. I used the CodeRunner application to write all of my code and SQLWorkbench to verify the database entries. The project is hosted on a Google Cloud Virtual Machine instance with the SQL instance connected, both of which were discussed in previous coursework. The project initially proposed interfacing with Google Calendar and Vision API’s which took a lot of independent research as API interfacing was not discussed in any previous coursework.

Most of the research for this project was done via the internet and searches. This includes API documentation, Python library documentation, examples of implementation, stack-overflow, and additional coding resources. The research was mostly done during the first stage of the project development with some additional research obtained as needed throughout the duration of the project.

While learning about Google’s Calendar API, I was also researching Vision API and OCR alternatives. During this time, I discovered Tesseract and Pytesseract both of which are Python OCR libraries. In order to keep costs down on the Google Cloud instance, it was decided to implement these OCR libraries over Google’s Vision API. The calendar API unfortunately turned out to be quite a bit more complex than initially expected and caused a mild delay in development of some additional features. As part of lifelong learning, I would like to continue to

work towards implementing this portion of the project to gain further experience with API interfacing. The software design process was revisited several times during the development to maintain the overall design process. This includes identifying a “need” the software would fill, proposing how the software can meet the “need” identified, preparing a design document which details how the application will meet those needs, and finally the development of the application itself. During the course of this project development, those attributes were reviewed frequently to maintain consistency of design and proof of concept. As with any application or software design, additional improvements and feature rethinking/reworking continues to evolve as the software itself evolves. This application also evolved as features were implemented, modified, tested, debugged, reimplemented, and so on. Now that I fully understand the software design and development process and how much detail and work goes into a fully functioning application, I can certainly appreciate the role of the development team and what it takes to produce a usable and valuable product.

Link to Med-Minder GitHub Repository:
https://github.com/WMaroney/Med-Minder