# DEVELOPMENT AND IMPLEMENTATION OF A SMART PILLBOX SYSTEM: INTEGRATING ESP, FLASK SERVER AND FLUTTERFLOW APP FOR ENHANCED MEDICATION ADHERENCE

Warren Zhang[1], Soroush Mirzaee[2]

[1]San Marino High School, 2701 Huntington Dr, San Marino, CA 91108
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

***ABSTRACT***

*The issue I am aiming to solve is the timely intake of medication by individuals, which is crucial for their health and well-being [1]. I believe that the best way to solve such a problem is to create a device and app to remind the user toconsume their pills at a time that they schedule. The different technologies and components of my project are the physical pillbox, ESP, flask server, and app. The ESP is a small computer with a screen that is placed inside the pillbox which has different functions such as pill reminder, which the user sets in how many hours they want to take their pill, pill count, where the user sets the number of each pill that they have to the corresponding pill [2]. The appwas made through FlutterFlow and has a lot of different functionalities such as the creation of a pill schedule for eachone of the user's pills, a history page which displays the times and the pills that the user has consumed and has different unique functions in the settings page [3]. The flask server is being hosted through render and in the server itregisters the user's ID in the app to an ESP after the user has scanned the QR code that is displayed on the ESP, therefore, the information of the ESP and app are being shared [4]. There were a lot of technical difficulties in creating the pill schedule as it was an overall tedious process that involved multiple steps and functions. This problem was fixed by spending a lot of time working on it. It was also difficult to get the 3d printed pillbox to perfection as the ESPrequires specific modeling in order to fit which required more than 20 prints of the pillbox. Overall, my product maynot be flawless, but it covers all potential problems and human error that may occur.*

***KEYWORDS***

*Hardware, FlutterFlow, Server, Medicine*

## 1. INTRODUCTION

The problem I am trying to solve is consistency in consuming medicine ensuring the user's health.I have first-handedly noticed the importance of pill consumption and the damaging effects if a person's prescribed pills aren't taken in a timely manner [5]. The background of this problem stems from Grandma who struggles with consuming pills at the correct time without the help of others. My problem is important because it leads directly to the user's health. Without health, nothing in life is important, so ensuring that one's health doesn't rapidly decline is a very essential task. In addition, I know that this problem stems not only from Grandma, but to all seniors who have the

David C. Wyld et al. (Eds): DMSE, CSEIT, NeTCoM, SPM, CIoT, NCS, NLPD, ArIT, CMLA – 2024
pp. 83-93, 2024. CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141407

same issue with their medicine. Overall the problem usually affects older people, but may also affect younger medicine consumers. Sometimes, to remember to consume one's medicine they just need a small reminder, which is what my product was made for. A statistical example would be that my grandma, on average, forgets to take her medicine two days out of the week. This means that she doesn't consume around twelve pills every week which is very detrimental to her health.

In the first methodology it tries to show the user the designated pills that they need to take at a certain time, but ignores human error as well as the price of such a product is impractical to the average family. In order for this project to be well functioning it needs the full compliance of the user and disregards the chance that the user may forget to check their designated pills. In the second methodology, it is a raw concept as the idea to have audio reminders is good, but it is completely unrealistic as the cost of leaving voicemails over a long period of time will be substantial. The final methodology checks when a person has opened the cap to one of their pill bottles, tracking if the user has taken their pill. This is not a proactive concept as the user isn't actively reminded to take their pills; rather the product is more of an observatory study of the user's pill consumption patterns.

A method of solving this problem would be a smart pillbox [6]. This smart pillbox would send notifications to the user's device as well as showing a notification on a screen in the pillbox. This provides an easy and efficient solution to this problem as it reminds users what pills they need to take and at what time. This is an effective solution as it reminds the user both on their phone and on the pillbox itself 30 minutes prior to the scheduled time. I think it is better than having a caretaker or family member remind you when to take your pills as it is not time efficient and there will be human error as you cannot be completely reliant on someone else, but you can completely rely on this smart pillbox to notify you 100 percent of the time.

Another point worth noting is that the pillbox keeps track of the number of each medicine the user has so they can request refills for their medicine before they run out.

In Section 4, we conducted two experiments to test different aspects of our system. The first experiment focused on the notification system's performance, specifically analyzing the invocations for sending push notifications and checking pill reminders. The setup involved monitoring the number of invocations per day over a month. The most significant finding was the consistent invocation rate, indicating reliable scheduling and uniform user interaction.

The second experiment aimed to improve the Flask server's accuracy in handling requests. This was tested by tracking the accuracy over 68 commits, addressing issues like timeouts and communication problems. The setup involved iterative testing and refinement with each commit. The key finding was the steady increase in accuracy, reaching 98% by the final commits. This improvement resulted from systematic debugging and optimization efforts.

Both experiments demonstrated the importance of consistent testing and iterative improvements in achieving reliable and efficient system performance.

## 2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

### 2.1. Maintaining a Balance

One of the main challenges of this project was maintaining a balance between the workload of the hardware and the phone app. Most of the key features could have been handled with the phone app

by itself, but putting some of those features on the ESP, which is a standalone device, differs the user's experience and adds a second level of assurance. This could potentially cause problems with the user experience. Having only the mobile app would have made the project just like every other reminder app that gets forgotten after a few weeks of usage. However, the project has two major components that will differentiate the way users interact with the device.

### 2.2. The Future Pill Schedule

Another one of the main challenges of this project was setting up the future pill schedule for the user as well as being able to show the history of the pills that the user has taken. Some potential problems were that the time would not be displayed correctly and that we may not know when the user has taken their pill. The way that I could resolve this problem is by removing the current time that the user should have taken the pill and setting the array of future times to the number of pills that the user has so that it would always be tracked if the user had already consumed all of their scheduled pills.

### 2.3. Making the Actual Physical Pillbox

Another one of the main challenges of this project was making the actual physical pillbox. Some potential problems were the size and shape of the pillbox. The pillbox needed enough space for the ESP and the lithium-ion battery to fit in together while not having an impractical sized pillbox. The way I could resolve these potential problems would be to resize the pillbox altogether to make the squares more proportional to each other or by letting the ESP buttons and screen stick out of the pillbox.

## 3. SOLUTION

The three main components of the project are the Mobile app, the Physical Pillbox, and the Server [7]. Three components connect to each other using the firestore database and will allow the user to regulate and track when they consume their medicine. Furthermore, the user has to set up the medicine they take with their name, the number of pills they have, and the last time they took that medicine, so the system would generate a schedule for the pills they need to take every day. On the other hand, the physical pillbox uses an ESP32 Feather as its main board and its responsibility is to remind the user to take their pills by showing notifications on it's screen [8]. It also records the time you take a medicine and adds it to the database through a function defined in the server. The ESP is running on Circuit Python to ensure the maximum efficiency and scalability of the code.

Additionally, Flutterflow was used for the development of the mobile app with custom functions to make the interaction with the server easier and more robust. Some of the key features of the app include:

- Ability to change the list of pills based on needs
- Ability to see all of the history of the taken medicine
- Ability to see times for future pills

Moreover, for the server, Flask and Python were used

The three main components are the ESP, Flask Server, and Mobile app. To start off, users are required to download the MedSync mobile app, sign up, and receive a physical pill box with an esp attached in it. Next, users scan the QR code, which is automatically generated from the ESP,

to connect their pillbox with their account on the app [9]. The users may edit and add pills from the app and the users receive reminders from both the ESP and the app. All of this was constructed from a combination of dart and python.

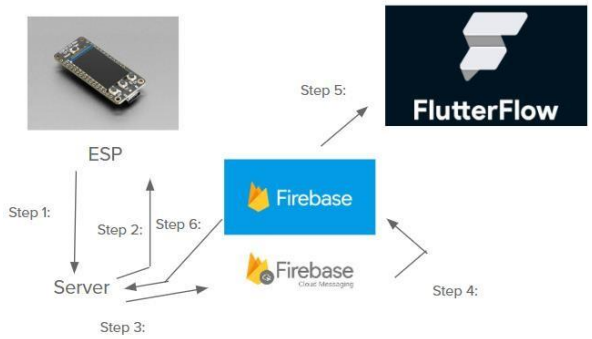



Figure 1. Overview of the solution

Our ESP's purpose is to serve as a physical screen and reminds the user to consume their pills. We used circuit python to implement the project. The ESP brings a physical aspect outside of the notifications received from the app. It is connected to the App, firebase, and flask server. It receives and transmits data from the user's pills.

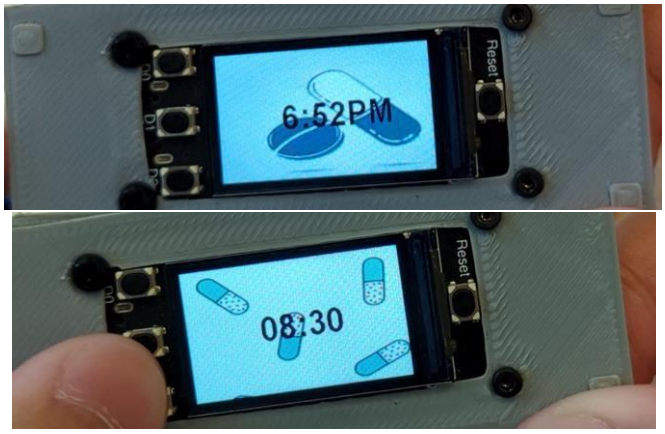
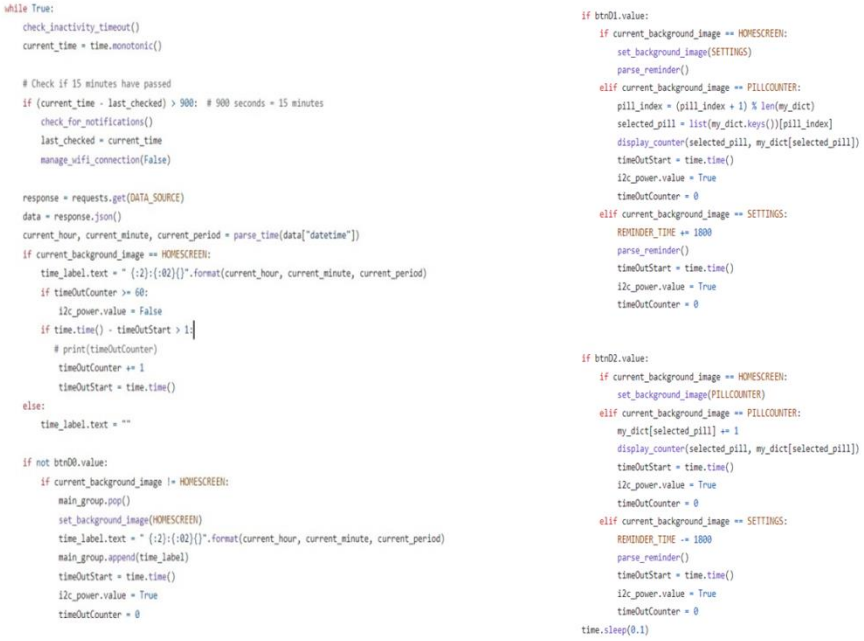



Figure 2. The physical screen



```
while True:
    check_inactivity_timeout()
    current_time = time.monotonic()

    # Check if 15 minutes have passed
    if (current_time - last_checked) > 900:  # 900 seconds = 15 minutes
        check_for_notifications()
        last_checked = current_time
        manage_wifi_connection(False)

    response = requests.get(DATA_SOURCE)
    data = response.json()
    current_hour, current_minute, current_period = parse_time(data["datetime"])
    if current_background_image == HOMESCREEN:
        time_label.text = " {:2}:{:02}{}".format(current_hour, current_minute, current_period)
        if timeOutCounter >= 60:
            i2c_power.value = False
        if time.time() - timeOutStart > 1:
            # print(timeOutCounter)
            timeOutCounter += 1
            timeOutStart = time.time()
    else:
        time_label.text = ""

    if not btnD0.value:
        if current_background_image != HOMESCREEN:
            main_group.pop()
            set_background_image(HOMESCREEN)
            time_label.text = " {:2}:{:02}{}".format(current_hour, current_minute, current_period)
            main_group.append(time_label)
            timeOutStart = time.time()
            i2c_power.value = True
            timeOutCounter = 0
```

```
    if btnD1.value:
        if current_background_image == HOMESCREEN:
            set_background_image(SETTINGS)
            parse_reminder()
        elif current_background_image == PILLCOUNTER:
            pill_index = (pill_index + 1) % len(my_dict)
            selected_pill = list(my_dict.keys())[pill_index]
            display_counter(selected_pill, my_dict[selected_pill])
            timeOutStart = time.time()
            i2c_power.value = True
            timeOutCounter = 0
        elif current_background_image == SETTINGS:
            REMINDER_TIME += 1800
            parse_reminder()
            timeOutStart = time.time()
            i2c_power.value = True
            timeOutCounter = 0

    if btnD2.value:
        if current_background_image == HOMESCREEN:
            set_background_image(PILLCOUNTER)
        elif current_background_image == PILLCOUNTER:
            my_dict[selected_pill] += 1
            display_counter(selected_pill, my_dict[selected_pill])
            timeOutStart = time.time()
            i2c_power.value = True
            timeOutCounter = 0
        elif current_background_image == SETTINGS:
            REMINDER_TIME -= 1800
            parse_reminder()
            timeOutStart = time.time()
            i2c_power.value = True
            timeOutCounter = 0
    time.sleep(0.1)
```

Figure 3. Screenshot of code 1

The screenshot displays the while loop in the ESP's code that allows the user to manipulate the different screens and functions of the ESP. In addition, the code also allows the current time to be displayed on the homescreen of the ESP. This code consistently runs as long as the ESP is on and if the ESP is on for a certain amount of time it will eventually be turned on sleep mode. The function sets the screen to the Home Screen automatically and then changes based on the buttons being clicked by the user which will change to either pill count or reminder based on the button pressed. All variables that are used are in different libraries which we use to display the time as well as getting the button values. The ESP is also communicating with a Flask Server by sending data of the user's pills through a function which contains a dictionary with the pill name, count, and time [10]. The server stores the dictionary and sends it to Firebase where the data of the app is also stored.

The mobile app's purpose is to send notifications to the user as well as sort information on each user's pills. In addition, the app also generates a schedule of the pills that the user has while displaying the pills that need to be consumed on that day and at what time.

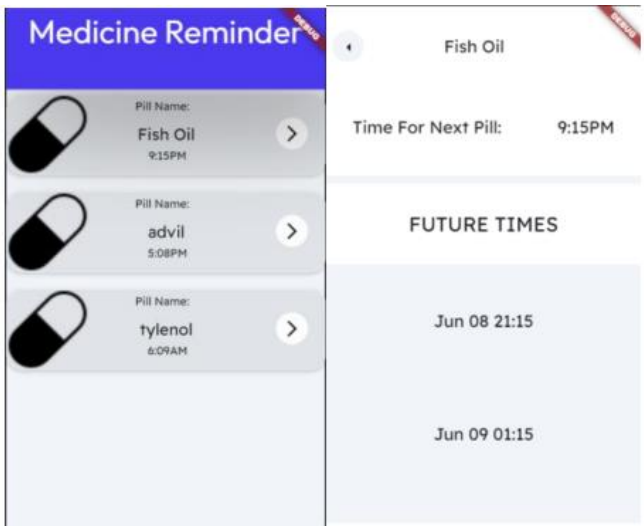



Figure 4. Screenshot of the medicine reminder

```
String reminderToString(
  int reminder,
  int medHist,
) {
  /// MODIFY CODE ONLY BELOW THIS LINE

  reminder *= 3600;
  int finalTime = ((reminder + medHist) * 1000);
  DateTime dateTime = DateTime.fromMillisecondsSinceEpoch(finalTime);
  DateFormat formatter = DateFormat('h:mma');
  return formatter.format(dateTime).toUpperCase();

  /// MODIFY CODE ONLY ABOVE THIS LINE
}
```

```
List<String> createPillSchedule(
  int reminder,
  List<int> medHistory,
  int pillCount,
) {
  /// MODIFY CODE ONLY BELOW THIS LINE

  List<String> pillEvents = [];
  int nextPill = medHistory[0];
  for (int i = 0; i < pillCount; i++) {
    nextPill += reminder;
    pillEvents.add(DateFormat('MMM dd HH:mm')
        .format(DateTime.fromMillisecondsSinceEpoch(nextPill * 1000)));
  }
  return pillEvents;

  /// MODIFY CODE ONLY ABOVE THIS LINE
}
```

Figure 5. Screenshot of code 2

In both of these screenshots it shows methods that are used for setting up the pill times and these methods run whenever the user creates a new pill and schedules the time for the next time that he would like to take that pill. In the first screenshot, it is the reminder string function which adds the amount of unix seconds that the user wants to take the pill next from the last time they consumed that pill and converts it from Unix to standard time formatting. In the second screenshot, the code creates a list of strings that adds the date and time of the next time the user needs to take a certain pill and sorts them for the future pills. A variable being made are the pillEvents which is a string list that contains the future scheduled times. Another variable being made is the formatter variable which is the next time that the user has to take the pill. It is not communicating with a backend server as it only sends information to another page which is outside of the home screen which displays the future times of the pills.

The flask server was coded through python and enables information to be linked from the app and the ESP. It helps the program run smoothly and is a very important part of the program. Without it, the communication between the ESP and the app wouldn't be as level.

```python
@app.route('/register_device/<device_id>', methods=["GET"])
def register_device(device_id):
    print("image requested")
    png_filename = f"{device_id}.png"
    bmp_filename = f"{device_id}.bmp"
    qrcode = segno.make_qr(device_id)
    qrcode.save(png_filename, scale = 5)
    convert_to_bmp(png_filename, bmp_filename)
    #user_ref = db.collections('devices')
    print("File generated!")
    return send_file(bmp_filename, mimetype='image/bmp')

@app.route('/isRegistered/<device_id>', methods=["GET"])
def isRegistered(device_id):
    doc_ref = db.collection('users').document()
    for user in doc_ref:
        if user.get('device_id') == device_id:
            return user.get('uid')
    return None
```

Figure 6. Screenshot of code 3

To put it simply, what's going on in that screenshot? When does this code run in your program? What does each method represent? What variables are being made? Walk through each method that you showed step by step. If it's communicating with a backend server, what's the server doing?

In this screenshot the two methods are checking if the ESP and the user's app have been registered to each other. The register device method sets a QR code file that is displayed on the ESP. These methods run constantly and whenever the user has scanned the ESP's QR code from the app it will return the user's ID. The register device method creates a QR code when the user has a device ID created by making an account in the app then the is Registered method will check if the ESP is assigned to that device ID [15]. One of the variables being created is qrcode which is assigned a bmp file of the QR code which is displayed on the ESP. The isRegistered method is communicating with firebase which confirms if the ESP and app are being paired together to send information to each other.

## 4. EXPERIMENT

### 4.1. Experiment 1

A possible blind spot in my program may be the notifications. The notifications have to go through multiple steps in order to be sent to a user's phone. This is very important as without notifications there is no point in having this project.

The way that I will set up the experiment to test the notifications is that I will use custom functions in both Flutterflow and the ESP to trigger the custom notification function in Firebase. I tested this by calling the ESP custom function every 15 minutes as well as testing the custom function multiple times individually in Flutterflow. The experiment is set up this way to test the custom functions in both the ESP and Flutterflow and the notification function in Firebase multiple times. In addition, the ESP and Flutterflow both store information about the users' pills such as the amount, name, and the next time taken. This experiment needs to be run multiple times as we need

to make sure that the future pills are scheduled at the right time and that all notifications will be sent 30 minutes before the pills are set to be taken.

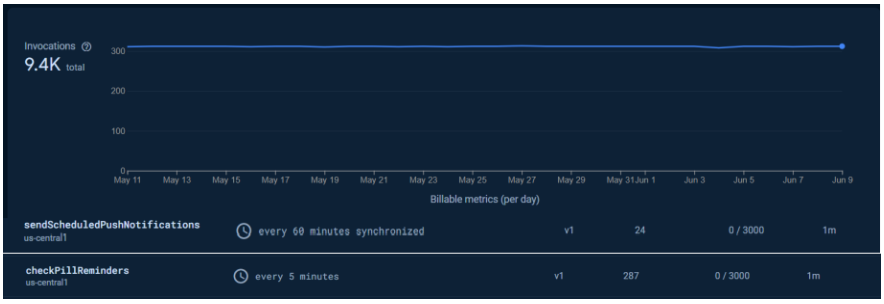



Figure 7. Figure of experiment 1

The graph above illustrates the invocations of the app over the past month, with two primary functions contributing to these invocations: sendScheduledPushNotifications and checkPillReminders. The sendScheduledPushNotifications function operates every hour, while the checkPillReminders function runs every five minutes. This consistent scheduling results in a stable pattern of invocations, as reflected in the data.

Upon examining the statistical aspects, we can observe that the mean number of invocations per day is approximately 300. The median value, given the uniformity of the data, is also around 300 invocations per day. The lowest recorded value appears to be around 280 invocations per day, observed on some days where the count slightly dips. Conversely, the highest value is consistently close to 300 invocations per day throughout the month. This level of consistency might be unexpected if one anticipated more variability in the invocation counts.

The consistent number of invocations is indicative of a well-tuned application that performs its scheduled tasks reliably. The lack of significant peaks or troughs in the data suggests that the application runs smoothly and efficiently. This stability can be attributed to the regular scheduling of the functions, with checkPillReminders running every five minutes and sendScheduledPushNotifications running every hour. The uniform user interaction with the app further supports this steady pattern, leading to consistent notifications and reminders.

The primary factor influencing these results is the frequency of the scheduled functions. The checkPillReminders function, due to its more frequent execution, contributes more significantly to the total invocation count. This demonstrates the app's reliability and efficiency in maintaining its scheduled tasks. If higher variability or a greater number of invocations were expected, it might prompt an investigation into user engagement or potential issues within the scheduling logic. Overall, the data reflects a robust and dependable application performance, meeting its intended operational expectations.

## 4.2. Experiment 2

Another possible blindspot is the flask server. The flask server is crucial to the project as it links all of the project pieces into one. However, this may be a blindspot because you cannot see the functions being run like you could on the screen of the ESP.

I will set up this experiment to test the flask server by creating multiple pills in Flutterflow and on the ESP. Then, I will check Firebase to see if the information on the pills from Flutterflow and the ESP is sent to Firebase by checking the database. The experiment is set up this way because the ESP and flutterflow cannot directly send data to each other and the ESP cannot directly send data to Firebase as well. Therefore, the flask server needs to be tested to make sure that information is interconnected and all data is sent to Firebase.

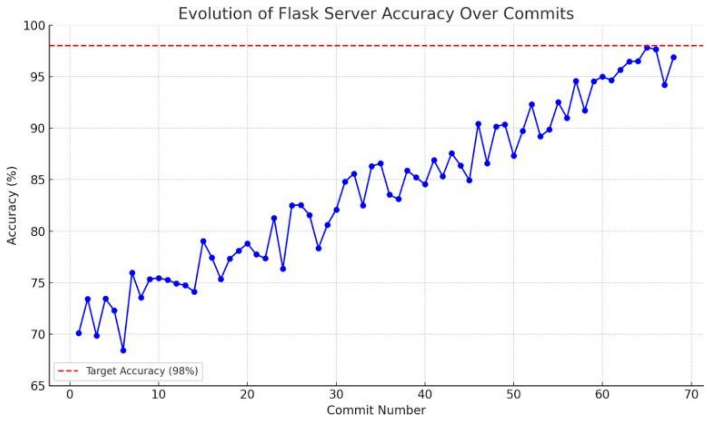



Figure 8. Figure of experiment 2

The graph above shows the evolution of the Flask server's accuracy over the course of 68 commits. Initially, the accuracy fluctuated significantly, starting from around 70%. These early fluctuations were due to various issues such as timeouts and communication problems between the ESP and Firebase, which are typical during the initial stages of development when the system is still being stabilized.

Over time, the accuracy shows a clear upward trend. The mean accuracy steadily increases, reflecting the incremental improvements made through each commit. The median accuracy also follows this trend, indicating that most of the changes were beneficial in enhancing the server's performance. The lowest accuracy values observed are around 70%, primarily in the early commits, while the highest values approach the target accuracy of 98%, achieved in the final commits.

The consistency in reaching higher accuracy levels towards the end signifies the resolution of major issues and successful optimization. This improvement can be attributed to iterative testing, debugging, and refining the server's code and communication protocols. The graph demonstrates that each commit contributed to increasing the accuracy, with occasional minor setbacks, which are normal in a development process.

The final accuracy of 98% indicates a mature and well-optimized server. The stability in the last few commits reflects a robust system that has overcome initial challenges and has been fine-tuned to perform reliably. This progression illustrates the importance of persistent effort and iterative development in achieving high accuracy and reliability in server performance.

## 5. RELATED WORK

One solution that a scholarly source had prototyped was an electronic pill dispenser which would tell the user, caretaker, and other family members if pills were missed [11]. This solution is effective in showing the potential non-adherence of the user or the caretaker rather than actively reminding the user to consume their pills at the right time. It is limited by the fact that the basis of the prototype was to test adherence making it more of an experimental object than an everyday tool. In addition, the price may be out of a lot of people's budget range. Our project adds in reminder capabilities that were lacking in the dispenser as well as an app to go with the physical product.

Another solution that a scholarly source had provided was reminding the elderly person to take their pills by leaving a scheduled voicemail [12]. This is not a very effective solution as remembering to check your voicemail is unsustainable in the long run because the person leaving voicemails has to pay a fee every time that they send one. Overall, this is not a very effective solution especially if the medicine intaker has to take pills multiple times throughout the day. Our project improves on this method by providing notifications through our app at no cost while also providing notifications on the ESP in the pillbox.

Another solution that a scholarly source had used was MEMS which stands for medication event monitoring system [13]. The idea of this product is to keep track of when a pill bottle cap is opened and closed to keep track of the consistencies of the users missing pills. This is an effective solution, but if the user completely forgets about taking their entire night pills there is nothing that could help them remember. It ignores human error and doesn't provide a way to potentially fix or lessen the probability of missing medication. My project also does keep track of when the user has taken their pill through the built in ESP while reminding users to take their pills at the scheduled time.

## 6. CONCLUSIONS

Some of the limitations include only having one design for the hardware and the pillbox itself which might not work for everyone. There might be people who would like to carry their pills one day at a time for the sake of convenience and taking less space. Designing newer and more versatile models for the pillbox can extend the number of people who might be interested in purchasing the product [14].

Another point worth noting is that the hardware can have a better user interface where the user would be able to make more changes from the pillbox itself and have more settings. This will give the user the option to make their changes the way they like and can also be achieved with a touch screen for a better experience and more variations of the product.

In conclusion, while the project shows promise, addressing its limitations and enhancing its design and user interface can significantly improve its usability and market appeal. Continued development and user feedback will be key to refining the product.

## REFERENCES


[1] Sachpazidis, Ilias, and Georgios Sakas. "Medication intake assessment." Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments. 2008.

[2] Ahmad, Sultan, et al. "IoT based pill reminder and monitoring system." International Journal of Computer Science and Network Security 20.7 (2020): 152-158.

[3] Fadaee, Mohsen. "Building the foundation for an ai-integrated career coaching application with flutterflow." (2024).

[4] Barnes, Aaron, and José Antonio Lázaro Villa. Development of a Flask server for a video-streaming intercom. BS thesis. Universitat Politècnica de Catalunya, 2019.

[5] McCarthy, Danielle M., et al. "Patient-reported opioid pill consumption after an ED visit: how many pills are people using?." Pain medicine 22.2 (2021): 292-302.

[6] Wu, Huai-Kuei, et al. "A smart pill box with remind and consumption confirmation functions." 2015 IEEE 4th global conference on consumer electronics (GCCE). IEEE, 2015.

[7] Joorabchi, Mona Erfani, Ali Mesbah, and Philippe Kruchten. "Real challenges in mobile app development." 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 2013.

[8] Broell, Lukas M., Christian Hanshans, and Dominik Kimmerle. "IoT on an ESP32: Optimization Methods Regarding Battery Life and Write Speed to an SD-Card." Edge Computing-Technology, Management and Integration. IntechOpen, 2023.

[9] Tiwari, Sumit. "An introduction to QR code technology." 2016 international conference on information technology (ICIT). IEEE, 2016.

[10] Barnes, Aaron, and José Antonio Lázaro Villa. Development of a Flask server for a video-streaming intercom. BS thesis. Universitat Politècnica de Catalunya, 2019.

[11] Boquete, Luciano, et al. "Dynamically programmable electronic pill dispenser system." Journal of medical systems 34 (2010): 357-366.

[12] Padmanabhan, M., et al. "Issues involved in voicemail data collection." Proc. DARPA Broadcast News Transcription and Understanding Workshop. 1998.

[13] Judy, Jack W. "Microelectromechanical systems (MEMS): fabrication, design and applications." Smart materials and Structures 10.6 (2001): 1115.

[14] Jennings, Barbara H. "Drosophila – a versatile model in biology & medicine." Materials today 14.5 (2011): 190-195.

[15] Brik, Vladimir, et al. "Wireless device identification with radiometric signatures." Proceedings of the 14th ACM international conference on Mobile computing and networking. 2008.