1 Problem Statement

The failure of patients to comply with their medication is a huge challenge. This is because they lack sufficient drug details leading to worsening conditions. Furthermore, using traditional medication reminder methods like pill boxes may inconvenience people thus discouraging them from sticking to their drugs.

The aim of our program is to tackle these difficulties by notifying users about their schedules and offering vital information regarding drug interactions and side effects. Enhancing individuals' ability to make knowledgeable decisions regarding their health and the management of their medication, ultimately leading to better adherence and effectiveness of treatment.

2 Project Goals and Objectives/Deliverables

The goal of the program is to ensure patient safety and medication compliance through the availability of complete pharmaceutical data along with awareness.

The development of the mobile medication reminder system is aimed at improving user accessibility and convenience. The platform will be a mobile application that will enable people to check their adherence levels wherever they are. The application will be developed in Swift for iOS devices and Python on Android.

- **Medication Reminder Application:** A Python/Swift application with full functionality for managing medications and setting reminders.
- **Testing Suite:** Unit and integration tests to guarantee the accuracy and dependability of the features of the program in a range of situations.
- **User Registration and Authentication:** Enable users to safely register accounts and verify their identities in order to gain access to the medication reminder system.
- **Medication Management Page:** Design a page for users to add medication information, including medication names and dosages.
- **Creating reminders:** Developing algorithms to notify the user when the next dose is appropriate, when a medication is about to run out based on medication schedules, taking into account factors such as dosing intervals.

3 Introduction

Many people find it difficult to stick to their medication schedule. The situation here only gets worse as the lack of the information about the drugs makes it even more difficult for the patients to get the medication alerts and the traditional medication reminders are hard to reach. In order to curtail these barriers, a medication reminder software for patients' mobile devices can be an effective solution. The program serves as a pill reminder in addition to alerting users to potential adverse consequences of taking many drugs at once. The program's aim is to guarantee patient safety and medication adherence by providing comprehensive pharmaceutical data and raising awareness. The goal of developing the mobile medication reminder system is to increase user convenience and accessibility. People will be able to check their adherence levels wherever they are via the platform's mobile application. The program will be created in Swift for iOS

devices and in Python for Android devices in order to reach a wider audience. This strategy focuses on developing a mobile product that will meet the needs of numerous groups, leading to its widespread usage and successful outcomes.

The report is orgnized as follows. The project scope is in Section 4. The success factors and benefits in Section 5 The limitations and restrictions are in Section 6. The used dataset details are presented in Section 7. Section 8 and 9 discuss the used programming languages and the practical results. Section 10 presents a comprehensive comparison between the used programming languages. Lastly, the workload, timeline, and tasktable are in Section 11 and 12.

4 Project Scope

Individuals who have a certain medication regimen are the focus of the system since it assists them in being reminded on a regular basis of their prescription appointments at the times that have been defined.

5 Success Factors and Benefits

- **Better Medication Adherence:** The system guarantees that the patients take their medicines at the right time by sticking to the prescriptions given to them and notifying them about their time to take their medicine and in doing so eliminate incidents of skipping the drug or delaying taking it.
- **Reduce Medication Errors:** The system holds the record of the dosages schedule. Thus it got rid of human failures involving forgetting to take the dose or taking a wrong pill amount. This would surely come in handy to the ones who regularly use more than one type of drugs or who consume drugs less than single time per day.
- **Integration with Healthcare Providers:** Allow the users to freely share their medication history and schedule with the healthcare providers. Information provided through remote monitoring will also enable caregivers to monitor the patient's adherence and treatment management during office appointments.
- **Remind the users to Restock:** The software will determines when there will be shortages of medicines. and serve as a reminder for the user to purchase a new supply of medication, offering flexibility for the users, allowing them to decide how many pills they have and how many they take by day.
- **Increase Patients Awareness of side effects:** The system contributes to the user's knowledge of potential side effects of the drug. Accuracy and safety are both important roles of medical imaging, as the guide the outcome of patients' treatments. This better care helps avoid possible health risks.

6 Limitations/Restrictions

Here are some potential limitations or restrictions:

1. **Language Barrier:** The app is designed to operate in the English language, which may be a problem for Non-English speakers.

2. **Data Privacy Concerns:** Privacy and safety of users personal health information may have a notable impact on the overall product effectiveness.

3. **Reliability on User Input:** The success of the system relies on providing a valid data.

4. **Resource Limitation:** Time, funding, or expertise may be the factors that will impact the system.

5. **User Acceptance:** The success of the project depends on the users acceptance.

6. **Medical Disclaimer:** This system should contain an alert indicating that it is not a substitute for a doctor, diagnostic, or treatment.

7 Dataset

For our experiment, we used a large dataset that we obtained from Kaggle that included details on more than 248,000 drugs from different producers all around the world. This dataset is an invaluable resource for our project, offering comprehensive information that is essential for both facilitating user interactions and controlling medicines. The collection includes several key characteristics for every drug, such as:

1. **Medication Name:** With approximately 248,000 Medication names included.

2. **Side effects:** The addition of adverse effects linked to each medicine is one of the dataset's most important features.

We can improve user experience by delivering thorough information on medications, including their names and any potential side effects, by utilizing this Kaggle dataset in our project. This guarantees the best possible care for their medical issues and empowers users to make educated decisions about their prescription schedule [1].

8 Selected Programming Language 1: Python

Python was chosen to build the Medication Reminder Application. Below, a code snippet is provided to demonstrate its use. This snippet clearly illustrates how Python supports these crucial functionalities. Detailed visuals of the application in action are depicted in Figures 1-7. The code is provided in Appendix 12. The image in Figure 1 displays the main menu of a Medication Reminder Application, showing user options like registration, adding medication, and more.

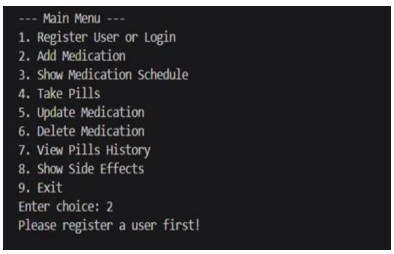


Figure 1: Main menu

Figure 2 highlights the process of a new user registration where the user enters their name and age and is successfully registered.

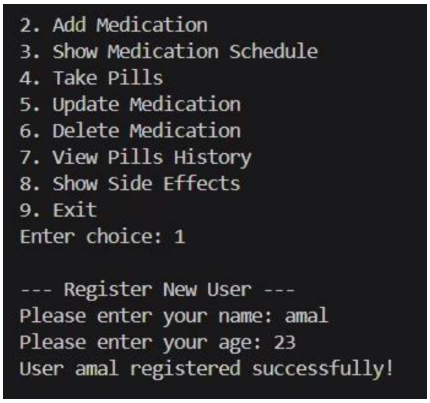
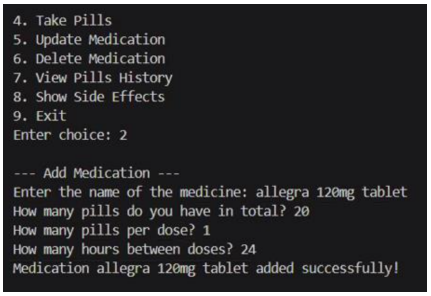
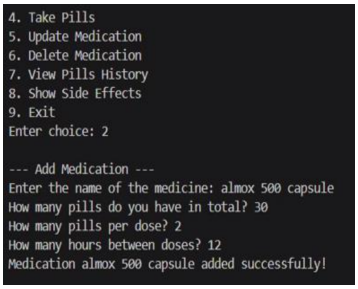


Figure 2: Choice 1

The images show the addition of two medications: allerga and almox, specifying pill count and dosing intervals, with successful confirmation messages for each Figures 3a and 3b.



(a) Choice 2a



(b) Choice 2b

Figure 3: Choice 2

Selecting option 3, "Show Medication Schedule," from the main menu displays the current medication details for "allegra 120mg tablet" and "almox 500 capsule" on the screen. The schedule lists the next dosage times along with the number of pills taken and pills left for each medication, as shown in Figure 6.

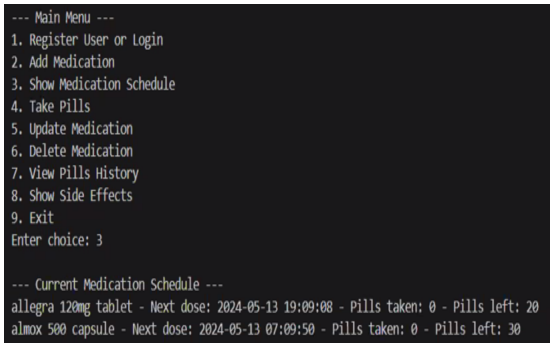
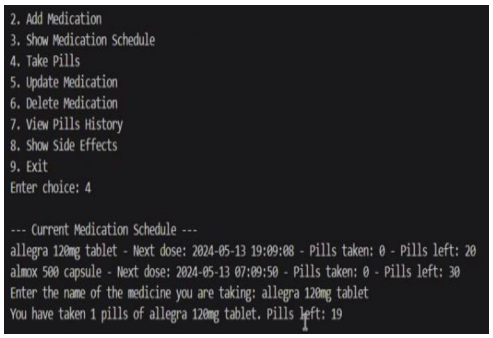
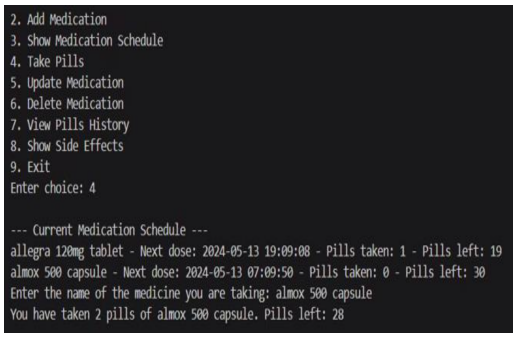


Figure 4: Choice 3

This image showcases the functionality of taking medication through the Medication Reminder Application. When option 4, "Take Pills," is selected, the user first enters the name of the medication "almox 500 capsule" and confirms taking 2 pills. The application updates the count, indicating that 28 pills are left. The sequence for taking "allegra 120mg tablet," where one pill is taken, leaving 19 pills, is also shown. These processes help users manage their medication intake efficiently, as demonstrated in Figure 5a and 5b.



(a) Choice 4a



(b) Choice 4b

Figure 5: Choice 4

This image displays the process of updating medication details in the Medication Reminder Application. After selecting option 5, "Update Medication," the user specifies the medication "almox 500 capsule" to update, entering new values for the total pills count (15), pills per dose (1), and the dosing interval (12 hours). The application confirms the updates with a message indicating that the "almox 500 capsule" was updated successfully, as shown in Figure 6

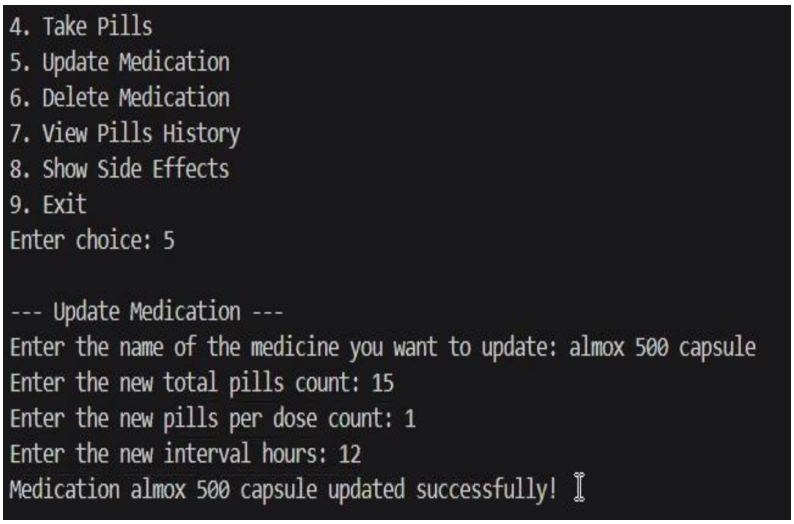


Figure 6: Choice 5

This image illustrates the user accessing the "View Pills History" function in the application by selecting option 7 from the main menu. The history screen displays detailed records for medications, showing "allegra 120mg tablet" with an original total of 20 pills, 1 taken, leaving 19 pills. Similarly, "almox 500 capsule" is shown with an original total of 30 pills, 2 taken, and 15 left, as shown in 7.

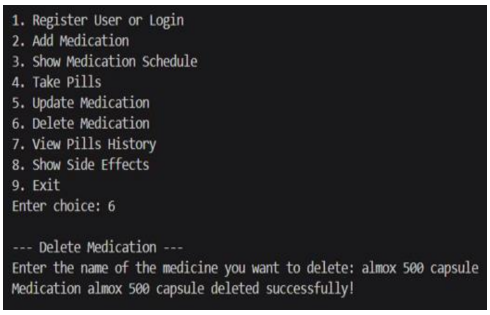


Figure 7: Choice 7

This image shows the user accessing the "Show Side Effects" function by selecting option 8 from the main menu of the application. Upon entering the medication name "almox 500 capsule," the application displays its associated side effects: Vomiting, Allergic reaction, Nausea, and Diarrhea. This feature helps users be aware of potential adverse reactions to medications they are taking, as demonstrated in Figure 8.

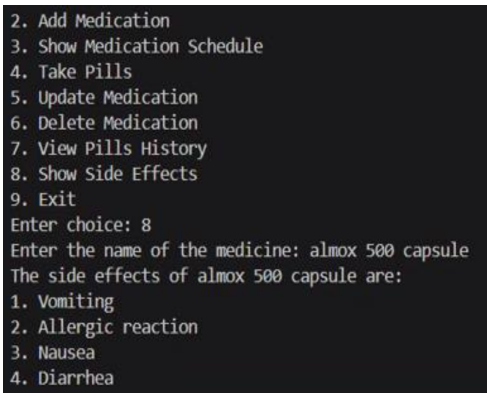


Figure 8: Choice 8

Figure 9 displays the process of medication deletion in the Medication Reminder Application. After choosing option 6, "Delete Medication," from the main menu, the user types in "almox 500 capsule" for deletion. The application confirms the successful removal of the medication, helping users keep their medication list updated.

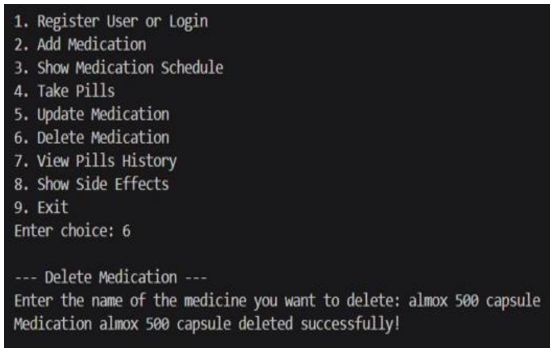


Figure 9: Choice 6

Figure 10 displays the outcome of selecting option 3, "Show Medication Schedule," from the main menu to verify updates after medication deletion. It shows the remaining "allegra 120mg tablet" with details about the next dose, pills taken, and pills left.

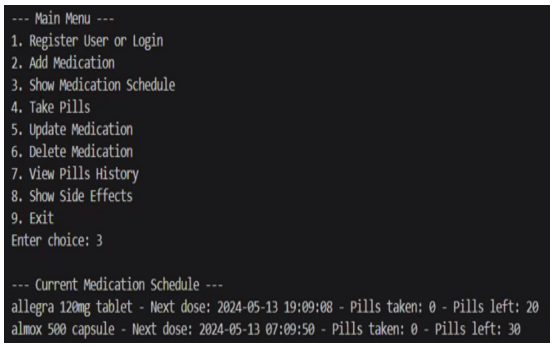


Figure 10: Choice 3

Figure 11 shows the selection of option 9, "Exit," from the main menu of the application, confirming the program's closure with the message "Exiting program".

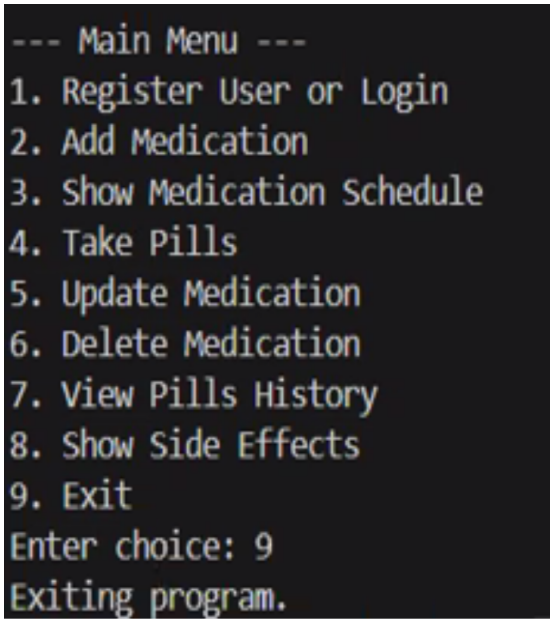


Figure 11: Choice 9

9 Selected programming language 2: Swift

Swift was chosen as the second programming language to build the Medication Reminder Application. Below, a code snippet is provided to demonstrate its use in setting medication reminders and managing inventory. Detailed visuals of the application in action are depicted in Figures 6-12. The code is provided in Appendix 12. Figure 12 displays the main menu of a Medication Reminder Application, showing user options in main menu like registration, adding medication, and more.

```
--- Main Menu ---
1. Register User or Login
2. Add Medication
3. Show Medication Schedule
4. Take Pills
5. View Pills History
6. Update Medication
7. Delete Medication
8. Print Side Effects for Medication
9. Exit
```

Figure 12: Swift main menu

The user registration and login process in the first choice is shown in Figure 13a and 13.

```
Enter choice: 1

--- Register New User ---
Please enter your name: Khaznah
Please enter your age: 23
User Khaznah registered successfully!
```

(a) Registration

```
Enter choice: 1

--- Register New User ---
Please enter your name: Khaznah
Please enter your age: 23
Welcome back, Khaznah!
```

(b) Login

Figure 13: Choice 1

In Figure 14, the addition of medication "Adol", specifying pill count and dosing intervals, with successful confirmation messages.

```
Enter choice: 2

--- Add Medication ---
Enter the name of the medicine: Adol
How many pills do you have in total? 24
How many pills per dose? 1
How many hours between doses? 24
Medication Adol added successfully!
```

Figure 14: Choice 2

Figure 15 displays the "Current Medication Schedule" from a Medication Reminder Application, listing the details on next doses and pills remaining. Figure 16 shows the "Medication Schedule" function, where the user logs taking 1 pill of "Adol", updating the pills left to 23.

```
Enter choice: 3

--- Current Medication Schedule ---
Adol - Next dose: Monday, 13 May 2024 at 7:16:33 AM Arabian Standard Time -
Pills taken: 0 - Pills left: 24
```

Figure 15: Choice 3

```
Enter choice: 4

--- Current Medication Schedule ---
Adol - Next dose: Monday, 13 May 2024 at 7:16:33 AM Arabian Standard Time -
Pills taken: 0 - Pills left: 24
Enter the name of the medicine you are taking: Adol
You have taken 1 pills of Adol. Pills left: 23
```

Figure 16: Choice 4

While Figure 17a and 17b show the Pills History before and after the deletion function.

```
Enter choice: 5

--- Pills History ---
Adol - Original Total Pills: 24, Total Pills Taken: 1, Pills Left: 23
```

(a) Choice 5 - Before deletion

```
Enter choice: 5

--- Pills History ---
```

(b) Choice 5 - After deletion

Figure 17: Choice 5

Figure 18 shows the "Update Medication" function, where the user can update the total pills count, dose count, and interval hours.

```
Enter choice: 6

--- Update Medication ---
Enter the name of the medicine you want to update: Adol
Enter new total pills count: 20
Enter new pills per dose count: 2
Enter new interval hours: 12
Medication Adol updated successfully!
```

Figure 18: Choice 6

Figure 19 shows the user selecting "Delete Medication" from the menu, where the medication will be deleted from the history and schedule.

```
Enter choice: 7

--- Delete Medication ---
Enter the name of the medication you want to delete: Adol
Medication Adol deleted successfully!
```

Figure 19: Choice 7

Figure 20 shows the user selecting "Side Effects" from the menu, where the side effects will printed based on the uploaded database.

```
Enter choice: 8

--- Side Effects ---
Enter the name of the medication: Adol
No side effects found for Adol.
```

Figure 20: Choice 7

Lastly, Figure 21 shows the user selecting "Exit" from the menu, leading to a message confirming the successful termination of the program.

```
Enter choice: 9
Exiting program.
Program ended with exit code: 0
```

Figure 21: Choice 8

10 Comparison between two selected programming languages

10.1 Programming Language

Python: Python is the most widely used programming languages. Technology has found its way in the software development as well as influences the construction of the websites. This domain is being used in data science, machine learning, and artificial intelligence. Python is specialized in some collection of libraries and frameworks which help in solving multiple issues.

Swift: Swift is a general-purpose, multi-paradigm, object-oriented, functional, imperative, and block-structured programming language. Apple Inc. developed Swift from its modern perspective on programming languages and applied safety measurements, and software design patterns specifically for the purpose of iOS apps, for macOS apps, for watchOS apps and for tvOS apps [2].

10.2 Syntax

Python: The syntax of Python is distinguished by its focus on readability and flexibility. The main characteristics comprise of indentation for defining code blocks, obviating the necessity for braces; dynamic typing, enabling variable types to be deduced at runtime; and succinct comments using the hash mark (#). Functions are defined using the "def" keyword, whereas classes are defined using the "class" keyword. Methods within classes have an explicit "self" argument to access instance variables. Control structures encompass if, elif, else for conditional actions, as well as for and while loops for iteration, which provide direct iteration over elements of sequences. Modules are included in the code using the import statement, and the handling of errors is controlled via try and except blocks, allowing for effective management of exceptions. The utilization of this syntactic structure not only improves the readability and ease of maintaining the code, but also streamlines the entire programming procedure [3].

Swift: The syntax of this language has been crafted with much care to be precise and simple, at the same time. Data types, including Integers, Floats, Doubles, Strings, Booleans, Arrays,

Dictionaries, and more, are supported by the language. Additionally, Swift functions are flexible, allowing for multiple return values via tuples and supporting default values for parameters. In the realm of control flow, developers can effectively create codebases that are well-organized and structured by utilizing a combination of advanced constructs like guard and defer, together with a repertoire of traditional statements like if, else, switch, for, and while. Object-oriented programming (OOP) principles form the foundation of Swift, giving developers the freedom to use enums, classes, and structures as they see fit. In particular, classes provide inheritance; as value types, structs, and enums offer modular and scalable code architectures [4].

10.3 Dependencies

Python: Dependencies are essential software components needed for your project to function correctly and prevent any runtime errors. PyPI (the Python Package Index) is a reliable source for packages that can assist you in various tasks. There are numerous ways to manage and add dependencies to a Python project, just like a data scientist would explore different tools and methods for dependency management. For instance: Pip, pip, described as the "most popular tool" for installing Python Packages and handling their dependency problems, is definitely very popular among the Python community. Install the package by using the command:

```
pip install <packagename>
```

Regrettably, pip does not make any effort to resolve dependency conflicts. As an illustration, when you install two packages, package A might necessitate a distinct version of a dependency compared to what package B requires [5].

Swift: A package dependency consists of a Git URL to the source of the package, and a requirement for the version of the package. The Swift Package Manager performs a process called dependency resolution to figure out the exact version of the package dependencies that an app or other Swift package can use. The Package.resolved file records the results of the dependency resolution and lives in the top-level directory of a Swift package [6].

10.4 Structured/Functional/Object Oriented

10.4.1 Python:

- Object Oriented programming paradigms

  Objects serve a crucial role in the object-oriented programming paradigm. Objects are essentially instances of a class that encompass both data members and method functions. In addition, the object-oriented style connects data members and methods to promote encapsulation. By utilizing inheritance, code can be easily reused.

- **Imperative/Procedural Programming Paradigms**

  When using Procedure Oriented programming paradigms, the code is organized into functions and executed step by step. This allows for a clear and sequential set of instructions for the computer to follow. This approach promotes code modularity, which is typically achieved through functional implementation. With this programming paradigm, organizing related items becomes a breeze. Each file serves as a container, making organization a seamless process.

- **Functional Programming Paradigms**

  Functional programming paradigms involve binding everything in a pure mathematical functions style. This approach is commonly referred to as declarative paradigms, as it relies on the use of declarations rather than explicit statements. It utilizes mathematical functions and treats each statement as a functional expression, executing it to generate a value. Using lambda functions or recursion are common techniques employed in its implementation. The paradigms primarily emphasize the aspect of problem-solving rather than the approach to solving it. Using functions as values and passing them as arguments can enhance the readability and comprehensibility of the code.

10.4.2 Swift:

- **Object Oriented Programming Paradigms**

  Object-Oriented Programming (OOP) is at the core of Swift, introduced by Apple to address the limitations of Objective-C. Launched at WWDC 2014, Swift merges Python's readability with C++'s speed and is known for its conciseness and type safety. While retaining many OOP patterns from its predecessors, Swift also aims to address the paradigm's limitations, such as scalability and over-reliance on inheritance. Despite its rapid rise in popularity, Swift still upholds traditional OOP elements like UIViewController classes but lacks advanced reactive programming constructs. Swift adeptly balances modern demands with traditional strengths, making it both a language of today and tomorrow [7].

- **Functional Programming Paradigms**

  Swift has seamlessly addressed the limitations of Objective-C by offering a language that combines Python's readability with C++'s speed. As a concise, type-safe, and versatile programming language, Swift retains strong ties to Object-Oriented Programming (OOP) patterns, despite known OOP challenges such as scalability issues and a preference for inheritance over composition. Rooted in OOP with its use of UIViewController classes and the absence of constructs like Promises and Futures, Swift effectively utilizes OOP where it's most effective. While incorporating functional programming elements like enhanced closure syntax and support for functions like map and reduce, Swift predominantly functions within an OOP framework, adeptly bridging traditional and modern programming paradigms [7].

- **Structured Programming Paradigms** Structured programming is a programming paradigm that focuses on breaking down a program's logic into smaller, more manageable structures. It emphasizes clear, understandable code by avoiding the use of unstructured control flow mechanisms. Swift encourages developers to use constructs like functions, loops, conditionals, and control flow statements in a structured and organized manner [8].

10.5 Memory Management

10.5.1 Python:

Python's memory management is handled by an internal memory manager that operates within a private heap, where all Python objects and data structures are stored. A raw memory allocator directly interacts with the operating system's memory manager to ensure sufficient space in the heap for Python's data. Additionally, specific allocators are responsible for managing various types of objects, such as integers, strings, tuples, and dictionaries. Each allocator

is designed to meet the unique storage requirements and performance considerations of these objects. These allocators have their own specialization, but they still follow the guidelines set by the Python memory manager to maintain consistency with the private heap's policies. The Python interpreter takes care of managing the heap, and users do not have direct control over it. However, they can manipulate object pointers within the heap. Having a well-organized memory management system is crucial for optimizing memory usage in Python, allowing for effective storage management across various object types.

10.5.2 Swift:

Automatic reference counting (ARC) is the memory management system which automatically tracks down the references to an object, instance objects which are well known in computer science and automatically deallocate memory once an object has no more use, causing memory leaks and program persistent performance.Swift utilizes ARC and is structured as a collection of projects, each managed in separate repositories. These projects include the Swift compiler command-line tool, the standard library that is part of the language, core libraries offering higher-level functionality, the Swift REPL with an integrated LLDB debugger, support for Xcode playgrounds, and the Swift package manager, which handles the distribution and building of Swift source code [2].

10.6 Memory Allocation/De-allocation

Python efficiently manages memory allocation and de-allocation through a dedicated private heap space for all Python objects and data structures. A memory allocator interacts with the operating system's memory manager to ensure that Python has enough space for its data requirements. Allocating memory for objects at runtime is a crucial aspect of programming. It allows for dynamic memory allocation, which is done during the execution of a program rather than during compilation. It provides the ability to allocate memory based on the program's requirements, allowing data structures and objects to adjust in size as needed. Python manages dynamic memory allocation primarily through mechanisms such as garbage collection and reference counting. Python's memory management system efficiently handles the allocation and deallocation of memory on the heap, where dynamically allocated objects reside. Python uses reference counting to handle memory deallocation, automatically freeing up space when an object's reference count reaches zero [9].

In Swift, memory management is handled automatically through Automatic Reference Counting (ARC). Memory is set aside to hold the instance of a class or struct that you create. ARC counts the number of references to a specific object that is out there. ARC deallocates the memory that is linked to an object when there are no more references to it. This automated procedure guarantees effective memory management without forcing the developer to manually manage memory. Strong reference cycles, in which objects hold references to one another and ARC is unable to deallocate them, are a concern for developers. Swift offers tools like weak and unowned references to help with this [10].

10.7 Functions Comparison

10.7.1 registerUser()

- **Swift Programming Language:**
  func registerUser() -> String : This line defines a function named registerUser that takes

no parameters and returns a String. print()is a function that print messages for the users to guide them through the program. print("", terminator: ""): This line prints a prompt asking the user to enter their name without adding a new line character at the end. let variable = readLine()!: This line reads the input from the user. readLine() reads input from the program and returns an optional string (String?). The ! force unwraps the optional to get the actual string value entered by the user. if users[name] != nil. This line checks if the entered name exists in users dictionary or not. If it exists then print function will print a welcome message, if it does not exist. Then users[name] = ["name": name, "age": age, "medications": []] will create a new entry in users. Return name will end the function and return the name of the user.

```
    func registerUser() -> String {
    print("\n--- Register New User ---")
    print("Please enter your name: ", terminator: "")
    let name = readLine()!
    print("Please enter your age: ", terminator: "")
    let age = readLine()!
    if users[name] != nil {
        print("Welcome back, \(name)!")
        return name
    }
    users[name] = ["name": name, "age": age, "medications": []]
    print("User \(name) registered successfully!\n")
    return name
}
```

- **Python Programming Language:**
  def register user() is used to create the function register user(), and function print() is to print messages for the users to guide them through the program. variable = input() takes user input and stores it inside variables. if name in users: This line checks if the entered name already exists in the users' dictionary. users[name] = 'name': name, 'age': age, 'medications': []. This line checks if the entered name exists in users' dictionary, if it does not exist, then a new entry will be created in users. Return name will end the function and return the name of the user.

```
        def register_user():
    print("\n--- Register New User ---")
    name = input("Please enter your name: ")
    age = input("Please enter your age: ")
    if name in users:
        print(f"Welcome back, {name}!")
        return name
    users[name] = {'name': name, 'age': age, 'medications': []}
    print(f"User {name} registered successfully!\n")
    return name
```

10.7.2 showSchedule()

- **Swift Programming Language:**
  func showSchedule(user: String) : This line defines a function named showSchedule that takes a user parameter of type String. if let attempts to retrieve the medication information for the specified user from the users dictionary. If the user exists and has medications, it casts the value associated with the "medications" key to an array of dictionaries with string keys and any value types for med in userMedications : This line starts a loop iterating over each medication dictionary in the userMedications array. Then the program will retrieve the next ode time, total pills the user has, and check if the number of pills left is less than or equal to 3. Then use print() to print all information for the medication.

```
func showSchedule(user: String) {
print("\n--- Current Medication Schedule ---")
if let userMedications = users[user]?["medications"] as? [[String: Any]] {
    for med in userMedications {
        let nextDoseStr =
        (med["nextDoseTime"] as! Date).description(with: .current)
        let pillsLeft = med["totalPills"] as! Int
        let warningMsg = pillsLeft <= 3 ? " (Warning: Low on medication!)"
        : "" print("\(med["name"]!) - Next dose: \(nextDoseStr)
        - Pills taken: \(med["doseTaken"]!) - Pills left:
        \(pillsLeft)\(warningMsg)")
    } } }
```

- **Python Programming Language:**
  This function takes a user parameter. And start a for loop iterating over each medication dictionary in the list of medications associated with the specified user in the users dictionary.. and retrives the next ode time using strftime() method. Then retrieves the total number of pills and check if they are less than or equal to 3. Then print all the medication information for the user.

```
    def show_schedule(user):
print("\n--- Current Medication Schedule ---")
for med in users[user]['medications']:
    next_dose_str = med['next_dose_time'].strftime("%Y-%m-%d %H:%M:%S")
    pills_left = med['total_pills']
    warning_msg = " (Warning: Low on medication!)" if pills_left <=
    3 else ""
    print(f"{med['name']} - Next dose: {next_dose_str} - Pills taken:
    {med['dose_taken']} - Pills left: {pills_left}{warning_msg}")
```

10.7.3 deleteMedication()

- **Swift Programming Language:**
  The function takes a user parameter of type String. let medName = readLine()!: This line reads the input from the user for the name of the medication to be deleted from the

console. readLine() reads input from the user and returns an optional string (String?). then use if var to retrieve the medication information for the specified user from the users' dictionary. Then use the filter method to create a new array excluding the medication with the specified name. then update the medications associated with the specified user in the users' dictionary with the filtered userMedications array.

```
func deleteMedication(user: String) {
    print("\n--- Delete Medication ---")
    print("Enter the name of the medication you want to delete: ", terminator:
    "")  let medName = readLine()!
    if var userMedications = users[user]?["medications"] as? [[String: Any]] {
        userMedications = userMedications.filter { ($0["name"] as?
        String)?.lowercased() != medName.lowercased() }
        users[user]?["medications"] = userMedications
        print("Medication \(medName) deleted successfully!\n")
    } else {
        print("No medications found for deletion.")
    } }
```

- **Python Programming Language:**
  This function takes a user parameter. It takes a variable = input() which takes the medicine name from the users which they want to delete from their schedule. It then goes through a for loop that iterates for each medication dictionary in the list of medications associated with the specified user in the users' dictionary. Then check if the name of the current medication matches the name the user entered (case insensitive). Then use a remove() function to delete the matching name if found. And then use return to end the function. If no matching is found, then an error message is printed.

```
def delete_medication(user):
    print("\n--- Delete Medication ---")
    med_name = input("Enter the name of the medicine you want to delete: ")
    for med in users[user]['medications']:
        if med['name'].lower() == med_name.lower():
            users[user]['medications'].remove(med)
            print(f"Medication {med_name} deleted successfully!\n")
            return
    print("Medication not found.")
```

10.7.4 readSideEffectsCSV()

- **Swift Programming Language:**
  This function takes a parameter medName of type String.then attempts to retrieve the side effects associated with the given medication name from the sideEffectsData dictionary. It uses optional binding (if let) to unwrap the optional value associated with the medName key. Then starts a loop iterating over each side effect in the sideEffects array. then prints each side effect preceded by a hyphen, formatting it as a bullet point. If no side effects are found, then an error message will be printed.

```
func printSideEffectsForMedication(medName: String) {
if let sideEffects = sideEffectsData[medName] {
    print("\nSide Effects for \(medName):")
    for effect in sideEffects {
        print("- \(effect)") } } else {
    print("No side effects found for \(medName).")
} }
```

- **Python Programming Language:**
  The function first filters the "medication_dataset" DataFrame to retrieve information about the medication by comparing the user input to the name in the dataset. The comparison is case insensitive. And check if checks if the filtered DataFrame is not empty. Then initialize an empty list to store the side effects of the medication. Otherwise, print an error message. Then start a for loop to iterate over all side effects listed on the dataset. Select the values and append them to the side effects list. Then print all the values stored in the data set. if no values are found then print an error message.

```
    def display_side_effects(med_name):
med_info = medication_dataset[medication_dataset['name'].str.lower()
== med_name.lower()]
if not med_info.empty:
    print(f"The side effects of {med_name} are:")
    side_effects = []
    for i in range(1, 15):
        side_effect = med_info[f'sideEffect{i}'].iloc[0]
        if not pd.isna(side_effect):
            side_effects.append(side_effect)
    if side_effects:
        for i, side_effect in enumerate(side_effects, start=1):
            print(f"{i}. {side_effect}")
    else:
        print("No side effects listed.")
else:
    print("Medication not found in the dataset.")
```

11 Workload and Timeline

Our project timeline follows the Agile process. The flexible and iterative way to develop software. Agile is particularly a good fit for our program. It is all about adapting quickly, working together, and making things better as we go. Agile approach ensures that our team adjust to changing needs as they come up, providing opportunities for feedback and incorporating improvements throughout the development process [11]. The time line is illustrated in Figure 22.

Figure 22: Timeline

12 Task Table

Name	Tasks	Status	Deadline
Khaznah	• Swift Code (With help of group members) • Limitations/Restrictions • Organize the report on LaTeX	Resolved	05/12/2024
Dema	• Introduction • Project Scope • Timeline	Resolved	05/12/2024
Amal	• The Dataset • Python Code (With help of group members) • Comparison between two selected programming languages (Python Language) • Tasktable	Resolved	05/12/2024
Layan	• Selected programming language (Python Language) • Comparison between two selected programming languages (Swift Language)	Resolved	05/12/2024
Ghaida	• Problem statement • Project Goals and Objectives/Deliverables	Resolved	05/12/2024
Tuqa	• Success factors and benefits • Evaluate the programming paradigm	Resolved	05/12/2024

References

[1] Shudhanshu Singh. 250k medicines usage, side effects and substitutes. Kaggle, 2023. Accessed on May 12, 2024.

[2] GeeksforGeeks. Swift programming language, 2024. Accessed on May 12, 2024.

[3] GeeksforGeeks. Python syntax, 2024. Accessed on May 12, 2024.

[4] Documentation. Accessed on May 12, 2024.

[5] Coursera. Programming in swift: Benefits of this popular coding language, 2023. Coursera.

[6] Swift.org. Package dependency. Accessed on May 12, 2024.

[7] Method. Swift blurs the lines of programming paradigms. Accessed on May 12, 2024.

[8] docs.swift.org. Structures and classes. Accessed on May 12, 2024.

[9] Memory management. Accessed on May 12, 2024.

[10] Automatic reference counting. Accessed on May 12, 2024.

[11] Harleen K Flora, Swati V Chande, and Xiaofeng Wang. Adopting an agile approach for the development of mobile applications. *International Journal of Computer Applications*, 94(17):43–50, 2014.

Python Code

```
import time

from datetime import datetime, timedelta

import pandas as pd


medication_dataset = pd.read_csv(r'C:\Users\Acer\Desktop\medicine_dataset.csv')


users = {}


def register_user():

    print("\n--- Register New User ---")

    name = input("Please enter your name: ")

    age = input("Please enter your age: ")

    if name in users:

        print(f"Welcome back, {name}!")

        return name

    users[name] = {'name': name, 'age': age, 'medications': []}

    print(f"User {name} registered successfully!\n")

    return name


def add_medication(user):

    print("\n--- Add Medication ---")

    med_name = input("Enter the name of the medicine: ")

    total_pills = int(input("How many pills do you have in total? "))
```

```
    pills_per_dose = int(input("How many pills per dose? "))
    interval_hours = int(input("How many hours between doses? "))
    next_dose_time = datetime.now() + timedelta(hours=interval_hours)

    medication = {
        'name': med_name,
        'original_total_pills': total_pills,  # Store original total for history
        'total_pills': total_pills,
        'pills_per_dose': pills_per_dose,
        'next_dose_time': next_dose_time,
        'dose_taken': 0  # Track the total number of pills taken
    }
    users[user]['medications'].append(medication)
    print(f"Medication {med_name} added successfully!\n")

def show_schedule(user):
    print("\n--- Current Medication Schedule ---")
    for med in users[user]['medications']:
        next_dose_str = med['next_dose_time'].strftime("%Y-%m-%d %H:%M:%S")
        pills_left = med['total_pills']
        warning_msg = " (Warning: Low on medication!)" if pills_left <= 3 else ""
        print(f"{med['name']} - Next dose: {next_dose_str} - Pills taken: {med['dose_taken']}

def take_pills(user):
```

```
    show_schedule(user)
    med_name = input("Enter the name of the medicine you are taking: ")
    for med in users[user]['medications']:
        if med['name'].lower() == med_name.lower():
            if med['total_pills'] >= med['pills_per_dose']:
                med['total_pills'] -= med['pills_per_dose']
                med['dose_taken'] += med['pills_per_dose']
                warning_msg = " (Warning: Low on medication!)" if med['total_pills'] <= 3 els
                print(f"You have taken {med['pills_per_dose']} pills of {med_name}. Pills lef
            else:
                print(f"Not enough pills left to take a dose of {med_name}.")
            return
    print("Medication not found.")


def update_medication(user):
    print("\n--- Update Medication ---")
    med_name = input("Enter the name of the medicine you want to update: ")
    for med in users[user]['medications']:
        if med['name'].lower() == med_name.lower():
            total_pills = int(input("Enter the new total pills count: "))
            pills_per_dose = int(input("Enter the new pills per dose count: "))
            interval_hours = int(input("Enter the new interval hours: "))
            med['total_pills'] = total_pills
```

```
            med['pills_per_dose'] = pills_per_dose
            med['next_dose_time'] = datetime.now() + timedelta(hours=interval_hours)
            print(f"Medication {med_name} updated successfully!\n")
            return
    print("Medication not found.")

def delete_medication(user):
    print("\n--- Delete Medication ---")
    med_name = input("Enter the name of the medicine you want to delete: ")
    for med in users[user]['medications']:
        if med['name'].lower() == med_name.lower():
            users[user]['medications'].remove(med)
            print(f"Medication {med_name} deleted successfully!\n")
            return
    print("Medication not found.")

def main_menu(user):
    print("\n--- Main Menu ---")
    print("1. Register User or Login")
    print("2. Add Medication")
    print("3. Show Medication Schedule")
    print("4. Take Pills")
    print("5. Update Medication")
    print("6. Delete Medication")
```

```
    print("7. View Pills History")
    print("8. Show Side Effects")
    print("9. Exit")

    choice = input("Enter choice: ")
    return choice

def view_pills_history(user):
    print("\n--- Pills History ---")
    for med in users[user]['medications']:
        original_total = med['original_total_pills']
        pills_taken = med['dose_taken']
        pills_left = med['total_pills']
        print(f"{med['name']} - Original Total Pills: {original_total}, Total Pills Taken: {p

def display_side_effects(med_name):
    med_info = medication_dataset[medication_dataset['name'].str.lower() == med_name.lower()]
    if not med_info.empty:
        print(f"The side effects of {med_name} are:")
        side_effects = []
        for i in range(1, 15):
            side_effect = med_info[f'sideEffect{i}'].iloc[0]
            if not pd.isna(side_effect):
```

```
                side_effects.append(side_effect)
        if side_effects:
            for i, side_effect in enumerate(side_effects, start=1):
                print(f"{i}. {side_effect}")
        else:
            print("No side effects listed.")
    else:
        print("Medication not found in the dataset.")



def main():
    current_user = None
    while True:
        choice = main_menu(current_user)
        if choice == '1':
            current_user = register_user()
        elif choice == '2':
            if current_user:
                add_medication(current_user)
            else:
                print("Please register a user first!")
        elif choice == '3':
            if current_user:
                show_schedule(current_user)
```

```
    else:
        print("No user registered!")
elif choice == '4':
    if current_user:
        take_pills(current_user)
    else:
        print("No user registered or medication added!")
elif choice == '5':
    if current_user:
        update_medication(current_user)
    else:
        print("No user registered or medication added!")
elif choice == '6':
    if current_user:
        delete_medication(current_user)
    else:
        print("No user registered or medication added!")
elif choice == '7':
    if current_user:
        view_pills_history(current_user)
    else:
        print("No user registered or medication added!")
elif choice == '8':
```

```
            if current_user:
                med_name = input("Enter the name of the medicine: ")
                display_side_effects(med_name)
            else:
                print("No user registered or medication added!")
        elif choice == '9':
            print("Exiting program.")
            break
        else:
            print("Invalid choice, please try again!")


if __name__ == "__main__":
    main()
```

Swift Code

```
import Foundation

// Simulated "database" for storing user and medication information
var users: [String: [String: Any]] = [:]
var sideEffectsData: [String: [String]] = [:]

func registerUser() -> String {
    print("\n--- Register New User ---")
    print("Please enter your name: ", terminator: "")
    let name = readLine()!
    print("Please enter your age: ", terminator: "")
    let age = readLine()!
    if users[name] != nil {
        print("Welcome back, \(name)!")
        return name
```

```
    }
    users[name] = ["name": name, "age": age, "medications": []]
    print("User \(name) registered successfully!\n")
    return name
}

func addMedication(user: String) {
    print("\n--- Add Medication ---")
    print("Enter the name of the medicine: ", terminator: "")
    let medName = readLine()!
    print("How many pills do you have in total? ", terminator: "")
    guard let totalPills = Int(readLine()!) else {
        print("Invalid input for total pills. Please enter a valid number.")
        return
    }
    print("How many pills per dose? ", terminator: "")
    guard let pillsPerDose = Int(readLine()!) else {
        print("Invalid input for pills per dose. Please enter a valid number.")
        return
    }
    print("How many hours between doses? ", terminator: "")
    guard let intervalHours = Int(readLine()!) else {
        print("Invalid input for interval hours. Please enter a valid number.")
        return
    }
    let nextDoseTime = Date().addingTimeInterval(TimeInterval(intervalHours) * 3600)

    let medication: [String: Any] = [
        "name": medName,
        "originalTotalPills": totalPills,  // Store original total for history
        "totalPills": totalPills,
        "pillsPerDose": pillsPerDose,
        "nextDoseTime": nextDoseTime,
        "doseTaken": 0  // Track the total number of pills taken
    ]
    if var userMedications = users[user]?["medications"] as? [[String: Any]] {
        userMedications.append(medication)
        users[user]?["medications"] = userMedications
    }
    print("Medication \(medName) added successfully!\n")
}

func showSchedule(user: String) {
    print("\n--- Current Medication Schedule ---")
    if let userMedications = users[user]?["medications"] as? [[String: Any]] {
        for med in userMedications {
            let nextDoseStr = (med["nextDoseTime"] as! Date).description(with: .current)
```

```
            let pillsLeft = med["totalPills"] as! Int
            let warningMsg = pillsLeft <= 3 ? " (Warning: Low on medication!)" : ""
            print("\(med["name"]!) - Next dose: \(nextDoseStr) - Pills taken: \(med["doseTake
        }
    }
}

func takePills(user: String) {
    showSchedule(user: user)
    print("Enter the name of the medicine you are taking: ", terminator: "")
    let medName = readLine()!
    if let userMedications = users[user]?["medications"] as? [[String: Any]] {
        for medIndex in 0..<userMedications.count {
            if userMedications[medIndex]["name"] as? String == medName {
                var updatedMed = userMedications[medIndex]
                let totalPills = updatedMed["totalPills"] as! Int
                let pillsPerDose = updatedMed["pillsPerDose"] as! Int
                if totalPills >= pillsPerDose {
                    let updatedTotalPills = totalPills - pillsPerDose
                    updatedMed["totalPills"] = updatedTotalPills
                    let doseTaken = updatedMed["doseTaken"] as! Int
                    updatedMed["doseTaken"] = doseTaken + pillsPerDose
                    var updatedUserMeds = userMedications
                    updatedUserMeds[medIndex] = updatedMed
                    users[user]?["medications"] = updatedUserMeds
                    let warningMsg = updatedTotalPills <= 3 ? " (Warning: Low on medication!)
                    print("You have taken \(pillsPerDose) pills of \(medName). Pills left: \(
                    return
                } else {
                    print("Not enough pills left to take a dose of \(medName).")
                    return
                }
            }
        }
    }
    print("Medication not found.")
}

func updateMedication(user: String) {
    print("\n--- Update Medication ---")
    print("Enter the name of the medicine you want to update: ", terminator: "")
    let medName = readLine()!

    if var userMedications = users[user]?["medications"] as? [[String: Any]] {
        for medIndex in 0..<userMedications.count {
            if userMedications[medIndex]["name"] as? String == medName {
                print("Enter new total pills count: ", terminator: "")
```

```
                guard let newTotalPills = Int(readLine()!) else {
                    print("Invalid input for total pills. Please enter a valid number.")
                    return
                }
                print("Enter new pills per dose count: ", terminator: "")
                guard let newPillsPerDose = Int(readLine()!) else {
                    print("Invalid input for pills per dose. Please enter a valid number.")
                    return
                }
                print("Enter new interval hours: ", terminator: "")
                guard let newIntervalHours = Int(readLine()!) else {
                    print("Invalid input for interval hours. Please enter a valid number.")
                    return
                }

                // Update medication details
                userMedications[medIndex]["totalPills"] = newTotalPills
                userMedications[medIndex]["pillsPerDose"] = newPillsPerDose
                userMedications[medIndex]["nextDoseTime"] = Date().addingTimeInterval(TimeInt

                users[user]?["medications"] = userMedications
                print("Medication \(medName) updated successfully!")
                return
            }
        }
        print("Medication not found.")
    }
}

func readSideEffectsCSV(filePath: String) {
    do {
        let csvFileContents = try String(contentsOfFile: filePath)
        let csvLines = csvFileContents.components(separatedBy: .newlines)
        for line in csvLines {
            let lineComponents = line.components(separatedBy: ",")
            if lineComponents.count >= 43 { // number of columns
                let medName = lineComponents[1]
                let sideEffects = Array(lineComponents[7..<47]) // Extract side effects colum
                sideEffectsData[medName] = sideEffects.map { $0.trimmingCharacters(in: .white
            }
        }
    } catch {
        print("Error reading CSV file:", error)
    }
}

func printSideEffectsForMedication(medName: String) {
```

```
    if let sideEffects = sideEffectsData[medName] {
        print("\nSide Effects for \(medName):")
        for effect in sideEffects {
            print("- \(effect)")
        }
    } else {
        print("No side effects found for \(medName).")
    }
}

func mainMenu() -> String {
    print("\n--- Main Menu ---")
    print("1. Register User or Login")
    print("2. Add Medication")
    print("3. Show Medication Schedule")
    print("4. Take Pills")
    print("5. View Pills History")
    print("6. Update Medication")
    print("7. Delete Medication")
    print("8. Print Side Effects for Medication")
    print("9. Exit")

    print("Enter choice: ", terminator: "")
    return readLine()!
}

func viewPillsHistory(user: String) {
    print("\n--- Pills History ---")
    if let userMedications = users[user]?["medications"] as? [[String: Any]] {
        for med in userMedications {
            let originalTotal = med["originalTotalPills"] as! Int
            let pillsTaken = med["doseTaken"] as! Int
            let pillsLeft = med["totalPills"] as! Int
            print("\(med["name"]!) - Original Total Pills: \(originalTotal), Total Pills Take
        }
    }
}

func deleteMedication(user: String) {
    print("\n--- Delete Medication ---")
    print("Enter the name of the medication you want to delete: ", terminator: "")
    let medName = readLine()!
    if var userMedications = users[user]?["medications"] as? [[String: Any]] {
        userMedications = userMedications.filter { ($0["name"] as? String)?.lowercased() != m
        users[user]?["medications"] = userMedications
        print("Medication \(medName) deleted successfully!\n")
    } else {
```

```
        print("No medications found for deletion.")
    }
}



func main() {
    // Load side effects data from CSV file
    readSideEffectsCSV(filePath: "/Users/khaznahalhajri/Desktop/aplprojectlvlv/aplprojectlvlv

    var currentUser: String? = nil
    while true {
        let choice = mainMenu()
        switch choice {
        case "1":
            currentUser = registerUser()
        case "2":
            if let currentUser = currentUser {
                addMedication(user: currentUser)
            } else {
                print("Please register a user first!")
            }
        case "3":
            if currentUser != nil {
                showSchedule(user: currentUser!)
            } else {
                print("No user registered!")
            }
        case "4":
            if currentUser != nil {
                takePills(user: currentUser!)
            } else {
                print("No user registered or medication added!")
            }
        case "5":
            if currentUser != nil {
                viewPillsHistory(user: currentUser!)
            } else {
                print("No user registered or medication added!")
            }
        case "6":
            if currentUser != nil {
                updateMedication(user: currentUser!)
            } else {
                print("No user registered or medication added!")
            }
        case "7":
```

```
            if currentUser != nil {
                deleteMedication(user: currentUser!)
            } else {
                print("No user registered!")
            }
        case "8":
            print("\n--- Side Effects ---")
            print("Enter the name of the medication: ", terminator: "")
            let medName = readLine()!
            printSideEffectsForMedication(medName: medName)
        case "9":
            print("Exiting program.")
            return
        default:
            print("Invalid choice, please try again!")
        }
    }
}

main()
```