Final Project Report

[EZMedi]

1. Executive Summary

EZMedi is an iOS application designed to assist patients with chronic diseases in identifying the medicines and managing their medications more effectively. The project successfully developed a user-friendly application that identifies medication through barcode scanning and provides an effective reminder system. The system aims to effectively allow patients with chronic diseases to better manage their medications and facilitate the process of medicine-taking. This reduces the risk of missing dosages and enhances medication adherence.

Despite facing challenges related to database integration and UI design, the project ended up with an easy-to-use interface and successfully grabbing data from the RxNorm database using Python through strategic planning and step-by-step problem solving, ultimately delivering a valuable tool for the users.

2. Introduction

By leveraging technology to streamline the process of identifying and tracking medication intake, EZMedi addresses a crucial need in the healthcare related field. The project's relevance lies in its potential to enhance treatment effectiveness and patient autonomy in medication management. EZMedi is developed to specifically address the needs of patients with chronic

diseases providing a valuable and reliable tool for medication management and access to medication information.

3. Project Objectives and Scope

Project Objectives: The system aims to effectively allow patients with chronic diseases to better manage their medications and facilitate the process of medicine-taking. Registered users will be able to update the patient's personal medical profile as well as generate reminders when needed. The system is primarily intended to connect users with a list of their own medications and reminders.

Project Scope: EZMedi's scope was confined to offering a personal medication library, barcode scanning for medication identification, search medicine name functionality using the RxNorm database, and setting medication reminders. It was not integrated with healthcare provider systems at this stage. The system will adhere to various regulatory requirements and policies, including HIPAA for data protection and ISO/IEC 27001 for information security management. Hardware limitations, such as camera quality for barcode scanning, are also taken into account, along with factors like CPU capabilities, available memory, and network connectivity. Some of the system's limitations also lie in its interactions with external systems, such as the RxNorm API, and its need to support parallel operation to serve multiple users simultaneously.

4. Achievements and Challenges

Project Achievements: Successfully developed a functional iOS application integrated with

barcode scanning and medicine name input feature. Achieved user-friendly design, effective reminder system, and ensured data privacy compliance using Google Firebase.

Challenges Faced: The main challenges included operating within a limited budget, ensuring data privacy, and integration with RxNorm Database.

5. Implementation Overview

Development Tools:

For the development stage, Xcode and Swift were chosen for their capabilities in iOS app creation. Swift's powerful syntax allowed us to create a responsive and user-friendly interface, also it is easier to implement the notification and barcode scanning feature with some of the existing functions. And Xcode facilitated the app's testing and debugging. Google Firebase, as the measure used to store the user-related information, provided a secure backend infrastructure, enabling real-time data synchronization and user authentication, including the verification of email and password format. The RxNorm database was integral for accessing reliable and up-to-date medication information, ensuring that users receive accurate data for their medication needs.

Code Highlights:

1. **Create new account and store the user information to firebase:**

```
private func createNewAccount (){
    FirebaseManager.shared.auth.createUser(withEmail: email, password: password){
        result, err in
        if let err = err {
            print("Failed to create user!", err)
            self.LoginStatusMessage = errorMessage(err)
            return
        }

        self.LoginStatusMessage = "Account created successfully"
        self.storeUserInformation()
    }
}

private func storeUserInformation(){
    guard let uid = FirebaseManager.shared.auth.currentUser?.uid else{
        return
    }

    print(uid,"this is uid")

    let userData = ["email": self.email, "uid": uid, "username": self.username]
    FirebaseManager.shared.firestore.collection("users")
        .document(uid).setData(userData){
            err in
            if let err = err{
                print(err)
```

2. **After searching a medicine and check it the user could choose to add the medication to their libraries (storeMedicine(medicine_ndc: String))**

```
private func storeMedicine(medicine_ndc: String) {

    guard let uid = FirebaseManager.shared.auth.currentUser?.uid else{
        return
    }

    FirebaseManager.shared.firestore.collection("users").document(uid)
        .getDocument { document, error in
            if let error = error {
                print("\(error)")
            } else if let document = document, document.exists{

                var updateLibrary: [String] = []

                if let existingLibrary = document.data()?["medicineLibrary"] as? [String]{
                    updateLibrary = existingLibrary

                    if existingLibrary.contains(medicine_ndc){
                        DispatchQueue.main.async {
                            self.vm.errorMessage = "Medicine already exists in library."
                            self.showAlert = true
                        }
                        return
                    } else {
                        self.vm.errorMessage = "You have successfully update your medicine library."
                        self.showAlert = true
                    }
                }

                updateLibrary.append(medicine_ndc)

                FirebaseManager.shared.firestore.collection("users").document(uid)
                    .setData(["medicineLibrary": updateLibrary], merge:true){ err in
                        if let err = err{
                            print("There is an error: \(err)")
                        } else {
                            print("Medicine stored successfully: \(medicine_ndc)")
                        }
                    }

            } else {
                FirebaseManager.shared.firestore.collection("users").document(uid)
                    .setData(["medicineLibrary": [medicine_ndc]], merge:true){ err in
                        if let err = err{
                            print(err)
                            return
                        } else {
                            print("Medicine stored successfully: \(medicine_ndc)")
                        }
                    }
            }
```

3. **Connect to an RxNorm API**

I choose the specific API that takes the string as a parameter and then returns RXCUI, the identification number given by the National Library.

I also use a dictionary to store each corresponding parameter of the API. I set the “name” parameter as the input given by the user. The allsrc parameter is set to 0 so that we only acquire the active concepts and disregard the expired ones. I also tuned the search mode to number 2 to activate a so-called best search mode.

```python
def get_RXCUI_byname(drug_name = name):
    # Input Name output RXCUI(the id in RxNorm)
    global log_file_path
    base_url = "https://rxnav.nlm.nih.gov"
    path = "/REST/rxcui.xml"
    query_params = {
        "name": drug_name,
        "allsrc": 0,
        "search": 2
    }
    url = f"{base_url}{path}"


    try:
        # try getting API
        response = requests.get(url, params=query_params)
        response.raise_for_status()  # raise http error
        print("Success 1")
        xml_data = response.content
    except requests.RequestException as error:
        # print the error
        print("Error: ", error)
        return None
```

The best search mode would first search for a medication that has the **exact** name that the user would input. If not found, it would search through its medicine library regardless of word order, upper or lower case, punctuations and suffixes. It would extend the abbreviations and revise the salt forms as well.

- Word order
- Case sensitivity
- Punctuation
- Suffixes that might be English inflections
- Abbreviations, e.g., "hctz" is expanded to "hydrochlorothiazide"
- Salts forms, e.g. "morphine sulfate" becomes "morphine"

4. Unit Testing of Searching Functions

4.1 The code of testing function

The following code is designed to test the scripts that call our API. I import the unittest package in python with many built-in function to assist our testing process. As unit testing cases we considered both name-based and barcode-based cases. I input various scenarios.

```python
import unittest
from Search import search_by_name, search_by_barcode

class TestEzmediApp(unittest.TestCase):

    def test_existed_drug_name(self):
        """ Test existed drug name """
        self.assertIsNotNone(search_by_name("Ibuprofen"))

    def test_nonexisted_drug_name(self):
        """ Test non-existed drug name """
        self.assertIsNotNone(search_by_name("Aspirin"))

    # Test Normalized Search
    def test_Upper_Case_drug_name(self):
        """ Test drug name with random Upper Case"""
        self.assertIsNotNone(search_by_name("LoRatADinE"))

    def test_Upper_Case_drug_name(self):
        """ Test drug name with typo"""
        # Should be prednisone
        self.assertIsNotNone(search_by_name("prednison"))

    def test_punctuated_head_drug_name(self):
        """ Test punctuated drug name """
        self.assertIsNotNone(search_by_name("!@#Sertraline"))

    def test_punctuated_tail_drug_name(self):
        """ Test punctuated drug name """
        self.assertIsNotNone(search_by_name("Lisinopril##"))
```

```python
    def test_long_drug_name(self):
        """ Test Long Names """
        self.assertIsNone(search_by_name("a" * 1000))

    # Testing With NDC Input
    def test_numeric_NDC(self):
        """ Test purely numeric NDC """
        self.assertIsNotNone(search_by_barcode("00904629161"))
    def test_ten_digits_NDC(self):
        """ Test NDC with '-' in between """
        self.assertIsNotNone(search_by_barcode("11523-7020-1"))
    def test_ten_digits_NDC(self):
        """ Test NDC with ten digits """
        self.assertIsNotNone(search_by_barcode("0781-1506-10"))
    def test_nine_digits_NDC(self):
        """ Test NDC with nine digits """
        self.assertIsNotNone(search_by_barcode("11523-7020"))
    def test_eight_digits_NDC(self):
        """ Test NDC with eight digits """
        self.assertIsNotNone(search_by_barcode("0071-0157"))

    # Should return None
    def test_abnormal_digits_NDC(self):
        """ Test NDC with a lot of digits """
        self.assertIsNone(search_by_barcode("0"*100))
        self.assertIsNone(search_by_barcode("0"))
```

```python
    def test_punctuated_middle_drug_name(self):
        """ Test punctuated drug name """
        self.assertIsNotNone(search_by_name("Lisin@@opril"))

    def test_abbreviation(self):
        """ Test Abbreviation Names """
        # MTX is the abbreviation for "Methotrexate"
        self.assertIsNotNone(search_by_name("MTX"))

    def test_saltforms(self):
        """ Test Overspecified Names """
        # Should be revised to for "oxycodone"
        self.assertIsNotNone(search_by_name("oxycodone-hydrochloride"))


    # Should Return None
    def test_invalid_drug_name(self):
        """ Test invalid drug name """
        self.assertIsNone(search_by_name("InvalidDrugName"))

    def test_empty_drug_name(self):
        """ Test empty drug name """
        self.assertIsNone(search_by_name(""))

    def test_numeric_drug_name(self):
        """ Test Name with pure numbers """
        self.assertIsNone(search_by_name("1234"))
```

4.2 The results of unit testing

16 out of 17 tests are achieved. The only test case that we did not manage is searching with names that have special character within the input. We can not disregard all inputs without special character because normalized searching includes punctuations from the beginning and the end. We have not yet figured out how to resolve such a situation and aim to improve this in the future.

```
======================================================================
FAIL: test_punctuated_middle_drug_name (__main__.TestEzmediApp)
Test punctuated drug name
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\test\SWE\Testing.py", line 34, in test_punctuated_middle_drug_name
    self.assertIsNotNone(search_by_name("Lisin@@opril"))
AssertionError: unexpectedly None

----------------------------------------------------------------------
Ran 17 tests in 34.227s

FAILED (failures=1)
```

6. Live Demonstration

Demonstration Overview:

For the live demonstration, the EZMedi iOS application will be presented. First, the team would present the login and logout features with user registration. Then present the main functionalities including name input searching, barcode scanning, personal medication library, and the reminder page.

7. Conclusion

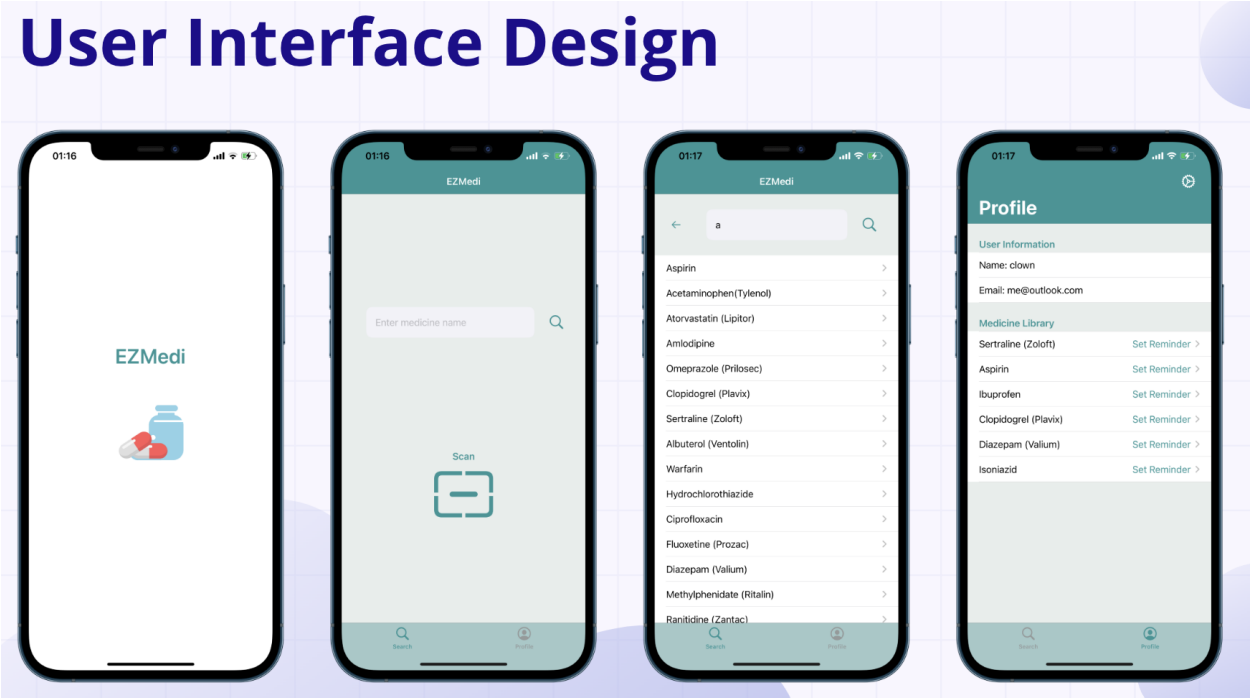
EZMedi's main achievement lies in the smooth usage and intuitive user interface design. Its successful implementation of the main functionalities, which includes the development of the reminder which enables efficient management of medication reminders setting. The integration of the Google Firebase Database allows the system to store user information and protect user privacy effectively.

The challenges faced during the project were turned into learning opportunities, leading to a more resilient and user-focused application. The dedication of the team to overcome obstacles related to data privacy, budget constraints, and the need for continuous operation has resulted in an application that not only meets but exceeds user expectations in terms of functionality, security, and reliability.

The improvement in the near future would be the actual integration of the RxNorm API into the Swift frontend, either through directly implementing in the swift code or connecting between the Python backend code, which is already implemented successfully, and the Swift frontend code. The reminder could be improved by adding more flexibility on the date range and the frequency of the reminder. Also, since the RxNorm database only includes the NDC code, the system should connect with the database which could convert the NDC code into the UPC-A barcode to make it more user friendly.

8. Appendices

Include any supplementary materials, additional documentation, or detailed information that supports the content presented in the report.



9. References

Zanjal, Samir V., and Girish R. Talmale. "Medicine reminder and monitoring system for secure health using IOT." Procedia Computer Science 78 (2016): 471-476. doi.org/10.1016/j.procs.2016.02.090.

"Swift - A Powerful Open Language That Lets Everyone Build Amazing Apps." Apple Developer, Apple Inc., developer.apple.com/documentation/swift/.

Color Palette: #ECF4D6 #9AD0C2 #2D9596 #265073

Build a Live Barcode and Text Scanner iOS App with SwiftUI & VisionKit

2021 SwiftUI Tutorial for Beginners (3.5 hour Masterclass)

SwiftUI Firebase Chat 11: Send and Save Messages to Firestore

Unit Testing a SwiftUI application in Xcode | Advanced Learning #17

Swift: Create Tab Bar Controller Programmatically (Swift 5, Xcode 11) - 2020 iOS

iOS Swift Tutorial: Use APIs with Swift UI & Build a Book Barcode Scanner