هەرێمی کوردستان - عێراق
زانکۆی جیهان
کۆلێژی زانست
بەشی زانستی کۆمپیوتەر

اقليم كوردستان – العراق
جامعة جيهان
كلية العلوم
قسم علوم الحاسبات

Kurdistan Region – Iraq
Cihan University
College of Science
Department of Computer Science

Transforming Pharmacy Management: A Mobile Application System for Improving Operational Efficiency

A graduation project submitted to

Department of Computer Science/College of Science

In Partial Fulfillment of the Requirements

For the Degree of Bachelor of Science

In Computer Science

By

Brivana Ahmed

Mustafa Ahmed

Belal Akram

Supervised by:

Asst. Lect. Alyaa Asaad

(2023-2024)

Supervisor's Certificate

I certify that this project entitled "**Pharmacy Management System Mobile App**" is prepared under my supervision by the students: (Belal Akram, Brivana Ahmed, Mustafa Ahmed) at the Department of Computer Science, College of Science, Cihan University - Erbil in partial requirement for the degree of Bachelor of Science in Computer Science.

Supervisor

Asst. Lect. Alyaa Asaad

/ / 2024

Certificate

We certify that this graduation project research "**Pharmacy Management System Mobile App**" was read and examined the students: (Belal Akram, Brivana Ahmed, Mustafa Ahmed) in its contents and in what is related to in our opinion it meets the standards of a graduation project research for the degree of BSc. in Computer Science.

Examination Committee

Head of Committee	**Member of Committee**
Signature:	Signature:
Name:	Name:
Date:	Date:

Supervisor

Signature:

Name:

Date:

Head of Department

Signature:

Name:

Date:

Acknowledgment

We extend our heartfelt gratitude to our supervisor, Asst. Lect. Alyaa Asaad, whose unwavering guidance and encouragement have been invaluable throughout every stage of this endeavor. Asst. Lect. Alyaa Asaad expertise and constructive feedback have played a crucial role in refining our ideas and ensuring the highest quality of our work.

Our appreciation also goes to our families and friends for their unyielding support in myriad ways. Whether by lending an ear to our ideas, serving as a sounding board for our thoughts, or offering words of encouragement when we needed them most, their love and support have been the cornerstone of this project's success.

We wish to acknowledge the dedicated staff and faculty at the Computer Science Department of Cihan University - Erbil, who have fostered a rich and stimulating learning environment. The numerous opportunities to learn from and collaborate with fellow students, coupled with the resources and facilities provided, have greatly contributed to our academic journey.

Lastly, our sincere gratitude extends to all individuals who participated in our research or shared valuable information and insights. Your contributions have been instrumental in shaping our understanding of the topic and informing our conclusions.

Thank you all for your invaluable help and unwavering support.

Warm Regards,

Mustafa , Brivana & Belal

Abstract

The proposed research project explores the development of a Pharmacy Management System Mobile App to enhance the efficiency and accessibility of pharmacy operations where the traditional pharmacy management systems often face challenges related to cumbersome processes, limited mobility, and constrained communication channels.

The integration of mobile technology into pharmacy management by using Flutter for designing the app and Firebase for handling the data, with the goal of making the app work smoothly on different mobile devices seeks to overcome these limitations by offering a user-friendly platform that leverages of the smartphones, results show that the app fits well into the daily activities of pharmacies, making them more accurate and faster.

This app aims to provide real-time access to critical information, streamline inventory management, and facilitate improved communication channels for pharmacy professionals. By incorporating a mobile-centric approach, the project contributes to the ongoing discourse on optimizing healthcare services through innovative technological solutions and represents a significant step towards modernizing pharmacy operations.

Table of Content

Table of Figures

Chapter One: Introduction

1.1 Overview

The pharmaceutical landscape is continuously evolving, driven by advancements in technology and an increasing demand for streamlined healthcare services. As part of this evolution, the integration of mobile technology into pharmacy management systems has emerged as a promising solution to enhance efficiency and accessibility.

Healthcare practice relies heavily on evidence-based decisions and requires the use of quality healthcare data. Health Management Information Systems (HMIS) are among the core elements of health system building blocks. However, setting reveals a lack of adequate information regarding the quality of health information data [1]. One of the major technologies that aids contemporary healthcare solutions is smart and connected wearables [2]. Smart healthcare heralds a multi-level transformation in the medical industry, encompassing shifts in the medical model from disease-centric to patient-centred care, the construction of informatization from clinical to regional medical sectors, and a paradigm shift in medical management from general to personalized approaches. The profound impact of information technologies since the mid-1990s extends beyond academia to shape global economic growth.

In various industries, including pharmacy, these technologies have been recognized as pivotal drivers of progress. The integration of technology and the internet into pharmacy practice marks a significant milestone, with technology and automation playing instrumental roles in streamlining business processes.

Information technologies have permeated diverse aspects of pharmacy practice, from oncology to pharmacokinetics, facilitated by the widespread availability of the internet and the proliferation of healthcare-specific smartphone applications. This pervasive adoption of IT in pharmaceutical practices underscores its growing significance and relevance in optimizing pharmacy operations and enhancing patient care.

Improving the performance of inventory control systems of healthcare items minimizes inventory-related costs, potentially leading to a reduction in the cost of treatment and providing overall satisfaction to patients [3].

Traditional pharmacy management systems often face challenges related to cumbersome processes, limited mobility, and constrained communication channels. Recognizing the need for

innovation, this research project delves into the development of a Pharmacy Management System Mobile App.

This app aims to leverage the ubiquity of mobile devices to provide a dynamic and user-friendly platform for managing various aspects of pharmacy operations. The ubiquity of smartphones and the prevalence of mobile applications in various industries underscore the potential benefits of adopting a mobile-centric approach in pharmacy management.

By seamlessly integrating into the workflow of pharmacy professionals, the proposed mobile app seeks to overcome the limitations of traditional systems, offering real-time access to information, inventory management, and improved communication channels. In this backdrop, the research explores the intersection of mobile technology and pharmacy management, aiming to contribute to the ongoing discourse on optimizing healthcare services through innovative technological solutions.

1.2 Current Challenges in Pharmacy Management Systems and Enhancement opportunity

The "digital revolution," or the move from analog to digital technology, has changed our lives and opened up new opportunities in a variety of societal activities. [4].

One of the primary reasons for launching this app is to facilitate the pharmacist's management system. Many pharmacists face problem that is related to medicine expiration date, quantity and side effects. Each pharmacist has to buy, get or look for the medicine it will take time.

Pharmacists consider it hard to keep accurate records and successfully manage stock levels since there's a lack of synchronization caused by paper-based operations and traditional methods for inventory management.

1.3 Objectives of Developing Mobile Application for Pharmacies

This application is specifically developed to streamline and simplify everyday pharmacy operations. Helping the pharmacists aiming to make their work as effortless as possible.

One key feature is the notification system: pharmacists receive automatic alerts about the availability of products in stock, eliminating the need for manual inventory checks. Furthermore, the app tracks expiration dates and proactively notifies pharmacists of any products nearing expiration or already expired, ensuring that outdated medications are promptly removed from circulation.

Furthermore, the app makes the pharmacy more accessible by offering a user-friendly layout with easily accessible critical information. This design contributes to more dependable and effective pharmacy administration by cutting down on the amount of time pharmacists must spend on repetitive duties and lowering the possibility of errors.

The main purpose of this research project is to develop a mobile app for managing pharmacies that solves problems found in traditional pharmacy systems. By using advanced mobile technology, this app aims to provide a complete solution that makes operations run smoother, improves the accuracy of information, and ensures that patients receive safe care throughout their treatment.

Chapter Two: Literature Review

2.1 Overview

Mobile applications have become integral tools in various industries, including healthcare. This literature review explores the existing body of knowledge related to mobile applications in pharmacy management systems, shedding light on their impact, challenges, and potential benefits.

The literature review offers a comprehensive exploration of the development and impact of Pharmacy Management Systems (PMS) and Information Systems (IS) in healthcare contexts, Health information systems offer many potential benefits for healthcare, including financial benefits and for improving the quality of patient care [5].

delve into the challenges and strategies associated with implementing PMS in resource-limited settings, illuminating effective system deployment techniques. In the context of increasing deployments of unified pharmacy management systems and the merging of hospitals into single health systems, pharmacy informatics (PI) teams face growing opportunities and challenges.

Pharmacy leaders must carefully consider the impact of technology utilization within multihospital health systems, taking into account organizational structures and their effects on technology implementations and dedicated support teams. Common challenges in implementing electronic medical records (EMR) and other technologies in multihospital systems include harmonizing practices, addressing platform compatibility, and ensuring interoperability. Collaboration between PI and information technology teams is crucial to develop practical strategies for implementing pharmacy automation and software, thereby facilitating safe and effective patient care. Key areas of focus include organizational structures impacting informatics teams, pharmacy integration and standardization, formulary management, data analytics, and clinical decision support systems. Addressing these challenges is essential for ensuring successful implementation and utilization of pharmacy management systems in the evolving healthcare landscape. [6].

The literature highlights the critical need to address the unique challenges and requirements faced by pharmacy management systems, particularly in regions like the

North of Iraq. With ongoing advancements in technology, it becomes increasingly vital to enhance access to and usability of Pharmacy Management System (PMS) solutions, especially in resource-constrained settings. By improving the availability and usability of PMS solutions, pharmacy operations can be optimized, ultimately leading to better patient care outcomes. In the context of this research

across different regions of the world, showcasing a wide range of functionalities tailored to meet the unique needs of pharmacies in various settings. These systems encompass features such as inventory management, prescription processing, patient records management, and billing functionalities, providing comprehensive support for pharmacy workflow automation, every pharmacy strives to reach out to its most valued clients; therefore, this industry uses new technologies in this respect to match the correct sales method and improve consumer satisfaction [7].

Chapter Three: Methodology

3.1 System Architecture and Design

After we have laid the groundwork for our research strategy, we must design the Pharmacy Management System's architecture. The application's overall structure and its interconnected parts are shown in the system structure design that follows.

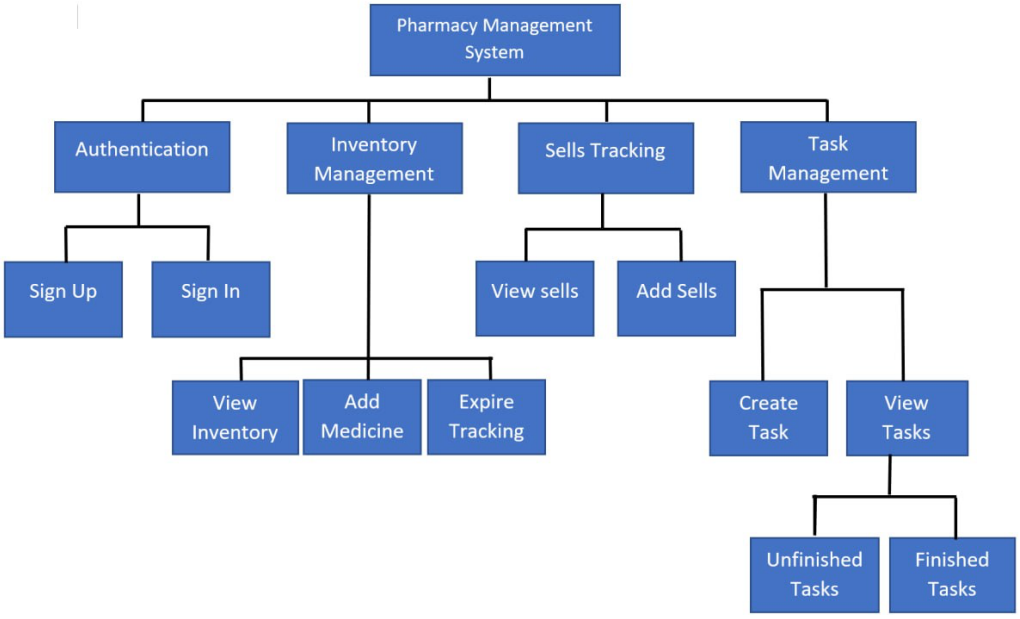

Figure 3-1 System Architecture and Design

This graphic shows the interactions between the four modules of the application authentication, inventory management, sales tracking, and task management and gives a macro view of its architecture. Prior to delving into the particulars of the technologies and development tools used to create the system, it is imperative that we comprehend its design.

After providing an overview, the chapter will go into detail about the development environment and the individual technologies used, including the frontend Flutter, the backend programming language Dart, the backend services Firebase, and the integrated development environment Android Studio. The contributions of each technology to the functionality of the relevant modules shown in the system design are explained in these sections.

3.2 System Main Requirements:

This study investigates the software architecture and technologies utilized in a mobile application development project, with a primary focus on the integration of Dart programming language constituting 60% of the codebase, alongside Flutter framework and Firebase backend services. Through a detailed analysis of the codebase, it is found that Dart serves as the foundational element, offering simplicity, performance, and robustness, while Flutter complements it by facilitating the creation of visually appealing and responsive user interfaces across multiple platforms. The incorporation of Firebase backend services enhances data management and synchronization, contributing to the application's scalability and real-time capabilities. This research underscores the significance of thoughtful technology selection and integration in achieving desired outcomes in mobile application development, providing insights into contemporary practices in software architecture and technology utilization within the domain.

3.3 Introduction to flutter

Flutter is an open-source UI framework developed by Google in May 2017 at the Google I/O developer conference. that allows developers to build native-quality mobile, web, and desktop applications from a single codebase. Flutter is a valuable modern tool used to create stunning cross-platform applications that render native code on each device and OS. Flutter is compatible with Android, iOS, Linux, Windows, etc. [10]

3.3.1 key feature of flutter

Its core strength lies in its ability to expedite development through a variety of features. Firstly, Flutter streamlines the process by allowing developers to write code once and deploy it seamlessly across diverse platforms, including iOS, Android, web, and desktop environments. This is made possible through its utilization of the Dart programming language, also developed by Google. Additionally, Flutter offers a rapid development cycle with its hot reload functionality, enabling developers to instantly visualize and implement changes to the codebase without the need for restarting the application, thus significantly accelerating the iteration process. Its expressive and flexible UI toolkit empowers developers to craft visually stunning and customizable user interfaces

using a rich set of widgets. These widgets can be effortlessly tailored to adhere to the design language of specific platforms or brands, ensuring a consistent and polished user experience.

3.4 Strength of Using Dart Programing Language

Dart is an open-source programming language that is general-purpose, object-oriented, and has C-style syntax developed by Google in 2011. Dart programming aims to build UIs for web and mobile applications. It is continuously being enhanced and supports compiling into native machine code, enabling it to create mobile apps. Dart borrows elements from Java, JavaScript, C#, among others, and can be said to be a strongly typed language. [11]

3.5 Utilizing Android Studio for Mobile Application Development:

This section describes how to create Android apps with Google's Android Studio, an integrated development environment (IDE) that offers all the tools you need to create apps quickly and effectively.

Installation and First Setup: Downloading and installing Android Studio from the official website is the first step in the procedure. When creating a new project, developers first choose an appropriate template and set up the project name, package name, and programming language (Java or Kotlin).

Development Process: To write and polish code, developers utilize Android Studio's sophisticated code editor, which offers real-time error detection and code completion among other capabilities. The IDE makes it possible to develop user interfaces using both visual editing and XML coding.

Testing and Optimization: To test apps in a variety of scenarios, Android Studio provides an emulator, or developers can utilize a real device. Debugging and performance analysis tools are essential for maximizing the functionality of the program.

Deployment: The application is ready for release once it satisfies all requirements. Android Studio makes it easier to package and distribute apps by supporting version control systems, project management, library and SDK integration, and project integration. [12]

3.6 Integrating Firebase for Backend of Pharmacy Mobile Application:

This section describes how to create a pharmacy management system using Firebase, a cloud-based platform from Google. A set of resources and services offered by Firebase make it easier to construct web and mobile applications.

Utilized Core Firebase Services:

- ✓ Authentication: Controls safe user registrations and logins.
- ✓ Cloud Firestore: Provides an instantaneous database for effective data synchronization and storage.
- ✓ Storage: Makes it easier to save and retrieve photos from user profiles.

Choosing Reasons for utilizing Firebase:

- ✓ Firebase is selected due to its extensive feature set that improves the functionality of applications:
- ✓ Real-time data synchronization is essential to the timeliness and correctness of the system's data since it guarantees instantaneous updates across user interfaces.
- ✓ Scalability and Security: As the number of users increases, the system is supported with scalable solutions and strong security for handling user data.
- ✓ Analytics and Cloud Functions: Utilizing serverless computing, this technology streamlines application logic and provides insights into user activity.

Benefits of Integration: Firebase's smooth integration with other Google services improves the effectiveness and responsiveness of the application. For a top-notch user experience, this comprises functions like real-time updates, tailored content distribution, and ongoing performance improvement.

By offering crucial services for data storage, security, and application scalability, Firebase's integration greatly improves the pharmacy management system and advances the objective of productive and successful application development and maintenance. [13]

3.6.1. Firebase Configuration Requirements:

Firebase was used to connect to and configure the pharmacy management system mobile application's backend through a number of necessary processes. The first step in the procedure was creating a Google account and going to firebase.com. By choosing the "Add project" button and giving the project a name that complies with particular character and symbol requirements, a new project was started inside the Firebase console. In order to provide critical performance insights, Google Analytics was added as a required component by choosing a default account and modifying the tracking parameters to match the goals of the application.

To establish a real connection to the program, required to navigate to the Firebase dashboard, click the Android icon, and then input the package name of the application, which could be located in the build.gradle file. This stage also allowed the optional addition of a SHA-1 debug signing certificate and a moniker. After that, the google-services.json file was carefully downloaded and added to the app's designated directory, making sure the file name stayed the same to avoid connection issues.

The last step was to update the build.gradle file for the application to include the dependencies and plugins required by the Firebase SDK. Firebase was smoothly incorporated into the pharmacy management system mobile application thanks to this thorough setup process, which also provided a scalable, secure, and effective backend

infrastructure with essential metrics for continued performance. The required steps were done as shown in the figures below:

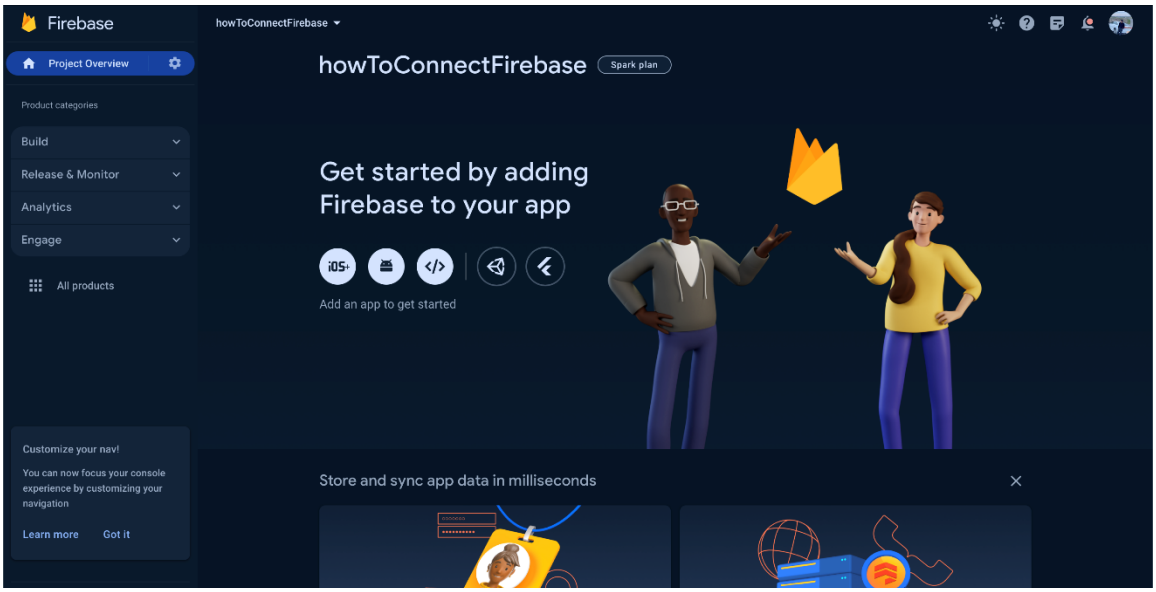

Figure 3-2 project dashboard on firebase

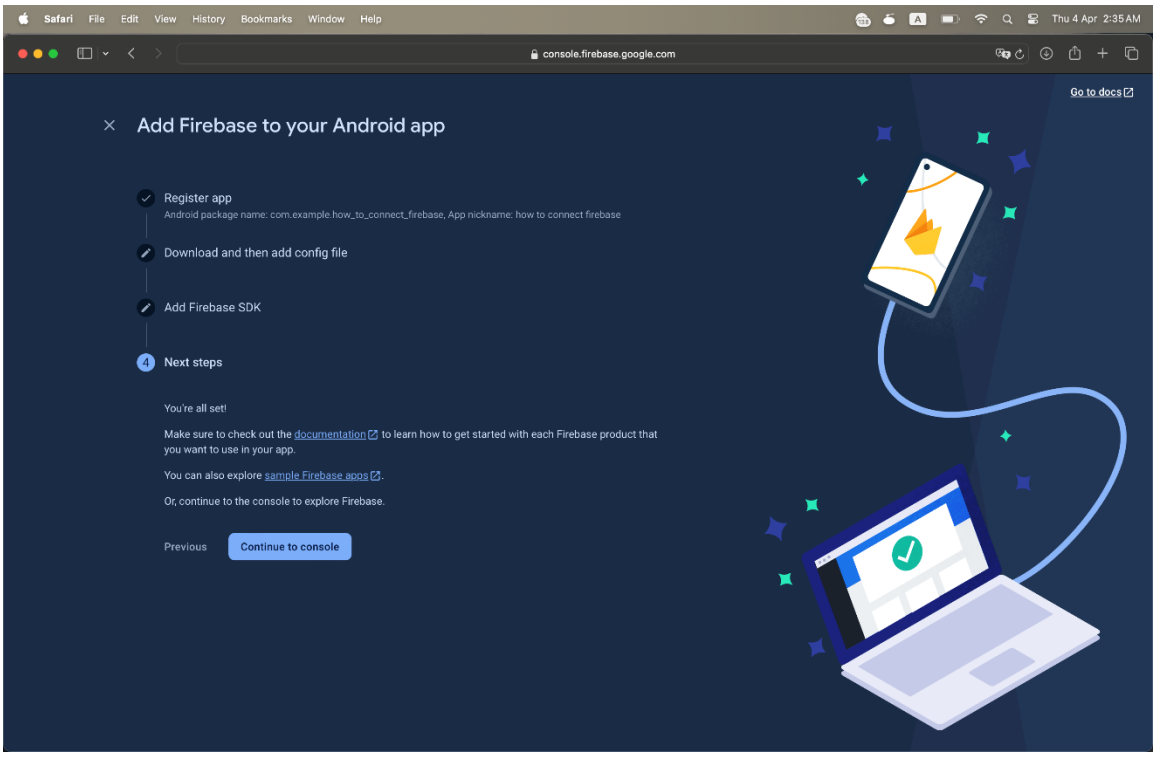

Figure 3-3 complete configuring Firebase to flutter app

Chapter Four: Implementation and Result

4.1 Implantation Overview:

This thesis's implementation chapter examines the actual creation and distribution of the Pharmacy Management System mobile application, which is intended to greatly improve pharmacy operations. Everything from the basic setup and safe user authentication to the most important functions like inventory management, sales tracking, and expiration tracking is covered in depth. The chapter emphasizes the combination of Android Studio for thorough development efforts, Firebase for reliable backend services, and Flutter for user-friendly UI design. Utilizing these technologies, a user-friendly and responsive system that effectively oversees pharmacy operations was created. The chapter also describes the selection of "flutter_launcher_icons: ~0.13.1" as a unique app icon that represents the main principles and goals of the Pharm Assist initiative, acting as a noteworthy emblem within the Application

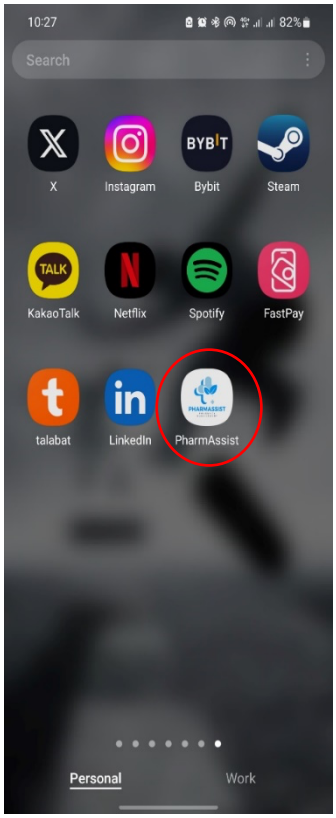

Figure 4-1 Application Icon

4.2 User Authentication Process

This section outlines the user authentication process within the Pharmacy Management System application, crucial for ensuring secure access and maintaining the integrity of user data. Upon launching the app, pharmacists are prompted to log in using their existing credentials—email and password—or through a Google sign-in for added convenience. For new users, the app provides an option to register by clicking the signup button, where they must fill in details such as their username, email, password, and confirm the password to create an account.

Upon successful login, the application utilizes Firebase's auth function to authenticate users and securely manage sessions. The system captures and stores the user's email as a unique identifier (UserId) to facilitate personalized interactions and secure access to user-specific data within the app. For those creating a new account, the registration process is similarly safeguarded with Firebase Authentication, ensuring that all user information is securely managed and integrity-checked before being stored. This robust authentication framework supports the application's security protocols, thereby safeguarding pharmacy operations and sensitive data management.

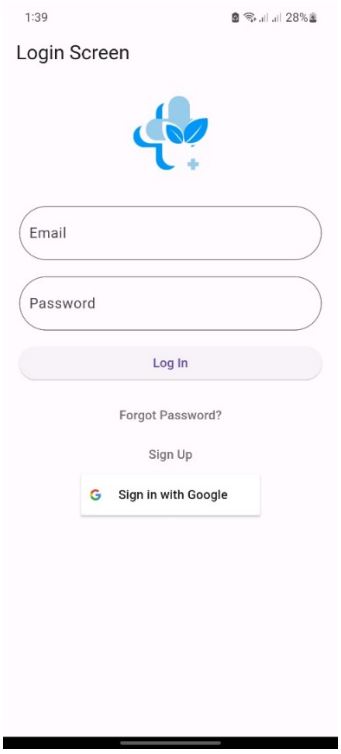

Figure 4-2 Login Page

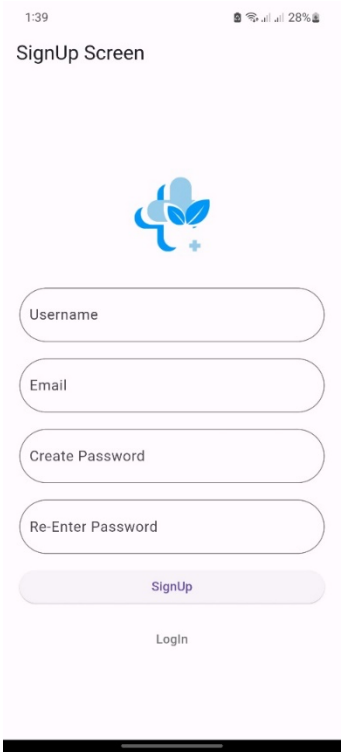

Figure 4-3 Signup Page

Here if the email and password is correct , we save the email as UserId using userProvider

```
UserProvider userProvider = Provider.of<UserProvider>(
    context, listen: false);
userProvider.setUserId(email);
userProvider.setPharmacyId(pharmacyId);
// Pass user email to Dashbord widget
Navigator.pushReplacement(
  context,
  MaterialPageRoute(
    builder: (context) => DashboardPage(),
  ),  // MaterialPageRoute
);
```

Figure 4-4 save the userId , PharmacyId

If the pharmacist doesn't have their account they should fill the fields in order to create new account in this app, they should enter their username, email, password and re-enter password, then click on signup.

Here we use firebase 'auth' function to create the user account in firebase Authentication

```
await _auth.createUserWithEmailAndPassword(
    email: email, password: password);

// Create a new document in Firestore with the email as
await _firestore.collection('users').doc(email).set({
  'user': user,
  'email': email,
  'name' : 'name',
  'pharmacyId' : ''
  // Add more fields as needed
});
```

Figure 4-5 adding the user info to the firebase

If the pharmacists forget their password by clicking on forget my password in the login page it will navigate them to this screen so they can reset their password by entering their email and click on send a reset password to your email.

Code : To send a password change email we use

await_auth.sendPasswordResetEmail(email: email); as shown in figure No. (8).

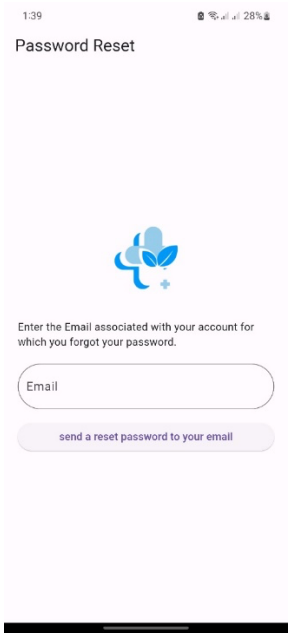

Figure 4-6 Forget Password Page

4.3 Dashboard for the Core Functionalities of Pharmacy Management

Users who log into the Pharmacy Management System are greeted with a dashboard that compiles all of the data they need to run a pharmacy effectively. The dashboard has the following important sections:

- ✓ **Medicine Inventory**: This feature helps pharmacists maintain track of their inventory without the need for manual counting by displaying current stock levels and retrieving real-time data from Firebase using the getMedicinesCount() function.
- ✓ Pharmacists can monitor and record sales with the use of sales tracking, which helps with day-to-day operations and inventory selections.
- ✓ **Expiration Alerts:** To ensure drug safety and compliance, this feature uses the getExpiringCount() function to alert users when a medication is going to expire or has already done so.
- ✓ **A chart showing the sales** over the last seven days is also included on the dashboard, giving a brief visual summary of the operation of the company.

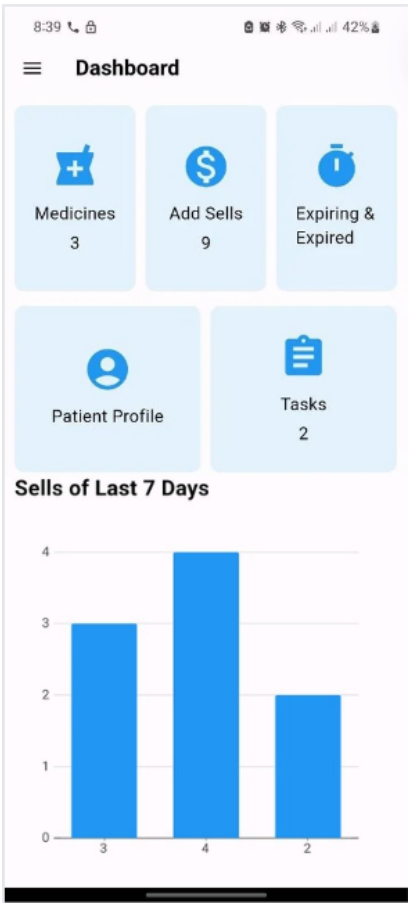

Figure 4-7 Main Dashboard

First, when the user login the first page is dashboard page this page shows the pharmacists all information that they need, first widget is Medicines shows the inventory how much the pharmacy has in their stocks. Sells the pharmacists can use it when they sell the drugs to the patients. Expiration show you the expired medicines or the one that are near to be expired. Here we implement a chart to help the pharmacists see the last seven days of sales that help the pharmacists do their work with a professional way.

```dart
Future<int> getMedicinesCount() async {
  final QuerySnapshot snapshot = await FirebaseFirestore.instance.
  collection('pharmacies').
  doc(pharmacyId).
  collection('medicines').
  get();

  return snapshot.docs.length;
}
```

Figure 4-8 fetch Medicine count

Code: to get how many medicines from firebase database we use function getMedicinesCount(), that showing below:

```dart
Future<int> getExpiringCount() async {
  // Get today's date
  DateTime now = DateTime.now();

  // Define the start and end dates for the range (e.g., 24 hours before and
  DateTime startDate = DateTime(now.year, now.month, now.day - 1); // 24 hou
  DateTime endDate = DateTime(now.year, now.month, now.day + 1); // 24 hours

  // Query shipments within the date range
  QuerySnapshot shipmentsSnapshot = await FirebaseFirestore.instance
      .collection('pharmacies')
      .doc(pharmacyId)
      .collection('medicines')
      .where('shipments.date', isGreaterThanOrEqualTo:
       startDate, isLessThan: endDate)
      .get();

  // Initialize the count of medicines
  int medicinesCount = 0;

  // Iterate over each document in the shipments collection
  for (QueryDocumentSnapshot doc in shipmentsSnapshot.docs) {
    // Get the list of shipments for the current document
    List<dynamic> shipments = doc['shipments'];

    // Iterate over each shipment in the list
    for (dynamic shipment in shipments) {
      // Extract the shipment date
      DateTime shipmentDate = DateTime.parse(shipment['date']);

      // Check if the shipment date is within the specified range
      if (shipmentDate.isAfter(startDate) && shipmentDate.isBefore(endDate))
        // Increment the count of medicines associated with this shipment
        medicinesCount++;
      }
    }
  }

  return medicinesCount;
}
```

we use getExpiringCount() function

to get expiring count and put it in the expiring and expired container.

Figure 4-9 fetch Expiring medicine

4.4 Inventory Management

This Module of the Pharmacy Management System created to simplify the difficult process of controlling pharmaceutical supply is the Inventory Management module. With the help of this advanced system, pharmacists can monitor their inventory levels in real time, which helps them manage stock effectively, cut down on waste, and streamline ordering procedures.

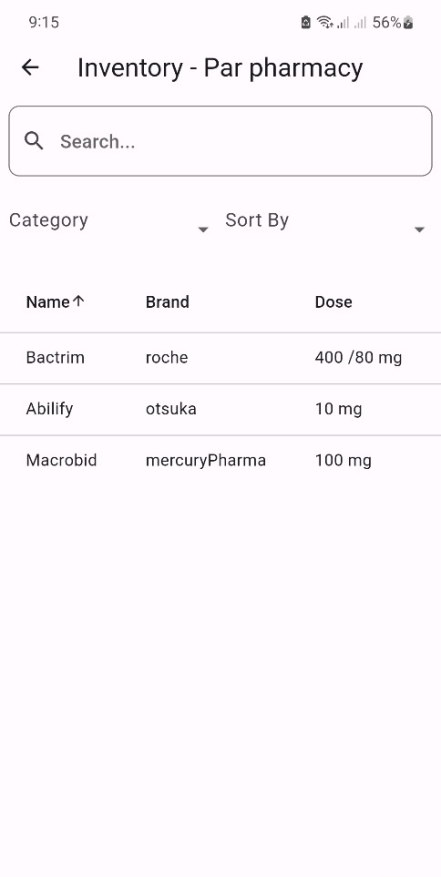

Figure 5 Inventory module

```dart
// Fetch all medicines for the pharmacyId
final querySnapshot = await FirebaseFirestore.instance
    .collection('pharmacies')
    .doc(pharmacyId)
    .collection('medicines')
    .get();

final List<Map<String, dynamic>> newData = [];

for (var doc in querySnapshot.docs) {
  final id = doc.id;
  final data = doc.data();
  final name = data['Name'] ?? '';
  final dose = data['Dose'] ?? '';
  final brand = data['Brand'] ?? '';
  final shipments = data['shipments'] ?? [];

  newData.add({
    'id': id,
    'Name': name,
    'Dose': dose,
    'Brand': brand,
    'Shipments': shipments,
  });
}
```

Figure 4-11 showing of fetch medicines function

This is the inventory page that show us all the medicine that the pharmacist inserts it. They can search the item in the search box or they can do filter The pharmacists can either insert it manually or scan the medicine by clicking on the button right icon, this page show how much of this medicine available in the stock.

To get all the medicines in in the pharmacy we use function _fetchMedicines()

Then it will show add it to the newData array then it will display as shown below:

Figure 4-12 Scanning The product

While, the pharmacists have the choice to choose wither they insert the medicines manually or by scan if they choose scan the camera will open to them and they start to scan the medicine using the camera.

```dart
Future<void> _scanBarcode() async {
  String barcodeScanRes = await FlutterBarcodeScanner.scanBarcode(
    '#ff6666', // Scanner overlay color
    'Cancel', // Cancel button text
    true, // Use flash
    ScanMode.BARCODE, // Scan mode
  );

  if (!mounted) return;

  setState(() {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Scanned Barcode: $barcodeScanRes $widget.pharmacyId'),
        duration: const Duration(seconds: 3), // Adjust the duration as needed
      ),  // SnackBar
    );

    _showAddMedicineDialog(barcodeScanRes );
  });
}
```

Figure 4-13 scanBarcode function

The _scanBarcode function using "flutter_barcode_scanner: ^2.0.0" so it can open the camera using (after getting permission to do so), and will look for a barcode as soon as it get the Barcode it will call the function _showAddMedicineDialog(), the dialog is showing Figure 30.

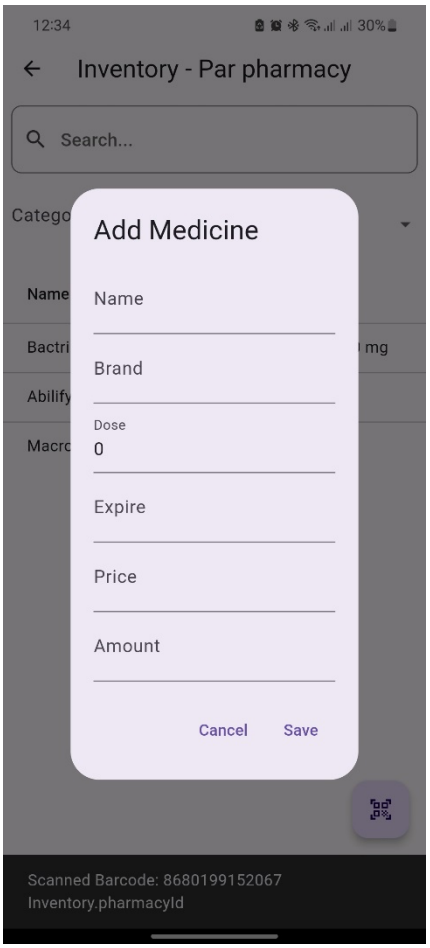

Figure 4-14 Adding Product

After, scanning the medicine this page will show to enter the information of the medicine like the name of the medicine brand dose expiration price and amount this information help of tracking the medicine throw this application.

4.5 Sales Management

Pharmacists are able to observe how daily sales affect their inventory thanks to the dashboard's smooth integration with the sales monitoring system. This integration makes it easier to identify slow-moving items that might need to be restocked frequently and fast-moving commodities that might benefit from discounts or promotional tactics to move inventory before it expires.

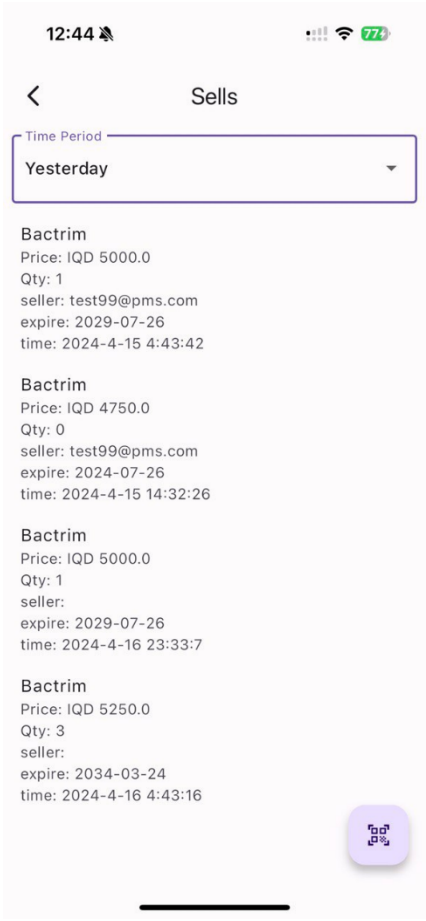

Figure 4-15 Sells Manager

The Sells history here is the medicines that the pharmacists he sell all the information of the medicines that sell the past few days or either yesterday, and the pharmacists can sell new product to the patient by scanning the product barcode in order to sell the products and they will be in a one place so they can track the usage of the medicines that they sell.

```dart
Future<List<Map<String, dynamic>>> fetchSellsData( {String? selectedDate}) async {
  selectedDate ??= DateTime.now().toString().substring(0, 10);; // Use current date if no date is pr

  try {
    if (selectedDate == '0') {
      // Retrieve all sells
      QuerySnapshot sellsQuerySnapshot = await FirebaseFirestore.instance
          .collection('pharmacies')
          .doc(pharmacyId)
          .collection('sells')
          .get();

      List<Map<String, dynamic>> allSellsData = [];
      for (QueryDocumentSnapshot doc in sellsQuerySnapshot.docs) {
        QuerySnapshot dailySellsQuerySnapshot = await doc.reference.collection('dailySells').get();
        dailySellsQuerySnapshot.docs.forEach((dailyDoc) {
          allSellsData.add(dailyDoc.data() as Map<String, dynamic>);
        });
      }
      if (allSellsData.isNotEmpty) {
        return allSellsData;
      } else {
        return [];
      }
    } else if (selectedDate == '7') {
      // Retrieve sells data for the last 7 days
      List<Map<String, dynamic>> sellsData = [];
      for (int i = 0; i < 7; i++) {
        // Calculate the date i days ago
        DateTime date = DateTime.now().subtract(Duration(days: i));
        String dateString = date.toString().substring(0, 10);

        QuerySnapshot dailySellsQuerySnapshot = await FirebaseFirestore.instance
            .collection('pharmacies')
            .doc(pharmacyId)
            .collection('sells')
            .doc(dateString)
            .collection('dailySells')
            .get();

        dailySellsQuerySnapshot.docs.forEach((doc) {
          sellsData.add(doc.data() as Map<String, dynamic>);
        });
       // print(sellsData);
      }

      if (sellsData.isNotEmpty) {
        return sellsData;
      } else {
        return [];
      }
    } else {
      // Retrieve sells data for the specified date
      QuerySnapshot dailySellsQuerySnapshot = await FirebaseFirestore.instance
          .collection('pharmacies')
          .doc(pharmacyId)
          .collection('sells')
          .doc(selectedDate)
          .collection('dailySells')
          .get();

      List<Map<String, dynamic>> sellsData = [];
      dailySellsQuerySnapshot.docs.forEach((doc) {
        sellsData.add(doc.data() as Map<String, dynamic>);
      });
```

Figure 4-16 fetch sells data function

When, the pharmacists need to sell the product they have to scan the products in order to put it in the application so everything will be organized. After, the pharmacists scan the product they have to check the expiration and the quantity that they sell to the patient and the price of the medicines will be shown above in the dialog section.

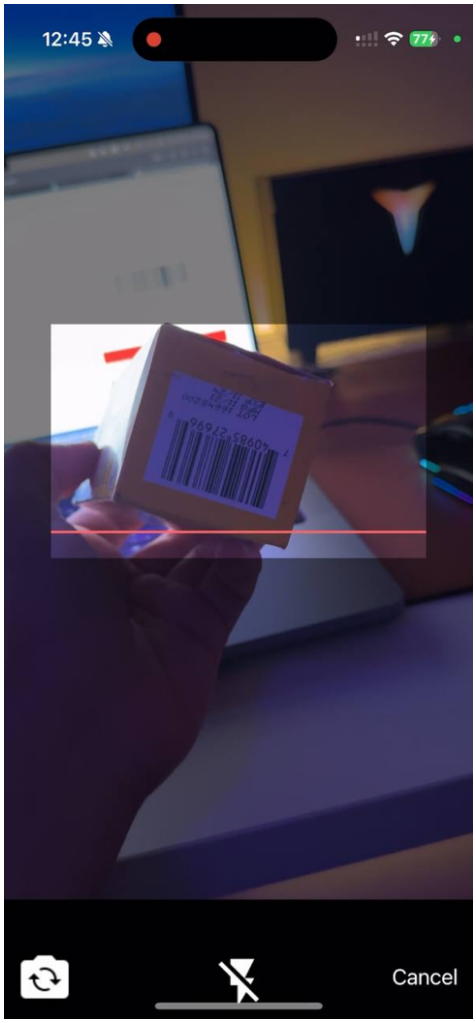

Figure 4-17 Scan Sell Product

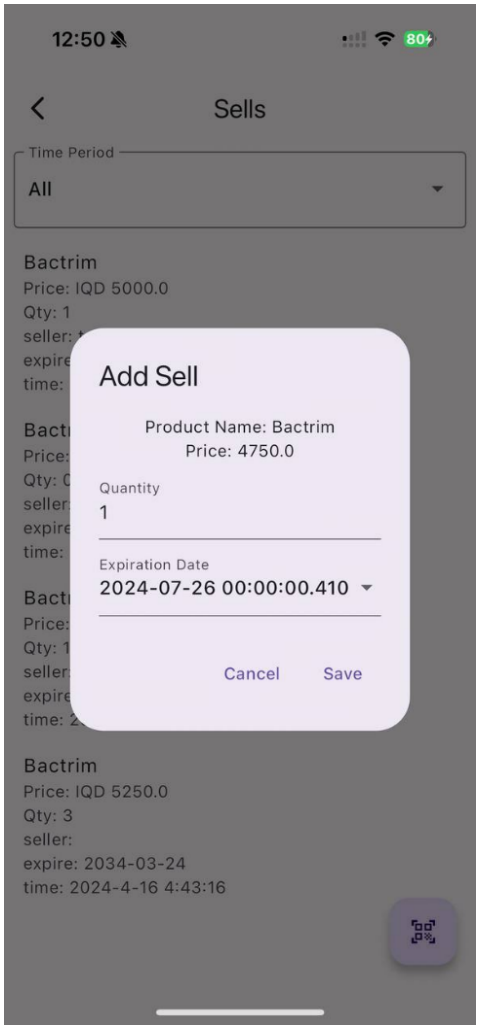

Figure 4-18 Adding Product Information

```dart
void _showAddSellDialog(BuildContext context, String scannedBarcode) async {
  String productName = '';
  double price = 0.0;
  int quantity = 0;
  String selectedExpirationDate = '';
  List<String> expirationDates = [];
  List<dynamic> shipments = [];
  try {
    final docRef = FirebaseFirestore.instance
        .collection('pharmacies')
        .doc(pharmacyId)
        .collection('medicines')
        .doc(scannedBarcode);

    final docSnapshot = await docRef.get();

    if (docSnapshot.exists) {
      final data = docSnapshot.data() as Map<String, dynamic>;
      productName = data['Name'] ?? '';
      quantity = data['quantity'] ?? 0;

      // Assign value to shipments
      shipments = data['shipments'] ?? [];

      if (shipments.isNotEmpty) {
        expirationDates = shipments.map((shipment) {
          Timestamp expireTimestamp = shipment['expire'];
          return expireTimestamp.toDate().toString();
        }).toList();

        expirationDates.sort((a, b) =>
            DateTime.parse(a).compareTo(DateTime.parse(b)));

        final currentDate = DateTime.now();
        for (final date in expirationDates) {
          final expirationDate = DateTime.parse(date);
          if (expirationDate.isAfter(currentDate)) {
            selectedExpirationDate = date;
            for (final shipment in shipments) {
              Timestamp expireTimestamp = shipment['expire'];
              String expireDate = expireTimestamp.toDate().toString();
              if (expireDate == selectedExpirationDate) {
                price = shipment['price'] != null ? double.parse(
                    shipment['price'].toString()) : 0.0;
                break;
              }
            }
            break;
          }
        }
      }
    }
  } catch (error) {
    print(error);
  }
```

Figure 4-19 _showAddSellDialog part1

```dart
  } catch (error) {
    print(error);
  }

  showDialog(
    context: context,
    builder: (BuildContext context) {
      return StatefulBuilder(
        builder: (context, setState) {
          return AlertDialog(
            title: const Text("Add Sell"),
            content: SingleChildScrollView(
              child: Column(
                children: [
                  Text(selectedExpirationDate.isNotEmpty
                      ? 'Product Name: $productName'
                      : 'No product name available'),  // Text

                  Text(selectedExpirationDate.isNotEmpty
                      ? 'Price: $price'
                      : 'No price available'),  // Text
                  TextFormField(
                    initialValue: '1',
                    decoration: const InputDecoration(
                      labelText: 'Quantity',
                    ),  // InputDecoration
                    keyboardType: TextInputType.number,
                    onChanged: (value) {
                      quantity = int.tryParse(value) ?? 0;
                    },
                  ),  // TextFormField
                  DropdownButtonFormField<String>(
                    value: selectedExpirationDate,
                    decoration: const InputDecoration(
                        labelText: 'Expiration Date'),  // InputDecoration
                    onChanged: (value) {
                      setState(() {
                        selectedExpirationDate = value!;
                        for (final shipment in shipments) {
                          Timestamp expireTimestamp = shipment['expire'];
                          String expireDate = expireTimestamp.toDate()
                              .toString();
                          if (expireDate == selectedExpirationDate) {
                            price = shipment['price'] != null ? double.parse(
                                shipment['price'].toString()) : 0.0;
                            break;
                          }
                        }
                      });
                    },
                    items: expirationDates.map((date) {
                      return DropdownMenuItem(
                        value: date,
                        child: Text(date),
                      );  // DropdownMenuItem
                    }).toList(),
                  ),  // DropdownButtonFormField
                ],
              ),  // Column
            ),  // SingleChildScrollView
            actions: [
              TextButton(
                onPressed: () {
                  Navigator.of(context).pop();
                },
                child: const Text("Cancel"),
              ),  // TextButton
              TextButton(
                onPressed: () {
                  addSell(scannedBarcode, productName, price, quantity,
                      selectedExpirationDate);
                  Navigator.of(context).pop();
                },
                child: const Text("Save"),
              ),  // TextButton
            ],
          );  // AlertDialog
        },
      );  // StatefulBuilder
    },
  );
}
```

Figure 4-20_showAddSellDialog part 2 (UI)

In the expiration part here is a list of the medicines that they are expired or has a few time to be expired each medicine show you the name of it the expiration and how any days left to be use and the colors will make it easier when the pharmacists see the color is red that means the medicine is fully expired and when there are a few days like a month it will be in an orange color.

Figure 4-21 *Expiration Screen*

```dart
Future<void> _fetchExpiredMedicines() async {
  try {
    // Query the Firestore collection for expired medicines
    QuerySnapshot expiredMedicinesSnapshot = await FirebaseFirestore.instance
        .collection('pharmacies')
        .doc(pharmacyId)
        .collection('medicines')
        .where('shipments', isLessThanOrEqualTo: [
      {'expire': Timestamp.now()}//This ensures that at least one shipment has an expire field
                                 // greater than or equal to current time
    ])
        .get();

    // Convert the retrieved documents into a list of maps
    List<Map<String, dynamic>> expiredMedicines = expiredMedicinesSnapshot.docs
        .map((doc) => doc.data() as Map<String, dynamic>)
        .toList();
    print('Expired Medicines: $expiredMedicines');

    // Update the state with the retrieved expired medicines
    setState(() {
      _expiredMedicines = expiredMedicines;
    });
  } catch (error) {
    print('Error fetching expired medicines: $error');
  }
}
```

Figure 4-22 _fetchExpiredMedicines function

4.6 Task Management

This asynchronous function `_fetchExpiredMedicines()` queries the Firestore database for expired medicines associated with a particular pharmacy ID. It retrieves documents from the 'medicines' collection where at least one shipment has an expire field less than or equal to the current timestamp. The retrieved documents are converted into a list of maps representing expired medicines. The function then updates the state with these expired medicines, utilizing `setState()` to trigger UI updates. In case of any errors during the process, the function catches and prints the error message for debugging purposes. This function facilitates efficient management of expired medicines within the app's pharmacy management system.

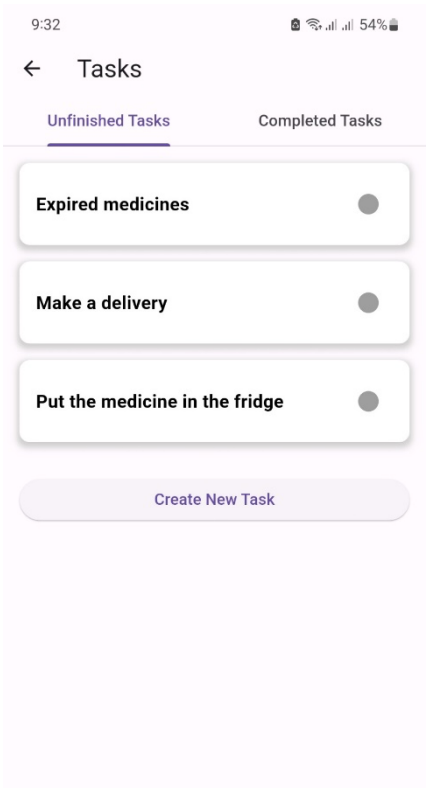

Figure 4-23ask Page

After, clicking on the task this page split into two parts the unfinished task and completed task, in the unfinished task here it will show the task that the pharmacists didn’t do when the user click on the task a pop-up menu going to show you the full description of that medicine. of when clicking on the radio button the task will go from unfinished task to the completed tasks.

```dart
@override
Widget build(BuildContext context) {
  return SingleChildScrollView(
    child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: [
          StreamBuilder(
            stream: getTasksStream(context),
            builder: (context, AsyncSnapshot<QuerySnapshot> snapshot) {
              if (snapshot.connectionState == ConnectionState.waiting) {
                return const CircularProgressIndicator();
              } else if (snapshot.hasError) {
                return Text('Error: ${snapshot.error}');
              } else {
                List<Widget> taskItems = snapshot.data!.docs.map((doc) {
                  Map<String, dynamic> data = doc.data() as Map<String, dynamic>;
                  return Padding(
                    padding: const EdgeInsets.only(bottom: 16.0),
                    child: TaskItem(
                      task: Task(
                        documentId: doc.id,
                        isCompleted: data['isCompleted'],
                        title: data['title'],
                        description: data['description'],
                      ),  // Task
                    ),  // TaskItem
                  );  // Padding
                }).toList();

                return Column(
                  children: taskItems,
                );  // Column
              }
            },
          ),  // StreamBuilder
          const SizedBox(height: 16.0),
          ElevatedButton(
            onPressed: () {
              createTaskDocument(context);
            },
            child: const Text('Create New Task'),
          ),  // ElevatedButton
        ],
      ),  // Column
    ),  // Padding
  );  // SingleChildScrollView
}
```

Figure 4-24 tasks

This `build` method overrides the default behavior to construct the user interface of the widget. It returns a `SingleChildScrollView` containing a `Column` widget. Inside the `Column`, there's a `StreamBuilder` that listens to a stream of tasks and displays a loading indicator while waiting for data, or an error message if an error occurs. If data is received, it maps each task document to a `TaskItem` widget and displays them vertically. Below the `StreamBuilder`, there's an `ElevatedButton` widget to create a new task when pressed. The UI is structured to efficiently display and interact with task data fetched from the Firestore database, enhancing user experience and productivity in managing tasks .

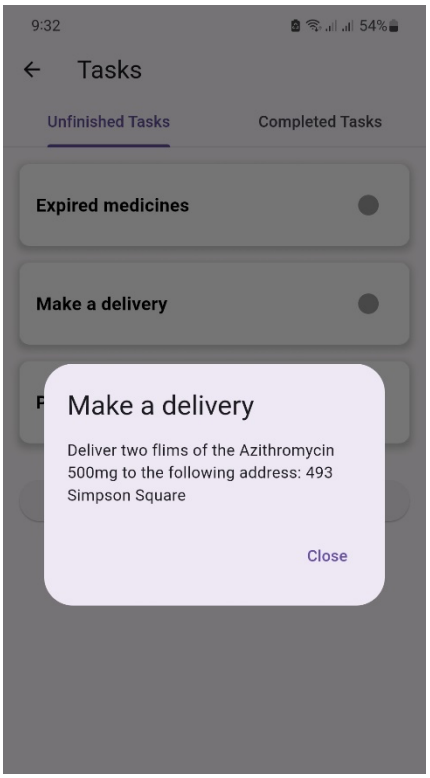

Figure 4-25 More Information

After, the pharmacists click on the task it will show them this page which is the information of this task.

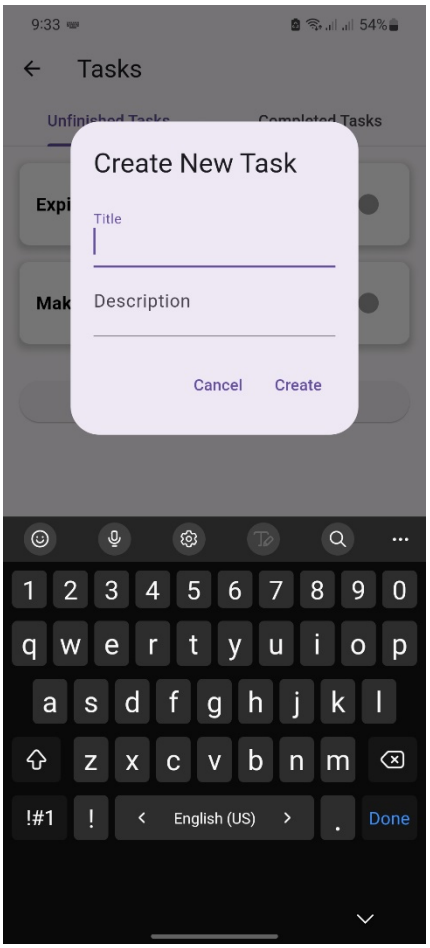

Figure 4-26 Create A New Task

Here, the pharmacists can add a new task with the name and the description of this task and click on create then the task will be created.

```dart
Future<void> createTaskDocument(BuildContext context) async {
  TextEditingController titleController = TextEditingController();
  TextEditingController descriptionController = TextEditingController();

  return showDialog<void>(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Create New Task'),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            TextField(
              controller: titleController,
              decoration: InputDecoration(labelText: 'Title'),
            ),  // TextField
            TextField(
              controller: descriptionController,
              decoration: InputDecoration(labelText: 'Description'),
            ),  // TextField
          ],
        ),  // Column
        actions: <Widget>[
          TextButton(
            onPressed: () {
              Navigator.of(context).pop();
            },
            child: Text('Cancel'),
          ),  // TextButton
          TextButton(
            onPressed: () async {
              try {
                UserProvider userProvider = Provider.of<UserProvider>(context, listen: false);
                String userId = userProvider.userId;
                CollectionReference tasksCollection = FirebaseFirestore.instance.
                collection('users').
                doc(userId).collection('tasks');
                await tasksCollection.add({
                  'title': titleController.text,
                  'description': descriptionController.text,
                  'isCompleted': false,
                  'created_at': DateTime.now(),
                });
                if (kDebugMode) {
                  print('Task document added successfully');
                }
                Navigator.of(context).pop();
              } catch (error) {
                if (kDebugMode) {
                  print('Error adding task document: $error');
                }
              }
            },
            child: Text('Create'),
          ),  // TextButton
        ],  // <Widget>[]
      );  // AlertDialog
    },
  );
}
```

Figure 4-27 create task function

The `createTaskDocument` function initializes text editing controllers for title and description fields. It then displays an `AlertDialog` prompting the user to create a new task with input fields for title and description. Upon clicking the "Create" button, it retrieves the user's ID, adds a new task document to Firestore with the provided data, and closes the dialog. Error handling ensures smooth execution, with debug mode printing error messages if any. This function facilitates the creation of tasks within the app, integrating user input with Firestore for efficient task management.

4.7 System Settings and Navigation

On the top left corner there is a slide bar icon as shown in the figure above leading us to the main parts of our program. starting with the picture name and email of the pharmacists, The home icon leading the pharmacists to the dashboard, inventory show the available medicines in the stoke and the full information of the medicines, sales to see all the sales that the pharmacists do in the pharmacy, notes to add notes to the medicines or the pharmacy, settings has all the rules and others that you have in the settings, logout to sign out from the account.

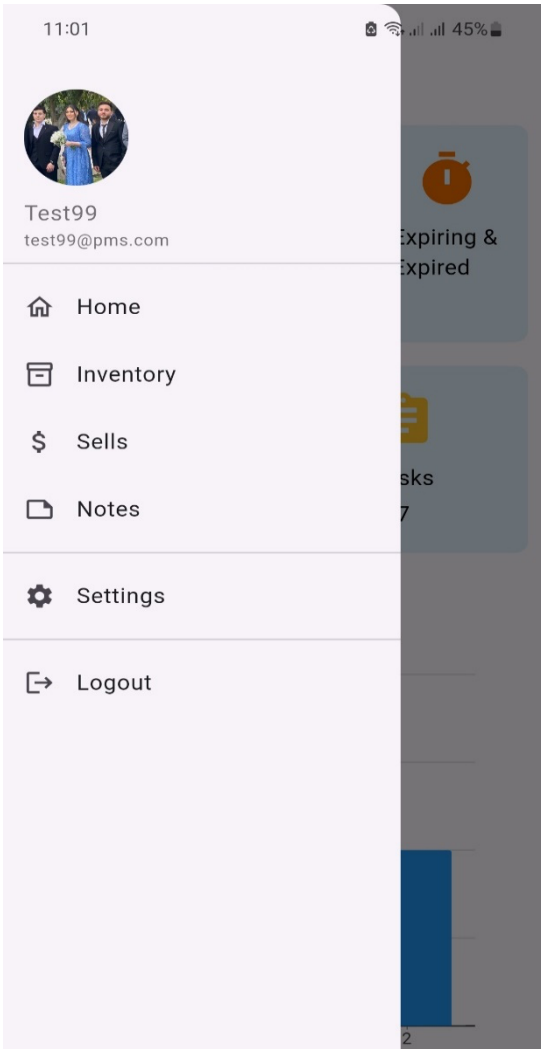

Figure 4-28 Slide Menu

```
class NavBar extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Drawer(
      child: FutureBuilder<DocumentSnapshot>(
        future: FirebaseFirestore.instance.collection('users').doc(userEmail).get(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text("Error: ${snapshot.error}"));
          } else {
            var userData = snapshot.data!.data() as Map<String, dynamic>?;
            var userName = userData?['name'] ?? 'Name';
            var userEmail = userData?['email'] ?? 'Email';

            return ListView(
              padding: const EdgeInsets.fromLTRB(0, 20, 0, 0),
              children: [
                DrawerHeader(
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      GestureDetector(
                        onTap: () => uploadImage(context),
                        child: CircleAvatar(
                          backgroundImage: NetworkImage(
                            userData?['photoURL'] ?? 'https://firebasestorage.googleapis.com/v0/b/p
                          ),  // NetworkImage
                          radius: 40,
                        ),  // CircleAvatar
                      ),  // GestureDetector
                      SizedBox(height: 10),
                      Text(
                        userName,
                        style: TextStyle(
                          color: Colors.black54,
                          fontSize: 16,
                        ),  // TextStyle
                      ),  // Text
                      Text(
                        userEmail,
                        style: TextStyle(
                          color: Colors.black54,
                          fontSize: 12,
                        ),  // TextStyle
                      ),  // Text
                    ],
                  ),  // Column
                ),  // DrawerHeader
                ListTile(
                  leading: Icon(Icons.home_outlined),
                  title: Text('Home'),
                  onTap: () {
                    Navigator.pop(context);
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => DashboardPage( ),
                      ),  // MaterialPageRoute
                    );
                  },
                ),  // ListTile

                ListTile(
                  leading: Icon(Icons.inventory_2_outlined),
                  title: Text('Inventory'),
                  onTap: () {
                    Navigator.pop(context);
                    Navigator.push(
                      context,
                      MaterialPageRoute(
                        builder: (context) => Inventory( ),
                      ),  // MaterialPageRoute
                    );
                  },
                ),  // ListTile

                ListTile(
                  leading: Icon(Icons.attach_money_outlined),
                  title: Text('Sells'),
                  onTap: () {
                    Navigator.pop(context);
                    Navigator.push(
                      context,
                      MaterialPageRoute(
```

Figure 4-29 class NavBar

The `NavBar` class creates a drawer widget for app navigation and user profile display. It utilizes a `FutureBuilder` to asynchronously retrieve user data from Firestore, displaying loading indicators and error messages as necessary. Once data is fetched, it populates the

drawer header with the user's name, email, and profile picture, allowing for user identification. The drawer contains a ListView of navigation items like Home, Inventory, Sells, Notes, Settings, and Logout, each associated with specific onTap actions for navigation. UserProvider is utilized for managing user data, ensuring smooth logout functionality. Overall, the `NavBar` enhances app navigation and user interaction within the interface.

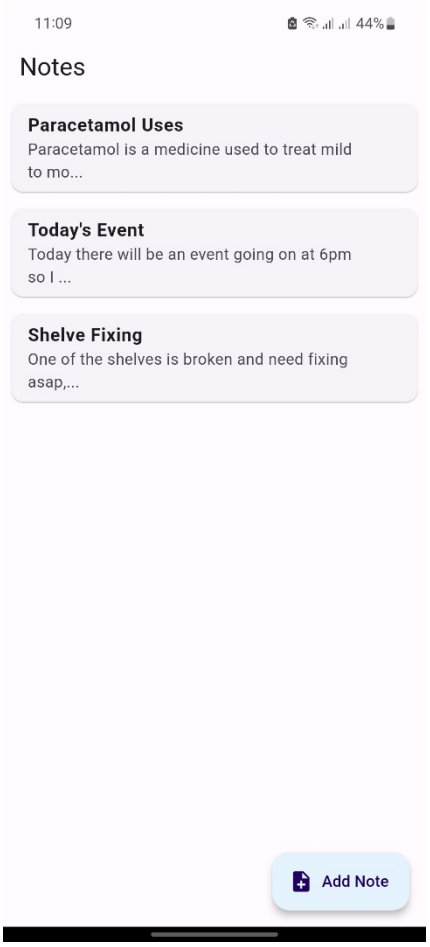

Figure 4-30 Notes

In, the slide menu has the note section, in this section the pharmacists can add the note to remind them about some medicine or some work to do.

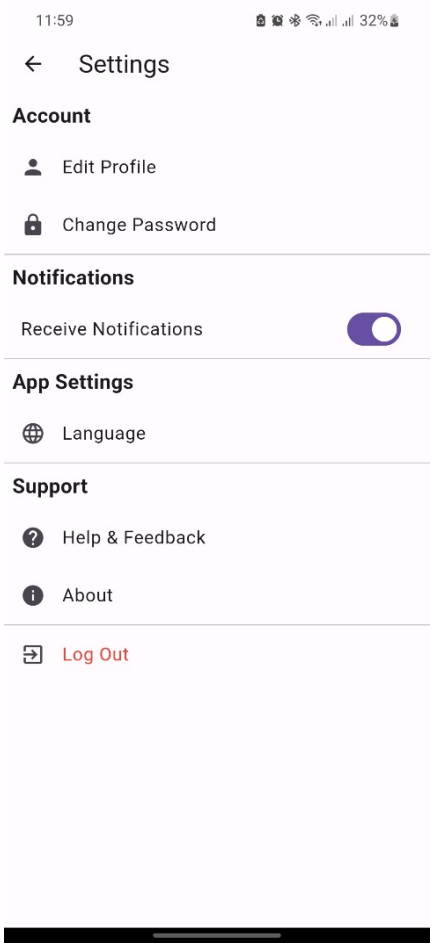

Figure 6-31 Setting Menu

In, the setting part we have multi choices to choose which is edit profile to edit the name and the picture of the pharmacists, Change password to change the security key of the pharmacists, Notification to let the pharmacists receive the notification about the sales expiration and so one, Language to select the language, Help and feedback from the user experience we can adjust the application to make it faster and more reliable, About which is a paragraph will show a small description of this app.

```dart
void _handleChangePassword() {
  String currentPassword = _currentPasswordController.text;
  String newPassword = _newPasswordController.text;
  String confirmPassword = _confirmPasswordController.text;

  // Validate password fields
  if (currentPassword.isEmpty || newPassword.isEmpty || confirmPassword.isEmpty) {
    _showSnackBar('Please fill in all fields');
    return;
  }

  // Check if new password matches the confirm password
  if (newPassword != confirmPassword) {
    _showSnackBar('New password and confirm password do not match');
    return;
  }

  // Implement your logic for changing the password here

  // Clear text fields after successful password change
  _currentPasswordController.clear();
  _newPasswordController.clear();
  _confirmPasswordController.clear();

  _showSnackBar('Password changed successfully');
}

void _showSnackBar(String message) {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text(message),
      duration: Duration(seconds: 2),
    ),  // SnackBar
  );
}
}
```

Figure 4-32 _handleChangePassword

The `_handleChangePassword` function facilitates the process of changing the user's password within the app. It retrieves values from text controllers for the current, new, and confirm password fields, then validates them. If any field is empty or the new password doesn't match the confirm password, it displays a corresponding error message. Upon successful validation, it executes the logic for changing the password (to be implemented by the developer) and clears the text fields. Additionally, it shows a snackbar with a success message indicating the password change. The `_showSnackBar` function is a utility function used to display snackbar messages within the app. These functionalities enhance user experience and security by providing clear feedback and ensuring proper password management.

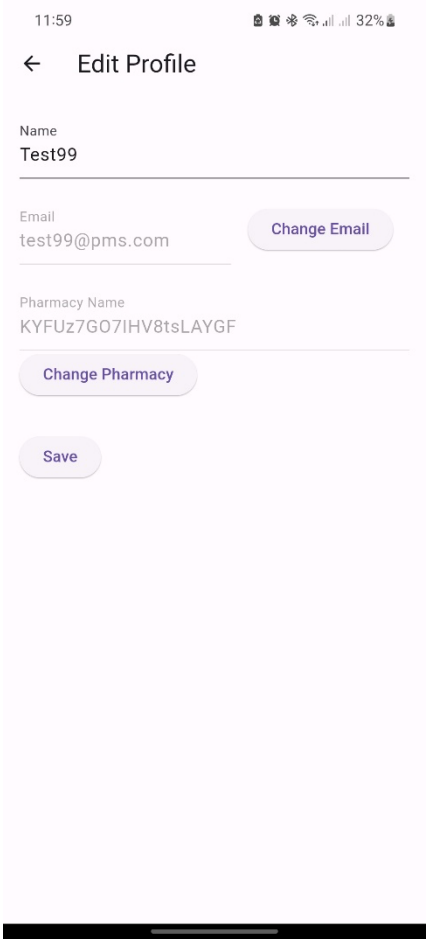

Figure 4-33 Edit Profile

If, the pharmacists want to change the profile of them they can do it by clicking on edit profile and from then they can change the name email and name of the pharmacy and by clicking on save then the data will save in the database of firebase.

```dart
Future<void> _updateProfile(String newName, String newEmail, String newUser, String newBio) async {
  UserProvider userProvider = Provider.of<UserProvider>(context, listen: false);
  try {
    await FirebaseFirestore.instance.collection('users').doc(userProvider.userId).update({
      'name': newName,
    });

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Profile updated successfully!'),
      ), // SnackBar
    );
  } catch (error) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Error updating profile: $error'),
        backgroundColor: Colors.red,
      ), // SnackBar
    );
  }
}
```

Figure 4-34 _updateProfile function

The `_updateProfile` function updates the user's profile information with the provided new name, email, username, and bio. It accesses the user's ID from the `UserProvider` using the `Provider.of` method and attempts to update the corresponding document in the Firestore database with the new name. Upon successful update, it shows a snackbar with a success message indicating the profile update. If an error occurs during the update process, it displays a snackbar with an error message including the error details, with a red background for better visibility. This function ensures smooth profile management within the app, providing feedback to users regarding the success or failure of the update operation.

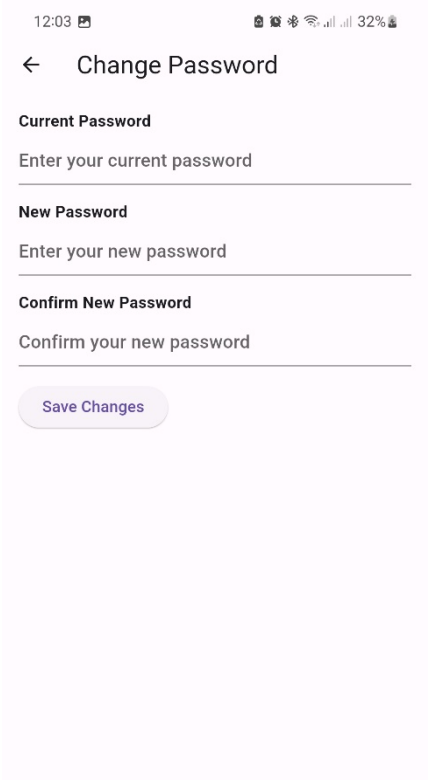

Figure 4-35 Change Password

Changing the password is required so in this page the pharmacists can change their password by filling the current which is the password that are using new password and confirmation of that password which is repetition of the new password and all the new data will be saved in the database by clicking on the save changes.

Figure 4-36 Setting the language

The pharmacists can change the language of the application based on the language that they want by clicking on language icon, by this step all the application will be changed by the pharmacists.

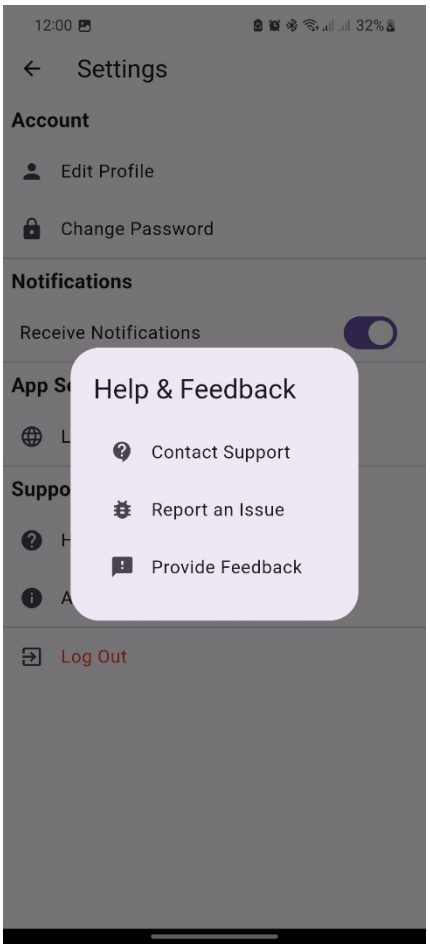

Figure 4-37 Feedback And Support

This is an important step in the application so we take feedback from the users or pharmacists so we can develop or update and fix the problems inside the application.

And if the pharmacists have some issue they can contact the support in order to fix their problem.

```dart
_buildListItem(Icons.help, 'Help & Feedback', () {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Help & Feedback'),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            ListTile(
              leading: Icon(Icons.contact_support),
              title: Text('Contact Support'),
              onTap: () {
                _contactSupport(context);
              },
            ),  // ListTile
            ListTile(
              leading: Icon(Icons.bug_report),
              title: Text('Report an Issue'),
              onTap: () {
                // Implement report issue functionality
              },
            ),  // ListTile
            ListTile(
              leading: Icon(Icons.feedback),
              title: Text('Provide Feedback'),
              onTap: () {
                // Implement feedback functionality
              },
            ),  // ListTile
          ],
        ),  // Column
      );  // AlertDialog
    },
  );
}),
```

Figure 4-38 Feedback And Support Dialog

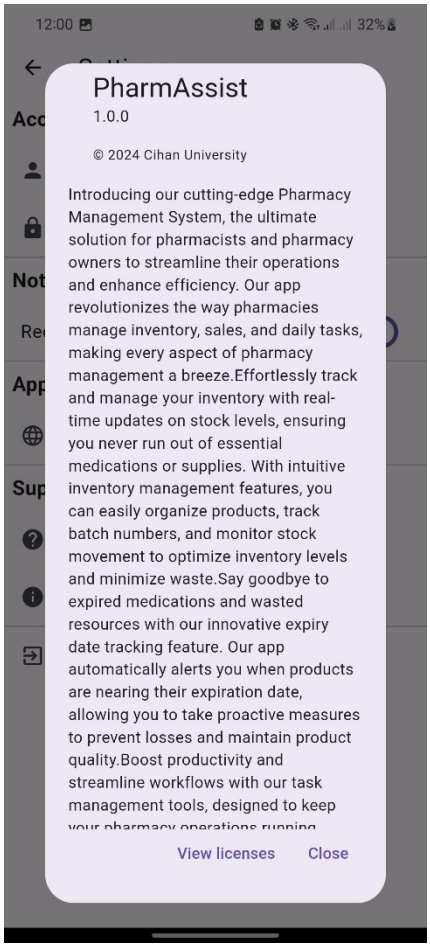

Figure 4-39 About the Application

When clicking (about) icon it will show a page that writing about the application in details and the use of application in general.

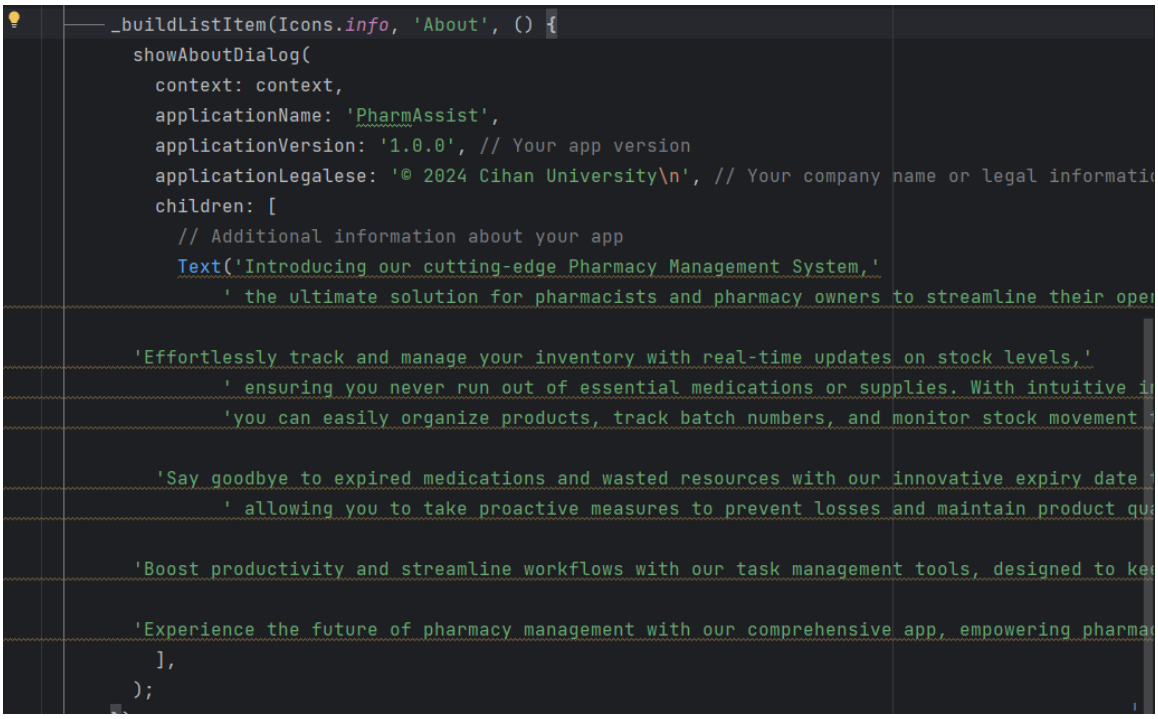

Figure 4-40 About Dialog

The `_buildListItem` function constructs a list item widget representing an "About" section in the app. Upon tapping the item, it displays an about dialog using the `showAboutDialog` method. The dialog contains information about the application, including its name, version, and legal information. Additional details about the app's features and benefits are provided in the dialog, highlighting its capabilities in streamlining pharmacy management operations. The text is structured to convey the app's innovative features such as inventory management, expiry date tracking, and task management. Overall, this function enhances user understanding of the app's functionalities and promotes its value proposition in revolutionizing pharmacy management.

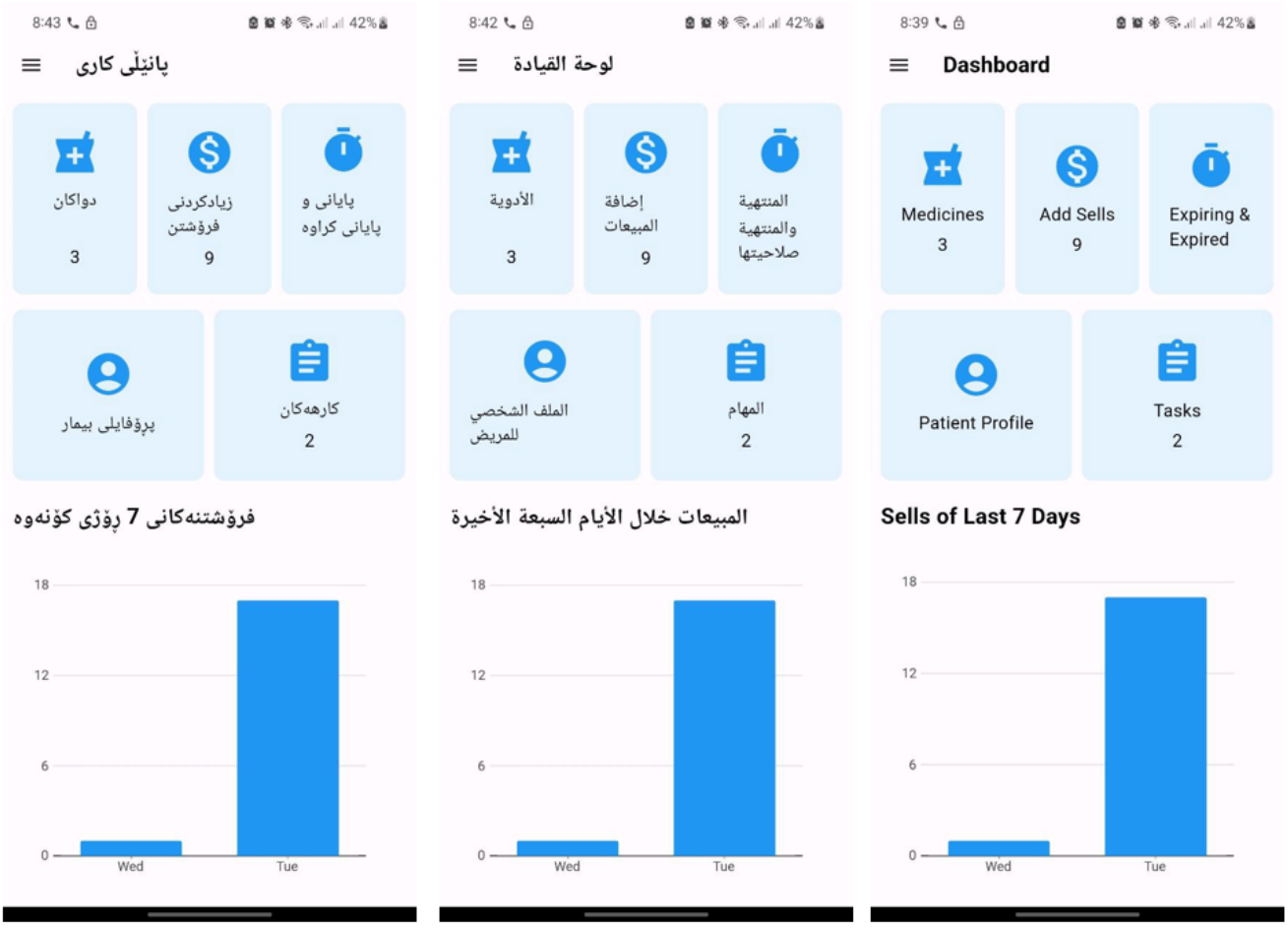

Figure 4-41multi-language

The multi-language feature allows users to interact with an application in their preferred language. This feature enhances accessibility and user experience by providing content, labels, and messages in multiple languages, catering to a diverse user base. Users can easily switch between languages within the application settings, enabling them to understand and navigate the interface more effectively.

Implementing multi-language support involves translating all textual elements of the application, including user interface components, error messages, notifications, and help documentation, into different languages. This translation process ensures that users can comprehend and engage with the application regardless of their language proficiency.

```
//------------------------ language ---------------------------
Map<String, String> arabic = {
  'login_screen': 'شاشة تسجيل الدخول',
  'log_in': 'تسجيل الدخول',
  'error': 'خطأ',
  'forgot_password': 'هل نسيت كلمة المرور',
  'sign_up': 'سجل',
  'sign_in_with_google': 'تسجيل الدخول باستخدام جوجل',
  'user_not_found': 'لا يوجد مستخدم مسجل بهذا البريد الالكتروني.',
  'dashboard': 'لوحة القيادة',
  'medicines': 'n\الأدوية',
  'add_sells': 'إضافة المبيعات',
  'expiring_expired': 'المنتهية والمنتهية صلاحيتها',
  'patient_profile': 'الملف الشخصي للمريض',
  'tasks': 'المهام',
  'sells_of_last_7_days': 'المبيعات خلال الأيام السبعة الأخيرة',
  'home': 'الصفحة الرئيسية',
  'inventory': 'المخزون',
  'sells': 'المبيعات',
  'notes': 'ملاحظات',
  'settings': 'الاعدادات',
  'logout': 'تسجيل الخروج',
  'search': 'بحث',
  'category': 'الفئة',
  'name': 'الاسم',
  'brand': 'العلامة التجارية',
  'dose': 'الجرعة',
```

Figure 4-42 Map<String, String>

```
Map<String, String> getTranslations() {

  switch (lang) {
    case 'ar':
      return arabic;
    case 'ku':
      return kurdish;
    case 'en':
      return english;
    case 'jp':
      return japanese;
    default:
      return english;
  }
}
```

Figure 4-43 getTranslations() function

The `Map<String, String>` in (Figure 4-42) structure efficiently stores translations for different languages in key-value pairs. It allows quick retrieval of text based on the user's language preference, enabling multi-language support in the application. This feature enhances accessibility and user experience for diverse linguistic audiences.

This code in (Figure 4-43) defines a function, `getTranslations()`, which returns language-specific translations based on the selected language. It uses a `switch` statement to select the appropriate translation map (Arabic, Kurdish, English, Japanese) based on the value of the `lang` variable. If the selected language isn't recognized, it defaults to English.

Chapter Five: Conclusion and Future Work

5.1 Conclusion

The research highlights that the successful implementation of the mobile app can transform pharmacy management by automating routine tasks and reducing administrative burdens on pharmacists and staff. This frees up more time for direct patient care and counseling, improving the quality of service provided. Additionally, the app can help manage medication adherence by sending reminders to patients, thus promoting better health outcomes and potentially reducing hospital readmissions. Furthermore, the app's capability to integrate with electronic health records can facilitate better coordination of care across healthcare providers. This can lead to more comprehensive medication reviews and a holistic approach to patient treatment plans. The app may also enable the implementation of tele pharmacy services, allowing pharmacists to offer consultations and dispense medications remotely, especially in underserved or rural areas. The mobile app's data analytics features offer opportunities for continuous improvement by tracking performance metrics, identifying areas for optimization, and supporting data-driven decision-making. Its ability to securely store and transmit patient data ensures compliance with privacy regulations and builds trust with patients.

5.2. Future Work

5.2.1. Patient profile:

An extensive record of a patient's medical background and details is called a patient profile. Personal information, medical history, previous and present prescriptions, genetic diseases, known allergies, treatment plans, immunization records, and other pertinent health information are usually included. Healthcare professionals, including pharmacists, physicians, and nurses, can use this profile as a useful tool to learn more about a patient's medical history and to deliver safe, individualized care.

References

[1] Kebede, M., Adeba, E. and Chego, M., 2020. Evaluation of quality and use of health management information system in primary health care units of east Wollega zone, Oromia regional state, Ethiopia. BMC medical informatics and decision making, 20, pp.1-12.

[2] Li, W., Chai, Y., Khan, F., Jan, S.R.U., Verma, S., Menon, V.G., Kavita, F. and Li, X., 2021. A comprehensive survey on machine learning-based big data analytics for IoT-enabled smart healthcare system. Mobile networks and applications, 26, pp.234-252.

[3] Saha, E. and Ray, P.K., 2019. Modelling and analysis of inventory management systems in healthcare: A review and reflections. Computers & Industrial Engineering, 137, p.106051.

[4] Chathuranga, I.K., 2021. Smart Computerized Pharmacy Management System (Doctoral dissertation).

[5] Nahi, A.A., Flaih, L., Jasim, K. and Hossian, R., Review of IoT and Robotics application (Special case Study in Iraq and Kurdistan Region, 2022. 4th International Conference on Communication Engineering and Computer Science (CIC-COCOS'2022).

[6] S. Lalitha V, E. Robert R, R. Edgar L and J. Robert N, "Integration of Mobile Health Technology in the Treatment of Chronic Pain: A Critical Review", published in (Jul 2017), https://www.proquest.com/openview/59e7b81cf5fb0a4f7b261fc3ed940933/1?pq-origsite=gscholar&cbl=47693

[7] R. Mirzaeian1, F. Sadoughi2*, S. Tahmasebian3 and M. Mojahedi4, "Progresses and challenges in the traditional medicine information system: A systematic review", published in 2019, https://jppres.com/jppres/

[8] Dehnavieh, R., Haghdoost, A., Khosravi, A., Hoseinabadi, F., Rahimi, H., Poursheikhali, A., Khajehpour, N., Khajeh, Z., Mirshekari, N., Hasani, M. and Radmerikhi, S., 2019. The District Health Information System (DHIS2): A literature

review and meta-synthesis of its strengths and operational challenges based on the experiences of 11 countries. Health Information Management Journal, 48 (2), pp.62-75.

[9] Zhan, Y., Han, R., Tse, M., Ali, M.H. and Hu, J., 2021. A social media analytic framework for improving operations and service management: A study of the retail pharmacy industry. Technological Forecasting and Social Change, 163, p.120504.

[10] Flutter Documentation: Flutter Docs

[11] Dart Programming Guides: Dart Guides

[12] Visual Studio Code Documentation: VS Code Docs

[13] Firebase Documentation: Firebase Docs