PHARMACIST

Mobile and Ubiquitous Computing – 2023/24

Course: MEIC
Campus: Alameda

Group Nº07
Ana Inês Jin, Nº 99176
Inês de Almeida Pissarra, Nº 99236
Miguel Cunha da Silva Eleutério, Nº99287

Introduction

In this project we developed a mobile application for the Android system: PharmacIST. The application offers the user the possibility of managing and finding pharmacies and medicines they need in an easy, intuitive, and quick way. Below we describe the application's features. Each (number) corresponds to a screenshot of the application shown on the last page of this report, numbered in the upper left corner.

Features

- The application starts with the login page where the user can choose between doing the login (1), sign up (2) (additional component 3.4 – User Accounts) or login as a guest. Login as a guest is a quick access option that allows the user to access some of the application features (such as search for pharmacies), while others are reserved for users with an account (such as having favorite pharmacies).
- The main page of the application shows a map (3) where the user can check their current location (which needs location permissions) and pharmacies' locations. Their favorite pharmacies are marked with the star marker. The map can be dragged, address searched, or centered on current location. The application allows users to create a username in the application/system (4), (5). In case the user logs in as a guest, to make this username persistent, it is stored in SharedPreferences. If the user has an account logged in, it stores the information in the database.
- The application also offers the user the possibility to create a new pharmacy (6), (7), (8). To do this, the user must enter the name of the pharmacy, location (7) (pick location on map, use address, or current location) and photo (from the camera or gallery). For the latter, authorizations are requested from the user to access the camera and storage.
- Tapping a pharmacy marker goes to that pharmacy's information panel (9). In the pharmacy information panel we can find the pharmacy's name, address, and picture. It has a histogram of ratings and a list of the pharmacy's available stock (with a button to purchase the medicine (10), and tapping medicines opens the medicine information panel (19)). There are also five other important buttons:
    - a heart button to add/remove pharmacy from favorites;
    - a button to rate the pharmacy (1 to 5 stars, additional component 3.3 – User Ratings) (11);
    - a button to show location on map (12) where the user can also check the directions to navigate there (13);
    - A button to manage the stock (14), where users can scan barcodes (15) to create a new medicine (16) or update an existing medicine's stock.
    - A button to share the pharmacy (additional component 3.5 – Social Sharing To Other Apps) (14), (17).

    (There is also a button for meta moderation, but the functionality ended up not being implemented.)
- The application also offers the possibility to list all the existing medicines in the pharmacIST system, as well as their closest pharmacy with stock, if there's any. The user may also search medicine by name (18).
- Tapping any of the medicine will open its corresponding information panel (19), which displays the medicine's name, description, picture and a list of all the nearby (~30km radius) pharmacies

and corresponding stock of this medicine. Clicking any pharmacy takes the user to that pharmacy's information panel [9]. Additionally, there's also a bell icon, which enables notifications whenever this medicine is added to a favorite pharmacy (example of a notification in [20]).

Back-end

The back-end service used to implement the application was Firebase. Firebase was chosen for its real-time database capabilities, easy synchronization across multiple devices and built-in authentication services. It allowed efficient development and reliable data management without the need for complex infrastructure setup.

Resource Frugality

To make the application faster, there is a button where the user asks to search for pharmacies in that area, and only pharmacies within a radius of around 30 kilometers are loaded and shown, which prevents excessive loading of possibly unused data. Besides that, both the pharmacy panel's medicine list and the screen of all medicines implement lazy loading. That is, the resources are only loaded as the user scrolls through the list, which also prevents excessive loading. It is made possible by addOnScrollListener, which detects when the user has scrolled down. The app loads more data when it is at the end of the list and has more data to load. This is not done on the medicine panel, because all pharmacies must be loaded to ensure we get the closest distance, but to prevent loading all pharmacies, these are restricted to the same radius as in the main map (30 km). All images are only downloaded when the content becomes available on screen. Beforehand we simply download a reference (link) to the image and there's also image caching handled by Gradle. Additionally, whenever the user is using mobile data, secondary images are loaded as a local placeholder image and only downloaded from the server when the user presses them.
To implement the notifications, a service that runs in the background was utilized, leveraging the addSnapshotListener option of the Firebase Firestore database. When a pharmacy introduces a new medicine with available stock, the service promptly receives this information and triggers notifications to alert users about new medicines available, if the user has active notifications for that medicine and the pharmacy is one of the user's favorites.

Context Awareness and Privacy

As the user moves, their position in the application is also updated. As this happens, the application also updates the nearby pharmacies, displaying a button the user may press to open the panel of the closest pharmacy. The pharmacy is not displayed automatically, due to being possibly intrusive to the user's experience.

Caching

To implement the cache, SQLiteDatabase was used to manage local data efficiently. A DatabaseHelper class was created to handle database creation and

version management, defining tables for pharmacies, medicines, favorite pharmacies, and medicines with notifications. All data from the application is cached as the user accesses it. When the user has internet connection, pharmacies within a larger range are preloaded to cache, thus using the principle of spatial locality (if a location is searched, the location nearby will tend to be searched soon). The pharmacy repository is renewed when the user clicks on the pharmacy search button in an area that is not in the cache, or when the user takes an action that could change the pharmacies. When a user is logged in, the names of their favorite pharmacies and the medicines with active notifications are stored in cache. This approach ensures faster actions within the application, eliminating the need for constant database queries. This cached information is refreshed during relevant actions and when the user closes and reopens the application.
As mentioned, images are cached and managed by Gradle. Requests to the server can be done offline and will be queued until connection is online, except for registering a pharmacy, which requires internet connection, due to using an API.
This approach improved app performance by minimizing network requests and enabling offline functionality.

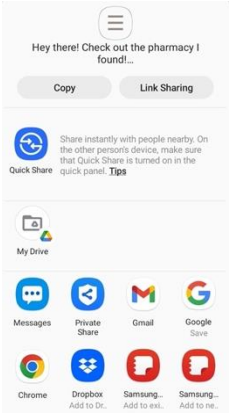
Additional Components

The application contains the following additional components:

3.1 Securing Communication: All messages between the application and the database use HTTPS (Hypertext transfer protocol secure), ensuring integrity and confidentiality. In addition, data at rest in Firebase is protected by encryption, and account passwords are stored securely using hashing and salting techniques, ensuring that the original passwords are not stored directly, but rather in an encrypted form that protects against unauthorized access and brute force attacks.

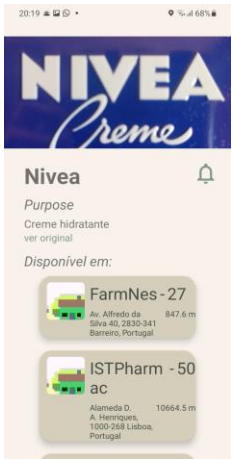
3.3 User Ratings: This feature was implemented with the library *MPAndroidChart* to draw the bar chart that displays all ratings of the pharmacy. Additionally, the Android widget *RatingBar* allows users to give a rating using a star-based system. Only logged-in users can rate pharmacies, and submitting a new rating overwrites any previous rating by the same user.

3.4 User Accounts: With the help of Firebase, we implemented user accounts using a set of APIs and SDKs provided by the platform, which simplifies the process of creating, authenticating and managing user accounts. This includes using methods like createUserWithEmailAndPassword() to create user accounts with email and password, signInWithEmailAndPassword() to authenticate users with email and password, and other APIs to handle profile updating.

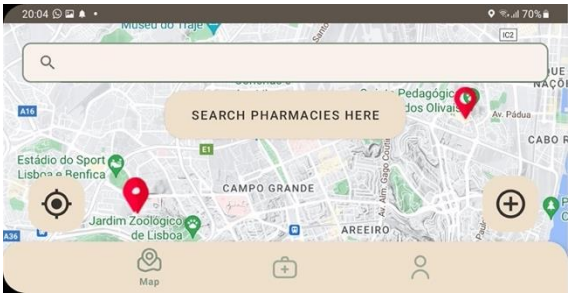


3.5 Social Sharing To Other Apps: The application offers the possibility to share information about a pharmacy in an easy way, using the Android's Intent system. It allows the sharing of information in different types of applications such as messaging applications, email, and social media.

3.6 Localization: For static strings, such as button labels and Toast messages, we used *strings.xml* files to provide translation between Portuguese and English. Instead of using Google Translate API as recommended in the project description, we opted for Google ML Kit to translate user-submitted content. Since texts like pharmacy names or medicine names are not translatable, the application only offers translation for medicine descriptions. Additionally, users have the option to view the original or translated text.



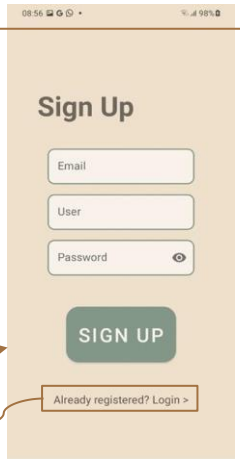
3.7 UI Adaptability: Rotation: By ensuring the layout of each activity in the application is encapsulated within a scroll view, we provide an instant adaptation for each orientation, allowing the user scroll the application vertically when the phone is oriented horizontally.



3.8 UI Adaptability: Light/Dark Theme: To ensure seamless adaptation to both light and dark themes, we created specific attributes to define the colors of elements in the application, and these attributes are then defined in *themes.xml*. When creating elements in xml files, we used attributes instead of colors so that it can seamlessly transition between light and dark modes.

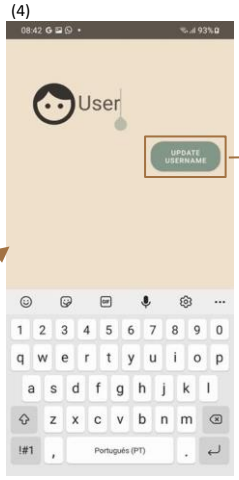
(1)

(2)

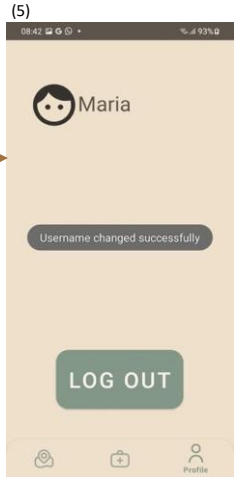


(3)

(4)

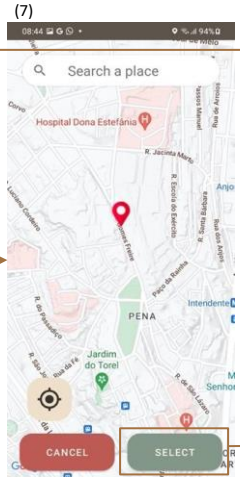


(5)



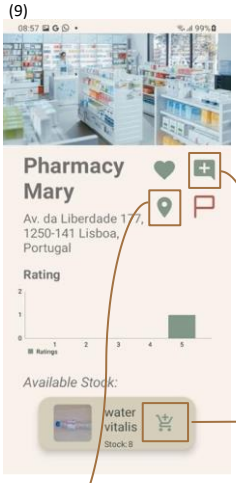
(6)

(7)

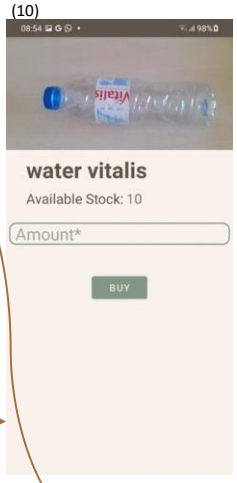


(8)

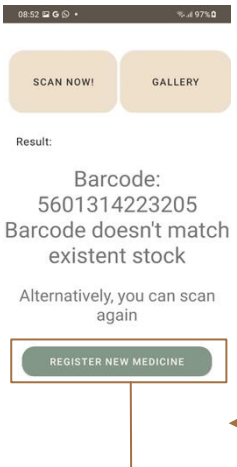
(9)



(10)

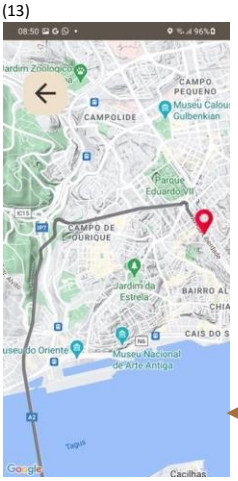


(15)

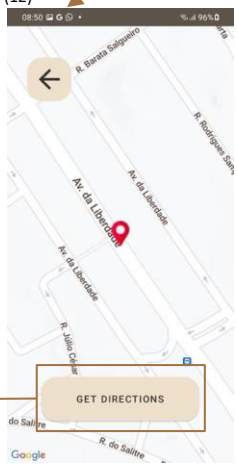


(14)

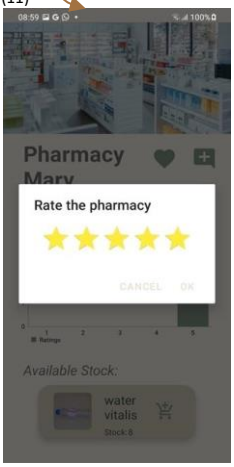
(13)



(12)

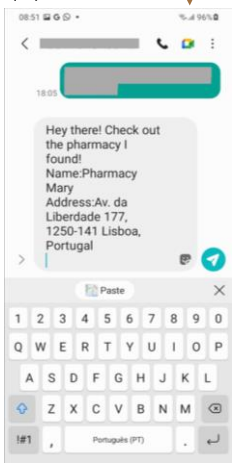


(11)

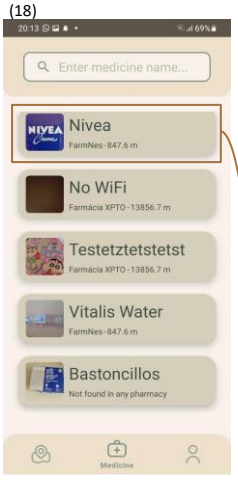


(16)

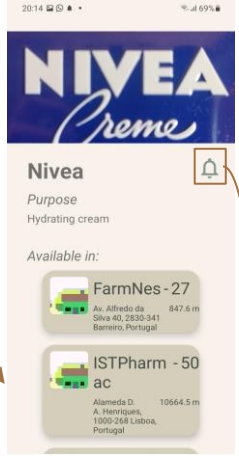
(17)



(18)



(19)



(20)