PharmacIST - G23

Lucas Pinto - ist1110813
MEIC-A
Instituto Superior Técnico

Miguel Rocha - ist1110916
MEIC-T
Instituto Superior Técnico

1 Mandatory Features

The mandatory functionalities were implemented in totality, allowing users to signup, login, or just use a guest account, showing the pharmacies in a map with the distinction of their favourites, the detail pages of pharmacies and medicines, and the core value of crowdsourcing the stocks and adding/removing pharmacies and medicines.

There was a detour from specification: it was requested that the favourite pharmacy's medicines were to have notifications when updated, but it was decided that it would be more flexible to allow the user to choose which medicines to get notifications from, from any pharmacy. In other words, independently if a pharmacy is favourite, a user only needs one click to subscribe to the pharmacy's medicine updates, receiving a notification every time the stock changes. This, however, leaves the favourite functionality of a pharmacy to be only visual. In the real world, if the users claimed preference over the specification and after user testing, the fix could be implemented.

1.1 Backend

The backend was implemented in Kotlin using Spring Boot. It does not persist data into a database, but good practices were followed to allow for that functionality to be smoothly implemented. The requests are RESTful and use the Problem JSON specification, being the client the one to initiate the requests. For the notifications, Server Sent Events were implemented to allow for real-time updates whilst avoiding unnecessary polling.

1.2 Application Flow

1.2.1 Main Screen

The application starts with an initial screen, containing the name of the application and a button to start. Behind the scenes, this activity checks if the user is logged in or not. This information is stored using shared preferences, containing the user id and token, which allows them to perform certain actions on the backend. Moreover, the user could have logged in previously, and the token could be now invalid. Remember that the backend is a separate component from the application, and we cannot consider the token to be valid forever. The token is validated by issuing a request to the backend and checking if the response is positive or not. It is also possible that the user is logged in (the token is present and valid) but there is no network available. Therefore, the logged in information is cleaned, but the user cannot go to the authentication screen, which would not make sense. Instead, he goes to the Dashboard screen, in guest mode. Of course, if the token is invalid, the application navigates to the authentication screen.

At the same time, if the user has Wi-Fi access, the application takes this opportunity to download the information about the pharmacies (along with images). So, if the user disconnects after that, the information should be available. Note that the medicine stock information is not downloaded, as this is very volatile information, and could change by the minute. Nonetheless, the server returns a "last updated at" field for pharmacy medicines, allowing users to logic about how updated a medicine is, as the application is crowdsourced (e.g. a pharmacy in a village populated by elders that do not use the application/smartphones, where the only updates are made by random travellers).

Finally, if the user wants, he can also check the info screen with both the developer information, with options to navigate to each individual GitHub repositories, or send an email to the application support.

1.2.2 Authentication Screen

This activity allows the user to create a new account, log in, or proceed in guest mode. In case the user disconnects from the network, already in this activity, a popup appears, blocking the user from any action, until he re-connects again. As future work, this could be improved by navigating to the Dashboard, in guest mode. After the creation of a new user, log in, or proceeding in guest mode, he will be redirected to

the Dashboard Activity, throwing the current activity away from the Android application stack as the authentication was done successfully.

1.2.3 Dashboard Screen

This activity uses the Google Maps API to display the map with the available pharmacies, and user current location (conditional to user permission). The user can focus in his location, search for a specific place in the world (text search), select a pharmacy, logout (upper left symbol), search for medicines, open/close pharmacies (ordered by distance to the user), and even scan a barcode. Favourite pharmacies appear green in the screen. The Places API was used to get available world places (with household precision), and to transform places into coordinates. If the user is in guest mode, they can do everything except creating new pharmacies, since it was decided that that is a protected action due to meta moderation. Otherwise, the user could just long press any place in the map, and an activity for pharmacy creation is opened with already resolved pharmacy location.

The place information, displayed in the end of the screen, is automatically updated, as the user navigates in the map. This is a nice feature, but one that consumes lots of data. Therefore, it's only available if the user is using a Wi-Fi connection.

If the user started the application without Wi-Fi connection, the cache is empty, and it still remains without network connection, an error popup is displayed, since it's virtually impossible to do anything until data can be retrieved from the backend. Otherwise, data is displayed in the screen, even only cached data. Note that pharmacy data should not change very often. It's not every day that new pharmacies are created. Therefore, there is no automatically refresh for pharmacy data. If the cache is empty and the user suddenly connects to network, the pharmacies and requested and cached.

When this activity is resumed again, the pharmacies are refreshed in order to update the list of pharmacies, for instance, after the user has just created one.

In this activity, only the pharmacies' own information are loaded, and not the list of medicines, since this saves a lot of space. Yet, as future work, this activity should only request the pharmacies IDs, and not the other information.

1.2.4 Create New Pharmacy Screen

When it comes to creating a new pharmacy, one possibility would be to use the same activity that required the action: Dashboard activity. Yet, in the perspective of the user, we think that makes more sense to use another activity for that, as, for instance, the back button would result in the termination of this activity, and the user would return to the Dashboard activity, like nothing happened. Of course, if the user does this, the data (field texts, and others) would be lost forever, which makes sense since the user explicitly requested to go back to this action.

The address can still be edited in this activity, with support for world places search, even thought the address comes prefilled when the user long pressed a specific place in the map to create the pharmacy. The activity only allows the user to create a pharmacy if the user takes a picture. After confirmation, it finishes.

1.2.5 Pharmacy Panel View

This activity displays the list of medicines, with possibility to navigate to the Medicine Panel activity. If the user is logged, they can do all mandatory features specified in the project description, as well as share the pharmacy and medicine list over any social network (uses implicit intents for this) or email. If the devices disconnect the network, the view is updated not to allow any operation that requires communication with the backend.

If there is a Wi-Fi connection, the activity starts by requesting the list of medicines. Otherwise, it checks if there are cached medicine information, before starting the periodic refresh. This improves the user experience as the user won't probably see a delay before seeing the medicines on the screen, in case they're on mobile data.

When this activity starts, a background job (using coroutines) is used to keep refreshing the list of medicines, since this information is relatively volatile. On the contrary, the pharmacy information is never refreshed, as it shouldn't change often, unless the user re-starts the application again with Wi-Fi enabled, and the cache is refreshed. For this, polling was used, with different period times, depending if the user is on Wi-Fi or Mobile data.

A new activity should be used to create a new medicine, just as explained in the Pharmacy Panel View section. Yet, due to time constraints, that was not implemented, and the same activity is used for the medicine creation, whilst still achieving intended functionality.

1.2.6 Medicine Panel view

This activity uses the user location to display the closest pharmacies to the user, that have the medicine in question. If the user didn't activate the location until it arrives here (localization cached not available), a popup (with possibility to close) is shown, and as long as the location is available, the activity loads the closest pharmacies (ordered by distance) and shows them, with possibility to open the Maps to navigate to them. Since the location is being refreshed, if the user starts this activity again, a new list of pharmacies is obtained.

1.3 Application Architecture

In most activities, the paradigm is the same. The code is divided in 3 main components:

Activity that contains logic and creates Composable Views

Views that use Jetpack Compose to draw the UI

View Model that holds UI state (e.g. pharmacies/medicines lists). It is also responsible for making use of the backend service to make requests, because the View Model has access to the View Model coroutine scope, a mechanism of Kotlin for creating lightweight threads in order to make asynchronous work. The View Model is not coupled to the Activity, allowing it to keep living if the activity is re-created.

To hold state, Mutable State Flows are used, a feature of Jetpack compose that allows for UI to re-render whenever a Mutable State changes.

The Ok-HTTP package is used to make HTTP requests to the backend server, in opposition to retrofit, since it offers more control on how the response is handled. To make it more efficient, the suspendCoroutine() function is used, suspending the coroutine that issues the request until the response arrives, and an IO Ok-HTTP thread wakes the coroutine again. A possibility would be to use an active waiting: Corroutine.sleep(), but it would be less efficient.

When a request is made, it's possible that it results in a failure, for whatever reason. Therefore, some requests return a Result (either Success or Failure), allowing for the display of an error message to the user. As mentioned in the backend section, the backend offers a RESTful API, with the Problem-JSON media type, allowing to inform about any existing errors.

1.4 Caching

Caching was implemented, as part of the project requirements. Room was chosen as the technology to hold cached Pharmacy information, Medicine information, and medicine stock information. Every time this data is requested, it's immediately stored in cache.

For Pharmacy and Medicine images, Room, being an SQL based DB, could be used to store the bytes. Yet, we used a more appropriate storage: the Android external file system, but not accessible to other applications, as there is no motive to.

1.5 Network

To deal with network state, since most activities use and need to know about the change in the state, Mutable State Flows are used, in order for the composable functions to be updated whenever the state changes and execute decisions accordingly. The 3 possible, and necessary, states are: on Wi-Fi, on Mobile Data, and not connected to network.

1.6 Location

We use two methods of localization:

- The Google Maps API, which deals with the symbol that shows the location in the map, and gives the coordinates when the user taps in a specific location, which is necessary to translate that position to an address, using the Places API (requires the coordinates);
- The FusedLocationProviderClient, which gives us access to the user localization coordinates, allowing to request the Backend pharmacies by proximity. This library, also, allows retrieving the cached localization, which is handy if the user granted the necessary localization permissions, but, then, turned off the localization. In this case, we used the cached localization instead. Finally, duo to time constraints, we didn't fix the bug on the Dashboard activity, where the app should have requested the user localization periodically, in order to update nearby pharmacies. We only used the last localization, which is the one that is obtained when the user turns on the localization service.

1.7 Resource Frugality

As said previously, pharmacy data is only fetched when required. For instance, in the application startup to refresh the cache when the user is on Wi-Fi. Another place is in the Dashboard activity to show the nearby pharmacies, as you want to search where the map is, as well as in the medicine's view because there may be pharmacies that have not been fetched prior that have said medicine. One could think "why not refresh the pharmacy information on metered data?" - well, pharmacy data changes very rarely, and it's much more probable that the user will get Wi-Fi faster than a new Pharmacy is created.

On the contrary, since the medicine stock is much more volatile, it is refreshed in a timely manner. If the user has internet connection, a background job will keep refreshing medicine the stock just for that pharmacy. Note that, only the stock is refreshed, and not the medicine info. The time interval between refreshes varies if the user is on Wi-Fi or metered data. Duo to time constraints, we didn't have time to implement a setting option to change the interval time.

A pharmacy and medicine image is only automatically fetched if the user is using Wi-Fi, otherwise, if on metered data, a placeholder is shown to load the image from the Backend server.

To use resources more efficiently, collections are paginated by the server, i.e., only returning a portion of the data instead of everything. For example, in the map activity, only 20 pharmacies whose location appears on the map are loaded, and not others that are far away - if we're at IST we only want the Lisbon ones, not the French. The same happens in the

Pharmacy View for the medicines list and on the Medicines View for the pharmacies list, where data is only loaded as scroll occurs, but also by pre-fetching to hide latency.

1.8 Limitations

The backend, as it is not the main focus of the course, was not a priority. It does not persist data across restarts, and always starts with a predefined sample state.

For instance, if a logged user creates a pharmacy, the server re-starts, and the Application maintains the pharmacy in cache, it will display it in the Google Maps map. However, when the user clicks it, an exception is thrown, finishing the activity, because we're trying to access a pharmacy that does not exist in the server.

In normal application behaviour, pharmacies are not deleted, therefore such exception would never exist.

2 Additional Components

2.1 Securing Communication

The backend is behind the Caddy reverse proxy, which takes care of the certificate management automatically, using Let's Encrypt, for an acquired domain. The used server (one of the group member's computers) also hosts other applications, and the reverse proxy allows the usage of the same domain/ip for multiple applications, not requiring the use of a dedicated ip:port for the PharmacIST backend.

2.2 Meta Moderation

Following the specification, whenever a user flags a pharmacy, it will stop appearing for them. This is done, in the backend, by filtering out the pharmacies the authenticated user has flagged. Because the endpoint is public, it will only filter (and allow the user to flag) if they are authenticated. When a threshold is reached (3 for demonstration simplicity purposes), the pharmacy is disabled, not showing up for anyone, not even guests.

Any user that has a threshold of disabled pharmacies (3 for demonstration simplicity purposes) will have their account disabled and be locked out of it - the login is disabled.

2.3 User Accounts

A simple solution was set up to allow accounts to be persisted on the backend. It simply uses a $Map < Id, User >$, and stores information in plaintext, as this is not a cybersecurity class. It also stores the list of favourite pharmacies and subscribed medicines.

This way, app clients can log in and register new accounts, and even use their account in different mobile phones.

2.4 Social Sharing To Other Apps

Two buttons are displayed on the pharmacy details page: one with the email icon, for users to share through email; and one with a share icon, where users can select the application they want to share to (like Instagram, Facebook, Twitter, ...)

2.5 UI Adaptability: Rotation

Whilst the application was only tested on three devices (two physical and the emulator), the design allows for screen size variations. It also supports rotation, updating the components so that the content is seen clearly.

2.6 UI Adaptability: Light/Dark Theme

Jetpack Compose's components adapt themselves to the theme being used by the system. For future work, a toggle for changing the theme independently of the system preference would be a more flexible solution.

2.7 Translations

Android Resources are used for translations: one for English (default), and Portuguese. Only static strings were translated, i.e., strings that come from the backend are not translated, like Medicine names, and such. Places API and Google Maps also translate the addresses by themselves. As future work, an API for translation could be used for this, so when a new pharmacy comes to the backend, it would translate it into the different supported languages.

2.8 Ratings

Ratings were fully implemented, by showing a star rating bar in the pharmacy details view, that when the user clicks a star, it will rate it, automatically refreshing the histogram. Needless to say, guests may not rate a pharmacy, only view the rating.

The rating is shown all over the app when a pharmacy is displayed, just like the distance if the user has given permission.

3 Further Work

3.1 Dynamic Data Localization

By using the Google Translate API (or similar) to translate user generated data, as currently the developed application only has internationalization support for the static strings (e.g. buttons, titles)

3.2 Recommendations

No recommendation system was implemented, nor is the current database storing what is purchased by certain users, only the current stock of the pharmacies' medicines. In the future, an update to the backend service could present users with pharmacies and medicines that are closer to the users' needs, without any changes to the Android frontend.