Mypills

Memòria del projecte

Cesc Sasal i Lasaosa

Desenvolupament d'aplicacions mòbils per a IOS i Android amb Flutter (SEPE15-24)
CIFO L'HOSPITALET (**S**ervei **O**cupació **C**atalunya)
Tutor: **Eduard Carreras**

Taula de continguts

Objectiu i justificació del projecte

És una necessitat per a les persones grans que prenen moltes pastilles diàries, com la meva mare, que pren set de diferents, però també els no tan grans, doncs jo mateix fa molts anys que en prenc tres diàries diferents.

Les persones que prenen moltes pastilles diferents tenen ajuda en algunes farmàcies, però no totes les farmàcies ho fan, i moltes cobren per aquest servei. També es dona el cas d'algunes revisions freqüents de certes medicacions, i no es pot modificar el que ja s'ha preparat en la farmàcia, i el cas de persones que no prenen tantes pastilles, senzillament fer servir un pastiller i es programen ells mateixos alarmes en el mòbil... però aquestes alarmes no sempre són una bona solució, i són sovint ignorades.

Per descomptat l'usuari serà principalment una persona d'edat avançada, i sovint tindrà alguna persona que li ajudarà, qui podrà fer servir més fàcilment els apartats de configuració (actualment són senzills perquè no incorporen les posologies, però la introducció del nom dels medicaments ja pot ser més complicada).

Descripció general de la app

● Descripció general de la app (Què he fet i com ho he fet a grans trets)

És una aplicació per gestionar les alarmes per a persones que prenen una medicació, i tenen establerta una posologia; actualment es troba limitada a preses *diàries* de pastilles (*pendent d'ampliació*).

S'han definit unes «alertes» i una configuració que consisteix en dos horaris setmanals diferents. Pot triar-se els dies de la setmana per a un horari o per a l'altre. El cas general és associar les pastilles amb l'horari de menjar, però existeixen casos particulars: quan s'ha de prendre abans de menjar (per millorar l'absorció), quan s'ha de prendre després de menjar, principalment per evitar la ingesta amb l'estomac buit, i un quart cas, comptant el cas habitual indistint d'abans o després, que requereix prendre les pastilles separades dels menjars, per afavorir l'absorció.

Fonamentalment l'aplicació té dues vessants:

1. La configuració. La part visual de l'aplicació hi seria sobretot aquí, perquè pot requerir programar fins i tot els medicaments receptats per mostrar una representació visual en les pantalles d'alarma. No s'han fet grans avenços en aquest apartat.
2. La notificació a pantalla sencera, amb possibles futures accions (el disseny inicial tenia tres pantalles, però probablement n'hi ha prou amb dues).

La part important de l'aplicació és la seva arquitectura, fent servir un connector per gestionar la programació d'alarmes, i un altre per mostrar la notificació en pantalla sencera, fins i tot amb el dispositiu bloquejat.

Les alarmes es programen per parelles: una acció per activar-la, i una altra per desactivar-la, però cada alarma és independent; un cas d'ús seria tenir quatre alarmes (es a dir, vuit en total), que, a més podrien incorporar alguna alarma que no fos diària (punt encara no programat). En total, un cas requeriment de molta diversitat d'alarmes no seria estrany que incorporés vuit alarmes (setze reals, perquè van per parelles), i difícilment superaria les dotze (vint-i-quatre reals).

(nota: algun punt del codi fa que en el moment del lliurament les alarmes programades presentin problemes, a més que només s'ha implementat una opció per a cada menjar. Afortunadament les proves en debug funcionen perfectament).

Funcionalitats desenvolupades

● Funcionalitats desenvolupades

1. Programació d'alarmes, notificació en pantalla sencera i desactivació de l'alarma, amb opció de retardar-la, tant manualment, com automàticament després d'un temps estipulat.
2. Control de les vegades que una alarma s'ha desactivat automàticament per temps sense ser atesa, considerant-la «perduda».
3. Cas d'alarma atesa per l'usuari diferenciat del cas d'alarma «perduda».

Observació referent a la versió de *debug*

Alguna funcionalitat té comportament específic en «*mode debug*». És important fer-ho constar, perquè alguna de les funcionalitats pot sobtar si es desconeix el funcionament diferenciat del «*mode debug*»:

1. El **temps acceptat per a la durada de l'alarma en la configuració** està limitat; el límit és 1/10 del valor real.
    - El temps acceptat per a la durada del retard («*snooze*») queda també alterat, atès que és funció del valor màxim acceptat per a la durada de l'alarma.
2. El **so de l'alarma** està desactivat (per evitar la molèstia de les contínues proves).
3. Existeix una **pantalla de proves**. Permet llançar una alarma (inexistent amb id=3, alarma de proves), i un botó en la pantalla principal de l'aplicació (la pantalla de configuració) permet accedir-hi.

4. La funció de «***finalitzar aplicació***» executa un «**exit**(0)» que s'evita en la versió «*release*» (la documentació d'Android indica que no ha de fer-se, i probablement podria ser un problema per publicar una aplicació.
    - Executar «exit(0)» en «mode *debug*» té l'efecte addicional de treure les funcionalitats (*callbacks*) d'alarma programades, què, segons diu la documentació del connector, queden vinculades al rellotge del sistema, tot i que no queda del tot clar a quin rellotge es refereix... l'app del sistema? Probablement, atès que les funcionalitats es programen per a dia i hora.

És cert que no he tingut temps de fer gaires proves en «versió de release», punt que queda en la llista de tasques pendents.

Arquitectura i llibreries (*connectors*)

Connectors vinculats a l'arquitectura d'alarmes:

- **Android: android_alarm_manager_plus**
    - Gestió de les alarmes; s'executen fins i tot amb el dispositiu bloquejat. Es preserven les alarmes en cas de desconnectar el dispositiu.
- **Android i iOS: futter_foreground_task**
    - Permet executar un «*servei foreground*», característica de l'arquitectura d'Android que mostra una notificació en la barra d'estat perquè els usuaris sàpiguen que l'app està executant una tasca. Aquesta aplicació ens permet obrir la pantalla de l'aplicació, convertint la notificació en una «***notificació de pantalla sencera***».
    - S'ha de destacar que la implementació en iOS és una simulació del comportament d'Android, perquè no és una característica del sistema iOS.
- **Android i iOS: flutter_ringtone_player**
    - Permet incorporar so a les alarmes

```
@pragma('vm:entry-point')
static Future<void> callback() async {
  FlutterRingtonePlayer.playAlarm(
    looping: true,
    asAlarm: true,
    volume: 1.0,
  );
}
```

Text 1: Exemple trivial d'alarma

«*Isolates*»

D'aquesta manera l'aplicació queda dividida en tres diferents «*isolates*»:

1. *Isolate* **principal de l'aplicació**.
2. *Isolate* del **servei background** gestor d'alarmes, viu fins i tot quan la pantalla de l'aplicació ja no hi és al dispositiu (vinculat amb el rellotge del sistema). Aquí s'executa el *connector* ***android_alarm_manager_plus*** , un *isolate* del sistema, en concret en el de l'*AlarmManager.* Això comporta que l'alarma pot executar codi *Dart* senzill, fins comunicar amb la nostra aplicació, sempre que s'estigui executant, però no pot executar pantalles.
3. *Isolate* del **servei de primer pla**, que es crea amb cada alarma, i es destrueix quan finalitza. Aquest *isolate* obre una notificació i la pantalla de notificació de l'aplicació (notificació de pantalla sencera). Un servei en primer pla (foreground service: ***flutter_foreground_task***), amb els permisos adequats, pot iniciar automàticament (o per requeriment de l'usuari) una instància de l'aplicació, generant l'efecte de «***notificació de pantalla sencera***». Incloc referència a un article que explica els serveis en primer pla: Foreground Services in Flutter: Run tasks when your App is minimized.

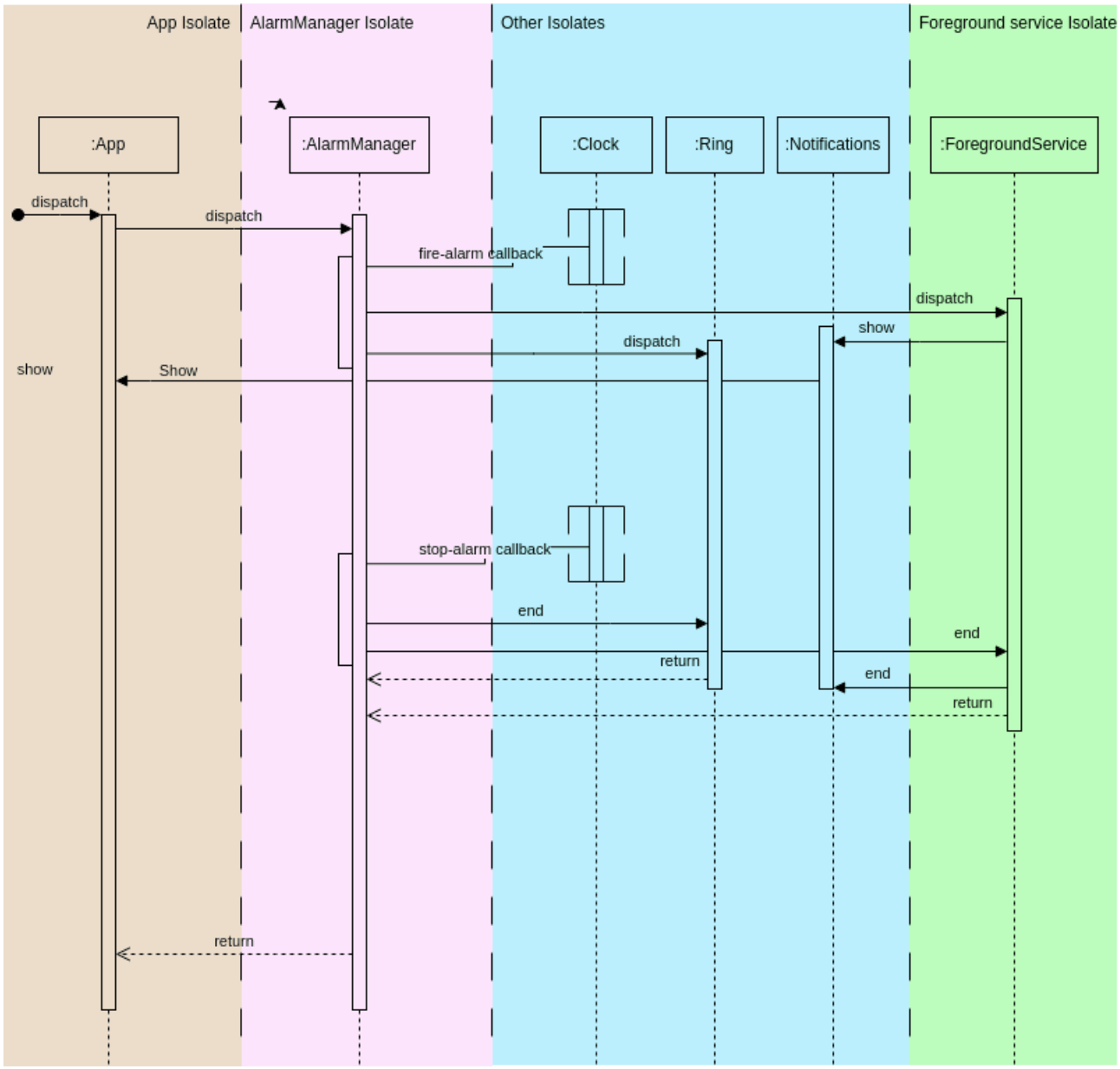


Figura 1: Exemple cicle de vida: cas alarma perduda (sense repetició)

En l'exemple es poden veure les activacions y crides més bàsiques entre els element dels tres «*isolates*», per al cas d'una alerta programada i el seu event «*auto-snooze*» també programat, per al cas concret quan actua l'«*auto-snooze*» per detenir l'alerta.

Nota: per a la categoria dels «*isolates*» del sistema (en blau) s'ha aproximat el funcionament.

Pantalles d’inici

En l’aplicació es diferencien dues pantalles principals (d’inici):

1. Pantalla de configuració, que s’obre quan manualment obrim l’aplicació.
2. Pantalla d’alarma, que s’obre com a «**notificació de pantalla sencera**».

> **App** → **/config** → defineix alarmes d’***AndroidAlarmManager***
>
> **Alarma d’*AndroidAlarmManager*** → ***FlutterForegroundTask*** → *Notificació* → **/alarm** *(obre app)*

Connectors vinculats a la configuració (dades):

- **provider** (*vist a classe*). Només es fa servir en les pantalles de configuració, perquè són les que modifiquen les dades. La notificació en pantalla sencera escriu l’objecte «*alarma*» per a preservar l’estat entre les diferents crides (activa/retardada/nombre de reintents); aquest objecte no té camps de configuració, i la pantalla de configuració només crea/destrueix aquests objectes.
- **shared_preferences** (*vist a classe*). Es fa servir la versió asíncrona, carregant les dades en la pantalla d’inici i fent servir escriptura asíncrona. La pantalla d’alarma fa servir un «*singleton*» que es recarrega quan sona l’alarma (però no quan s’atura

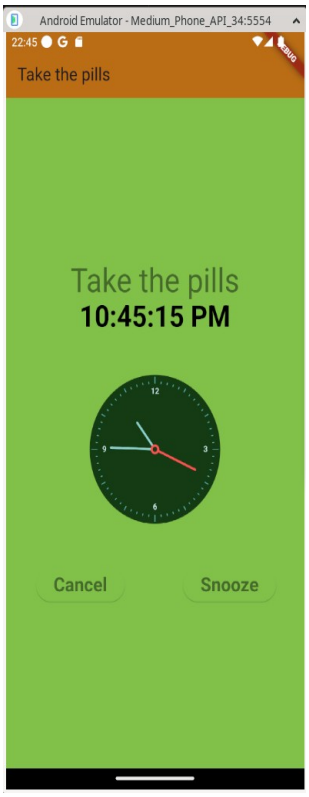
Altres Connectors

- **permission_handler**: Sol·licita permisos a l’usuari.
- **day_picker**: *Widget* visual.
- **one_clock**: *Widget* visual.
- **intl**: Per al «***DateFormat***» localitzat (noms dels mesos i dels dies de la setmana) i «***NumberFormat***».
- **logging**: Actualment només en *debug*, per mostrar informació a la consola. Previst d’incorporar-lo en noves funcionalitats.

Pantalles

● Llistat de pantalles, incloent descripció i captures de pantalla

La pantalla central és la pantalla que actua com a notificació a pantalla sencera; actualment només n'hi ha una, tot i que el disseny incorporava tres (per permetre a l'usuari prendre les pastilles amb l'alarma activada, sense so).



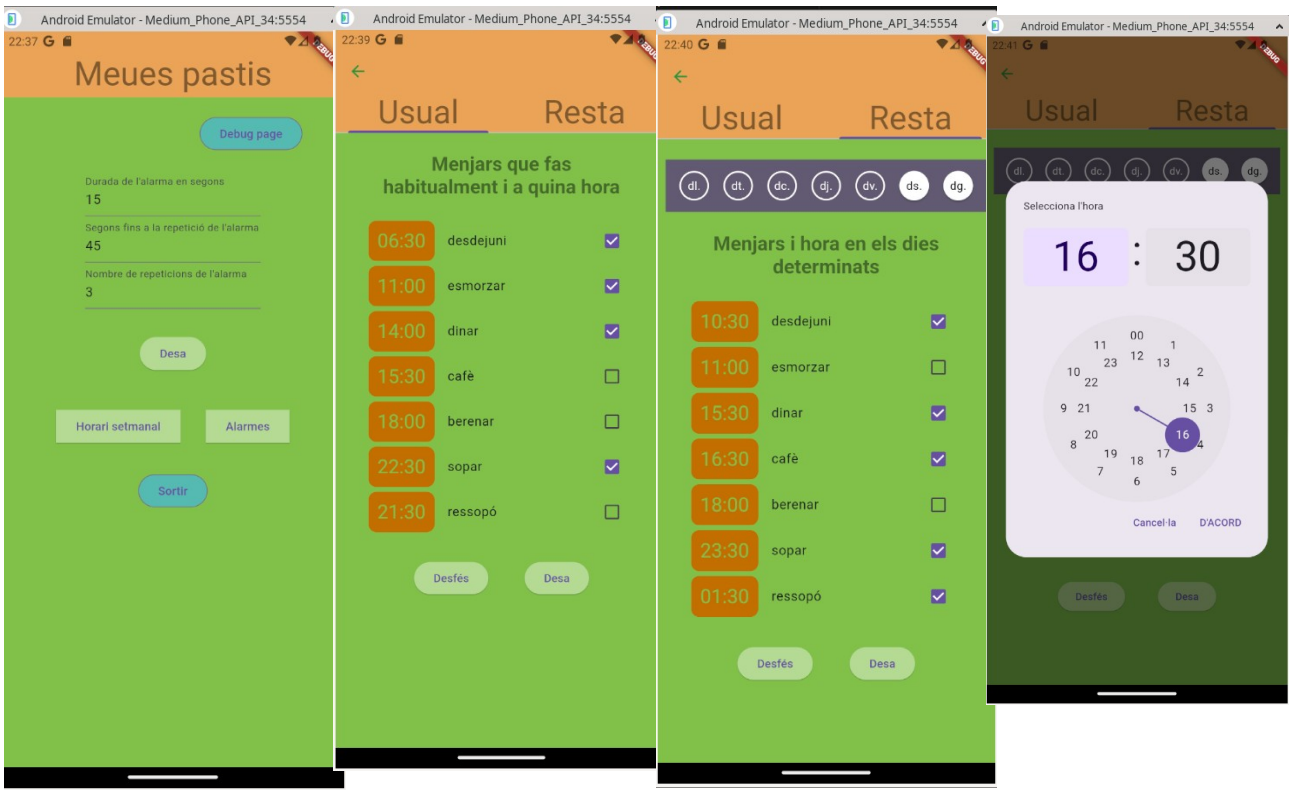
La configuració tampoc està finalitzada, i actualment consisteix en tres apartats:

1. Paràmetres bàsics (numèrics).
2. Quins menjars fem i en quin horari, així com la distribució dividint en dues alternatives els dies de la setmana.
3. Alarmes que s'han d'activar, vinculades amb els menjars.

Pantalla de paràmetres

En realitat queden alguns paràmetres avançats que no es troben encara en les pantalles: el temps abans i després de menjar segons l'enumerat «molt abans», «abans», «durant», «després» i «molt després», tot i que el valor «molt després només s'aplica a final del dia, en la resta és equivalent a «molt abans» de la menjada anterior.

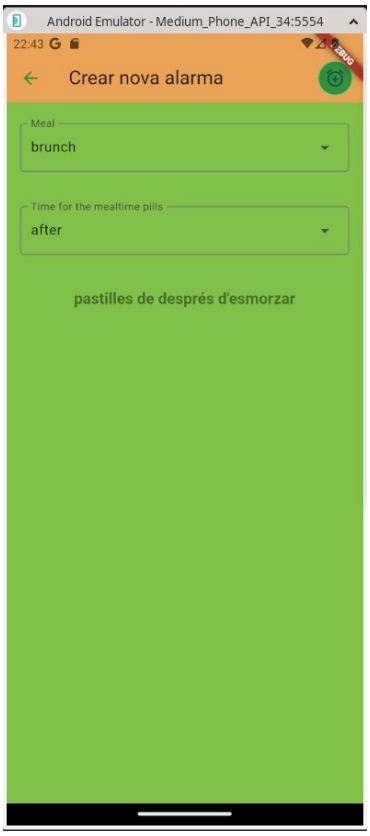
Configuració dels menjars i dies de la semana

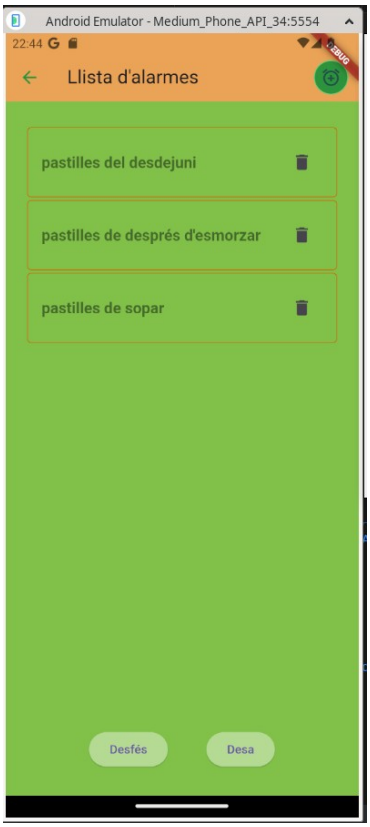


La configuració dels menjars i els horaris dels menjars es distribueix en dues alternatives, assignant els dies de la setmana.

Configuració de les alarmes

Les alarmes es configuren indicant a quin menjar s'associen, indicant a més la variació { «molt abans», «abans», «durant», «després» i «molt després» }; Només l'última menjada incorpora el cas «molt després»,





Decisions tècniques

S'ha optat per definir un tipus «alarma» amb una clau immutable, definida pel menjar al que està associada i la condició dins del menjar (molt abans, abans, durant, després, molt després). És cert que aquesta clau ha d'ampliar-se per ampliar la casuística, però segueixen sent un conjunt de casos ben definits, i ja està previst el seu disseny (però no implementat).

Té els seus inconvenients, doncs per fer una modificació es requereix la creació d'un nou objecte, però la forta vinculació amb el maquinari del dispositiu (alarmes del rellotge) fa que això fins i tot sigui un avantatge.

Evolució

● El programa té clarament funcionalitat pendent d'implementar. Desconec si després pot requerir altres funcionalitats, fora de les previstes. En concret, deixar constància de les accions de l'usuari (quan s'han perdut alarmes, quan s'han retardat, situacions de mobil apagat...), i tenir unes senzilles estadístiques seria una funcionalitat molt interessant que encara no s'ha abordat ni dissenyat.

Errors actuals

Actualment existeixen dos problemes encara no corregits, i un tercer que ha aparegut en l'últim moment:

1. S'ha activat la capacitat d'obrir la pantalla d'alarma des de la notificació, pel cas que l'usuari l'hagi tancat; tot i això, encara no s'ha implementat des de la notificació enviar el missatge amb el valor de l'alarma actualment activa. El resultat és que s'obre la pantalla, però sense els botons per parar l'alarma o retardar-la.
2. El problema de la **sincronització dels esdeveniments** que accedeixen a l'alarma en curs; aquest problema es menciona amb detall en l'apartat «*dificultats trobades*», en el subapartat «*Falta d'experiència en els mecanismes de codi asíncron i isolates*», i es proposen dues solucions, una primera, fent servir un «isolate» per al tipus de dades, i una altra, probablement més pràctica i eficient, aprofitant la persistència dels objectes «Alarma» i considerant els dos casos que no queden inclosos.
3. Actualment les alarmes han deixar de cridar a la funció que les activa/desactiva des de la configuració; probablement és un petit error provocat per les preses de l'últim moment...

Punts a optimitzar o sense acabar

S'ha cuidat força l'arquitectura de l'aplicació, tot i això, alguns detalls queden per revisar:

1. Les variacions de les alarmes (molt abans, abans, després...) no estan implementades. Tampoc s'han incorporat valors en la configuració.
2. El missatge «wakeup» s'envia dues vegades, una per cadascú dels altres dos diferents isolates existents. És un punt que requereix atenció i millora.
3. La funcionalitat de cancel·lar no està disponible en la pantalla de configuració d'horaris i dies de la setmana, tot i que és senzilla d'implementar.
4. El disseny de pantalla no està preparat per a un canvi d'orientació, i no s'ha provat exhaustivament en diferents mides de pantalla. També s'ha de provar el comportament sense l'opció «*debug*».
5. Alguns texts no s'han traduït encara, tot i que gairebé tots ho estan.
6. El codi encara no s'ha documentat (documentació de les funcions, em refereixo), tot i que té molts comentaris en l'interior; ha tingut canvis fins l'últim moment, i no ha quedat temps addicional.
7. La validació dels tres valors numèrics de la primera pantalla de configuració es van fer ràpidament; clarament es requereix un «*widget*» específic per fer la feina i no repetir codi; a més cal afegir altres valors numèrics de configuració, com ja s'ha comentat.

Desenvolupaments pendents (no iniciats)

1. Les **pantalles addicionals de gestió de l'alarma**, existents en el disseny inicial, per a millorar la usabilitat (en sortir de la primera pantalla s'atura el so de l'alarma) i l'experiència d'usuari. S'ha de considerar la tendència a ignorar una alarma que sona cada dia a la mateixa hora, i sobretot considerant que les persones a les que l'aplicació està adreçada poden ser persones d'edat avançada.
    - Tampoc es vol perdre usabilitat per a usuaris més avançats, i la **funcionalitat de «endarrerir» l'alarma es pot incorporar en un botó de la notificació**, a més de deixar-la en la pantalla principal de l'alarma, on es troba actualment (la notificació quedarà visible si s'afegeixen més pantalles de gestió de l'alarma, fins i tot si s'obre una altra aplicació).
2. La funcionalitat de «***log***» gairebé no va ser descrita en la descripció inicial, però sempre ha estat present, perquè serà una **funcionalitat important** per a que un usuari avançat, o un cuidador encarregat de la configuració de l'aplicació, pugui fer seguiment de les accions de l'usuari (si es pren les pastilles, si es perden alarmes...).
3. Fer canvis en la programació de les alarmes actualment no afecta les alarmes ja programades fins a que s'executen. Es fa necessari desprogramar les alarmes que ja no corresponen quan es modifica la configuració, per a **evitar alarmes incorrectes**.
4. En la descripció de l'aplicació es gestionaven les **prescripcions**, un punt important que no ha quedat temps per a desenvolupar.
5. Només s'ha diferenciat **alarmes en dos horaris segons els dies de la setmana**. S'ha pensat afegir també dies del calendari «especials» (festius...), però no sembla un punt important: l'horari és difícil de preveure i la feina de programar-ho no paga la pena.
    - Probablement més pràctic serà tenir més de dos horaris setmanals, per a gent que encara treballa o que té activitats programades alguns dies de la setmana; si un dels horaris s'associa amb «festiu» fins i tot podria prendre sentit tenir dies del calendari senyalats.
6. En canvi, és important incorporar **alarmes que no siguin només diàries**, perquè algunes prescripcions no són diàries, sent freqüents els casos mensuals, quinzenals o setmanals, diferenciant-se a més diferents freqüències setmanals.

Reflexió sobre l'aprenentatge

● L'aprenentatge ha estat interessant, perquè s'han explorat aspectes d'arquitectura. El projecte no ha permès practicar gaires pantalles, que és un punt que no és un dels meus forts, i em cal practicar.

Impressió personal sobre el resultat

○ En poques paraules: ajustat al previst, però curt en funcionalitats implementades. Queden principalment pantalles, així que la feina important està feta.

Dificultats trobades

Falta d'experiència en arquitectura Android i Frontend

Les principals dificultats han estat la meva ignorància de l'**arquitectura d'Android**, atès que, fora d'aplicacions *Angular* treballades en un curs de CIFO just anterior a aquest, no havia treballat i desconeixia, amb la dificultat afegida de l'enfocament des d'un punt de vista d'alt nivell (*Flutter*) desconeixent els aspectes natius, que feia que de vegades no tenia clar on buscar la informació, perquè les referències a aspectes natius concrets sovint pressuposen coneixements de l'arquitectura nativa.

També el meu coneixement en **desenvolupament «*frontend*»** és molt mínim, i en aquest sentit m'ha agradat molt l'enfocament de Flutter, perquè simplifica molt el disseny per als programadors, i els aspectes concrets del disseny no són una part fonamental per al desenvolupament inicial de l'aplicació.

Codi asíncron

Referent a dificultats concretes, vaig tenir algunes dificultats per fer el seguiment del **comportament dels connectors** quan el seu comportament era estrany; el principal problema venia d'un procediment «*async*», pel qual el meu codi no esperava (no feia servir «***await***» en la crida); això el vaig trobar i corregir gràcies a la incorporació de les opcions en «***analysis_options.yaml***», concretament **unawaited_futures**:

```
analyzer:
  language:
    strict-casts: true
    strict-inference: true
    strict-raw-types: true
  linter:
    rules:
      - unawaited_futures
```

Falta d'experiència en els mecanismes de codi asíncron i «*isolates*»

Això ens porta al gran problema de no fer servir mecanismes de sincronització: s'ha de dissenyar un «*isolate*» per forçar la sincronització; Per exemple: si de mentre que es gestiona la finalització d'una alerta, un esdeveniment requerit per l'usuari, el rellotge llança un esdeveniment per repetir l'alarma («*snooze*»). Ambdós esdeveniments s'executaran barrejats, perquè tenen instruccions

«*await*» que permeten l'execució d'altres tasques quan encara no s'ha acabat l'execució del procediment que gestionava l'esdeveniment anterior.

En el cas del programa « *MyPills*» aquest cas es dona amb els objectes de la classe «Alarma», que no s'he protegit per falta d'experiència en codi asíncron, tot i conèixer be la concurrència. La solució «ortodoxa» seria gestionar aquests objectes en un «*isolate*» separat fent servir canals com a comunicació (els canals actuen com a cues, incorporant un ordre), el codi és molt senzill si es dissenya l'arquitectura correctament des d'un inici.

Actualment, per evitar conflictes, s'ha incorporat variable en memòria, un booleà, i sembla que temporalment és una solució vàlida, perquè es veu que Dart no preserva els valors de variables després d'un «*await*», però no és una solució vàlida, atès que no és un comportament documentat.

Tot i això la **solució** tampoc és tan senzilla:

1. Les operacions de proves no es troben vinculades a un objecte de la classe «Alarma», i la solució incorporant un booleà pot resultar la més adequada, particularment per als casos de prova. La solució actual s'hauria de mantenir a més a més de qualsevol solució associada als objectes «Alarma»; posteriorment, si es requereix una situació més complexa per les proves, es pot incorporar un objecte «*Alarma*» fictici.
2. La vinculació amb l'objecte «Alarma» és important, atès que pot permetre gestionar casos on una alarma solapa amb una altra, per exemple, si es retarda repetidament una alarma propera a una altra. Fer la gestió dels esdeveniments vinculada a cadascuna de les alarmes fa que el codi sigui reentrant entre alarmes diferents, tot i que no solucionaria el cas de dues execucions simultànies de dos esdeveniments per a una mateixa alarma, tot i que aquest cas es limita a «stop» i «auto-snooze» que es podria gestionar com a cas particular, en comptes de fer servir un «isolate» separat comunicat per missatges, solució que incorpora nous problemes.
3. La pràctica requereix operacions més prioritàries que unes altres, en concret, per exemple, «stop» prioritària sobre «snooze», particularment pel cas «auto-snooze», la repetició de l'alarma. Això complica l'arquitectura (requereix alguna mena de «dispatcher»), però com l'aplicació es gestiona en «minuts» podríem dir que el cas presentat com a exemple és l'únic conflicte a resoldre, una vegada les operacions de les alarmes es gestionen independentment unes d'altres.

Arquitectura del provider

Una altra gran dificultat va ser la gestió del ***provider***, tant per haver incorporat un «*singleton*», com per haver incorporat el «*ChangeNotifierProvider*» només en una de les pantalles, la branca que ho feia servir (la branca de la configuració), en comptes de fer-ho en l'inici de l'aplicació.

Altres problemes més anecdòtics

També es pot mencionar alguns problemes ja amb el temps just per al lliurament:

- La programació dels **validadors** per als camps d'introducció de dades em va comportar algun problema.
- L'editor «*Visual Studio Code*» de vegades afegeix automàticament «***imports***»; en una ocasió va afegir una referència incorrecta que generava un error força críptic en execució, i em va costar tres o quatre hores solucionar-ho; finalment vaig recórrer a «*Android Studio*», i l'informe de l'error, més detallat, em va permetre ràpidament detectar el problema.

Canvis fets des del document d'Anàlisi previ

○ Ja s'ha comentat que la pantalla d'alarma no es troba acabada, i queden les altres (dues) pantalles per programar; aquestes dues pantalles tenen certa relació amb les prescripcions mèdiques, que tampoc s'han fet.

En general crec que el desenvolupament s'ha ajustat molt al previst, tot i que la implementació ha quedat una mica curta, i queden funcionalitats per implementar; el desenvolupament s'ha centrat en els aspectes d'arquitectura, que alguns no eren encara clars en el document previ (no es pot dissenyar sense un bon coneixement de les eines, en aquest cas, dels connectors).

Reflexió general.

Què m'ha aportat el desenvolupament del projecte?

Per a mi ha estat la primera aproximació seriosa a l'arquitectura dels mòbils, sent l'anterior curs d'Angular la meva única experiència prèvia. També el meu coneixement de les arquitectures «*frontend*» pràcticament es limita algun lloc web en els inicis d'Ajax, allà per l'any 2000, sense haver tornat a desenvolupar interfícies d'usuari. En aquest sentit el curs m'ha agradat força, perquè l'enfocament de «*Flutter*» trobo que és molt més racional que els diferents intents «unificadors» que ha hagut des de l'aparició d'Ajax en Web; sé que existeixen altres tecnologies també interessants, en concret «*Kotlin-Multiplatform*» i «*Compose-Multiplatform*», que seria un equivalent a *Flutter*, però en codi «*natiu*», tot i que no les conec en absolut.

Aportacions del curs al teu futur professional

- El cert és que el mon d'*Angular* em va decebre una mica, perquè la seva aplicació a dispositius mòbils te moltes limitacions. En el cas de *Flutter* veig que també existeixen limitacions, sobretot si es vol fer una aplicació multiplataforma que requereix punts d'arquitectura molt personalitzats, perquè pots trobar que els connectors no hi són disponibles per a totes les plataformes, però per a dissenyar aplicacions que principalment requereixen pantalles, accedir al dispositiu i alguns connectors molt freqüents, *Flutter* és una

gran solució que, a més, simplifica una gran problemàtica: el disseny de les pantalles que ja no han de dibuixar-se (recordo que informàtica era la única llicenciatura d'enginyeria que no incorporava el dibuix tècnic com assignatura de primer curs).