UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeries
Industrial i Aeronàutica de Terrassa

DESENVOLUPAMENT D'UNA APLICACIÓ ANDROID PER AL MICROSCOPI AISCOPE

Memòria

Màster Universitari en Enginyeria Industrial

NOM DE L'ESTUDIANT: Xavier Gavarró Busquets

NOM DEL DIRECTOR: David Gonzalez Diez

DATA D'ENTREGA: 27 d'abril de 2020

Declaració d'honor

I declare that,

the work in this Master Thesis / Degree Thesis (choose one) is completely my own work,

no part of this Master Thesis / Degree Thesis (choose one) is taken from other people's work without giving them credit,

all references have been clearly cited,

I'm authorised to make use of the research group related information I'm providing in this document.

I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by *The Universitat Politècnica de Catalunya - BarcelonaTECH*.

Student: Xavier Gavarró Busquets

Signature:

Date: 27/04/2020

Title of the Thesis: Desenvolupament d'una Aplicació Android per al Microscopi AiScope

Llista de Continguts

Llista d'Il·lustracions

Llista de Taules

1.Introducció

1.1. Marc del projecte

Aquest projecte està estretament vinculat amb AiScope, una iniciativa sense ànim de lucre que des de fa uns anys ha estat desenvolupant tecnologia de codi obert per tal de crear una eina de diagnosi de malalties infeccioses. El resultat d'aquest desenvolupament és un microscopi intel·ligent que utilitza intel·ligència artificial per a millorar el diagnosi d'algunes de les malalties infeccioses més perilloses.

La idea darrera aquest desenvolupament és que, per exemple, la malària, la tuberculosi i els paràsits intestinals són tres de les deu malalties més mortíferes a les zones amb menys recursos (més de 3 milions de morts anuals). Aquestes malalties tenen un tractament senzill si es detecten a temps, però en comunitats amb menys recursos, la falta de diagnosi provoca moltes morts evitables.

El microscopi està dissenyat per a utilitzar-se amb un telèfon mòbil. Aprofitant la càmera incorporada als dispositius i utilitzant les lents del microscopi, es pot fotografiar les mostres de sang per a després analitzar-les amb el software de diagnosi, també desenvolupat per AiScope.

Addicionalment, s'hi ha afegit una sèrie de motors i mecanismes que permeten desplaçar, ampliar i enfocar la mostra. I aquí és on entra l'aplicació desenvolupada en aquest projecte.

L'objectiu principal de l'aplicació és el de proporcionar una eina per a controlar de forma senzilla els motors del microscopi. Com que el microscopi mateix ja està dissenyat per a utilitzar-lo amb la càmera d'un mòbil, el més lògic era desenvolupar una aplicació mòbil que en permetés el control a la vegada que es fa fotografies de la mostra.

A banda d'aquest objectiu principal, però, també es va creure convenient proporcionar a l'usuari de l'aplicació altres serveis des de la mateixa interfície, com la de gestionar les fotografies preses amb l'aplicació, accés als plànols de construcció del microscopi, o l'opció de fer donacions per a donar suport a la iniciativa.

Un altre aspecte a tenir en consideració previ al desenvolupament, és que l'empresa fabricant de mòbils BQ tenen un pacte amb AiScope, on els primers els proporcionaran telèfons mòbils per a la utilització dels microscopis, de manera que l'aplicació havia de ser, com a mínim, compatible amb dispositius Android.

1.2. Objectius

L'objectiu del projecte és el de desenvolupar un prototip d'aplicació d'Android funcional que permeti el següent:

- Utilitzar la càmera del telèfon mòbil per a fer fotografies
- Rebre els inputs de l'usuari per a controlar els microscopis AiScope
- Gestionar les imatges preses amb l'aplicació
- Proporcionar a l'usuari la documentació necessària per a la fabricació i construcció d'un microscopi
- Facilitar el contacte amb l'equip d'AiScope
- Facilitar les donacions econòmiques per a donar suport a la iniciativa

1.3. Abast

Es contemplen dins l'abast del projecte els següents aspectes:

- Definició de les necessitats d'AiScope
- Definició i disseny de l'arquitectura de l'aplicació
- Cerca d'informació i formació en les tecnologies necessàries per a desenvolupar el programa definit a l'arquitectura
- Desenvolupament de l'aplicació
- Definició dels passos a seguir una vegada finalitzat el projecte

Queden fora de l'abast del projecte els següents aspectes:

- Integració amb els microscopis AiScope. Queda fora de l'abast per l'aïllament provocat pel COVID-19, que ha fet impossible l'obtenció d'un microscopi per a fer el desenvolupament.
- Disseny gràfic de les interfícies d'usuari, només es proporcionarà una interfície simple

1.4. Requeriments

Els requeriments del projecte són els següents:

- El temps de dedicació aproximada al projecte és de 360h
- La data d'entrega de la memòria és el 27 d'abril de 2020
- L'aplicació ha d'estar alineada amb les necessitats d'AiScope
- L'aplicació ha de ser compatible amb dispositius de telèfon Android
- Tota la gestió s'ha de poder realitzar des de la pròpia aplicació

1.5. Context

A dia d'avui, una de les causes més altes de mortalitat és la falta de diagnosi. Per posar algun exemple, la malària, la tuberculosi i els paràsits intestinals es troben entre les 10 malalties més mortíferes a les regions menys desenvolupades quan es disposa d'un tractament senzill si es detecten a temps.

Per a posar-ho en perspectiva, l'OMS estima que a nivell mundial hi ha uns 3400 milions de persones de 91 països diferents que corren el risc de contraure la malària, 1100 milions d'aquests tractant-se d'un risc alt. La zona més afectada per aquesta malaltia és la regió d'Àfrica, on es produeixen un 91% de les morts per malària a nivell mundial, i d'on dues terceres parts d'aquests són nens menors de 5 anys.

Arrel d'aquestes dades i amb l'objectiu de combatre aquesta falta de diagnosi neix la iniciativa AiScope, una iniciativa sense ànim de lucre que treballa per a desenvolupar un microscopi intel·ligent de codi obert capaç de diagnosticar aquestes malalties.

Il·lustració 1. Logotip d'AiScope

Tot comença quan Eduardo Peire va descobrir el Fold Scope. Fold Scope és un microscopi plegable que buscava revolucionar la diagnosi portàtil. Aquest prototip, però, té una sèrie de problemes.

El primer i més important és que per a utilitzar-lo es necessita personal qualificat, fet que comporta una important despesa de temps i de diners, recursos que precisament no disposen les regions més necessitades del producte. L'altre problema era que si bé aconseguia ampliar la imatge, aquesta quedava massa deformada com per a poder distingir els paràsits de la mostra.

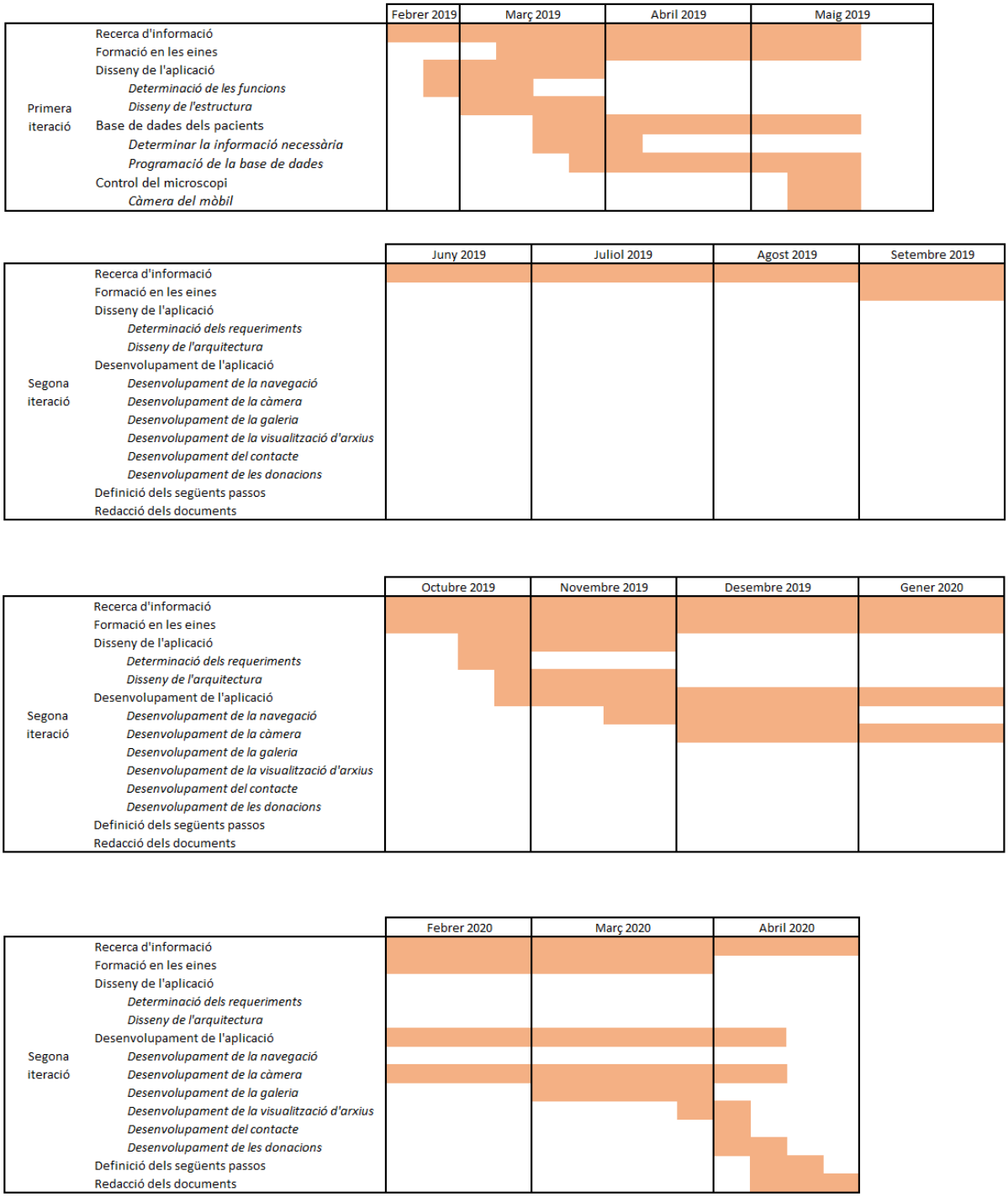
Així doncs, el primer objectiu que es va fixar va ser el d'aconseguir una ampliació de la imatge de 1000x augments sense perdre nitidesa i mantenint un cost de producció baix. Al mateix temps que es va començar a dissenyar aquest nou microscopi, també va desenvolupar un software basat en Python i utilitzant mètodes de machine learning que permet el diagnosi de les malalties a partir de fotografies.

Il·lustració 2. Prototip actual de l'AiScope

L'AiScope està dissenyat per a ser compatible amb un telèfon mòbil de manera que utilitzant la càmera del dispositiu es pot fer fotografies de la mostra ampliada. Addicionalment, l'última iteració del microscopi disposa d'una sèrie de motors i mecanismes que permeten enfocar i desplaçar la mostra, de manera que el següent pas lògic era el de desenvolupar una aplicació per a mòbil que permetés integrar-ho tot en un sol dispositiu.

1.6. Evolució temporal

Taula 1. Diagrama de Gantt

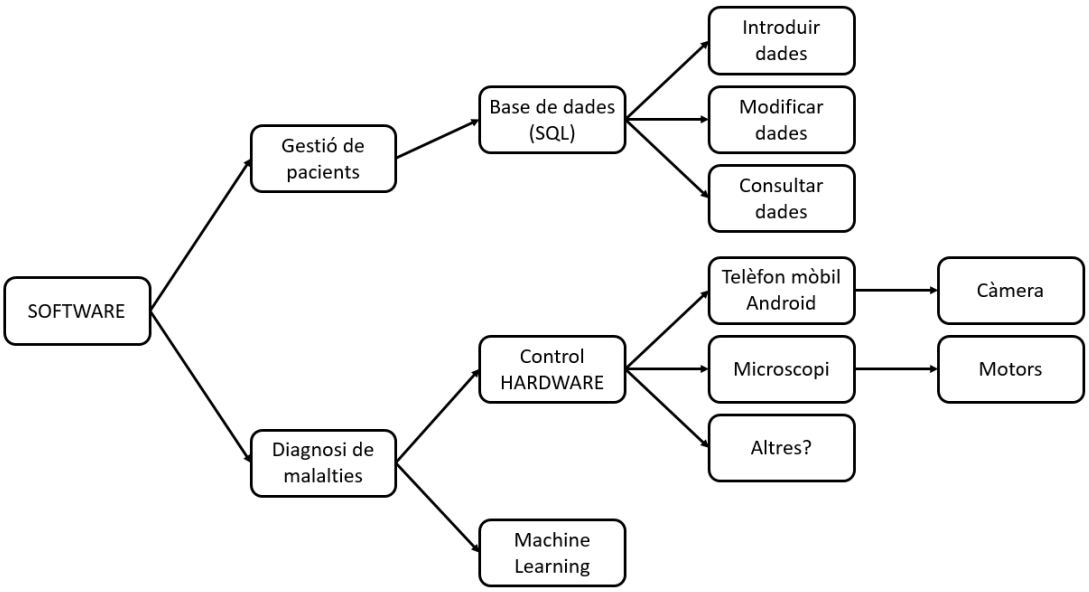


2. Estat de l'art

2.1. Evolució de l'aplicació

Inicialment el projecte s'encara amb la intenció de desenvolupar-lo amb Python principalment perquè era un llenguatge del que ja tenia coneixement i experiència.

Aquesta primera fase té l'objectiu de desenvolupar una aplicació a utilitzar-se per a realitzar diagnosis al terreny, motiu pel que, a banda de desenvolupar la utilització de la càmera també es dissenya una base de dades per a emmagatzemar informació.



Il·lustració 3. Disseny de la primera iteració de l'aplicació

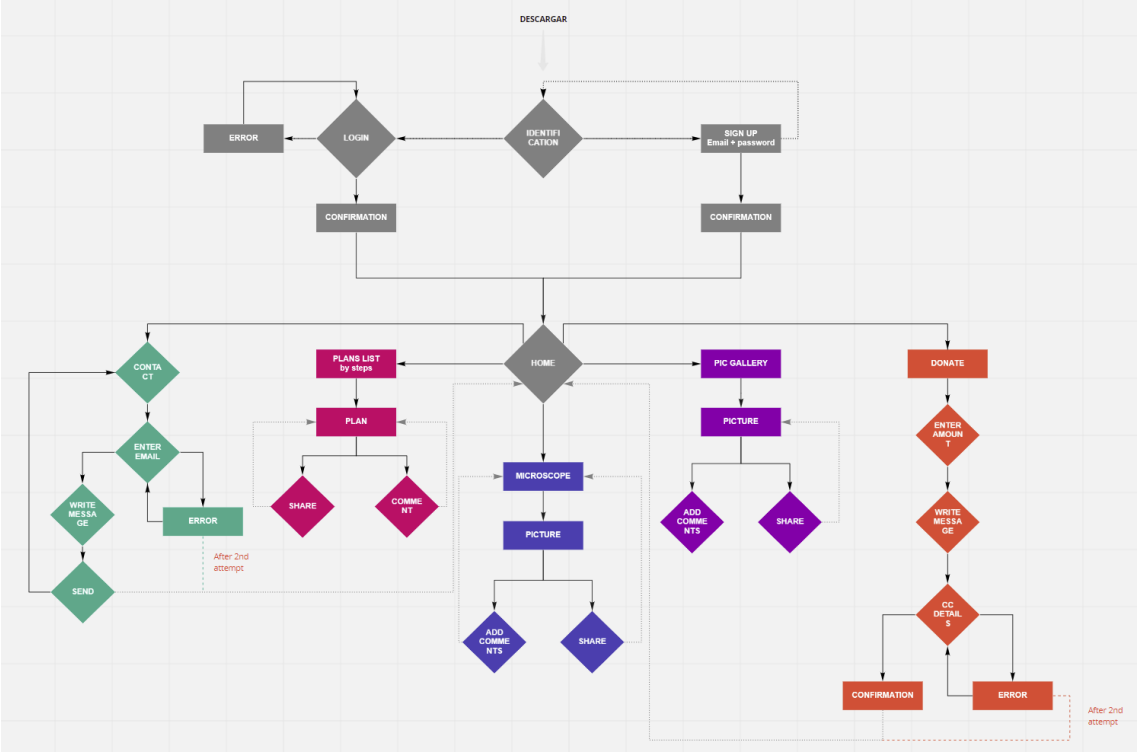
S'encara el desenvolupament prenent de base la llibreria Kivy, de Python, dissenyada especialment per a desenvolupar aplicacions multi-plataforma i per a facilitar la introducció i recepció d'inputs a la pantalla per part de l'usuari.

Aquest enfocament, però, comporta un problema que s'ha acabat demostrant insuperable, ja que Python no disposa de cap paquet que permeti la utilització de la càmera de dispositius Android, de manera que s'acaba descartant la iteració i es replanteja el projecte.

Durant la transició entre iteracions, també hi ha un replantejament per part d'AiScope de quin era l'objectiu que havia de tenir l'aplicació. Aquest passa de

ser una aplicació de camp a una aplicació amb l'objectiu fer conèixer la iniciativa i incentivar-ne la participació.

Així doncs, la segona iteració de l'aplicació acaba sent pràcticament un projecte diferent, encara que la base d'aquesta es segueix mantenint: ús i control del microscopi a través de la càmera del telèfon.



Il·lustració 4. Disseny de la segona iteració de l'aplicació

2.2. Android

Android és un sistema operatiu per a telèfons mòbils i tauletes que entra en escena l'any 2007 amb l'objectiu de poder competir al mateix nivell que Apple i el seu iOS que en aquella època semblava que acabaria amb el monopoli total dels smartphones.

El que va començar sent un sistema destinat a telèfons mòbils i tauletes tàctils s'ha acabat utilitzant en una gran varietat de dispositius, convertint-se en el principal proveïdor del mercat. Per a posar-ho en perspectiva, a data de març de 2020 Android controlava el 72,26% del mercat internacional de tecnologia mòbil comparat amb el 27,03% d'iOS.

Il·lustració 5. Logotip d'Android

Android en sí es comença a desenvolupar per una empresa independent amb el mateix nom, Android Inc. fins que Google decideix comprar-la l'any 2005. Coincidint amb la sortida al mercat del sistema operatiu, es crea també l'OHA (*Open Handset Alliance*), un consorci format per les principals empreses de fabricants de mòbils i altres dispositius intel·ligents i que s'ha encarregat de desenvolupar i donar suport al sistema operatiu des de llavors.

2.3. Desenvolupament per Android

Des d'un primer moment, Android ha donat suport al desenvolupament actiu d'aplicacions per als seus dispositius publicant una sèrie d'API a tots els aparells que permeten als desenvolupadors utilitzar-les per a consultar informació del dispositiu o enviar peticions.

Així doncs, per a desenvolupar una aplicació per Android el primer que s'ha de fer és desenvolupar un codi que gestioni aquestes API dels dispositius per, d'aquesta manera, poder interactuar amb el dispositiu.

Aquest procés, però, requereix d'uns coneixements i una inversió de temps importants, de manera que per a facilitar la feina als desenvolupadors i no obligar a interactuar directament amb les API dels dispositius, Android va desenvolupar una sèrie de llibreries de Java que fan precisament aquesta funció.

El fet de ser un llenguatge suportat de manera oficial implica que Java ha sigut durant molt de temps el llenguatge més utilitzat per a la programació en Android, ja que les llibreries estan sempre al dia amb les versions del sistema operatiu.

L'any 2017 Android va començar a reconèixer Kotlin com a segon llenguatge oficial, de manera que des de llavors també li ha estat donant suport mantenint les llibreries actualitzades.

A causa d'aquest suport oficial d'Android per a Java i Kotlin, la gran majoria d'aplicacions estan escrites en un d'aquests dos llenguatges, però això no ha evitat iniciatives per a facilitar la programació en d'altres com per exemple Python.

2.4. Llenguatges de programació

2.4.1. Python

Python el crea Guido van Rossum a principis de la dècada dels 90 amb la finalitat de proporcionar als usuaris un llenguatge de programació d'accés obert i centrat principalment en la llegibilitat del codi, és a dir, en facilitar la feina dels programadors.

Il·lustració 6. Logotip de Python

De fet, la filosofia darrera del llenguatge es resumeix en un document anomenat *The Zen of Python*, una col·lecció dels 19 principis de programació que regeixen el funcionament i l'evolució de Python. Les més rellevants són les següents:

> *Beautiful is better than ugly.*
> *Explicit is better than implicit.*
> *Simple is better than complex.*
> *Complex is better than complicated.*
> *Readability counts.*

Els elements que aconsegueixen aquesta facilitació per l'usuari són els següents:

- Alta flexibilitat

  Python permet diversos tipus de programació segons la necessitat de l'usuari. Ja sigui orientada a objectes o estructural

- Alta adaptabilitat

Era habitual en els inicis de la programació que els llenguatges estiguessin formats per un sol paquet que contingués totes les funcions i tipus de dades disponibles per a aquell llenguatge.

Python, en canvi, va ser un llenguatge pioner en basar-se en l'estructura d'un nucli de codi senzill, on només s'hi incloguessin les funcions indispensables, i incloure la possibilitat d'estendre les funcionalitats mitjançant paquets o llibreries.

El fet de tenir el llenguatge estructurat d'aquesta manera facilita moltíssim el manteniment de les llibreries existents i el desenvolupament de noves, ja que es pot treballar en cada una d'elles individualment. Actualment la gran majoria dels llenguatges en ús han adoptat aquesta estructuració

- Gramàtica senzilla

Python busca proporcionar una gramàtica i sintaxi més simple i ordenada que els altres llenguatges de l'època, jugant amb els espais en blanc al codi i utilitzant paraules en anglès on d'altres utilitzen signes de puntuació.No es pot optimitzar tot

Una de les coses que tenia clares el seu creador, és que no es pot ser el millor en tots els aspectes, així que des d'un inici s'ha permès als desenvolupadors canviar de llenguatge en determinades seccions i processos.

Un exemple és en la velocitat de funcionament: en algunes situacions concretes, C és més àgil executant-se que Pyhton, de manera que si es prioritza la velocitat més que la llegibilitat, el llenguatge ofereix eines per a fer la traducció.

2.4.2. Kotlin

Kotlin és un llenguatge que surt d'una iniciativa impulsada per JetBrains, justificant-lo perquè no existia al mercat un llenguatge que tingués totes les característiques que els interessaven. L'únic que s'hi apropava era un llenguatge anomenat Scala, però aquest tenia la pega de tenir uns temps de compilació molt alts.

Arrel d'això, Kotlin agafa de base Java, un llenguatge que destaca en especial per la velocitat de compilació. De fet, Kotlin es dissenya per a ser executat en una JVM (*Java Virtual Machine*), una màquina virtual dissenyada per a executar Java en ordinadors.

Il·lustració 7. Logotip de Kotlin

La primera versió oficial surt a principis de l'any 2016 i just l'any següent Google anuncia que proporcionarà suport oficial pel llenguatge, i dos anys més tard, el 2019, anuncia que Kotlin passa a ser el llenguatge preferit per a desenvolupar aplicacions. Tant és així, que s'espera que en un futur no gaire llunyà es deixi de donar suport oficial a Java i que Kotlin passi a ser l'únic llenguatge amb suport de primer nivell.

Una de les avantatges de Kotlin respecte Java, és que la sintaxi és molt més concisa. Per a posar-ho en perspectiva, el software desenvolupat amb Kotlin ocupa un 40% menys de volum que el de Java.

A més introdueix les variables que no poden tenir un valor nul, evitant així els NPE (*null pointer exception*), un dels problemes crítics més habituals que es produeixen amb java. Addicionalment, els dos llenguatges són 100% interoperables, és a dir, des d'una línia de codi en Java es pot cridar funcions en Kotlin i viceversa.

La seva estructura és la mateixa que l'adoptada per Python en el sentit que està format per un nucli base amb les funcions indispensables i es pot afegir funcionalitats amb la importació de llibreries.

A diferència d'altres llenguatges, Kotlin no tindrà problemes de comptabilitat amb Android ja que Google proporciona suport de primer nivell pel llenguatge, de manera que és una aposta segura a mig i llarg plaç.

2.4.3. Elecció del llenguatge

El llenguatge escollit inicialment per a desenvolupar el projecte és Python pel principal motiu que era el llenguatge amb el que tenia experiència i coneixements, decisió que acaba resultant un error que porta a haver de descartar la feina realitzada durant 3 mesos.

Per a la segona iteració, es fa una recerca més extensiva de l'estat del desenvolupament en Android i s'acaba escollint Kotlin, un dels llenguatge suportat oficialment per Google. Aquesta decisió acaba sent força lògica i en retrospectiva, la que s'hauria d'haver pres des d'un inici.

Si bé és cert que en comparació amb Java hi havia la remota possibilitat que algun dels elements de l'aplicació no es pogués completar perquè en el moment d'iniciar el projecte encara no tenia totes les llibreries desenvolupades, el fet que es tracti d'un llenguatge interoperable amb Java presenta la solució a aquest problema. La decisió queda reforçada quan Google anuncia que Kotlin passa a ser el llenguatge recomanat per al desenvolupament d'Android.

Taula 2. Comparació entre llenguatges de programació

	Python	*Java*	*Kotlin*
Coneixements a l'inici del projecte	Sí	No	No
Llibreries per a desenvolupament Android	Parcial	Sí	Sí
Suport oficial de Google	No	Sí	Sí
Garantia de desenvolupament	No	Sí	Parcial
Interoperabilitat	No	No	Sí
Garantia de futur	No	Parcial	Sí

2.5. Entorn de desenvolupament (IDE)

De la mateixa manera que al llarg del temps s'ha anat desenvolupant i millorant els editors de text, on cada vegada realitzen més funcions i proporcionen ajuda als usuaris, amb la programació ha passat exactament el mateix.

Si bé és cert que teòricament per a desenvolupar software només es necessitaria un editor de text simple com el bloc de notes, per exemple, a banda del codi hi ha moltíssimes tasques més que s'han de realitzar.

Aquestes tasques venen facilitades pels entorns de desenvolupament o IDE (*integrated development environament*), software dissenyat específicament per a desenvolupar software. D'IDEs n'hi ha moltíssims i de molts tipus: alguns es poden utilitzar per a múltiples llenguatges i d'altres es centren únicament en una tecnologia.

Cada un té les seves avantatges, però al final tots realitzen una sèrie de tasques comunes, independentment del llenguatge o tecnologia:

- Suport pel llenguatge

  L'IDE identifica el llenguatge que s'està utilitzant i proporciona les eines necessàries per a importar les llibreries (i descarregar-les si és necessari), així com eines de *linting*. Una eina de *linting* és l'equivalent a un corrector ortogràfic d'un programa d'ofimàtica, marca els errors de gramàtica i sintaxi.

- Destacament del codi

  Virtualment tots els IDE proporcionen una eina similar, on el software detecta el tipus de paraula que s'ha redactat. S'acostuma a destacar de colors diferents els constructors, els comentaris, les funcions, les variables i els tipus de dades. Alguns *linters* també proporcionen informació addicional per a facilitar la lectura.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    // Inflate the layout for this fragment
    val view : View! = inflater.inflate(R.layout.fragment_gallery, container, attachToRoot: false)

    recyclerView = view.findViewById(R.id.galleryRecyclerView)
    navController = Navigation.findNavController(
        requireActivity(), R.id.nav_host_fragment
    )

    viewModel = ViewModelProvider(requireActivity()).get(GalleryViewModel::class.java)
    viewModel.imageData.observe(viewLifecycleOwner, Observer {...})
    layoutManager = GridLayoutManager(activity, spanCount: 3)
    recyclerView.layoutManager = layoutManager

    return view
}
```

Il·lustració 8. Fragment de l'aplicació on s'aprecia els canvis de color del codi

- Compleció de paraules

A mesura que s'escriu el mateix IDE detecta activament a les llibreries disponibles o a altres seccions del software les funcions i variables que coincideixen amb les paraules escrites, i dona l'opció d'autocompletar la paraula amb les diverses opcions.

- Debugging

Una dels principals obstacles que es plantegen a l'hora de desenvolupar són els bugs, errors en el codi. El procés de debug és el procés pel qual es detecta els errors que es produeixen a l'execució per a poder-ne trobar l'origen i arreglar-los. Els IDE proporcionen eines que faciliten aquesta tasca.

- Control de versions

Qualsevol software al que s'estigui donant suport activament requereix de desenvolupament constant, encara que sigui de manteniment. Per tal de disposar de punts de control i còpies de seguretat, els IDE proporcionen un control de versions del software integrat.

Pel projecte s'ha hagut de treballar amb dos IDE diferents. Per a la primera etapa de Python, l'escollit va ser el Visual Studio Code, mentre que per l'aplicació definitiva en Kotlin s'ha utilitzat Android Studio.

2.5.1. Visual Studio Code

Visual Studio Code és un IDE desenvolupat per Microsoft dissenyat per a operar amb Windows, Linux i macOS. És un software publicat sota una llicència de codi obert altament personalitzable gràcies a la possibilitat d'instal·lar-hi extensions que afegeixen funcionalitats.

És gràcies a les extensions que es tracta d'un dels IDE més versàtils disponibles i utilitzats actualment. De fet, l'any 2019 es va convertir en el més utilitzat per a desenvolupar just 4 anys després de la seva primera publicació l'any 2015.

La principal avantatge de la que disposa és la seva gran versatilitat gràcies a la instal·lació d'extensions, fet que el fa compatible amb pràcticament tots els llenguatges.

2.5.2. Android Studio

Android Studio és l'IDE desenvolupat i promocionat per Google per a desenvolupar aplicacions d'Android ja sigui en Java o Kotlin. És un IDE especialment dissenyat per a desenvolupar aplicacions Android, independentment del dispositiu objectiu.

Addicionalment, té una sèrie d'eines especialment útils per al desenvolupament Android. Les més destacades són les següents:

- Constructor basat en Gradle
- Emulador virtual de dispositius Android
- Debugging en temps real d'aplicacions
- Plantilles de codi per al desenvolupament de components habituals
- Estructura i organització automàtica dels arxius del projecte
- Generació de codi automàtic per a determinades funcionalitats

Si bé no es tracta d'un software de codi obert, sí que és de lliure accés i descàrrega, de manera que s'ha convertit, inevitablement, en l'IDE més popular per al desenvolupament en Android.

2.6. Llibreries destacades

2.6.1. Kivy

Com ja s'ha comentat en una secció anterior, Android només reconeix dos llenguatges: Java i Kotlin, els únics als que dona suport oficial. Això vol dir que per a programar aplicacions d'Android en altres llenguatges, s'ha de desenvolupar llibreries que interaccionin amb les API dels dispositius.

Una d'aquestes llibreries és Kivy, una llibreria de Python i de codi obert destinada a desenvolupar aplicacions amb interaccions de l'usuari amb la pantalla. Aquesta llibreria permet la interoperabilitat entre diferents sistemes operatius: Android, iOS, Linus, Windows, OS X i Raspberry Pi, és a dir, el mateix codi pot funcionar en pràcticament qualsevol dispositiu.

Il·lustració 9. Logo de Kivy

La gran avantatge d'aquesta llibreria és que permet desenvolupar aplicacions multi-touch de manera senzilla ja que conté suport per a tractar inputs provinents de ratolí, teclat físic o virtual, pantalla o botons dels dispositius.

Addicionalment proporciona una sèrie d'elements predeterminats per a la interacció amb l'usuari, i introdueix un nou llenguatge per a dissenyar la interfície gràfica molt influenciat per l'html i el javascript, el kivy.

Cal destacar, però, que és una llibreria que funciona i es manté activa gràcies al desenvolupament i la dedicació d'un grup d'usuaris d'internet, de manera que es córrer el risc que un dia decideixin de donar-hi suport i, per tant, quedi obsoleta.

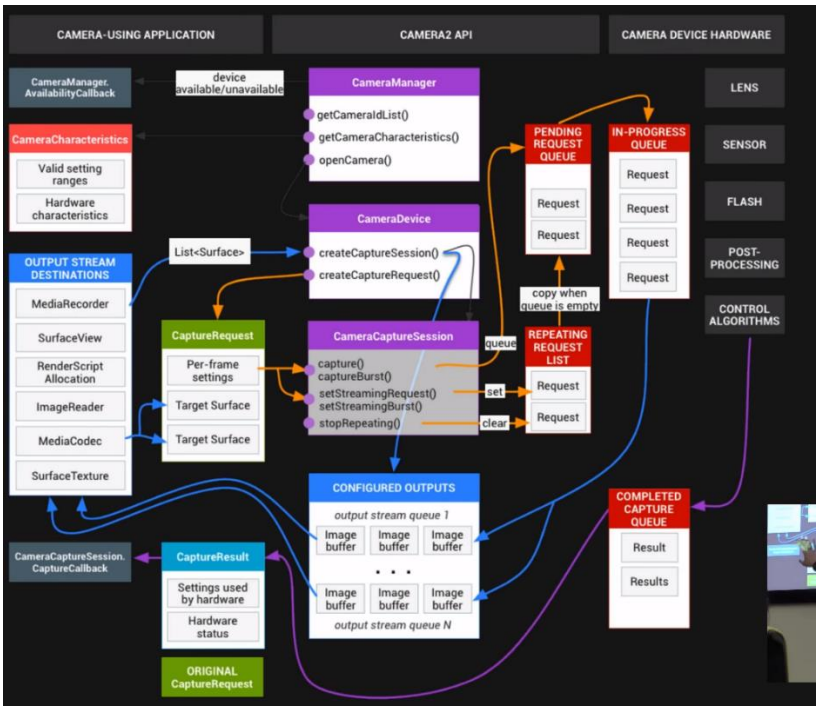
2.6.2. CameraX

La configuració i la utilització de la càmera és un dels desenvolupament més complexos per Android que hi ha actualment, i tot això és a causa de l'API publicada per a la càmera.

L'API estàndard per a la càmera és la publicada sota el nom Camera2 i és una evolució de l'original implementada amb la versió 21 de les API d'Android. La versió original era relativament senzilla d'utilitzar, simplement es seleccionava la càmera que es volia utilitzar i s'enviaven peticions de realitzar fotografies o gravar vídeos.

Si bé és cert que era senzilla d'utilitzar, a la vegada era bastant limitada. Fora de les funcions que es proporcionaven de forma estàndard, no es podia fer gran cosa i la càmera era com una caixa negra: funcionava però no es sabia ben bé com.

Amb l'objectiu de donar accés als desenvolupadors a controls de la càmera més avançats, Android va publicar l'API Camera2 que dona accés a funcions i controls més complexos de la càmera però que a la vegada elimina la funcionalitat antiga. Això implica un augment més que considerable de la dificultat de desenvolupaments relacionats amb la càmera.



Il·lustració 10. Flux d'utilització de l'API Camera2

Aquesta dificultat ha provocat que el projecte s'estanqui durant força temps en aquesta etapa, on l'única solució possible era l'adquisició dels coneixements necessaris per a utilitzar l'API Camera2.

A l'octubre de 2019, Google anuncia el llançament d'una nova llibreria CameraX amb l'objectiu de facilitar l'ús i el desenvolupament de la càmera. Aquesta es publica Extret de la documentació oficial de la llibreria:

"CameraX is a Jetpack support library, built to help you make camera app development easier. It provides a consistent and easy-to-use API surface that works across most Android devices, with backward-compatibility to Android 5.0 (API level 21).

While it leverages the capabilities of camera2, it uses a simpler, use case-based approach that is lifecycle-aware. It also resolves device compatibility issues for you so that you don't have to include device-specific code in your code base. These features reduce the amount of code you need to write when adding camera capabilities to your app."

Aquesta llibreria retorna a una funcionalitat similar a la que proporcionava l'API de la càmera original, simplificant-ne altra vegada la implementació i desenvolupament, i a la vegada desencallant el projecte.

S'ha de tenir en compte que encara no hi ha cap versió estable publicada, l'aplicació tot just utilitza una versió 1.0.0-alpha06 i per tant s'ha de contemplar la possibilitat que es produeixi algun error durant l'execució. Queda pendent com a tasca futura actualitzar la versió de la llibreria a una d'estable quan quedi publicada.

3. Primera iteració

3.1. Descripció general

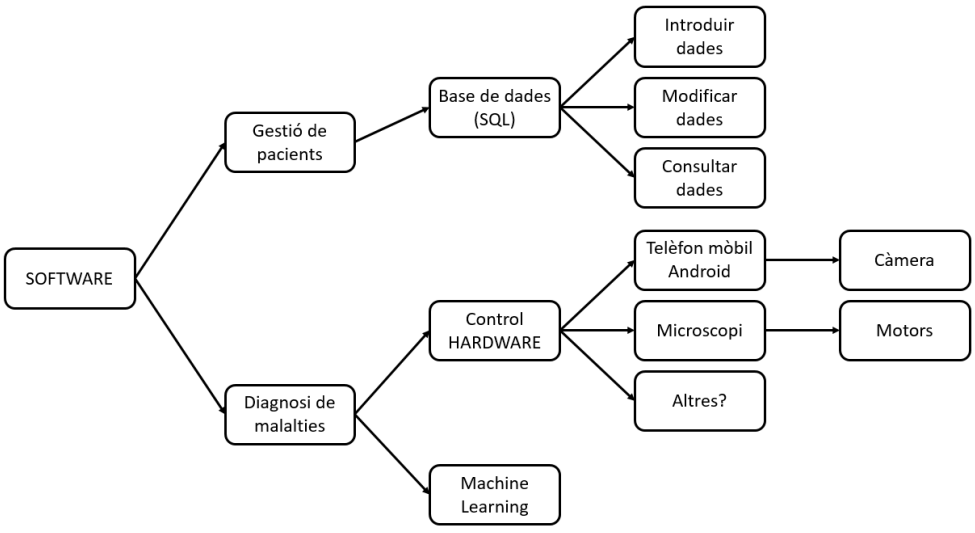
La primera versió de l'aplicació tenia l'objectiu de ser l'eina de camp per al diagnosi de les malalties utilitzant el microscopi. Aquesta tenia quatre objectius principals:

- Integrar l'ús de la càmera i el control del microscopi
- Disposar d'una base de dades amb la informació dels pacients, fotografies fetes i diagnosi
- Integrar aquesta base de dades local amb una altra base de dades global a través d'internet
- Valorar la possibilitat d'implementar un software de diagnosi

Aquesta primera iteració es va començar a desenvolupar amb Python utilitzant Visual Studio Code com a IDE. El projecte es va dividir en 3 seccions principals:

- Disseny i desenvolupament d'una base de dades
- Desenvolupament del control de la càmera i el microscopi
- Integració del software de diagnosi

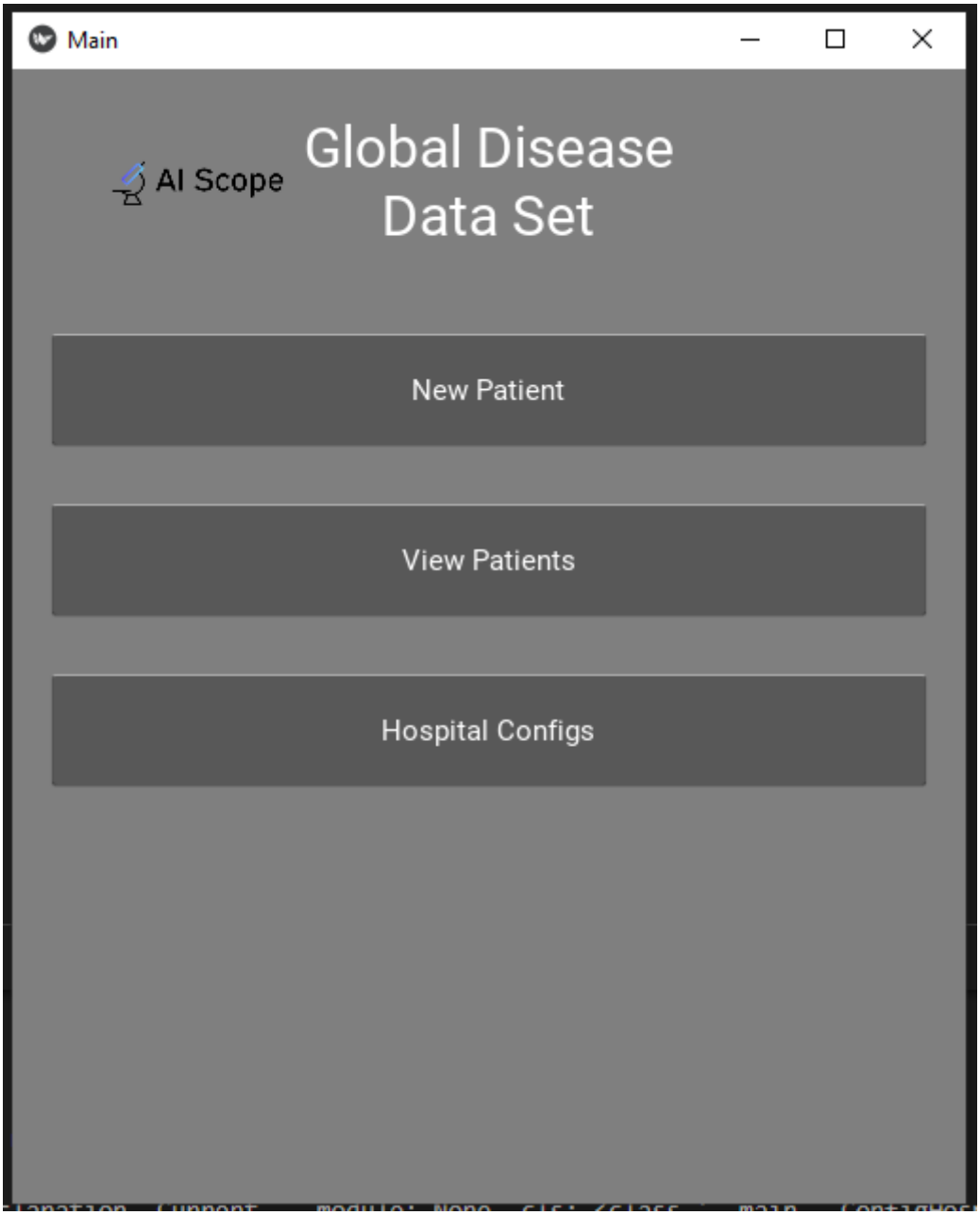
Tenint en compte aquestes característiques i objectius de l'aplicació, es va procedir a fer un disseny estructural bàsic de l'aplicació:



Il·lustració 11. Arquitectura de la primera iteració

Per a la navegació i la interacció de l'usuari amb les diverses funcions de l'aplicació s'ha utilitzat la llibreria Kivy, pel suport que proporciona amb les aplicacions multi-touch de dispositius smartphones.

La navegació s'ha anat desenvolupant a mesura que les seccions han estat completades, de manera que només s'ha arribat a configurar part de la navegació corresponent a la secció de la base de dades.



Il·lustració 12. Pantalla principal de la gestió de pacients (executat amb Windows).

3.1.1. Tasques programades

A partir de l'arquitectura inicial, es va definir una sèrie de tasques per a la realització del projecte:

1. Base de dades dels pacients
   1.1. Determinar la informació necessària
      1.1.1. Tipus de dades
      1.1.2. Tipus d'entrada
   1.2. Programació de la base de dades
      1.2.1. Creació de la taula
      1.2.2. Sistema d'entrada de nous pacients
      1.2.3. Sistema de modificació de pacients existents
      1.2.4. Transferència de dades amb altres dispositius
2. Control del microscopi
   2.1. Càmera del mòbil
      2.1.1. Accés a càmera
      2.1.2. Realitzar fotografia
   2.2. Motors del microscopi
      2.2.1. Moviments bàsics
      2.2.2. Moviments limitats
   2.3. Coordinació càmera/motors
      2.3.1. Configurar moviments predeterminats
3. Diagnosi de malalties
   3.1. Adaptació software original
      3.1.1. Lectura i comprensió del software
      3.1.2. Adaptació de software
   3.2. Diagnosi a partir de les fotografies preses
      3.2.1. Determinar format del diagnosi
      3.2.2. Determinar el tipus d'accés a les fotografies
      3.2.3. Coordinar control microscopi amb software

3.2. Base de dades

L'objectiu final de l'aplicació és el de proporcionar una eina amb la qual diagnosticar malalties a un nombre elevat de persones, de manera que per a poder dur un seguiment acurat d'aquests diagnosis és indispensable disposar d'una base de dades.

Per al disseny de la base de dades, el primer que s'ha de tenir clar és quants entitats o objectes diferents interaccionen amb l'aplicació, ja que per a cada un d'ells haurà de disposar d'una taula per a dipositar-hi la informació.

La base de dades definida és només una versió inicial, hauria fet falta una revisió dels camps i valors de cada un dels objectes.

3.2.1. Objectes

3.2.1.1. Diagnosticador

Persona que utilitzarà l'aplicació i el microscopi per a fer els diagnosis dels pacients i responsable de gestionar la base de dades amb la informació dels pacients i dels diagnosis. Hauria de contenir informació com:

- Nom
- Cognoms
- Data de naixement
- Coneixements mèdics i/o tècnics
- Nivell de responsabilitat
- Identificador de diagnosticador únic

```
class Location(Base):
    '''Location database declaration'''
    __tablename__ = 'locations'

    id = Column(Integer, primary_key=True)
    name = Column(String(50))
    country = Column(String(50))

    def __repr__(self):
        return "<Location(name='%s', country='%s', identifier='%s')>" % (
            self.name, self.country, self.identifier)
```

Il·lustració 13. Declaració de la taula relacional de Localitats

3.2.1.2. Localitat

Els diagnosis es realitzen en hospitals quan és possible, però en alguns pobles o llocs específics no n'hi ha. Així doncs, enlloc de referir-se a hospitals a la base de dades reben el nom de Localitat d'on és necessari disposar d'algunes dades que, sobretot, són necessàries per a agrupar els pacients. La informació que ha de contenir és:

- Nom de la localitat
- País
- Regió
- Id de localitat únic

3.2.1.3. Pacient

Persona a la que se li ha realitzat un o més diagnosis. La informació que ha de contenir és:

- Nom
- Cognoms
- Data de naixement
- Qüestions mèdiques rellevants
- Localitat (relació amb l'objecte)
- Identificador de pacient únic

3.2.1.4. Diagnosi

Cada interacció operada per un Diagnosticador a un Pacient concret. La informació que ha de contenir és:

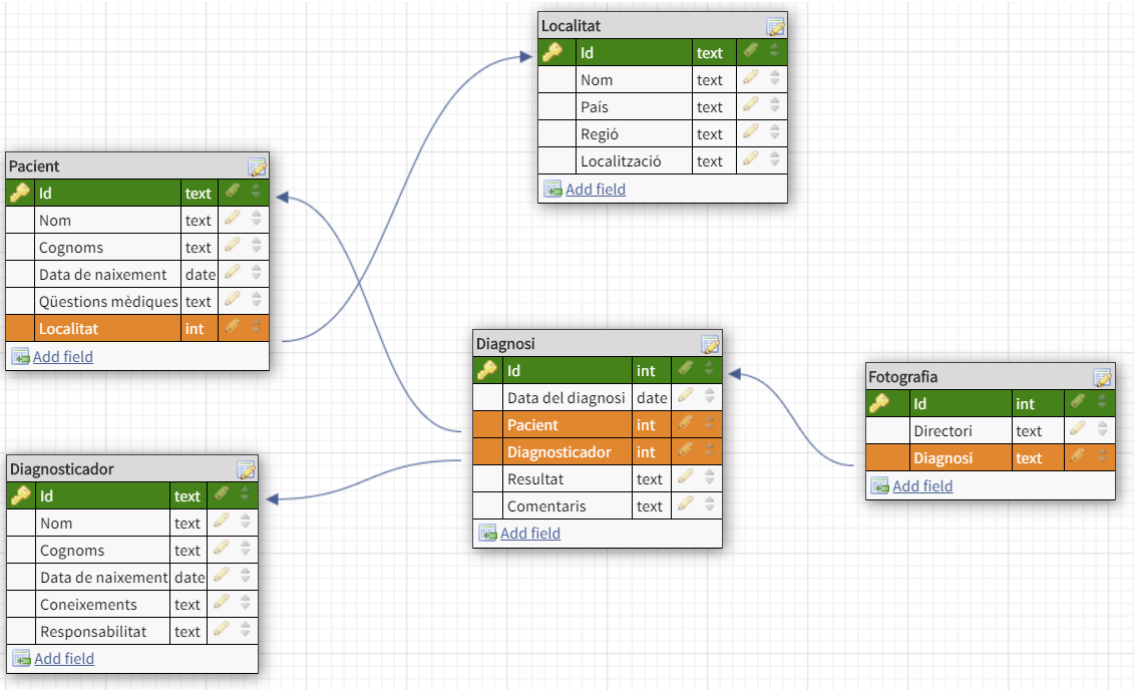
- Data del diagnosi
- Pacient (relació amb l'objecte)
- Diagnosticador (relació amb l'objecte)
- Tipus de mostra
- Resultat
- Espècies de bactèria
- Comentaris
- Identificador de diagnosi únic

3.2.1.5. Fotografia

Cada diagnòstic està format per una o més fotografies de la mostra, de manera que també s'ha de disposar d'informació sobre aquestes fotografies per a poder-les explotar. La informació que ha de contenir és:

- Directori de la fotografia
- Diagnosi (relació amb l'objecte)
- Identificador de fotografia únic

Així doncs, l'esquema de la base de dades queda així:



Il·lustració 14. Esquema de la base de dades

3.2.2. Desenvolupament de la base de dades

Una vegada dissenyat l'esquema de la taula de dades, el següent pas és desenvolupar-la. La tecnologia escollida per a fer-ho és SQLalchemy, una llibreria de Python per a la creació i gestió de bases de dades relacionals.

La generació de la base de dades amb no és especialment complicat, el que sí que comporta més complexitat és la integració amb l'aplicació i les interaccions amb l'usuari.

El funcionament de la secció es divideix en dues parts, cada una dirigida per un tipus d'arxiu diferent i es poden diferenciar entre el software en sí i la interfície gràfica.

3.2.2.1. Software

El software és el que s'encarrega d'executar les funcions i la lògica de l'aplicació i està íntegrament escrit en Python. Està format per diversos arxius organitzats en carpetes (o paquets) per a mantenir un ordre i facilitar la lectura i gestió posterior.

Per a desenvolupar la base de dades s'ha creat 3 arxius en Python:

- main.py
- database.py
- database.db

3.2.2.1.1. main.py

És l'arxiu central de l'aplicació i el primer que s'executa a l'iniciar l'aplicació. Està format per diverses classes, una per a cada element de la llibreria de Kivy que s'ha utilitzat.

Cada una d'aquestes classes conté com a mínim un mètode d'inicialització, on la primera vegada que s'executa se li assignen els valors i les interfícies gràfiques necessàries per al seu funcionament.

Pul funcionament de Kivy, cada element visual que es declara a la interfície gràfica ha de disposar d'una classe pròpia declarada a l'iniciar l'aplicació. És també en aquestes classes on s'assignen les interaccions rebudes de l'usuari a les funcions necessàries per a impactar a la base de dades.

```
def set_id(self, hosp):
    '''function to save the designed hospital in a global variable'''
    global CURRENT_HOSPITAL
    query = session.query(Hospital).filter(Hospital.id.like(hosp))
    CURRENT_HOSPITAL = query.one()
    print(CURRENT_HOSPITAL.name)
```

Il·lustració 15. Exemple de funció que posteriorment s'assignarà a un botó

Una altra de les funcions d'aquesta funció principal, és la d'importar i executar el codi contingut a l'arxiu database.py, l'encarregat de la creació i gestió de la base de dades.

3.2.2.1.2. database.py

Aquest arxiu basa el seu funcionament en la llibreria SQLalchemy i s'encarrega de la gestió de la base de dades.

El primer que es fa a l'arxiu és establir una connexió a l'arxiu database.db, arxiu que contindrà l'estructura i la informació introduïda per l'usuari. Una vegada realitzada la connexió, es declara la base de dades objecte per objecte.

Una de les avantatges de la llibreria és que compara activament l'arxiu database.db amb l'estructura que s'està declarant, de manera que si coincideixen els noms dels objectes i els camps exactes, els detecta com a ja creats i simplement estableix una connexió.

En canvi, si detecta alguna diferència entre l'estructura declarada a la classe i el que es troba a la base de dades, actualitza el segon amb l'estructura del primer mantenint el màxim de la informació ja declarada i sempre que sigui possible.

Cada objecte es declara com si fos una classe, heretant les característiques d'una taula relacional de la llibreria ('Base'). Juntament amb la classe, també es declaren totes les variables que ha de contenir l'objecte, indicant el tipus de camp que es tracta.

```
class Case(Base):
    '''Particular cases studies class declaration'''
    __tablename__ = 'cases'

    id = Column(Integer, primary_key=True)
    hospital_id = Column(Integer, ForeignKey('hospitals.id'))
    micro_settings = Column(Integer, ForeignKey('microscopesettings.id'))
    name = Column(String(50))
    age = Column(Integer)
    sex = Column(String)
    pregnant = Column(Boolean)
    nationality = Column(String)
    hemogoblin = Column(String)
    others = Column(String)
```

Il·lustració 16. Declaració d'una classe per a la base de dades

Addicionalment, a tots els objectes de la base de dades se'ls configura una funció que el que fa és retornar una línia de text amb els valors d'un registre concret, d'aquesta manera es pot visualitzar els valors des de la funció principal main.py.

```
def __repr__(self):
    return "<Case(hospital_id='%s', micro_settings='%s', name='%s', \
        age='%s', sex='%s', pregnant='%s', nationality='%s', \
        hemogoblin='%s', others='%s')>" % (
            self.hospital_id, self.micro_settings, self.name, self.age, \
            self.sex, self.pregnant, self.nationality, self.hemogoblin, \
            self.others)
```

Il·lustració 17. Funció per a retornar els valors d'un registre

3.2.2.1.3. database.db

Aquest arxiu és el que conté les dades introduïdes per l'usuari i l'aplicació des d'on l'aplicació es comunica activament per a guardar i recuperar dades. Si aquest arxiu es perd o es corromp, es perden totes les dades.

3.2.3. Interfície gràfica

La interfície gràfica de l'aplicació s'ha desenvolupat utilitzant el llenguatge introduït per la llibreria Kivy i que porta el mateix nom. És un llenguatge que fa recordar molt a l'html o a l'xml per la seva manera d'estructurar els elements.

Quan s'executa el codi principal, contingut a l'arxiu main.py, una de les funcions que es realitza és inicialitzar la llibreria Kivy. El que fa aquesta és buscar al directori on hi ha l'arxiu del codi per un altre que tingui el mateix nom però canviant l'extensió per "kv". En el cas de main.py, l'aplicació busca l'arxiu anomenat main.kv i automàticament l'associa com la interfície gràfica.

A l'arxiu main.kv es declaren i es defineixen tots els objectes i elements que s'utilitzen a l'aplicació. També és l'encarregada de gestionar la navegació entre pantalles i des d'on s'associen les funcions declarades a la funció principal als diferents botons.

El primer que es declara és el gestor de navegació, un ScreenManager. En aquesta secció s'enumeren les diferents pantalles que conté l'aplicació i se'ls associa un id, utilitzat després per la navegació.

```
ScreenManagement:
    id: manager
    transition: FadeTransition()
    StartScreen:
        id: startscreen
        name: 'Start Screen'
    PatientScreen:
        id: patientscreen
        name: 'Patient Screen'
    HospitalScreen:
        id: hospitalscreen
        name: 'Hospital Screen'
    HospitalConfig:
        name: 'Hospital Configuration'
```

Il·lustració 18. Declaració del ScreenManager

Una vegada enumerades les pantalles, aquestes s'han de declarar. Dins d'aquesta declaració també és on es declaren els diferents components i les funcions que realitzen de ser necessari.

```
<StartScreen@Screen>:
    StackLayout:
        canvas.before:
            Color:
                rgba: .5,.5,.5,1
            Rectangle:
                pos: self.pos
                size: self.size
        BoxLayout:
            size_hint: 1,.2
            spacing: 10
            AsyncImage:
                source: 'aiscope.png'
                pos_hint: {'center_x': .5, 'center_y': .5}
                size_hint: 1,1
            Label:
                text: 'Global Disease DataSet\nVersion 0.2'
                valign: 'top'
                halign: 'right'
        FloatLayout:
            size: self.size
            pos: self.pos
            Button:
                text: 'START'
                font_size: 40
                size_hint: .3,.2
                pos_hint: {'center_x': .5, 'center_y': .7}
                on_release: app.root.current = 'Patient Screen'
```

Il·lustració 19. Declaració d'una pantalla en Kivy.

Per exemple, a la imatge anterior es pot veure un exemple de declaració d'una pantalla, com indica l'inici de la declaració (<StartScreen@Screen>). Destacar també la funció de destacar el text, es pot distingir els diferents elements elements segons els color de ràpidament.

Per a aquesta pantalla en concret s'ha utilitzat 3 layouts diferents: Stack, Box i Float, amb la característica que el BoxLayout i el FloatLayout queden compresos dins del primer.

Dins el BoxLayout es declaren dos elements, el primer és una imatge de l'icona d'AiScope carregat al directori de l'aplicació, i el segon es tracta simplement d'una etiqueta de text.

Finalment al FloatLayout s'està declarant un botó, el botó d'inici de la secció de la base de dades de l'aplicació. Se li assigna la funció de viatjar a la pantalla "Patient Screen" segons es premi el botó.

Similarment a la pantalla que s'ha descrit, s'ha fet les declaracions de la resta a mesura que la base de dades i l'aplicació en ha estat evolucionant.

3.2.4. Control del microscopi

El control del microscopi es pot dividir a la seva vegada en dues seccions diferenciades:

- Control de la càmera del telèfon mòbil
- Control del microscopi

Al final, el microscopi es controla a partir de les imatges que es veuen a la pantalla, de manera que el més lògic és desenvolupar el control de la càmera i posteriorment afegir-hi el control del microscopi.

Aquest punt, però, ha acabat resultant impossible de solucionar sense una inversió molt considerable de temps, i fins i tot llavors sense cap assegurança d'acabar funcionant.

Actualment no existeix cap llibreria estable que permeti utilitzar la càmera d'un telèfon Android directament des del mateix telèfon mòbil. És cert que hi ha certes alternatives, com per exemple utilitzar una xarxa local i connectar-hi tant el

telèfon com un ordinador, llavors des de l'ordinador es pot accedir a les imatges de la càmera. Totes les solucions, però, impliquen el trencament d'un dels requeriments:

- Tota la gestió s'ha de poder realitzar des de la pròpia aplicació

Degut a que la utilització de la càmera ha resultat ser virtualment impossible amb Python, la primera iteració s'ha acabat desestimant com a una possibilitat real.

4. Transició entre iteracions

Arribat al punt on ha deixat de ser factible continuar desenvolupant l'aplicació en Python, s'ha hagut de fer un replantejament complet de l'aplicació, ja que l'enfocament inicial no era l'indicat.

Així doncs, el primer que s'ha de fer és una recerca més detallada i extensa de l'estat de l'art en desenvolupament d'aplicacions Android i de les eines concretes que s'hauran d'utilitzar per a desenvolupar l'aplicació, aquesta vegada sense entrebancs.

Els resultats d'aquesta recerca han quedat detallats a la Introducció del document, però la conclusió final és que el més adient per a desenvolupar aplicacions en Android és utilitzar les eines suportades oficialment ja que aquestes asseguren la possibilitat d'utilitzar tots els recursos disponibles en un dispositiu mòbil.

Aquest període de transició entre iteracions de l'aplicació ha estat marcat, bàsicament, per recerca i formació en el llenguatge escollit (Kotlin), l'eina de desenvolupament (Android Studio) i les diverses llibreries i funcionalitats concretes que s'han acabat utilitzant a la versió presentada.

Durant aquest període, es comunica a l'equip d'AiScope que el projecte quedaria ajornat uns mesos per a la necessitat de canviar de tecnologia, petició a la qual no posen cap entrebanc.

Una vegada s'ha començat a reprendre el projecte, es reactiva la comunicació amb AiScope, on es decideix redirigir els objectius de l'aplicació, fet que també provoca una necessitat de redissenyar-la per complet.

5. Segona iteració

5.1. Descripció general

Durant la fase de transició entre les iteracions, des d'AiScope es replanteja la funció i la finalitat que ha de tenir l'aplicació. El que inicialment havia de ser una eina per a fer diagnosis al terreny capaç d'emmagatzemar informació dels pacients i diagnosis realitzats, es decideix que sigui una aplicació per a fer conèixer la iniciativa a un major nombre de persones.

Es redissenya l'aplicació per a ser utilitzada com una eina d'oci, per a impulsar més persones a col·laborar amb la iniciativa i a fer-ne conèixer l'existència. Addicionalment, la seva funcionalitat principal, que segueix sent l'ús i control del microscopi a través de la càmera del telèfon, també servirà com a base per a l'aplicació que finalment s'acabarà utilitzant en el terreny.

Les funcions que ha de tenir aquesta nova iteració són:

- Realització de fotografies mitjançant la càmera del telèfon
- Control dels microscopis a partir d'inputs a la pantalla
- Gestió de les fotografies realitzades, possibilitat de compartir-les a les xarxes socials
- Eines per a impulsar la col·laboració amb AiScope

Una altra de les funcions que inicialment s'hi volia implementar és la possibilitat de registrar-se com a un usuari de l'aplicació, d'aquesta manera AiScope disposaria d'una base de dades amb gent interessada en col·laborar i a qui es podria impactar amb campanyes de correu electrònic, per exemple.

Aquesta idea es va desestimar de seguida perquè per la normativa de la Unió Europea sobre la protecció de dades dels usuaris (GDPR) és molt estricta i requereix d'un equip legal per a gestionar bases de dades, un recurs del que actualment AiScope no disposa.

Així doncs, amb els objectius principals de l'aplicació marcats, el següent pas és definir el llistat de tasques.

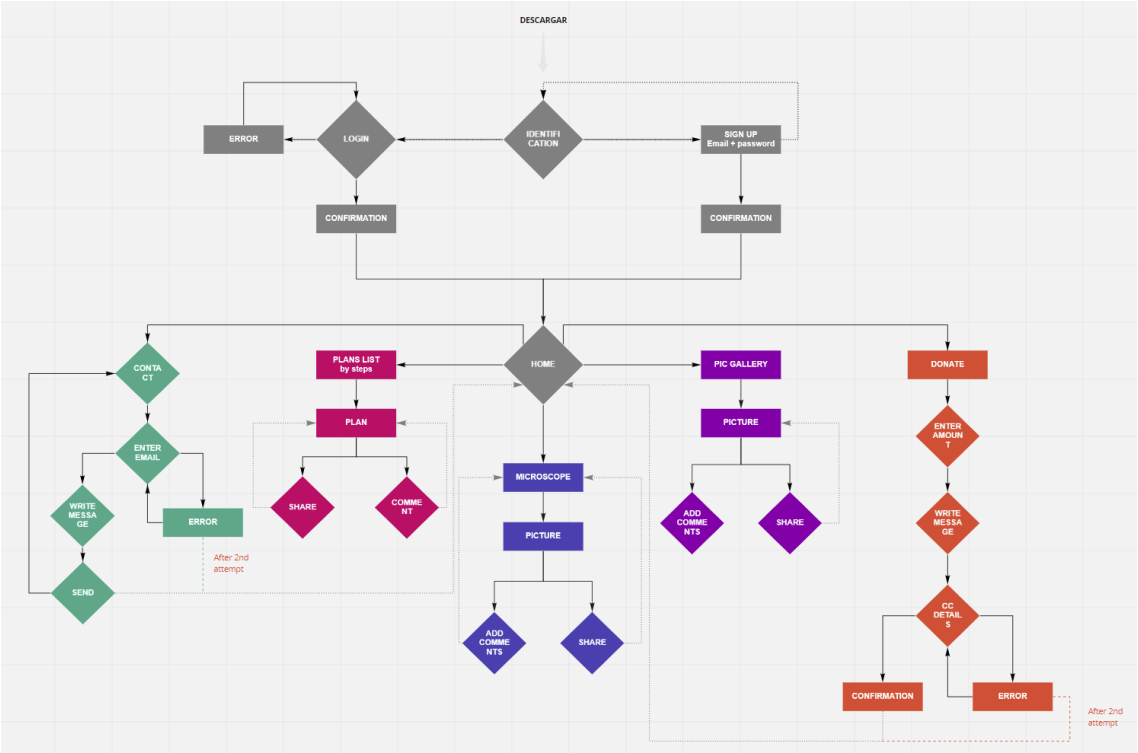
5.1.1. Tasques programades

1. Disseny de l'aplicació
   1.1. Identificació dels requeriments
   1.2. Disseny de l'arquitectura
      1.2.1. Identificació de les funcions necessàries
      1.2.2. Definició de pantalles
      1.2.3. Disseny de la navegació
2. Càmera
   2.1. Configuració de la càmera
      2.1.1. Accés a la càmera
      2.1.2. Representació a la pantalla
   2.2. Configuració de les funcions
      2.2.1. Realització de fotografies
      2.2.2. Emmagatzematge d'arxius
      2.2.3. Accés a les fotografies realitzades
   2.3. Recepció d'inputs de l'usuari
      2.3.1. Gestos a la pantalla
3. Galeria
   3.1. Disseny de la galeria
      3.1.1. Identificació dels elements necessaris
      3.1.2. Estructuració dels arxius
   3.2. Desenvolupament
      3.2.1. Desenvolupament de la vista de quadrícula
      3.2.2. Desenvolupament de la vista individual
4. Col·laboració amb AiScope
   4.1. Definició de les eines
   4.2. Desenvolupament
      4.2.1. Visualització d'arxius
      4.2.2. Contacte
      4.2.3. Donacions

5.2. Disseny de l'aplicació

Partint dels requeriments identificats per l'aplicació, el primer que s'ha de dissenyar és l'arquitectura de l'aplicació. Seguint les recomanacions per a desenvolupar aplicacions amb múltiples pantalles, s'enfoca en una estructura basada en una activitat principal que gestiona el fil principal d'execució, acompanyada de fragments per a gestionar cada una de les funcions de l'aplicació

En el disseny ha tingut molta influència un mock-up de l'aplicació que prepara AiScope com a referència del producte final que imaginaven. Aquest mock-up és senzillament una maqueta gràfica de com hauria de ser l'aplicació, no hi ha programació involucrada de manera que no s'ha pogut aprofitar excepte per a estructurar l'aplicació.



Il·lustració 20. Estructura de l'aplicació proporcionada per AiScopre

Aquesta estructura que proporciona AiScope conté també l'estructura per a registrar-se i iniciar sessió com a usuari de l'aplicació, de manera que aquesta primera part de l'estructura s'ignora, almenys per a aquesta versió.

Per l'estructura, es veu que AiScope proposa una estructura amb 5 seccions diferenciades, que encaixen bastant amb els requeriments plantejats:

- Càmera/microscopi
- Galeria d'imatges
- Visualització de plànols
- Contacte
- Donacions

Per a poder mantenir l'estructura i la navegació plantejada a l'esquema, és necessària la utilització d'un component de navegació per al desplaçament entre seccions principals, on es dirigeixi a l'usuari a la pantalla "inicial" de cada una de les branques.

Des de cada una d'aquestes pantalles inicials és necessari desenvolupar accions de navegació entre les pantalles, ja que no es pot utilitzar el mateix component per a desplaçar-se per les seccions: la navegació ha de ser explícita.

Finalment, queda definir les pantalles que es requereixen per a cada una de les seccions:

- Pantalles globals
    - Inici
- Càmera
    - Control càmera, pantalla principal
    - Fotografia individual
- Galeria d'imatges
    - Galeria, pantalla principal
    - Fotografia individual
- Visualització de plànols
    - Selecció d'arxius, pantalla principal
    - Visualització d'arxiu individual
- Contacte
    - Pantalla principal
- Donacions
    - Pantalla principal

Com que les funcions que es realitzen a la visualització de les fotografies individuals tant de la galeria com la càmera són les mateixes, aquesta pantalla pot ser la mateixa. Això implica un total de 7 pantalles.

5.3. Navegació

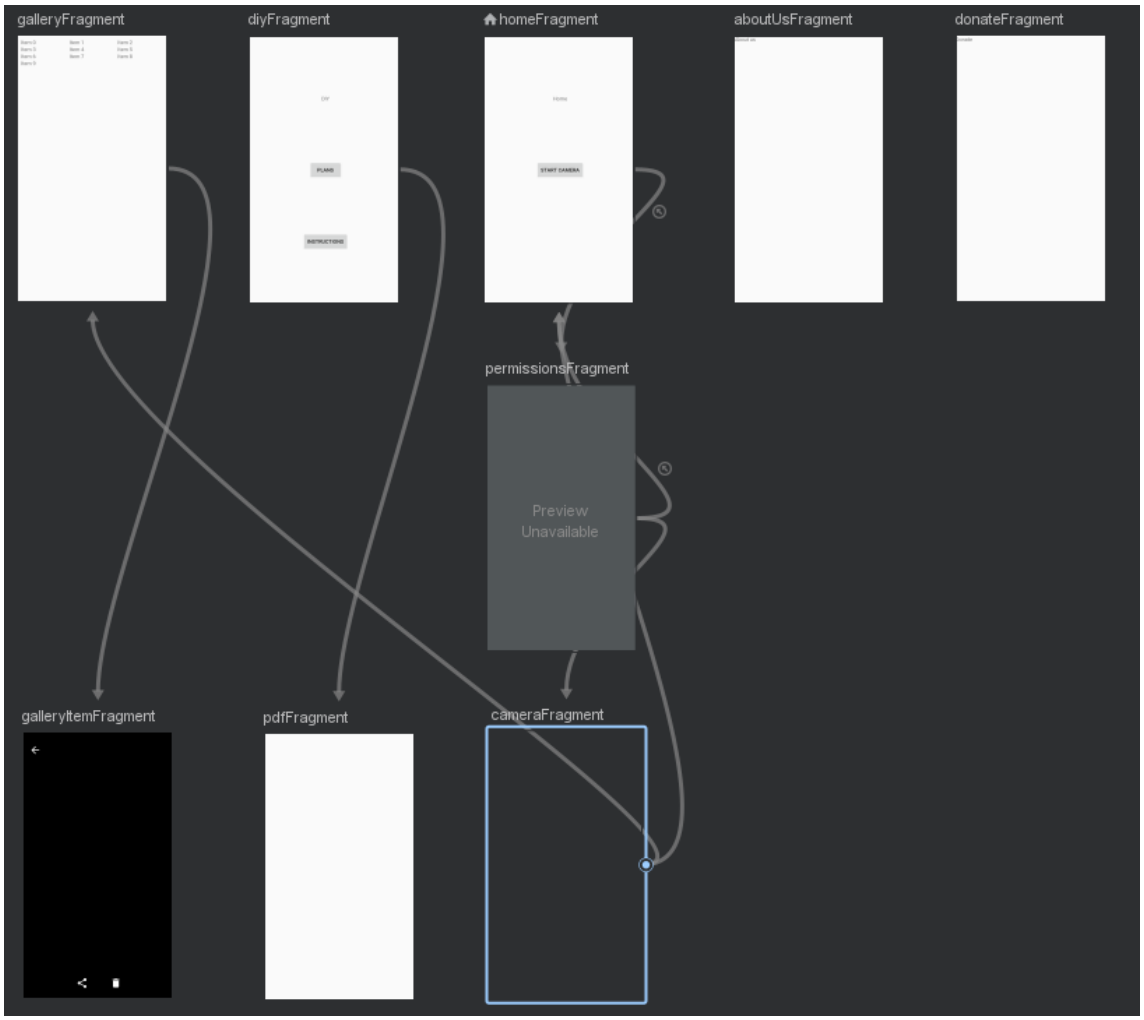
Per a coordinar aquesta navegació, s'ha utilitzat el component *Android Jetpack's Navigation*. La característica principal d'aquest component és que consisteix de 3 elements diferents:

- Esquema de navegació (*navigation graph*): Arxiu en format XML on es detalla tota la informació relacionada amb la navegació. Això inclou totes les seccions amb contingut de l'aplicació, anomenades destinacions, així com tots els possibles recorreguts que es poden prendre entre seccions.
- *NavHost*: Contenidor buit on es visualitzarà les diferents destinacions de l'esquema de navegació.
- *NavController*: Objecte que dirigeix la navegació que es produeix dins el *NavHost*, és qui gestiona els canvis de pantalla i de contingut a mesura que els usuaris es desplacen per l'aplicació.

Així doncs, realment l'aplicació només disposa d'una sola pantalla de visualització amb un component dinàmic que varia segons l'input de l'usuari.

Addicionalment, i per a facilitar la navegació dels usuaris, s'hi ha afegit una barra de navegació a la part inferior de la pantalla que permet el desplaçament entre les 5 seccions principals de l'aplicació.

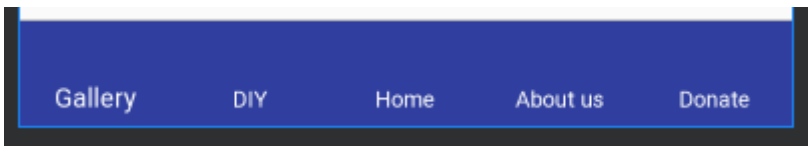
A l'esquema següent es pot apreciar les diferents pantalles de les que disposa l'aplicació. L'esquema està distribuït en 5 columnes diferenciades, on cada una representa les diferents seccions de l'aplicació.



Il·lustració 21. Representació gràfica de la navegació de l'aplicació

Aquestes destinacions estan connectades entre elles mitjançant accions de navegació, representades amb fletxes a l'esquema. Aquestes accions són les que s'utilitzen al codi per a desplaçar-se entre les diferents pantalles amb una excepció: les destinacions d'entrada de cada una de les seccions estan connectades a través d'un altre component, una barra de navegació.

Les pantalles inicials, però, no estan connectades entre elles, no s'hi ha configurat cap acció. Això és perquè la navegació entre seccions està controlada per un altre element, una barra de navegació inferior.



Il·lustració 22. Barra de navegació

Aquesta barra de navegació és força habitual de veure en altres aplicacions i és un component que es situa a la part inferior de la pantalla per a desplaçar-se entre els nivells superiors de l'aplicació. Un dels requisits per a utilitzar-la, és declarar quins són els fragments que formaran part d'aquesta:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/galleryFragment"
        android:title="Gallery"/>
    <item
        android:id="@+id/diyFragment"
        android:title="DIY"/>
    <item
        android:id="@+id/homeFragment"
        android:title="Home"/>
    <item
        android:id="@+id/aboutUsFragment"
        android:title="About us"/>
    <item
        android:id="@+id/donateFragment"
        android:title="Donate"/>
</menu>
```

Il·lustració 23. Declaració dels elements de la barra de navegació

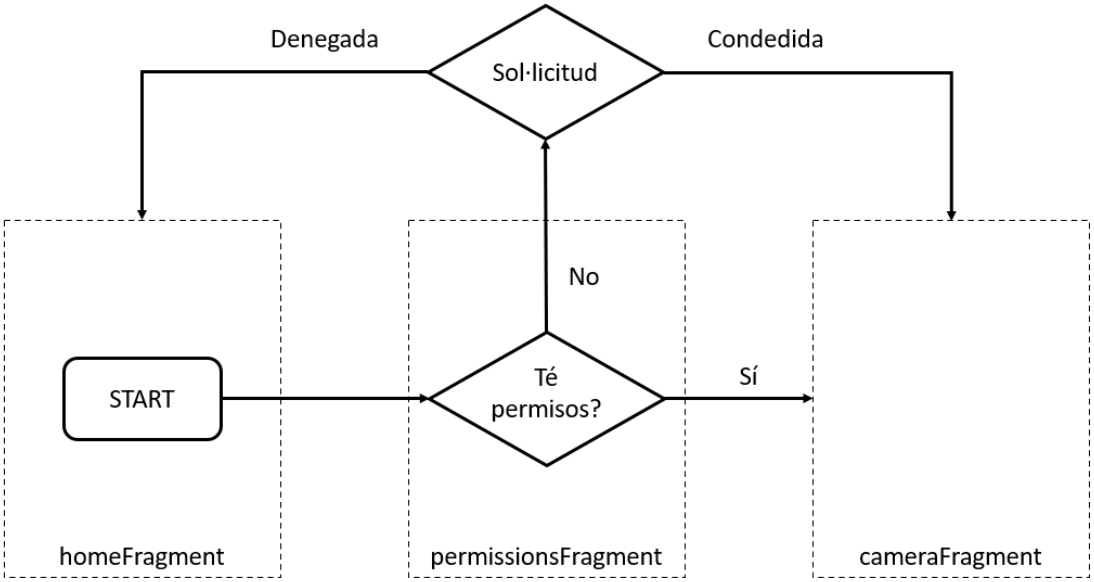
5.4. Càmera

La secció de la càmera és la principal de l'aplicació, ja que l'objectiu final és el d'utilitzar la càmera per a fer fotografies mitjançant el microscopi. Com a tal, s'aprofita la pantalla inicial de l'aplicació per a que funcioni a la vegada com a punt d'accés a la càmera.

5.4.1. Permisos

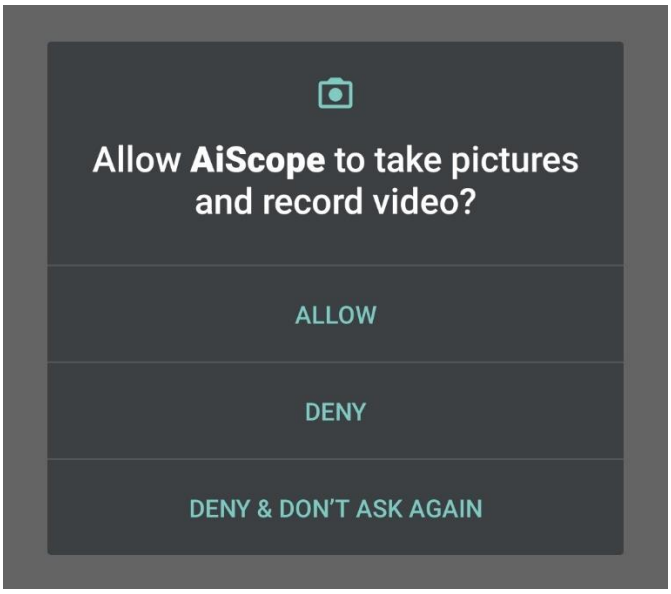
La qüestió amb la càmera és que es tracta d'un recurs que es considera "perillós" en el sentit que per a poder-la utilitzar, l'usuari ha de donar permís explícit. Això implica dues coses: la primera és que s'ha de declarar al manifest de l'aplicació que l'aplicació necessita permisos per a utilitzar la càmera, i la segona és que abans d'iniciar-la s'ha de fer la comprovació que, en efecte, es disposa dels permisos.

Per tal de fer aquesta validació, s'implementa un pas intermedi entre la pantalla d'inici i la càmera, un fragment amb la funció de consultar si l'aplicació disposa dels permisos necessaris.



Il·lustració 24. Flux de decisió previ a l'ús de la càmera

Si l'usuari ja ha concedit explícitament el permís d'accés a la càmera, es redirigeix automàticament fragment de la càmera. Si l'aplicació no disposa del permís, s'envia una petició al sistema del telèfon. Com que el sistema detecta que es tracta d'un recurs perillós, aquest genera una pantalla preguntant a l'usuari si el concedeix. Depenent de la resposta, l'usuari continua cap al fragment de la càmera o se'l retorna a la pantalla d'inici.



Il·lustració 25. Exemple de sol·licitud del permís d'ús de la càmera

Una altra funció que també pot complir aquest fragment, és la comprovació de si es detecta algun microscopi connectat al dispositiu, d'aquesta manera s'evitaria errors a l'intentar establir una connexió amb els listeners d'inputs. Aquesta funcionalitat no s'ha pogut implementar per la impossibilitat d'aconseguir físicament un model del microscopi pel confinament del coronavirus, així que es deixa plantejada com a futur pas.

Addicionalment s'estableix aquest fragment com a destí de l'aplicació en cas de reiniciar-se l'aplicació des de la càmera. Això vol dir que en cas de minimitzar l'aplicació des de la secció de la càmera, al reprendre'n l'ús es dirigeix l'usuari a la comprovació de permisos.

S'ha configurat així perquè els permisos es poden denegar des de la configuració del dispositiu, i si s'intentés utilitzar la càmera sense els permisos necessaris l'aplicació patiria un error crític que la faria tancar. Per evitar-ho, a cada reinici de la càmera es torna a fer el procés de comprovació de permisos.

5.4.2. Configuració de la càmera

Per a configurar la càmera amb la llibreria AndroidX, el fragment realitza dues funcions cada vegada que se'n genera una vista:

- Recuperar i representar a la pantalla la vista prèvia de la càmera
- Configurar el controlador per a realitzar fotografies

5.4.2.1. Vista prèvia

La vista prèvia de la càmera és la imatge que està rebent la càmera a temps real i el que s'ha de representar a la pantalla. La funció per a obtenir aquesta vista prèvia es configura amb 3 arguments:

- Direcció de la càmera
- Proporció de la pantalla
- Rotació de la pantalla

Una vegada configurada la vista prèvia, s'assigna al component designat per a la seva representació. En aquest cas el component és un ViewFinder, un component dissenyat per a suportar elements interactius.

```
// Set up the view finder use case to display camera preview
val viewFinderConfig : PreviewConfig  = PreviewConfig.Builder().apply { this: PreviewConfig.Builder
    setLensFacing(lensFacing)
    // We request aspect ratio but no resolution to optimize the use cases
    setTargetAspectRatio(screenAspectRatio)
    // Set initial target rotation
    setTargetRotation(viewFinder.display.rotation)
}.build()

// Use the auto-fit preview builder to automatically handle size changes
preview = AutoFitPreviewBuilder.build(viewFinderConfig, viewFinder)
```

Il·lustració 26. Configuració de la vista prèvia de la càmera

5.4.2.2. Controlador de fotografies

D'altra banda, el controlador per a realitzar les fotografies s'informa amb els següents arguments:

- Direcció de la càmera
- Proporció de la pantalla
- Rotació de la pantalla
- Tipus de captura

3 dels arguments coincideixen amb els de la vista prèvia, d'aquesta manera coincideixen també les configuracions d'entrada i de sortida de les imatges. Per a la captura, però, també se li assigna un tipus de captura, MAX_QUALITY, perquè prioritzi la qualitat de la imatge abans de la velocitat de captura.

```
// Set up the capture use case to allow users to take photos
val imageCaptureConfig : ImageCaptureConfig =
    ImageCaptureConfig.Builder().apply { this: ImageCaptureConfig.Builder
        setLensFacing(lensFacing)
        setCaptureMode(CaptureMode.MIN_LATENCY)
        // We request aspect ratio but no resolution to match preview config but letting
        // AiScope optimize for whatever specific resolution best fits requested capture mode
        setTargetAspectRatio(screenAspectRatio)
        // Set initial target rotation
        setTargetRotation(viewFinder.display.rotation)
    }.build()

imageCapture = ImageCapture(imageCaptureConfig)
```

Il·lustració 27. Configuració de la captura d'imatges

5.4.3. Configuració de les funcions

Les funcions que ha de disposar el fragment són 3:

- Realitzar fotografies
- Emmagatzemar els arxius
- Visualitzar les fotografies

5.4.3.1. Realitzar fotografies

Per a realitzar les fotografies, s'assigna el controlador de fotografies al botó central de la pantalla. Per a fer-ho es configura un listener al botó que realitza 3 funcions:

- Crea un directori per a contenir l'arxiu de la fotografia
- Executa el controlador de fotografies
- Executa una animació per a indicar que s'ha realitzat la fotografia

El controlador de fotografies conté una funció específica per a fer fotografies, i per a executar-la s'hi inclou 3 arguments: el directori creat prèviament, una referència al fil d'execució principal del fragment, i un listener per a la gestió de l'emmagatzematge de la imatge.

```
// Listener for button used to capture photo
view.findViewById<ImageButton>(R.id.cameraCaptureButton).setOnClickListener { it: View!
    // Get a stable reference of the modifiable image capture use case
    imageCapture?.let { imageCapture ->

        // Create output file to hold the image
        val photoFile : File =
            createFile(
                outputDirectory,
                FILENAME,
                PHOTO_EXTENSION
            )

        // Setup image capture listener which is triggered after photo has been taken
        imageCapture.takePicture(photoFile, mainExecutor, imageSavedListener)

        // Display flash animation to indicate that photo was captured
        container.postDelayed({
            container.foreground = ColorDrawable(Color.WHITE)
            container.postDelayed(
                { container.foreground = null }, ANIMATION_FAST_MILLIS
            )
        }, ANIMATION_SLOW_MILLIS)
    }
}
```

Il·lustració 28. Listener del botó per a realitzar fotografies

5.4.3.2. Emmagatzemar els arxius

Una vegada realitzada la fotografia, aquesta es guarda automàticament al directori definit, que en aquest cas és la carpeta d'arxius multimèdia propis de l'aplicació. Pel fet de guardar-se en una carpeta "privada", és a dir, que el sistema no hi té necessàriament accés directe, aquest arxiu queda virtualment invisible, els sistema no sap que existeix.

Això és un problema perquè no es tindria accés a la imatge des d'altres aplicacions, impedint així poder-la compartir o enviar, un dels requeriments de l'aplicació. Per a solucionar aquesta situació Android disposa d'una llibreria anomenada MediaScanner que s'encarrega d'afegir els arxius a la llista de distribució del sistema segons el tipus d'arxiu.

Així doncs, una vegada guardada la foto el fragment recupera el tipus d'arxiu, que per l'aplicació és sempre JPG, i amb MediaScanner envia el directori i el tipus d'arxiu de la imatge. El listener també conté una detecció d'errors on, en cas de no haver pogut guardar l'arxiu, es retorna l'error que s'ha produït per a analitzar-lo amb un debugger.

```
private val imageSavedListener = object : ImageCapture.OnImageSavedListener {
    override fun onError(
        error: ImageCapture.ImageCaptureError, message: String, exc: Throwable?
    ) {
        Log.e(TAG,  msg: "Photo capture failed: $message")
        exc?.printStackTrace()
    }

    override fun onImageSaved(photoFile: File) {
        Log.d(TAG,  msg: "Photo capture succeeded: ${photoFile.absolutePath}")
        // Update the gallery thumbnail with latest picture taken
        setGalleryThumbnail(photoFile)

        // Other apps will not be able to access our images unless we
        // scan them using [MediaScannerConnection]
        val mimeType : String?  = MimeTypeMap.getSingleton()
            .getMimeTypeFromExtension(photoFile.extension)
        MediaScannerConnection.scanFile(
            context, arrayOf(photoFile.absolutePath), arrayOf(mimeType),  callback: null
        )
    }
}
```

Il·lustració 29. Listener per a les imatges guardades

5.4.3.3. Visualitzar les fotografies

Per a visualitzar les fotografies realitzades s'afegeix un botó a la pantalla. Aquest botó té dues funcionalitats associades:

- Navegació al fragment de fotografia
- Representació de l'última fotografia feta

La funcionalitat principal és la de navegar al fragment per a la visualització de fotografies individual, i concretament la primera imatge que es mostrarà és l'última realitzada.

La segona funcionalitat és purament visual, i és que el botó s'emplena de forma dinàmica amb una vista prèvia de l'última imatge guardada, funció que s'actualitza cada vegada que s'obre la càmera o es fa una nova fotografia.

```
private fun setGalleryThumbnail(file: File) {
    val thumbnail : ImageButton! = container.findViewById<ImageButton>(R.id.cameraPhotoButton)
    thumbnail.post {
        thumbnail.setPadding(4dp.toInt())
        Glide.with(thumbnail)
            .load(file)
            .apply(RequestOptions.circleCropTransform())
            .into(thumbnail)
    }
}
```

Il·lustració 30. Funció per a actualitzar la imatge del botó d'accés a les fotografies

5.4.4. Recepció d'inputs de l'usuari

Per la recepció dels inputs de l'usuari s'assigna un listener de gestos de la pantalla al component que s'utilitza per a representar la vista prèvia de la càmera.

Aquest listener s'activa quan detecta contacte a la pantalla i a mesura que el dit es desplaça retorna els valors de la velocitat de cada una de les coordenades X i Y en píxels per segon.

Són aquestes velocitats que posteriorment s'hauran d'enviar al microscopi, però de moment simplement es retorna un missatge visible des de l'entorn de desenvolupament.

```
viewFinder.setOnTouchListener { _, event ->
    when (event.actionMasked) {
        MotionEvent.ACTION_DOWN -> {
            // Reset the velocity tracker back to its initial state.
            mVelocityTracker?.clear()
            // Retrieve a new VelocityTracker object to watch the velocity of a motion.
            mVelocityTracker = mVelocityTracker ?: VelocityTracker.obtain()
            // Add a user's movement to the tracker.
            mVelocityTracker?.addMovement(event)
        }
        MotionEvent.ACTION_MOVE -> {
            mVelocityTracker?.apply { this: VelocityTracker
                val pointerId: Int = event.getPointerId(event.actionIndex)
                addMovement(event)
                computeCurrentVelocity( units: 1000)

                // Log velocity of pixels per second
                Log.d( tag: "",  msg: "X velocity: ${getXVelocity(pointerId)}")
                Log.d( tag: "",  msg: "Y velocity: ${getYVelocity(pointerId)}")
            }
        }
        MotionEvent.ACTION_UP, MotionEvent.ACTION_CANCEL -> {
            // Return a VelocityTracker object back to be re-used by others.
            mVelocityTracker?.recycle()
            mVelocityTracker = null
        }
    }
    true ^setOnTouchListener
}
```

Il·lustració 31. Listener de gestos a la càmera

5.5. Galeria d'imatges

La secció de la galeria serveix per a visualitzar i gestionar les fotografies fetes amb la càmera de l'aplicació. Per a complir l'objectiu el directori de les imatges que s'utilitza és el directori d'arxius multimèdia de l'aplicació.

S'utilitza aquest directori privat per dos motius:

- No s'ha de filtrar les imatges per a mostrar únicament les de l'aplicació
- No es requereix de permisos implícits de l'usuari

L'altra alternativa de directori era la carpeta designada pel sistema pels arxius multimèdia, que per una banda no es requeriria l'és de MediaScanner al guardar els arxius, però per l'altra requereix de la petició de permisos a l'usuari i el disseny d'una manera de filtrar les imatges de la resta d'aplicacions, però s'ha acabat desestimant.

5.5.1. Disseny de la galeria

La galeria es pot dividir en dues parts segons les funcions de cada una:

- Galeria d'imatges
- Visualització individual

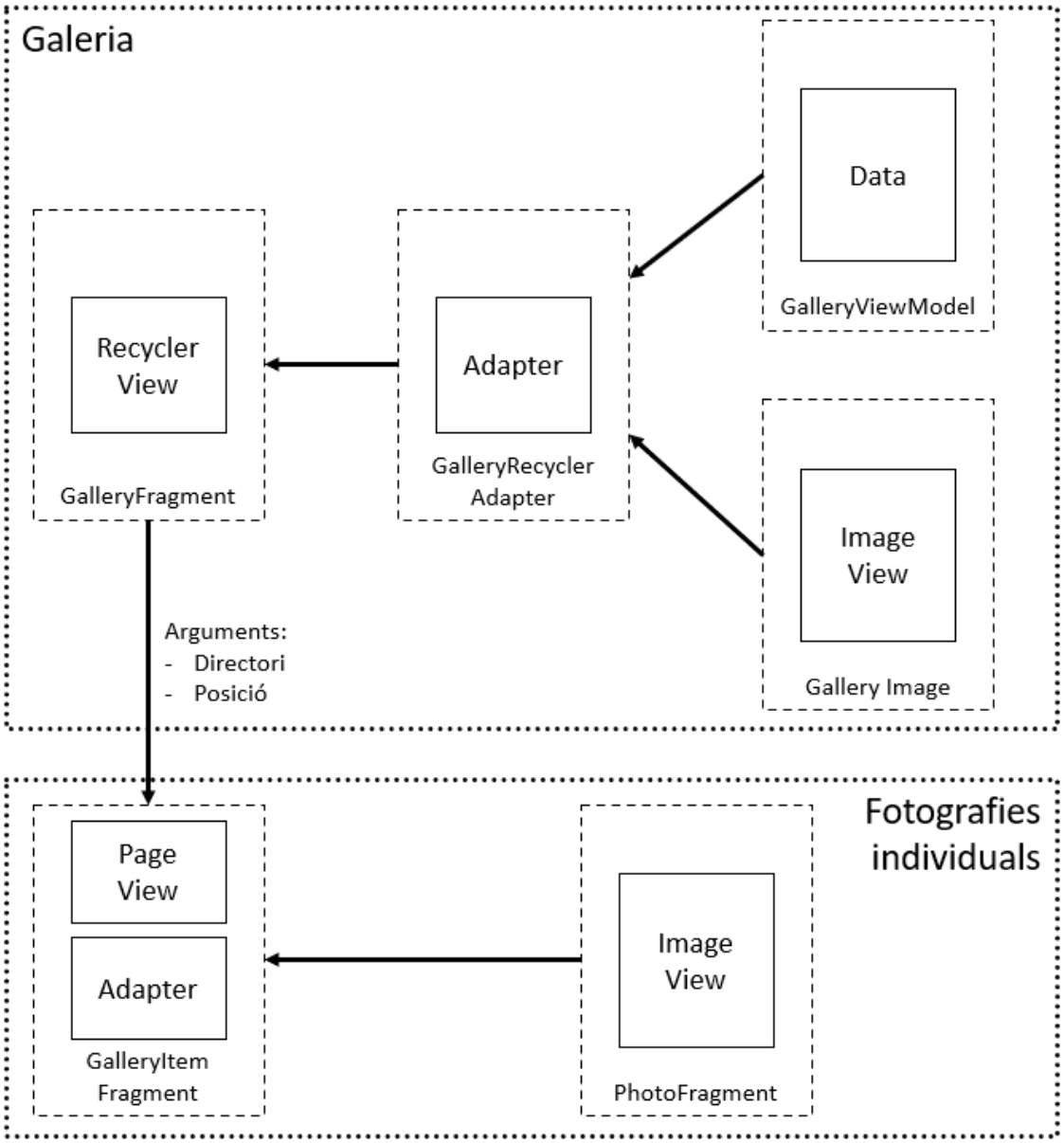
Són parts diferents de la secció perquè cada una d'elles utilitza un component diferent que funcionen independentment l'un de l'altre, l'únic que tenen en comú és que utilitzen la mateixa font de dades: les imatges de l'aplicació.

Per a la galeria s'utilitza un RecyclerView, un component que a partir d'una llista d'elements genera una subcomponent per a cada un d'ells. Per a les fotografies individuals s'utilitza un PageViewer, un component que té un funcionament similar al RecyclerView ì genera una pantalla completa per a cada un dels elements de la llista.

Com que es tracta d'elements diferents, per tal de navegar entre ells s'ha d'afegir dues dades juntament amb la intenció de navegar:

- Directori de les imatges
- Posició de la imatge

A partir del directori, la secció de la fotografia genera la llista d'imatges, mentre que la posició indica quin dels elements es pretén visualitzar.



Il·lustració 32. Flux del funcionament de la Galeria

De l'estructura de la secció de la galeria s'ha de tenir en compte que també existeix la possibilitat d'accedir al fragment de la visualització d'imatges individuals des del fragment de la càmera, així que s'ha de dissenyar la configuració del fragment tenint en compte aquesta ruta d'accés.

5.5.2. Desenvolupament

5.5.2.1. Desenvolupament de la vista de quadrícula

La vista de quadrícula o de galeria s'ha desenvolupat utilitzant un RecyclerView. Els RecyclerView són components molt útils per a representar en pantalla una llista d'elements que, realment, poden ser pràcticament qualsevol cosa.

En un RecyclerView intervenen diversos components que interaccionen entre ells per a mostrar les dades. El component que ho engloba tot és el mateix RecyclerView, un contenidor que s'afegeix a al layout de la interfície gràfica.

Posteriorment, aquest contenidor s'omple amb de visualitzacions mitjançant el gestor del layout, que per a l'aplicació és del subtipus GridLayoutManager, equivalent a la llista de quadrícula. Aquesta quadrícula es defineix amb 3 columnes, és a dir que a cada línia de la galeria hi haurà un màxim de 3 imatges.

```
val view : View! = inflater.inflate(R.layout.fragment_gallery, container, attachToRoot: false)

recyclerView = view.findViewById(R.id.galleryRecyclerView)
layoutManager = GridLayoutManager(activity, spanCount: 3)
recyclerView.layoutManager = layoutManager
```

Il·lustració 33. Declaració del RecyclerView i del gestor del layout

Cada una d'aquestes imatges que es representen a la pantalla estan controlades per un component anomenat ViewHolder. Cada ViewHolder és l'encarregat de gestionar una sola imatge mitjançant el layout que s'ha desenvolupat per a representar-les.

```
override fun onCreateViewHolder(
    parent: ViewGroup,
    viewType: Int
): GalleryRecyclerAdapter.ViewHolder {
    val inflater : LayoutInflater! = LayoutInflater.from(parent.context)
    val view : View! = inflater.inflate(R.layout.gallery_image_item, parent, attachToRoot: false)
    return ViewHolder(view)
}

inner class ViewHolder(itemView: View) :
    RecyclerView.ViewHolder(itemView) {
    val imgView: ImageView = itemView.findViewById<ImageView>(R.id.galleryImage)
}
```

Il·lustració 34. Declaració dels ViewHolder i assignació al layout corresponent

Al seu torn, aquests objectes ViewHolder estan gestionats per un adaptador, que s'encarrega d'associar cada un dels ViewHolder a les imatges corresponents. Això ho fa associant cada element a una posició numèrica i executant el mètode onBindViewHolder() s'utilitza aquesta posició per a determinar els continguts de l'objecte basant-se en aquesta posició.

```
override fun onBindViewHolder(holder: GalleryRecyclerAdapter.ViewHolder, position: Int) {
    val galleryImage : File  = galleryImages[position]
    Glide.with(context)
        .load(galleryImage)
        .into(holder.imgView)
    holder.imgView.setOnClickListener{ it: View!
        itemListener.onImageItemClick(galleryImage, position)
    }
}
```

Il·lustració 35. Funció onBindViewHolder() per a assignar els continguts del ViewHolder

Finalment queda recuperar i retornar en una llista les imatges que disponibles al directori de l'aplicació, funció que es realitza utilitzant una classe de suport, GalleryViewModel. Aquesta classe s'encarrega de recuperar les imatges asíncronament, d'aquesta manera s'agilitza el procés de càrrega perquè la recuperació de les imatges es realitza en un segon pla.

```
private fun listAllImages() {
    outputDirectory = MainActivity.getOutputDirectory(app)
    imageData.postValue(outputDirectory.listFiles()?.toList()?.sortedDescending())
}

fun refreshData() {
    CoroutineScope(Dispatchers.IO).launch { this: CoroutineScope
        listAllImages()
    }
}
```

Il·lustració 36. Funció per a generar la llista de les imatges disponibles

Aquestes imatges queden guardades en un atribut, que s'utilitza des del fragment principal de la galeria com a argument per a declarar l'adaptador.

```
viewModel = ViewModelProvider(requireActivity()).get(GalleryViewModel::class.java)
viewModel.imageData.observe(viewLifecycleOwner, Observer { it: List<File>!
    val adapter = GalleryRecyclerAdapter(requireContext(), it,  itemListener: this)
    recyclerView.adapter = adapter
})
```

Il·lustració 37. Recuperació de les imatges i declaració de l'adaptador

Finalment queda assignar a cada un dels ViewHolder un listener per a navegar a la visualització individual de la imatge al clicar una imatge concreta. Per a carregar la imatge correcta, és necessari enviar el directori on es troba la imatge i la seva posició.

```
override fun onImageItemClick(image: File, position: Int) {
    Navigation.findNavController(requireActivity(), R.id.nav_host_fragment).navigate(
        GalleryFragmentDirections
            .actionGalleryFragmentToGalleryItemFragment(
                viewModel.outputDirectory.absolutePath,
                position
            )
    )
}
```

Il·lustració 38. Listener assignat als ViewHolder

5.5.2.2. Desenvolupament de la visualització individual

La vista individual s'ha desenvolupat un PageViewer, un component molt útil per a representar fragments sencers i desplaçar-se entre ells fent lliscar el dit per la pantalla.

El seu funcionament és molt similar al d'un RecyclerView, de fet, es tracta d'un component que hereta moltes de les seves característiques. La diferència principal és que mentre els RecyclerView utilitzen objectes ViewHolder per a representar els elements, els PageViewer mostren fragments sencers.

Pel que fa a l'estructura del seu funcionament, el PageViewer és un contenidor que s'afegeix a la interfície gràfica al que se li assigna un adaptador a partir d'una llista d'elements que en el cas de l'aplicació són imatges.

```
/** Adapter class used to present a fragment containing one photo or video as a page */
inner class MediaPagerAdapter(fm: FragmentManager) : FragmentStatePagerAdapter(fm) {
    override fun getCount(): Int = mediaList.size
    override fun getItem(position: Int): Fragment = PhotoFragment.create(mediaList[position])
}
```

Il·lustració 39. Declaració de l'adaptador del PageViewer

L'adaptador, per a cada element de la llista, genera un nou fragment de fotografia enviant com a argument l'arxiu a representar. El fragment, a la seva vegada, utilitza aquest argument per a carregar la imatge com a un ImageView i retornar-la.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    val args : Bundle  = arguments ?: return
    val resource : {Comparable<{File & Int}> & Serializable}  = args.getString(FILE_NAME_KEY)?.let { it: String
        File(it)
    } ?: R.drawable.ic_photo
    Glide.with(view).load(resource).into(view as ImageView)
}

companion object {
    private const val FILE_NAME_KEY = "file_name"

    fun create(image: File) = PhotoFragment().apply { this: PhotoFragment
        arguments = Bundle().apply { this: Bundle
            putString(FILE_NAME_KEY, image.absolutePath)
        }
    }
}
```

Il·lustració 40. Càrrega de la imatge a partir d'un argument

Una vegada carregada la fotografia a la pantalla es configura les funcions dels botons de gestió de la fotografia.

5.5.2.2.1. Compartir imatge

Per a poder compartir la imatge s'ha de fer una gestió prèvia, i és que s'ha de declarar al manifest de l'aplicació que aquesta es tracta d'un proveïdor d'arxius per a la resta d'aplicacions del sistema.

```
<!-- FileProvider used to share photos with other apps -->
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="com.kotlin.aiscope.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/filepaths" />
</provider>
```

Il·lustració 41. Definició del proveïdor d'arxius al manifest

Amb el proveïdor declarat ja es pot configurar la funció per a compartir. Per a compartir un arxiu, el que s'ha de fer és enviar una petició al sistema amb una intenció d'enviar l'arxiu, punt a partir és el sistema el que s'encarrega de detectar les aplicacions disponibles.

Per a configurar aquesta intenció s'utilitza una sèrie d'arguments:

- Tipus d'arxiu
- Directori de l'arxiu
- Tipus d'acció
- Flags extra

El tipus d'arxiu es recupera directament de l'arxiu visualitzat, encara que en principi hauria de ser sempre JPG, i pel directori s'utilitza una funcionalitat del proveïdor declarat al manifest. El tipus d'acció és ACTION_SEND i se li proporciona un flag extra que permeti a la intenció desxifrar el directori de l'arxiu enviat.

Una vegada definida la intenció, es llença la petició al sistema, que s'encarrega de la resta.

```
// Handle share button press
view.findViewById<ImageButton>(R.id.galleryItemShareButton).setOnClickListener { it: View!
    // Make sure that we have a file to share
    mediaList.getOrNull(mediaViewPager.currentItem)?.let { mediaFile ->

        // Create a sharing intent
        val intent : Intent = Intent().apply { this: Intent
            // Infer media type from file extension
            val mediaType : String? = MimeTypeMap.getSingleton()
                .getMimeTypeFromExtension(mediaFile.extension)

            // Get URI from our FileProvider implementation
            val uri : Uri! = FileProvider.getUriForFile(
                view.context, authority: BuildConfig.APPLICATION_ID + ".fileprovider", mediaFile
            )

            // Set the appropriate intent extra, type, action and flags
            putExtra(Intent.EXTRA_STREAM, uri)
            type = mediaType
            action = Intent.ACTION_SEND
            flags = Intent.FLAG_GRANT_READ_URI_PERMISSION
        }

        // Launch the intent letting the user choose which app to share with
        startActivity(Intent.createChooser(intent, "Share using"))
    }
```

Il·lustració 42. Configuració del botó per a compartir la imatge

5.5.2.2.2. Eliminar la imatge

Per a eliminar la imatge es configura primer un missatge de confirmació per a l'usuari per evitar esborrar-ne alguna per error. Aquest missatge de confirmació disposa de dues respostes possibles.

```
// Handle delete button press
view.findViewById<ImageButton>(R.id.galleryItemDeleteButton).setOnClickListener { it: View!
    AlertDialog.Builder(view.context, android.R.style.Theme_Material_Dialog)
        .setTitle("Confirm")
        .setMessage("Delete current photo?")
        .setIcon(android.R.drawable.ic_dialog_alert)
        .setPositiveButton(android.R.string.yes) {...}

        .setNegativeButton(android.R.string.no,  listener: null)
        .create().showImmersive()
}
```

Il·lustració 43. Declaració del listener del botó d'elimnar fotografies

Si la resposta és negativa, es notifica a l'usuari que l'eliminació no s'ha dut a terme i es deixa la imatge tal i com estava inicialment. En canvi, si la resposta és positiva s'executa una sèrie de processos.

- Eliminació de l'arxiu
- Notificació al sistema
- Notificació a l'adaptador
- Si no queden imatges, retorn a la pantalla anterior

De forma similar que amb la creació d'arxius, quan aquests s'eliminen també s'ha de notificar el sistema i la resta d'aplicacions que aquest ja no existeix per a que tinguin constància d'eliminar qualsevol relació amb aquest que pugui provocar un error d'execució.

També es notifica a l'adaptador del ViewPager per a que actualitzi la imatge que s'està visualitzant, ja que aquestes es guarden en el cache. Això vol dir que si no rep la notificació, la imatge eliminada es podria seguir visualitzant mentre no hi hagués un canvi de fragment.

Finalment es realitza la comprovació que quedin fotografies per a mostrar. Si eliminant la imatge el directori queda buit, es redirigeix l'usuari a la pantalla que estava visualitzant abans de la fotografia.

```
.setPositiveButton(android.R.string.yes) { _, _ ->
    mediaList.getOrNull(mediaViewPager.currentItem)?.let { mediaFile ->

        // Delete current photo
        mediaFile.delete()

        // Send relevant broadcast to notify other apps of deletion
        MediaScannerConnection.scanFile(
            view.context, arrayOf(mediaFile.absolutePath), mimeTypes: null, callback: null
        )

        // Notify our view pager
        mediaList.removeAt(mediaViewPager.currentItem)
        mediaViewPager.adapter?.notifyDataSetChanged()

        // If all photos have been deleted, return to the previous
        if (mediaList.isEmpty()) {
            fragmentManager?.popBackStack()
        }
    }
}
```

Il·lustració 44. Gestió de l'eliminació de les imatges

5.6. Visualització d'arxius

Una de les eines que s'afegeix a l'aplicació per a incentivar la col·laboració amb AiScope és la possibilitat de visualitzar arxius amb instruccions de muntatge del microscopi.

L'objectiu darrera aquesta funció és la de facilitar els usuaris la construcció dels microscopis i, per tant, la realització de fotografies per a compartir amb les xarxes.

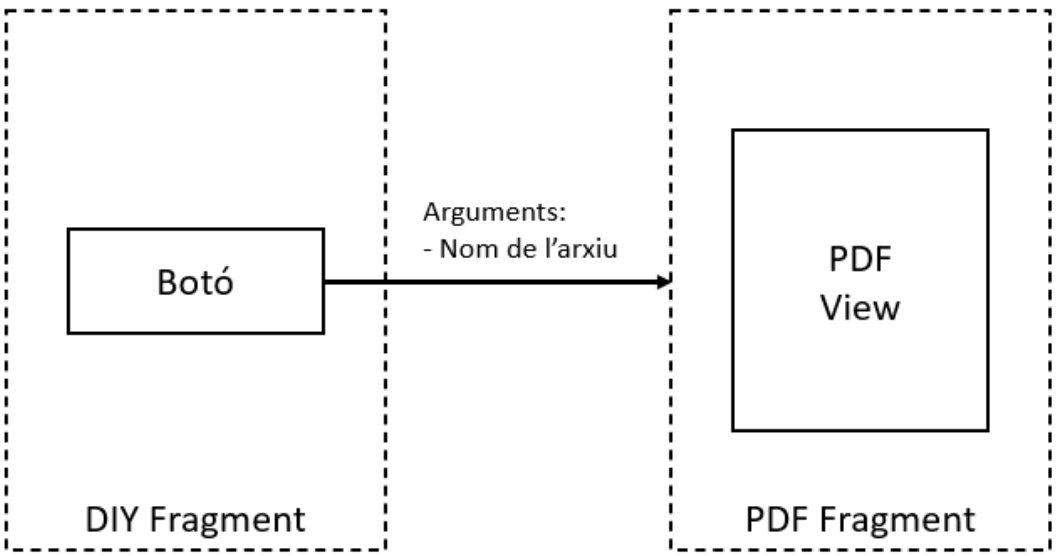
Per aquesta iteració es decideix mostrar dos arxius en PDF, encara que no necessàriament són els arxius que es voldria tenir en una versió final del producte.

5.6.1. Disseny de l'estructura

Aquesta secció es dissenya al voltant d'un document anomenat PdfViewer que serveix per a, precisament, visualitzar PDF dins una aplicació Android. També es té en compte que la quantitat d'arxius a mostrar pot variar, de manera que el fragment de visualització ho ha de tenir en compte.

La secció està formada per dos fragments amb dues funcions diferenciades:

- Selecció d'arxius
- Visualització d'arxius



Il·lustració 45. Estructura de la secció per a visualitzar PDF

Els arxius a representar es troben en un paquet intern de l'aplicació, Assets, paquet que conté arxius utilitzats per l'aplicació que no són les imatges de l'usuari. Per a indicar al fragment de visualització l'arxiu que s'ha de representar, s'afegeix als arguments de la navegació el nom de l'arxiu a representar.

5.6.1.1. Desenvolupament

5.6.1.1.1. Selecció d'arxius

El fragment de la selecció d'arxius està format per un botó per a cada un dels arxius a visualitzar amb la funció associada de navegar al fragment de visualització a partir del nom de l'arxiu associat.

```
val view : View! = inflater.inflate(R.layout.fragment_diy, container, attachToRoot: false)
val plansButton: Button = view.findViewById(R.id.plansButton)
plansButton.setOnClickListener { it: View!
    Navigation.findNavController(requireActivity(), R.id.nav_host_fragment)
        .navigate(
            DiyFragmentDirections
                .actionDiyFragmentToPdfFragment( pdfName: "plans.pdf"))
}
val insButton: Button = view.findViewById(R.id.instructionsButton)
insButton.setOnClickListener { it: View!
    Navigation.findNavController(requireActivity(), R.id.nav_host_fragment)
        .navigate(DiyFragmentDirections
            .actionDiyFragmentToPdfFragment( pdfName: "instructions.pdf"))
}
```

Il·lustració 46. Configuració de la navegació dels botons de selecció

5.6.1.1.2. Visualització d'arxius

Cada vegada que es rep una visualització d'un PDF, es navega al fragment per a la visualització. Aquest fragment està format per un component PdfViewer.

A la creació de la visualització, el fragment recupera el nom de l'arxiu dels arguments de navegació i l'utilitza per a carregar l'arxiu al component PdfViewer utilitzant les funcions incorporades amb l'objectes.

Aquest component carrega totes les pàgines del PDF individualment, de manera que es configura una notificació per a l'usuari si alguna d'aquestes causa un error de càrrega.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    val view : View! = inflater.inflate(R.layout.fragment_pdf, container, attachToRoot: false)

    pdfView = view.findViewById(R.id.pdfView)
    pdfView.fromAsset(
        getPdfNameFromArgs(args)
    )
        .defaultPage( defaultPage: 0)
        .onPageError { page, _ ->
            Toast.makeText(
                context,
                 text: "Error at page: $page",
                Toast.LENGTH_LONG
            )
                .show()
        }
        .load()
    return view
}

companion object {
    fun getPdfNameFromArgs(args: PdfFragmentArgs): String {
        return args.pdfName
    }
}
```

Il·lustració 47. Recuperació i càrrega de l'arxiu PDF a visualitzar

5.7. Contacte

La funció d'aquest fragment és la de redactar i enviar un correu electrònic a la direcció de contacte d'AiScope. Per a fer-ho només es requereix d'un sol fragment que conté dues entrades de text i un botó per a enviar el missatge.

La funció del botó és la de configurar i enviar una intenció al sistema, que és qui s'encarrega de la gestió del correu. En aquesta intenció se li assigna una acció d'enviament, indicant que és un correu electrònic amb el tipus d'informació i de dades que s'està enviant.

També s'afegeix l'assumpte, el missatge i la direcció de correu a través dels arguments extra, dades que s'obtenen a partir dels dos editors de text i d'un valor constant a l'aplicació, corresponent a la direcció de correu electrònic.

```
val view : View! = inflater.inflate(R.layout.fragment_contact, container, attachToRoot: false)

val subjectEditText : EditText! = view.findViewById<EditText>(R.id.subjectEditText)
val messageEditText : EditText! = view.findViewById<EditText>(R.id.messageEditText)
val emailBtn : Button! = view.findViewById<Button>(R.id.emailButton)

emailBtn.setOnClickListener { it: View!
    val subject : String = subjectEditText.text.toString().trim()
    val message : String = messageEditText.text.toString().trim()

    val mIntent : Intent = Intent().apply { this: Intent
        action = Intent.ACTION_SEND
        putExtra(Intent.EXTRA_EMAIL, arrayOf(EMAIL))
        putExtra(Intent.EXTRA_SUBJECT, arrayOf(subject))
        putExtra(Intent.EXTRA_TEXT, arrayOf(message))
        data = Uri.parse( uriString: "mailto:")
        type = "message/rfc822"
    }
    try {
        startActivity(mIntent)
    }
    catch (e: Exception) {
        Toast.makeText(requireContext(), e.message, Toast.LENGTH_LONG).show()
    }
}
```

Il·lustració 48. Configuració i enviament del correu electrònic

5.8. Donacions

Per a les donacions només es requereix un sol fragment, on es configura un RadioGroup per a contenir una llista seleccionable de quantitats de diners amb una breu explicació del que permet aquella donació

Aquest RadioGroup se li assigna un listener per als canvis de l'element seleccionat que actualitza un valor intern corresponent al valor de la donació que es vol fer.

```
val radioGroup : RadioGroup! = view.findViewById<RadioGroup>(R.id.radioGroup)
radioGroup.setOnCheckedChangeListener { _, checkedId ->
    when (checkedId) {
        R.id.twentyRadioButton ->
            amount = 20

        R.id.fiftyRadioButton ->
            amount = 50

        R.id.oneHundredRadioButton ->
            amount = 100

        R.id.twoFiftyRadioButton ->
            amount = 250

        R.id.fiveHundredRadioButton ->
            amount = 500
    }
}
```

Il·lustració 49. Listener per a la selecció de la quantitat de la donació

També s'assigna un listener al botó per a realitzar la donació. Aquest botó genera una URL corresponent a la pàgina de donacions d'AiScope a partir del valor seleccionat, enllaç que s'obre amb el navegador per defecte del sistema.

Per a obrir el navegador, es configura i s'envia una intenció al sistema que notifica que es tracta d'una acció de visualització d'un enllaç. Abans de generar la intenció, però, es fa una comprovació que s'hagi seleccionat un dels elements de la llista. De no ser així es notifica a l'usuari de la necessitat de tenir un dels elements seleccionat.

```
val donateButton : Button!  = view.findViewById<Button>(R.id.donateButton)
donateButton.setOnClickListener { it: View!
    if (amount > 0) {
        val intent : Intent  = Intent().apply { this: Intent
            val uri : Uri!  = Uri.parse( uriString: PAYPAL_URL + amount.toString())
            data = uri
            action = Intent.ACTION_VIEW
        }
        startActivity(intent)
    } else {
        Toast.makeText(
            context,
            text: "Please select a value",
            Toast.LENGTH_LONG
        )
            .show()
    }
}
```

Il·lustració 50. Creació i enviament de la intenció de donació

6. Futurs passos

El resultat final d'aquest projecte ha acabat sent un prototip d'aplicació que implementa la gran majoria dels requeriments de l'aplicació final d'AiScope, però que en qualsevol cas es tracta d'un producte acabat.

Hi ha una sèrie d'elements que s'ha de treballar i/o desenvolupar per tal de donar per acabada l'aplicació:

- Desenvolupament de la interfície gràfica
- Integració amb el microscopi
- Cerca i resolució de bugs
- Actualització de les llibreries
- Manteniment de l'aplicació

6.1. Desenvolupament de la interfície gràfica

Tal i com s'indica a la introducció del projecte, el desenvolupament de la interfície gràfica no entra a l'abast, ja que no es tracta d'un aspecte purament tècnic de l'aplicació.

Pel fet de ser un aspecte inevitable d'una aplicació Android, sí que se n'ha desenvolupat una, però s'ha limitat únicament a un desenvolupament funcional. És a dir, s'ha afegit els components necessaris a la interfície però mantenint el seus aspectes per defecte.

Tècnicament es podria publicar l'aplicació amb l'aspecte actual, però un dels aspectes clau per a l'èxit de les aplicacions és la seva estètica, una aplicació atractiva visualment té moltes més possibilitats de ser utilitzada que la resta.

Així doncs, un dels passos a seguir per a finalitzar l'aplicació és el de desenvolupar la interfície gràfica. Per a desenvolupar interfícies es pot utilitzar diverses eines, ja que al final aquesta està formada per múltiples arxius de format XML.

Una d'elles és el mateix Android Studio, utilitzat per a desenvolupar la resta de l'aplicació. Aquesta té la principal avantatge proporciona ajudes per a la

configuració el funcionament dels components de la pantalla, però a nivell de disseny gràfic no és ni de bon tros la millor.

El que sí que és cert, però, és que independentment de l'eina que s'utilitzi, sempre s'haurà d'acabar adjuntant a l'aplicació amb Android Studio, on serà necessari modificar algunes de les configuracions existents al prototip.

Per al disseny gràfic de les aplicacions, Android proporciona una sèrie d'estàndards i comprovacions de qualitat que recomana seguir durant el desenvolupament per tal d'assegurar un funcionament i aspecte homogenis.

6.2. Integració amb el microscopi

L'objectiu final de l'aplicació és el de controlar un microscopi AiScope a través de la pantalla del telèfon, de manera que l'aplicació no podrà donar-se per a finalitzada fins que la integració estigui desenvolupada.

Aquesta integració consta de tres parts:

- Recepció d'inputs de l'usuari
- Connexió i enviament de dades al microscopi
- Recepció i traducció de les dades a moviments dels motors

La recepció dels inputs ja està configurada a l'aplicació, el que ara retorna únicament la velocitat de moviment del dit per la pantalla. Aquestes dades s'hauran d'enviar al microscopi d'alguna manera, així que és necessari desenvolupar aquesta connexió.

Aquesta connexió es pot realitzar de diverses maneres, encara que les més factibles són a través d'USB o de Bluetooth. En qualsevol cas, aquesta decisió dependrà dels elements dels que disposa el microscopi i la possibilitat o no d'instal·lar els components necessaris: una antena Bluetooth o una entrada USB.

Els microscopis disposen d'una Raspberry Pi per a controlar els motors, de manera que es pot aprofitar aquest element per a publicar una sèrie d'API per a accedir i controlar els motors. Aquesta connexió s'haurà de realitzar des del mòbil, ja que és el component dominant de la integració: és el que enviarà les ordres.

L'avantatge per al desenvolupament de la integració amb una Raspberry Pi, és que existeix una quantitat molt elevada de paquets i llibreries, tant per Android com per la Raspberry, per a la seva integració i comunicació.

Els inputs de l'usuari s'hauran de transformar en un format utilitzable pels motors, que depèn del model i de la configuració del controlador que disposen. Addicionalment, aquest desplaçament dels motors s'haurà de calibrar d'alguna manera, ja que la velocitat de desplaçament de la mostra ha de dependre de l'augment de la imatge.

6.3. Cerca i resolució de bugs

Pel requeriment de temps el projecte, la cerca i resolució de bugs ha quedat fora de l'abast. Encara que, indirectament, durant el desenvolupament se n'ha trobat i resolt algun, aquest és un procés que requereix temps i dedicació exclusiva per a testejar totes i cada una de les funcions disponibles de l'aplicació en el màxim de situacions i circumstàncies possibles.

Addicionalment, un dels següents passos és desenvolupar la integració amb el microscopi AiScope, una integració que per a la seva complexitat és més que probable que contingui bugs a resoldre, de manera que el procés s'haurà de repetir quan aquesta estigui finalitzada.

6.4. Actualització i manteniment

Android és un sistema operatiu que evoluciona constantment, fet que implica que les seves aplicacions hagin d'evolucionar amb ell. Per a no perdre les funcionalitats de l'aplicació en els dispositius o sistemes operatius més moderns a mesura que aquests evolucionin, és necessari anar actualitzant periòdicament les versions de les llibreries o modificar els components que ho requereixin.

De fet, durant l'elaboració del projecte múltiples de les llibreries utilitzades han publicat noves versions que, a mesura que ha sigut possible, s'han actualitzat també a l'aplicació.

Una d'elles, però, no ha sigut possible d'actualitzar. Aquesta és la llibreria CameraX, que com s'ha comentat anteriorment s'utilitza una versió alfa. Durant el desenvolupament s'ha començat a publicar versions beta de la llibreria, però

aquestes han canviat prou algunes de les funcions per a que aquestes no siguin compatibles sense tornar a desenvolupar aquella secció.

Addicionalment, Android ha publicat un nou component gràfic: el ViewPager2. Aquest component substitueix l'antic ViewPager, que està previst que acabi deixant de ser funcional a finals d'aquest any, de manera que serà necessari actualitzar-la.

7. Conclusions

Un dels meus objectius personals a l'inici del projecte era el d'aprofundir i posar en ús els meus coneixements de programació, ja que feia temps que volia iniciar una transició de l'enginyeria industrial "pura" a una enginyeria de software, més centrada en el desenvolupament.

Volia aprofitar l'oportunitat que em brindava aquest projecte per a experimentar en primera persona el desenvolupament de software d'inici a fi, experiència que m'hauria de servir com a porta d'entrada a posicions laborals relacionades amb el desenvolupament de software.

L'evolució del projecte ha deixat en evidència la meva ignorància inicial en el desenvolupament d'aplicacions Android. Si bé és cert que tenia certs coneixements de programació i havia desenvolupat algun script en Python, ni de bon tros s'acostaven al nivell que realment es requereix per a desenvolupar una aplicació sencera.

Fins al moment de començar el projecte, tot el software que havia desenvolupat es limitava a codis de màxim 200 línies per a automatitzar tasques diàries on interactuava amb software com l'Excel o SAP per a extreure dades i posteriorment representar-les.

La meva principal errada va ser emprendre el desenvolupament de l'aplicació de la mateixa manera que enfocava el software que havia desenvolupat fins al moment. L'augment de la complexitat del projecte hauria requerit realitzar una recerca molt més àmplia, no només enfocada en el desenvolupament en Python, sinó en el desenvolupament en Android en general.

En retrospectiva, hi ha força coses que hauria fet diferent a l'inici del projecte, però la primera hauria sigut aconsellar-me d'algú expert en el desenvolupament en Android, m'hauria estalviat molts mal de caps, en especial el de trobar-m en un punt on em vaig veure obligat a descartar la feina de 3 mesos.

Personalment, encara que d'una banda em sento satisfet d'haver pogut completar el projecte m'ha quedat alguna espina clavada que m'hauria agradat acabar, en especial la integració amb el microscopi.

És una llàstima que per esdeveniments de causa major com el COVID-19 no haig pogut aconseguir un microscopi físic amb el que poder fer proves i desenvolupar la integració i deixar una aplicació acabada.

Per concloure vull expressar la meva intenció de seguir col·laborant amb AiScope per a, com a mínim, arribar a publicar l'aplicació i possiblement en noves iteracions i versions d'aquesta.

8. Bibliografia

1. AI Scope – Diagnosing global diseases with Computer vision https://aiscope.net Accés: 18/02/2019
2. GitHub - theaiscope/aiscope https://github.com/theaiscope/aiscope Accés: 18/02/2019
3. Crunch base https://www.crunchbase.com/organization/ai-scope Accés: 18/02/2019
4. Welcome to Python.org https://www.python.org/ Accés: 19/02/2019
5. 3.6.8 Documentation https://docs.python.org/3/ Accés: 19/02/2019
6. PEP 20 -- The Zen of Python, 2004 Consulta: 19/02/2019
7. Visual Studio Code https://code.visualstudio.com/ Accés: 19/02/2019
8. Documentation for Visual Studio Code https://code.visualstudio.com/docs Accés: 19/02/2019
9. Python - Visual Studio Marketplace https://marketplace.visualstudio.com/items?itemName=ms-python.python Accés: 19/02/2019
10. Kivy: Cross-platform Python Framework for NUI Development https://kivy.org/ Accés: 24/02/2019
11. Kivy · GitHub https://github.com/kivy/ Accés: 24/02/2019
12. Welcome to Kivy - Kivy 1.10.1 documentation https://kivy.org/doc/stable/ Accés 24/02/2019
13. Kivy Garden by kivy-garden https://kivy-garden.github.io/gallery.html Accés 27/02/2019
14. Ulloa, Roberto (2013). *Kivy: Interactive Applications in Python* Consulta 01/03/2019
15. SQLAlchemy - The Database Toolkit for Python https://www.sqlalchemy.org/ Accés 19/03/2019
16. Object Relational Tutorial - SQLAlchemy 1.3 Documentation https://docs.sqlalchemy.org/en/13/orm/tutorial.html Accés 19/03/2019
17. Stack Overflow - Where Developers Learn, Share, & Build Careers https://stackoverflow.com Accés 19/03/2019
18. Android | The platform pushing what's possible https://www.android.com/ Accés 05/06/2019

19. Android Developers https://developer.android.com/ Accés 05/06/2019
20. Android Studio Preview | Android Developers https://developer.android.com/studio Accés 05/06/2019
21. Meet Android Studio | Android Developers https://developer.android.com/studio/intro Accés 05/06/2019
22. Kotlin and Android | Android Developers https://developer.android.com/kotlin Accés 05/06/2019
23. Kotlin Bootcamp for Programmers https://www.udacity.com/course/kotlin-bootcamp-for-programmers--ud9011 Accés 05/06/2019
24. Documentation | Android Developers https://developer.android.com/docs Accés 25/06/2019
25. Developer Guides | Android Developers https://developer.android.com/guide Accés 25/06/2019
26. Navigation | Android Developers https://developer.android.com/guide/navigation Accés 17/11/2019
27. Camera | Android Developers https://developer.android.com/training/camera#documentation Accés 03/12/2019
28. android.hardware.camera2 | Android Developers https://developer.android.com/reference/android/hardware/camera2/package-summary Accés 03/12/2019
29. Imai, Tomoaki (2018): *Understanding Camera2 API from callbacks* https://proandroiddev.com/understanding-camera2-api-from-callbacks-part-1-5d348de65950 Accés 10/12/2019
30. Android Camera2 API by Huyen Tue Dao https://www.youtube.com/watch?v=KhqGphh6KPE Accés 27/12/2019
31. CameraX | Android Developers https://developer.android.com/training/camerax Accés 12/02/2020
32. Getting Started with CameraX https://codelabs.developers.google.com/codelabs/camerax-getting-started/#0 Accés 12/02/2020
33. camera-samples/CameraXBasic at master · android/camera-samples · GitHub https://github.com/android/camera-samples/tree/master/CameraXBasic Accés 28/02/2020

34. Create a List with RecyclerView | Android Developers https://developer.android.com/guide/topics/ui/layout/recyclerview Accés 04/03/2020
35. GitHub - android/sunflower: A gardening app illustrating Android development best practices with Android Jetpack. https://github.com/android/sunflower Accés 08/03/2020
36. Slide between fragments using ViewPager | Android Developers https://developer.android.com/training/animation/screen-slide Accés 18/03/2020
37. views-widgets-samples/ViewPager2 at master · android/views-widgets-samples · GitHub https://github.com/android/views-widgets-samples/tree/master/ViewPager2 Accés 18/03/2020
38. Access app-specific files | Android Developers https://developer.android.com/training/data-storage/app-specific Accés 30/03/2020
39. Sharing files | Android Developers https://developer.android.com/training/secure-file-sharing Accés 01/04/2020
40. Martorell, Joana Aina: *AI Scope: against global diseases* Consulta 24/04/2020
41. Fernández, Ariadna: *Project of low-cost microscope automation for data gathering* Consulta 24/04/2020