Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Aplicación móbil de control e xestión agroforestal

Estudante: Damián Pardiñas Rodríguez
Dirección: Óscar Fresnedo Arias
Dirección: Francisco Laport López

A Coruña, setembro de 2020.

" Se podes soñalo, podes facelo."
Walt Disney

Agradecementos

A todas as persoas que me apoiaron durante este longo camiño, especialmente meus pais piares fundamentais. Ademais, non me podo esquecer de Óscar e Francisco titores neste proxecto, grazas pola oportunidade e pola gran axuda que me brindastes neste proxecto.

Resumo

Os sectores forestal e da agricultura teñen un gran peso na economía a nivel nacional e, especialmente, en Galicia. Unha mostra diso é o feito de que arredor do 65% do terreo parcelario de Galicia é masa forestal. Os labores de agricultura e forestais conlevan unha gran carga de traballo e organización, dende o control dos traballos realizados sobre cada parcela ata a xestión económica dos gastos e beneficios derivados das mesmas. Existen moitos traballos que necesitan dun exhaustivo control, por exemplo, a localización de certas parcelas pola dificultade que presenta a xeografía da nosa contorna ou o control de produtos fitosanitarios que deben ser aplicados sobre cada cultivo, xa que son produtos que poden ser daniños para o traballador ou para o usuario final do cultivo se a aplicación e control dos mesmos non se fai con total detalle. Neste caso, a aplicación dos produtos fitosanitarios dependen do tamaño da parcela, o cultivo ou da praga a tratar. Obter esta información e asegurar unha correcta aplicación destes produtos é moi relevante para garantir a seguridade. Debido a estes problemas que se poden apreciar no contexto agroforestal, nace a necesidade de elaborar unha solución para facilitar a organización e o control das cargas de traballo asociadas aos diferentes tipos de cultivos ou parcelas, para axudar a garantir que os tratamentos requiridos son aplicados de forma adecuada seguindo as recomendacións sanitarias e, finalmente para xestionar os gastos e beneficios derivados do traballo. Para iso, neste proxecto abordarase o deseño e implementación dunha ferramenta na que o usuario co seu móbil poida xeolocalizar unha parcela mediante a posición GPS, obtendo deste modo información (tamaño, tipo de parcela, ...) útil para diversos traballos. Ademais, a aplicación permitirá ter un rexistro e control dos traballos realizados nesas parcelas así como a xestión económica das mesmas.

Abstract

The forestry and agricultural sectors have a great weight in the economy at a national level and, especially, in Galicia. One example is the fact that around 65% of the land in Galicia is forest. The agricultural and forestry work is very labor intensive and requires a high degree of organization, since the control of the work done on each plot is linked to the economic management of expenses and benefits derived from them. There are many jobs that need an exhaustive control, for example, the location of certain plots because of the difficulty presented by the terrain of our region or the control of phytosanitary products that must be applied to each crop, since these are products that can be harmful to the worker or to the final user of the crop if the application and control of these products is not carried out in detail. For example, the application of phytosanitary products depends on the size of the plot, the crop or

the pest to be treated. Obtaining this information is very important to guarantee safety. Due to these problems that can be appreciated in the agroforestry context, it is needed to elaborate a solution to facilitate the organization and the control of the work loads associated to the different types of crops or plots, to help guarantee that the required treatments are applied in an adequate way following the sanitary recommendations and, finally, to manage the costs and benefits derived from the work. For this purpose, this project addresses the design and implementation of a software tool in which the user with his mobile can locate a plot through GPS position, thus obtaining information (size, type of plot, ...) useful for various jobs. This tool will also allow to have control over the work done on these plots as well as the economic management of them.

Palabras chave:

- Aplicación móbil
- Flutter
- Android
- iOS
- Agricultura
- Forestal
- Xeolocalización
- Parcela
- Xestión
- Control
- Agrotab

Keywords:

- Mobile app
- Flutter
- Android
- iOS
- Agriculture
- Forestry
- Geolocation
- Plot
- Management
- Control
- Agrotab

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

1.1 Contexto e Motivación

Os sectores forestal e da agricultura son piares da economía da maior parte dos países do mundo. A nivel nacional, estes sectores teñen un gran peso na economía como demostra o feito de que, segundo o Instituto Nacional de Estadística (INE), o grupo o que pertencen ditos sectores tivo unha oferta a prezo de mercado de 33.017 millóns de euros no ano 2019 [1]. Ademais, supuxo o emprego directo de 461.200 persoas no mesmo período de tempo. Outro dato que merece a pena ser destacado é que, no segundo trimestre de 2020, supuxo o emprego directo de 452.600 persoas. Isto reflexa a importancia destes sectores xa que, tendo en conta a incidencia da pandemia do Covid-19, o impacto desta apenas se notou a diferencia de outros moitos sectores da economía. Isto demostra que se trata de sectores estratéxicos e esenciais para a nosa economía en calquera situación.

Se nos centramos en Galicia, estes sectores aínda cobran maior relevancia. Unha mostra diso é o feito de que o 65% do terreo parcelario de Galicia é masa forestal e ademais, segundo a Consellería de Medio Rural, unhas 55.220,85 ha son de monte público e unhas 663.488,96 ha son de Montes Veciñais en Man Común (MVMC) [2].

Como en calquera outro sector existe unha necesidade de control e xestión das diferentes actividades que é preciso realizar. En concreto, os labores de agricultura e forestais conlevan unha gran carga de traballo e organización, dende o control dos traballos realizados sobre cada parcela ata a xestión económica dos gastos e beneficios derivados das mesmas. Existen moitos traballos que necesitan dun exhaustivo control como, por exemplo, a localización de certas parcelas pola dificultade que presenta a xeografía da nosa contorna ou o control de produtos fitosanitarios que deben ser aplicados sobre cada cultivo, xa que son produtos que poden ser daniños para o traballador ou para o usuario final do cultivo se a aplicación e control dos mesmos non se fai con total detalle. De feito, segundo o real decreto 1311/2012 [3], onde se establece o marco de actuación para o uso sostible dos produtos fitosanitarios, indícase a

obrigatoriedade de levar o control e xestión das aplicacións deste tipo de produtos. Esta tarefa tamén é condición necesaria para a solicitude de axudas da Política Agrícola Común (PAC).

Os traballos nestes sectores acostuman a ser moi físicos durante toda a xornada, ó que lle hai que sumar o tempo de xestión e control. Por esta razón, nace a necesidade de elaborar unha solución para facilitar ditas tarefas. Un traballador non pode levar consigo todos os papeis derivados do historial dunha determinada parcela para poder administrala "in situ". Pero se esa información a puidese obter e administrar desde o seu dispositivo móbil, as cargas de traballo reduciríanse significativamente e simplificaríase enormemente todo o proceso de xestión.

Mediante a creación dunha aplicación móbil pretendese dar ó traballador a posibilidade de controlar e xestionar as parcelas e as tarefas derivadas das mesmas, como a aplicación de tratamentos ou rexistrar traballos de cultivo. Do mesmo xeito, coa aplicación será posible, de forma automática, extraer información sobre os gastos e beneficios derivados dos cultivos para cada parcela e tamén en global.

Un exemplo práctico da vantaxe da aplicación proposta sería o caso no que un traballador tivera que aplicar un produto fitosanitario sobre unha determinada parcela. En principio, a secuencia de pasos requiridos sería a seguinte:

1. Localizar a parcela cos servizos de Sigpac ou Catastro e así obter a superficie total da parcela. Con estes datos, podería determinar a cantidade de produto fitosanitario necesario para a aplicación sobre a mesma.

2. Ir ó lugar onde adoite almacenar os produtos fitosanitarios e comprobar a cantidade da que dispón, para saber se ten o necesario para a aplicación ou necesita adquirir máis.

3. Buscar a información de aplicación para o produto fitosanitario.

4. Aplicar o produto sobre a parcela.

5. Buscar a información sobre o número de rexistro e prazo de seguridade para o produto fitosanitario.

6. Anotar a man todos os datos da parcela, do produto fitosanitario, data de aplicación e prazo de seguridade no caderno de campo.

Coa solución construída neste proxecto, todo este proceso reduciríase a 3 sinxelos pasos, facendo toda a xestión dende o móbil:

1. Mirar a dispoñibilidade de produto no apartado de "Alpendre" e ver a información da aplicación.

2. Seleccionar a parcela e engadir a tarefa de aplicar produto fitosanitario introducindo a cantidade usada e o prazo de seguridade.

3. Aplicar o produto fitosanitario sobre a parcela.

1.2 Obxectivos

Coa creación da aplicación móbil preténdese dar ó traballador a posibilidade de xestionar e controlar todas as tarefas derivadas dunha parcela dunha maneira rápida e sinxela, para así poder rebaixar os tempos e cargas de traballo.

Os obxectivos deste proxecto pódense resumir nos seguintes puntos:

- Xeolocalizar unha parcela e obter determinada información sobre a mesma.
- Levar a cabo un rexistro dos traballos/tarefas realizadas sobre unha determinada parcela.
- Levar a cabo un rexistro de cultivos, produtos fitosanitarios e produtos fertilizantes que se teñan almacenados para a súa aplicación sobre unha determinada parcela.
- Poder xestionar os gastos e beneficios mediante unha análise estadística.
- Poder gardar na nube toda a información anterior para poder acceder desde outro dispostivo Android ou iOS.

1.3 Estructura do documento

A presente memoria que describe o proxecto realizado estrutúrase nos seguintes capítulos:

1. **Introdución:** Capítulo actual no cal se fai un primeiro contacto coa natureza e motivación do proxecto.

2. **Estado da arte:** Neste capítulo realízase un estudo analizando o sector agrícola e forestal na actualidade, así como os métodos ou ferramentas que usan estes sectores para cubrir as necesidades de control e xestión.

3. **Fundamentos tecnolóxicos:** Neste capítulo realízase unha descrición exhaustiva das ferramentas, tecnoloxías e linguaxes de programación usadas durante todo o proxecto.

4. **Metodoloxía:** Neste capítulo preséntase unha descrición da metodoloxía seguida durante o desenvolvemento do proxecto.

5. **Análise de requisitos:** Neste capítulo realízase a análise de requisitos e análise de riscos do proxecto.

6. **Planificación e análise económico:** Neste capítulo explícase a planificación inicial do proxecto co seu correspondente seguimento e faise unha estimación dos custos do proxecto.

7. **Deseño da aplicación:** Neste capítulo abórdase o deseño de marca, diagramas da aplicación e prototipado.

8. **Desenvolvemento:** Neste capítulo descríbese en detalle o desenvolvemento da aplicación seguindo as diferente interaccións realizadas.

9. **Conclusións:** Neste capítulo realízase unha reflexión sobre o proxecto e trázanse algunhas liñas futuras.

Capítulo 2

Estado da arte

Neste capítulo búscase analizar algunhas ferramentas existentes no mercado, as cales cubran de certa maneira os obxectivos deste proxecto. Dado que a necesidade de control dos produtos fitosanitarios e fertilizantes está condicionada por unha normativa nacional, este capítulo divídise en ferramentas internacionais e ferramentas nacionais.

2.1 Internacional

2.1.1 Granular

Granular é un software para a administración de explotacións gandeiras de Estados Unidos, que axuda aos gandeiros coas xestións derivadas da plantación de cultivos en grandes extensións [4]. Un dos seus principais inconvenientes é que non ten soporte para tratamentos fitosanitarios. Está máis enfocado no sistema de regadío dos cultivos e nas transaccións económicas. O seu funcionamento está baseado nunha folla de cálculo (figura 2.1) e dá soporte a aplicacións de escritorio e móbil. É de pago e está enfocado ás grandes explotacións.

2.1.2 Farmbrite

Farmbrite é un software de administración de explotacións gandeiras de Estados Unidos que está enfocado aos típicos ranchos americanos [5]. Dado que toda a xestión enfócase nas necesidades derivadas do coidado dos animais (figura 2.2) e non da parcela, tampouco ten soporte para tratamentos fitosanitarios. Esta ferramenta permite análises financeiros de todos os movementos realizados. Sen embargo, só está dispoñible para equipos Windows, é de pago e ten un custo de 15 $ mensuais.

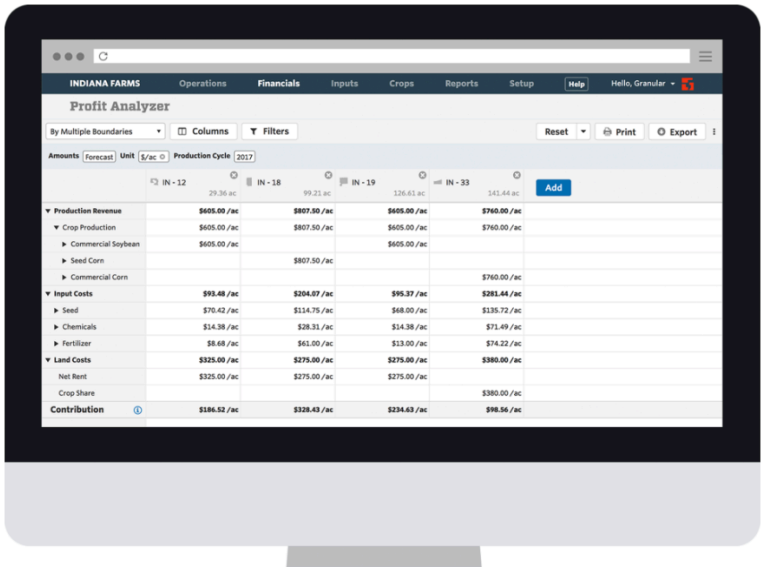

Figura 2.1: Aparencia de Granular.

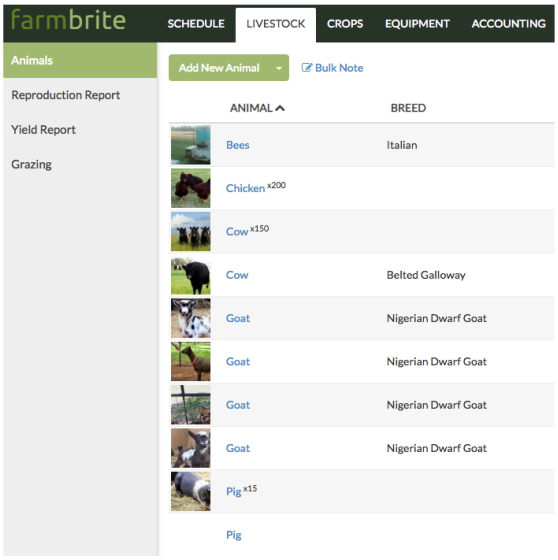

Figura 2.2: Aparencia de Farmbrite.

2.1.3 EasyFarm

EasyFarm é un software para a xestión dunha granxa [6]. É a ferramenta máis veterana de todas as citadas, xa que naceu en 1983. Está máis enfocada á xestión económica derivada dunha empresa ou actividade comercial, xa que permite administrar empregos e os gastos derivados. Non ten soporte para tratamentos fitosanitario e está baseada en follas de cálculo, como se pode apreciar na figura 2.3. Ademais, é de pago e ten un custo de 33 $ mensuais.

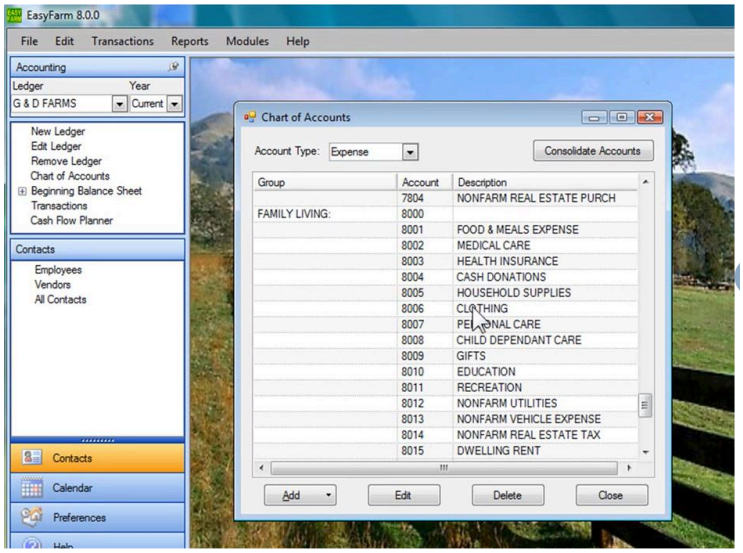

Figura 2.3: Aparencia de Easyfarm.

2.2 Nacionais

2.2.1 Caderno de campo

É a solución en papel que aporta o ministerio de agricultura para o control dos produtos fitosanitarios [7]. Como se pode ver na figura 2.4, non se pode controlar ou xestionar ningún ámbito económico relacionado cos traballos agrícolas ou forestais, dado que a súa única finalidade é cubrir as obrigas do real decreto 1311/2012 [3]. Existe tamén unha versión en folla de cálculo. Esta solución é gratuita.

Figura 2.4: Aparencia de Caderno de Campo.

2.2.2 Agroptima

Agroptima é un software para a xestión de explotacións agrícolas, o cal permite administrar todo o desenvolvemento dos cultivos dunha maneira sinxela e intuitiva [8]. Está baseado no caderno de campo anteriormente mencionado. Ten versión móbil pero está derivada da de escritorio (figura 2.5). É de pago e o custo vai condicionado en función do tamaño da explotación.

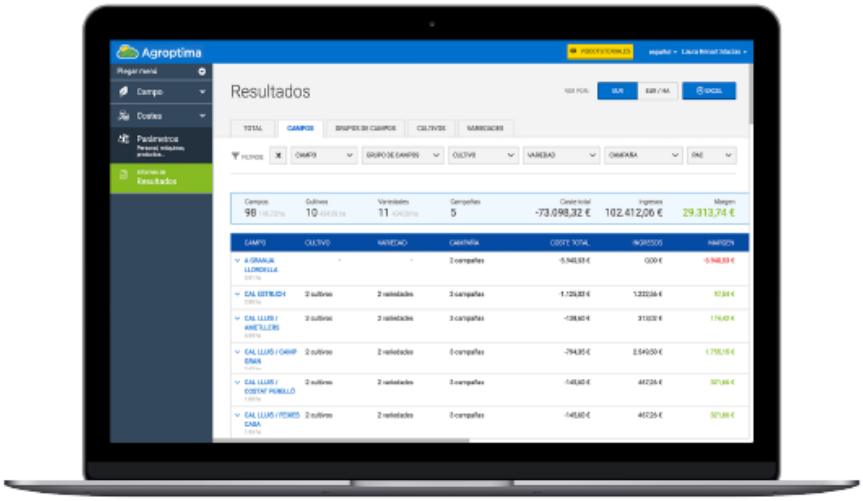

Figura 2.5: Aparencia de Agroptima.

2.2.3 ISAGRI

ISAGRI é un software para a xestión de explotacións agrícolas, o cal permite administrar a man de obra e os diferentes recursos dunha explotación [9]. Do mesmo xeito que no caso anterior, está baseado no caderno de campo anteriormente mencionado, e ten unha versión móbil que está derivada da de escritorio (figura 2.6). É de pago e o custo vén condicionado pola cantidade de hectáreas.

2.3 Conclusión

Despois da revisión do estado da arte, podemos concluir que as alternativas presentadas non cobren todos os obxectivos planeados. Como podemos ver na táboa 2.1, por un lado, as solucións internacionais non están pensadas para o uso e necesidades da nosa contorna. Non dispoñen dun sistema de xestión de produtos fitosanitarios nin a posibilidade de xeolocalizar unha parcela. En canto ás solucións nacionais todas están baseadas no caderno de campo polo

Figura 2.6: Aparencia de ISAGRI.

que deixan de lado a necesidade de xestionar estatisticamente cada parcela e os beneficios derivados das mesmas.

	Xeolocalización	Xestión tarefas	Control fitosanitarios	Xestión económica	Gratuita	Gardado na nube
Granular	-	✓	-	✓	-	✓
Farmbrite	-	-	-	✓	-	✓
EasyFarm	-	✓	-	✓	-	-
Caderno de campo	-	-	✓	-	✓	-
Agroptima	-	✓	✓	✓	-	✓
ISAGRI	-	✓	✓	✓	-	✓

Táboa 2.1: Comparación das ferramentas do mercado.

Capítulo 3

Fundamentos Tecnolóxicos

Neste capítulo búscase analizar as ferramentas, linguaxes de programación e tecnoloxías usadas para o completo desenvolvemento do proxecto.

3.1 Ferramentas

3.1.1 VSCode

Visual Studio Code (VSCode) é un lixeiro e poderoso editor de código creado por Microsoft [10]. Caracterízase por ser compatible con varias linguaxes de programación e pola súa flexibilidade xa que lle permite ó usuario un control total sobre a súa aparencia (figura 3.1) e o seu funcionamento. Ademais, está totalmente integrado con Git, polo que toda a xestión do repositorio pode facerse desde o mesmo editor. A súa versatilidade radica no uso de extensións que o fan válido practicamente para calquera traballo. Para este proxecto, esta ferramenta usouse para o completo desenvolvemento da aplicación, usando as extensións de Dart e Flutter [11] [12]. A primeira destas extensións da soporte para empregar a linguaxe Dart e a segunda da soporte completo á lóxica de Flutter.

3.1.2 Android Studio

Android Studio é o Integrated Development Environment (IDE) oficial para o desenvolvemento de aplicacións para a plataforma Android (figura 3.2), e está baseado en IntelliJ IDEA de JetBrain [13]. Está dispoñible para as principais plataformas e caracterízase por dar soporte integrado a todas as tecnoloxías pertencentes ó Google Cloud Platform. Tamén proporciona un dispositivo virtual de Android que se utiliza para executar e probar as aplicacións que se poidan desenvolver. Para este proxecto, esta ferramenta usouse para probar e executar as diferentes partes da aplicación mediante o seu dispositivo virtual. Ademais, a súa instalación incorpora o Software Development Kit (SDK) de Android necesario para o desenvolvemento

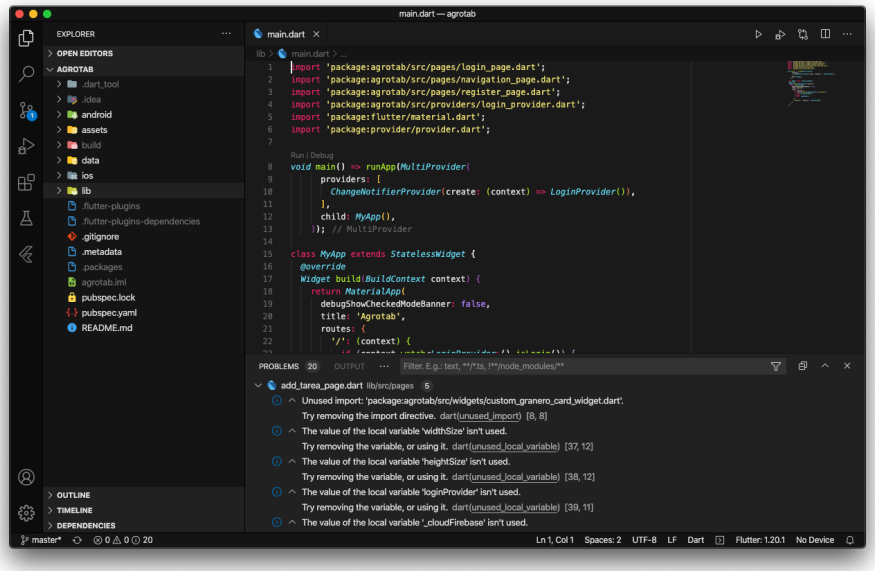

Figura 3.1: Aparencia de VSCode.

en Flutter.

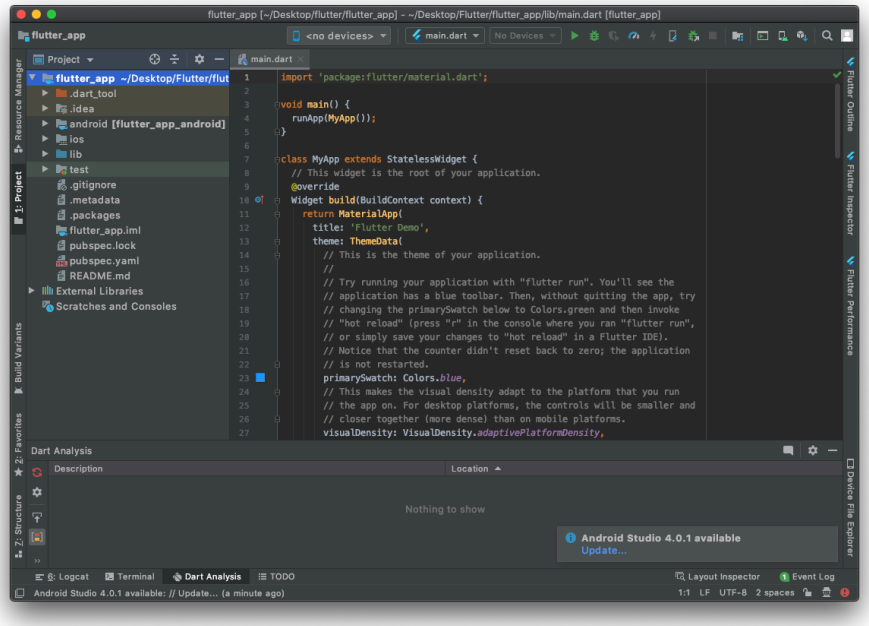

Figura 3.2: Aparencia de AndroidStudio.

3.1.3 Xcode

Xcode é o IDE oficial de Apple, e so está dispoñible para a plataforma macOS [14]. Caracterízase por conter un conxunto de ferramentas destinadas ó desenvolvemento de software para macOS, iOS, iPadOS, watchOS e tvOS. Incorpora un dispositivo virtual que permite probar e executar as aplicacións que se van a desenvolver. Para este proxecto, esta ferramenta usouse para poder manexar as configuracións e administracións de permisos en iOS (figura 3.3), tamén para poder probar e executar a aplicación mediante o seu dispositivo virtual. De xeito similar ó comentado para Android Studio, a súa instalación incorpora o SDK de iOS necesario para o desenvolvemento en Flutter.

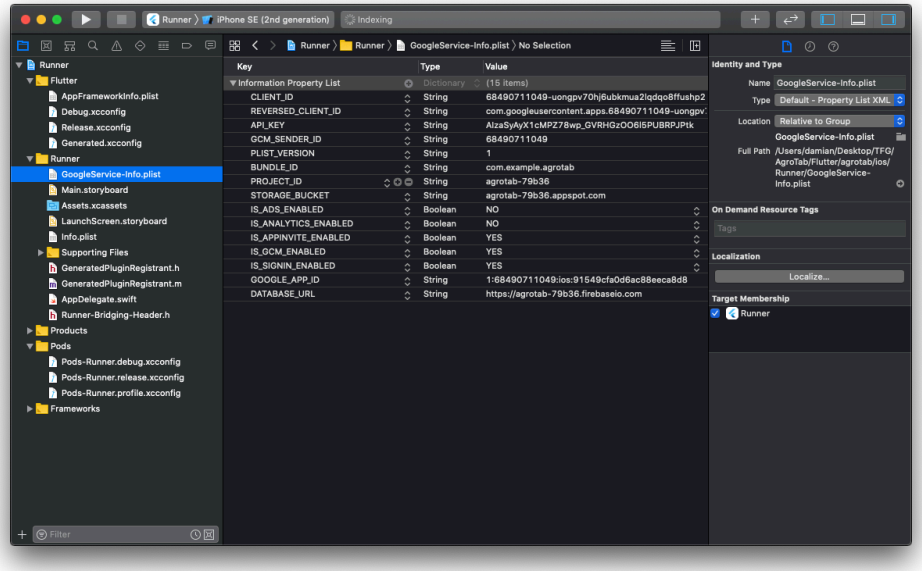

Figura 3.3: Aparencia de Xcode.

3.1.4 Adobe XD

Adobe XD é un editor de gráficos vectoriais creado por Adobe Inc para deseñar e crear interfaces para páxinas web e aplicacións móbiles [15]. É gratuíto e está dispoñible para as plataformas Windows e macOS. Caracterízase por aportar ferramentas para o deseño de propotipado (figura 3.4), dende algo básico ata un deseño máis complexo, permitindo a configuración e visualización dos fluxos de traballo e experiencia do usuario ao navegar. Para este proxecto, esta ferramenta usouse para poder deseñar o prototipado inicial e manexar o fluxo de traballo e experiencia do usuario ao navegar entre as diferentes pantallas da aplicación.

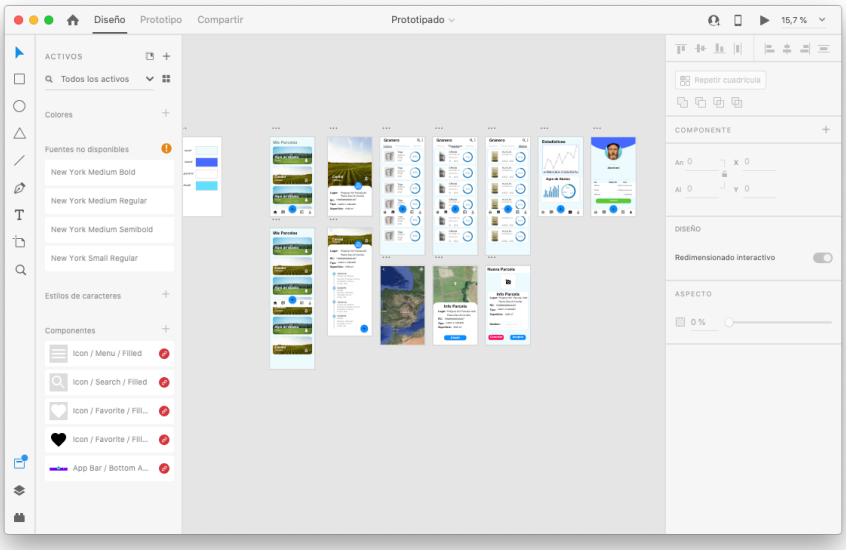

Figura 3.4: Aparencia de Adobe XD.

3.1.5 Adobe Ilustrator

Adobe Ilustrator é un editor de gráficos vectoriais creado por Adobe Inc para o deseño gráfico e ilustración [16]. É parte de AdobeCreativeCloud e está presente para as plataformas Windows e macOS. Para este proxecto, esta ferramenta usouse para poder deseñar o logo da aplicación e diversas imaxes vectoriais usadas na User interface (UI) da aplicación.

3.1.6 Adobe Photoshop

Adobe Photoshop é un editor de fotografías creado por Adobe Inc para o retoque de fotografías e gráficos [17]. É parte de AdobeCreativeCloud e está presente para as plataformas Windows e macOS. Para este proxecto, esta ferramenta usouse para realizar retoques sobre ilustracións usadas no logo e diversas partes da aplicación.

3.1.7 Notion

Notion é unha aplicación que proporciona compoñentes como bases de datos, taboleiros de Kanban, wikis e calendarios entre outros [18]. Ten unha versión gratuíta e está dispoñible para as principais plataformas incluíndo dispositivos móbiles. Caracterízase por proporcionar a posibilidade de crear sistemas de xestión de coñecemento, toma de notas e xestión de proxectos entre outros (figura 3.5). Esta ferramenta usouse para a xestión de todo o relacionado co proxecto, toda a planificación facendo uso de taboleiros Kanban, tamén como repositorio para gardar documentación usada en toda a análise, deseño e desenvolvemento.

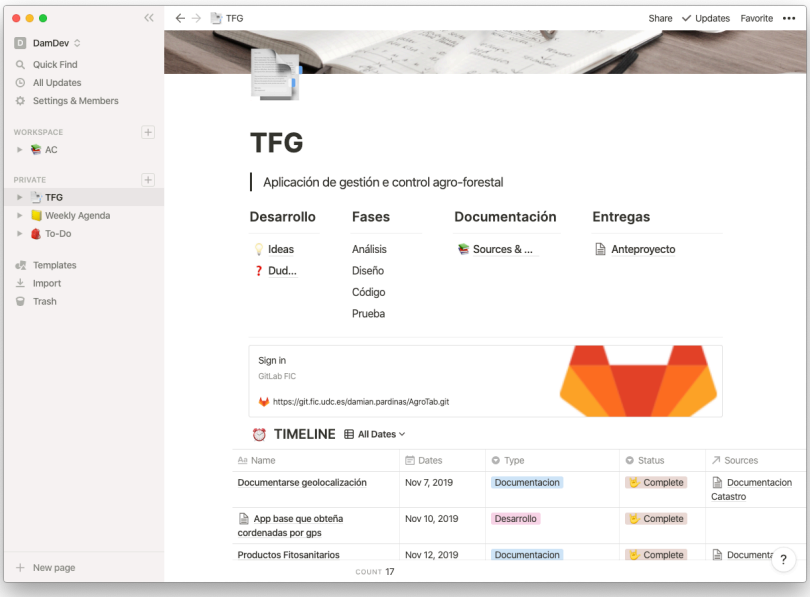

Figura 3.5: Aparencia de Notion.

3.2 Linguaxes de programación

3.2.1 Dart

Dart é unha linguaxe de programación de código aberto orientado a obxectos e creado por Google [19]. Naceu en 2011 coa finalidade de ser unha alternativa a JavaScript, e a súa última versión estable data de febreiro de 2019. Caracterízase por ser altamente versátil, polo que pode ser usado para o desenvolvemento de aplicacións móbil/escritorio, *scripts* e *back-end*. Algúns dos puntos fortes desta linguaxe son:

- Optimización para UI: Dart é sinxelo de aprender, xa que conta cunha sintaxe familiar, e tamén está optimizado para a construción de interfaces con operandos que facilitan a concorrencia e a espera.

- Desenvolvemento produtivo: Dart ten a capacidade de facer cambios no código fonte de forma iterativa usando a característica *"hot reload"*, a cal é unha recarga en quente para ver instantaneamente o resultado do cambio realizado coa aplicación en execución.

- Rendemento en todas as plataformas: Dart compila as aplicacións en código máquina nativo para o seu inicio instantáneo. Esto permite que as aplicacións desenroladas utilizando esta linguaxe estén optimizadas para cada plataforma.

Esta linguaxe será a máis usada no proxecto dado que esta é a linguaxe adoptada por Flutter para levar acabo o completo desenvolvemento dunha aplicación.

3.2.2 Kotlin

Kotlin é unha linguaxe de programación de tipado estático que corre sobre a máquina virtual de Java [20]. Foi creada principalmente por JetBrains. Naceu en 2016 e a última versión estable é a 1.3.72 que data de novembro de 2019. Foi adoptado por Android Studio como a linguaxe por defecto (elixible entre esta e Java) para o desenvolvemento de aplicacións en Android. Algúns dos puntos fortes desta linguaxe son:

- Concisa: Kotlin ten a capacidade de reducir drasticamente a cantidade de código lixo.
- Segura: Kotlin pode evitar erros como os de punteiro a nulo.
- Facilidade de uso: Pódese empregar en calquera IDE de Java ou construílo dende a liña de comandos.

Esta linguaxe úsase no proxecto para definir a configuración interna da aplicación e o manexo de permisos relacionados co hardware do disposivo Android.

3.2.3 Swift

Swift é unha linguaxe de programación multiparadigma creada por Apple e enfocada ó desenvolvemento de aplicacións para iOS e macOS [21]. Naceu en 2014 e a última versión estable é a 5.2.5 que data de marzo de 2020. Caracterízase por ser rápida e eficaz, e intégrase á perfección con código escrito en Objective-C. Algúns dos puntos fortes desta linguaxe son:

- Segura: O feito de estar orientada a patróns de deseño, a obriga de declarar tipos de datos e que o compilador comprobe os mesmos fai que Swift aporte moita seguridade.
- Rápida: Un mesmo desenvolvemento é valido para diferentes plataformas. Por exemplo, un mesmo desenvolvemento permite obter unha aplicación que será válida tanto para iOS como para macOS.
- Facilidade de corrección: Swift aporta un control de erros avanzado, o cal permite recuperar dunha forma controlada os erros non esperados na programación.

Esta linguaxe úsase no proxecto para as configuracións internas e o manexo de permisos relacionados co hardware do dispositivo iOS.

3.2.4 JavaScript

JavaScript é unha linguaxe de programación interpretada [22]. Caracterízase por estar baseada en prototipos, é multi-paradigma e dinámica. Tamén presenta soporte para programación orientada a obxectos e, dende 2012, todos os navegadores modernos soportan completamente ECMAScript 5.1, o cal é unha versión de JavaScript. Esta linguaxe úsase no proxecto para todo o desenvolvemento realizado en Node.js.

3.2.5 CEL

Common Expression Language (CEL) é unha linguaxe que implementa unha semántica común para a avaliación dunha expresión, permitindo que as diferentes aplicacións interoperen máis facilmente [23]. Ten unha sintaxe parecida a C++, Go, Java ou TypeScript. Algúns dos puntos máis importantes son:

- Sinxeleza.
- Velocidade.
- Pouco mantemento.
- Moi extensible.
- Desenvolvemento amigable.

Esta linguaxe úsase no proxecto para manexar a política de seguridade de Firebase, mediante o desenvolvemento e configuración das denominadas *"Firebase Rules"*.

3.3 Tecnoloxías

3.3.1 Flutter

Flutter é un *framework* de código aberto creado por Google para crear aplicacións, compiladas de forma nativa, para móbil, web e escritorio a partir dunha única base de código [24]. Naceu en maio de 2017 e a última versión estable 1.20.1 data de agosto de 2020. Flutter permite crear unha aplicación para varias plataformas pero necesita do SDK de cada unha delas para poder realizar dito traballo. Por exemplo, para poder crear unha aplicación para Android necesítase o SDK de Android e para crear unha aplicación para iOS necesítase o SDK correspondente de iOS. Está dispoñible para o seu uso en múltiples plataformas, pero este uso vese en realidade condicionado polo hardware co que imos traballar. Por exemplo, no caso de realizar unha aplicación iOS, imos precisar dispoñer dun dispositivo con macOS, xa que esta é a única plataforma onde podemos instalar Xcode para poder realizar o posterior

desenvolvemento da aplicación. En canto ó desenvolvemento en si, Flutter pode operar tanto con dispositivos físicos como cos simuladores de Android Studio ou Xcode (figura 3.6), se ben isto está condicionado a que ambos estean instalados no mesmo equipo onde se está a implemetar a aplicación. O desenvolvemento con Flutter está baseado no uso dos chamados *widgets*. Un *widget* en Flutter representa unha descrición inmutable de parte da interface de usuario. Tanto gráficos, textos coma animacións están creadas utilizando *widgets*.

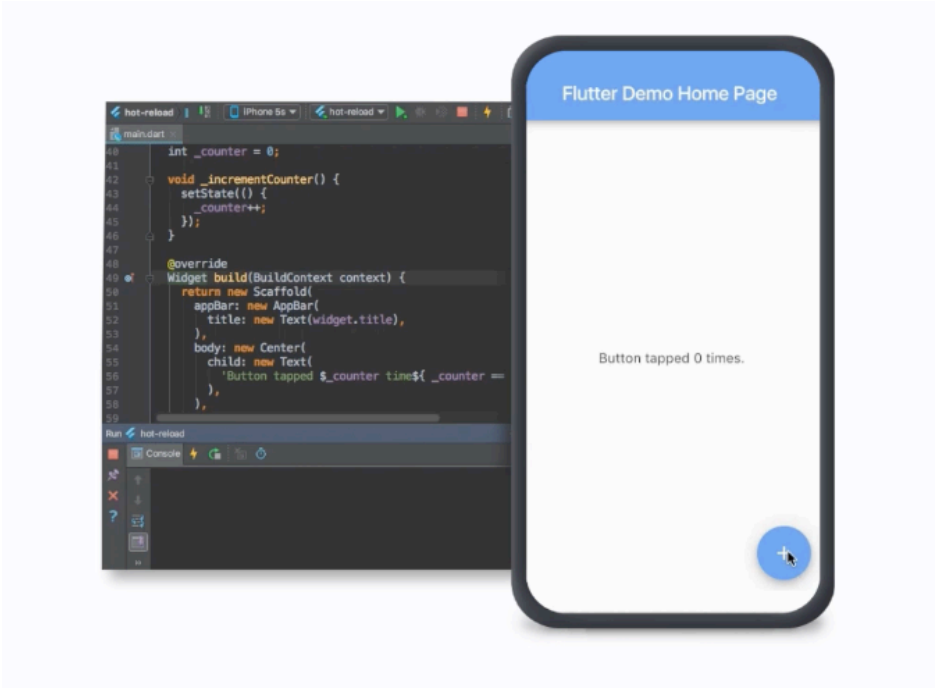

Figura 3.6: *Framework* Flutter facendo uso do simulador iOS de Xcode.

Os puntos máis destacados desta tecnoloxías son:

- Calidade nativa: As aplicacións nativas son aquelas que se desenvolven especificamente para un determinado sistema operativo e que, polo tanto, están optimizadas para aproveitar as características dese sistema operativo. O desenvolvemento en Flutter incorpora todas as vantaxes das aplicacións nativas pero baixo un código único, é dicir, sen necesidade de implementar un código diferente para cada sistema considerado.
- Experiencia de uso: Flutter presenta axudas como incluír os paquetes correspondentes ó deseño baseado en Material Design de Google ou Cupertino de Apple, polo que a experiencia de uso é óptima e as interfaces de usuario son idénticas as desenvolvidas por calquera das compañías.
- Tempo de carga: Flutter aporta un alto rendemento e, como resultado, experiméntanse tempos de carga por debaixo do segundo en calquera das plataformas.

- Desenvolvemento áxil e rápido: Mediante a característica *"hot reload"* pódese ver os cambios realizados no código en tempo real no dispositivo ou nos simuladores.

A pesar destas vantaxes hai que destacar a xuventude do *framework*. Esta circunstancia pode acarrear certos problemas como falta de documentación ou problemas de estabilidade, dado que a maior parte dos *widgets* dispoñibles en pub.dev [25], o cal é o repositorio de widgets oficial de Flutter, están en fase beta. Para crear un plugin en Flutter pódese facer uso doutros, isto crea unha cadea de dependencias, dado que Flutter está en continuas actualizacións e os plugins ó ser desenvolvidos por terceiros, non todos actualizan as dependencias. Polo que este factor pode facer o desenvolvemento máis complexo do que en realidade debería de ser.

3.3.2 Android

Android é un sistema operativo creado por Android Inc, empresa adquirida posteriormente por Google, para o seu uso en dispositivos móbiles con pantalla táctil [26]. Actualmente, Android é o sistema operativo móbil máis empregado no mundo. Está baseado no kernel de Linux e outros software de código aberto. Android pode adaptarse a múltiples resolucións de pantallas e soporta diversos tipos de conexión. Naceu en 2008 e a última versión estable, Android 10, saíu en 2020. En canto á súa composición, o núcleo esta implementado en C e C++, mentres que o UI está desenvolvido en Java/Kotlin.

3.3.3 iOS

iOS é un sistema operativo creado por Apple para o seu uso en dispositivos móbiles como iPhone, iPad e iPod, xa que Apple non permite a instalación de iOS en hardware de terceiros [27]. iOS deriva de macOS que, a súa vez, está baseado en DarwinBSD, e pode adaptarse a múltiples resolucións de pantallas e soporta diversas conexións. Naceu en 2007 e a última versión estable, iOS 13.6, saíu en 2020. En canto a súa composición, está implementado en C, C++ e Objective-C/Swift.

3.3.4 Node.js

Node.js é un *framework* en tempo de execución de JavaScript que está deseñado para construír aplicacións de redes escalables [28]. Caracterízase por ser asíncrono cunha E/S de datos sobre unha arquitectura orientada a eventos e está baseado no motor V8 de Google. A finalidade de usar Node.js neste proxecto foi evitar que o código JavaScript se execute en navegadores, como ocorre na maioría dos casos, para pasar a executarse nun servidor. É de código aberto, naceu en 2009 e a última versión estable 12.14.0 saíu en decembro de 2019. Neste proxecto usarase para crear unha aplicación que permita servir grandes documentos .Json a

Cloud Firebase, debido a que nativamente non dispón de ningunha opción de importación de ficheiros .json. En concreto para importar os datos de produtos fitosanitarios e fertilizantes.

3.3.5 Firebase

Firebase é unha plataforma que persegue facilitar o desenvolvemento e a creación de aplicacións móbiles ou web [29]. Deste xeito, o seu uso permite acadar unha elevada calidade nas aplicacións finais de forma rápida e eficiente. O principal obxectivo de Firebase é mellorar o rendemento das aplicacións mediante o uso de diversas funcionalidades destinadas a conseguir que as aplicacións sexan moito máis manexables, seguras e de fácil acceso para os usuarios. As súas principais carecterísticas pódense resumir nos seguintes puntos:

- Soporta múltiples plataformas: Firebase dispón de soporte para aplicacións móbiles, tanto iOS como Android, e para aplicacións web.
- Desenvolvemento gratuíto: o uso da plataforma é gratis. Os únicos custos son en función da escalabilidade, polo que para o momento de desenvolvemento da aplicación o custo é 0.
- Análise: Permite ter un control avanzado do rendemento das aplicacións mediante diversas métricas de análise. Estes datos recadados facilitan a toma de decisións.
- Permite monetizar unha aplicación: Firebase ten a posibilidade de xerar diñeiro mediante anuncios e publicidades varias dentro da aplicación que se desenvolva.
- Permite un rápido crecemento: É o obxectivo principal de Firebase, que mediante os servicios ofrecidos, de forma simple e rápida, poder crear unha aplicación segura e escalable.

Os principais servicios que ofrece pódense agrupar en:

- Realtime database: É o servicio de base de datos orixinal de Firebase, o cal permite sincronización de datos en tempo real [30].
- Cloud Firestore: É o substituto de Realtime database, ademais de ser o recomendado por Google para un desenvolvemento en dispositivos móbiles [31]. Permite o almacenamento e sincronización de datos entre usuarios e dispositivos a escala global, todo isto mediante unha base de datos NoSQL aloxada na nube. Tamén permite sincronización en tempo real e soporte sen conexión.
- Cloud Functions: Permite realizar funcións propias dun *back-end* sen necesidade de administrar ou crear un servidor propio [32]. Estas funcións pódense activar con eventos de servicios Firebase ou de terceiros por medio de *webhooks*.

- Authentication: Permite administrar de maneira sinxela e segura todos os usuarios da aplicación que se está a desenvolver [33]. Ofrece varios métodos de autenticación, como usando o correo electrónico ou mediante as principais redes sociais entre outros.
- Hosting: Ten capacidade de aloxar unha páxina web, e automaticamente asígnalle un certificado SSL gratuíto para ter unha experiencia segura e confiable [34].
- Cloud Storage: Permite almacenar e compartir contido xerado polos usuarios, como imaxe, voz ou vídeo [35].
- Firebase ML: É a última incorporación a Firebase, e permite engadir características baseadas no *machine learning* ás aplicacións [36].

Neste proxecto faremos uso dos servizos Cloud Firebase para almacenar toda a información derivada da aplicación, Authentication para poder manexar o rexistro e inicios de sesión dos usuarios e Cloud Storage para poder almacenar contido multimedia.

Capítulo 4

Metodoloxía

Unha metodoloxía de desenvolvemento pódese definir como: "Un conxunto de procedementos, técnicas, ferramentas e documentación auxiliar que axudan ós desenvolvedores nos esforzos de crear un novo *software*" [Avison and Fitzgerald, 2003].

Na actualidade existen unha gran cantidade de metodoloxías a elixir para o desenvolvemento dun proxecto. Estas acostúmanse a clasificar en dous grupos [38]:

Metodoloxías tradicionais

- Baseadas en normas que nacen dos estándares seguidos polos desenvolvedores.
- Resistencia ós cambios.
- Proceso moi controlado.
- Existe un contrato prefixado.
- O cliente interactúa co equipo de desenvolvemento mediante reunións.
- Dirixidas a grupos grandes.
- Moitos roles.
- A arquitectura de *software* é indispensábel.

Metodoloxías áxiles

- Baseadas en heurísticas que xorden de prácticas de produción de código.
- Preparadas para cambios durante o transcurso do proxecto.
- Proceso menos controlado.

- Non existe un contrato tradicional ou é moi flexible.
- O cliente forma parte do desenvolvemento.
- Dirixida a grupos pequenos, menos de 10 integrantes.
- Poucos roles.
- Menos énfase na arquitectura de *software*.

Tendo en conta as carateristícas do proxecto que se vai desenvolver, considerouse que as metodoloxías áxiles se axustan mellor a este tipo de proxectos. Por esa razón, decidiuse empregar Scrum co apoio de Kanban para poder controlar o estado das tarefas a desenvolver en cada incremento.

4.1 Scrum

A metodoloxía Scrum para o desenvolvemento áxil de *software* é un marco de traballo deseñado para concibir unha colaboración eficaz dos equipos no desenvolvemento dun proxecto. Esta metodoloxía fai uso dun conxunto de regras e artefactos propios e, ademais, define uns roles que xeran a estrutura necesaria para o seu correcto funcionamento [39].

Scrum utiliza un enfoque incremental baseado no control empírico dos procesos, o cal aporta transparencia, control e adaptación. Desta forma, conséguese que un equipo Scrum sexa auto-xestionado, multifuncional e traballe mediante interaccións. A entrega do produto efectuase por iteracións de xeito que, en cada unha delas, créanse novas funcionalidades ou modifícanse as que o cliente requira sobre o produto.

Scrum define tres roles principais:

- *Scrum master*: Ten a función de asegurar que o equipo está adoptando a metodoloxía, as súas practicas, valores e normas.
- *Produto owner*: É unha soa persoa que representa ó cliente. Polo tanto, é o responsable de maximizar o valor do produto e o traballo do equipo de desenvolvemento.
- Equipo de desenvolvemento: É o encargado de converter as necesidades do cliente en iteracións funcionais do produto.

Na figura 4.1 pódense apreciar os principais elementos, eventos e actores involucrados nun proceso baseado en Scrum. Como se pode ver, Scrum define un evento principal denominado *Sprint*, que é unha xanela de tempo onde se crea unha versión utilizable do produto, é dicir, un incremento. Cada un destes *Sprints* está considerado un proxecto independente e componse dos seguintes elementos:

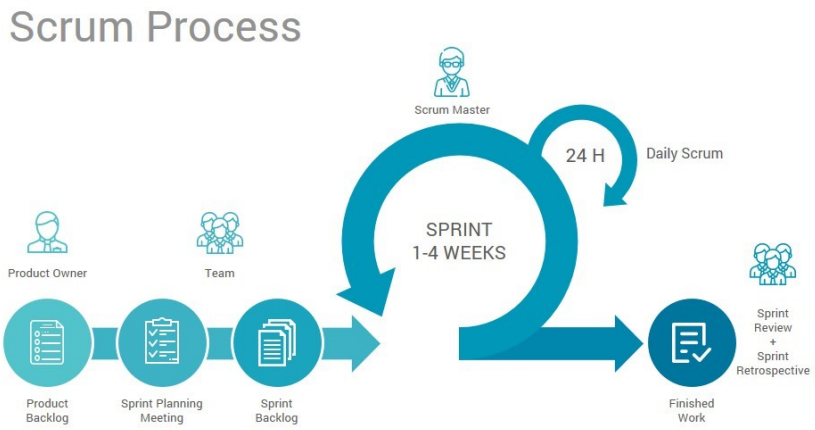

Figura 4.1: Proceso *Scrum*.

- Planificación do *Sprint*: Reunión na que se define o plan de traballo.
- *Daily Scrum*: Reunión diaria realizada polo equipo para analizar o alcanzado dende a última reunión.
- Traballo de desenvolvemento.
- Revisión do *Sprint*: Reunión que se realiza ó final do *Sprint* na que o equipo analiza os problemas que xurdiron, mostra o produto e o seu funcionamento.

Neste proxecto seguirase esta metodoloxía áxil xa que se axusta ás necesidade do mesmo. O desenvolvemento será incremental polo que, seguindo a planificación por *Sprints* que marca Scrum, poderemos ter, en certo modo, control do proceso e podemos cambiar algunha necesidade sobre os requisitos iniciais conforme avance o proxecto. Para o manexo das tarefas dentro de cada *Sprint*, farase uso dos taboleiros Kanban para así poder organizar as tarefas en pendentes, en desenvolvemento e finalizadas.

4.2 Kanban

Kanban é unha metodoloxía áxil que se caracteriza por ser moi fácil de usar, de actualizar e de asumir por parte do equipo de desenvolvemento. Destaca pola súa xestión de tarefas moi visual, a cal permite ver dunha ollada o estado do proxecto [40]. Os principios de Kanban son:

- Calidade.
- Simplicidade.
- Mellora continua.

- Flexibilidade.

O punto forte de Kanban radica na súa maneira de xestionar as tarefas. Esta xestión faina mediante un taboleiro onde só existen 3 columnas que se corresponden cos estados no que as tarefas se poden atopar:

- Non comezada.
- Comezada.
- Finalizada.

Desta forma, cunha simple ollada ó taboleiro, podemos ver o estado do proxecto e calquera membro do equipo pode ter un control sobre os estados de cada tarefa.

Este proxecto fai uso dos taboleiros Kanban para poder manexar as tarefas de cada *Sprint* dunha maneira rápida e sinxela. A integración dos taboleiros de Kanban na metodoloxía do proxecto realizouse facendo uso da ferramenta Notion [18]. Na figura 4.2, pode apreciarse como Notion permite organizar as tarefas indicando unha data prevista de inicio e clasificar estas por tipo. Ademais, permite asociar cada tarefa cunha páxina que alberga o resultado desta.

⏰ TIMELINE All Dates

Name	Dates	Type	Status	Sources
Documentarse geolocalización	Nov 7, 2019	Documentacion	Complete	Documentacion Catastro
App base que obteña cordenadas por gps	Nov 10, 2019	Desarrollo	Complete	
Productos Fitosanitarios	Nov 12, 2019	Documentacion	Complete	Documentación Productos Fitosanitarios
Solicitud de Datos productos fitosanitarios e abonos	Nov 13, 2019	Datos	In Progress	
WMS catastro	Nov 20, 2019	Documentacion	Complete	
WMS sigpac	Nov 24, 2019	Documentacion	Complete	

Figura 4.2: Taboleiro Kanban na ferramenta Notion.

4.3 Metodoloxía no proxecto

Para aplicar esta metodoloxía ó proxecto, fixéronse pequenas modificacións debido a que, normalmente, está orientada a equipos de maior tamaño. Con respecto ós roles da metodoloxía, os titores deste proxecto tomaron o rol de *Scrum Master*, mentres que o autor do proxecto tomou os roles de *Product Owner* e de equipo de desenvolvemento. En canto ó propio desenvolvemento do proxecto, este dividiuse en tres *Sprints* de 80 horas, tendo reunións cos titores do proxecto ao final de cada un deles para realizar a revisión e planificación do seguinte *Sprint*. Dado que a realización da *Daily Scrum* non ten sentido pola natureza do proxecto, utilizáronse os taboleiros *Kanban* como apoio para ter un control das tarefas a realizar cada día.

Capítulo 5

Análise de requisitos

Neste capítulo abordarase a especificación de requisitos do producto a desenvolver, como se traducen esos requisitos nos correspondentes casos de uso e, finalmente, farase unha análise de riscos do proxecto.

5.1 Especificación de requisitos

Unha especificación de requisitos é unha descrición completa do comportamento do sistema que se vai desenvolver. En concreto, describiranse os actores que interactúan coa aplicación e os casos de uso do sistema:

5.1.1 Actores

Un actor é unha entidade externa ó sistema que realiza algun tipo de interacción co mesmo. Para este proxecto, só exite un actor principal:

ACT-<01>	**Usuario**
Descrición	Este actor representa ó rol da persoa que vai a realizar as tarefas dispoñibles na aplicación.

Táboa 5.1: Actor 1: Usuario.

5.1.2 Casos de uso

Un caso de uso é unha descrición da secuencia de interaccións que se producen entre un actor e o sistema cando o actor quere realizar unha determinada tarefa usando o sistema. Para este proxecto e, tendo en conta as funcionalidades que debería ter a aplicación, identificáronse os seguintes casos de uso:

CU-<01>	**Sign-Up**	
Descrición	Usuario rexístrase na aplicación.	
Precondición	-	
Secuencia Normal	**paso**	**Acción**
	1	Sistema mostra os campos: nome, email e contrasinal.
	2	Usuario introduce nome, email, contrasinal e rexístrase.
	3	Sistema valida os datos e envía mail de confirmación.
	4	Usuario confirma email.
	5	Fin do caso de uso.
Postcondición	Usuario queda rexistrado na aplicación.	
Excepcións	**paso**	**Acción**
	2'	Campos baleiros: O sistema notifícao.
	3'	Email ou contrasinal incorrectos: O sistema notifícao.
	3'	Email xa rexistrado: O sistema notifícao.
Importancia	Alta	

Táboa 5.2: Caso de uso 1: Sign-Up.

CU-<02>	**Sign-In**	
Descrición	Usuario identifícase na aplicación.	
Precondición	Usuario ten que estar rexistrado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Sistema mostra os campos: email e contrasinal.
	2	Usuario introduce email, contrasinal e inicia sesión.
	3	Sistema valida os datos e accede á conta do usuario.
	4	Fin do caso de uso.
Postcondición	Usuario queda identificado na aplicación.	
Excepcións	**paso**	**Acción**
	2'	Campos baleiros: O sistema notifícao.
	3'	Email ou contrasinal incorrectos: O sistema notifícao.
	3'	Email non rexistrado: O sistema notifícao.
Importancia	Alta	

Táboa 5.3: Caso de uso 2: Sign-In.

CU-<03>	**Sign-Out**	
Descrición	Usuario pecha a sesión na aplicación.	
Precondición	Usuario ten que estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó perfil.
	2	Usuario pecha a sesión.
	3	Sistema pecha a sesión do usuario.
	4	Fin do caso de uso.
Postcondición	Usuario queda coa sesión pechada na aplicación.	
Importancia	Alta	

Táboa 5.4: Caso de uso 3: Sign-Out.

CU-<04>	**Forgot-Password**	
Descrición	Restablecer contrasinal esquecido.	
Precondición	Email debe ser válido e estar rexistrado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario solicita restablecer contrasinal.
	2	Sistema mostra campo email.
	3	Usuario introduce email e solicita restablecer.
	4	Sistema envía mail para retablecer contrasinal.
	5	Usuario restablece contrasinal.
	6	Fin do caso de uso.
Postcondición	Contrasinal anterior deixa de ser válido.	
Importancia	Media	

Táboa 5.5: Caso de uso 4: Forgot-Password.

CU-<05>	**Visualizar parcelas**	
Descrición	Ver unha lista de todas as parcelas engadidas.	
Precondición	Usuario debe estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Sistema mostra lista de parcelas.
	3	Fin do caso de uso.
Postcondición	-	
Importancia	Alta	

Táboa 5.6: Caso de uso 5: Visualizar parcelas.

CU-<06>	**Engadir parcela**	
Descrición	Engadir unha nova parcela á lista de parcelas.	
Precondición	Usuario debe estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Usuario solicita engadir parcela.
	3	Sistema mostra mapa.
	4	Usuario solicita localización GPS.
	5	Sistema mostra mapa na posición actual.
	6	Usuario selecciona parcela e engádea.
	7	Sistema mostra información da parcela e campos: tipo parcela (Agrícola/Forestal), imaxe e nome.
	8	Usuario introduce tipo parcela, nome, selecciona imaxe e engade a parcela.
	6	Fin do caso de uso.
Postcondición	Parcela queda engadida na aplicación.	
Excepcións	**paso**	**Acción**
	6’	Parcela xa engadida: o sistema notifícao.
Importancia	Alta	

Táboa 5.7: Caso de uso 6: Engadir parcela.

CU-<07>	**Visualizar produtos**	
Descrición	Ver produtos engadidos ó alpendre, agrupados por tipo.	
Precondición	Usuario debe estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó alpendre.
	2	Sistema mostra lista dun tipo de produto.
	3	Usuario pode escoller tipo de produto a mostrar.
	4	Fin do caso de uso.
Postcondición	-	
Importancia	Alta	

Táboa 5.8: Caso de uso 7: Visualizar productos.

CU-<08>	**Engadir produto tipo Fitosanitario ou Fertilizante**	
Descrición	Engadir un novo produto do tipo fitosanitario ou fertilizante ó alpendre.	
Precondición	Usuario debe estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó alpendre.
	2	Usuario solicita engadir un novo produto.
	3	Sistema mostra lista de todos os produtos posibles agrupados por tipo.
	4	Usuario escolle tipo de produto a engadir e selecciona un.
	5	Sistema mostra información sobre o produto e os campos: cantidade, unidade e prezo.
	6	Usuario introduce cantidade, unidade (Kg/L), prezo e engade produto.
	7	Sistema valida os datos e engade o produto ó alpendre.
	8	Fin do caso de uso.
Postcondición	-	
Excepcións	**paso**	**Acción**
	6’	Campos baleiros: O sistema impide engadir produto.
Importancia	Alta	

Táboa 5.9: Caso de uso 8: Engadir poduto tipo fitosanitario ou fertilizante.

CU-<09>	**Engadir produto tipo Cultivo**	
Descrición	Engadir un novo produto do tipo cultivo ó alpendre.	
Precondición	Usuario debe estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó alpendre.
	2	Usuario solicita engadir un novo produto.
	3	Sistema mostra os campos: nome, fabricante, concepto, tipo, cantidade, unidade e prezo.
	4	Usuario introduce nome, fabricante, concepto, tipo, cantidade, unidade (Kg/Und.), prezo e engade o produto.
	5	Sistema valida os datos e engade o produto ó alpendre.
	6	Fin do caso de uso.
Postcondición	-	
Excepcións	**paso**	**Acción**
	4'	Campos baleiros: O sistema impide engadir produto.
Importancia	Alta	

Táboa 5.10: Caso de uso 9: Engadir poduto tipo cultivo.

CU-<10>	**Eliminar produto**	
Descrición	Eliminar un produto do alpendre.	
Precondición	Usuario debe estar identificado na aplicación e o produto debe estar engadido ó alpendre.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó alpendre.
	2	Usuario selecciona un produto e solicita eliminalo.
	3	Sistema elimina o produto do alpendre.
	4	Fin do caso de uso.
Postcondición	Produto queda eliminado do alpendre.	
Excepcións	**paso**	**Acción**
	2'	Produto usado nunha tarefa: O sistema impide eliminar o produto.
Importancia	Media	

Táboa 5.11: Caso de uso 10: Eliminar un produto do alpendre.

CU-<11>	**Eliminar parcela**	
Descrición	Eliminar unha parcela da aplicación.	
Precondición	Usuario debe estar identificado na aplicación e a parcela debe estar engadida na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Usuario selecciona unha parcela e solicita eliminala.
	3	Sistema elimina a parcela da aplicación.
	4	Fin do caso de uso.
Postcondición	Parcela queda eliminada da aplicación.	
Excepcións	**paso**	**Acción**
	2’	Parcela contén tarefa: O sistema notifícao e impide eliminar a parcela.
Importancia	Media	

Táboa 5.12: Caso de uso 11: Eliminar unha parcela da aplicación.

CU-<12>	**Visualizar tarefas dunha parcela.**	
Descrición	Ver os diferentes traballos que se realizan nunha parcela.	
Precondición	Usuario debe estar identificado na aplicación e a parcela debe estar engadida na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Usuario selecciona unha parcela.
	3	Sistema mostra tarefas realizadas na parcela ordeadas por data.
	4	Fin do caso de uso.
Postcondición	-	
Importancia	Alta	

Táboa 5.13: Caso de uso 12: Visualizar tarefas dunha parcela.

CU-<13>	**Engadir tarefa a unha parcela.**	
Descrición	Engadir unha nova tarefa a unha determinada parcela.	
Precondición	Usuario debe estar identificado na aplicación e a parcela debe estar engadida na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Usuario selecciona unha parcela e solicita engadir unha tarefa.
	3	Sistema mostra os campos: data, tipo tarefa (fitosanitario, fertilizante, colleita, plantación e outro), produto do alpendre, cantidade, prezo do labor e prazo de seguridade.
	4	Usuario introduce os campos e engade a tarefa.
	5	Sistema engade tarefa na parcela.
	6	Fin do caso de uso.
Postcondición	Tarefa queda engadida na parcela.	
Excepcións	**paso**	**Acción**
	2’	Prazo de seguridade vixente: Sistema notifícao e impide engadir tarefas.
	4’	Campos baleiros: sistema impide engadir tarefa.
Importancia	Alta	

Táboa 5.14: Caso de uso 13: Engadir tarefa a unha parcela.

CU-<14>	**Eliminar tarefa dunha parcela**	
Descrición	Eliminar unha tarefa dunha determinada parcela.	
Precondición	Usuario debe estar identificado na aplicación e a tarefa debe estar engadida na parcela.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Usuario selecciona unha parcela e solicita eliminar unha tarefa.
	3	Sistema elimina tarefa da parcela.
	4	Fin do caso de uso.
Postcondición	Tarefa queda eliminada da parcela.	
Importancia	Alta	

Táboa 5.15: Caso de uso 14: Eliminar tarefa dunha parcela.

CU-<15>	**Obter información sobre un produto fitosanitario.**	
Descrición	Obter información de compostos e aplicación dun determinado produto fitosanitario.	
Precondición	Usuario debe estar identificado na aplicación e o produto debe estar engadido na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó alpendre.
	2	Usuario selecciona un produto e solicita obter información.
	3	Sistema descarga pdf con información.
	4	Fin do caso de uso.
Postcondición	-	
Importancia	Media	

Táboa 5.16: Caso de uso 15: Obter información dun produto fitosanitario.

CU-<16>	**Obter información sobre un produto fertilizante**	
Descrición	Obter información de compostos e aplicación dun determinado fertilizante.	
Precondición	Usuario debe estar identificado na aplicación e o produto debe estar engadido na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó alpendre.
	2	Usuario selecciona un produto e solicita obter información.
	3	Sistema mostra páxina web con información.
	4	Fin do caso de uso.
Postcondición	-	
Importancia	Media	

Táboa 5.17: Caso de uso 16: Obter información dun produto fertilizante.

CU-<17>	**Obter información sobre un cultivo**	
Descrición	Obter información dun determinado cultivo.	
Precondición	Usuario debe estar identificado na aplicación e o produto debe estar engadido na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó alpendre.
	2	Usuario selecciona un produto e solicita obter información.
	3	Sistema mostra páxina web con información.
	4	Fin do caso de uso.
Postcondición	-	
Importancia	Media	

Táboa 5.18: Caso de uso 17: Obter información dun cultivo.

CU-<18>	**Obter información catastral dunha parcela**	
Descrición	Obter información dunha determinada parcela.	
Precondición	Usuario debe estar identificado na aplicación e a parcela debe estar engadida na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Usuario selecciona unha parcela e solicita obter información.
	3	Sistema mostra páxina web con información.
	4	Fin do caso de uso.
Postcondición	-	
Importancia	Media	

Táboa 5.19: Caso de uso 18: Obter información catastral dunha parcela.

CU-<19>	**Obter navegación dunha determinada parcela.**	
Descrición	Abrir posición da parcela en Google Maps ou Apple Maps e obter indicacións.	
Precondición	Usuario debe estar identificado na aplicación e a parcela debe estar engadida na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó inicio.
	2	Usuario selecciona unha parcela e solicita obter navegación.
	3	Sistema mostra Google Maps ou Apple Maps con posición.
	4	Fin do caso de uso.
Postcondición	-	
Importancia	Media	

Táboa 5.20: Caso de uso 19: Obter navegación dunha parcela.

CU-<20>	**Obter estadísticas financeiras.**	
Descrición	Obter estadísticas financeiras de forma global e para cada parcela para o ano vixente.	
Precondición	Usuario debe estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese as estadísticas.
	2	Sistema mostra beneficios fronte ós gastos, descomposición dos gastos a nivel global e o mesmo para cada parcela de forma individual.
	3	Fin do caso de uso.
Postcondición	-	
Importancia	Alta	

Táboa 5.21: Caso de uso 20: Obter estadísticas financeiras.

CU-<21>	**Obter información usuario.**	
Descrición	Obter información do usuario	
Precondición	Usuario debe estar identificado na aplicación.	
Secuencia Normal	**paso**	**Acción**
	1	Usuario diríxese ó perfil.
	2	Sistema mostra email, nome, data de rexistro, nº parcelas, balance actual e avatar.
	3	Fin do caso de uso.
Postcondición	-	
Importancia	Alta	

Táboa 5.22: Caso de uso 21: Obter información do usuario.

5.1.3 Requisitos non funcionais

RNF-<01>	**Usabilidade**
Descrición	Medida na cal a aplicación pode ser usada polos usuarios conseguindo os obxectivos específicos con efectividade, eficiencia e satisfacción.
Importancia	Alta

Táboa 5.23: Requisito non funcional 01: Usabilidade

RNF-<02>	**Escalabilidade**
Descrición	Permitir engadir novas funcionalidades no futuro sen perder a calidade da aplicación.
Importancia	Alta

Táboa 5.24: Requisito non funcional 02: Escalabilidade.

RNF-<03>	**Seguridade**
Descrición	A aplicación debe manter seguros os datos e información dos usuarios.
Importancia	Alta

Táboa 5.25: Requisito non funcional 03: Seguridade.

5.2 Análise de riscos

Un risco pódese definir como un evento ou circunstancia cuxa probabilidade de ocorrencia é incerta pero que, no caso de ocorrer, ten un efecto negativo sobre os obxectivos do proxecto. É por isto que nace a necesidade de facer unha análise de riscos:

1. Estimación da duración do proxecto.
    - Tipo: Risco de proxecto.
    - Probabilidade: Moi probable.
    - Impacto: Crítico.
    - Causa: A duración do proxecto abrangue 3 meses e divídese en diferentes fases como: documentación, análise, deseño e desenvolvemento. En calquera das fases pódese subestimar ou sobreestimar a súa duración, polo que pode producir un gran impacto no proxecto.
    - Mitigación: Realizar unha planificación do proxecto analizando cada *Sprint* e engadindo un maior tempo as tarefas que se poden subestimar.
2. Carencias no uso das tecnoloxías.
    - Tipo: Risco de proxecto.
    - Probabilidade: Moi probable.
    - Impacto: Crítico.
    - Causa: Dado que no proxecto úsanse tecnoloxías novas para o estudante, isto pode facer que o desenvolvemento sexa custoso e pode propiciar a necesidade dunha formación previa, a cal pode ter un impacto na duración do proxecto.
    - Mitigación: Realizar unha análise concisa das tecnoloxías a usar no proxecto.
3. Casos de uso non cubren os obxectivos do proxecto.
    - Tipo: Risco de proxecto.
    - Probabilidade: Improbable.
    - Impacto: Marxinal.
    - Causa: Debido á complexidade do proxecto e a natureza dos seus obxectivos, a elaboración dos casos de uso pode resultar complexa.
    - Mitigación: Realizar unha descrición da secuencia normal de actuación para cada caso de uso.

4. Casos de uso non abordados.

    - Tipo: Risco de proxecto.
    - Probabilidade: Probable
    - Impacto: Crítico.
    - Causa: Debido ó tamaño do proxecto, pode ocorrer que algúns casos de uso non sexan identificados, provocando isto un grave problema durante o desenvolvemento.
    - Mitigación: Creación dun deseño de UI para facilitar a visibilidade das funcionalidades para cubrir os obxectivos e desta maneira poder descubrir novos casos de uso.

5. Estabilidade da tecnoloxía.

    - Tipo: Risco técnico.
    - Probabilidade: Moi probable.
    - Impacto: Catastrófico.
    - Causa: O proxecto fai uso de tecnoloxías relativamente novas como Flutter, de xeito que a maior parte dos paquetes para o desenvolvemento están en fase beta e o *framework* está con constantes actualizacións que sobrescriben a anterior, polo que existe unha obrigatoriedade para actualizar a mesma. Isto en moitos casos fai que manter unha versión estable do entorno de desenvolvemento sexa imposible, o que pode causar un impacto catastrófico en casos de uso xa terminados de desenvolver ou impactar negativamente na duración do proxecto.
    - Mitigación: Realizar unha investigación previa para cada paquete en busca de versións estables.

Capítulo 6

Planificación e análise económico

Neste capítulo abordarase a planificación do proxecto tendo en conta os casos de uso definidos no capítulo 5. En primero lugar, vanse defininir os diferentes *backlogs* indicando os casos de uso que se implementarán en cada *Sprint* e, posteriormente, detallarase a duración, demoras e custos do proxecto.

6.1 Product backlog

Despois de analizar os requisitos do sistema e identificar os casos de uso correspondentes, decidiuse organizar o *Product backlog* do produto do seguinte xeito:

- Xestión básica de usuarios: agrupa as historias de usuarios que corresponden cos casos de uso CU-<01> (táboa 5.2), CU-<02> (táboa 5.3) e CU-<03> (táboa 5.4).
- Xestión alpendre: agrupa as historias de usuario que corresponden cos casos de uso CU-<07> (táboa 5.8), CU-<08> (táboa 5.9) e CU-<09> (táboa 5.10).
- Xestión das parcelas: agrupa as historias de usuario que corresponden cos casos de uso CU-<05> (táboa 5.6) e CU-<06> (táboa 5.7).
- Xestión das tarefas: agrupa as historias de usuario que corresponde cos casos de uso CU-<12> (táboa 5.13) e CU-<13> (táboa 5.14).
- Xerar informe estadístico: agrupa as historias de usuario que corresponden co caso de uso CU-<20> (táboa 5.21).
- Xestión avanzada de usuarios: recuperar contrasinal + información: agrupa as historias de usuario que corresponden cos casos de uso CU-<04> (táboa 5.5) e CU-<21> (táboa 5.22).

- Xestión avanzada das parcelas e do alpendre: borrar datos + obter información: agrupa as historias de usuario que corresponden cos casos de uso CU-<10> (táboa 5.11), CU-<11> (táboa 5.12), CU-<14> (táboa 5.15), CU-<15> (táboa 5.16), CU-<16> (táboa 5.17), CU-<17> (táboa 5.18), CU-<18> (táboa 5.19) e CU-<19> (táboa 5.20).

6.2 Sprint backlogs

O desenvolvemento do proxecto organizouse en tres *Sprints* coa seguinte distribución das tarefas do *Product backlog* nos correspondentes *Sprint backlogs*:

- Sprint 1
  - Xestión básica de usuarios.
  - Xestión alpendre.
- Sprint 2
  - Xestión das parcelas.
  - Xestión das tarefas.
  - Xerar informe estadístico.
- Sprint 3
  - Xestión avanzada de usuarios: recuperar contrasinal + información.
  - Xestión avanzada de parcelas e do alpendre: borrar datos + obter información.

6.3 Planificación

A planificación do proxecto foi realizada coa ferramenta Notion. Esta ferramenta permítenos analizar a estrutura do proxecto e elaborar unha planificación axeitada a partir desos datos. Na figura 6.1, pode apreciarse como a planificación está enfocada en 5 puntos principais:

1. Análise: Neste punto abórdase a elección e documentación da idea do proxecto, a análise e documentación das tecnoloxías para levalo acabo, e a especificación de requisitos. É unha parte indispensable para poder comezar o desenvolvemento da aplicación. A súa duración estimouse en 160 horas debido a envergadura das tarefas, tendo en conta ademais que a maior parte destas son para a documentación da idea do proxecto. Na realidade presentou unha demora de 24 horas, e isto ocorreu debido a subestimar o tempo necesario para o estudo e documentación das tecnoloxías.

Tarefa		Horas estimadas	Horas reais
1	**Analises**	**160**	**184**
2	**Deseño**	**40**	**44**
3	**Desenvolvemento**	**240**	**262**
3.1	**Sprint 1**	**80**	**80**
3.1.1	Inicio do proxecto	16	10
3.1.2	Xestión de usuarios	30	34
3.1.2.1	Servizos	10	10
3.1.2.2	*Front end*	20	24
3.1.3	Xestión alpendre	30	32
3.1.3.1	Servizos	10	12
3.1.3.2	*Front end*	20	20
3.1.4	Revisión	4	4
3.2	**Sprint 2**	**80**	**113**
3.2.1	Xestión das parcelas	30	65
3.2.1.1	Servizos	10	40
3.2.1.2	*Front end*	20	25
3.2.2	Xestión das tarefas	30	34
3.2.2.1	Servizos	10	10
3.2.2.2	*Front end*	20	24
3.2.3	Xerar informe estadístico	16	10
3.2.3.1	Servizos	4	2
3.2.3.2	*Front end*	12	8
3.2.4	Revisión	4	4
3.3	**Sprint 3**	**64**	**69**
3.3.1	Xestión de usuarios: recuperar contrasinal + info	30	56
3.3.1.1	Servizos	10	40
3.3.1.2	*Front end*	20	16
3.3.2	Xestión das parcelas e Xestión alpendre: borrar datos + obter info.	30	12
3.3.2.1	Servizos	10	4
3.3.2.2	*Front end*	20	8
3.3.3	Revisión	4	1
4	**Probas**	**8**	**6**
5	**Memoria**	**56**	**60**
	Total:	**504**	**556**

Figura 6.1: Planificación do proxecto.

2. Deseño: Neste punto trátase a elección do patrón de deseño, o deseño da UI e o deseño de marca. É unha parte indispensable para poder comezar co desenvolvemento da aplicación. A súa duración estimouse en 40 horas, tendo en conta que a maior parte delas se dedicarán ó deseño de marca dado que é unha tarefa moi laboriosa. Na realidade presentou unha demora de 4 horas, e isto ocorreu debido á complexidade na elección do logo final da aplicación.

3. Desenvolvemento: Neste punto trátase a construción da aplicación. Como se comentou, está dividido en 3 *Sprints* seguindo a organización das historias de usuario detallada

no *Sprint backlog*. A súa duración estimouse en 240 horas debido a envergadura das tarefas. Na realidade presentou unha demora de 22 horas. Isto foi debido ós contratempos ocasionados pola creación dun paquete para facer uso do servizo WMS do catastro no *Sprint 2* e, tamén, por cambio de versións dos paquetes de Firebase, que fixo necesario volver a construír os servizos de *CloudFirebase* e *AuthFirebase*.

4. Probas: Neste punto trátase a realización das probas finais sobre a aplicación xa completa. A súa duración estimouse en 8 horas e non houbo atrasos xa que as probas se realizaron con éxito.

5. Memoria: Neste punto trátase a realización da memoria do proxecto. A súa duración estimouse en 56 horas. Na realidade presentou unha demora de 4 horas, e isto ocorreu debido a subestimar o tempo de realización da mesma.

En resumo, e como se pode apreciar na figura 6.1, planificáronse inicialmente un total de 504 horas para completar o proxecto. Na práctica, produciuse unha demora de 52 horas, polo que o número real de horas subiu ata as 556 horas.

O custo do proxecto tamén se viu incrementado debido a esta demora. Para realizar unha estimación dos custos asociados ao desenvolvemento da aplicación, é necesario ter en conta tres tipos de recursos: humanos, materiais ou hardware e as ferramente software. Por unha parte, os recuros materiais e as ferramentas software usadas no desenvolvemento do proxecto teñen un custo cero, dado que os dispositivos hardware xa estaban dispoñibles e que se usaron para o software versións de proba, versións *Community* ou simplemente eran gratuítos. Por outro lado, os custos asociados aos soldos dos recursos humanos estimáronse en [41]:

- *Scrum Master* cun soldo de 2500€ mensuais (160 horas).
- Equipo de desenvolvemento cun soldo de 2000€ mensuais (160 horas)

Na figura 6.2, pódese apreciar unha comparativa entre as horas planificadas e as reais, xunto cos custos finais do desenvolvemento do proxecto.

Recursos humanos	Horas Previstas	Horas Reais	Custo Real
Scrum Master 1	12	9	140,62 €
Scrum Master 2	12	9	140,62 €
Desenvolvedor	504	556	6950 €
	Custo Total:		**7231,24 €**

Figura 6.2: Comparativa das horas planificadas e reais, e custo total do proxecto.

Capítulo 7

Deseño da aplicación

Neste capítulo abordarase o deseño de toda a aplicación, dende a UI e o deseño de marca ata o deseño da base de datos. Ambos aspectos axudarán a poder refinar a especificación de requisitos e tamén á construción da aplicación na fase de desenvolvemento.

A aplicación está deseñada seguindo o patrón Modelo-Vista-Controlador (MVC). Trátase dun patrón de arquitectura software que permite expresar como organizar e estruturar os compoñentes do sistema software e as relacións existentes entre cada un deles. Na figura 7.1, pode apreciarse como o MVC está organizado en tres capas diferentes:

- Modelo: É a capa onde se traballa cos datos, polo que conterá o mecanismo para acceder a información e tamén para actualizar o seu estado.

- Vista: É a capa que contén o código que vai producir a visualización das interfaces de usuario, é dicir, permite mostrar unha representación do estado do modelo.

- Controlador: É a capa que contén o código necesario para responder as accións que se solicitan na aplicación.

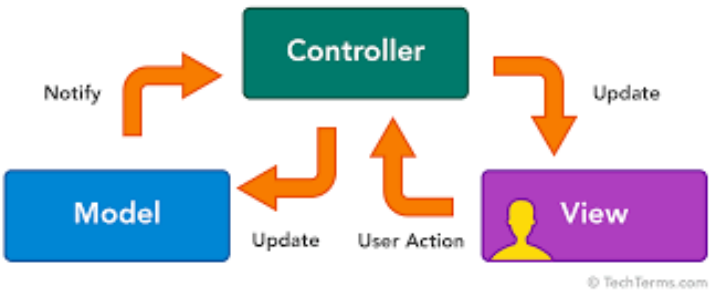

Figura 7.1: Modelo-Vista-Controlador.

7.1 Deseño da base de datos

Como se comentou anteriormente, para aloxar os datos da aplicación vanse utilizar as utilidades que proporciona Firebase. En concreto, a base de datos Cloud Firebase é unha base de datos NoSQL. Neste caso, esta base de datos é o que se chama unha base de datos documental, dado que está estruturada en documentos e coleccións de documentos. Polo tanto, o seu deseño ten que considerar que o seu uso vaise facer mediante un acceso ordeado a documentos. O deseño acadado foi o seguinte:

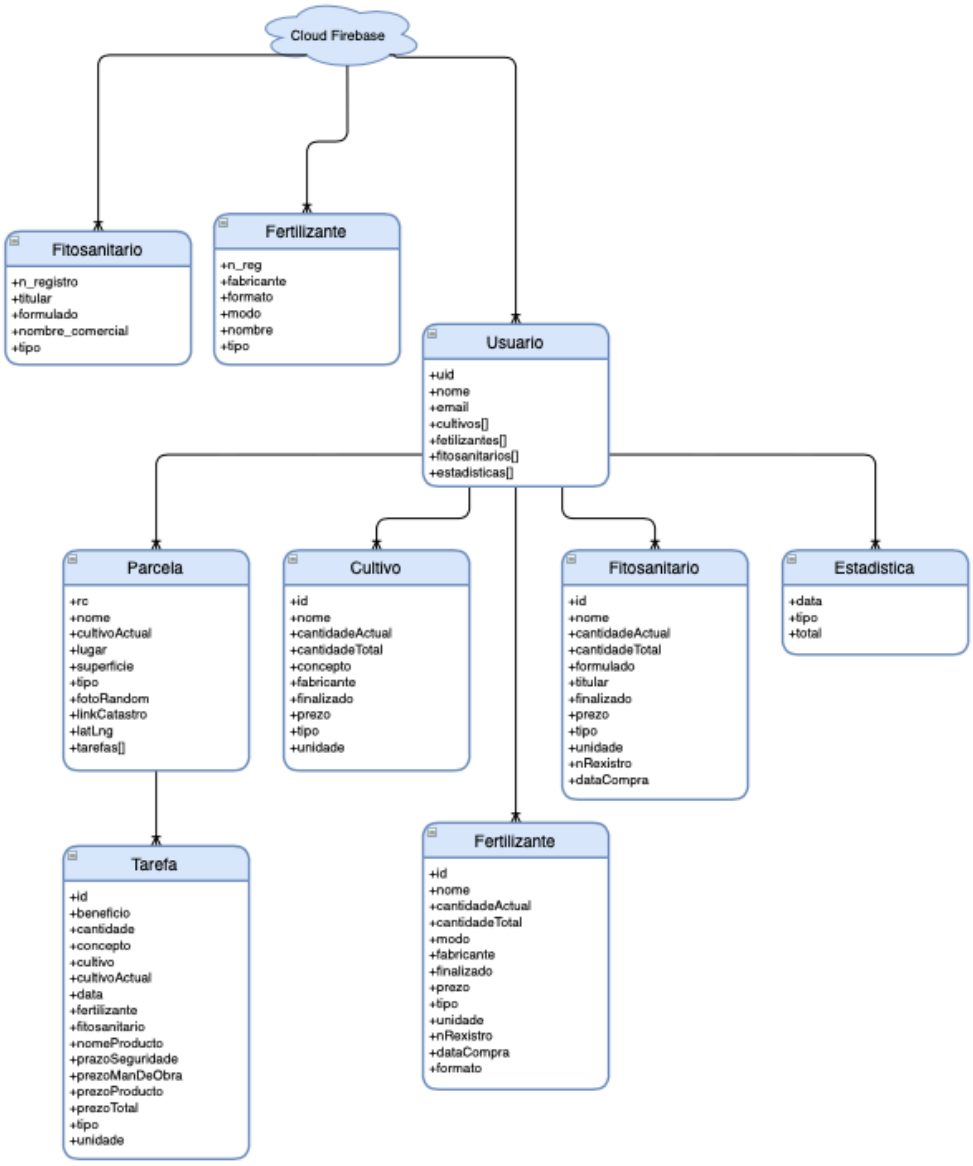

Figura 7.2: Deseño da base de datos en Cloud Firebase.

Na figura 7.2 pódese observar como o deseño consta de 3 entidades principais: Fitosanitario, Fertilizante e Usuario. As dúas primeiras gardan toda a información correspondente a este tipo de produtos que aparecen na listaxe de produtos legais en España, mentres que a terceira corresponde ó usuario da aplicación. Esta última contén as entidades Parcela, Cultivo, Fertilizante, Fitosanitario, Estadística e Tarefa.

7.2 Deseño de UI

O deseño da UI facilita obter unha perspectiva visual da aplicación, para así poder obter casos de uso que non se tiveron en conta ou arranxar os xa existentes. Tamén lle facilita ó desenvolvedor a tarefa de crear o UI definitivo e o manexo de estados.

Para este proxecto o deseño de UI foi realizado coa ferramenta Adobe XD [15], obtendo o seguinte resultado:

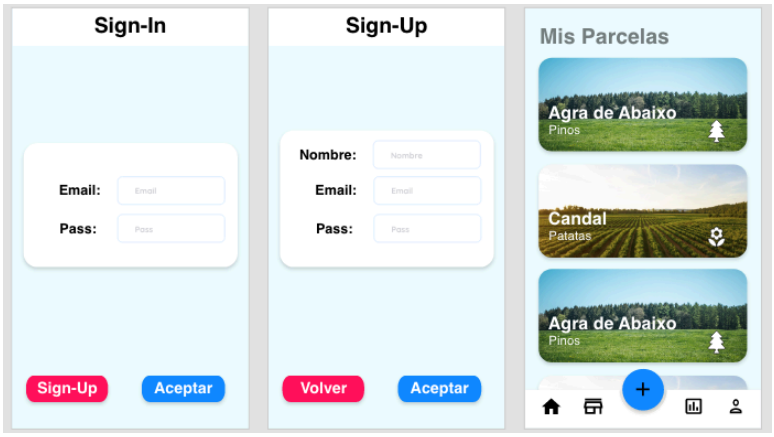

Figura 7.3: UI: Sign-In, Sign-Up e Inicio.

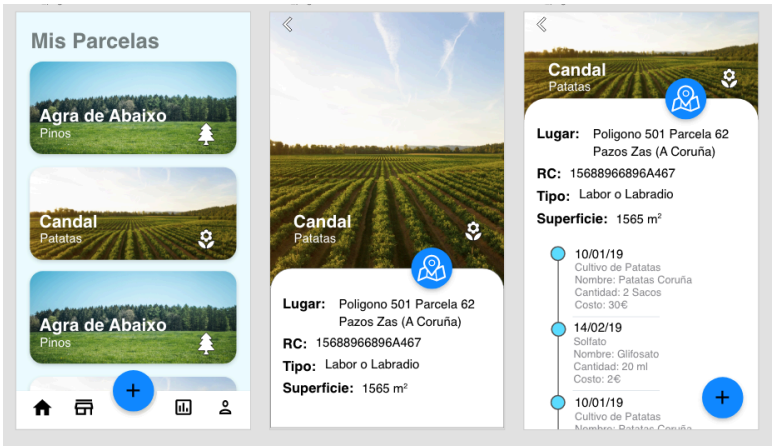

Figura 7.4: UI: Inicio, parcela e tarefas dunha parcela.

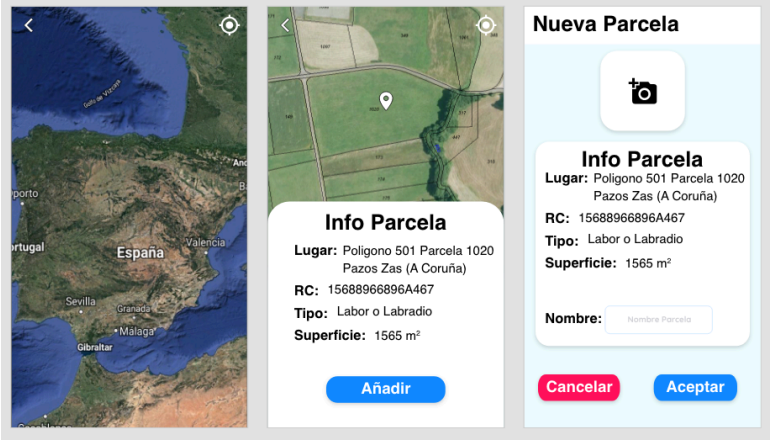

Figura 7.5: UI: Engadir unha parcela.

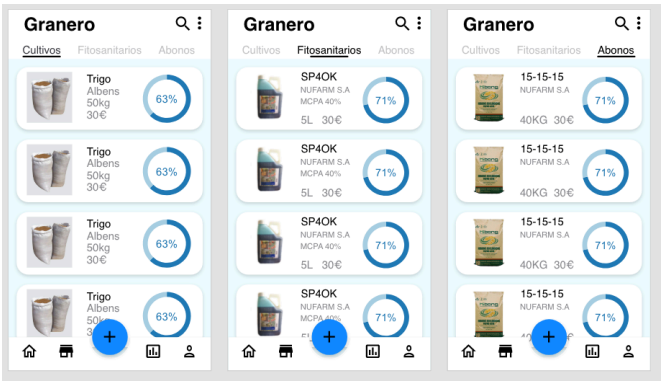

Figura 7.6: UI: Vista do Alpendre.

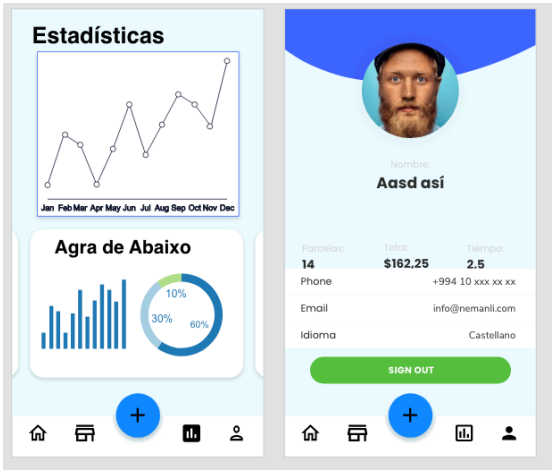

Figura 7.7: UI: Estadísticas e perfil.

7.3 Deseño de marca

Unha marca é unha promesa de beneficio para un cliente, unha palabra que o cliente identifica cunha calidade que lle reporta indirectamente un beneficio. Para calquera produto, o uso dunha marca é importante, pero para unha aplicación móbil é imprescindible. Unha marca ten que xerar confianza, e os clientes usan ou adquiren unha aplicación pola confianza que teñen de que será a mellor opción para resolver as súas necesidades. É por iso que unha marca ten que ofrecer unha imaxe única, transmitir uns valores inconfundibles e xerar unha vinculación emocional co cliente [42].

Para crear unha marca realízase un proceso de creación chamado *"Branding"*. Para levar a cabo este proceso, deberanse ter en conta os seguintes principios:

- Posicionamento.
- Personalidade.
- Valores.
- Sistema e marca.
- Identidade verbal.
- Identidade visual.

En definitiva, o *Branding* é un proceso de construción de lazos de confianza entre o produto e o público obxectivo [43].

Tendo en conta o contexto deste proxecto, a marca que se vai crear ten que ser moi representativa dos valores agrícolas e forestais, colores e símbolos que fagan referencia á natureza. A partir destas ideas, o nome elexido para a aplicación é *Agrotab*, que se pode ver como unha combinación de dúas palabras:

- Agro: referido ó mundo da agricultura e forestal.
- Tab: referido a palabra pestana en ingles, debido á forma de interactuar coa aplicación.

Ata este punto elaborouse a identidade verbal do produto que evoca unha representación dos seus valores. A continuación, ven unha das partes máis importantes da identidade visual, a necesidade de crear un logo que identifique de forma sinxela a marca e o produto. Para esta tarefa usáronse as ferramentas Adobe Photoshop e Adobe Ilustrator [17] [16]. Como se explica no proceso *Branding*, o deseño dun logo non é único, o normal é realizar unha cantidade diversa deles para obter os pros e contras de cada un ata acadar o definitivo. Na figura 7.8, poden verse algúns dos logos que se foron deseñando durante este proceso.

Figura 7.8: Exemplo de logos deseñados.

Despois dunha análise onde se tivo en conta a simpleza e que o logo ten que como funcionar nome para os títulos da aplicación e tamén como logo de icona, a elección final foi a seguinte (figura 7.9):

Figura 7.9: Logo definitivo.

O deseño é simple e está formado por unha combinación do nome e un símbolo. Como se pode apreciar, substituíuse a letra "o" por un símbolo que representa unha folla rodeando uns campos. Este símbolo central, e a substitución da letra, fai que o logo estea compensado de esquerda a dereita. Ademais, este símbolo pode ser usado sen problemas como logo da icona.

Capítulo 8

Desenvolvemento

Neste capítulo detallarase a grandes trazos os *sprints* realizados durante o desenvolvemento da aplicación.

8.1 Sprint 1

8.1.1 Inicio do proxecto

Comezouse este *sprint* coa creación dun repositorio de Git no GitLab da FIC. Este repositorio usarase para levar o control das versións orixinadas durante o desenvolvemento. O primeiro é crear a estrutura base de Flutter, por defecto Flutter crea unha app de exemplo a cal é un simple contador (figura 8.1).

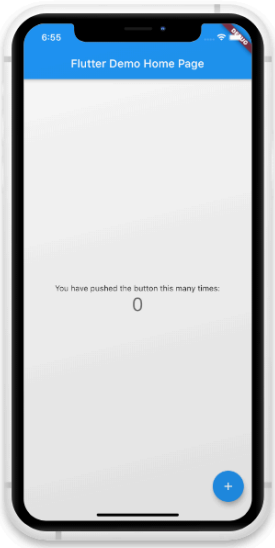

Figura 8.1: Exemplo app base de Flutter.

Como se pode apreciar na figura 8.2, Flutter crea unha estrutura de cartafoles que determina como se van organizar os ficheiros do proxecto. Sen contar os ficheiros ocultos, os máis destacables e cos que se traballa ó longo do desenvolvemento son:

- android: Cartafol que contén o código da app en Android. Só é preciso modificar estes ficheiros para realizar configuracións específicas para este sistema operativo, dado que Flutter non da soporte para facelo.

- build: Cartafol que contén a dinámica de Flutter e as traducións de código Dart a Kotlin para android e Swift para iOs. Este cartafol nunca se debe modificar, xa que o fai automaticamente Flutter en cada compilación do código.

- ios: Cartafol que contén o código da app para iOs. Só é preciso modificar estes ficheiros para realizar configuracións específicas para este sistema operativo, dado que Flutter non da soporte para facelo. É importante destacar que este cartafol só existirá se o desenvolvemento se fai nun equipo Mac.

- lib: Cartafol principal do proxecto posto que contén toda a estrutura de ficheiros que se utilizarán para o desenvolvemento do proxecto.

- pubspec.yaml: É a columna vertebral de Flutter xa que, dende este ficheiro, pódese configurar totalmente o comportamento da aplicación. Usarase para engadir ficheiros multimedia externos e para engadir os paquetes de terceiros que se usen no desenvolvemento.

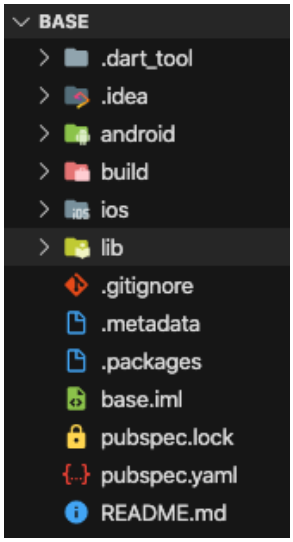

Figura 8.2: Disposición dos cartafoles en Flutter.

Inicialmente, o cartafol *lib* só contén o ficheiro *main.dart*. Para seguir unha metodoloxía de boas prácticas, engadirase un cartafol chamado *src* que conterá os seguintes elementos (figura 8.3):

- models: Cartafol que contén todos os modelos usados na aplicación.
- pages: Cartafol que contén as páxinas da aplicación.
- providers: Cartafol que contén os *providers* utilizados para o manexo de estado da aplicación.
- search: Cartafol que contén os buscadores de palabras usados na aplicación.
- services: Cartafol que contén todos os servizos usados na aplicacion.
- widgets: Cartafol que contén elementos gráficos reutilizados nas diferentes páxinas da aplicación.
- size_config.dart: Ficheiro que dispón os tamaños de pantalla do dispositivo onde se está executando a aplicación.

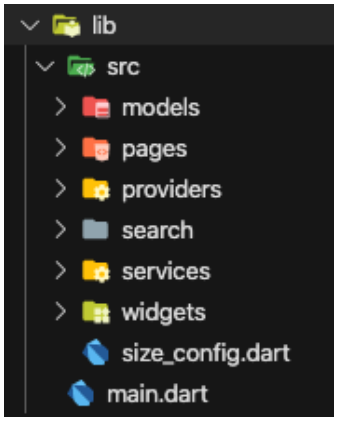

Figura 8.3: Disposición do cartafol "lib" aplicando boas prácticas en Flutter.

Para o manexo de estados da aplicación optouse por usar o paquete Provider. Este paquete serviranos para controlar e compartir un estado dun servizo, obxecto, etc. O seu funcionamento radica en crear unha clase denominada *Provider* na cal se define un obxecto modelo do que se vai a compartir. Este paquete dispón de diversas funcións útiles como: *ChangeNotifierProvider* que, simplemente, escoita cambios de *widgets* fillos que notifican ó pai, onde está creada esta función, e cando escoita un cambio volve a construir a páxina pero aplicando o novo estado. Para incorporar este paquete deberase introducir no ficheiro *pubspec.yaml* o nome do paquete e a versión do mesmo como se indica na documentación deste [44].

O seguinte paso é asociar a aplicación con Firebase. Para isto, deberase crear un proxecto na páxina de Firebase indicando os elementos a usar. Neste caso, os elemento de Firebase que se usaron foron Cloud Firestore, Cloud Storage e Authentication. Firebase non recoñece de forma automática un proxecto Flutter xa que, para Firebase, só existen apps de escritorio ou apps móbiles, e as apps móbiles interprétaas como Android ou iOs. Polo tanto, toda a configuración deberase de facer individualmente para cada plataforma. No directorio do proxecto creado en Firebase permite engadir cada app, polo que deberase de configurar cada unha delas aplicando estas configuracións no cartafol de Android e iOS do proxecto Flutter, respectivamente. Ditas configuracións están detalladas na páxina de Firebase completamente pero, para o caso de iOS, é recomendable abrir a carpeta iOS do proxecto Flutter directamente en xCode para poder aplicar de forma máis sinxela ditos cambios. Despois de realizar estas configuracións, deberemos de instalar en Flutter os seguintes paquetes:

- firebase_core [45],
- firebase_auth [46],
- cloud_firestore [47],
- cloud_storage [35].

Para isto realizarase o mesmo paso que co paquete anterior, é dicir, simplemente engádese o nome do paquete e a versión do mesmo no ficheiro *pubspec.yaml* do xeito no que se indica na documentación de cada un destes paquetes.

Tras realizar todas estas instalacións e configuracións, compilarase o código Flutter e executarase no simulador ou nun dispositivo con acceso a internet, considerando ambos sistemas (Android e iOS). Tras realizar este paso, na páxina do proxecto en Firebase, xa se recoñecerán as aplicacións (figura 8.4) e o proxecto Flutter quedará completamente configurado para comezar co desenvolvemento dos casos de uso.

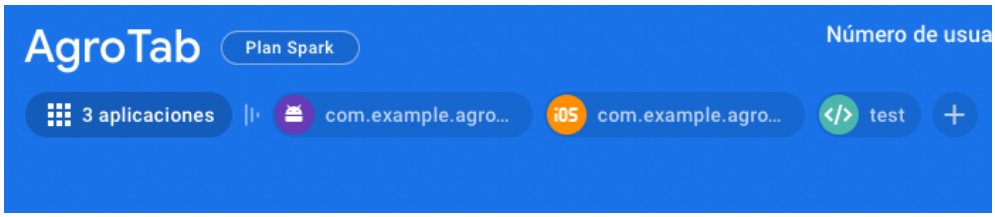

Figura 8.4: Apps engadidas no proxecto de Firebase.

8.1.2 Xestión de usuarios

En primeiro lugar, abordaranse os casos de uso *Sign-Up* (táboa 5.2), *Sign-In* (táboa 5.3) e *Sign-Out* (táboa 5.4), correspondentes á parte da xestión de usuarios. Comezouse polo caso

de uso *Sign-Up* no que se considera o rexistro dun novo usuario na aplicación. Na figura 8.5, móstrase un diagrama de fluxo no que se detallan as operación necesarias para implementar este caso de uso.

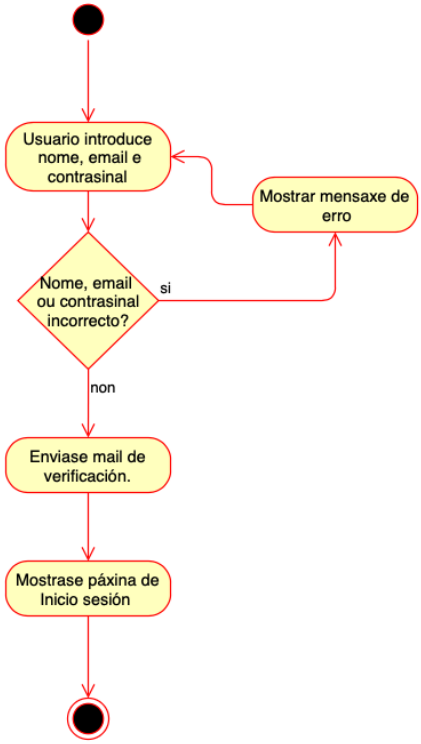

Figura 8.5: Diagrama de actividades: Sign-Up.

O primeiro paso é configurar o modo de rexistro en Firebase. Para isto, na páxina do proxecto en Firebase, no apartado de autenticación, indicamos o método "email e contrasinal" (figura 8.6), que será o usado no proxecto.

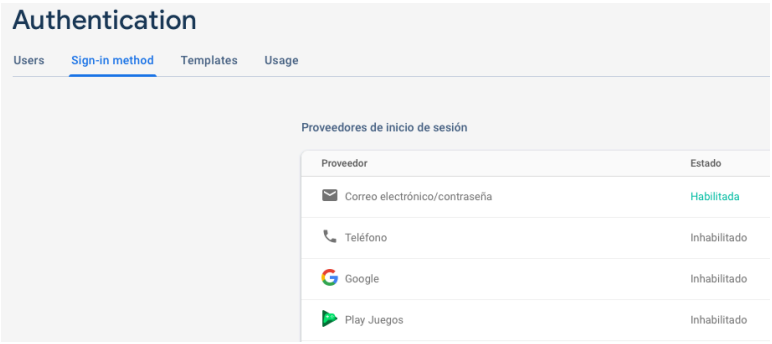

Figura 8.6: Firebase: Métodos de autenticación.

Cando un usuario inicie sesión no sistema, o servizo de autenticación deberá devolvernos un obxecto que poidamos compartir co manexador de estados a toda a aplicación para que, dese xeito, se poida acceder a esa información en calquera punto da mesma. Ademais, crearase no cartafol correspondente o modelo *UserModel*, cos seguintes atributos:

- uid: id que identifica a cada usuario e que se usará para localizar o documento onde se almacena toda a información do usuario gardada no Cloud Firebase.
- email: email co que o usuario se rexistrou na aplicación.
- emailVerificacion: booleano que indica se o usuario verificou o email para poder completar o rexistro.
- nombre: nome indicado polo usuario durante o rexistro.
- fechaAlta: data na que tivo lugar o rexistro do usuario.

Despois de realizar as configuración necesarias e a construción do modelo, o seguinte paso é crear o servizo dentro da app. Para isto, crearase o servizo *FirebaseAuthService* dentro do cartafol *services*. Para crear esta clase usaremos os paquetes instalados durante o inicio do proxecto, en concreto *firebase_auth* xa que, importando este paquete, poderemos crear unha instancia da clase *FirebaseAuth* que se usará para definir o comportamento deste servizo. Tanto as funcións de rexistro, como as de inicio de sesión desta clase, devolven un obxecto *User*. Este obxecto corresponde co usuario en Firebase, polo que, o primeiro que deberemos facer, é unha conversión deste obxecto ó creado anteriormente no modelo (figura 8.7).

```
UserModel _firebaseUserToUserModel(User firebaseUser) {
  return firebaseUser != null
      ? UserModel(
          firebaseUser.uid,
          firebaseUser.email,
          firebaseUser.emailVerified,
          firebaseUser.displayName,
          firebaseUser.metadata.creationTime)
      : null;
}
```

Figura 8.7: Servizo AuthFirebase: firebaseUserToUserModel.

É conveniente realizar esta conversión dado que o obxecto reportado por Firebase inclúe moitos atributos inútiles. Ademais, usar un modelo propio permite controlar ou engadir novos atributos moi útiles para o manexo de estados.

Unha vez creado este conversor, xa se pode implementar a función do servizo para rexistrar un novo usuario con email e contrasinal (figura 8.8). Nesta función, crearase o novo

usuario, engadirase o nome que este indicou e enviarase unha mensaxe ó email introducido polo usuario para a verificación do mesmo. Todas estas tarefas veñen incorporadas no paquete importado, e a configuración do mail de verificación poderemos realizala dende a páxina do proxecto en Firebase (figura 8.9). A función creada devolverá un obxecto *UserModel* no caso de que o rexistro sexa satisfactorio ou un *null* no caso de que ocorra algún erro durante o proceso de rexistro.

```dart
Future registerWithEmailAndPassword(
    String email, String password, String nombre) async {
  try {
    UserCredential result = await _auth.createUserWithEmailAndPassword(
        email: email, password: password);
    User firebaseUser = result.user;
    await firebaseUser.updateProfile(displayName: nombre);
    await firebaseUser.sendEmailVerification();
    return _firebaseUserToUserModel(_auth.currentUser);
  } catch (e) {
    print(e.toString());
    return null;
  }
}
```

Figura 8.8: Servizo AuthFirebase: registerWithEmailAndPassword.

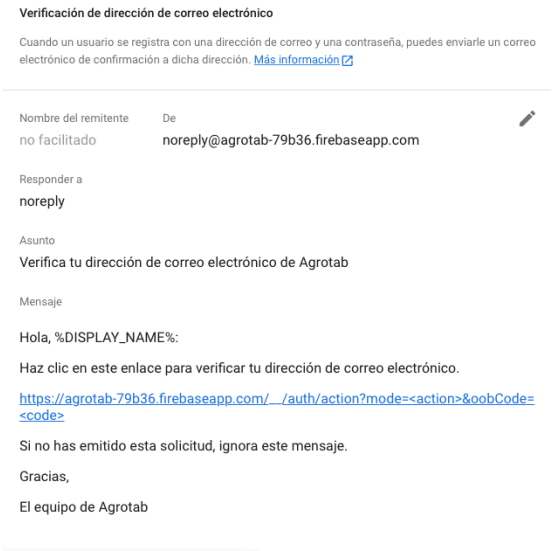

Figura 8.9: Exemplo da mensaxe enviada para verificar email.

Por último, créase o *front end* correspondente. Para isto, crearemos unha nova páxina denominada *RegisterPage* onde se detalla toda a parte visual, sempre apoiándonos no deseño de marca. Esta páxina deberemos incorporala como unha ruta para que o manexador de estados a administre dende o *main.dart*.

O seguinte caso de uso que se vai tratar é o de *Sign-In*. Este caso de uso permite o inicio de sesión dun usuario na aplicación. Na figura 8.10, amósase o diagrama de fluxo coas diferentes accións que é necesario levar a cabo.

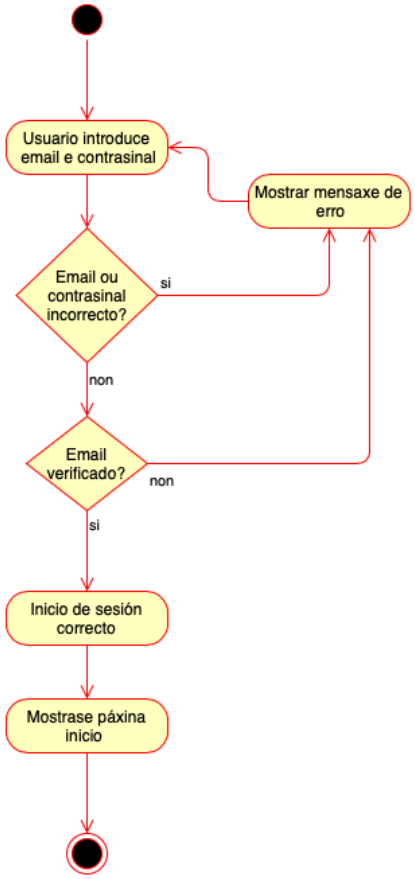

Figura 8.10: Diagrama de actividades: Sign-Up.

Seguindo a mesma filosofía que para o caso de uso anterior, créase a función dentro do servizo *FirebaseAuthService* que implementará a funcionalidade de iniciar sesión con email e contrasinal (figura 8.11). A función creada devolverá un obxecto *UserModel* no caso de que o inicio de sesión sexa satisfactorio ou, en caso de erro, devolverá o seguinte:

- *ERROR_WRONG_PASSWORD*: un erro que indica que o email ou o contrasinal introducido teñen algún erro na súa forma.

- *ERROR_USER_NOT_FOUND*: un erro que indica que o email introducido non corresponde con ningún usuario rexistrado.

- *ERROR_INVALID_EMAIL*: un erro que indica que o elemento introducido no campo do email non é valido xa que está rexistrado por outro usuario.

- *null*: calquera outro erro que poida ocorrer durante o proceso de inicio de sesión.

```
Future loginWithEmailAndPassword(String email, String password) async {
  try {
    UserCredential result = await _auth.signInWithEmailAndPassword(
        email: email, password: password);
    User firebaseUser = result.user;
    return _firebaseUserToUserModel(firebaseUser);
  } catch (e) {
    print(e.toString());
    if (e.toString().startsWith('PlatformException(ERROR_WRONG_PASSWORD')) {
      return 'ERROR_WRONG_PASSWORD';
    } else {
      if (e.toString().startsWith('PlatformException(ERROR_USER_NOT_FOUND')) {
        return 'ERROR_USER_NOT_FOUND';
      } else {
        if (e
            .toString()
            .startsWith('PlatformException(ERROR_INVALID_EMAIL')) {
          return 'ERROR_INVALID_EMAIL';
        } else {
          return null;
        }
      }
```

Figura 8.11: Servizo AuthFirebase: loginWithEmailAndPassword.

Por último, créase o *front end* correspondente para este caso de uso. Para iso, crearemos unha nova páxina denominada *LogInPage* onde detallaremos toda a parte visual sempre apoiándonos no deseño de marca. Esta páxina deberémola incorporar como unha ruta para que o manexador de estados a administre dende o *main.dart*. Durante o deseño da páxina poderanse controlar os erros que devolve a función anterior e mostrar as mensaxes pertinentes. Ademais, mediante o atributo *emailVerificado* do modelo *UserModel* que obtemos como resposta da función do servizo, poderemos controlar se o usuario verificou o email ou non e, en caso de non facelo, poderemos mostrar a correspondente mensaxe de erro.

O seguinte caso de uso é o de *Sign-Out*, no que se aborda o peche da sesión do usuario na aplicación. Na figura 8.12, pódese apreciar o correpondente diagrama coa secuencia de accións necesarias para implementar este caso de uso.

De novo, é preciso crear unha función no servizo *FirebaseAuthService* para implementar a funcionalidade que permite pechar a sesión do usuario (figura 8.13). A función creada non devolverá nada no caso de que o peche de sesión sexa satisfactorio pero, se algún erro se orixina durante este proceso, devolverá un *null*.

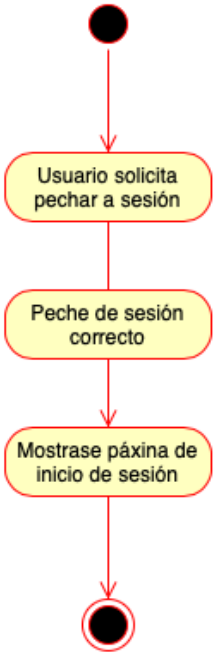

Figura 8.12: Diagrama de actividididades: Sign-Out.

```
Future logout() async {
  try {
    return await _auth.signOut();
  } catch (e) {
    print(e.toString());
    return null;
  }
}
```

Figura 8.13: Servizo AuthFirebase: loginWithEmailAndPassword.

Por último, créase o *front end* correspondente. Para iso, crearemos unha nova páxina denominada *PerfilPage* onde detallaremos toda a parte visual, sempre apoiándonos no deseño de marca. Esta páxina deberemos incorporala como unha ruta para que o manexador de estados a administre dende o *main.dart*.

O resultado visual do desenvolvemento destes casos de uso é o que se mostra na seguinte figura:

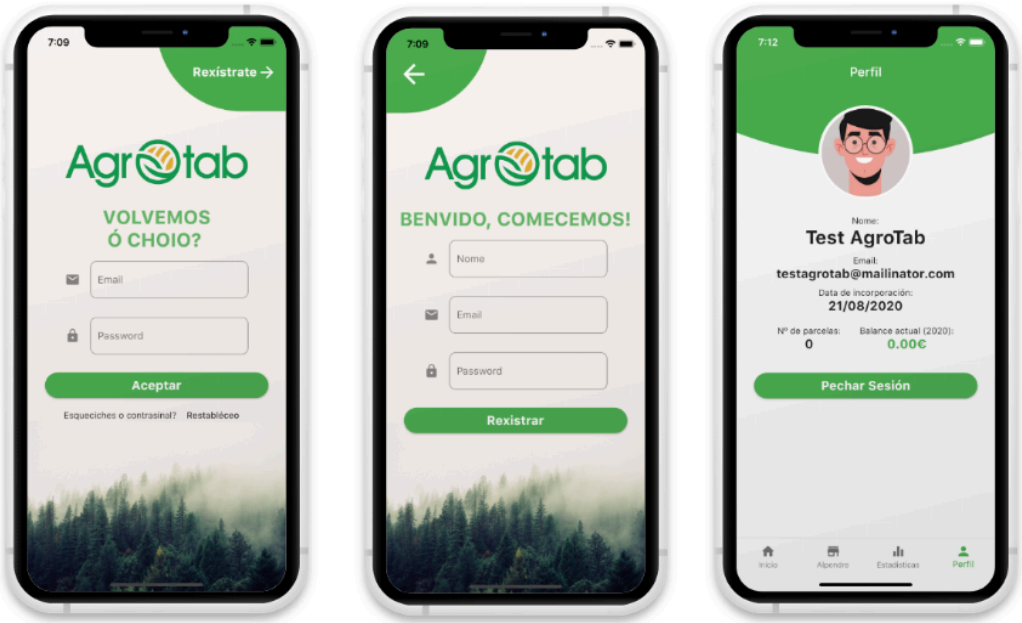

Figura 8.14: Resultado visual da xestión de usuarios.

8.1.3 Xestión do alpendre

Nesta sección, abordaranse os casos de uso *Visualizar produtos* (táboa 5.8), *Engadir produto tipo Fitosanitario ou Fertilizante* (táboa 5.9) e *Engadir produto tipo Cultivo* (táboa 5.10). O primeiro paso para implementar estos casos de uso foi o deseño da base de datos, posto que é preciso construír o servizo correspondente en Cloud Firebase.

Como se pode observar no deseño acadado na figura 7.2, a base de datos parte de tres coleccións de documentos principais:

- Usuarios: correspóndese coa información almacenada de cada usuario rexistrado na aplicación.
- Fitosanitarios: correspóndese coa lista de fitosanitarios rexistrados no ministerio de agricultura, os cales son os que se poden adquirir en España.
- Fertilizantes: correspóndese coa lista de fertilizantes rexistrados no ministerio de agricultura, os cales son os que se poden adquirir en España.

Neste punto preséntase un problema xa que o sistema de Cloud Firebase non dispón de ningún mecanismo para importar ficheiros .csv ou .json. Sen embargo, necesítanse importar os .json correspondentes ás listas con todos os produtos fitosanitarios e fertilizantes. Estes

datos son necesarios para que o usuario poida buscar na aplicación calquera produto e engadilo directamente ó seu alpendre virtual, dado que estes produtos son os únicos legais que o usuario podería empregar nas súas parecelas. Neste punto, pódese optar por dúas solucións:

- Engadir a man cada elemento individualmente.
- Construír unha aplicación que importe ficheiros .json a Cloud Firebase.

Dado que os dous ficheiros .json que se necesitan importar na base de datos teñen unha media de 2000 elementos cada un, a primeira opción queda totalmente descartada. Para crear a aplicación de carga de datos usaremos Node.js, que se deberá asociar a Firebase da mesma forma que se fixo no punto 8.1.2 aínda que, neste caso, é unha aplicación de escritorio. Despois de seguir os pasos que Firebase indica durante este proceso, descargarase un ficheiro *service_key.json* que conterá toda a información para a conexión coa base de datos. Este .json será usado polo *package* de Firebase en Node.js para realizar directamente a conexión coa base de datos do proxecto. Para engadir os documentos .json simplemente faise un bucle que lea todos os ficheiros que se encontren nun determinado cartafol e que, elemento a elemento, os engada na base de datos (figura 8.15).

```
fs.readdir(directoryPath, function(files) {

  files.forEach(function(file) {
    var lastDotIndex = file.lastIndexOf(".");
    var cartafol = require("./files/" + file);

    cartafol.forEach(function(obj) {
      firebase
        .collection(file.substring(0, lastDotIndex))
        .doc(obj.itemID)
        .set(obj)
        .then(function(docRef) {
          console.log("Documento engadido");
        })
        .catch(function(error) {
          console.error("Error: ", error);
        });
    });
  });
});
```

Figura 8.15: Node.js: Bucle para engadir .json a Cloud Firebase.

Despois de engadir os datos, comezamos co desenvolvemento dos casos de uso correspondentes á xestión do alpendre nos que se trata de visualizar os diferentes produtos gardados e engadir novos produtos. Os produtos que se gardarán no alpendre virtual poden ser de 3 tipos: fitosanitarios, fertilizantes e cultivos. Nos dous primeiros, o usuario tan só poderá engadir aqueles que estean nos .json que anteriormente se importaron a Cloud Firebase coa

aplicación creada en *Node.js*. No caso dos cultivos, o usuario poderá engadir calquera produto, polo que se lle mostrará un formulario cos campos necesarios para engadir o cultivo. Na figura 8.16, móstrase o diagrama coa secuencia de accións necesarias para levar a cabo os casos de uso relacionados coa xestión do alpendre.

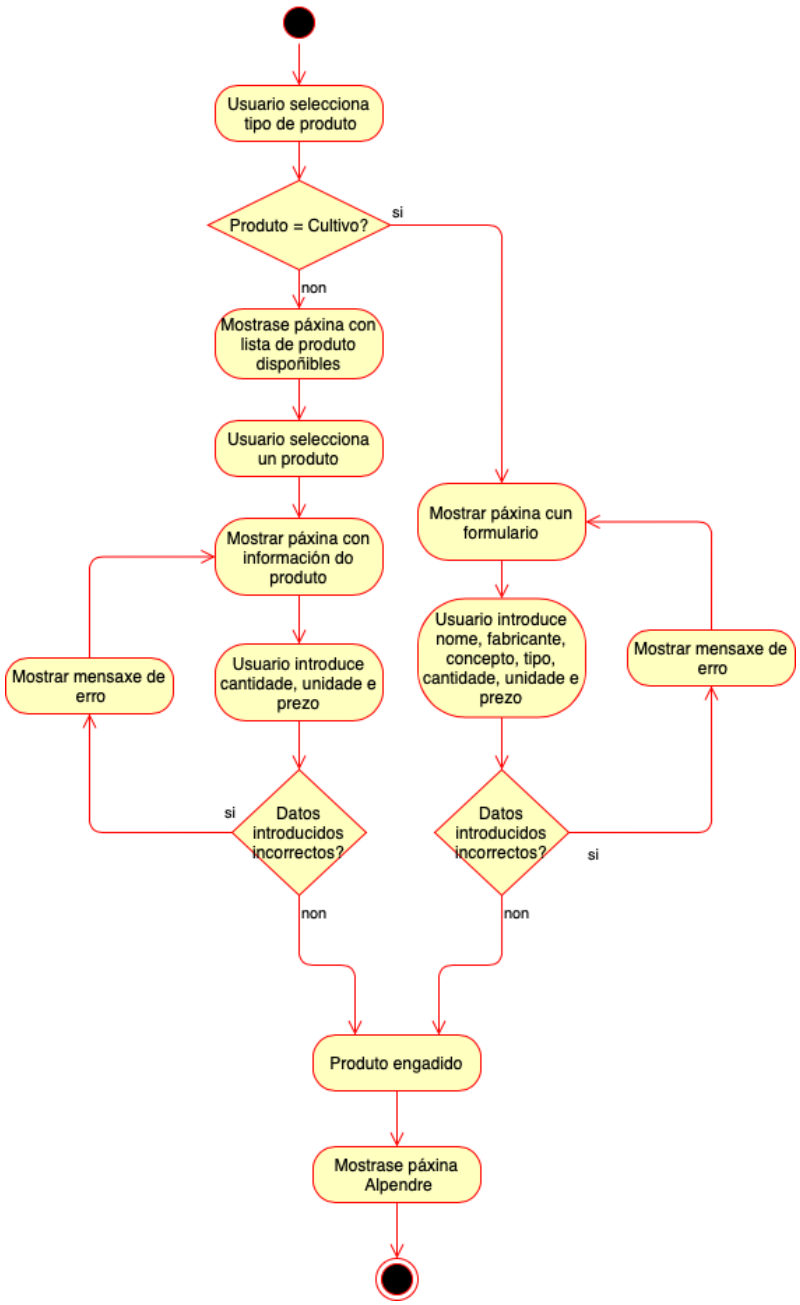

Figura 8.16: Diagrama de actividades: Xestión do alpendre.

Cando un novo produto se engada ó alpendre, o servizo deberá devolvernos un obxecto

que poidamos compartir co manexador de estados a toda a aplicación. Por tanto, temos que crear no cartafol correspondente os modelos *FertilizanteModel*, *FitosanitarioModel* e *CultivoModel*, cada un dos cales terá os seus correpondentes atributos:

- *FertilizanteModel*: nome, fabricante, formato, modo, tipo, cantidadeTotal, cantidadeActual, finalizado, id, prezo, nRexistro, unidade e dataCompra.
- *FitosanitarioModel*: nome, formulado, titular, cantidadeTotal, cantidadeActual, finalizado, id, prezo, nRexistro, unidade e dataCompra.
- *CultivoModel*: nome, fabricante, concepto, tipo, cantidadeTotal, cantidadeActual, finalizado, id, prezo, nRexistro e unidade.

Despois da construción dos modelos, o seguinte paso é crear o servizo dentro da app. Para isto, crearase o servizo *FirebaseCloudService* dentro do cartafol *services*. Para crear esta clase usaremos os paquetes instalados durante o inicio do proxecto, en concreto *cloud_firestore* xa que, importando este paquete, poderemos crear unha instancia da clase *FirebaseFirestore* que usaremos para definir o comportamento deste servizo. Ademais, implementaranse tres funcións dentro do servizo: *addFitosanitario*, *addFertilizante* e *addCultivo*. Estas funcións reciben como parámetro o identificador do usuario e o modelo do produto e engádeno na base de datos. Como podemos ver na imaxe 8.17, é preciso crear un novo documento ó que se lle engaden os atributos do obxecto do modelo que se pasa como parámetro.

```
Future<void> addCultivo(String uid, CultivoModel cultivo) async {
  return await _database.doc('usuarios/$uid').collection('cultivos').add({
    'nombre': cultivo.nombre,
    'fabricante': cultivo.fabricante,
    'concepto': cultivo.concepto,
    'tipo': cultivo.tipo,
    'cantidadTotal': cultivo.cantidadTotal.toString(),
    'cantidadActual': cultivo.cantidadActual.toString(),
    'finalizado': cultivo.finalizado,
    'precio': cultivo.precio.toString(),
    'unidade': cultivo.unidade,
  }).then((value) {
    value
        .set({'id': value.id}, SetOptions(merge: true))
        .then((value) => print('Cultivo añadido'))
        .catchError((error) => print(error));
  }).catchError((error) => print(error));
}
```

Figura 8.17: Servizo CloudFirebase: Exemplo de función para engadir un elemento a base de datos.

Para recuperar os datos dos produtos da base de datos, decidiuse usar as utilidades de *Streams*. Para iso, créanse as funcións *getFitosanitario*, *getFertilizante* e *getCultivo* tal e como se mostra na figura 8.18.

```dart
Stream<QuerySnapshot> getFitosanitarios(String uid) {
  return _database.collection('usuarios/$uid/fitosanitarios').snapshots();
}

Stream<QuerySnapshot> getFertilizantes(String uid) {
  return _database.collection('usuarios/$uid/fertilizantes').snapshots();
}

Stream<QuerySnapshot> getCultivos(String uid) {
  return _database.collection('usuarios/$uid/cultivos').snapshots();
}
```

Figura 8.18: Servizo CloudFirebase: Exemplo de función para obter streams de documentos da base de datos.

Grazas ó uso de *Streams*, os datos mostrados sempre van estar actualizados, dado que Flutter proporciona unha función denominada *StreamBuilder* que axuda a crear unha páxina dinámica. Esta función escoita os cambios do *stream* correspondente e, cando detecta algún cambio, sobrescribe a páxina cargando agora os novos datos recibidos. Desta forma, cando un usuario estea nunha determinada páxina e suceda algún cambio, automaticamente se verá reflexado.

Por último, créase todo o *front end* necesario para visualizar e engadir os diferentes tipos de produtos. Para iso, crearemos unha nova páxina denominada *GraneroPage* onde detallaremos toda a parte visual, sempre apoiándonos no deseño de marca. Esta páxina deberemos incorporala como unha ruta para que o manexador de estados a administre dende o *main.dart*. Durante o deseño da páxina poderanse controlar os erros que poidan acontecer e mostrar as mensaxes pertinentes. Ademais, facendo uso das funcións predefinidas en Flutter, crearanse buscadores para poder filtrar e seleccionar o produto sen ter que buscalo por toda a lista.

O resultado visual do desenvolvemento destes casos de uso pode verse na figura 8.19.

8.2 Sprint 2

8.2.1 Xestionar parcelas

Para comezar co segundo *sprint* do proxecto, abordáronse os casos de uso *Visualizar parcelas* (táboa 5.6) e *Engadir parcela* (táboa 5.7). O obxectivo é poder xeolocalizar o dispositivo no que se está executando a aplicación e obter información catastral sobre a parcela na que estamos para que, en caso de interese, esta se poida engadir ó rexistro de parcelas da aplicación. Para poder saber dunha forma máis visual a parcela sobre a que estamos, o mellor é debuxar as contornas das parcelas sobre un mapa satélite. Para esta tarefa, o catastro dispón dun sistema Web Map Service (WMS) que entrega imaxes coa distribución parcelaria de toda España excepto País Vasco e Navarra [48].

Neste punto, atopámonos cun novo problema, xa que Flutter recomenda o uso do paquete

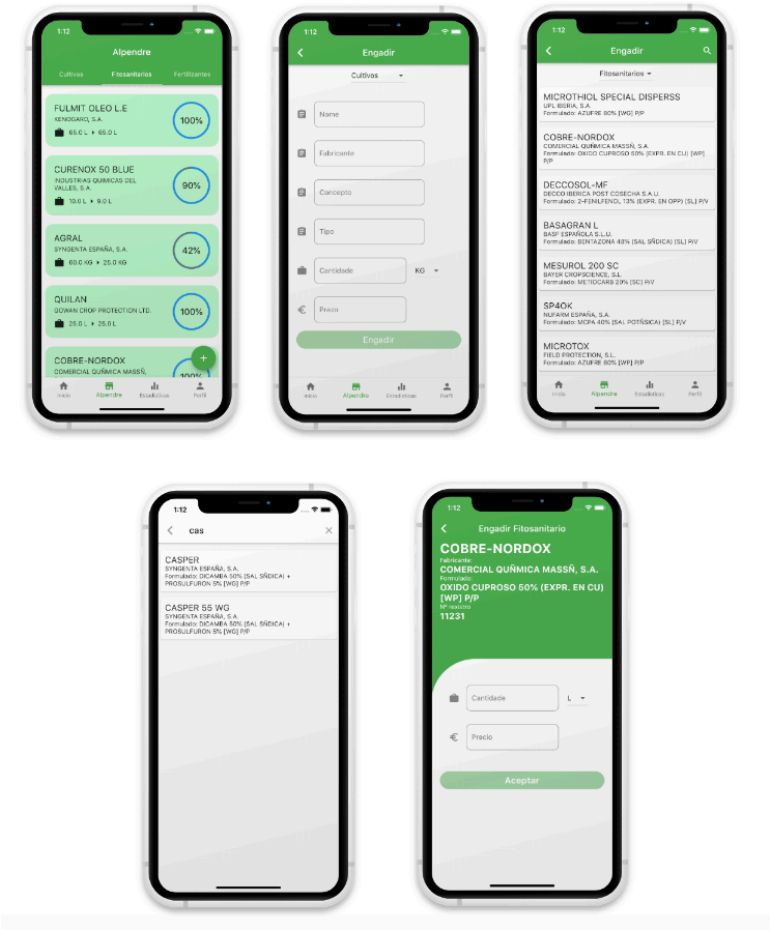

Figura 8.19: Resultado visual da xestión do alpendre.

google_maps para o manexo de mapas nunha aplicación, o cal é unha librería baseada en *Google Maps JavaScript API v3* . Sen embargo, debido a xuventude de Flutter, esta librería non incopora todas as funcionalidades de *Google Maps JavaScript API v3* como, por exemplo, soporte para WMS. Para solucionar este problema, consideráronse 3 posibles solucións:

- Usar un paquete externo con soporte para WMS.
- Crear código html con *Google Maps JavaScript API v3* para dar soporte WMS mediante as funcións javascript.
- Crear un paquete de soporte WMS mediante o uso de diversos paquetes externos para Flutter.

Durante esta fase do proxecto, probáronse as tres solucións. A primeira alternativa non funcionou para a versión WMS do sistema WMSInspire de catastro. A segunda foi funcional pero o seu rendemento era limitado é facía que a súa usabilidade fose nula. Por iso, decidiuse optar

pola terceira solución xa que, aínda que é a máis laboriosa, o resultado cumpre os requisitos funcionais e de rendemento que se espera dunha aplicación de este estilo.

Para poder crear esta solución, primeiro analizouse o servizo WMSInspire de catastro. Os servizos WMS dispoñen de tres funcións principais:

- GetCapabilities: devolve un ficheiro xml coa estrutura do servizo e as características do mesmo.
- GetMaps: devolve imaxes en diversos formatos que se despregan como unha capa ata formar un mapa.
- GetFeatureINFO: devolve información catastral da posición que se indica.

Facendo uso do paquete *flutter_map*, que permite cargar capas de mapas e superpoñer unha por encima doutra, conseguiuse crear un mapa cunha capa de google maps satélite como base é as imaxes procedentes da función *GetMaps* como *overlay* (figura 8.20). Ademais, coa axuda do paquete *Location*, pódese centrar o mapa creado referenciando unha posición en coordenadas (Latitude-Lonxitude). Polo tanto, para desenvolver esta funcionalidade deberemos instalar en Flutter os seguintes paquetes:

- flutter_map [49],
- location [50].

Para facer isto, indícase o nome do paquete e a versión do mesmo no ficheiro *pubspec.yaml* seguindo as indicacións que aparecen na documentación de cada un destes paquetes.

```
layers: [
  TileLayerOptions(
    urlTemplate:
        'http://{s}.google.com/vt/lyrs=s&x={x}&y={y}&z={z}',
    subdomains: ['mt0', 'mt1', 'mt2', 'mt3'],
    tileProvider: NonCachingNetworkTileProvider(),
  ), // TileLayerOptions
  TileLayerOptions(
    opacity: 0.5,
    wmsOptions: WMSTileLayerOptions(
      transparent: true,
      baseUrl:
          'http://ovc.catastro.meh.es/cartografia/INSPIRE/spadgcwms.aspx?',
      format: 'image/png',
      layers: ['cp.cadastralparcel'],
      styles: ['cp.cadastralparcel.default'],
    ), // WMSTileLayerOptions
    maxZoom: 18.0,
    minZoom: 15.0,
  ), // TileLayerOptions
```

Figura 8.20: Exemplo da disposición de capas en *flutter_map*.

Unha vez se localiza unha parcela sobre o mapa creado coas diferentes capas, pódese obter información da parcela coa función GetFeatureINFO, que ten como parámetros de entrada as coordenadas por cuadrantes (e.g.: oeste= -18.5, este= 5.3, sur = 26.2, norte= 44.8). Para pasar das coordenadas estándar a este tipo de coordenadas, o paquete *Location* proporciona un conversor que, básicamente, habilita unha petición GET que devolve un xml con esa información.

Para manexar todo isto, creamos un servizo chamado *CatastroService* no cartafol *services*. Neste servizo crearase unha función denominada *getRCParcelaMedianteLatLng* na que, mediante varias peticións GET, obterase a información dunha parcela. En primeiro lugar, obterase o xml da función GetFeatureINFO coa referencia catastral da parcela e, posteriormente, obterase superficie, lugar e ligazón á ficha da parcela en catastro. Estes datos empaquétanse nun mapa de Strings e devólvese á vista do mapa para que este mostre a parcela e a información da mesmas. Un sistema WMS retorna a información dinamicamente polo que, para emular ese comportamento, deberase crear un sistema de marcador para que recoñeza a pulsación da pantalla e devolva as coordenadas dese punto. Isto faise usando o paquete *Location* coa axuda do manexador de estados *Provider*. A función que se ten que crear debe permitir que, cada vez que o usuario pulse a pantalla do seu dispositivo, se active o marcador no punto exacto onde o usuario elixiu, e logo, mediante o manexador de estados, pódese volver a construír a interface para que o cambio sexa instantáneo e se mostre a información da parcela seleccionada. Ademais, creouse outra variante desta función que substitúe a pulsación da pantalla polo movemento GPS do dispositivo polo que, cando o dispositivo desprace, a aplicación devolverá a posición actual no mapa e a información da parcela na que se encontre.

Despois de solucionar toda a problemática co manexo de mapas en Flutter, comezamos co desenvolvemento dos casos de uso mencionados. Nestes casos de uso, abordarase a xestión das parcelas do usuario de xeito que se poida visualizar as parcelas gardadas e engadir parcelas novas. Na figura 8.21, móstrase o correspondente diagrama de fluxo coa secuencia de accións necesarias para implementar todo o relacionado coa xestión das parcelas.

Cando se engade unha nova parcela, o servizo correspondente deberá devolvernos un obxecto que poidamos compartir co manexador de estados a toda a aplicación. Esto permite que se poida acceder a información do obxecto creado desde calquera punto da aplicación. Polo tanto, o seguinte paso consiste en crear o modelo *ParcelaModel* no cartafol corrrespondente. Os atributos que se definen para este modelo son: nome, lugar, rc, tipo, superficie, cutivoActual, fotoRandom, linkCatastro e latLng.

Despois da construción do modelo, o seguinte paso é crear o servizo dentro da app encargado de engadir novas parcelas. Para isto, incorpórase a función *addParcela* no servizo *FirebaseCloudService*. Esta función recibe como parámetro o identificador do usuario e o modelo da parcela e, como podemos apreciar na imaxe 8.22, crea un novo documento ao que se lle engaden os atributos do modelo que recibe como parámetro.

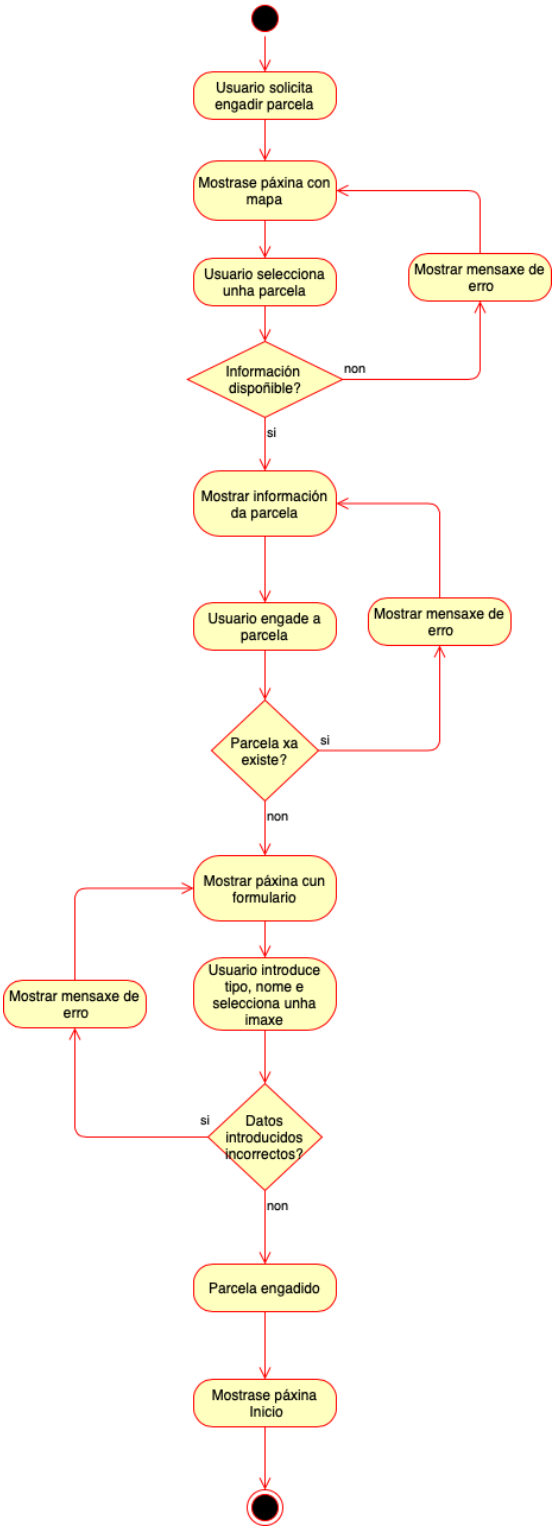

Figura 8.21: Diagrama de actividades: Xestión parcelas.

```dart
Future<void> addParcela(String uid, ParcelaModel parcela) async {
  return await _database
      .doc('usuarios/$uid')
      .collection('parcelas')
      .doc(parcela.rc)
      .set({
        'nombre': parcela.nombre,
        'lugar': parcela.lugar,
        'rc': parcela.rc,
        'tipo': parcela.tipo,
        'superficie': parcela.superficie.toString(),
        'cultivoActual': parcela.cultivoActual,
        'fotoRandom': parcela.fotoRandom,
        'linkCatastro': parcela.linkCatastro,
        'latLng': parcela.latLng
      })
      .then((value) => print('parcela añadida'))
      .catchError((error) => print(error));
}
```

Figura 8.22: Servizo CloudFirebase: Función engadir parcela.

Para obter as parcelas da base de datos, úsase de novo *Streams*. En concreto, créase a función *getParcelas* que recupera todo a información para unha determinada parcela:

```dart
Stream<QuerySnapshot> getParcelas(String uid) {
  return _database.collection('usuarios/$uid/parcelas').snapshots();
}
```

Por último, créase toda a parte do *front end* necesaria para que o usuario poida visualizar os datos da súas parcelas e engadir novas parcelas na súa conta. Para iso, crearemos unhas novas páxina denominadas *InicioPage* e *MapPage* onde detallaremos toda a parte visual, sempre sendo fieis ao deseño de marca. Estas páxinas deberemos incorporalas como unha ruta para que o manexador de estados as administre dende o *main.dart*. Ademais, o usuario poderá dispoñer da vista correspondente ao mapa coas diferentes capas que foi creado facendo uso do paquete e do servizo que se implementou para solucionar a falta de soporte WMS. Tamén, engadíronse aspectos visuais que melloran a usabilidade, como un filtro por cultivo para mostrar as parcelas e un xerador aleatorio de imaxes en función do tipo de parcela. O resultado visual do desenvolvemento destes casos de uso móstrase na figura 8.23.

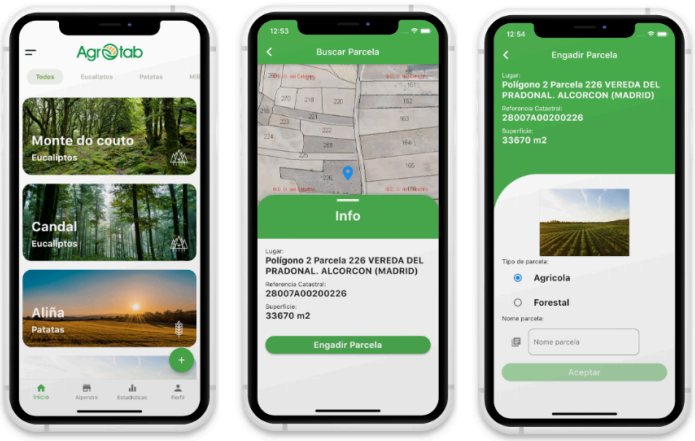

Figura 8.23: Resultado visual da xestión das parcelas.

8.2.2 Xestión das tarefas

A continuación, abordaranse os casos de uso *Visualizar tarefas dunha parcela* (táboa 5.13) e *Engadir tarefa a unha parcela* (táboa 5.14), co obxectivo de permitir que o usuario poida engadir unha nova tarefa definindo o tipo, data, cantidade e prezo da mesma. Na figura 8.24, móstrase o diagrama coas accións necesarias para implementar todo o relacionado coas tarefas.

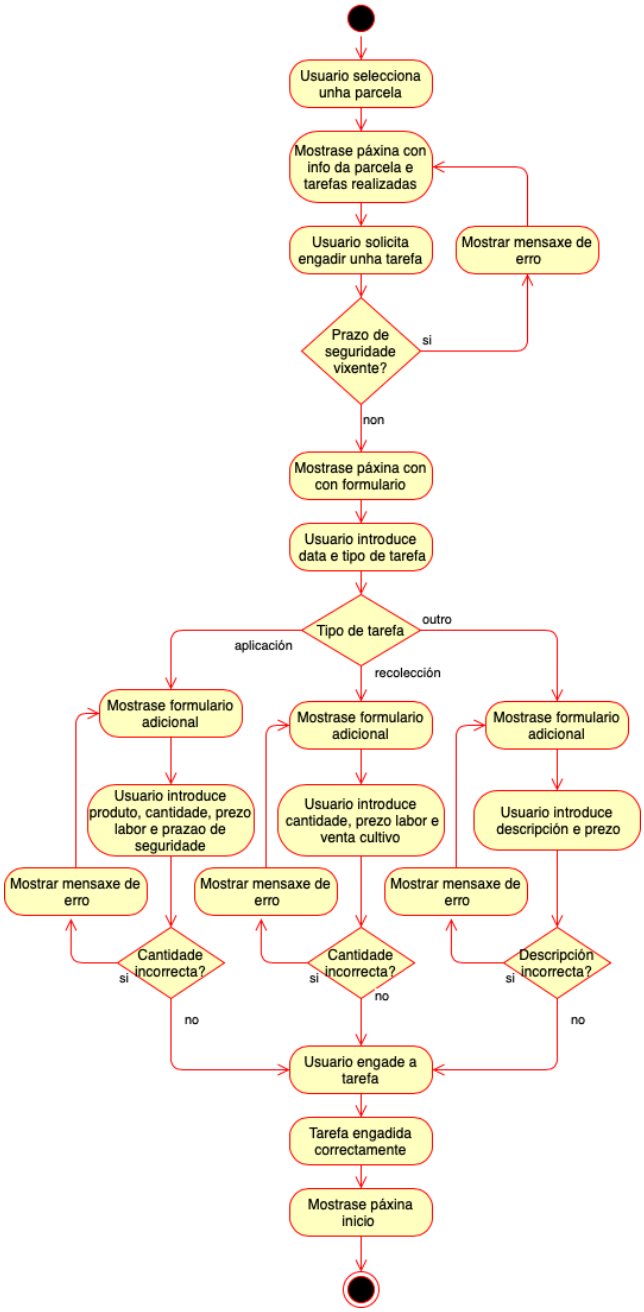

Figura 8.24: Diagrama de actividades: Xestión das tarefas.

Cando se engade unha nova tarefa, o servizo correspondente deberá devolvernos un obxecto que poidamos compartir co manexador de estados a toda a aplicación. Polo tanto, é necesario crear o modelo *TareaModel* no cartafol correspondente. Os atributos que forman parte do modelo para as tarefas son: data, tipo, concepto, fitosanitario, cultivo, fertilizante, prezoManDeObra, prezoProduto, prezoTotal, beneficio, cantidade, cultivoActual e prazoSeguridade.

Despois da construción do modelo, o seguinte paso é crear o servizo dentro da app. Para isto, engádese a función *addTarea* no servizo *FirebaseCloudService*. Esta función recibe como parámetro o identificador do usuario, a referencia catastral da parcela e o modelo da parcela e, como podemos ver na imaxe 8.25, crea un novo documento na base de datos ao que se lle engaden os atributos do modelo que se pasa como parámetro.

```
Future<void> addTarea(String uid, String rc, TareaModel tarea) async {
  return await _database
      .doc('usuarios/$uid/parcelas/$rc')
      .collection('tareas')
      .add({
    'fecha': Timestamp.fromDate(tarea.fecha),
    'tipo': tarea.tipo,
    'concepto': tarea.concepto,
    'cultivoActual': tarea.cultivoActual,
    'fitosanitario': tarea.fitosanitario == null
        ? ''
        : _database
            .doc('usuarios/$uid/fitosanitarios/${tarea.fitosanitario.id}'),
    'cultivo': tarea.cultivo == null
        ? ''
        : _database.doc('usuarios/$uid/cultivos/${tarea.cultivo.id}'),
    'fertilizante': tarea.fertilizante == null
        ? ''
        : _database
            .doc('usuarios/$uid/fertilizantes/${tarea.fertilizante.id}'),
    'precioManoDeObra': tarea.precioManoDeObra.toString(),
    'prazoSeguridade': tarea.prazoSeguridade,
    'precioProducto': tarea.precioProducto.toString(),
    'precioTotal': tarea.precioTotal.toString(),
    'beneficio': tarea.beneficio,
    'cantidade': tarea.cantidade
  }).then((value) {
    value
        .set({'id': value.id}, SetOptions(merge: true))
        .then((value) => print('Tarea añadida'))
        .catchError((error) => print(error));
```

Figura 8.25: Servizo CloudFirebase: Función engadir tarefa.

O usuario só poderá elixir para a tarefa os produtos que teña engadidos no alpendre polo que, cando unha tarefa faga uso de certa cantidade dun produto, hai que propagar o cambio para que se vexa reflexado no rexistro do alpendre virtual. Ademais, é preciso comprobar que haxa no alpendre suficiente cantidade do produto que se vai aplicar. Para conseguir isto, engadíronse na función anterior certas condicións para actualizar a cantidade dun produto en función do seu tipo (figura 8.26).

```
if (tarea.fertilizante != null) {
  DocumentReference documento = _database
      .doc('usuarios/$uid/fertilizantes/${tarea.fertilizante.id}');
  documento.get().then((value2) {
    value.set({
      'nombreProducto': value2.data()['nombre'],
      'unidade': value2.data()['unidade']
    }, SetOptions(merge: true)).then((value) => print('Nombre añadido'));
    documento.update({
      'cantidadActual': (double.parse(value2.data()['cantidadActual']) -
              tarea.cantidade)
          .toString()
    }).then((value) => print('cantidade actualizada'));
  });
}
```

Figura 8.26: Exemplo de actulización da cantidade dun produto.

Para obter a información das tarefas da base de datos, creouse a función *getTareas* (figura 8.30), que fai uso de *Streams*. Esta función recibe como parámetro o identificador do usario e da parcela correspondente da que se quere obter a información.

```
Stream<QuerySnapshot> getTareas(String uid, String rc) {
  return _database
      .collection('usuarios/$uid/parcelas/$rc/tareas')
      .orderBy('fecha')
      .snapshots();
}
```

Figura 8.27: Servizo CloudFirebase: Función para obter as tarefas da base de datos.

Por último, impleméntase toda a parte do *front end* correspondente para que o usuario poida engadir novas tarefas e visualizar a información destas. Para iso, crearemos unhas novas páxinas denominadas *ParcelaPage* e *TareaPage* onde se detalla toda a parte visual das diferentes pantallas, sempre apoiándonos no deseño de marca. Como de costume, estas páxinas deberemos incorporalas como unha ruta para que o manexador de estados as administre dende o *main.dart*. Durante o deseño das páxinas, poderanse controlar os diferentes erros e mostrar as mensaxes pertinentes. Ademais, controlarase o prazo de seguridade para unha parcela calculando a data de vencemento asociada as súas tarefas e impedindo engadir novas tarefas a esa parcela se o prazo aínda está vixente.

O resultado visual do desenvolvemento destes casos de uso pode apreciarse en detalle na figura 8.31. Como se pode ver na imaxe da dereita, no caso de querer engadir unha tarefa sobre unha parcela que ten vixente algún prazo de seguridade dun tratamento anterior, a aplicación amosa un erro indicando esta circunstancia.

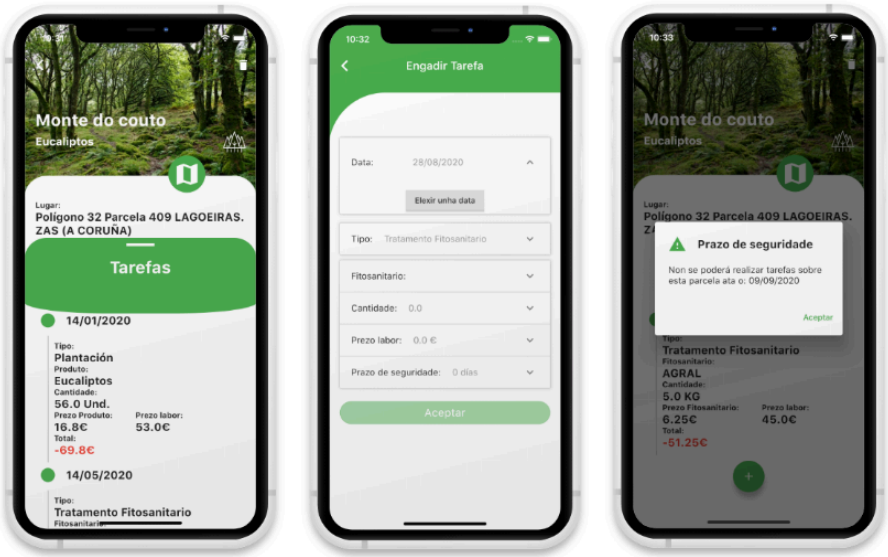

Figura 8.28: Resultado visual da xestión das tarefas.

8.2.3 Informe estadístico

No final deste segundo *sprint*, abordarase o caso de uso *Obter estadísticas financeiras* (táboa 5.21). A súa finalidade é xerar un informe estadístico para o usuario sobre o estado financeiro das súas explotacións no ano actual. O informe está composto por tres partes:

- Gráfica lineal que representa a evolución de beneficios fronte a gastos.
- Gráfica de sectores que represente o desglose dos gastos.
- Lista cunha combinación dos datos anteriores pero para cada parcela.

Para obter a terceira parte do informe, usarase a función *getTareas*, que xa usamos na *Xestión de tarefas*, pasándolle como parámetro a referencia catastral de cada parcela. Para as dúas primeiras partes do informe, deberase crear un rexistro na base de datos para poder almacenar os datos de todas as tarefas dunha forma global. Para facer isto, engadiuse a función *addDatoEstadistico* no servizo *FirebaseCloudService*. Esta función recibe como parámetro o identificador do usuario, tipo de tarefa, cantidade, data e identificador da tarefa e, como podemos ver na imaxe 8.29, permite crear un novo documento con toda esa información centralizada.

```dart
Future<void> addDatoEstadistico(String uid, String tipo, double cantidad,
    DateTime fecha, String id) async {
  return await _database
      .doc('usuarios/$uid')
      .collection('estadisticas')
      .doc(id)
      .set({
        'tipo': tipo,
        'total': cantidad,
        'fecha': Timestamp.fromDate(fecha)
      })
      .then((value) => print('dato Estadistico añadido'))
      .catchError((error) => print(error));
}
```

Figura 8.29: Servizo CloudFirebase: Función engadir dato estadístico.

Para obter a información estadística da base de datos, creouse a función *getDatosEstadisticos* que, de novo, fai uso de *Streams* e que devolve os datos ordeados por data para maior comodidade (figura 8.30).

```dart
Stream<QuerySnapshot> getDatosEstadisticos(String uid) {
  return _database
      .collection('usuarios/$uid/estadisticas')
      .orderBy('fecha')
      .snapshots();
}
```

Figura 8.30: Servizo CloudFirebase: Función para obter os datos estadísticos da base de datos.

Por último, implementouse a parte do *front end* correspondente. En concreto, creouse unha nova páxina denominada *EstadisticasPage* onde detallaremos toda a parte visual das estadísticas, sempre mantendo o deseño de marca. Esta páxina deberemos incorporala como unha ruta para que o manexador de estados a administre dende o *main.dart*. O resultado visual do desenvolvemento destes casos de uso é o seguinte:

Figura 8.31: Resultado visual do informe estadístico.

8.3 Sprint 3

En primeiro lugar, resulta importante mencionar que, durante o desenvolvemento deste *sprint*, xurdiu un problema que retrasou o comezo dalgunhas tarefas planificadas para este *sprint*. Firebase deixou de dar soporte ás versións estables dos paquetes que se estaban a utilizar na creación desta aplicación, polo que se tiveron que actualizar todos á última versión. Isto trouxo consigo diversos problemas como, por exemplo, que Firebase modificou a nomenclatura das funcións e obxectos nos paquetes *cloud_firistore* e *firebase_auth* para pasar a usar a mesma que usan as librerías de Javascript. Esta modificación fixo que se tiveran que volver a construír por completo ambos servizos adaptando todo o código á nova nomenclatura, o que supuxo un gran contratempo con respecto a planificación inicial para este *sprint*.

8.3.1 Xestión de usuarios: Novas funcionalidades

Unha vez resolto o problema comentado anteriormente, abordáronse os casos de uso *Forgot-Password* (táboa 5.5) e *Obter información usuario* (táboa 5.22). Comezamos co primeiro caso de uso, que ten como finalidade permitir restablecer o contrasinal de usuario no caso de esquecelo.

O usuario poderá solicitar, introducindo o seu email, que lle envíen un correo para restablecer o seu contrasinal. Para acadar isto, é preciso engadir a función *sendEmailNewPassword* no servizo *FirebaseAuthService*. Esta función recibe como parámetro o email do usuario e fai uso dunha funcionalidade do paquete *firebase_auth* que permite restablecer o contrasinal dun usuario, como se pode apreciar na imaxe 8.32. Ademais, esta funcionalidade permite controlar que o email introducido sexa correcto.

```
Future sendEmailNewPassword(String email) async {
  try {
    return await _auth.sendPasswordResetEmail(email: email);
  } catch (e) {
    print(e.toString());
    if (e.toString().startsWith('PlatformException(ERROR_INVALID_EMAIL')) {
      return 'ERROR_INVALID_EMAIL';
    } else {
      if (e.toString().startsWith('PlatformException(ERROR_USER_NOT_FOUND')) {
        return 'ERROR_USER_NOT_FOUND';
      } else {
        return null;
      }
    }
  }
}
```

Figura 8.32: Servizo AuthFirebase: Función restablecer contrasinal.

Continuase co segundo caso de uso, que ten como finalidade mostrar a información do usuario, entre a que destaca:

- Nome.
- Email.
- Data de incorporación.
- Número de parcelas.
- Balance actual.
- Imaxe avatar.

Para obter o nome, email e data faise uso do *UserModel* que xa está compartindo esa información co manexador de estados. O número de parcelas e o balance actual obtense facendo uso das funcións *getParcelas* e *getDatosEstadisticos* creadas anteriormente. Por último, para obter a imaxe de avatar é necesario crear o servizo *FirebaseStorageService* dentro do cartafol *services*. Para crear esta clase usaremos os paquetes instalados durante o inicio do proxecto, en concreto *cloud_storage* xa que, importando este paquete, poderemos crear unha instancia da clase *FirebaseStorage* que usaremos para definir o comportamento do novo servizo. A continuación, crearemos 2 funcións: *addPhoto* e *getPhoto*. A primeira das funcións recibe como parámetro o identificador do usuario, a imaxe e un cartafol, implementa a lóxica para gardar a foto no sitio indicado e retorna unha url para poder consultala (figura 8.33). A segunda función devolve a url correpondente á foto de avatar do usuario.

```
class FirebaseStorageService {
  final _storage = FirebaseStorage.instance;

  Future<String> addPhoto(File file, String uid, String rc) async {
    StorageReference ref = _storage.ref().child('$uid/$rc');
    StorageUploadTask upload = ref.putFile(file);
    StorageTaskSnapshot response = await upload.onComplete;
    String url = await response.ref.getDownloadURL();
    return url;
  }

  Future<String> getPhoto(String uid, String rc) async {
    StorageReference ref = _storage.ref().child('$uid/$rc');

    String url = await ref.getDownloadURL();
    return url;
  }
}
```

Figura 8.33: Servizo StorageFirebase: Exemplo de función para engadir unha imaxe.

Para rematar, modifícase o *front end* para incorporar estas novas funcionalidades. En concreto, modifícase a páxina *LoginPage*, para incluír a posibilidade de restablecer o contrasinal, e a páxina *PerfilPage* para mostrar toda a información dun usuario.

8.3.2 Xestión de parcelas e xestión de produtos: Novas funcionalidades

Os derradeiros casos de uso que quedan por implementar son os relativos a eliminar datos e obter información dos diferentes elementos da aplicación. Para eliminar un dato, hai que engadir a correspondente función no servizo *FirebaseCloudService* que faga uso da operación *.delete()*, preestablecida no paquete *cloud_firebase*. A condición que se define para poder eliminar un produto é que este non fora usado nunha tarefa, mentres que a condición para eliminar unha parcela é que non conteña tarefas. Isto faise desta forma para protexer a integridade dos datos rexistrados na aplicación. Ademais, cando se elimina unha tarefa, é preciso reestablecer a cantidade de produto usado nesa tarefa e actualizar o seu valor no alpendre.

Para os casos de uso de obtención de información, todos constrúense con peticións GET a unha url. En particular, realízanse as seguintes accións:

- Produto Fitosanitario: descárgase un pdf coa información sobre composto, aplicación e prazos de seguridade.
- Produto Fertilizante: móstrase páxina web do ministerio de agricultura con información sobre o composto.
- Parcela : móstrase páxina web con información catastral e tamén permite abrir localización da parcela en Google Maps (Android) ou Maps (iOS).

Capítulo 9

Conclusións

Tras rematar o desenvolvemento deste proxecto e cubrir os obxectivos propostos, pódese facer unha reflexión sobre o mesmo. Algunha das conclusións que se poden extraer de todo o traballo realizado son:

- A creación deste proxecto foi unha proposta propia que naceu como unha oportunidade de crear unha solución a diversos problemas no sector agroforestal.
- Cumpríronse os obxectivos, polo que a aplicación creada cubre as expectativas iniciais.
- En certo modo, logrouse construír unha ferramenta moi potente e que pode resultar moi útil no sector agroforestal.
- As funcionalidades, a interface do usuario é a súa usabiliade fan que aplicación sexa moi completa.
- A combinación de deseño de UI e deseño de marca fixo que a aplicación mostre un aspecto moi coidado e profesional.
- A nivel persoal, este proxecto supuxo todo un reto e a curva de aprendizaxe foi longa ao ter que usar tecnoloxías coas que non tivera contacto ó longo da carreira. Isto fixo que tivera que aprender linguaxes novas como Dart, JavaScript, ..., tecnoloxías novas como Flutter , Node.js, Firebase, WMS e tamén aprender a manexar estados nunha aplicación real con *Provider*.
- A necesidade de xestionar tarefas propias dun proxecto grande (planificación, análise de riscos, ...) e ter que buscar solucións aos diferentes imprevistos que foron xurdindo durante o desenvolvemento foi unha gran aprendizaxe e experiencia que pode axudarme no meu futuro profesional.

En definitiva, quedo cun sentimento de satisfacción moi grande ó ver como, a partir dunha idea, naceu xa non só unha aplicación de control e xestión agroforestal, senón que naceu Agrotab.

9.1 Liñas de traballo futuro

As características da aplicación fan que engadir novas funcionalidades sexa relativamente fácil. Algunha das posibles melloras que se poden implementar nun futuro son:

- Internalización
- Roles de usuario: para dar paso ó uso da aplicación a nivel empresa.
- *ToDo List*: construír unha lista de tarefas diaria para poder ter unha organización.
- Grupos: poder crear un grupo de traballo con varios usuarios e compartir tarefas a realizar.

Apéndices

Apéndice A

Paquetes externos

Neste apéndice detállanse todos os paquetes de terceiros usados en Flutter durante o desenvolvemento deste proxecto:

- **firebase_core** [45]
  - Versión: 0.5.0
  - Autor: firebase.google.com
- **firebase_auth** [46]
  - Versión: 0.18.0+1
  - Autor: firebase.google.com
- **cloud_firestore** [47]
  - Versión: 0.14.0+2
  - Autor: firebase.google.com
- **firebase_storage** [51]
  - Versión: 4.0.0
  - Autor: firebase.google.com
- **provider** [44]
  - Versión: 4.3.1
  - Autor: dash-overflow.net
- **flutter_custom_clippers** [52]
  - Versión: 1.1.1

  - Autor: dlohani.com.np

- **flutter_circular_chart** [53]
  - Versión: 0.1.0
  - Autor: victor.oc@gmail.com
- **getwidget** [54]
  - Versión: 1.1.0
  - Autor: getwidget.dev
- **flutter_svg** [55]
  - Versión: 0.18.0
  - Autor: dnfield.dev
- **url_launcher** [56]
  - Versión: 5.5.0
  - Autor: flutter.dev
- **intl** [57]
  - Versión: 0.16.1
  - Autor: dart.dev
- **location** [50]
  - Versión: 3.0.2
  - Autor: bernos.dev
- **flutter_map** [49]
  - Versión: 0.10.1+1
  - Autor: jpryan.me
- **http** [58]
  - Versión: 0.12.2
  - Autor: dart.dev
- **xml2json** [59]

  - Versión: 4.2.0
  - Autor: darticulate.com

- **html** [60]
  - Versión: 0.14.0+3
  - Autor: dart.dev

- **sliding_up_panel** [61]
  - Versión: 1.0.2
  - Autor: akshathjain.com

- **image_picker** [62]
  - Versión: 0.6.7+4
  - Autor: flutter.dev

- **pie_chart** [63]
  - Versión: 3.1.1
  - Autor: ayushpguptaapg@gmail.com

- **charts_flutter** [64]
  - Versión: 0.9.0
  - Autor: mit@google.com

- **carousel_slider** [65]
  - Versión: 2.2.1
  - Autor: serenader.me

- **animate_do** [66]
  - Versión: 1.7.2
  - Autor: fernando-herrera.com

- **maps_launcher** [67]
  - Versión: 1.2.2+1
  - Autor: julienscholzo@hotmail.de

Relación de Acrónimos

CEL Common Expression Language. 17

IDE Integrated Development Environment. 11, 13

INE Instituto Nacional de Estadística. 1

MVC Modelo-Vista-Controlador. 45

MVMC Montes Veciñais en Man Común. 1

PAC Política Agrícola Común. 2

SDK Software Development Kit. 11, 13, 17

UI User interface. 14, 15, 19, 40, 43, 45, 47, 79

VSCode Visual Studio Code. 11

WMS Web Map Service. 44, 65–67, 70, 79

Bibliografía

[1] "Páxina web oficial do ine." [En liña]. Dispoñible en: https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736164439&menu=resultados&idp=1254735576581

[2] "Páxina web da conselleria de medio rural con documentación sobre o sector forestal." [En liña]. Dispoñible en: https://mediorural.xunta.gal/sites/default/files/temas/forestal/estatistica/AEF2019/index.html#p=1

[3] "Páxina web do real decreto 1211/2012." [En liña]. Dispoñible en: https://www.boe.es/buscar/act.php?id=BOE-A-2012-11605

[4] "Páxina web oficial de granular." [En liña]. Dispoñible en: https://granular.ag

[5] "Páxina web oficial de farmbrite." [En liña]. Dispoñible en: https://www.farmbrite.com

[6] "Páxina web oficial de easyfarm." [En liña]. Dispoñible en: http://www.easyfarm.com

[7] "Páxina web descarga do caderno de campo." [En liña]. Dispoñible en: https://www.mapa.gob.es/es/agricultura/temas/sanidad-vegetal/productos-fitosanitarios/uso-sostenible-de-productos-fitosanitarios/

[8] "Páxina web oficial de agrotima." [En liña]. Dispoñible en: https://www.agroptima.com/es/

[9] "Páxina web oficial de isagri." [En liña]. Dispoñible en: https://www.isagri.es/geofoliasmartphone-1464.aspx

[10] "Páxina web oficial de vscode." [En liña]. Dispoñible en: https://code.visualstudio.com

[11] "Páxina web da extensión dart para vscode." [En liña]. Dispoñible en: https://marketplace.visualstudio.com/items?itemName=Dart-Code.dart-code

[12] "Páxina web da extensión flutter para vscode." [En liña]. Dispoñible en: https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter

[13] "Páxina web oficial de android studio." [En liña]. Dispoñible en: https://developer.android.com/studio

[14] "Páxina web oficial de xcode." [En liña]. Dispoñible en: https://developer.apple.com/xcode/

[15] "Páxina web oficial de adobe xd." [En liña]. Dispoñible en: https://www.adobe.com/products/xd.html

[16] "Páxina web oficial de adobe ilustrator." [En liña]. Dispoñible en: https://www.adobe.com/products/illustrator.html

[17] "Páxina web oficial de adobe photoshop." [En liña]. Dispoñible en: https://www.adobe.com/products/photoshop.html

[18] "Páxina web oficial de notion." [En liña]. Dispoñible en: https://www.notion.so

[19] "Páxina web oficial de dart." [En liña]. Dispoñible en: https://dart.dev

[20] "Páxina web oficial de kotlin." [En liña]. Dispoñible en: https://kotlinlang.org

[21] "Páxina web oficial de swift." [En liña]. Dispoñible en: https://developer.apple.com/swift/

[22] "Páxina web sobre javascript." [En liña]. Dispoñible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[23] "Páxina web sobre cel." [En liña]. Dispoñible en: https://opensource.google/projects/cel

[24] "Páxina web oficial de flutter." [En liña]. Dispoñible en: https://flutter.dev

[25] "Páxina web oficial de pub.dev." [En liña]. Dispoñible en: https://pub.dev

[26] "Páxina web oficial de android." [En liña]. Dispoñible en: https://www.android.com/intl/es_es/what-is-android/

[27] "Páxina web oficial de ios." [En liña]. Dispoñible en: https://www.apple.com/es/ios/ios-13/

[28] "Páxina web oficial de node.js." [En liña]. Dispoñible en: https://nodejs.org/en/

[29] "Páxina web oficial de firebase." [En liña]. Dispoñible en: https://firebase.google.com

[30] "Páxina web con documentación de realtime database." [En liña]. Dispoñible en: https://firebase.google.com/products/realtime-database

[31] "Páxina web con documentación de cloud firebase." [En liña]. Dispoñible en: https://firebase.google.com/products/firestore

[32] "Páxina web con documentación de cloud functions." [En liña]. Dispoñible en: https://firebase.google.com/products/functions

[33] "Páxina web con documentación de authentication." [En liña]. Dispoñible en: https://firebase.google.com/products/auth

[34] "Páxina web con documentación de hosting." [En liña]. Dispoñible en: https://firebase.google.com/products/hosting

[35] "Páxina web con documentación de cloud storage." [En liña]. Dispoñible en: https://firebase.google.com/products/storage

[36] "Páxina web con documentación de firebase ml." [En liña]. Dispoñible en: https://firebase.google.com/products/ml

[37] D. Avison and G. Fitzgerald, *Information systems development: methodologies, techniques and tools*. McGraw-Hill, 2003.

[38] J. H. Canós and M. C. P. P. Letelier, "Metodologías ágiles en el desarrollo de software," 2012.

[39] "Páxina web oficial de scrum." [En liña]. Dispoñible en: https://www.scrum.org

[40] "Páxina web con documentación sobre kanban." [En liña]. Dispoñible en: https://www.atlassian.com/agile/kanban

[41] "Páxina web con informe sobre os salarios en españa no sector das tic." [En liña]. Dispoñible en: https://www.tecnoempleo.com/informe-empleo-informatica.php

[42] "Páxina web sobre importancia dunha marca." [En liña]. Dispoñible en: https://magentaig.com/importancia-marca-marketing/

[43] "Páxina web sobre *Branding*." [En liña]. Dispoñible en: https://ignaciojaen.es/la-importancia-de-la-marca-para-el-negocio/#En_busca_del_Storytelling_de_cada_marca

[44] "Páxina web oficial do paquete provider." [En liña]. Dispoñible en: https://pub.dev/packages/provider

[45] "Páxina web oficial do paquete *firebase_core*." [En liña]. Dispoñible en: https://pub.dev/packages/firebase_core

[46] "Páxina web oficial do paquete *firebase_auth*." [En liña]. Dispoñible en: https://pub.dev/packages/firebase_auth

[47] "Páxina web oficial do paquete *cloud_firestore*." [En liña]. Dispoñible en: https://pub.dev/packages/cloud_firestore

[48] "Páxina web oficial do wmsinspire de catastro." [En liña]. Dispoñible en: http://www.catastro.minhap.es/webinspire/index.html

[49] "Páxina web oficial do paquete *flutter_map*." [En liña]. Dispoñible en: https://pub.dev/packages/flutter_map

[50] "Páxina web oficial do paquete *location*." [En liña]. Dispoñible en: https://pub.dev/packages/location

[51] "Páxina web oficial do paquete *cloud_storage*." [En liña]. Dispoñible en: https://pub.dev/packages/firebase_storage

[52] "Páxina web oficial do paquete *flutter_custom_clippers*." [En liña]. Dispoñible en: https://pub.dev/packages/flutter_custom_clippers

[53] "Páxina web oficial do paquete *flutter_circular_chart*." [En liña]. Dispoñible en: https://pub.dev/packages/flutter_circular_chart

[54] "Páxina web oficial do paquete *getwidget*." [En liña]. Dispoñible en: https://pub.dev/packages/getwidget

[55] "Páxina web oficial do paquete *flutter_svg*." [En liña]. Dispoñible en: https://pub.dev/packages/flutter_svg

[56] "Páxina web oficial do paquete *url_launcher*." [En liña]. Dispoñible en: https://pub.dev/packages/url_launcher

[57] "Páxina web oficial do paquete *intl*." [En liña]. Dispoñible en: https://pub.dev/packages/intl

[58] "Páxina web oficial do paquete *http*." [En liña]. Dispoñible en: https://pub.dev/packages/http

[59] "Páxina web oficial do paquete *xml2json*." [En liña]. Dispoñible en: https://pub.dev/packages/xml2json

[60] "Páxina web oficial do paquete *html*." [En liña]. Dispoñible en: https://pub.dev/packages/html

[61] "Páxina web oficial do paquete *sliding_up_panel*." [En liña]. Dispoñible en: https://pub.dev/packages/sliding_up_panel

[62] "Páxina web oficial do paquete *image_picker*." [En liña]. Dispoñible en: https://pub.dev/packages/image_picker

[63] "Páxina web oficial do paquete *pie_chart*." [En liña]. Dispoñible en: https://pub.dev/packages/pie_chart

[64] "Páxina web oficial do paquete *charts_flutter*." [En liña]. Dispoñible en: https://pub.dev/packages/charts_flutter

[65] "Páxina web oficial do paquete *carousel_slider*." [En liña]. Dispoñible en: https://pub.dev/packages/carousel_slider

[66] "Páxina web oficial do paquete *animate_do*." [En liña]. Dispoñible en: https://pub.dev/packages/animate_do

[67] "Páxina web oficial do paquete *maps_launcher*." [En liña]. Dispoñible en: https://pub.dev/packages/maps_launcher