Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática (Mención en Ingeniería del Software)

Aplicación multiplataforma para Prevención de Riesgos Laborales con Flutter y Firebase

Autor: **D. Alejandro Romero Pacho**

Tutor: **D. Blas Torregosa García**

Agradecimientos

Este proyecto no se habría llevado a cabo de no ser por todas esas personas que han estado a mi lado durante el transcurso de la carrera. En especial:

A los compañeros y amigos, los que ya tenía y los que he ido conociendo, por su compañía y apoyo en los peores momentos.

A los profesores, en especial a mi tutor Blas por aportarme sus conocimientos y guiarme durante el desarrollo de este proyecto.

Y por último, a las tres personas más importantes de mi vida: mis padres y mi hermana. Tanto por su apoyo emocional y económico, como sus sabios consejos durante mi carrera académica, que en muchas situaciones me han ayudado a seguir y que es posible que solo no hubiera podido superar.

Resumen

El objetivo de este Trabajo de Fin de Grado es desarrollar una aplicación multiplataforma para la realización de inspecciones de riesgos laborales, facilitando y agilizando la labor de los trabajadores del sector que actualmente tienen que realizar dicho trabajo a papel.

La aplicación funciona con un inicio de sesión que comprueba con la base de datos si el usuario existe, y en caso de que así sea permita al usuario acceder a la aplicación. Una vez ha iniciado sesión, podrá realizar inspecciones de riesgos laborales, teniendo a su vez la posibilidad de añadir nuevos riesgos que se vaya encontrando durante la inspección junto con una valoración de los mismos. Una vez acabada la inspección, se dará la posibilidad de generar un informe para agrupar los datos de dicha inspección.

El proyecto ha sido desarrollado utilizando el Framework Flutter y la base de datos Firebase para el desarrollo. Para la planificación se ha utilizado la metodología ágil Scrum.

Abstract

The objective of this Final Degree Project is to develop a multi-platform aplicaciónlication for the creation of occupational risk inspections, making easier the work of the inspectors who until now have to carry out such work on paper.

The aplicaciónlication works with a login that checks with the database if the user exists, and if it´s the case it allows the user to access the aplicaciónlication and be able to carry out an inspection of occupational risks, being able to add risks that are found during the inspection along with an assessment. Once the inspection is finished, it will be possible to generate a report to group the data of the inspection.

The project has been developed using the Flutter Framework and the Firebase database for development. It is also important to say that the agile Scrum methodology has been used for planning.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Riesgos laborales

Se entiende como riesgo laboral a los peligros que puedan llegar a ocurrir en una profesión y/o tarea profesional concreta, dichos riesgos están directamente relacionados con el entorno o el lugar de trabajado, en los cuáles dependiendo de diferentes factores se pueden originar diferentes accidentes o siniestros que pueden acabar provocando daños físicos o psicológicos en un trabajador. La mejor forma de evitar que estos riesgos se vuelvan realidad es teniendo buenos planes de prevención, por ello es necesaria la implementación de un Sistema de Gestión y Seguridad en el trabajo, cuyos requisitos se encuentran actualmente establecidos por la norma ISO 45001. [ISO15]

Los riesgos y la dimensión de los daños causados varían en función del tipo de trabajo, por lo que los riesgos laborales dependen en gran medida del lugar de trabajo. [Bee15]

1.1.1. ISO 45001

Es una norma que se rige por igual internacionalmente y especifica los requisitos para un sistema de gestión de salud y seguridad ocupacional, proporcionando además las indicaciones especificas para su uso, permitiendo a las empresas que sus trabajadores estén en condiciones seguras y saludables y evitando accidentes en el trabajo y problemas de salud. [Wik19]

A causa de la situación actual (crisis sanitaria por la pandemia), IAF decidió ampliar el plazo de migración a la nueva norma, por lo que los certificados OHSAS 18001 se extenderán hasta seis meses, y el periodo para finalizar la migración entre estas normas se ha aplazado hasta el 30 de Septiembre de 2021. Pese a esto es más interesante centrarse en la nueva norma que es la que va a estar presente los próximos años. [AEN21]

Vamos a ver algunos de los beneficios que ha traído a las organización la implantación de este sistema de gestión de seguridad y salud conforme a lo especificado en la ISO 45001:

- Menor índice de lesiones y enfermedades relacionadas con el trabajo.

- Disminución de muertes relacionadas con el trabajo.
- Disminución de peligros o reducción de los riesgos relacionados con la seguridad y salud laboral.
- Mejoría en el desempeño y la eficacia de la seguridad y salud laboral.
- Es una buena muestra de el compromiso de una empresa con la seguridad de sus trabajadores.
- Mejora la reputación de su marca.
- Aumenta la motivación y el compromiso de los empleados.
- Toma de conciencia y desarrollo de una cultura preventiva de la organización.

1.1.2. Prevención de riesgos laborales

Es una disciplina que busca generar una mejora en la seguridad y salud de los trabajadores en las empresas, para ello, se centra en la aplicación de diversas medidas y el desarrollo de actividades que logren prevenir los riesgos derivados de las condiciones del trabajo, teniendo como principal herramienta la evaluación de riesgos laborales llevada a cabo en cada empresa por especialistas en Prevención de Riesgos Laborales, es por ello que actualmente todas las empresas tienen un especialista en el ámbito que advierte sobre los diferentes riesgos a los nuevos trabajadores. [dSySL]

Para que una empresa pueda cumplir de manera sencilla con esto, se creo una lista que cuenta actualmente con 346 riesgos. Para entender mejor que se podría llegar a considerar un riesgo en un trabajo, vamos a ver una lista de los más comunes (esto depende mucho del sector de la empresa, por lo que el hecho de ser muy común en general no implica que vaya a ser así en todos los lados): [con20]

- **Riesgo eléctrico:** Hay muchos equipos utilizados para desempeñar tareas muy habituales que requieren del consumo de electricidad, como son por ejemplo la propia iluminación o los ordenadores. Esto en sí no supone un problema, pero el hecho de necesitar consumo eléctrico implica tener instalaciones eléctricas, y en caso de que estas estén en mal estado o presenten algún tipo de deterioro pueden llegar a provocar cortocircuitos que supongan daños materiales o incluso personales en el peor de los casos. [con20]
- **Incendio y explosión:** La posibilidad de que haya una explosión o que se produzca un incendio, puede parecer algo poco común, pero una mala calidad del cableado o una mala conexión de los diferentes enchufes puede llegar a provocar cualquiera de estos dos riesgo. En caso de este tipo de riesgo, no bastará con asegurarse de que las instalaciones estén en buen estado para evitarlo, sino que tendrán que tener un plan de contingencia en caso de que sucedan, ya que aún maximizando esfuerzos para evitarlos pueden llegar a suceder y pueden suponer perdidas muy graves. [con20]

- **Contaminación acústica:** Pese a que este no parezca un riesgo, puede llegar convertirse en uno. Para entender esto mejor vamos a ver un ejemplo: en el caso de que las actividades habituales de la empresa conlleven sonidos altos como puede ser caso de las discotecas, o algunos talleres en los que la maquinaria empleada provoque sonidos fuertes y molestos, esto puede acabar ocasionando problemas serios de audición o trastornos psicológicos en los trabajadores. [con20]
- **Temperaturas extremas:** Este es un riesgo que muchas empresas ignoran, lo cual es un error bastante grave, ya que temperaturas demasiado bajas o altas pueden llegar a desencadenar problemas de salud en los trabajadores que tengan que sufrirlas durante muchas horas (resfriados, infartos, golpes de calor...). [con20]
- **Exposición a pantallas de ordenador:** Se trata de un riesgo que está ganando mucha popularidad recientemente, ya que cada vez son más las tareas que se llevan a cabo con dispositivos electrónicos, lo que provoca que muchos trabajadores tengan que pasar un gran número de horas delante de sus pantallas. Esto puede llegar a provocar problemas de visión. [con20]
- **Virus, infecciones bacterianas y otros agentes orgánicos:** Otro riesgo que hasta hace poco pertenecía al grupo de los ignorados. Hablamos de la presencia de bacterias, hongos, virus y otros agentes orgánicos en los espacios de trabajo, por ello, es muy importante mantener una higiene adecuada en los entornos de trabajo para evitar así el contagio. [con20]
- **Problemas psicológicos:** Por último, vamos a hablar de los riesgos relacionados con problemas psicológicos. Cuando hablamos de riesgos laborales, generalmente pensamos en lesiones físicas como heridas o fracturas. Sin embargo, estos últimos van más allá, pues la salud mental puede verse fácilmente perjudicada si las condiciones de trabajo son inapropiadas. Los trastornos psicológicos más habituales cuando hablamos de trabajo son depresión, estrés o ansiedad. [con20]

1.1.3. Inspección de riesgos laborales

Cuando un inspector está realizando una inspección de riesgos laborales tiene que añadir cada riesgo que se encuentra, ya sea potencial o existente y realizar una evaluación del mismo dándole un valor final según 3 parámetros:

- **Nivel de deficiencia:** Es la magnitud obtenida de la relación entre el conjunto de los riesgo detectados y su relación con los posibles incidente y la efectividad de las medidas de prevención implementadas en el lugar de trabajo. [Saf19]
  - **Muy deficiente (10):** Los factores de riesgos generados por las deficiencias encontradas tienen una alta probabilidad de acabar provocando daños mayores. No existen métodos que puedan llegar a evitar que esto suceda o que logren una protección en caso de que sucedan. [Saf19]
  - **Deficiente (6):** Los factores de riesgos generados por las deficiencias encontradas han de ser corregidos, ninguna de las medidas de prevención y protección que se utilicen aseguran una eficacia total ante el riesgo. [Saf19]

  - **Mejorable (2):** Los factores de riesgos generados por las deficiencias encontradas tienen menor importancia, pese a esto no deben de ser ignorados. El conjunto de medidas de prevención y protección existentes son altamente eficaces ante el riesgo. [Saf19]
  - **Aceptable (0):** No se han detectado deficiencias destacables. Las medidas de prevención y protección que se pueden aplicar controlan completamente el riesgo. [Saf19]

- **Nivel de exposición:** Es una medida de la frecuencia con la que se da exposición al riesgo. Para estimar el nivel de exposición de un riesgo, están implicados diferentes factores, de entre ellos que podemos destacar: los tiempos que está un trabajador en áreas peligrosas, las operaciones con maquinaria, etc... [4101]

  - **Continuada (4):** Cuando un riesgo sucede de manera frecuente a lo largo de la jornada laboral y con tiempos prolongados de exposición.
  - **Frecuente (3):** Cuando un riesgo sucede de manera frecuente a lo largo de la jornada laboral y con tiempos cortos de exposición.
  - **Ocasional (2):** Cuando un riesgo sucede de ocasional a lo largo de la jornada laboral y con tiempos cortos de exposición.
  - **Esporádica (1):** Cuando un riesgo sucede de manera irregular (no se repite en una misma jornada laboral, o no es no lo normal).

- **Nivel de consecuencias:** Para definir este nivel se ha popularizado el dividirlo en dos significados; por un lado, estarían los daños físicos y, por otro, los daños materiales. Dicha división está hecha para que se traten de manera independiente, teniendo como es lógico más peso los daños personales que los materiales [4101]. Daños personales y materiales:

  - **Mortal o catastrófico (100):** Un muerto o más si estamos hablando de daños personales. || Destrucción total del sistema en caso de los daños materiales.
  - **Muy grave (60):** Lesiones graves sobre el personal en cuando a los daños personales (lo más seguro es que la persona que sufra el daño requiera de un tiempo de recuperación). || Destrucción parcial del sistema en lo que refiere a los daños materiales, gran dificultad para reparar los daños sufridos.
  - **Grave (25):** Lesiones leves sobre el personal en cuando a los daños personales (es posible que la persona que sufra el daño requiera de un tiempo de recuperación). || Destrucción parcial del sistema en lo que refiere a los daños materiales, a diferencia de la clasificación muy graves, lo daños son más fáciles de reparar.
  - **Leve (10):** Lesiones leves sobre el personal en cuando a los daños personales (menores que en el caso de la clasificación grave, el empleado que sufre el riesgo no requerirá de un tiempo muy grande de recuperación). || Destrucción parcial del sistema en lo que refiere a los daños materiales, a diferencia de la clasificación graves, lo daños son aún más fáciles de reparar.

1.2. Aplicaciones móviles

Las aplicaciones móviles son programas software diseñados para ser ejecutados en teléfonos, tablets y otros dispositivos móviles. [Sof] Los sistemas operativos que hay en la actualidad para dispositivos móviles son principalmente:

- **Android:** Líder actual del mercado móvil en sistemas operativos, se basa en Linux
- **iOS:** Sistema operativo utilizada por los dispositivos de la marca aplicaciónle, en caso de los móviles los iphone.
- **Windows Phone:** Llamado anteriormente en Windows Mobile, es un sistema operativo móvil desarrollado por Microsoft.

Las tiendas en las que se pueden descargar las aplicaciones de estos tres sistemas operativos son:

- **Google Play:** Del sistema operativo Android.
- **aplicación Store:** Del sistema operativo iOS.
- **Windows Phone Store:** Del sistema operativo Windows Phone

1.2.1. Diferencia entre aplicaciones móviles y sitios web móviles

Cuando hablamos de una aplicación, nos estamos refiriendo a programas que se instalan en el mismo software del dispositivo, logrando que está aplicación se encuentre en dicho dispositivo de forma permanente y pueda usarse de forma continuada. Muchas de estas aplicaciones, requerirán de la conexión internet para su uso, pero no desaparecerán del dispositivo en caso de que no haya conexión. [Sof]

Vamos a ver también los sitio web móvil, son páginas que han sido creadas de manera responsiva, lo que permite que puedan visualizarse y ajustarse a diferentes tamaños de pantallas. Viéndolo de otra manera, simplemente se tendrá que colocar la url de la página web que se desee buscar desde el buscador de tu Smartphone o tablet y en cuanto aparezca, verás cómo la misma se adapta al tamaño de tu pantalla, esto no descargará ningún programa en tu dispositivo. [Sof]

1.2.2. Aplicaciones nativas

Son aplicaciones diseñadas exclusivamente para un único sistema operativo móvil, y que por lo tanto pueden aprovechar sus funciones al máximo. Son exclusivo para una plataforma o dispositivo en particular. En la actualidad, gran parte de la aplicaciones móviles están diseñadas para los sistemas operativos Android y iOS de manera exclusiva, lo que hace que una aplicación Android no pueda se usada en iOS y viceversa. [Her20]

La ventaja que tienen dichas aplicaciones, en que al estar diseñadas para un sistema operativo su rendimiento es alto y aportan una excelente experiencia de usuario. Después de todo, durante el desarrollo se prueba en un dispositivo nativo, por lo que las interfaces van a quedar ajustadas a medida, además al desarrollar una aplicación nativa se tiene acceso a un amplio abanico de APIs que ayudan a aumentar la funcionalidad y acelerar el desarrollo. [Her20]

Puede que todo lo que se ha dicho hasta ahora nos lleve a pensar que estas aplicaciones solo tienen ventajas, pero tienen un importante inconveniente, su costo. Para desarrollar y mantener una aplicación nativa se necesita un equipo de desarrollo exclusivo, lo que implica que si quieres una aplicación para Android y otra para iOS necesites dos equipos de desarrollo. [Her20]

1.2.3. Lenguajes utilizados

Puesto que hay muchos lenguajes empleados para el desarrollo de aplicaciones móviles vamos a ver solo los más populares en la actualidad:

Lenguajes Android

- **Java:** Según el índice de TIOBE, Java es el lenguaje de programación más popular a partir de junio de 2017. Si quieres desarrollar aplicaciones Android, el primero o uno de los primeros lenguajes que se te tendrían que venir a la cabeza es Java. Actualmente tiene una de las comunidades de desarrolladores más grandes y establecidas, lo cual significa que se puede conseguir fácilmente ayuda y soporte técnico. [Sir17]

- **Kotlin:** Kotlin se desarrolló con el objetivo de buscar soluciones a algunos de los problemas de Java. Muchas opiniones dicen que la sintaxis de Kotlin es más simple, más limpia y suele conllevar menos consumo de recursos (gran limitación de Java). Esto le permite centrarse más en buscar soluciones a problemas, en vez de luchar con sintaxis verbal. También, se puede utilizar Kotlin y Java juntos en el mismo proyecto, la cual es una solución muy potente. [Sir17]

Lenguajes iOS

- **Swift:** Recientemente aplicaciónle ha ido añadiendo característica de gran valor a este lenguaje, como por ejemplo: sintaxis simplificada, capacidad de localizar con precisión los errores durante el desarrollo. Los enormes esfuerzos de aplicaciónle para promover Swift indican claramente su intención de que este nuevo lenguaje se convierta en uno de los más populares para desarrollar en iOS. [R20]

- **Objective-C:** Fue el lenguaje de desarrollo original para iOS. Mientras que Swift se presenta como el futuro del desarrollo de iOS, muchos proyectos en producción aún dependen de Objective-C (muchos otros están empleando recursos en migrar a Swift). Por lo que se espera que la transición de Objective-C a Swift no sea algo rápido. [R20]

Lenguajes multiplataforma

- **TypeScript:** Es un conjunto de JavaScript y ofrece una mayor seguridad mediante la adición de tipificación estática opcional. Además de esto, ofrece un mejor soporte para el desarrollo de aplicaciones a gran escala. Lenguaje desarrollado y mantenido por Microsoft, que permite a los desarrolladores diseñar aplicaciones móviles multiplataforma. [Sir17]

- **JavaScript:** Lenguaje con una largar historia, muy popular para el desarrollo front-end. Su objetivo es permitir a los desarrolladores web hacer todo lo posible para mejorar la experiencia del usuario. [Sir17]

- **C#:** Es muy similar a Java y C++, lo cual se debe a que Microsoft ha adoptado algunas de las características de Java para simplificar su arquitectura, manteniendo C++ como diseño. Por último, una ventaja destacable es su amplia comunidad. [Sir17]

1.2.4. Frameworks utilizados

Al igual que con los lenguajes hay una gran cantidad de Frameworks por lo que solo vamos a ver los más utilizados en la actualidad: [ROS21]

- **Ionic:** Es un Framework de código abierto que se utiliza en el desarrollo de aplicaciones móviles híbridas, es decir, que combinan el HTML5, CSS y JavaScript logrando obtener como resultado aplicaciones con una interfaz bonita e intuitiva para el usuario. Dichas aplicaciones tienen como objetivo su comercialización o descargar en plataformas Android o iOS. [dev19]

- **React Native:** Es un Framework que utiliza JavaScript para crear aplicaciones reales nativas para iOS y Android, para ello, se basa en la librería de JavaScript llamada React, la cual se utiliza para la creación de componentes visuales. React native cambia el propósito de dicho componente para que, en vez de ser usados en páginas web, se puedan lanzar sobre plataformas móviles, en este caso iOS y Android. [Bla]

- **Native Android:** Es el SDK nativo de Android y proporciona a los desarrolladores acceso al uso de las bibliotecas API y las herramientas necesarias para crear, probar y depurar aplicaciones nativas para la plataforma Android. La gran mayoría de los desarrolladores que implementan partes de sus aplicaciones nativas usando C & C ++ actualmente prefieren utilizar este Framework, ya que aporta una gran facilidad y total flexibilidad para desarrollar aplicaciones para dispositivos móviles. [Bla]

- **Xamarin:** Es una plataforma de código abierto para compilar aplicaciones modernas multiplataforma haciendo uso de .NET. [pro20]

  Su principal ventaja es la facilidad a la hora de reutilizar código, en todas las aplicaciones multiplataforma que hayamos desarrollado y se encuentren en producción, habrá módulos iOS que se hayan tenido tenido que portar a Java, y/o módulos Android que se hayan tenido que portar a Objective-C o Swift.

- **Native Script:** Es un Framework de código abierto para desarrollar aplicaciones en iOS y Android. Fue ideado y desarrollado por Progress. la ventaja de las aplicaciones NativeScript es que se construyen utilizando lenguajes de programación independientes del dispositivo y sistema operativo como son JavaScript o TypeScript, lo que permite crear aplicaciones nativas para cualquier sistema operativo. [Wik21c]

Cabe destacar que en esta sección no se han incluido ni Flutter en los Frameworks, ni Dart en los lenguajes, ya que al ser lo utilizado para la realización de este trabajo estará explicado en la sección de tecnologías utilizadas. [yXC]

Capítulo 2

Tecnologías utilizadas

2.1. Flutter

Flutter es un SDK de código abierto de desarrollo de aplicaciones móviles creado por Google. Suele usarse para desarrollar interfaces de usuario para aplicaciones en Android, iOS y Web. [Wik] Los componentes importantes de Flutter incluyen:

- **Flutter engine:** Principalmente escrito en C++, proporciona soporte a bajo nivel para la renderización, para la cual utiliza Google Skia. Además, se vincula con SDKs de Android e iOS.4 utilizando MethodChannels y EventChannels que permiten la comunicación entre el Flutter engine y el nivel nativo del sistema operativo. [Wik]
- **Foundation library:** Está escrito en el lenguaje Dart (también desarrollado por Google), proporciona clases básicas y funciones que se suelen utilizar para construir las aplicaciones con Flutter, como APIs y librerías para comunicar con el motor. [Wik]
- **Widgets:** El diseño de las aplicaciones con Flutter se compone por una serie de Widgets anidados. Dichos Widgets representan una descripción inmutable de parte de la interfaz de usuario; todos lo que se encuentra en las interfaces de Flutter esta compuesto por Widgets. La idea es combinar todo tipo de Widgets para conseguir obtener una interfaz agradable y intuitiva. [Wik]
- **Design-specific Widgets:** El Framework de Flutter contiene dos librerías para Widgets que se utilizan de forma concreta para realizar interfaces para cada sistema operativo. Widget material implementa el diseño de Google del mismo nombre, y los Widgets Cupertino imita el lenguaje diseño de iOS de aplicaciónle . [Wik]

Sus principios son:

- **Desarrollo rápido:** Da vida a tu aplicación en milisegundos con Stateful ”Hot Reload”. Utilizando todo tipo de Widgets anidados de diferentes maneras se podrán crear interfaces nativas. Además, está la función anteriormente mencionada llamada ”Hot Reload”, la cual es una herramienta implementada por Flutter que permite

con sola una tecla actualizar de manera instantánea el código agregado a las clases. Por mi experiencia durante el desarrollo es algo muy útil que agiliza enormemente el proceso de desarrollo de interfaces, evitando tener que lanzar la aplicación por cada pequeño cambio. [Fluc]

- **Diseño de interfaces de usuario expresivas y flexibles:** Permite enviar funciones rápidamente con un enfoque en las experiencias nativas del usuario final. Esto se logra gracias a sus bibliotecas y a su la arquitectura extensible que y en capas que presenta. [Fluc]

- **Rendimiento nativo:** Los Widgets de Flutter incorporan todas las diferencias críticas de plataforma, como el desplazamiento, la navegación, los iconos y las fuentes, y su código de Flutter se compila en el código de máquina ARM nativo utilizando los compiladores nativos de Dart. [Fluc]

Con Flutter el montaje del interfaz UI se hace con Widgets, algunos de ellos con estado y otros sin estado, en función de si requiere ser actualizado dinamicamente o no (cabe destacar que los Widgets no son más que clases Dart creada para el diseño de interfaces en Flutter). La idea es igual a la que utiliza para realizar las interfaces el Framework React: es el mismo Widget el que se encarga del renderizado en función de la configuración y de su estado interno (dicho estado solo existirá en caso de que sea un Widget con estado, además para modificar dicho estado se utilizará la función setState()). [Esc20]

Por último, mencionar que es el mismo Framework el que se encargar de actualizar la interfaz cuando el estado de una clase cambia. Por tanto, una buena manera de resumir la estrategia de desarrollo sería: anidado de Widgets, que nos lleva a acabar teniendo pantallas cuya estructura es similar a árboles como el siguiente: [Esc20]

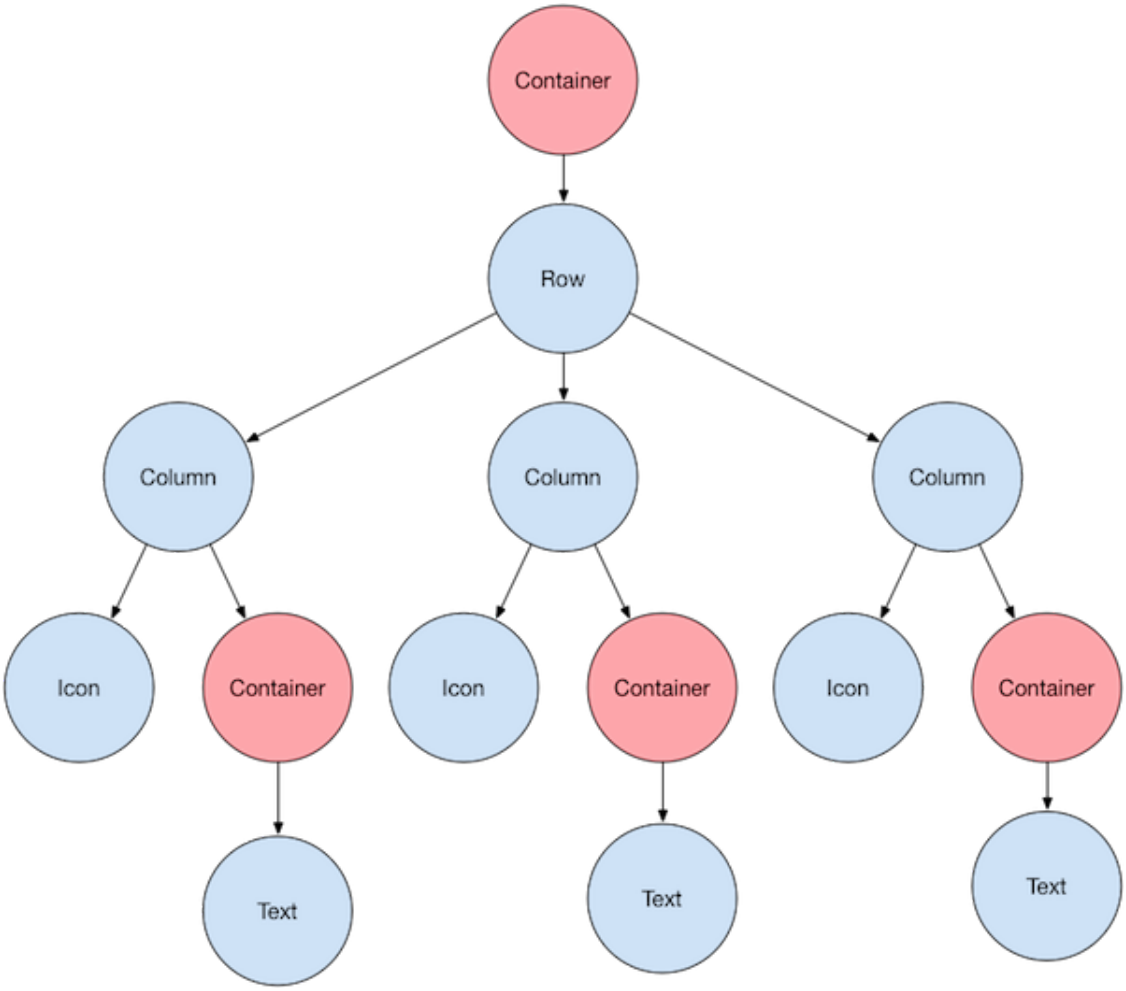


Figura 2.1: árbol de Widgets

Por último, vamos a ver las ventajas y desventajas de usar este Framework respecto a otros, dando también mi opinión, que pese a que solo he trabajado con Xamarin y Flutter en desarrollo de móvil me vale para hacer una pequeña comparación entre ambos:

2.1.1. Ventajas:

- **Tiempo de desarrollo más rápido:** Una de las mayores ventajas de Flutter es la rapidez en el desarrollo de aplicaciones. Gracias a la base de código única de Flutter (funcional tanto para Android como para iOS), puede iniciar su aplicación en múltiples plataformas. Esto ayuda a que los desarrolladores solo necesitan realizar un desarrollo en lugar de tener que realizar uno para cada sistema operativo. Por lo tanto, el desarrollo de Flutter ahorra esfuerzos de desarrollo lo que lo hace extremadamente rentable. Actualmente, muchas empresas tienen más de un grupo de desarrollo, en función de los sistemas operativos para los que se quiera desarrollar la aplicación, mientras que si se utiliza Flutter, es posible que la aplicación no sea tan potente como si fuera una aplicación nativa desarrollada para Android o iOS específicamente, pero en el caso de aplicaciones que no son tan potentes es una buena herramienta para ahorrar costes. Destacar que esta ventaja no es particular de Flutter, sino de todos los Frameworks multiplataforma. [Ebe21]

- **Personalización libre y buen renderizado:** La arquitectura en capas de Flutter puede ayudar al desarrollador a personalizar libremente los componentes de la interfaz de usuario, dependiendo de la manera en la que se aniden los Widgets se pueden obtener interfaces completamente diferentes, lo que genera en el desarrollador la sensación de tener el control de cada pixel de la pantalla. A diferencia de la mayoría de Frameworks, Flutter no necesita componentes de interfaz de usuario específicos de la plataforma. [Ebe21]

- **Función de Hot Reload:** La recarga en caliente de Flutter es una función revolucionaria que lo distingue de todos sus competidores y posiblemente una de las herramientas que agilice más el desarrollo. Gracias a ”hot reloading”, pueden realizar cambios en el código y verlos reflejados en la pantalla de manera instantánea en tiempo de ejecución. Esto evita que se tenga que volver a compilar el código cada cambio. Esto permite a los desarrolladores experimentar libremente con diferentes elementos de la interfaz de usuario al crear aplicaciones. Siendo muy útil sobre todo en los casos en los que quieres realizar muchos cambios básicos (cambios en los colores, posiciones de los elementos, tamaños...), permitiendo ver en todo momento como afectan dichos cambios. [Ebe21]

- **Rendimiento equivalente al de las aplicaciones nativas:** Debido a que la buena experiencia de usuario es el factor más importante que determina el éxito de una aplicación, un rendimiento deficiente puede hacer de una aplicación buena algo inservible. Esto es algo muy positivo para Flutter, ya que su parecido al desarrollo de aplicaciones nativas hace que su rendimiento sea ideal, (en mi caso una vez generé la apk el rendimiento era perfecto, funcionando casi de manera instantánea). De hecho, en mucho casos, ni si quiera la complejidad de dicha aplicación es algo que afecte negativamente al rendimiento. La razón principal detrás de esto, es que Flutter no depende de ningún código intermedio o puente JavaScript, manteniéndolo integrado directamente en el código de la máquina (lo que elimina la ralentización de ”recorrer” este puente), eliminando así los posibles errores que pueden aparecer durante el proceso de integración. [Ebe21]

- **Buena documentación y una comunidad solida:** Comunidad activa de desarrolladores. Además, también ofrece una extensa documentación en forma de blogs y tutoriales a través de su portal oficial. En lo que respecta a mi experiencia, es el Framework con mayor documentación de todos los que he utilizado, lo cual permite poder encontrar prácticamente cualquier cosa que se desee y una fácil solución a la gran mayoría de errores que puedan aparecer. [Ebe21]

2.1.2. Desventajas:

- **Demasiado reciente:** Flutter es uno de los Frameworks de desarrollo de aplicaciones más recientes y siempre está evolucionando. La gran mayoría de sus librerías no se encuentran en una versión final y requieren de actualizaciones de forma constante. Por lo tanto, podríamos decir que Flutter aún no está en su punto de máximo potencial, posiblemente de aquí a unos años se convierta en una herramienta mucho más potente. [Ebe21]

- **Archivos de gran tamaño:** Otra desventaja de usar Flutter es el tamaño de archivo de crear la propia aplicación. Esto hace que el tiempo que se tarda en iniciar sea en algunas ocasiones demasiado largo, esto es un indicar de bajo rendimiento que puede llegar a ser perjudicial para la experiencia del usuario. Pese a que esto es algo que sigue presente a medida que pasa el tiempo se va mejorando. [Ebe21]

- **Modificaciones y actualizaciones:** Dado que es una herramienta relativamente nueva, sufre actualizaciones de manera regular, haciendo que si un desarrollo se prolonga por un largo periodo de tiempo tenga que ser refactorizado debido a estas actualizaciones. En resumen, mantener el código se convierte en una tarea más compleja cuando el Framework no para de cambiar.

  Esta desventaja la he sufrido durante el desarrollo, ya que al realizar actualizaciones regularmente algunos de los Widgets utilizados previamente tras unas actualizaciones estaban desfasados, lo que lleva a tener que refactorizar el código cada cierto tiempo solo debido a las actualizaciones. [Ebe21]

- **Falta de pautas de desarrollo estándar:** Falta de pautas o normas bien redactadas que faciliten y estandaricen el desarrollo. Si bien es verdad que actualmente hay patrones como el Provider o el BLoC que han convertido el desarrollo en una tarea más sencilla, aún falta mucho en comparación a otras herramientas, sobre todo cuando se trata de aplicaciones complejas la falta de estas pautas puede llegar a ser un problema. [Ebe21]

- **Falta de claridad:** El código del programa puede volverse confuso al integrar los Widgets, esto es fácil de resolver modularizando las clases, separando en sub-Widgets la estructura. [ION20b]

Por último, vamos a ver uno de los patrones más utilizados para el desarrollo de aplicaciones con Flutter, y por lo tanto uno de los que tuve en mente para utilizar, al final acabe decidiendo utilizar el patrón Provider (el cual explico en la sección de diseño).

La razón por la cual me acabe decidiendo por utilizar el patrón Provider en lugar del BLoC, es debido a que busque información y en aplicaciones cuya complejidad no es muy grande se recomienda usar el Provider por simplicidad, mientras que en aplicaciones más complejas se recomienda usar el BLoC, ya que pese a ser más complejo se acaba obteniendo mayor claridad.

2.1.3. Patrón BLoC

El Patrón BLoC (Business Logic Component) fue diseñado por Paolo Soares y Cong Hu de Google, y fue presentado en la Dart Conference 2018. [Fer20b]

El objetivo con el equipo de Google diseño este patrón, fue la reutilización de código entre sus aplicaciones mobile, utilizando Flutter con Dart, y web, utilizando Angular Dart. [Fer20b]

En esta presentación de la Dart Conference 2018 Paolo Soares presentó el patrón pero es una implementación concreta para Dart que aplicaron en Flutter y Angular Dart. [Fer20b]

Sin embargo, el patrón tiene mucho más fondo que esto. Esto hace que este patrón pueda ser utilizado con otras herramientas como ocurre en el caso Redux. [Fer20b]

En resumen, el principio del patrón BLoC es tener algo que haga de intermediario entre la vista y el modelo, dichas clases serán los BLoCs y se utilizaran para hacer de puente entre ambos., que en este caso al utilizar clean architecture sería el dominio, con un estado observable. [Fer20a]

El patrón BLoC fue presentado por Google inicialmente con los siguientes objetivos:

- **Centralizar la lógica de negocio:** La idea es que toda la lógica de los componentes de las interfaces se agrupe en clases llamadas BLoC, dichas clases deberán ser ajenas a librerías (de esta manera lograremos que este completamente aislado). [Fer20b]

  Aislando en dichas clases toda la lógica logramos que la aplicación sea más fácil de escalar y aumente la facilidad para realizar futuros cambios sobre ella, la idea es ”similar” al funcionamiento de las APIs, los BLoCs serían las APIs para entenderlo mejor. [Fer20b]

- **Centralizar cambios de estado:** En estas clases se centralizaran todos los cambios de estado de los Widgets, de esta manera estas clases serán las únicas que se encarguen de recibir las acciones y actualizar el estado de las interfaces de la aplicación. [Fer20b]

  Dichas clases deberán ser observables puesto que serán las encargadas de comunicar a todos los Widgets los cambios de estado que se realicen en cada uno de ellos, además son los encargados de comunicar las diferentes partes de la aplicación.

- **Mapear al formato que necesita la vista:** Para conseguir un aislamiento total lo ideal es que sean los mismos BLoCs los que se encarguen de obtener el formato correcto de los datos, de esta manera se logrará que este completamente aislado y se pueda reutilizar en otro momento. Esto reafirma lo comentado anteriormente (la similitud con las APIs). [Fer20b]

Las clases BLoC se basan en un flujo de datos unidireccional:

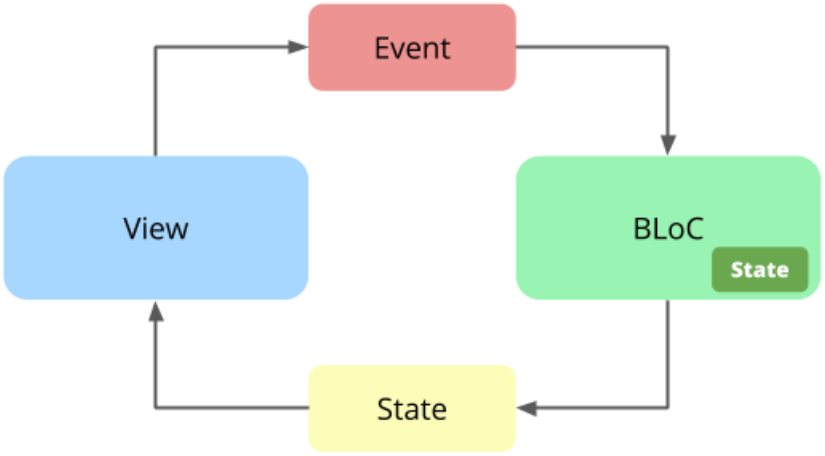


Figura 2.2: BLoC unidireccional

La conexión entre las interfaces y el estado va solo en una dirección, el estado es leído desde la pantalla para renderizar y mostrar el actual.

En caso de que se quiera modificar el estado, primero habrá que manejarlo desde el Widget y después modificando los valores del BLoC a través de llamadas al mismo, una vez que se actualice el BLoC este notificará actualizando todas las partes en las que está presente. [Fer20b]

En caso de que viéramos el funcionamiento de este patrón como una caja negra sería de la siguiente manera:

Figura 2.3: caja negra BLoC

Para entender mejor como utilizar este patrón tenemos que ponernos en que se tiene que crear un BLoC por cada pantalla.

Los pasos a seguir para trabajar con ello serían: [Fer20b]

- **Crea un BLoC para cada pantalla o página:** En algunos casos incluso se requiere más de un BLoC en una sola pantalla.
- **Crea un BLoC para cada pantalla o página:** Examinar cada pantalla y decidir qué Widgets necesitan realizar un seguimiento de un estado, que por lo tanto necesitará un BLoC, está es la razón por la que puede haber más de uno por pantalla, en caso de que en una pantalla se realice seguimiento de estados de más de un elemento se requerirán varias clases BLoC.
- **Enumerar los eventos:** Enumerar los diferentes eventos que están sucediendo dentro de una pantalla.
- **Enumerar los estados:** Aclarar los estados que deberían de salir del BLoC y las entradas que este debería de tener.
- **Construir el BLoC y usarlo:** Construir las clases y invocarlas desde las pantallas que lo requieran.

2.2. Dart

Dart es un lenguaje open source desarrollado en Google con el objetivo de permitir a los desarrolladores utilizar un lenguaje orientado a objetos y con análisis estático de tipo. Desde la primera versión estable en 2011, Dart ha cambiado bastante, tanto en el lenguaje en sí como en sus objetivos principales. Con la versión 2.0, el sistema de tipo de Dart pasó de opcional a estático, y desde su llegada, Flutter se ha convertido en el principal objetivo del lenguaje. [Div]

A diferencia de muchos lenguajes, Dart se creo teniendo en mente el objetivo de agilizar y facilitar el proceso de desarrollo. Por eso, tiene un gran conjunto de herramientas que vienen integradas con el propio lenguaje, gestor propio de paquetes, varios compiladores, analizador y formateador de código... Además, el compilador justo a tiempo hace que una vez se realice un cambio se ejecute de manera inmediata. [Div]

Una vez la aplicación se encuentre en producción, el código será compilado en lenguaje nativo del propio sistema operativo, por lo que no será necesario tener un entorno especial para ejecutar. En caso de que el desarrollo se realice para un sitio web la transpilación se hará a JavaScript. [Div]

En lo que refiere a la sintaxis, es muy similar a lenguajes como JavaScript, Java y C ++, lo que hace que si queremos aprender Dart conociendo uno o más de estos lenguajes se convierta en una tarea mucho más sencilla. En mi caso personal había trabajado con JavaScript y Java y se me hizo relativamente sencillo entender el funcionamiento de Dart. Además, otra gran ventaja de Dart para el diseño de aplicaciones móviles es que consta de una gran simplicidad y facilidad para la asincronía, y trabajar con generadores y iterables es extremadamente sencillo. [Div]

Dart está enfocado primariamente en la programación para smartphones, tabletas y ordenadores, pero recientemente aunque es menos común se ha utilizado también para servidores. El objetivo de este lenguaje es simplificar el desarrollo de aplicaciones. Tanto

Flutter como la herramienta de publicidad conocida como Google Ads esta programado con Dart. Además de este, hay muchos otros ejemplos de plataformas famosas que han utilizado Dart como lenguaje para trabajar. [ION20a]

2.3. Firebase

Usaremos Firebase para la base de datos, ya que era uno de los requisitos principales de la aplicación para poder utilizarla sin necesidad de cargarla.

De esta manera, una vez generada la apk se podrá utilizar desde cualquier dispositivo de manera gratuita, si se hubiera realizado con otra base de datos se hubiera necesitado un servidor para almacenarla. Firebase es una buena solución gratuita para aplicaciones pequeñas. La peculiaridad de Firebase respecto a MySql (base de datos más vista en la carrera) es que funciona por colecciones y documentos en vez de tablas.

La base de datos en la nube tiene colecciones que serían las tablas de MySql, dentro de cada colección tiene documentos que serían las filas en MySql, y dentro de cada documento tiene sus parámetros junto con una id única para su eliminación y modificación.

También, destacar que tiene un alojamiento (Firebase Storage) que se puede utilizar para almacenar archivos de mayor peso, el cual se ha utilizado para almacenar las imágenes de los usuario y de las evaluaciones de los riesgos.

2.4. Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft, teniendo como sistemas operativos objetivo Windows, Linux y macOS. De entre todas las herramientas que contiene las más destacables son: soporte para la depuración, control Git integrado, resaltado de sintaxis, finalización inteligente de código (aunque en la mayoría de casos requiere de paquetes extras) y colocación del código automática. También se puede personalizar para que los usuarios puedan cambiar los temas del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto.[Wik21d]

2.5. Emulador de Android

Android Emulator simula dispositivos Android en un ordenador, de esta manera puedes probar una aplicación para dispositivos móviles sin necesidad de usar uno físico.

Para la realización de la práctica, se ha utilizado un emulador con la versión de Android 10Q y dispositivos físicos para realizar pruebas más reales.

El emulador proporciona casi todas las funciones de un dispositivo Android real. Puede simular llamadas entrantes y mensajes de texto, especificar la ubicación del dispositivo, usar diferentes velocidades de red, probar sensores de rotación y otros sensores de hardware, visitar la tienda Google Play y más.

Las principales diferencias con respecto a un dispositivo real suelen venir cuando se intenta hacer uso de la ubicación, por lo que cuando he requerido de utilizar esta herramienta he optado por probar con un dispositivo físico. [dev20]

2.6. Git

Es un software de control de versiones diseñado por Linus Torvalds, se pensó en que fuera eficiente, confiable y de alta compatibilidad para el mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. [Wik21a]

Además de permitir llevar un control de versiones, lo cual ayuda para guardar versiones anteriores de código y poder recuperarlas en caso de que uno de los cambios haya sido contraproducente, ayuda a que un grupo de trabajo pueda coordinarse a la perfección, permitiendo que más de una persona pueda trabajar sobre el mismo archivo y los cambios que realicen ambos se mantengan sin perder ninguno de ellos.

En el caso de este TFG se ha realizado en la plataforma Git Hub y siguiendo la teoría de Git Flow (aplicada en todo lo posible, ya que al ser una persona algunas de las ramas son innecesarias).

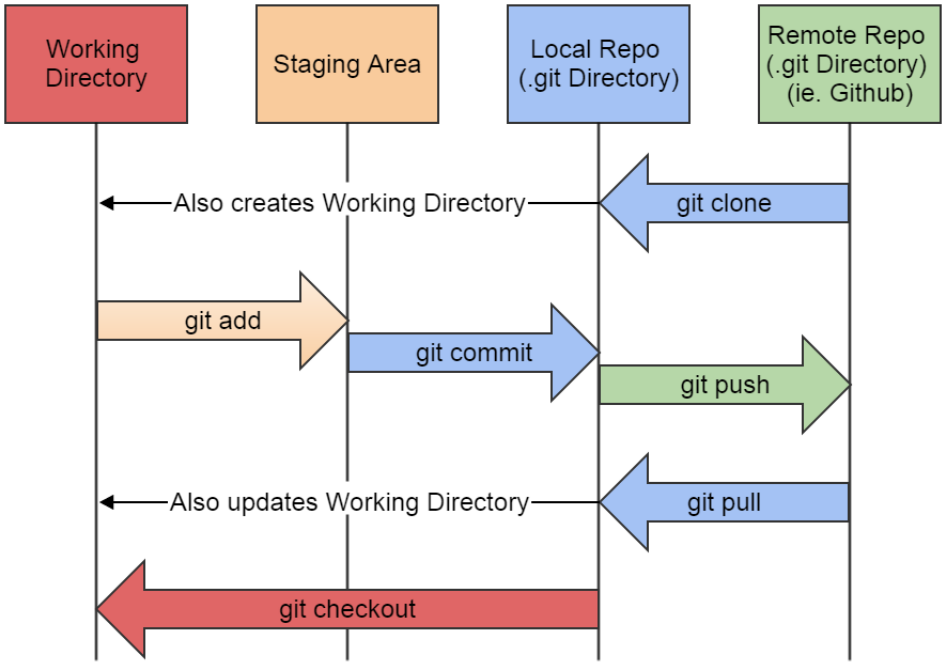


Figura 2.4: ejemplo básico uso Git

2.6.1. GitFlow

El flujo de trabajo GitFlow es un flujo de trabajo de Git que favorece el desarrollo continuo de software y las prácticas de implementación de DevOps. Fue Vincent Driessen en nvie quien lo publicó por primera vez y quien lo popularizó. El flujo de trabajo GitFlow define un modelo de creación de ramas estricto diseñado con la publicación del proyecto como fundamento. Proporciona un marco sólido para gestionar proyectos más grandes. [ATL]

La idea del flujo de trabajo GitFlow es tener las siguientes ramas:

- **Rama master:** Solo se mergearan los archivos una vez haya una versión estable de los cambios realizados, por lo que la idea es que en la versión main solo haya cambios cada cierto número de commits y, la suma de ellos supondrán por así decirlo, una versión diferente. [Lie18]
- **Rama de desarrollo:** Esta rama contendrá el historial completo del proyecto, es la rama a la que cada uno de los desarrolladores mergeará sus cambios, cuando en esta rama se sumen suficientes cambios como para contarlo como una versión se mergeara a la rama master. En resumen, es la rama en la que se junta el trabajo de todos los desarrolladores previamente a mergear a master. [Lie18]
- **Rama de desarrollador:** Esto es más opcional, pero por la experiencia obtenida en algunas asignaturas es recomendable a la hora de trabajar con un grupo de personas, su utilidad es por así decirlo crear un paso intermedio entre las ramas de funcionalidad y la de desarrollo, añadiendo estas ramas cada desarrollador mergeara sus cambios a su rama y una vez los tenga ahí los mergeara a la rama de develop (no es necesario pero ayuda a organizarse). [Lie18]
- **Ramas de función:** Estas últimas, son las ramas a las que se hace el push una vez realizado el cambio sobre el proyecto, la idea es que haya una por cada issue, por ejemplo: ramaDiseñoPantallaLogin. Estas ramas nunca interactúan directamente con la rama master, se mergean primero a la de desarrollador en caso de que la haya y luego a la de desarrollo. [Lie18]
- **Ramas de publicación:** Cuando la rama de desarrollo haya adquirido suficientes funciones para una publicación (o se este próxima la fecha de una publicación), se deberá bifurcar una rama de publicación a partir de una de desarrollo. Al crear esta rama, se inicia el siguiente ciclo de publicación, por lo que mientras se realicen los cambios necesarios en esta rama se podrán seguir añadiendo nuevas funcionalidad en las ramas súperiores a la de desarrollo (en la rama de publicación solo deberán producirse las soluciones de errores, la generación de documentación y otras tareas orientadas a la publicación). [ATL]

  De esta manera, se facilita que en caso de que el equipo sea de muchos desarrolladores, mientras que unos se centran en perfeccionar la versión de publicación, otros pueden ir trabajando sobre las nuevas funcionalidades.
- **Ramas de corrección:** Las ramas de corrección sirven para reparar rápidamente las publicaciones de producción. Las ramas de corrección son muy similares a las

ramas de publicación y de función, salvo por el hecho de que se basan en la master, no en la desarrollo. [ATL]

El tener una rama dedicada a la corrección de errores facilita que mientras se solucionen dichos errores, se pueda seguir con la producción y el desarrollo de las nuevas funciones. Se podría ver como un complemento a la rama de producción

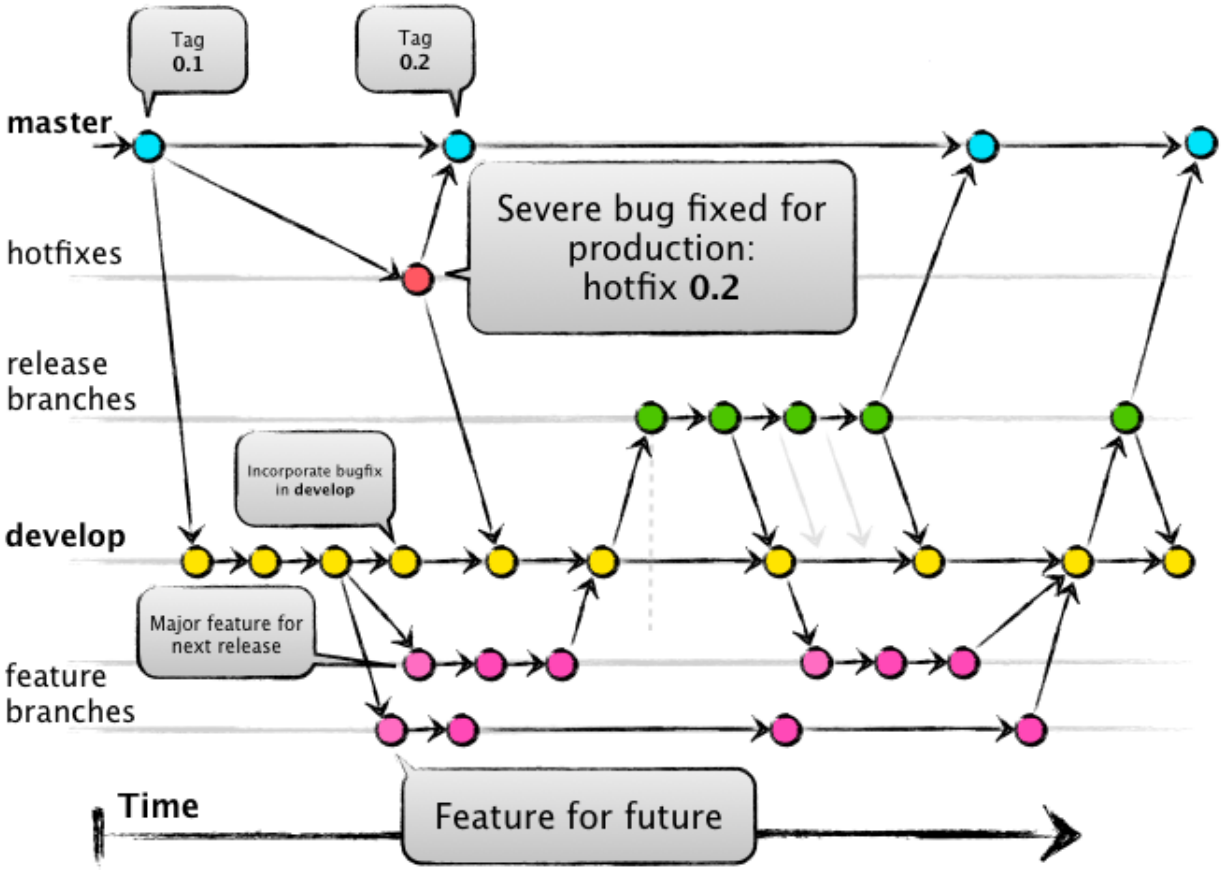


Figura 2.5: ejemplo ramas GitFlow

Una vez entendido lo que es, veamos el porque es tan importante utilizarlo en tu proyecto y en que situaciones renta aplicarlo:

¿Por qué implementarlo? [Roj21]

- Agiliza la comunicación entre el equipo de desarrollo y el de pruebas, lo que hace que el código lleva al segundo mucho más rápido.
- Disminuyen las posibilidad de aparición de errores humanos a la hora de mezclar archivos entre las diferentes ramas y/o perder cambios sobre archivos debido al trabajo sobre el mismo en paralelo.
- Aísla el momento de producción separándolo claramente del de desarrollo.
- Facilita que varios desarrolladores realicen cambios sobre el mismo archivo sin perder los cambios de ninguno de ellos.

¿Cuándo se recomienda implementarlo? [Roj21]

- Cuando el equipo de trabajo está al menos conformado por dos personas, cuanto mayor sea el número de personas más fácil será entender la utilidad de aplicar esta estrategia.
- Cuando se empleen metodologías ágiles.
- Cuando el proyecto varié con mucha frecuencia (alto número de commits).
- En caso de que el proyecto sea de mayor complejidad.
- Cuando se desee mantener una corrección de errores adecuada sin retrasar la adhesión de nuevas funcionalidades.

Ya sabiendo todo esto, por último, aclarar que el Git Flow que he aplicado para el proyecto no ha sido del todo correcto, ya que para una sola persona es algo innecesario. Por ello, yo he trabajado con la rama de master como si fuera la de desarrollo y creando ramas de función para cada issue.

De esta manera, creo una nueva rama para cada funcionalidad añadida y la mergeo a master, rama en la que almaceno todo el historial de cambios sobre el proyecto.

2.7. Astah Professional

Programa utilizado para la realización de diagramas UML (es el que más hemos utilizado durante la carrera y a la vez con el que más familiarizado estoy, por lo que me pareció el más correcto para esta tarea).

2.8. LaTeX

Es un sistema de composición de textos, orientado a la creación de documentos escritos que presenten una alta calidad tipográfica. Por sus características y posibilidades, es usado de forma especialmente intensa en la generación de artículos y libros científicos que incluyen, entre otros elementos, expresiones matemáticas. [Wik21b]

2.9. Pencil

Programa utilizado para la creación de bocetos de la interfaces durante la fase de análisis del proyecto.

Capítulo 3

Seguimiento del proyecto

3.1. Sprint 1 (10Febrero-17Febrero)

Cuadro 3.1: Tabla sprint 1

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
001	Documentación: Estructura del documento	2h	2h 45m	Completada
002	Documentación: Explicación teórica de Scrum	4h	5h	Completada
003	Documentación: Planificación inicial (costes, riesgos, metodología de proyectos...)	3h	3h	Completada
004	Documentación: Explicación teórica aplicaciones móviles	1h 30m	3h	Completada
005	Documentación: Explicación teórica riesgos laborales	1h 30m	3h	Completada
007	Aprendizaje inicial Flutter	30h	12h	Pendiente

Sprint asignado al comienzo de la redacción del documentos y al aprendizaje de las tecnologías, no se van a incluir ni el aprendizaje de como planificar un proyecto utilizando ”Scrum”, ni el de desarrollo de aplicaciones móviles, ya que se trato en asignaturas anteriores del grado, aunque en dicha asignatura se estudiaron aplicaciones Android y en este caso es multiplataforma la manera de proceder es similar.

En el caso de Scrum, como se ha dicho anteriormente, los sprints están compuestos por una serie de historias y cada historia es una petición de un cliente, por lo que en los primeros sprints no vamos a referirnos a ellos como historias sino como tareas, ya que al tratarse de funcionalidad externa al desarrollo (relacionada principalmente con el documento y el aprendizaje) no se podría considerar una historia.

3.2. Sprint 2 (17Febrero-24Febrero)

Cuadro 3.2: Tabla sprint 2

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
008	Aprendizaje inicial Flutter	30h	12h	Completada
009	Documentación: Realizar el análisis inicial (objetivos, requisitos, casos de uso y modelos de dominio)	5h	4h 20m	Completada
010	Diseño de la base de datos	3h	2h 30m	Completada
011	Documentación: Análisis funcionalidad de login y registro	1h 30m	2h	Completada
012	Documentación: Análisis funcionalidad gestión de usuarios	2h 30m	2h	Completada
013	Documentación: Análisis funcionalidad agregar inspección	2h 30m	2h	Completada
014	Documentación: Análisis funcionalidad agregar riesgos a una inspección	1h 30m	1h 15m	Completada
015	Documentación: Análisis funcionalidad evaluar un riesgo	1h 30m	1h 15m	Completada
016	Documentación: Análisis funcionalidad súper usuario (gestión inspectores)	1h 30m	1h 30m	Completada

Sprint asignado a la realización y redacción del análisis de la funcionalidad esperada para la aplicación, en este análisis se estudia la forma que se va a emplear para cumplir con los requisitos que se requieren para la aplicación, junto con la realización de los bocetos para cada una de las pantallas.

Además, se ha seguido con el aprendizaje del Framework para cuando empiece la parte de programación.

En los siguientes sprints, las historias van a estar relacionadas con la funcionalidad que se vaya a implementar (como por ejemplo: desarrollar el login de la aplicación) junto a la redacción del diseño y implementación.

3.3. Sprint 3 (8Marzo-15Marzo)

Cuadro 3.3: Tabla sprint 3

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
017	Crear aplicación Flutter y enlazar con proyecto Firebase	3h	2h 50m	Completada
018	Aprendizaje lectura y escritura de datos en Firebase usando Dart	4h	4h 20m	Completada
019	Diagrama de actividad de inicio de sesión	2h	1h 40m	Completada
020	Documentación: Parte de diseño referente a la parte de inicio de sesión	2h	2h 30m	Completada
021	Diseño de la pantalla de login	4h	3h 30m	Completada
022	Diseño de la pantalla de registro	3h	2h 30m	Completada
023	Aprendizaje navegabilidad entre pantallas	30m	1h	Completada

Sprint asignado a la creación del proyecto Flutter y a la realización y redacción de la parte de diseño de inicio de sesión, tanto de la funcionalidad de login como la de registro. En la redacción se incluyen tanto la parte de implementación, como el diagrama de actividad del inicio de sesión.

Además, se ha llevado a cabo la programación de las pantallas de registro y login, junto con el aprendizaje relacionado con la navegabilidad entre pantallas.

También se ha dedicado parte del tiempo de este sprint al aprendizaje de como hacer lecturas y escrituras con Firebase Cloud Storage.

3.4. Sprint 4 (15Marzo-22Marzo)

Cuadro 3.4: Tabla sprint 4

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
024	Implementar Firebase authenticator para el registro	2h	1h 30m	Completada
025	Implementar Firebase authenticator para el login	1h 30m	1h	Completada
026	Crear clases de dominio del modelo	3h	2h 40m	Completada
027	Organizar la lógica de lectura y escritura de base de datos en una clase aparte	30m	1h	Completada

Sprint asignado a la implementación de Firebase authenticator para el login y el registro con la base de datos, de esta manera se obtiene una seguridad mayor a una simple comprobación con el Cloud Storage.

Además, se han creado las clases de dominio del modelo y se han organizado las operaciones relacionada con la base de datos en una clase externa para mantener toda la lógica agrupada.

3.5. Sprint 5 (22Marzo-29Marzo)

Cuadro 3.5: Tabla sprint 5

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
028	Diagrama de actividad de gestión de usuario	2h	2h	Completada
029	Documentación: Parte de diseño referente a la parte de gestión de usuario	2h	3h	Completada
030	Diseño de la pantalla principal	2h	2h	Completada
031	Diseño del menú de navegación de la aplicación	1h 30m	1h 20m	Completada
032	Diseño de la pantalla de perfil	2h	1h 20m	Completada
033	Diseño de la pantalla de modificar perfil	1h	1h 10m	Completada
034	Organización carpetas del proyecto	30m	20m	Completada

Sprint asignado a la a la realización y redacción de la parte de diseño de las funcionalidades de gestión del usuario: pantalla principal, perfil, modificación de perfil y cambio de contraseña. En la redacción se incluyen tanto la parte de implementación, como el diagrama de actividad de las operaciones relacionadas con la gestión de usuario.

Además, se ha dedicado parte del tiempo de este sprint para organizar las carpetas del proyecto, separando así los modelos de las pantallas, y estas a su vez de los Notifiers.

3.6. Sprint 6 (5Abril-12Abril)

Cuadro 3.6: Tabla sprint 6

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
035	Aprendizaje del patrón Provider	1h	2h	Completada
036	Aprendizaje lectura y escritura de datos en Firebase usando Notifiers	1h	1h 20m	Completada
037	Mostrar datos usuario logeado en la pantalla de modificar perfil	2h	1h 20m	Completada
038	Permitir la modificación de campos de usuario	1h	1h 10m	Completada
039	Aprendizaje lectura y escritura de datos en Firebase Storage (para imágenes)	40m	1h	Completada
040	Permitir al usuario añadir y modificar una vez añadida una foto de perfil	1h	1h 30m	Completada

Sprint asignado al aprendizaje del patrón Provider, y a modificar la manera de realizar las escrituras y lecturas de la base de datos aplicando dicho patrón con la ayuda de las clases Notifier.

Además, realización de la lógica de la parte de modificar perfil y la carga de imagen de perfil del usuario, junto a la posibilidad de modificación de la misma.

3.7. Sprint 7 (20Abril-27Abril)

Cuadro 3.7: Tabla sprint 7

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
041	Aprendizaje realización de tests de los drivers en Flutter	1h	1h 40m	Completada
042	Diseño de test drivers	1h	1h 50m	Completada
043	Diseño de test de funcionalidad	1h	1h 10m	Completada
044	Realización de test de drivers de las páginas de login y registro	1h	1h 30m	Completada
045	Aprendizaje de patrón BLoC	1h	1h 30m	Completada
046	Diagrama de actividad de realización de una inspección	2h	2h	Completada
047	Documentación: Parte de diseño referente a la parte de realizar una nueva inspección	3h	4h 30m	Completada
048	Diseño de la pantalla de añadir inspección	2h	2h 40m	Completada
049	Diseño de la pantalla de añadir riesgos	2h 30m	2h 30m	Completada
050	Diseño de la pantalla de añadir sub-riesgos	1h	45m	Completada
051	Refactorizar la organización de carpetas del proyecto	20m	25m	Completada
052	Refactorizar la funcionalidad de cargar foto de perfil en la pantalla de perfil y de modificar perfil	20m	20m	Completada

Sprint asignado al aprendizaje de la realización de test drivers y diseño de los tests para las partes realizadas hasta ahora, es decir, las de gestión de usuario y de inicio de sesión. También se ha empleado parte del tiempo para aprender sobre el patrón BLoC, y estudiar la posibilidad de realizar el proyecto con este patrón en lugar del Provider.

Además, realización y redacción de la parte de diseño de agregar nueva inspección, junto con la programación de las pantallas de añadir inspección, seleccionar categoría y añadir riesgo, junto con el diagrama de actividad correspondiente a la secuencia de realización de una nueva inspección.

Por último, se ha dedicado parte del tiempo a reorganizar las carpetas del proyecto para aumentar la claridad del mismo y a separar la funcionalidad de cargar foto de perfil, evitando así la repetición de este código en dos pantallas.

3.8. Sprint 8 (27Abril-04Mayo)

Cuadro 3.8: Tabla sprint 8

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
053	Implementar la lógica de añadir inspección	40m	1h 20m	Completada
054	Implementar la lógica de añadir riesgo	30m	1h 20m	Completada
055	Diseño de la pantalla de lista de inspecciones	1h	1h 30m	Pendiente
056	Diseño de la pantalla de lista de riesgos por evaluar	30m	55m	Completada
057	Implementar la lógica de lista de inspecciones	30m	50m	Completada

Sprint asignado a la programación de la lógica de las pantallas de añadir inspección, seleccionar categoría y añadir riesgo. También se ha empleado el tiempo del sprint para la programación de la pantalla de lista de inspecciones y de su parte de lógica.

Además, realización y redacción de la parte de diseño de la pantalla de lista de inspecciones y de la pantalla de lista de riesgos por evaluar.

3.9. Sprint 9 (04Mayo-11Mayo)

Cuadro 3.9: Tabla sprint 9

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
058	Implementar la lógica de lista de riesgos por evaluar	2h	2h 30m	Completada
059	Diseño de la pantalla de realizar evaluación riesgo	2h	1h 50m	Pendiente
060	Implementar la lógica de realizar evaluación riesgo	1h	1h 40m	Pendiente
061	Solución retraso al actualizar la vista (al borrar un riesgo de la lista por evaluar)	1h	2h	Completada
062	Solución para mantener la pila de navegación correcta (que al dar atrás vuelva a la pantalla que toca)	1h	1h 55m	Completada

Sprint asignado a la programación de la lógica de lista de riesgo por evaluar y de la pantalla y la lógica de realización de evaluación de un riesgo.

Además, solucionar los problemas relacionados con el retraso al actualizar las pantallas, en este caso cuando se decidía borrar un riesgo ya evaluado había un retraso de unos segundos, y con la pila de navegación. Esto último, más que ser un error es algo ineficiente, al mantener la pila de navegación actualizada de manera correcta el usuario podrá volver a la pantalla anterior con el botón de atrás.

3.10. Sprint 10 (11Mayo-18Mayo)

Cuadro 3.10: Tabla sprint 10

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
063	Añadir imágenes a la evaluación de un riesgo, tanto el diseño como la lógica	2h	1h 40m	Pendiente
064	Añadir mensajes de error en evaluar riesgo y añadir inspección	1h	1h 10m	Completada
065	Refactor pantalla de lista de riesgos por evaluar para que se actualice dinámicamente cuando se elimina un riesgo	2h	1h 20m	Completada
066	Añadidos mensajes de confirmación al presionar el botón de volver atrás desde la pantalla evaluación, la de seleccionar riesgo y la de página principal	1h	45m	Completada
067	Refactor uso de la pila de navegación para moverse por la aplicación	1h	45m	Completada

Sprint asignado a la programación de la funcionalidad de añadir imágenes a la evaluación de un riesgo, además añadidos mensajes de error en las pantallas de evaluar riesgo y añadir inspección. Añadidos también mensajes de confirmación al intentar volver atrás desde las pantallas de evaluación, lista de riesgos por evaluar y principal.

Además, refactor de la pantalla de lista de riesgos por evaluar, para que se actualice de manera dinámica la lista de riesgos cuando se eliminar uno, y de la pila de navegación, la cual estaba mal actualizada en algunas de las pantallas.

3.11. Sprint 11 (18Mayo-25Mayo)

Cuadro 3.11: Tabla sprint 11

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
068	Cambios en el diseño del modal de recuperar contraseña	30m	40m	Completada
069	Implementar la lógica de recuperación de contraseña	1h	40m	Completada
070	Diseño de pantalla de inspección ya finalizada	2h	1h 30m	Pendiente
071	Implementar la lógica de cambio de contraseña y de email	1h 30m	50m	Pendiente
072	Refactor diseño del login y del registro	1h	1h 20m	Pendiente
073	Añadir ubicación a la pantalla de añadir nueva inspección	1h	45m	Pendiente

Sprint asignado a la programación de la funcionalidad de modificar contraseña y email, junto con la de recuperación de contraseña a través del email en caso de que el usuario lo requiera.

Además, realizados cambios sobre las interfaces de login y de registro y sobre el modal de recuperación de contraseña de la pantalla de login. También se ha empleado el tiempo para añadir ubicación a la creación de una nueva inspección, para ello se ha decidido añadir una nueva pantalla con un mapa.

3.12. Sprint 12 (25Mayo-01Junio)

Cuadro 3.12: Tabla sprint 12

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
074	Implementar la lógica de cambio de email	30m	40m	Completada
075	Diseño y implementar la lógica de la pantalla de cambio de contraseña	30m	50m	Completada
076	Añadir mapa para la ubicación a la pantalla de añadir nueva inspección	1h	2h 35m	Completada
077	Refactor de la manera en la que se realiza la inspección (añadir riesgo - evaluarlo)	40m	1h 30m	Completada
078	Diagrama de actividad referente a la funcionalidad de dar de baja usuario	1h 30m	2h	Completada
079	Documentación: Parte de diseño referente a la parte de dar de baja usuario	1h	1h 30m	Completada
080	Agregar usuario administrador	1h	1h 30m	Completada
081	Diseño de la pantalla de lista de inspectores	40m	40m	Completada

Sprint asignado a la programación de la pantalla del mapa para añadir una ubicación al agregar una nueva inspección, y a la refactorización de la manera en la que se gestionan las inspecciones, pasar a hacerlo de manera secuencial (el cliente indico que así es más intuitivo): 1-seleccionar categoría, 2-agregar riesgo y 3-evaluarlo.

Además, se ha empleado parte del tiempo de este sprint para la redacción y realización de la parte de diseño de la funcionalidad del usuario administrador de dar de baja a un inspector. También se ha diseñado la pantalla de lista de inspectores, y se ha agregado el nuevo usuario junto con los cambios en la pantalla principal que este requiere para acceder a su funcionalidad.

3.13. Sprint 13 (01Junio-08Junio)

Cuadro 3.13: Tabla sprint 13

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
082	Refactor limpieza de código (variables y imports que ya no utilizo) y unificar lenguaje	30m	1h 10m	Completada
083	Refactor diseño de la pantalla lista de inspecciones	30m	1h	Pendiente
084	Diseño de la pantalla de información inspector	1h 30m	1h	Pendiente
085	Añadir funcionalidad de súper usuario dar de baja a inspector	1h 30m	1h	Completada
086	Aprendizaje tests de integración	2h	4h	Completada
087	Tests de integración de inicio sesión (login y registro) de ambos usuarios	30m	1h 40m	Completada
088	Tests de integración de cambio de contraseña	30m	50m	Completada
089	Documentación: Pasar a limpio los sprints	4h	3h	Pendiente

Sprint asignado a la limpieza y organización del código y la realización de cambios sobre la pantalla de lista de inspecciones. Además, añadida pantalla de lista de inspectores y información de inspector, junto a la funcionalidad de darlos de baja.

Además, se ha empleado parte del tiempo de este sprint a la realización de test drivers de inicio de sesión y de cambio de contraseña.

3.14. Sprint 14 (08Junio-15Junio)

Cuadro 3.14: Tabla sprint 14

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
090	Documentación: Tecnologías utilizadas	6h	7h 30m	Completada
091	Documentación: Terminar de pasar a limpio algunos de los aspectos del análisis	1h 30m	1h	Completada
092	Documentación: Añadir bibliografía apuntada en sucio	2h	3h	Completada
093	Documentación: Agregar al documento los diagramas UML de análisis	20m	10m	Completada
094	Documentación: Agregar al documento los diagramas UML de diseño	20m	10m	Completada
095	Documentación: Pasar a limpio los sprints y redactar actas	4h	4h	Pendiente
096	Documentación: Redactar diseño funcionalidad inicio sesión	2h	3h	Completada

Sprint asignado a la documentación de las tecnologías utilizadas para la realización del proyecto y a la revisión del apartado de análisis.

Además, se ha utilizado el tiempo de este sprint para añadir correctamente las referencias indicadas durante el texto, y los diagramas uml.

3.15. Sprint 15 (15Junio-22Junio)

Cuadro 3.15: Tabla sprint 15

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
097	Documentación: Redactar parte de teoría de tests	2h	1h 30m	Completada
098	Programación test unitarios tanto de modelo como de cálculos	1h 30m	1h 30m	Completada
099	Prueba en el código para búsqueda de errores y solución de los mismo	1h	5h	Completada
100	Documentación: Redactar manual de usuario	2h	1h 50m	Pendiente
101	Documentación: Redactar parte te test tanto unitarios como de integración	2h	1h 10m	Completada
102	Test de integración de agregar inspección	2h	1h 40m	Completada
103	Test de integración de añadir riesgo	1h	55m	Completada
104	Test de integración de evaluar riesgo	2h	2h	Completada

Sprint asignado a la redacción de la parte teórica de las pruebas en Flutter y de los test unitarios y de integración realizados en el desarrollo. También se ha documentado la parte del manual de usuario.

Además, se ha empleado el tiempo de este sprint para la programación de los test unitarios de modelo y de operaciones, y para los de integración de agregar inspección, añadir riesgo y evaluar riesgo.

3.16. Sprint 16 (22Junio-28Junio)

Cuadro 3.16: Tabla sprint 16

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
105	Revisión final del código (limpieza y organización)	1h	50m	Completada
106	Añadidos cambios finales sobre la generación del informe	1h 30m	4h	Completada
107	Añadida latitud, longitud y altitud en la evaluación de un riesgo	30m	50m	Completada
108	Documentación: Redacción y realización pruebas end-to-end	2h	2h 25m	Completada
109	Documentación: Redactar manual de usuario	2h	1h 40m	Completada
110	Solución errores encontrados en la pila de navegación y en los margenes durante la realización de las pruebas end-to-end	50m	40m	Completada
111	Documentación: Redacción resumen, agradecimientos y conclusión	1h	45m	Completada
112	Documentación: Redacción líneas futuras	50m	40m	Completada
113	Documentación: Mejora en la plantilla de la memoria	1h	2h 30m	Completada
114	Documentación: Repaso final de la memoria y cambios	5h	6h 30m	Completada

Sprint asignado para hacer los últimos remates sobre el proyecto. En el, se ha realizado la redacción de los apartados de conclusión, agradecimientos, resumen y abstract, junto con la de las pruebas end-to-end y las líneas futuras. También se ha buscado mejorar la plantilla para que encaje con lo exigido por la universidad.

En cuanto al código se ha finalizado la funcionalidad de generar un informe, se ha añadido la localización a la evaluación de un riesgo y se ha realizado una revisión final.

Por último, se han solucionado algunos errores relacionado con la pila de navegación y se ha realizado un repaso final sobre la memoria para evitar errores de escritura.

3.17. Sprint 17 (07Julio-14Julio)

Cuadro 3.17: Tabla sprint 17

Tarea	Descripción	Tiempo estimado	Tiempo empleado	Estado
115	Arreglar dependencias plugins última versión de Flutter	1h	1h 30m	Completada
116	Añadir funcionalidad notificar baja al iniciar sesión	20m	50m	Completada
117	Volver a pasar los test para comprobar que las nuevas actualizaciones no han afectado a algo	1h	1h	Completada
118	Documentación: Añadir las fechas de la bibliografía	2h	2h 30m	Completada
119	Documentación: Añadir justificación para algunas de las pruebas	1h	1h	Completada
120	Documentación: Repaso final del documento	4h	5h	Completada
121	Documentación: Añadido presupuesto final del proyecto	2h	3h 20m	Completada
122	Documentación: Cambios en la estructura del anexo	40m	1h	Completada

Sprint asignado a las últimas tareas del proyecto, en el se han arreglado los errores generados a causa de las nuevas dependencias que han aparecido debido a la nueva actualización de Flutter. Además, se ha ampliado la funcionalidad relacionada con las bajas para que el usuario dado de baja sea notificado con el motivo cuando intenta acceder a la aplicación.

Por último, se han añadido los presupuestos finales del proyecto, además de una revisión final en busca de errores de escritura.

Capítulo 4

Planificación inicial

4.1. Introducción

En cuanto a como se realiza el desarrollo del software, las metodologías ágiles son las más rentables, permitiendo obtener un feedback por parte del cliente cada cierto tiempo, lo cual es perfecto para el diseño de una aplicación. La idea de usar esta metodología, es tener funcionalidad visible lo más pronto posible y ir añadiendo nueva para cada sprint, esto hará que el cliente pueda estar constantemente viendo el avance de la aplicación y haciendo sugerencias sobre mejoras y/o cambios.

Antes de empezar a entrar en más detalles, vamos a hablar brevemente de que trata la aplicación (explicando su utilidad) y los usuarios a los que estará dedicada:

La aplicación se va a realizar para facilitar el trabajo de los inspectores de riesgos laborales (único usuario al que esta dirigido), con el fin de evitar que este tenga que ir apuntando a mano todos los problemas con los que se encuentre. Con ayuda de esta aplicación, solo tendrán que ir añadiendo cada uno de los riesgos de la lista total de ellos, para poder así evaluarlos posteriormente basándose en el **nivel de deficiencia**, **nivel de exposición** y **nivel de consecuencias** y de manera opcional una descripción y/o una foto.

4.2. Planificación de costes y riesgos

Una vez definida la tecnología, hay que pasar a una fase esencial en cualquier tipo de proyecto, plantear los posibles riesgos y hacer una estimación de los costes asociados a la realización del proyecto.

En cuanto a los riesgos que se pueden contemplar son más simples de los que serían en un proyecto real, ya que al ser un proyecto que implica a dos personas (estudiante y tutor) se reducen respecto a un grupo más grande de trabajo:

- La falta de recursos (en este caso el único recurso sería el tiempo tanto del estudiante como del tutor).

- La imposibilidad de tener reuniones con el cliente para mostrarle los avances y que aporte feedback.
- Estimaciones no realistas (en cuanto al tiempo necesario para realizar las diferentes tareas).
- Desarrollar las funciones del software equivocadas.
- Desarrollar las interfaces erróneas.
- Chapado en oro (Gold plating): Adornar el sistema con cosas que no se usan, y que puedan llegar a conllevar gran coste de tiempo.
- Cambios de última hora en los requisitos.
- Que el software acabe siendo muy lento.
- Desarrollo demasiado difícil.

En cuanto a los costes, como en la mayoría de proyectos software son el trabajo de la persona (ya que no hay materiales que comprar), el único coste es el tiempo de las personas involucradas en el proyecto.

Al calcular la división del sueldo medio de un programador Junior anual, que es de 19.393 euros brutos, entre el número de horas que se trabaja un año (1888) nos da una media de 10.27 euros la hora, multiplicando esto por el número de horas totales de trabajo del proyecto tenemos los coste del proyecto:

Coste final del proyecto: 317h * 10.27€/h = 3255.59 euros

Cabe destacar que al total de horas, a parte del tiempo de las tareas de los sprints se han añadido los tiempos empleados para las reuniones y la investigación previa realizada para el proyecto (24h).

4.3. Objetivos y alcance

Una vez decidida la metodología a usar y presentada la idea de la aplicación, tendremos que ver los objetivos y el alcance del proyecto y una vez definidos ambos hacer una búsqueda de requisitos.

Primero vamos a hablar sobre los objetivos académicos:

- Familiarizarme con las herramienta Flutter para el desarrollo de aplicaciones móviles multi plataforma.
- Entender como se realiza un proyecto de software pasando por todas sus fases de principio a fin.

El objetivo principal de la aplicación es: **permitir que el usuario pueda realizar una inspección de riesgos laborales** y **permitir que el usuario pueda generar un informe con dicha inspección**, y en lo que respecta al alcance tendremos que hablar de otras cuestiones pare definirlo, ya que este estará definido por la idea del proyecto, sus objetivos, sus limitaciones y (de manera posterior) los posibles cambios. [34]

Puesto que la idea y el objetivo ya están definidos, quedaría hablar de las restricciones que puedan afectar al alcance, siendo estás sobre todo para mejoras de la aplicación, ya que la funcionalidad básica de la aplicación de permitir al usuario realizar una inspección no es muy compleja. Entres ellas las más destacables son:

- La gran cantidad de tipos de riesgos que hay y las diferencias que existen entre ellos dificulta el poder tener todos en cuenta.
- No todos los lugares de trabajo están afectados en igual medida por todos los riesgos de la lista, de hecho en algunos de ellos muchos riesgos serán inexistentes. Es decir la variedad de lugares de trabajo.
- El hecho de que todo lo que se añada debe de ser fácil de usar, ya que si es complejo no tendría ningún beneficio para el usuario.

4.4. Metodología Scrum

4.4.1. Creadores del Manifiesto Ágil y objetivos del mismo

El número de autores del manifiesto ágil fue 17: [ma01a]

Kent Beck Mike Beedle Arie van Bennekum Alistair Cockburn Ward Cunningham Martin Fowler James Grenning Jim Highsmith Andrew Hunt Ron Jeffries Jon Kern Brian Marick Robert C. Martin Steve Mellor Ken Schwaber Jeff Sutherland Dave Thomas

La idea principal de este manifiesto es que sus valores no se centran en cosas prácticas, sino que más bien buscan algo abstracto, un cambio de mentalidad en la organización de proyectos que acabe generando una nueva manera de entender las planificaciones de proyectos. Esto basa en cuatro pilares: [ma01a]

- Individuos e interacciones sobre procesos y herramientas.
- El software no solo debe funcionar sino que tiene que tener una amplia documentación.
- Mayor trato con el cliente.
- Respuesta ante el cambio sobre seguir un plan.

Para entender esto mejor, en dicho manifiesto están definidos 12 principios en los que además quedan claros los objetivos de dicho manifiesto: [ma01b]

- Alta prioridad a satisfacer al cliente mediante la entrega temprana y continua de 'software' con valor.

- Se aceptan cambios en los requisitos, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

- Entrega de 'software' funcional frecuente, entre dos semanas y dos meses, preferentemente en el periodo de tiempo más corto posible.

- Los responsables de negocio y los desarrolladores deberán trabajar juntos de forma cotidiana durante todo el proyecto.

- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

- Promueve la conversación cara a cara como mejor método de comunicar con el equipo de desarrollo y entre los miembros de diferentes equipos.

- El 'software' funcionando es la medida principal de progreso, solo se cuentan como avance las funcionalidades conseguidas hasta el momento.

- Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.

- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.

- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es muy importante ponerse siempre en el peor de los casos.

- Cuando un equipo se auto organiza, por lo general acaba encontrando mejores requisitos y realizando diseño de mayor calidad.

- Es importante que el equipo se junte para buscar mejoras aplicando esta estrategia, lo que hará que sean más eficientes a la larga.

Se centra en mejorar las estrategias de desarrollo de software aumentando la comunicación y relación entre todos los integrantes del equipo y a su vez la de estos con los clientes, esto hará que aumente su motivación, haciendo más fácil que el desarrollo acabe siendo un éxito. Además, el hecho de aumentar la relación con los clientes, provocará que se haga mayor hincapié en los intereses de estos, ayudando a que el producto sea más popular y aumentando las posibilidad de este tenga una gran demanda.

El principio se base en darle mayor importancia a las peticiones del cliente y mantenerlo informado de como avanza el desarrollo, mostrándole cada cierto tiempo software funcional y que así pueda dar feedback sobre el estado de dicho software.

4.4.2. Ventajas y desventajas y cuando aplicar cada metodología.

En el ciclo de vida en cascada, las fases se realizan de manera lineal y una única vez. Además, hasta la finalización de una de las fases no comienza la siguiente, las principales ventajas son:

- Implementación sencilla. [4319]
- Cantidad de recursos para la implementación son mínimos. [4319]
- La documentación se produce en cada etapa del desarrollo del modelo de cascada. Esto hace que la comprensión del diseño del producto sea una tarea mucho más sencilla. [4319]

El principal problema que viene de aplicar este ciclo de vida es que genera problemas cuando el resultado de unas de las fases no es el esperado, si una fase falla o tarda más de lo esperado las siguientes no pueden empezar y esto acabará generando problemas mayores.

En el ciclo de vida iterativo, en cada una de las iteraciones se hace una revisión juntos a posibles mejoras del producto, las principales ventajas son:

- Análisis de avances: al fraccionar el proyecto en etapas, el equipo de trabajo tiene la posibilidad de analizar los resultados en cada una de ellas e incorporar mejoras para la entrega final.
- Manejo de riesgos: este modelo de planificación por secciones también permite la evaluación de riesgos que pueden irrumpir durante la ejecución del proceso.
- Retroalimentación: al plantear un desarrollo dosificado, cada fase propicia el diálogo entre el equipo de trabajo, el líder del proyecto y, por supuesto, el cliente.
- Flexibilidad: no sólo en cuanto a la introducción de cambios, sino también en las situaciones en que los objetivos no hayan sido claramente definidos desde el inicio.

En el ciclo de vida ágil, la metodología de desarrollo flexible de un proyecto mediante el cual este se divide en pasos separados para crear así un esquema de trabajo práctico y funcional. Sus principales ventajas son: [EBF19]

- Facilita la búsqueda de errores, lo que hace que estos se encuentren de manera sencilla.
- Mucha importancia a la opinión del cliente.
- Retroalimentación rápida de los usuarios para los que se está haciendo el software.
- Dividir el proyecto en etapas es sencillo, y gracias a esto el equipo puede centrarse de manera individual en cada una de ellas. Esto permite trabajar más rápido.

- La adaptación del proyecto a la larga se hace más fácil. Esto permite que sea sencillo volver a organizar al equipo en relación con los nuevos requisitos y objetivos en caso de que surja un cambio que los haga variar.

El principal problema aquí sería la principal ventaja del cascada, en este caso la implementación es mucho más difícil y requiere de mayor gasto de recursos, es difícil al principio calcular con precisión el tiempo y dinero que va a costar.

Sobre cuando usar cada uno, en proyectos más sencillos y con menor probabilidad de fallo se recomienda usar cascada, ya que al haber menos probabilidades de fallar es más difícil que una de las fases falle, y en el caso de proyectos más complejos y grandes, en los que se cuenta con una mayor cantidad de tiempo y de recursos es más recomendable usar ágil o iterativo, siempre teniendo en cuenta que si el proyecto es muy grande, y requiere el trabajo de un gran número de personas, no será posible utilizar Scrum.

4.4.3. Historias épicas (epic), historias de usuario (story), puntos de historia de usuario, sprints y tareas en Scrum

Las historias de usuarios son las peticiones de clientes, lo deseable es que sean escritas por los propios usuarios y las historias épicas son aquellas que son demasiado grandes como para considerarse una sola historia, estas se podrían dividir en varias historias de usuario para su realización.

Los puntos de historia son un valor que representa la carga de una tarea, cuantos más puntos se le asignen a una tarea mayor carga de trabajo tendrá.

Las tareas en Scrum suelen ser por lo general divisiones de una historia de usuario y se les suele asignar a un trabajador, el tiempo de realización se suele procurar que equivalga como mucho a una jornada de trabajo, con estas tareas se puede seguir el avance de una historia.

Los sprints son miniproyectos que suelen ser de 2 a 4 semanas, la idea es dividir un proyecto grande en un número de entregas, para así ir teniendo en cada uno de los sprints un producto, que aumentará su funcionalidad por cada sprint que se realice, y así tener continuamente algo que enseñar a los clientes.

4.4.4. Cuando usar Scrum

Sobre la pregunta de en que proyectos es necesario el uso de Scrum, el caso más claro sería en sistemas complejos en los cuales los resultados se vuelven impredecibles y no podemos identificar una solución al principio, solo podemos examinar los resultados y adaptarnos. Se requieren niveles altos de creatividad, innovación y comunicación, el uso de Scrum, con sus ciclos de inspección y adaptación, es un marco de construcción ideal.

En nuestro caso, al ser una aplicación es ideal ya que después de cada sprint se pueden hacer nuevas sugerencias que ayuden a adaptar la aplicación lo máximo posible a las peticiones del cliente.

Otro caso aunque no tan claro sería el de sistemas complicados, pero en este caso sería mucho más fácil utilizar cascada.

4.4.5. Relación entre los tres artefactos y los tres roles de Scrum

Los roles en Scrum son:

- **Scrum Máster:** Pese a que su rol es el de líder del equipo, su función más importante es la de encargarse de eliminar los problemas que obstaculicen la consecución de los objetivos. Delega las tareas en sus trabajadores para que éstos puedan auto-organizarse, y puedan alcanzar un nivel de coordinación y colaboración exitoso. [Bar17]
- **Product Owner:** Es el encargado de transmitir las peticiones del cliente a los encargados del desarrollo. [Bar17]
- **Scrum Team:** Los pertenecientes a este grupo son los que se encargan de ejecutar las acciones previstas. Para el que el proceso de desarrollo sea un éxito, lo ideal es que sean un mínimo de 5 personas hasta un máximo de 9. [Bar17]
- **Usuarios:** Es el destinatario final del producto. Pese a ello, es importante saber diferenciarlo con el cliente, que es quien solicita el producto. Los clientes son los que lo compran mientras que los usuarios son los que van a usarlo. [Bar17]

Los artefactos son:

- **Lista de producto:** Es el documento que hace de pilar central en un proyecto Scrum. En él se reflejan todos los elementos necesarios para la ejecución del mismo, lo que lo convierte en la principal referencia en lo que se refiere a plantear y realizar cambios. [Bar17]
- **Lista de objetivos pendientes del sprint:** Esta lista entra en juego cuando algunos de los objetivos han tenido que ser dejados de lado por el momento y se tiene que pasar a una nueva fase. En el caso de que se de esta situación, haremos uso de la lista para no dejarlos de lado y añadirlos en un futuro. [Bar17]
- **Incremento:** Método para medir el progreso en cada etapa. Si estamos utilizando Scrum, es indispensable que cada iteración tenga un incremento, en caso contrario podríamos considerar que se ha fallado esa etapa. En resumen, podríamos decir que el producto final es la suma de todos los incrementos de las diferentes etapas. [Bar17]

El como se relacionan los roles con los artefactos, la lista de producto está relacionada con el Scrum Master, el cual hace uso de ella para ver que tareas hay y hacer una división entre las diferentes listas, asignando pequeñas tareas a cada uno de los integrantes.

La lista de objetivos pendientes está mayormente relacionada con el Scrum team, puesto que son los integrantes de este rol los que realizan las tareas serán a su vez los encargados de ir añadiendo a esta lista los objetivos que no logren cumplir.

El incremento se calcula cuando se reúnen los diferentes integrantes del grupo y cada uno indique en que ha avanzado, con la suma de todos los avances se calculara el incremento de esa etapa.

Capítulo 5

Análisis

5.1. Búsqueda de requisitos

Lista de requisitos:

- El sistema permitirá al usuario inspector registrarse.
- El sistema permitirá al usuario, (tanto inspector como administrador) acceder con su email y su contraseña.
- El sistema permitirá al usuario inspector añadir una nueva inspección de riesgos laborales.
- El sistema permitirá al usuario inspector añadir nuevos riesgos a una inspección.
- El sistema permitirá al usuario inspector evaluar un riesgo añadido a una inspección.
- El sistema permitirá a los usuarios, (tanto inspector como administrador) modificar su perfil, dando la posibilidad de cambiar nombre de usuario, email, contraseña, foto de perfil y teléfono de contacto.
- El sistema permitirá a los usuarios, (tanto inspector como administrador) restablecer su contraseña en caso de que la olviden.
- El sistema permitirá al usuario inspector gestionar sus inspecciones, tanto las que tienen en curso, como las finalizadas y las ya cerradas.
- El sistema deberá permitir al usuario inspector generar un archivo excel con la inspección.
- El sistema permitirá al usuario administrador acceder a la información de cada uno de los inspectores, dándole a su vez la posibilidad de dar de baja a cualquiera de ellos.
- El sistema deberá tener implementados métodos de seguridad para el acceso.

5.2. Diagrama de casos de uso

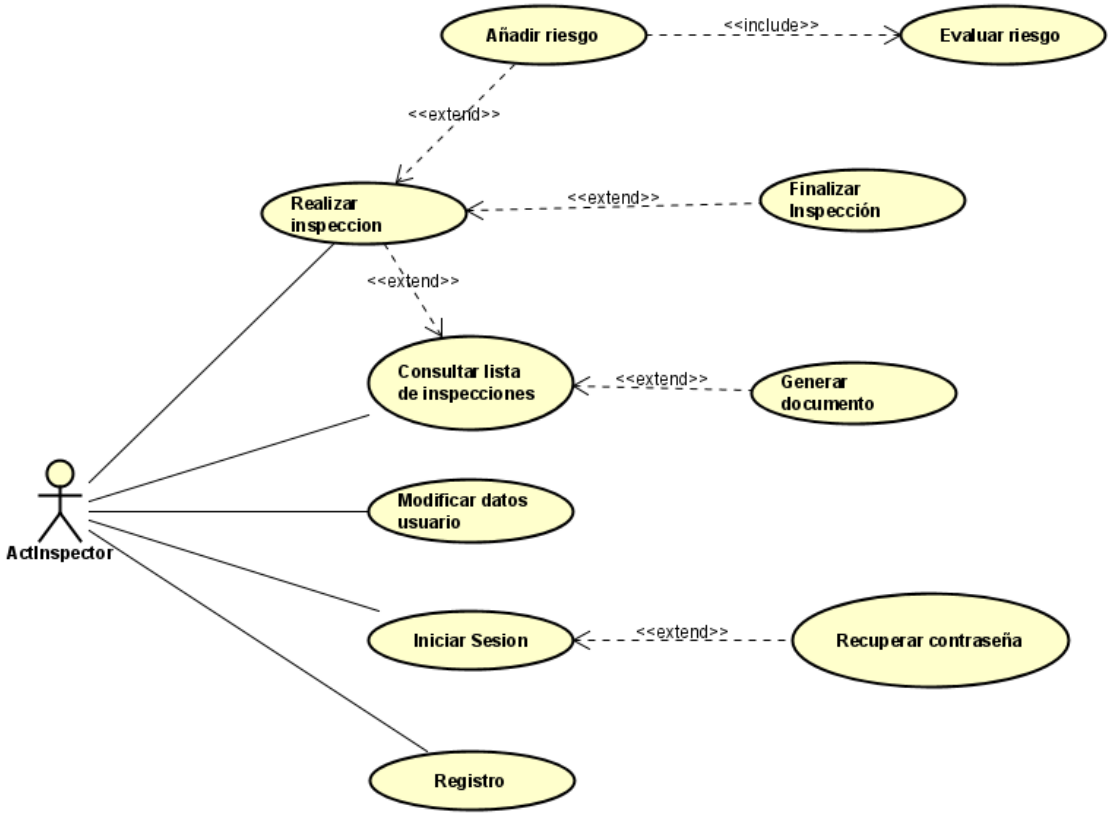


Figura 5.1: casos de uso del usuario inspector

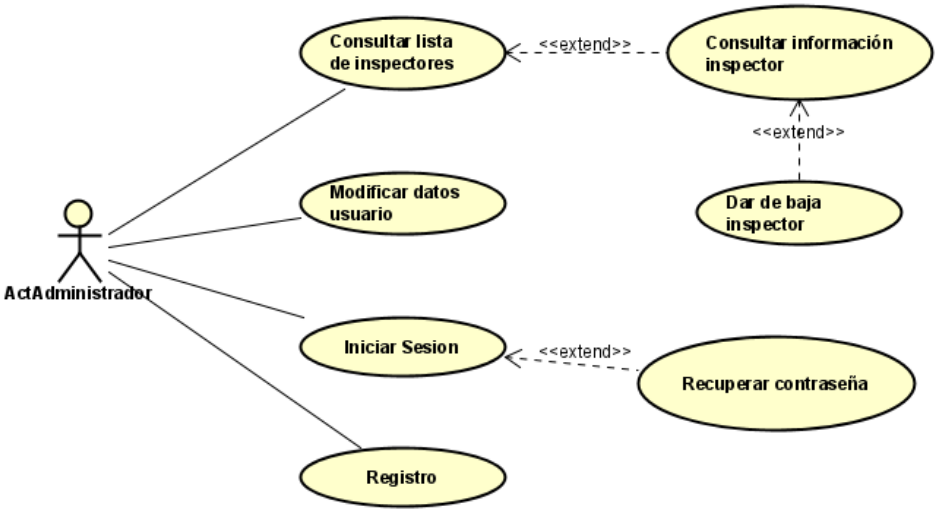


Figura 5.2: casos de uso del usuario administrador

5.3. Modelo de dominio

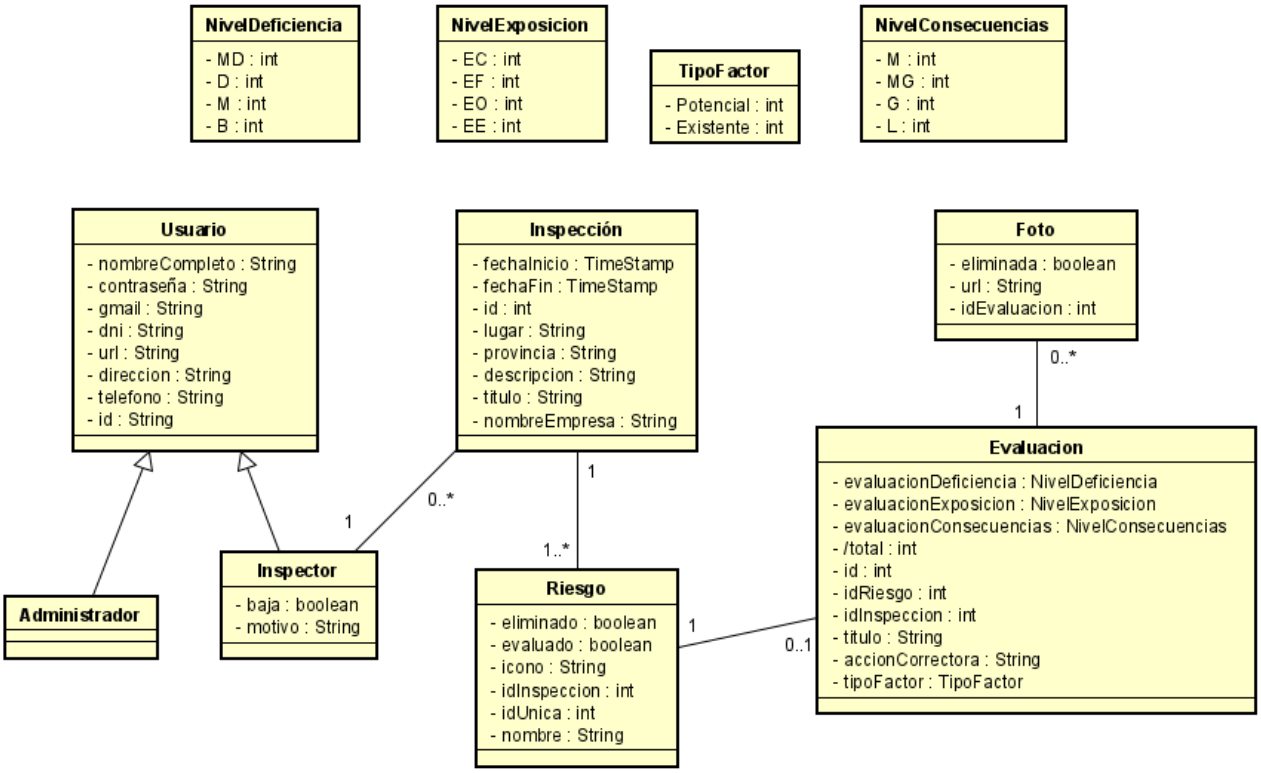


Figura 5.3: diagrama de clases

5.4. Usuarios

En la aplicación se diferenciarán dos tipos de usuarios: inspector y administrador.

Al registrarse en la aplicación el usuario será de tipo inspector, el usuario administrador será único y ha sido añadido manualmente a la base de datos. El inspector es el usuario objetivo de la aplicación, y por lo tanto será el que tenga acceso a la creación y gestión de inspecciones, mientras que el usuario administrador servirá para gestionar a los inspectores, pudiendo acceder a sus datos y teniendo la capacidad de darlos de alta y de baja.

Ambos usuarios tendrán igual acceso a las pantallas de gestión de usuario, por lo que la recuperación y modificación de datos será similar para los dos. Lo único que los diferencia es la funcionalidad principal de la aplicación.

En resumen, la funcionalidad del usuario administrador es tener acceso a todos los inspectores y poder gestionarlos, y la del usuario inspector realizar inspecciones y gestionar las que ya tiene realizadas.

5.5. Inicio Sesión

El acceso a la aplicación se hará a través de una pantalla de login con un usuario/email y una contraseña, por lo que para que el usuario pueda acceder requerirá previamente de tener una cuenta registrada. Además, también será necesaria una pantalla de registro en la que el usuario tendrá que introducir una serie de datos (email, contraseña, ”usuario”, nombre completo, dni, número de teléfono y dirección) para poder ser identificado junto con un usuario/emaill y una contraseña que utilizará para el acceso.

Los errores que se le podrán mostrar al usuario (es decir que estarán contemplados) serán:

- En ambas pantallas todos los campos de texto son necesarios, por lo que en caso de que se deje alguno en blanco, al darle al botón notificará al usuario que deben rellenarse todos los campos.
- En la pantalla de registro, los campos de contraseña y confirmación contraseña deberán de ser iguales, por lo que en caso de que no lo sean, se le notificará al usuario indicándole que ambos campos deben tener el mismo contenido.
- En la pantalla de login, en caso de que el email no pertenezca a un usuario registrado se le notificará al usuario, indicándole que no existe un usuario registrado con ese email.
- En la pantalla de login, en caso de que la contraseña no coincida con la de ese email, se le notificará al usuario que la contraseña introducida para ese email no es correcta.
- En la pantalla de registro, en caso de que se intente registrar un usuario con un email que ya está registrado, se le notificará indicándole que ese email ya está registrado.

También se incluirá un método de recuperación de usuario y contraseña que se podrá realizar a través del email, el usuario tendrá que introducir el email con el que se haya registrado, y se le enviará un link de recuperación al correo a través del cual podrá cambiar tanto la contraseña, como el usuario (en caso de que este se acabe usando como método de acceso).

Para la seguridad de los usuario registrados, se utilizarán los métodos ya existentes de Firebase.

Bocetos para la parte de inicio de sesión, tanto la pantalla de registro como la de login:

Figura 5.4: boceto pantalla de login

Figura 5.5: boceto pantalla de registro

5.6. Gestión de usuarios

Una vez iniciado sesión en la aplicación, se redireccionará a una página principal a través de la cual dependiendo del tipo de usuario, podrá acceder a una funcionalidad u otra (en caso del súper usuario a la gestión de inspectores, y en caso de inspector a la creación o gestión de inspecciones).

Como la gran mayoría de aplicaciones, tendrá también una serie de pantallas de gestión de usuarios:

- Perfil: Donde podrá ver sus datos y agregar, o cambiar en caso de que ya tenga, una imagen de perfil. Al intentar acceder a la cámara o a la galería deberá de pedir permisos al usuario.
- Modificar perfil: Donde podrá modificar sus datos, obviamente los que sean posibles: dirección, email, número de teléfono... Hay algunos datos que no se pueden modificar, como por ejemplo el dni. Para la modificación, se presentarán unos cuadros de texto que tendrán como valores predefinidos los datos actuales del usuario, por lo que si este le da a guardar sin modificar ninguno sus datos se mantendrán iguales.
- Cambio de contraseña: Por lo estudiado en aplicaciones móviles, es recomendable separar el cambio de contraseña de la modificación del resto de datos. La manera de proceder será la que se suele utilizar en estos casos, introducir la contraseña actual para confirmar que lo está utilizando el usuario, y posteriormente dos campos de texto, uno para que introduzca la nueva, y el otro para confirmarla. Los errores que podrán saltar de notificación al usuario son:
    - En caso de que alguno de los campos de texto se deje vacío, se le notificará al usuario exigiéndole que los rellene.
    - En caso de que la contraseña introducida como actual no sea correcta, se notificará al usuario, indicándole que esa contraseña no es correcta.
    - En caso de que la contraseña nueva y su confirmación no coincidan, se notificará al usuario indicándole que esos dos campos deben de coincidir.
    - En caso de que la contraseña nueva sea igual a la actual se notificará al usuario indicándole que la contraseña nueva tiene que ser diferente a la actual.
- Cerrar sesión: Esto no será una pantalla, sino un botón, o una opción del menú. Servirá para que cuando el usuario arranque la aplicación no entre directamente con el usuario logeado.

Con esto, el usuario podrá gestionar su propia cuenta y mantener actualizados sus datos personales pudiendo realizar cambios sobre ellos en todo momento.

Bocetos para la parte de gestión de usuarios, pantallas de: perfil, modificar perfil, cambio de contraseña y pantalla principal.

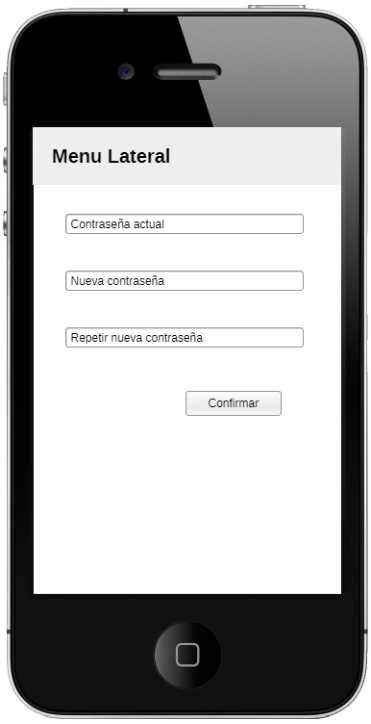


Figura 5.6: boceto pantalla cambiar de contraseña

Figura 5.7: boceto pantalla página principal

Figura 5.8: boceto pantalla cambiar de perfil

Figura 5.9: boceto pantalla cambiar de modificar perfil

5.7. Creación de una inspección

Como se mencionó en la anterior sección, una vez inicie sesión el usuario, se encontrará inicialmente en la pantalla principal, en la cual se le presentarán dos opciones, la primera de ellas será añadir una inspección:

Una vez haya accedido a esta opción, tendrá que rellenar un formulario que le creará una inspección, para ello, tendrá que introducir un titulo para la inspección, el nombre de la empresa en la que se va a realizar dicha inspección, una ubicación (la cual podrá añadir utilizando un mapa o introduciendo un lugar de manera manual), una pequeña descripción, y la Provincia en la que se encuentra la empresa (la aplicación está pensada para ser utilizada en España, por lo que se le presentarán solo las Provincias de España para introducir).

En caso de que el usuario no rellene todos los campos (puesto que son necesarios), se le notificará con un error debajo de cada campo de texto vació, especificando que son necesarios.

Cuando haya rellenado todos estos campos, se creará una nueva inspección asignada al usuario que la ha creado, con estado inicial "**en curso**", marcando la fecha de inicio de la inspección a la actual (es decir, a la del momento en el que se crea) y dejando en blanco la fecha de fin.

Bocetos para la parte de creación de una inspección, pantallas de: agregar nueva inspección y mapa.

Figura 5.10: boceto pantalla agregar nueva inspección

Figura 5.11: boceto pantalla mapa

5.8. Añadir un riesgo

Una vez se ha añadido una inspección, o se ha accedido a una ya creada, se redireccionará a la pantalla en la que se mostrarán los riesgos encontrados y evaluados de esa inspección, dando la posibilidad al usuario de finalizar la inspección, o de agregar un nuevo riesgo.

Antes de comenzar a hablar sobre como se añadirá un riesgo es conveniente decidir primero de que manera se van a cargar todos ellos, ya que actualmente hay más de 50 tipos de riesgos y cada tipo tiene entre 1 y 4 riesgos.

Por ello la manera a proceder será: Cargar dos archivos Json, uno con las categorías de riesgos, teniendo una id, una imagen y un titulo, y otro con los riesgos específicos que tendrá una id, una id del padre (es decir la categoría a la que pertenece), una imagen y un titulo. Teniendo estos dos Json cargados solo nos quedará mostrarlos en pantalla para que el usuario pueda elegir el riesgo que ha encontrado (cabe destacar que está es una lista fija por lo que el riesgo encontrado tiene que encajar con uno de los de la lista.

Ya teniendo los dos Json cargados, cuando el usuario presione el botón de agregar un nuevo riesgo se le mostrará una lista de cartas con todas las categorías que previamente han sido cargadas del Json, de cada una de ellas se podrá ver el titulo y la imagen. Una vez el usuario escoja una categoría se le mostrarán los riesgo pertenecientes a esa categoría, si la categoría tiene id=05 se mostrarán todos los riesgos que tiene como id padre la 05. Una vez seleccione uno de los riesgos pasará a la pantalla de evaluación para evaluar el riesgo encontrado.

Bocetos para la parte de creación de agregar riesgo, pantallas de: lista de riesgos evaluados y seleccionar categoría, la de añadir riesgo no se incluye porque es como la de seleccionar categoría pero con menos cartas:

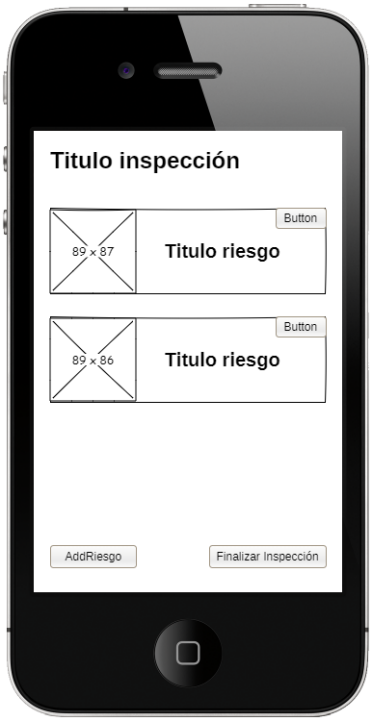


Figura 5.12: boceto pantalla lista de riesgos por evaluar

Figura 5.13: boceto pantalla seleccionar categoría del riesgo

5.9. Evaluar un riesgo

Una vez añadido el riesgo que se ha encontrado se redireccionará a la pantalla de evaluación de riesgo, en la cual se mostrará, al igual que en la vista de añadir riesgo, un formulario con una serie de campos por rellenar y un texto que indique el nombre del riesgo que se está evaluando.

Los campos por rellenar serán: un titulo para la evaluación, especificar si es un riesgo potencial o si es existente, el nivel de deficiencia, el nivel de exposición, el nivel de consecuencia y la acción correctora. Además, se presentará al usuario la posibilidad de agregar fotos para dicha evaluación, dándole la opción de cargarlas desde la cámara, o desde la galería. Una vez haya rellenado los campos obligatorios, (todos salvo las fotos) podrá finalizar la evaluación. En caso de que no se rellene alguno de los campos, se notificará al usuario con un error debajo de cada uno de los campos de texto que los deje vacíos especificando que son necesarios.

Al finalizar la evaluación, el riesgo previamente añadido cambiará de estado a evaluado, y se calculará en función de los tres niveles un valor final para dicho riesgo (cuanto más alto sea este valor, más crítico será el riesgo). Una vez añadido, se volverá a la pantalla de lista de riesgos de la inspección y se mostrará dicha lista con el nuevo riesgo añadido y un indicador establecido en función de su criticidad.

Para la clasificación según su criticidad se tomarán las siguientes referencias:

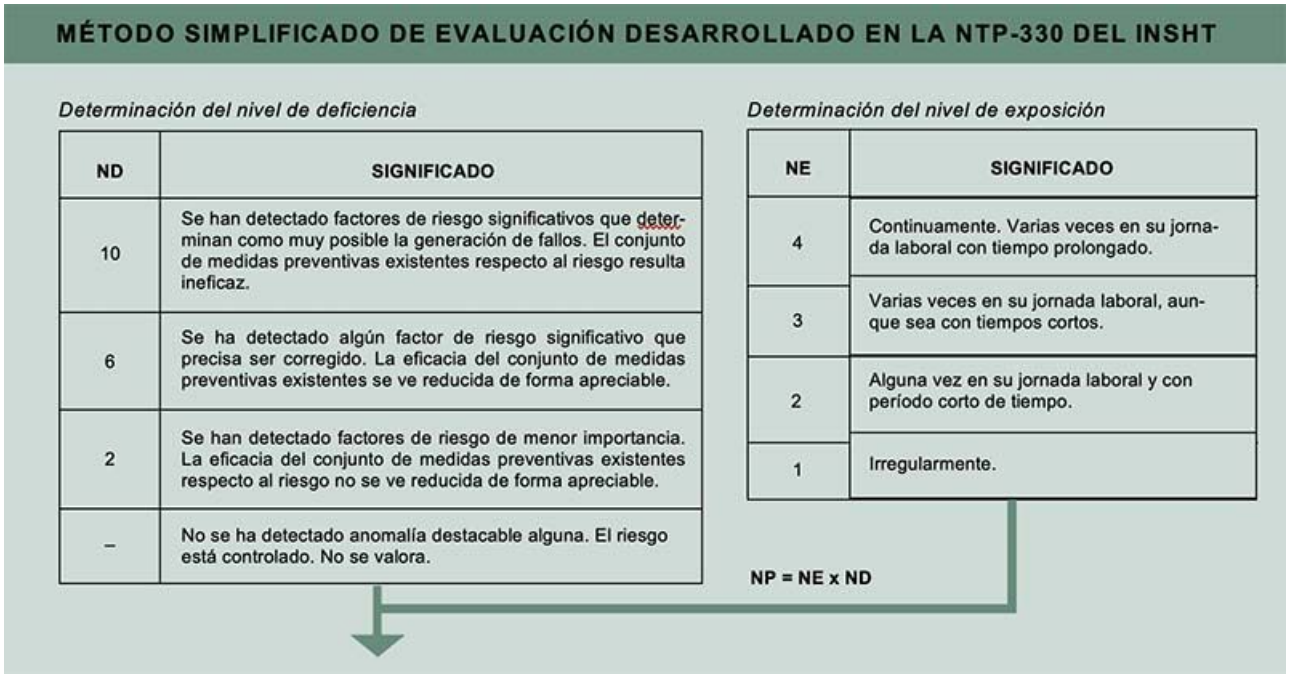
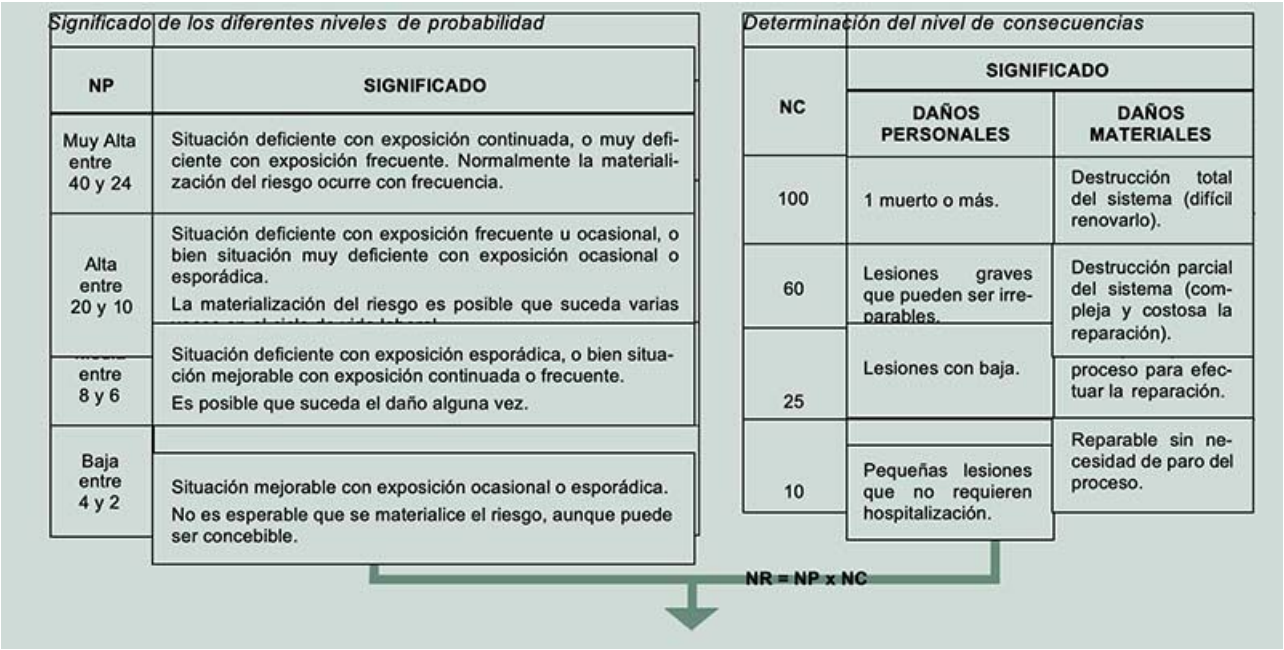


Figura 5.14: paso 1 cuenta



Significado de los diferentes niveles de probabilidad

NP	SIGNIFICADO
Muy Alta entre 40 y 24	Situación deficiente con exposición continuada, o muy deficiente con exposición frecuente. Normalmente la materialización del riesgo ocurre con frecuencia.
Alta entre 20 y 10	Situación deficiente con exposición frecuente u ocasional, o bien situación muy deficiente con exposición ocasional o esporádica. La materialización del riesgo es posible que suceda varias
entre 8 y 6	Situación deficiente con exposición esporádica, o bien situación mejorable con exposición continuada o frecuente. Es posible que suceda el daño alguna vez.
Baja entre 4 y 2	Situación mejorable con exposición ocasional o esporádica. No es esperable que se materialice el riesgo, aunque puede ser concebible.

Determinación del nivel de consecuencias

NC	SIGNIFICADO	
	DAÑOS PERSONALES	DAÑOS MATERIALES
100	1 muerto o más.	Destrucción total del sistema (difícil renovarlo).
60	Lesiones graves que pueden ser irre- parables.	Destrucción parcial del sistema (compleja y costosa la reparación).
25	Lesiones con baja.	proceso para efectuar la reparación.
10	Pequeñas lesiones que no requieren hospitalización.	Reparable sin necesidad de paro del proceso.

NR = NP x NC

Figura 5.15: paso 2 cuenta

NIVEL DE INTERVENCIÓN	NR	SIGNIFICADO
I	4000-600	Situación crítica. Corrección urgente.
II	500-150	Corregir y adoptar medidas de control.
III	120-40	Mejorar si es posible. Sería conveniente justificar la intervención y su rentabilidad.
IV	20	No intervenir, salvo que un análisis más preciso lo justifique.

Figura 5.16: nivel de intervención

Boceto para la parte de evaluación, pantalla de evaluar riesgo:

Figura 5.17: boceto pantalla evaluación

En esta pantalla se tiene que plantear si añadir la ubicación de cada riesgo, todavía no se sabe si es un requisito o no es necesario por lo que no lo vamos a incluir en el boceto inicial y en caso de que se requiera añadirlo se hará solo sobre el diseño.

5.10. Lista inspecciones

Como se mencionó anteriormente, una vez inicie sesión el usuario, se encontrará inicialmente en la pantalla principal, en la cual se le presentarán dos opciones, la segunda de ellas será consultar la lista de inspecciones que ha realizado dicho usuario:

Una vez haya accedido a esta opción podrá ver todas las inspecciones que ha realizado ordenadas y clasificadas en función de su estado (en curso, finalizadas, cerradas), se mostrarán primero las que están en curso, luego las finalizadas y por último, las cerradas, ordenándolas entre sí en función de su fecha de inicio. En caso de que la inspección este en curso al seleccionar una de las inspecciones llevará a las pantallas de lista de riesgos añadidos, y en caso de que su estado sea finalizada o cerrada he decidido plantear dos opciones y decidir durante el desarrollo cual de ellas es más óptima:

- La primera de ellas sería que se pueda acceder a cada una de las inspecciones, lo que llevaría a una nueva pantalla que tendría un pequeño resumen de la inspección y la posibilidad de cambiar su estado de finalizada a cerrada en caso de que no lo este, y poder descargar un excel con toda la información de esa inspección.

- La segunda sería similar a la primera, pero en vez de llevar a una nueva pantalla que las opciones de descargar excel y cambio de estado estuvieran directamente en la misma pantalla (la de lista de inspecciones).

Importante mencionar que la idea de esto es guardar un histórico de las inspecciones que ha realizado un usuario, por lo que aunque su estado sea finalizada se guardará igualmente, pudiendo descargar un excel con su información en cualquier momento. Es posible que se permita al usuario administrador borrar las inspecciones una vez estén aseguradas en otro lado para liberar así la interfaz del inspector.

Figura 5.18: boceto pantalla lista de inspecciones

5.11. Funcionalidad súper usuario

Al igual que con los usuarios de tipo inspector, cuando el usuario administrador inicie sesión se le redireccionará a la página principal, dándole también la oportunidad de acceder a su perfil y de poder modificar tanto sus datos personales como sus datos de acceso, la diferencia es que mientras que el usuario inspector tendrá dos opciones en la página principal (inspecciones y añadir inspección), el administrado tendrá solo una que será inspectores.

Si accede a esta opción se encontrará con una lista de inspectores en la que se mostrarán en tarjetas con su nombre completo y dni todos los inspectores que hay registrados en la base de datos, pulsando en cada una de las tarjetas podrá acceder a una pantalla en la que se mostrará una mayor descripción de dicho inspector junto con la posibilidad de darlo de baja.

Figura 5.19: boceto pantalla lista de inspectores

Figura 5.20: boceto pantalla información de inspector

Capítulo 6

Diseño y implementación

6.1. Diagramas de actividades

6.1.1. Inicio sesión

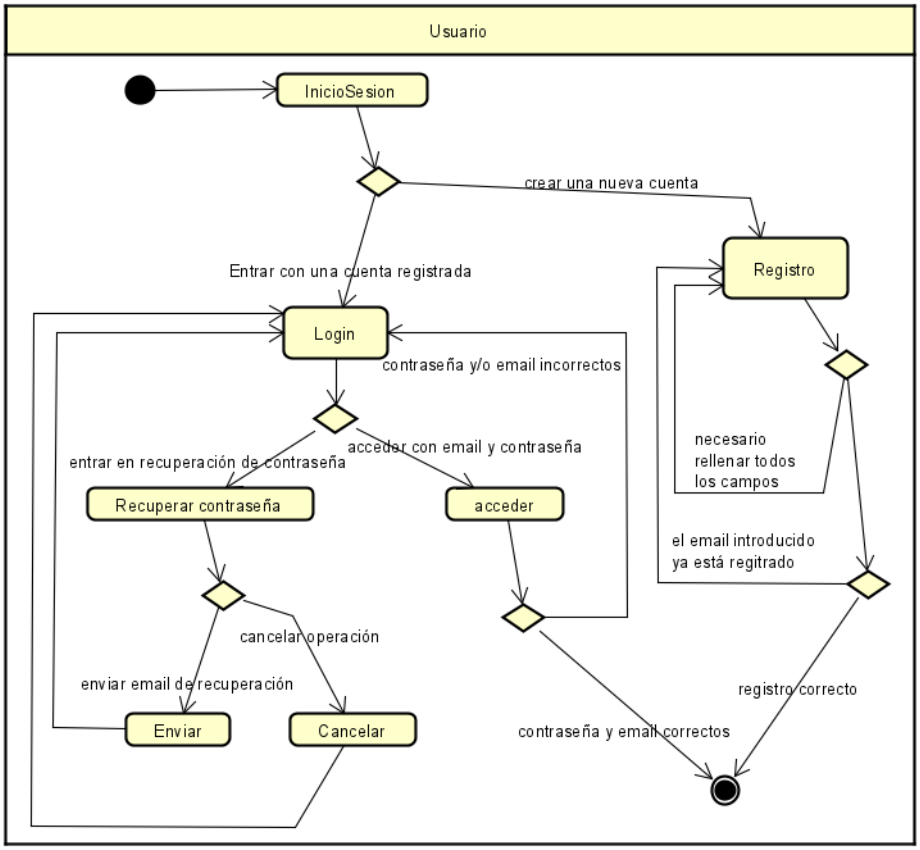


Figura 6.1: diagrama de actividad inicio de sesión

6.1.2. Modificar datos usuario

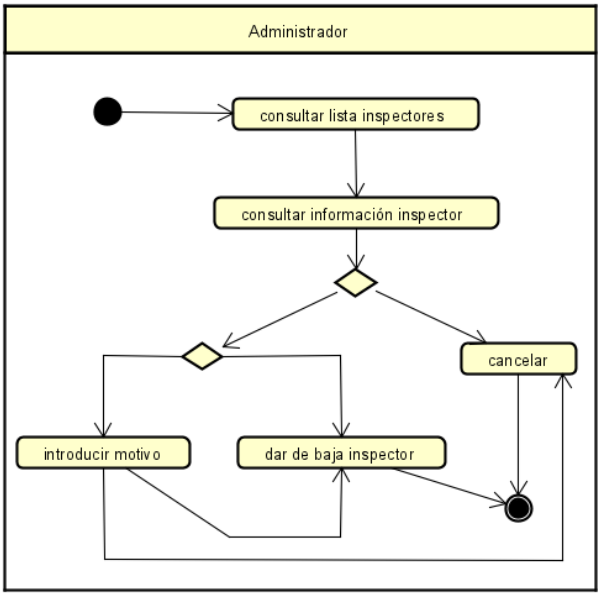


Figura 6.2: diagrama de actividad dar de baja inspector

6.1.3. Dar de baja a un inspector

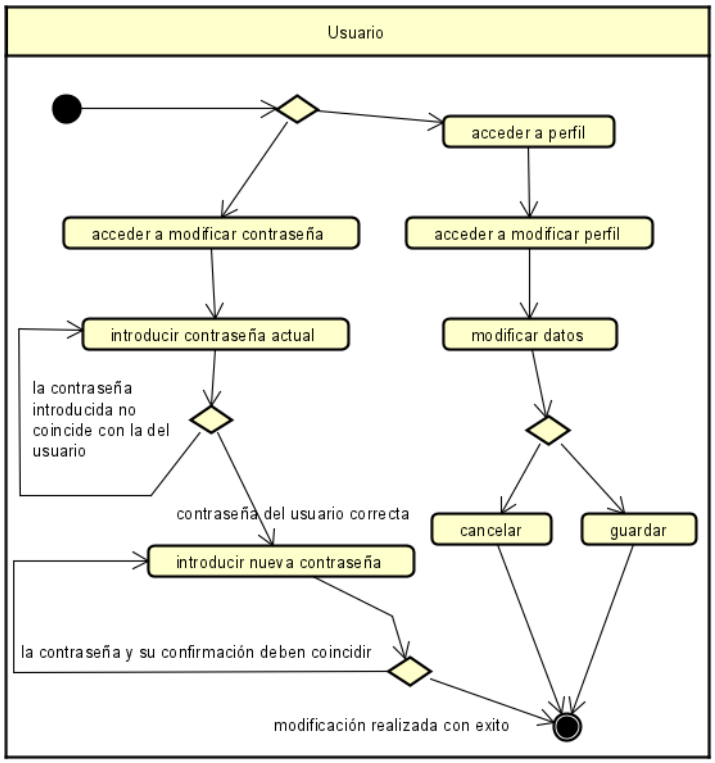


Figura 6.3: diagrama de actividad modificar datos usuario

6.1.4. Realizar una inspección

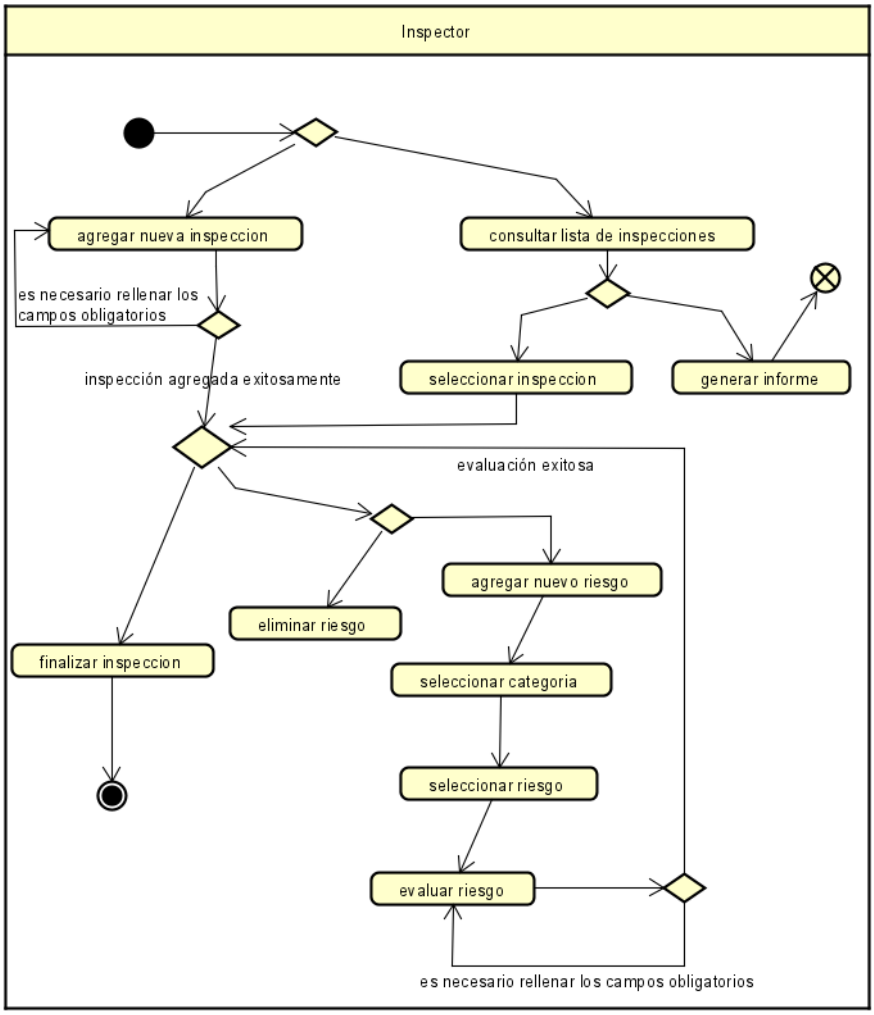


Figura 6.4: diagrama de actividad realizar una inspección

Una vez vistos los diagramas de actividad referentes a los principales casos de uso de la aplicación, pasaremos a ver el diagrama de paquetes (que nos ayudará a entender como va a estar distribuido el proyecto), y el diagrama de despliegue, que pese a que en el caso de esta aplicación es algo simple es muy útil para entender la arquitectura del sistema.

Importante mencionar que para evitar extenderme mucho con los diagramas de actividades, algunos de ellos contienen más de un caso de uso, como se ve en el diagrama de inicio de sesión que incluye tanto el caso de uso de login, como el de registro.

6.1.5. Diagrama de paquetes

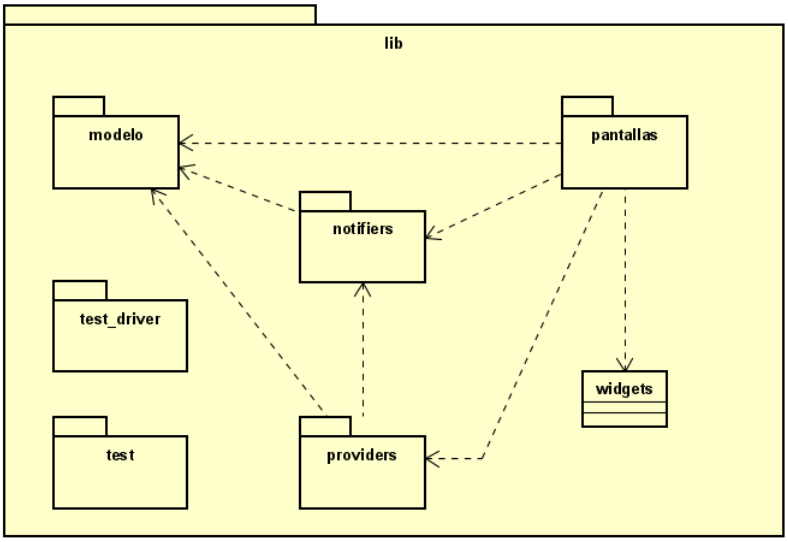


Figura 6.5: diagrama de paquetes

6.1.6. Diagrama de despliegue

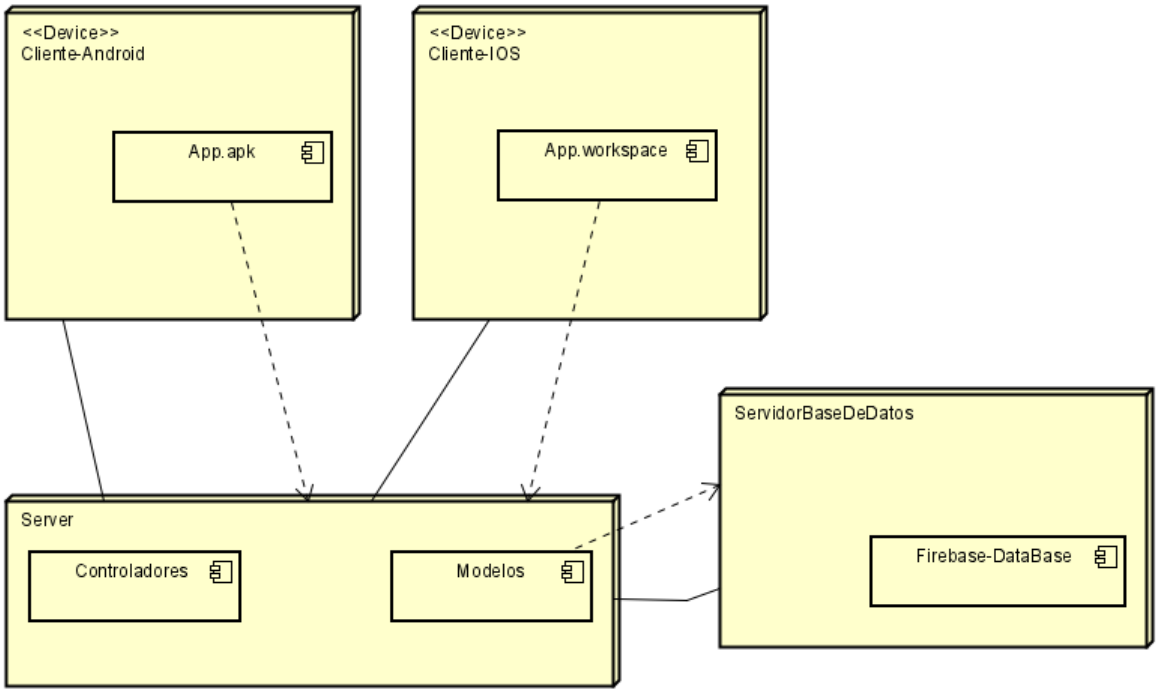


Figura 6.6: diagrama de despliegue

6.2. Introducción

Importante mencionar que en esta sección no se repetirán las explicaciones de todos los componentes y de su lógica, es decir, en caso de que para el diseño de la página de Login se utilice un Widget que se está utilizando de igual manera para la página Registro solo cambiando los parámetros, se explicará sólo en la sección de diseño de Login, en la de Registro simplemente se hará una mención. Debido a esto, las primeras pantallas de las que hable serán mucho más extensas que a medida que vaya avanzando.

También es importante de manera previa ver las dos posibilidades que tenemos a la hora de crear un Widget: con o sin estado.

Un Widget sin estado es un Widget estático, es decir, que no puede sufrir cambios. Icon, IconButtony Textson ejemplos de Widgets sin estado. [Flua]

Por otro lado tenemos a los stateful Widgets, o Widgets dinámicos. Éstos sí que pueden sufrir cambios debidos seguramente a una acción del usuario. Un claro ejemplo puede ser un checkbox, si el usuario pulsa sobre el su aspecto cambiará. Otro ejemplo puede ser un TextField, porque el usuario puede escribir sobre él y por tanto cambiarlo. Este tipo de Widgets tienen un objeto «state» asociado para gestionar su estado (de ahí el nombre). Si el estado cambia el Widget volverá a pintarse en la pantalla con la apariencia actualizada. Esto es muy importante porque cuando creemos nuestro propio stateful Widget tendremos también que crear otra clase state. [Gar]

Estos últimos suelen ser aquellos en los que se esta cargando información de la base de datos y mostrando por pantalla, los que utilizan por ejemplo StreamBuilders o los que llaman en algún momento a la función setState. En todos ellos la interfaz puede cambiar.

6.3. Patrón Provider

Por último, antes de empezar con el diseño de las diferentes funcionalidades de la aplicación, vamos a ver el patrón que se ha utilizado para la realización del código de la aplicación (no es el único patrón utilizado pero si el principal, ya que toda la aplicación está programada siguiendo la estructura que este plantea) controlar lo que se muestra por pantalla y sus cambios a través de Notifiers.

Provider es un manejador de estado. Nos permite la comunicación entre Widgets centralizando la información en una clase, la cual notifica el cambio de estado para que se redibujen los Widget (que están escuchando este Provider), mostrando los últimos cambios en la información. [Mon21]

Puede que no sea tan sofisticado como el patrón BLoC, pero para aplicaciones de tamaño pequeño/medio puede ser válido. Cuando mecanizamos el proceso del desarrollo de nuestras aplicaciones, se simplifican mucho las cosas aplicando este patrón. [Mor20]

Elementos del patrón Providers:

- **ChangeNotifier:** La clase o clases que contienen el modelo deben heredar de esta clase. Desde esta clase avisaremos a los listeners cuando el modelo cambie mediante

el método ”notifyListeners”. [Mor20]

- **ChangeNotifierProvider:** Tiene dos parámetros en su construcción: [Mor20]
  - **create:** En donde instanciaremos el modelo.
  - **child:** Los Widgets que cuelguen de aquí, serán notificados cuando invoquemos el método ”notifyListeners”.
- **Consumer:** A la hora de recuperar el modelo para su consulta, o modificación desde los diferentes Widgets de nuestra aplicación usaremos el Widget Consumer. Eventualmente podemos usar Provider.of(T). [Mor20]

Como podemos ver en la imagen de la siguiente página, el patrón se basa en mantener unos Notifiers de los cuáles podremos mantener actualizado su valor.

Para ello, podemos ver el caso de la imagen, supongamos que tenemos dos pantallas en las que se modifica el mismo modelo. La idea es que si se modifica en la pantalla uno, en la pantalla dos se actualice el cambio realizado en la otra, para ello, cuando se realiza un cambio se notifica al Provider, lo que provoca que este invoque el método ”notifyListeners” y el cambio quede presente en el Notifier en cualquiera de las pantallas. Esto implica que el cambio de la pantalla uno se mantiene en el de la dos.

La idea es que el patrón Provider nos permita compartir la información entre varias pantallas diferentes, ahora veremos una imagen con un ejemplo que nos ayude a entender esto.

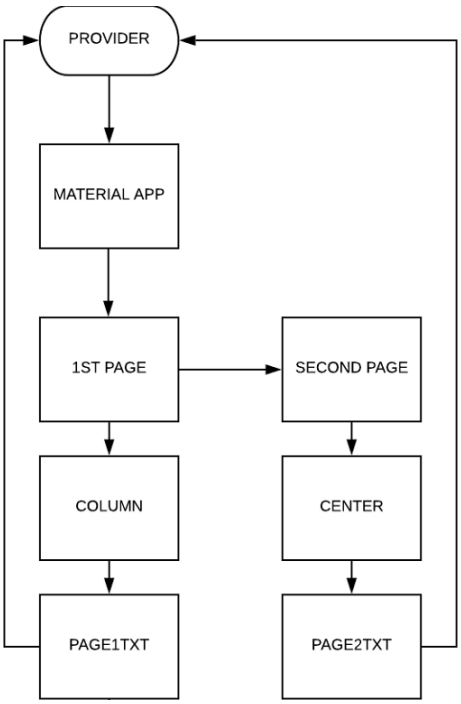


Figura 6.7: ejemplo Provider

Para acabar de ver esto vamos a tratar algunos ejemplos de mi código:

```
class InspeccionNotifier with ChangeNotifier{
  List<Inspeccion> _inspeccionList=[];
  List<Provincia> _provinciaList=[];
  Inspeccion _currentInspeccion;

  UnmodifiableListView<Inspeccion> get inspeccionList =>
  UnmodifiableListView(_inspeccionList);
  UnmodifiableListView<Provincia> get provinciaList =>
  UnmodifiableListView(_provinciaList);

  Inspeccion get currentInspeccion => _currentInspeccion;

  set inspeccionList(List<Inspeccion> inspeccionList){
    _inspeccionList=inspeccionList;
    notifyListeners();
  }

  set provinciaList(List<Provincia> provinciaList){
    _provinciaList=provinciaList;
    notifyListeners();
  }

  set currentInspeccion(Inspeccion inspeccion){
    _currentInspeccion=inspeccion;
    notifyListeners();
  }
}
```

Listing 6.1: Clase Notifier

En este fragmento de código se esta creando una clase Providers para el modelo inspección, de esa manera al notificar a toda la aplicación cual es la inspección actual, será la misma para todos los lugares en los que se este usando el Notifier inspección.

Por entenderlo sería algo así:

```
addInspeccion(Inspeccion i, InspeccionNotifier inspeccionNotifier)
  async {
  Map<String, dynamic> demoData={
    "id": i.getId(),
    "descripcion": i.getDescripcion(),
    "titulo": i.getTitulo(),
    "fechaFin": i.getFechaFin(),
    "fechaInicio": i.getFechaInicio(),
    "provincia": i.getProvincia(),
    "lugar": i.getLugar(),
    "estado": i.getEstado(),
    "nombreEmpresa": i.getNombreEmpresa(),
  };
  CollectionReference collectionReference =
      FirebaseFirestore.instance.collection('inspeccion');
  collectionReference.add(demoData);
  inspeccionNotifier.currentInspeccion=i;
}
```

Listing 6.2: Inicializar el Notifier

Una vez se invoca esta función al añadir una nueva inspección, cuando se acceda a otra pantalla que este invocando a inspeccionNotifier.currentInspeccion tendrá el valor que se le ha dado en esta función, por ejemplo:

```
void initState() {
  súper.initState();
  UsuarioNotifier userNotifier =
      Provider.of<UsuarioNotifier>(context, listen: false);

  if (userNotifier.currentUsuario != null) {
    _nombreController.text=userNotifier.currentUsuario.nombreCompleto
  ;
    _numeroController.text=userNotifier.currentUsuario.telefono;
    _emailController.text=userNotifier.currentUsuario.email;
  }
}
```

Listing 6.3: Utilizar el Notifier parte 1

Por último, mencionar que para que funcione tendremos que envolver nuestro Materialaplicación en un Widget llamado ChangeNotifierProvider. Para ello, en vez de hacerlo en todas las pantallas podemos declararlo de la siguiente manera en la clase main:

```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeaplicación();
  runaplicación(MultiProvider(
      Providers: [
        ChangeNotifierProvider(
          create: (context) => AuthNotifier(),
        ),
        ChangeNotifierProvider(
          create: (context) => UsuarioNotifier(),
        ),
        ChangeNotifierProvider(
          create: (context) => InspeccionNotifier(),
        ),
        ChangeNotifierProvider(
          create: (context) => RiesgoNotifier(),
        ),
        ChangeNotifierProvider(
          create: (context) => SubRiesgoNotifier(),
        ),
        ChangeNotifierProvider(
          create: (context) => RiesgoInspeccionNotifier(),
        ),
        ChangeNotifierProvider(
          create: (context) => RiesgoInspeccionEliminadaNotifier(),
        ),
      ],
      child: Myaplicación(),
    ));
}
```

Listing 6.4: Utilizar el Notifier parte 2

6.4. Inicio Sesión

Antes de empezar a hablar sobre el inicio de sesión, trataremos algo que es común para todas las pantallas, (por lo que solo se mencionará en esta, en el resto se dará por hecho) todas las pantallas estarán anidadas dentro de un Scaffold, que a su vez estará anidado dentro de un ChangeNotifierProvider (para poder así utilizar Notifiers a la hora de utilizar el patrón Provider).

También, es importante mencionar que todas las pantallas son Widgets con estado, por lo que para crear dicho Widget hay que crear un estado. Esto se hará de la siguiente manera, y al igual que lo anterior también será común para todas las pantallas:

```
class Login extends StatefulWidget {
  @override
  _LoginState createState() => _LoginState();
}

class _LoginState extends State<Login> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

Listing 6.5: Crear Widget con estado

Una vez dicho esto, pasemos a hablar más a fondo de la propia pantalla de Login. Sobre las opciones generales del Scaffold, está desactivada la propiedad ‘’resizeToAvoidBottomInset’’ para evitar que cuando se muestre el teclado a la hora de rellenar los campos de texto se descoloque toda la interfaz, y su cuerpo es un ‘’AnnotatedRegion’’ para poder darle un nombre a esa región, es decir todo lo que se anide dentro. Quedará algo así:

```
Widget build(BuildContext context) {
    return Scaffold(
      key: _ScaffoldKey,
      resizeToAvoidBottomInset: false,
      body: AnnotatedRegion<SystemUiOverlayStyle>(
        value: SystemUiOverlayStyle.light,
        child: GestureDetector(
          onTap: () => FocusScope.of(context).unfocus(),
          child: Stack(
```

Listing 6.6: Widget principal pantalla Login

El gesture detector lo utilizaremos para que cuando el usuario pulse fuera del teclado este se esconda (evitando así la incomodidad de tener que darle al botón de atrás cada vez que se quiera esconder). Por último, creamos una pila en la que se anidarán todo el contenido de la pantalla. Para entender como está hecho todo lo que viene a partir de ahora es importante tener en cuenta que al ser una pila y no una columna los elementos no se ponen uno encima de otro, sino que se añaden en el mismo espacio.

Por lo que sino queremos que los Widgets que están dentro de la pila se pisen entre ellos, tendremos que asignar espacio entre cada uno de ellos. Antes de empezar con esto, vamos a diseñar un fondo, para ello, utilizaremos un contenedor que ocupe toda la pantalla al que le daremos diferente color por áreas: (hasta el 20 %, hasta el 60 % y hasta el final).

```
colors: [
    Color(0xF992A5F9),
    Color(0xC142A5F9),
    Colors.blue,
],
stops: [0.2, 0.6, 0.9],
```

Listing 6.7: Fondo pantalla Login

Hasta este punto ya tenemos diseñada por así decirlo la plantilla, solo quedará añadir el contenido. Para ello, añadiremos primero un texto con tamaño de letra grande que hará de titulo ‘’Inicio Sesión’’ y añadiremos un espacio (por lo mencionado anteriormente de las pilas). Puesto que tendremos que hacer esto varias veces solo se mencionará aquí: La manera de añadir un espacio entre elementos es: SizedBox(height: 40.0) O SizedBox(width: 40.0)

Teniendo el título nos faltaran, dos cuadros de texto (para el email y la contraseña), un botón de inicio de sesión y dos textos pulsables para ir a login o recuperar la contraseña.

A la hora de crear los cuadros de texto, es muy similar a crear el titulo con la diferencia de que para el titulo utilizamos un Widget Text (que no da la posibilidad de escribir), y en los cuadros de texto se utilizan Widget TextField o TextFormField:

```
TextFormField(
    key: Key("emailField"),
    controller: _emailController,
    keyboardType: TextInputType.emailAddress,
    style: TextStyle(color: Colors.white),
    decoration: InputDecoration(
        border: InputBorder.none,
        contentPadding: EdgeInsets.only(top: 14.0),
        prefixIcon: Icon(
            Icons.email,
            color: Colors.white,
        ),
        hintText: 'Introduce tu email',
   ),
),
```

Listing 6.8: Campos de texto

Tanto para el campo de texto de email como el de contraseña, se utiliza el fragmento de código de encima. Añadiendo al igual que con el titulo una separación entre ellos.

Para el botón utilizaremos un RaisedButton:

```
RaisedButton(
        key: Key('loginButton'),
          onPressed: () async {
            if (_formKey1.currentState.validate() &&
                _formKey2.currentState.validate()) {
              iniciarSesionPassEmail();
            }
          },
          elevation: 5.0,
          padding: EdgeInsets.all(15.0),
```

```
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(50.0),
          ),
```

Listing 6.9: Botón pantalla Login

Por último, nos quedan los dos textos para ir a Registro y para recuperar la contraseña. Puesto que la idea es la misma se utiliza el mismo procedimiento para ambos. Crear una fila, y dentro de ella añadir primero un texto (Ejemplo: No tienes cuenta aún? Has perdido tu cuenta?) al igual que se hizo con el titulo pero con diferente estilo, y un InkWell que es un texto con un evento de pulsado.

```
Row(mainAxisAlignment: MainAxisAlignment.center, children: <Widget>[
      Text(
        'No tienes una cuenta aún? ',
        style: TextStyle(
          color: Colors.white,
          fontSize: 18.0,
          fontWeight: FontWeight.w400,
        ),
      ),
      InkWell(
        key: Key('goRegister'),
          onTap: () {
            Navigator.of(context).push(MaterialPageRoute(
                builder: (BuildContext context) => Registro()));
          },
    ]);
```

Listing 6.10: Textos con enlaces

De esta manera, cuando se pulse en el texto de ”Registrate” se redireccionará a la pantalla de Registro.

Antes de empezar a hablar de la lógica, solo falta tratar la parte de gestión de errores, es decir, como notificar al usuario que un campo es obligatorio o que se han introducido datos incorrectos.

Para ello, se utiliza un ScaffoldKey: final GlobalKey<ScaffoldState>_ScaffoldKey=GlobalKey<ScaffoldState>(); a través de la cual podemos mostrar un mensaje por pantalla, de está manera en la función de iniciar sesión, en caso de que alguno de los errores se dispare se mostrará en dicho Scaffold.

Una vez vista la parte de diseño de la interfaz, nos quedará hablar de la lógica de la pantalla. Para ello, previamente vamos a explicar como almacena Flutter el contenido de un campo de texto para poder acceder a ello cuando suceda un evento, como se puede ver en el código de los campos de texto hay un argumento del TextFormField que se llama controller y que está igualado a _emailController, pues es a través de esta variable que se hace la declaración. Se hace de la siguiente manera: final TextEditingController _emailController=TextEditingController(); a través de la cual obtendremos el contenido del campo de texto.

Una vez sabemos como obtener el contenido de los campos de texto, podremos crear un objeto usuario con dichos campos y pasarlo por parámetro a la función de inicioSesion de

la base de datos junto a su Notifier. (para entender mejor como funcionan los Notifier ir a Patrón Provider).

En las operaciones a base de datos, puesto que se está usando Firebase se utiliza la herramienta de authenticator que implementa Firebase para mayor seguridad, por lo que a la hora de realizar el login se tendrá que iniciar sesión en Firebase authenticator y settear el current user de Firebaseuser junto con el authNotifier (que se utiliza para mantener la sesión):

```
UserCredential authResult=await FirebaseAuth.instance
      .signInWithEmailAndPassexcel(email: user.email, passexcel: user.
   passexcel)
      .catchError((error) => print(error.code));
if (authResult != null) {
    User FirebaseUser=authResult.user;
    if (FirebaseUser != null) {
        authNotifier.setUser(FirebaseUser);
    }
}
```

Listing 6.11: Autenticación Firebase

En lo que respecta al diseño de la interfaz de la pantalla de Registro no es necesaria una mayor explicación ya que su código va a ser prácticamente igual.

En cuanto a la lógica, va a ser similar a la parte de login en lo que respecta a usar authenticator (teniendo en este caso que crear el usuario en vez de iniciar sesión), y en lo que se refiere al almacenamiento de datos se hará en el Storage de Firebase, dicho almacenamiento funciona con colecciones (que serían para entenderlo los dominios), y en cada colección se guardan instancias.

Por lo que tendremos 1º que crear el usuario auth y 2º guardar los datos en el Storage en la colección de usuario:

```
UserCredential authResult=await FirebaseAuth.instance
      .createUserWithEmailAndPassexcel(
        email: u.getEmail(),
        passexcel: u.getPassexcel(),
      )
      .catchError((error) => print(error.code));
  if (authResult != null) {
    User FirebaseUser=authResult.user;
    if (FirebaseUser != null) {
      Map<String, dynamic> demoData={
        "email": u.getEmail(),
        "numero": u.getPhone(),
        "dni": u.getDni(),
        "nombre": u.getNombre(),
        "contraseña": u.getPassexcel(),
        "tipo": "inspector",
        "url": null
      };
      CollectionReference collectionReference=FirebaseFirestore.
   instance.collection('usuario');
      collectionReference.add(demoData);
      authNotifier.setUser(FirebaseUser);
```

```
    }
  }
```

Listing 6.12: Registro Firebase

6.5. Página principal

Al iniciar sesión con éxito, o al registrar un nuevo usuario, se redireccionará a la pantalla principal. Importante mencionar que puesto que esta página no muestra nada cargado de la base de datos, se pensó desde un inicio para cargar los datos que se necesitan en pantallas posteriores, de esta manera se evitan retrasos en otras pantallas.

El diseño de esta pantalla se basa en una pila anidada en el Scaffold, y luego un contenedor para establecer el fondo (al igual que en la pantalla de login y registro), con la diferencia que en este caso si se muestra la barra de aplicación del Scaffold, para poder tener acceso al menú lateral del que se hablará posteriormente.

En este caso, anidamos una imagen y un Widget añadido llamado WillPopScope, utilizado para que si el usuario intenta pulsar el botón de atrás del teléfono se le pida confirmación junto a un mensaje, indicándole que en caso de confirmar se cerrará la aplicación.

Para la imagen:

```
Container(
    height: 380.0,
    decoration: BoxDecoration(
    image: DecorationImage(
        image: NetworkImage(
            "https://i2.wp.com/www.asesorus.es/wp-content/uploads
   /2019/11/prevencion-riesgos-laborales.jpg?fit=710%2C320&ssl=1"),
            fit: BoxFit.cover,
        ),
    ),
),
```

Listing 6.13: Imagen por link de internet

Para el WillPopScope:

```
return WillPopScope(
    onWillPop: _onWillPopScope,

Future<bool> _onWillPopScope() {
    return showDialog(
        context: context,
        builder: (_) => AlertDialog(
            title: Text("¿Seguro que quieres salir de la aplicación?"),
            actions: [
                new ElevatedButton(
                    onPressed: () {
                    SystemNavigator.pop();
                    },
                    style: ElevatedButton.styleFrom(
```

```
                        primary: Colors.blue, onPrimary: Colors.black,
    elevation: 5),
                    child: Text('SI'),
                ),
                new ElevatedButton(
                    onPressed: () {
                    Navigator.of(context).pop();
                    },
                    style: ElevatedButton.styleFrom(
                        primary: Colors.blue, onPrimary: Colors.black,
    elevation: 5),
                    child: Text('NO'),
                ),
            ],
        ),
    );
}
```

Listing 6.14: Mensaje confirmación al volver atrás

Debajo de la imagen cargada, hay un texto similar al titulo de inicio de sesión para dar la bienvenida a la aplicación y por último, cartas pulsables que dan acceso a la funcionalidad principal de la aplicación.

Puesto que la funcionalidad no es la misma para los dos tipos de usuarios, las cartas que se mostrarán serán diferentes en función del tipo de usuario, no se va a entrar en más detalles de la lógica debido a que se explicará posteriormente, pero la manera de comprobar el tipo de usuario es la siguiente:

```
userNotifier.currentUsuario.tipo == "inspector"
    ? cartasInspector()
    : cartasAdministrador(),
```

Listing 6.15: If en diseño de interfaz

En caso de que el usuario sea de tipo inspector, se mostrará el Widget cartasInspector, y en caso contrario se mostrara el Widget cartasAdministrador.

Para la realización de estas cartas (lista inspecciones y añadir inspección en caso del inspector y lista inspectores en caso del administrador) habrá que anidar un InkWell dentro de un contenedor. El contenedor servirá para crear la plantilla de la carta, (en este caso un cuadrado de color gris) y el InkWell para añadir un evento de pulsado. Puesto que el Widget está anidado en una pila y debido a el diseño nos interesa especificar posiciones exactas tendremos que utilizar el Widget Positioned, que nos permite establecer una posición para su Widget hijo (en este caso el contenedor). Por lo que un esquema general sería:

```
return Positioned(
  child: Container(
    child: Card(
      child: InkWell(
        child: Center(
          child: Column(
            children:[
              Icon()
```

```
                Text()
```

Listing 6.16: Estructura cartas administrador página principal

El código de encima sería para el caso del administrador, que tiene una sola carta-botón (lista de inspectores), para el caso del usuario inspector, al ser dos cartas, debajo de container habrá que añadir lo siguiente:

```
child: Row(
  children:[
    Card ( …
    Card ( …
```

Listing 6.17: Estructura cartas inspector página principal

Con esto estaría finalizado el diseño de la interfaz, ahora pasemos a ver la lógica de esta pantalla.

Como se mencionó al principio, esta pantalla se usa para cargar información de la base de datos para evitar futuros retrasos, por lo que al entrar a la pantalla se invocan los métodos de getInspecciones getProvincias y getUser a través de los cuáles se cargaran en sus respectivos Notifiers la información necesaria, para ello, tendremos que realizar las respectivas consultas a Firebase y guardar el resultado en el Notifier:

```
getInspecciones(InspeccionNotifier inspeccionNotifier) async {
    QuerySnapshot snapshot =
      await FirebaseFirestore.instance.collection('inspeccion').where('email', isEqualTo: FirebaseAuth.instance.currentUser.email).get();
    List<Inspeccion> inspeccionesList=[];
    snapshot.docs.forEach((document) {
        Inspeccion i=Inspeccion.fromMap(document.data());
        i.setIdDocumento(document.id);
        inspeccionesList.add(i);
    });
    inspeccionNotifier.inspeccionList=inspeccionesList;
}
```

Listing 6.18: Cargar inspecciones del usuario logeado

De esta manera tendremos cargadas todas las inspecciones del usuario que está con la sesión iniciada, pudiendo mostrar dicha información al instante cuando acceda a la pantalla de lista de inspecciones. No es necesario mostrar los otros dos métodos ya que su código es muy similar.

Además de lo visto hasta el momento, en esta pantalla también se realiza la conversión de los Json que almacenan los riesgos y los subriesgos a objetos, además de el guardado en sus respectivos Notifier.

```
inicializarRiesgos(RiesgoNotifier riesgoNotifier) async {
    final respuesta=await rootBundle.loadString('data/riesgos.Json');
    Map dataMap=Json.decode(respuesta);
    for (var i=0; i < dataMap['riesgos'].length; i++) {
        Riesgo r=new Riesgo(dataMap['riesgos'][i]['id'], dataMap['riesgos'][i]['nombre'], dataMap['riesgos'][i]['icono']);
        _riesgos.add(r);
    }
```

```
8       riesgoNotifier.riesgoList=_riesgos;
9       riesgoNotifier.bandera=1;
10  }
```

Listing 6.19: Conversión Json a objetos

La bandera se utiliza para que, en caso de que se vuelva a acceder a la pantalla principal no vuelva a realizar la conversión.

El resto de lógica que tiene esta pantalla es simplemente navegabilidad, tanto los botones del menú como las cartas de funcionalidad.

6.6. Gestión de usuario

Antes de empezar a hablar por separado de cada una de las pantallas dedicadas a la gestión de usuario, vamos a ver el menú lateral que habrá en cada una de ellas y en la página principal.

Para realizar un menú hay que anidar un ListView dentro de un drawer, que tendrá como Widgets hijos: un DrawerHeader en el que habrá una pequeña imagen, y un texto para estilizar el menú, además de un CustomList por cada elemento que se quiera añadir al menú. Dentro de cada CustomList habrá un texto, y un icono además del evento de pulsado que provoca la redirección a cada página del menú.

Una de las opciones del menú es la de cerrar sesión, para que así cuando el usuario vuelva a arrancar la aplicación acceda a la página de login y no a la principal. Dicha función tendrá que cerrar la sesión de authenticator y vaciar el valor del auth Notifier:

```
1  AuthNotifier authNotifier =
2          Provider.of<AuthNotifier>(context, listen: false);
3      await FirebaseAuth.instance
4          .signOut()
5          .catchError((error) => print(error.code));
6      authNotifier.setUser(null);
```

Listing 6.20: Funcionalidad cerrar sesión

6.6.1. Pantalla cambio de contraseña

Al igual que el resto en esta pantalla, hay una pila anidada en el Scaffold, con la diferencia de que en este caso hay que darle valor al parámetro drawer del Scaffold (esto hay que hacerlo en todas las pantallas en las que tenga que ser visible el menú lateral).

Dentro de la pila, crearemos un Widget form y anidaremos tres Widgets de campos de texto y un botón similar a los de login o registro. La única diferencia es que en este caso he incluido la notificación de errores al usuario en el propio campo de texto, ya que hay más de un posible error en cada campo. Para ello, tendremos que hacer uso de la función validator del TextFormField que se comprobara cada vez que se pulse el botón gracias a la key del form. Cuando se pulsa el botón de confirmar, se comprueba con la key del form los TextFormField que tienen función validator:

```
if (!formKey.currentState.validate()) return;

formKey.currentState.save();


        //De esta manera se invocan las tres funciones validator de los
    TextFormField:


validator: validatePassAntigua,
validator: validatePass,
validator: validatePassRepetida,
```

Listing 6.21: Formkey para validar los campos de texto

Dichas funciones llaman a otras que son las que hacen las comprobaciones necesarias, como por ejemplo en validatePassAntigua, en donde se comprueba si el campo de texto tiene contenido (ya que es un campo necesario) y en caso de que así sea, que la contraseña introducida sea igual a la actual del usuario (se requiere introducir la contraseña actual del usuario para poder cambiarla)

```
String validatePassAntigua(value) {
    UsuarioNotifier userNotifier =
        Provider.of<UsuarioNotifier>(context, listen: false);
    if (value.isEmpty) {
      return "Necesario";
    } else if (_contraActualController.text !=
        userNotifier.currentUsuario.passexcel) {
      return "La contraseña introducida no es correcta";
    }
    return null;
}
```

Listing 6.22: Ejemplo validación campo de texto

Las otras dos funciones de validación son muy similares a esta por lo que no se mostrará su código.

Una vez visto todo esto, solo nos quedará la lógica del botón una vez pase como correcta la validación de los tres campos, es decir el cambio de contraseña. Para realizar dicho cambio, tendremos que 1º cambiar la contraseña del authenticator y 2º cambiar la contraseña del Storage:

```
modificarPass(Usuario u, String id, AuthNotifier authNotifier,
   UsuarioNotifier usuario) async {
  CollectionReference collectionReference =
      FirebaseFirestore.instance.collection('usuario');
  collectionReference
      .doc(id)
      .update({'contraseña': u.passexcel});
    User FirebaseUser=await FirebaseAuth.instance.currentUser;
    FirebaseUser
        .updatePassexcel(u.passexcel).then((_){
      print("Contraseña cambiada con exito");
    }).catchError((error){
      print("Error al cambiar la contraseña: " + error.toString());
```

```
    });
    authNotifier.setUser(FirebaseUser);
    usuario.currentUser=u;
}
```

Listing 6.23: Cambio de contraseña Firebase

Lo único necesario de mención aquí es el hecho de que a la hora de modificar el documento del Storage se necesita su id. Como funciona Firebase Storage es con colecciones y dentro de cada colección documentos, y en caso de querer modificar un documento se tiene que especificar su id. La manera de conseguir esta id es a la hora de realizar el get del usuario, al cargar un documento de la base de datos carga también su id, aprovecho ese momento para guardar dicha id en el Notifier y tenerla para modificaciones futuras.

6.6.2. Pantalla perfil

De una manera similar a como se hace en las anteriores vistas, hay una columna anidada dentro de un Scaffold el cual tiene su atributo drawer establecido al menú del que hablamos previamente para poder así acceder al menú lateral. En sí, la vista es muy simple, esta compuesta por una foto cargada de internet al igual que se hacia en la página principal, que ocupa la mitad superior de la pantalla, una foto de avatar del usuario acompañada de dos pequeños textos con el nombre del usuario y su tipo, y un botón para acceder a modificar perfil.

Lo único interesante de mencionar en esta pantalla, puesto que la mayoría de elementos que la forman ya han sido explicados en pantallas previas, es la manera en la que se cargan los datos del usuario y la manera en la que se permite al usuario agregar una foto.

Lo primero, es bastante simple habiendo cargado el usuario en la pagina principal, ya que gracias a ello tenemos un objeto usuario cargado en el Notifier del usuario. De esa manera si queremos obtener sus datos solo tendremos que hacerlo a través del currentUsuario:

```
UsuarioNotifier userNotifier=Provider.of<UsuarioNotifier>(context);
Text(
  userNotifier.currentUsuario.nombreCompleto,
  style: TextStyle(
        fontSize: 28.0,
        fontWeight: FontWeight.bold,
    ),
),
```

Listing 6.24: Cargar datos del Notifier usuario

Para lo segundo, he decidido crear una clase aparte para no tener que repetir el código, ya que tanto esta vista como la de modificar perfil deberían de poder modificar la foto del usuario.

Previamente a llamar al Widget que hemos separado en otra clase, tendremos que comprobar si el usuario ya tiene una foto cargada. En caso de que ya tenga una foto cargada, se leerá esa foto de la base de datos y se mostrará, en caso contrario se mostrará una foto de un avatar. Para ello:

```
UsuarioNotifier userNotifier=Provider.of<UsuarioNotifier>(context);
userNotifier.currentUsuario.url == null
  ? Foto()
  : FotoCargada(),
```

Listing 6.25: Comprobar si usuario tiene foto cargada

La imagen se forma por un CircleAvatar, que cargará la imagen seleccionada en caso de que sea diferente de null (es decir, si se selecciona una imagen), o la imagen de la base de datos en caso de que tuviera una y sino la imagen de avatar:

```
CircleAvatar(
    backgroundImage: _imageFile == null
        ? NetworkImage(userNotifier.currentUsuario.url)
        : FileImage(File(_imageFile.path)),
),
CircleAvatar(
    backgroundImage: _imageFile == null
        ? AssetImage('assets/images/usuario.png')
        : FileImage(File(_imageFile.path)),
),
```

Listing 6.26: Widget para la imagen de perfil

Y un pequeño icono de una cámara en la esquina inferior derecha. Para ello, tendremos que anidar el icono dentro de un Positioned que nos permita situarlo en la esquina inferior izquierda y de un FlatButton para que tenga un evento de pulsado. Cuando el usuario pulse la cámara, se le mostrara un modal (es decir, una ventana emergente) para mostrar algo en una ventana emergente hay que hacer lo siguiente:

```
showModalBottomSheet(
  context: context,
  builder: ((builder) => ImageModal()),
);
```

Listing 6.27: Widget para lanzar un modal

Dicho modal, contendrá una columna anidada en un contenedor que tendrá por una parte un texto indicando al usuario que seleccione uno de los métodos, y por otra parte una fila con dos iconos anidados dentro de dos botones, uno de una cámara y uno de una galería, de esta manera es muy fácil para el usuario entender que botón sirve para cada cosa. Dependiendo de que icono pulse:

takePhoto(ImageSource.camera); o **takePhoto(ImageSource.gallery);**

La funcionalidad de ese método es cargar la imagen y guardarla en la base de datos, Para ello, vamos a usar otro almacenamiento en la nube que tiene Firebase, que sirve para guardar archivos pesados, a parte del cloud Storage que estamos usando para guardar los datos hay un Firebase Storage que nos vale para guardar archivos, de esa manera podremos guardar en el Firebase Storage las imágenes y en el cloud Storage la url que lleva a esas imágenes. Para cargar la imagen de la cámara o de la galería:

```
final pickedFile=await _picker.getImage(
      source: source,
);
```

```
setState(() {
      _imageFile=pickedFile;
});
```

Listing 6.28: Cargar la foto

Para guardarlo en la base de datos, 1º se guarda en el Firebase Storage:

```
String fileName=Path.basename(_imageFile.path);
FirebaseStorage Storage=FirebaseStorage.instance;
Reference ref=Storage.ref().child(fileName);
var imageFile=File(_imageFile.path);
UploadTask uploadTask=ref.putFile(imageFile);
var imageUrl=await (await uploadTask).ref.getDownloadURL();
uploadTask.then((res) {});
imagePath=imageUrl.toString();
```

Listing 6.29: Guardar foto en Firebase Storage

2º se guarda en el cloud Storage, en el documento del current usuario:

```
CollectionReference collectionReference =
    FirebaseFirestore.instance.collection('usuario');
collectionReference
    .doc(userNotifier.currentUsuario.getId())
    .update({'url': imageUrl.toString()});
```

Listing 6.30: Guardar foto en cloud Storage

Por último, solo faltaría la lógica del botón de modificar perfil, que simplemente sirve para redireccionar a la pantalla de modificar perfil

6.6.3. Pantalla modificar perfil

En cuanto a la parte de diseño, no tiene nada nuevo que merezca la pena destacar, tiene un Scaffold con una barra de aplicación que en vez de añadir el menú como se ha hecho en otras vistas, añade una flecha de ‘’atrás’’ que sirve para volver a la vista de perfil.

El contenido consta de una pila anidada dentro del Scaffold y dentro de esta un GestureDetector, para que al igual que en la pantalla de Login y Registro el usuario pueda ocultar el teclado pulsando fuera de él, y por último, un texto para indicar que es la modificación de perfil y tres campos de texto similares a los de login utilizados para cambiar los datos del usuario. La única diferencia que tienen estos campos de texto, es que inicialmente tienen cargados sus datos. Para ello, haremos uso de los Notifiers, dando valores iniciales a los controllers de los campos de texto:

```
@override
  void initState() {
    súper.initState();
    UsuarioNotifier userNotifier =
        Provider.of<UsuarioNotifier>(context, listen: false);
    if (userNotifier.currentUsuario != null) {
      _nombreController.text=userNotifier.currentUsuario.nombreCompleto
    ;
```

```
8         _numeroController.text=userNotifier.currentUsuario.telefono;
9         _emailController.text=userNotifier.currentUsuario.email;
10      }
11  }
```

Listing 6.31: Dar valor a los controllers con Notifier

Visto esto, solo nos quedará hablar de la lógica de los dos botones, el primero de ellos, será el caso en el que el usuario cancele, no se realizará ningún cambio y redireccionará a la pantalla de perfil y el segundo, que es el de confirmar, llamará a la función de actualizar datos. Dicha función tendrá que realizar dos operaciones diferentes, la primera de ellas, para la modificación de los datos normales y la segunda, para la modificación del email.

En caso de los datos normales, la manera de proceder será similar a como se ha hecho en otras pantallas, se creará un usuario objeto con los nuevos valores leídos de los controllers, y se pasará como parametro a una función de modificación a la base de datos, en la que se cargará y modificará el documento con esa id, y se guardará en el Notifier el nuevo objeto usuario con los campos modificados:

```
1  modificarUsuario(Usuario u, String id, AuthNotifier authNotifier,
       UsuarioNotifier usuario) async {
2    CollectionReference collectionReference =
3        FirebaseFirestore.instance.collection('usuario');
4    collectionReference
5    .doc(id)
6    .update({'email': u.email, 'numero': u.telefono});
7      User FirebaseUser=await FirebaseAuth.instance.currentUser;
8      FirebaseUser.updateEmail(u.email);
9      authNotifier.setUser(FirebaseUser);
10     usuario.currentUser=u;
11 }
```

Listing 6.32: Actualizar datos usuario

Como se puede ver en el código de encima, también se está modificando el email. Esto es similar a como se hizo con la contraseña, hay que cambiarlo tanto en el Storage como en el authenticator. Además, hay que actualizar ambos Notifier, tanto el de auth como el de usuario.

6.7. Creación de una inspección

La manera de comenzar con el diseño es similar a la de la pantalla de Login, una pila anidada dentro de un GestureDetector y a su vez este anidado dentro de un Scaffold. En las pantallas que definen la funcionalidad principal de la aplicación, es decir todas menos las de inicio de sesión y gestión de usuario se utiliza un fondo de pantalla diferente a estas y común entre ella, por lo que primero vamos a ver el diseño de este fondo.

Puesto que se va a usar en todas las pantallas a partir de ahora, vamos a sacarlo a una clase a parte (como hicimos con la foto de modificar perfil y perfil) para evitar tener que repetir código por cada vez que lo queramos usar.

Para definir el fondo, lo primero va a ser pensar que color sería adecuado para la aplicación y podría parecerle correcto al usuario (no tiene porque ser solo un color, se pueden combinar), al igual que se hizo en la pantalla login defino un color hasta el 20 % de la pantalla y otro hasta el final haciendo uso de un LinearGradient anidado en una BoxDecoration:

```
decoration: BoxDecoration(
    gradient: LinearGradient(
    begin: FractionalOffset(0.0, 0.2),
    end: FractionalOffset(0.0, 0.6),
    colors: [
        Color.fromRGBO(13, 0, 255, 1.0),
        Color.fromRGBO(175, 0, 255, 1.0)
    ],
)),
```

Listing 6.33: Definir colores fondo aplicación

Una vez está decidido el color de fondo, para evitar que quede muy lineal añadiré dos cuadrados de diferentes colores, uno en la parte superior y otro en la parte inferior. Para hacer el cuadrado habrá que hacer un contenedor de tamaño a elegir y dándole uno o más colores:

```
final cajaAzul=Transform.rotate(
    angle: -pi / 13.2,
    child: Container(
        height: 460.0,
        width: 460.0,
        decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(175.0),
            gradient:
              LinearGradient(colors: [Colors.cyanAccent, Colors.
    blueAccent]),
        ),
    ),
);
```

Listing 6.34: Creación cuadrados fondo aplicación

Utilizo la rotación para que quede girado y una esquina quede escondida en la parte inferior de la pantalla, ya que si se añadiera el cuadrado entero en la pantalla la saturaría mucho y podría llegar a ser perjudicial para la experiencia de usuario.

Una vez visto el diseño del fondo que se utilizará, vamos a seguir con el diseño de la pantalla de añadir inspección, puesto que ya hemos hablado de la estructura general y la definición del fondo, nos queda el último elemento que se a primera vista al entrar en la pantalla, un botón. Dicho botón, es el que muestra el dialogo que contiene todos los campos de texto para crear la inspección. El evento de pulsado llama a un Widget showDialog (solo se muestra cuando se pulsa el botón) que muestra una caja que ocupa el 80 % del ancho de la pantalla.

La caja tendrá un contenedor y una columna anidados en un SingleChildScrollView (Widget utilizado para que sino entra todo el contenido en la pantalla se pueda hacer scroll), el contenedor contendrá un mensaje de texto indicando que estamos en la pantalla

de agregar inspección y la columna contendrá un form con todos los campos de texto. Muchos de ellos los obviaremos, ya que tanto su diseño como la forma en la que notifican al usuario errores (por ejemplo: este campo es obligatorio), es muy similar a los de la pantalla de cambio de contraseña. Los únicos Widgets que aún no hemos visto hasta el momento son: el DropdownmenuItem y un mapa utilizado para la ubicación.

El DropdownmenuItem muestra todas las provincias (ordenadas alfabéticamente), añadidas previamente en la pantalla de página principal a su respectivo Notifier. Para ello, primero tendremos que añadirlas al menú:

```
List<String> listaProvincias=[];
    inspeccionNotifier.provinciaList.forEach((provincia) {
      listaProvincias.add(provincia.provincia);
    });
    listaProvincias.sort();
    for (int i=0; i < listaProvincias.length; i++) {
      _provincias.add(DropdownmenuItem(
        child: Text(listaProvincias[i]),
        value: listaProvincias[i],
      ));
    }
```

Listing 6.35: Cargar provincias y ordenarlas alfabéticamente

Una vez añadidas, solo quedará dar al controller el valor de la seleccionada (de manera similar a como se hace en los campos de texto):

```
return DropdownButtonFormField(
      decoration: InputDecoration(
          labelText: 'Provincia', labelStyle: TextStyle(fontSize: 20.0)
   ),
      value: _provinciaController,
      items: _provincias,
      onChanged: (value) {
        setState(() {
          _provinciaController=value;
        });
      },
);
```

Listing 6.36: DropdownmenuItem provincias

Antes de empezar con la lógica de la pantalla, vamos a ver la manera de realizar la ubicación, Para ello, se le presentan al usuario dos opciones, la primera de ellas funcionará como un campo de texto igual al resto, y sucederá cuando el usuario añada a mano la ubicación, y la segunda de ellas es cuando el usuario pulse el icono del mapa, lo que provocará que le redireccione a una nueva pantalla.

La nueva pantalla constará de dos partes, ambas anidadas dentro de un Scaffold y a su vez dentro de una columna: Un GoogleMap y un Contenedor con un botón y un texto.

El Widget GoogleMap tiene una gran variedad de parámetros a los que podemos dar valor, en mi caso, solo he utilizado la función onTap para tener un evento que reconozca en que lugar del mapa se está pulsando, el tipo de mapa, el punto inicial y las marcas. La idea es conseguir que cuando se apriete en un lugar se genere una marca y se actualice el

texto de la parte inferior con los datos de ese punto, por lo que ve en el evento de pulsado se generará una marca en el mapa (para que el usuario pueda confirmar con la vista que el lugar es el correcto), y se dará valor a la variable de localización. Puesto que la función para generar marcas esta diseñada para que solo admita una marca si el usuario vuelve a pulsar en el mapa se borrará toda la información mostrada de la antigua marca y se actualizará con la nueva.

```
child: GoogleMap(
    onTap: (taplicacióned) async {
        final coordenadas =
          new geoCo.Coordinates(taplicacióned.latitude, taplicacióned.
    lonGitude);
        var direccion=await geoCo.Geocoder.local
          .findAddressesFromCoordinates(coordenadas);
        var direccionInicial=direccion.first;
        getMarcas(taplicacióned.latitude, taplicacióned.lonGitude);
        setState(() {
            localizacion=direccionInicial.addressLine;
        });
    },
    mapType: MapType.normal,
    initialCameraPosition: CameraPosition(
        target: LatLng(posicion.latitude.toDouble(),
        posicion.lonGitude.toDouble()),
        zoom: 15.0),
    markers: Set<Marker>.of(marcas.values),
),
```

Listing 6.37: Widget GoogleMap

Para las marcas:

```
void getMarcas(double lat, double long) {
    MarkerId markerId=MarkerId(lat.toString() + long.toString());
    Marker _marker=Marker(
        markerId: markerId,
        position: LatLng(lat, long),
        icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.
    hueCyan),
        infoWindow: InfoWindow(snippet: 'Dirección'));
    setState(() {
      marcas.clear();
      marcas[markerId]=_marker;
    });
  }
```

Listing 6.38: Markers del mapa

Por último, mencionar que, como se puede ver en el código del GoogleMap Widget, está cogiendo el contenido de la variable posición como posición inicial. La manera de dar valor a esa variable es con la ubicación del usuario (esto provoca que probándolo con el emulador de Android de error):

```
@override
void initState() {
    súper.initState();
```

```
4       getCurrentLocation();
5   }
6
7
8   void getCurrentLocation() async {
9       Position currentPosicion =
10          await GeolocatorPlatform.instance.getCurrentPosition();
11      setState(() {
12        posicion=currentPosicion;
13      });
14  }
```

Listing 6.39: Actualizar datos usuario

Una vez el usuario haya fijado la posición que quiere y pulse el botón de confirmar se volverá a la pantalla de añadir inspección y se auto completará el campo de texto de lugar con la ubicación seleccionada en el mapa, para ello, en la navegación al mapa hay que hacer lo siguiente:

Navigator.push(context, MaterialPageRoute(builder: (_) =>Mapa())).then((value) setState(() _direccionController.text=inspeccionNotifier.currentInspeccion.lugar;););

De esta manera cuando salga del mapa se actualizará el campo de texto lugar con el valor del campo lugar del Notifier que previamente ha sido seteado al de la localización en el mapa.

Una vez visto el diseño y la lógica del mapa solo nos queda hablar de la lógica de la pantalla de añadir inspección, es decir el punto en el que se ha pulsado el botón de añadir y se han pasado todas las validaciones de los campos de texto.

La manera de proceder en este caso es prácticamente igual a la de registrar usuario, se crea un objeto inspección con el valor de los controllers, estado inicial en proceso, fecha inicial la actual y fecha final null, y se pasa a la función de añadir a la base de datos junto con su Notifier. Esta función es prácticamente igual a la de registrar usuario, pero quitando la parte del authenticator.

6.8. Añadir Riesgo

6.8.1. Pantalla lista de riesgos evaluados

La primera pantalla que vamos a ver es la de lista de evaluaciones que es la que se muestra al usuario cuando entra a una inspección (ya sea recién creada o a través de la lista de inspecciones), en este caso la estructura es parecida a la que vimos de la pantalla de página principal, empiezo con un WillPopScope en el que anido el Scaffold. El objetivo de esto es el mismo que en la página principal, cuando el usuario intente ir atrás utilizando los botones del teléfono se le notificará mostrándole un mensaje de que si acepta le sacará de la inspección (no voy a entrar en mayor detalle en esto ya que está explicado en el apartado de página principal).

Dentro del Scaffold añado una pila y con esto ya tendríamos la estructura de la pantalla. A partir de aquí podemos dividirla en cuatro elementos por así decirlo: La lista de riesgos evaluados, el botón de finalizar inspección, el botón de añadir riesgo y el texto de titulo indicando que estamos en la pantalla que muestra los riesgos evaluados. En la esquina inferior izquierda está el botón de añadir riesgo, cuya funcionalidad simplemente redirecciona la aplicación a la pantalla de añadir riesgo.

En la esquina inferior derecha está el botón de finalizar inspección, el cual invoca un modal similar al de recuperar contraseña del login o el de la foto de perfil con un mensaje de confirmación por si acaso el usuario lo ha pulsado sin querer o ha pasado algo por alto y se da cuenta en ese momento. Dicho modal, esta compuesto por un Widget ShowDialog y dentro un texto junto a dos ElevatedButton, uno que sirve para cancelar la operación y el otro para confirmar la finalización de la inspección.

Por último, queda la lista de riesgo evaluados, para realizar esta lista hago uso de un Widget que crea un enlace entre el programa y Firebase, lo que hace que la vista se actualice al momento si sufre cualquier cambio, para ello, en el campo stream hay que realizar una consulta y en el builder especificar el contexto, una vez hecho esto se podrá añadir lo que se quiera dentro del StreamBuilder:

```
return StreamBuilder(
    stream: FirebaseFirestore.instance
      .collection('riesgo')
      .where('idInspeccion',
        isEqualTo: inspeccionNotifier.currentInspeccion.id)
      .where('eliminado', isEqualTo: false)
      .where('evaluado', isEqualTo: true)
      .snapshots(),
    builder: (
      context,
      snapshot,
    ){
```

Listing 6.40: StreamBuilder lista riesgo evaluados parte 1

En mi caso, al tratarse de una lista, compruebo en primer momento que el snapshot (es lo que devuelve una consulta a Firebase) no sea nulo, y posteriormente creo la lista:

```
if (snapshot.data == null)
    return Center(
        child: CircularProgressIndicator(
            backgroundColor: Colors.red,
            valueColor: new Alwayseguridad y salud
   laboraloppedAnimation<Color>(Colors.teal),
        ),
    );
return ListView.builder
```

Listing 6.41: StreamBuilder lista riesgo evaluados parte 2

Para construir una lista hay que especificarle en el itemCount el tamaño de la lista, que en este caso será el tamaño de lo que devuelva la consulta, y por otra parte el itemBuilder, que tendrá el contexto y el índice. Una vez hecho esto dentro de la lista se podrán mostrar los datos como queramos, en mi caso he decidido utilizar cartas de colores en función del

nivel de criticidad del riesgo evaluado:

```
Color col;
if(snapshot.data.docs[index]['total'] <= 20){
  col=Colors.green;
}
if(snapshot.data.docs[index]['id'] > 20 && snapshot.data.docs[index]['
    id']<=120){
  col=Colors.yellow;
}
if(snapshot.data.docs[index]['id'] > 120 && snapshot.data.docs[index]['
    id']<=500){
  col=Colors.orange;
}
if(snapshot.data.docs[index]['id'] > 500){
  col=Colors.red;
}
```

Listing 6.42: Cambio color en función de la criticidad del riesgo

El siguiente paso es crear el riesgo, añadirlo a una fila y pasarlo a la función que genera las cartas. De esta manera, cada carta se creará en una fila:

```
SubRiesgo r=new SubRiesgo(
  snapshot.data.docs[index]['id'],
  snapshot.data.docs[index]['nombre'],
  snapshot.data.docs[index]['icono'],
  0,
  snapshot.data.docs[index]['idUnica']);
  List<TableRow> rows=[];
  rows.add(TableRow(children: [
    _crearBotonRedondeado(col, r),
        ]));
        return Table(children: rows);
```

Listing 6.43: ListView riesgos evaluados

La carta consta de cuatro partes, un contenedor de color variable dependiendo se la criticidad, el nombre del riesgo evaluado, la imagen del riesgo evaluado y un icono de borrado. Para ello, creo una fila en la que anido la imagen del riesgo y una columna, en la columna anido el titulo en la parte súperior con un Positioned y de igual manera, pero en la parte inferior el cuadro con el color que indica la criticidad. Para el icono de borrado utilizo un Positioned para establecer su posición en la parte súperior derecha de la carta.

No se va a incluir código del diseño de la carta debido a que la mayoría de Widgets empleados ya se han visto en apartados anteriores, solo cambia la manera de combinarlos.

Una vez vista toda la parte de diseño de la pantalla vamos a pasar a ver la parte de lógica:

Siguiendo el mismo orden que en el diseño lo primero a tratar sería el botón de finalizar inspección. Una vez el usuario pulse este botón y confirme cerrar la inspección, se llamará a una función de modificación de base de datos similar a las que hemos visto previamente, en la que utilizando la id del documento se modifica el estado de la inspección, cambiándolo de en proceso a pendiente y a su vez guardando este cambio en su Notifier.

Luego pasamos a la parte de lista de riesgo que consta de dos funcionalidades, la de borrar una evaluación y la de seleccionar una evaluación. En la primera de ellas realizo

un borrado lógico de dicha evaluación, cambiando su bit de eliminado de la base de datos de 0 a 1 (el método para realizar esta modificación es igual al de finalizar inspección), haciendo a su vez que el StreamBuilder no seleccione dicha evaluación porque tiene ese bit de eliminado a 1 (de ahí la importancia de usar un StreamBuilder en esta situación, mantiene dinámicamente actualizada la lista).

La segunda funcionalidad es la de pulsar sobre una de las cartas, en cuyo caso se notificará al Notifier de evaluación que la evaluación actual es la de esa carta y se mostrará la pantalla de evaluación con sus datos actuales, dándole a su vez la posibilidad de modificarlos.

```
onTap: () {
    for(int i=0; i < evaluacionRiesgoNotifier.evaluacionList.length ; i
    ++){
        if(evaluacionRiesgoNotifier.evaluacionList[i].idRiesgo == sr.
    idUnica){
            evaluacionRiesgoNotifier.currentEvaluacion=
    evaluacionRiesgoNotifier.evaluacionList[i];
        }
    }
```

Listing 6.44: Guardar la evaluación en el current de su Notifier

Siendo sr el riesgo que se pasa por parámetro a la función que crea las cartas.

6.8.2. Pantalla seleccionar riesgo

Una vez el usuario pulsa el botón de añadir un nuevo riesgo lo primero que se le mostrará es esta pantalla, que se llama pantalla de riesgos, pero realmente es la pantalla de categorías. La pantalla esta compuesta por un titulo y una lista de cartas de riesgos.

Al igual que las otras pantallas esta compuesta por un Scaffold y anidada dentro una pila, dentro de la pila tendremos por una parte el fondo de la aplicación, y por otro lado el titulo y la lista de riesgos. El titulo es simplemente un texto que indica que nos encontramos en la pantalla de seleccionar una categoría.

Por otro lado, la lista de riesgo es algo similar a lo visto en la pantalla de lista de riesgos evaluados, pero en este caso sin un StreamBuilder, ya que los datos que carga la lista han sido previamente guardados en sus Notifiers. A diferencia de la pantalla de evaluaciones las cartas serán cuadradas y se hará una tabla de 2 cartas por fila. Dichas cartas están compuestas por el icono y el nombre del riesgo. Dicho esto, no creo que haya que entrar en mayor detalle ya que los elementos que componen la interfaz de esta pantalla son muy similares a los ya visto en otras pantallas.

Al igual que con el diseño de la interfaz, la lógica de la pantalla es muy simple, cuando se pulsa en una de las cartas, se guarda el objeto con el que se ha creado dicha carta en su Notifier, para que así en la próxima pantalla se puedan mostrar solo los riesgos de esa categoría.

6.8.3. Pantalla seleccionar subriesgo

Por último, nos queda la pantalla de añadir subriesgo, que pese a llamarse así su funcionalidad es añadir un riesgo, la pantalla anterior mostraba todas las categorías de riesgos y esta muestra los riesgos de la categoría seleccionada.

En lo que respecta al diseño, la pantalla es prácticamente idéntica a la anterior, con la diferencia de que esta cargara la lista guardada en el Notifier de subriesgos en vez de el de riesgos, y que además, solo cargara los que tengan el idPadre igual al id del current riesgo almacenado previamente en el Notifier la pantalla anterior. Para ello:

```
SubRiesgoNotifier subRiesgoNotifier =
  Provider.of<SubRiesgoNotifier>(context, listen: false);
RiesgoNotifier riesgoNotifier =
  Provider.of<RiesgoNotifier>(context, listen: false);
List<SubRiesgo> subRiesgos=[];
for (int i=0; i < subRiesgoNotifier.subRiesgoList.length; i++) {
    if (subRiesgoNotifier.subRiesgoList[i].idRiesgoPadre ==
        riesgoNotifier.currentRiesgo.id) {
        subRiesgos.add(subRiesgoNotifier.subRiesgoList[i]);
    }
}
```

Listing 6.45: Lista riesgos cuya categoría coincide con la seleccionada

Generaremos el ListView con esta lista en vez de la del Notifier que se utilizo en la pantalla anterior:

```
return ListView.builder(
      physics: NeverScrollableScrollPhysics(),
      shrinkWrap: true,
      itemCount: subRiesgos.length,
```

Listing 6.46: Actualizar datos usuario

Por último, nos queda hablar de la parte de lógica, la cual es similar a la anterior pantalla. Una vez el usuario pulsa una de las cartas se calculará una id única para posteriormente poder guardarlo en la base de datos, puesto que el resto de campos ya los tenemos cargados en el Notifier de subriesgo, el cual ha sido igualado al valor de la carta pulsada para ello:

```
int idNueva=0;
if (riesgoInspeccionEliminadaNotifier.riesgoList.length != 0) {
    for (int i=0;
     i < riesgoInspeccionEliminadaNotifier.riesgoList.length;
     i++) {
        if (idNueva <=
          riesgoInspeccionEliminadaNotifier.riesgoList[i].idUnica) {
            idNueva =
              riesgoInspeccionEliminadaNotifier.riesgoList[i].idUnica +
    1;
        }
    }
}
sr.setIdUnica(idNueva);
RiesgoInspeccionNotifier riesgoInspeccionNotifier =
  Provider.of<RiesgoInspeccionNotifier>(context, listen: false);
```

```
16 riesgoInspeccionNotifier.currentRiesgo=sr;
```

Listing 6.47: Calculo id única del riesgo y guardar en el Notifier

Una vez se haya hecho esto, se redirecciona a la pantalla de evaluación.

6.9. Evaluación riesgo

Antes de empezar a tratar el diseño de la interfaz, vamos a ver las diferentes opciones de llegada que tienes esta pantalla, ya que se puede acceder a ella al añadir un nuevo riesgo o a través de la lista de riesgos evaluados, seleccionando uno de ellos.

Esto es importante, ya que en caso de que se acceda desde la lista de riesgos evaluados habrá que cargar los datos de la evaluación (se da al usuario la posibilidad de cambiarlos, pero lo lógico es que pueda ver los valores que había puesto antes). Para ello, hay que invocar como se ha hecho en otras pantallas a la función de initState e igualar el valor de los elementos de la pantalla a los de la currentEvaluación. Para ello:

```
1  @override
2  void initState() {
3      súper.initState();
4      EvaluacionRiesgoNotifier evaluacionRiesgoNotifier =
5        Provider.of<EvaluacionRiesgoNotifier>(context, listen: false);
6      if(evaluacionRiesgoNotifier.currentEvaluacion!=null){
7          _tituloController.text=evaluacionRiesgoNotifier.
       currentEvaluacion.titulo;
8          if(evaluacionRiesgoNotifier.currentEvaluacion.tipo ==
       TipoFactor.Existente){
9              _tipoFactorController="Existente";
10         }else{
11             _tipoFactorController="Potencial";
12         }
13         _accionCorrectoraController.text=evaluacionRiesgoNotifier.
       currentEvaluacion.accionCorrectora;
14         _valorDeficiencia=evaluacionRiesgoNotifier.currentEvaluacion.
       nivelDeficiencia.toDouble();
15         _valorExposicion=evaluacionRiesgoNotifier.currentEvaluacion.
       nivelExposicion.toDouble();
16         _valorConsecuencias=evaluacionRiesgoNotifier.currentEvaluacion.
       nivelConsecuencias.toDouble();
17     }
18 }
```

Listing 6.48: Cargar datos evaluación

En caso de que se haya accedido a través de la pantalla de añadir un nuevo riesgo currentEvaluación==null, por lo que no llegará a hacer nada de esto.

Una vez visto esto, pasemos a hablar del diseño de la interfaz. Al igual que se hizo con la pantalla de página principal y la de lista de evaluaciones en esta también he añadido un WillPopScope que utilizo para notificar al usuario en caso de que intente volver hacia atrás, que esto hará que se finalice la evaluación y le enviará nuevamente a la página de lista de riesgos evaluados.

En cuanto a la estructura general, es muy similar a la pantalla de añadir inspección, una pila anidada dentro de un GestureDetector (para lo mencionado anteriormente de poder ocultar el teclado pulsando fuera de él) y este a su vez anidado dentro de un Scaffold. Al igual que se hizo con la pantalla de add inspección, el contenido de la pantalla está en una caja centrado que ocupa el 90 % de la pantalla, con la diferencia que esta caja no se muestra al pulsar un botón, está visible en todo momento.

Dicha caja está formada por una columna y anidados dentro dos elementos principales: un contenedor para dar diseño al fondo ,y una columna que tiene todo el contenido de la caja. Dicha columna tiene anidados dentro, un contenedor que forma la parte súperior de la caja (el titulo y el nombre del riesgo que se está evaluando) y un form que a su vez tiene una columna anidada dentro. Dentro de esa columna es donde se encuentran todos los Widgets que formaran la evaluación.

De los campos de texto y el Dropdownmenu no voy a hablar puesto que se han explicado en otras pantallas, tanto la manera en la que se notifican al usuario los errores, como su diseño es igual a la utilizada para la pantalla de añadir inspección. Por ello, vamos a pasar a ver directamente los nuevos elementos añadidos.

Empecemos por los Sliders, barras utilizadas para indicar los niveles de consecuencia, exposición y deficiencia del riesgo. Para ello, vamos a anidar un contenedor y un SliderTheme (para dar diseño al slider) que a su vez tiene un Slider dentro de una columna, el primero se utiliza para personalizar el color del Slider, indicando los valores bajos de color verde, los intermedios color amarillo y los altos color rojo (logrando así que el slider sea más intuitivo):

```
Container(
    height: 7,
    width: _size.width * 0.8,
    decoration: BoxDecoration(
        border: Border.all(color: Colors.black, width: 0.1),
        gradient: LinearGradient(colors: [
            Color.fromRGBO(0, 255, 4, 1),
            Color.fromRGBO(143, 255, 0, 1),
            Color.fromRGBO(170, 255, 0, 1),
            Color.fromRGBO(255, 128, 0, 1),
            Color.fromRGBO(255, 89, 0, 1),
            Color.fromRGBO(255, 0, 0, 1),
        ]),
    borderRadius: BorderRadius.circular(10),
),
```

Listing 6.49: Establecer colores Sliders

y el segundo que es el Widget encargado de crear el Slider, este Widget tiene muchas opciones de diseño para cambiar el color de la bola, aumentar la opacidad o disminuirla en función de si está pulsada, en mi caso he decidido utilizar las siguiente:

```
activeTrackColor:
    Theme.of(context).primaryColor.withOpacity(0.1),
    inactiveTrackColor:
      Theme.of(context).primaryColor.withOpacity(0.1),
    trackShape: RoundedRectSliderTrackShape(),
    trackHeight: 4.0,
```

```
    thumbShape: RoundSliderThumbShape(enabledThumbRadius: 11.0),
    thumbColor: Colors.blueAccent,
    overlayColor: Theme.of(context).primaryColor.withAlpha(32),
    overlayShape: RoundSliderOverlayShape(overlayRadius: 20.0),
    tickMarkShape: RoundSliderTickMarkShape(),
    activeTickMarkColor: Colors.black,
    inactiveTickMarkColor: Colors.black,
    valueIndicatorShape: PaddleSliderValueIndicatorShape(),
    valueIndicatorColor:
      Theme.of(context).primaryColor.withOpacity(0.8),
    valueIndicatorTextStyle: TextStyle(
        color: Colors.white,
    ),
    showValueIndicator: ShowValueIndicator.always,
```

Listing 6.50: Modificar diseño gráfico del Slider

Puesto que de la gran mayoría de ellas se puede intuir su utilidad solo con el nombre y sus utilidades son simplemente estéticas, vamos a pasar a hablar de lo realmente importante que es el Slider en sí. En dicho Slider, tendremos que especificar: un value (que es como el controller de los campos de texto), y que varia en función del movimiento del Slider, un máximo y un mínimo para especificar los límites del Slider y un número de divisiones:

```
child: Slider(
    value: _valorDeficiencia,
    min: 0,
    max: 10,
    divisions: 3,
```

Listing 6.51: Widget Slider parte 1

y una función onChanged (value) para notificar el evento de cambio de valor, dentro de dicha función se actualiza el valor del controller a value y se crea un switch para marcar las opciones de valores que este puede tomar.

```
    onChanged: (value) {
        _valorDeficiencia=value;
        switch (value.ceil()) {
            case 0:
                _deficiencia=0;
                break;
            case 1:
                _deficiencia=2;
                break;
            case 2:
                _deficiencia=6;
                break;
            case 3:
                _deficiencia=10;
                break;
        }
```

Listing 6.52: Widget Slider parte 2

Con esto logramos, que a diferencia de un Slider normal en el que solo se establece el máximo y el mínimo y se puede tomar cualquier valor Double dentro de ese rango, en este

solo podrá tomar los valores especificados dentro del switch, de esta manera nos podremos ajustar a la perfección a los estándares establecidos para evaluar un riesgo.

Al igual que se ha hecho con este, hay otros dos Sliders para especificar el nivel de exposición y consecuencia del riesgo. Además, hay dos campos de texto, uno para dar un titulo a la evaluación y otro para especificar una acción correctora y un Dropdownmenu de selección con dos opciones (una en el caso de que sea un riesgo existente y otra en caso de que sea potencial) y dos botones, una para añadir fotos y otro para finalizar la evaluación.

En lo que respecta al botón para añadir imágenes, una vez se pulsa se le muestran al usuario las opciones de cargar desde la galería o desde la cámara y una vez este seleccione una foto esta se guardará en la base de datos junto con la id única del riesgo. De esta manera, se puede cargar de manera dinámica para mantener la lista de fotos actualizada.

Para ello, hago algo similar a lo que se hizo en la pantalla de lista de riesgos evaluados, crear un ListView dentro de un StreamBuilder, y que dicho Stream este cogiendo las fotos cuya idUnica de riesgo sea igual a la del riesgo que se está evaluando y que a su vez tengan el estado eliminado a false:

```
return StreamBuilder(
    stream: FirebaseFirestore.instance
      .collection('fotoRiesgo')
      .where('idRiesgo',
        isEqualTo: riesgoInspeccionNotifier.currentRiesgo.idUnica)
      .where('eliminada', isEqualTo: false)
      .snapshots(),
```

Listing 6.53: StreamBuilder lista de fotos del riesgo evaluado

logrando así, que cuando se añada una foto a la base de datos el StreamBuilder la cargue en la lista sin necesidad de recargar la pantalla, y en caso de que se borre una imagen, se realizará un borrado lógico en la base de datos estableciendo el valor del campo eliminado a true y provocando que la lista que está dentro del Stream elimine esa imagen:

```
eliminarFoto(FotoRiesgo f ) async {
    QuerySnapshot snapshot=await FirebaseFirestore.instance
      .collection('fotoRiesgo')
      .where('url', isEqualTo: f.path)
      .where('idRiesgo', isEqualTo: f.idRiesgoUnica)
      .get();
    snapshot.docs.forEach((document) {
        f.setIdDocumento(document.id);
    });
    CollectionReference collectionReference =
      FirebaseFirestore.instance.collection('fotoRiesgo');
    collectionReference
      .doc(f.getIdDocumento())
      .update({'eliminada': true});
}
```

Listing 6.54: Eliminar foto lista de fotos del riesgo evaluado

Como se puede ver en la imagen superior lo que almacenamos en la base de datos es la url de la foto, ya que al igual que con la foto de perfil el almacenamiento está hecho en el Storage de Firebase.

Lo que respecta a la manera de mostrar las fotos: se realiza en una lista con cartas de manera similar a como se hizo con los riesgos, en una tabla con filas de dos elementos cada una. Dicha carta solo muestra la imagen: (Muestro solo el código de la imagen puesto que el de la lista es muy similar al visto en las anteriores pantallas)

```
child: FadeInImage(
    height: 150.0,
    width: 150.0,
    fit: BoxFit.cover,
    placeholder: AssetImage('assets/img/original.gif'),
    image: NetworkImage(foto.path)),
```

Listing 6.55: Carta de la imagen del riesgo evaluado

Una vez visto esto ya habríamos acabado con la parte de diseño de la interfaz, solo nos quedará ver la lógica de esta.

Puesto que el tema de las validaciones ya lo vimos en la pantalla de añadir inspección, vamos a dar por hecho que ya se han validado todos lo campos y se va a pasar al apartado de agregar la evaluación. Para ello, tendremos primero, al igual que se hizo con el riesgo crear una id única para dicha evaluación, y una vez la tengamos, crear el objeto evaluación con dicha id y todos los campos que ha rellenado el usuario en esta misma pantalla.

Una vez tengamos creado el objeto de evaluación, se presentarán dos opciones por la misma razón que se dijo al principio de esta pantalla (se ha podido acceder a través de un riesgo ya evaluado o de uno nuevo), en caso de que se haya accedido a través de un riesgo ya evaluado estaremos modificando la evaluación, por lo que la operación contra la base de datos será de modificación y no tendremos que añadir el riesgo puesto que este ya está añadido. En el segundo caso tendremos que, primero añadir el riesgo, después añadir la evaluación y por último, modificar el estado del riesgo a evaluado.

Puesto que en ambos casos son operaciones contra la base de datos que ya se han tratado en apartados anteriores no vamos a entrar en más detalle, lo único necesario de mención aquí es la manera de calcular el valor final de la evaluación del riesgo, el cual se añadirá al documento del riesgo en la base de datos. Para ello, tendremos que multiplicar el nivel de deficiencia por el nivel de exposición, lo que nos dará el nivel de probabilidad, y a este último multiplicarle el nivel de consecuencia:

```
int _probabilidad=_deficiencia  * _exposicion;
int _resultadoFinal=_probabilidad * _consecuencias_;
```

Listing 6.56: Calculo valor final de la criticidad del riesgo

6.10. Lista de inspecciones

Antes de pasar a hablar del otro tipo de usuario vamos a ver la última funcionalidad del tipo de usuario inspector, que incluye una sola pantalla cuya utilidad es mostrar al inspector su lista de inspecciones.

Al igual que con la mayoría de pantallas su estructura cuenta con una pila anidada dentro de un Scaffold que carga el fondo de la aplicación y luego una columna que contiene el

titulo en la parte súperior y en la inferior la lista de inspecciones.

Dicha lista esta cargada previamente en su Notifier, por lo que solo hay que centrarse en como mostrar la lista y en la funcionalidad que vamos a añadir. Puesto que en el resto de pantallas se ha planteado así y ha funcionado de manera correcta se procederá a crear una tarjeta para cada inspección, dicha carta es muy similar a como se ha hecho en otras vista, salvo que en este caso el contenido va a estar dentro de una fila que estará dividida en tres parte, un texto, un cuadro y dos acciones.

Destacar también antes de pasar a hablar de sus tres parte que de manera previa al list view realizamos una ordenación de las inspecciones, primero en función de su estado para que se muestren primero las que estén ”en realización” y posteriormente las que estén ”cerradas”, y segundo en función de su fecha de inicio, mostrando primero las más nuevas.

En el caso del texto es un campo de texto similar a los vistos en otras pantallas con el titulo de la inspección.

El cuadro es un contenedor con color variable en función del estado de la inspección, en caso de que el estado sea ”en realización” será de color azul, y en caso de que el estado sea ”cerrada” se mostrará gris.

Por último, quedan los dos iconos pulsables que contienen la funcionalidad de cada tarjeta, uno para acceder a la realización de la inspección y otro para generar el informe.

En caso del primero no hay mucho que mencionar ya que su funcionalidad es simplemente la de redireccionar a la pantalla de lista de riesgos evaluados.

El segundo botón es el utilizado para generar un archivo csv.

6.11. Funcionalidad de súper usuario

Por último, nos queda hablar de la funcionalidad del usuario tipo administrados, en lo que respecta a dicha funcionalidad solo serán dos pantallas, y serán independientes a las del inspector, es decir, el usuario administrador podrá acceder a la funcionalidad del inspector y este no podrá acceder a la del administrador, a excepción de la funcionalidad de gestión de usuario que será igual para ambos tipos de usuario.

Por ello, cuando el usuario administrador accede a la aplicación la pantalla principal será diferente a la que se le muestre al inspector (para más información consultar diseño página principal).

6.11.1. Pantalla lista de inspectores

Una vez dicho esto, pasemos a ver la primera pantalla que se añade a la funcionalidad del usuario administrador, la de lista de inspectores, la cual muestra una lista de todos los inspectores dados de alta en la aplicación junto a una pequeña información de cada uno, dando a su vez la posibilidad de acceder a cada uno de ellos.

6.11.2. Pantalla información de inspector

La segunda pantalla que se añade será la de información del inspector, a la cual se accederá pulsando en alguna de las cartas de la lista de inspectores y es una pantalla que incluye información referente al inspector junto con un botón para dar de baja a dicho inspector y un cuadro de texto para indicar la razón.

Capítulo 7

Pruebas

7.1. Introducción

Antes de empezar a ver las pruebas realizadas para la aplicación, vamos a ver los ”principios del testing” para ayudar además a entender la razón por la que he optado por realizar las pruebas como lo he hecho.

Para ello, es muy importante entender que actualmente hay tres tipos de test muy comunes a la hora de realizar pruebas, empezando por una explicación más genérica para luego centrarnos más en las pruebas con Flutter:

- **Unit Test o Test Unitarios:** Se encargan de probar una unidad en especifico. [Vet]
- **Test de integración:** Verifica que dos o más unidades estén funcionando juntas correctamente como deberían. [Vet]
- **Test End to End:** Verifican que todas las unidades están funcionando correctamente.

El hecho de que existan estos tres tipos de test lleva a generar dudas referentes a cual se debería de usar para cada situación, y si es posible usar los mismos en todos los casos.

Por ello, apareció la pirámide del testing, que fue creada por Mike Cohn en su libro, Succeeding with Agile. Y en este, comenta que los unit test deberían ser la base de la pirámide, seguidos por los test de integración y finalmente los End to End. Según la Pirámide de Mike Cohn el grueso del testing de cualquier proyecto deberían ser las Pruebas Unitarias, luego las Pruebas de Componentes, Integración y API. Y donde se debería poner menos esfuerzo será en las de Interfaz de Usuario (GUI). [Coh20]

Esto se debe a que los test tratan de anticiparse a los errores y mientras que un error de bajo nivel, es decir en la lógica de la aplicación afecta tanto a la lógica como el diseño uno que se produce en el diseño afectará solo a si mismo. Además, no es difícil darse cuenta que es mucho más fácil que se produzcan errores a más bajo nivel y estos a su vez son mucho más difíciles de detectar que los producidos en las interfaces.

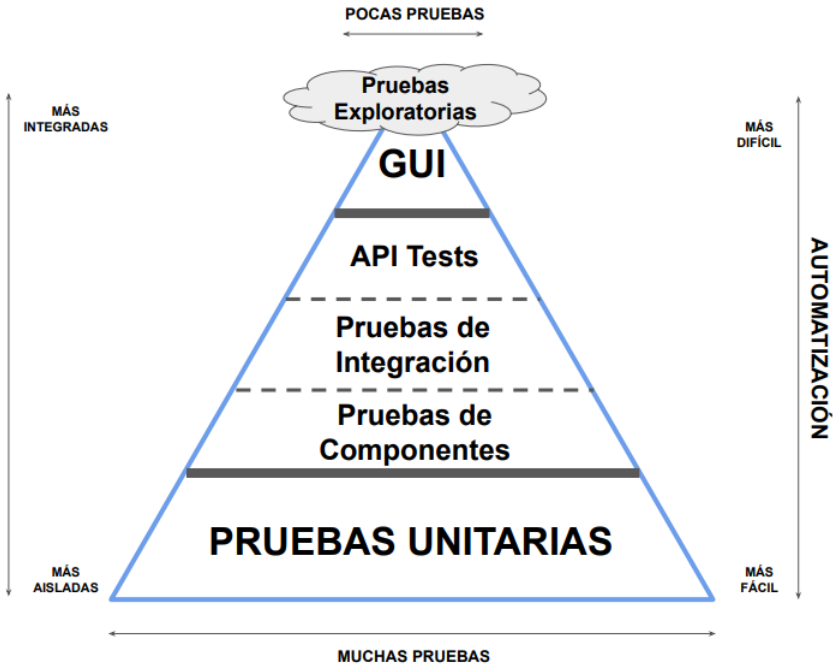


Figura 7.1: pirámide del agile testing

Una vez visto esto podemos pasar a hablar más especifico de como son los test en Flutter y los diferentes test que he realizado para la aplicación.

7.2. Testing en Flutter

Como hemos visto durante la introducción los test se dividen en tres tipos: unitarios, de integración y de interfaz, pero cuando entramos en cada tipo hay otros casos, por ejemplo: en el peldaño de la pirámide en el que se encuentran los de integración también están incluidos los de APIs y los de componentes. En el caso de Flutter en los test de integración también existen los de Widgets, no son lo mismo pero se les podría englobar en el segundo escalón de la pirámide.

Por lo que en Flutter nos quedaría una pirámide como la siguiente (siguiendo la lógica de Mike Cohn) Vamos a hablar más en detalle de cada uno de ellos:

- **Unit Test:** prueba una sola función, método o clase. El objetivo de una prueba unitaria es verificar la exactitud de una unidad lógica bajo una variedad de condiciones. Las dependencias externas de la unidad bajo prueba son generalmente mocked out. Las pruebas unitarias generalmente no leen ni escriben en disco, ni se renderizan en pantalla, ni reciben acciones del usuario desde fuera del proceso que ejecuta la prueba. [Flue]
- **Widget Test:** En otros Frameworks de UI se refieren a ellas como component test, prueba un único Widget. El objetivo de una prueba de Widgets es verificar que la UI

del Widget se vea e interactúe como se espera. Probar un Widget implica múltiples clases y requiere un entorno de prueba que proporcione el contexto apropiado del ciclo de vida del Widget. [Flue]

El Widget que se está probando debería ser capaz de recibir y responder a las acciones y eventos del usuario, realizar el diseño e instanciar Widgets child. [Flue]

- **Integration Test:** Prueba una aplicación completa o una gran parte de la aplicación. El objetivo de una prueba de integración es verificar que todos los Widgets y servicios que se están probando funcionan juntos de la forma esperada. Además, puedes usar pruebas de integración para verificar el rendimiento de tu aplicación. [Flue]

  Generalmente, un integration test se ejecuta en un dispositivo real o en un emulador de SO, como iOS Simulator o Android Emulator. La aplicación bajo prueba se aísla típicamente del código del controlador de la prueba para evitar sesgar los resultados. [Flue]

	Unit	Widget	Integration
Confianza	Low	Higher	Highest
Costo de Mantenimiento	Low	Higher	Highest
Dependencias	Few	More	Most
Rapidez de ejecución	Quick	Slower	Slowest

Figura 7.2: tests en Flutter

7.3. Test unitarios

En el caso de los test unitarios, puesto que el resto de funcionalidad se está probando en los test de integración, vamos a probar solo la funcionalidad de los modelos: los métodos de convertir de mapa a objeto cuando se lee de Firebase y la funcionalidad de la lógica fuera de operaciones a base de datos (funciones de ordenado de arrays, calculo de ids, calculo de nivel de riesgo...).

7.3.1. Test de modelos

En el caso de estos test, puesto que probar los setters y getters y probar el propio constructor se contemplo como algo innecesario se prueba solo el método utilizado para convertir un mapa en un objeto:

```
group('Metodo from map', () {
    test('riesgo', () async {
      QuerySnapshot snapshot=await FirebaseFirestore.instance
        .collection('riesgo')
        .where('idInspeccion', isEqualTo: 1)
        .get();
      snapshot.docs.forEach((document) {
        SubRiesgo r=SubRiesgo.fromMap(document.data());
        expect(r.idUnica, 1);
      });
    });
```

Listing 7.1: Test fromMap de los modelos

Esto tiene que funcionar así debido a que la lectura de la base de datos devuelve un mapa, y para evitar tener que realizarlo cada vez que se hace una lectura de la base de datos decidí crear un método fromMap para cada modelo que realizase esta conversión. Puesto que todos los modelos excepto el de provincias cuentan con el método fromMap, se prueba en todos para comprobar así que la conversión después de leer de la base de datos se realiza bien en todos los casos.

7.3.2. Test de operaciones

En el caso de estos test se han probado las siguiente funcionalidades:

- **Ordenar inspecciones por fecha:** Que las inspecciones se muestren ordenadas en función de su fecha de inicio, primero las más actuales y después las más antiguas.
- **Ordenar inspecciones por estado:** Que las inspecciones se muestren ordenadas en función de su estado, primero las que están en realización y posteriormente las que están cerradas.
- **Ordenar provincias alfabéticamente:** Que las provincias al agregar una nueva inspección (en el dropdown) se muestren ordenadas alfabéticamente.
- **Calcular id única de evaluación:** Coge el id de la última evaluación añadida y le suma uno, la idea es hacer un autoIncrement de MySql de manera manual.
- **Calcular id única de riesgo:** Coge el id del último riesgo añadido y le suma uno.
- **Calcular id inspección:** Coge el id de la última inspección y le suma uno.
- **Calculo del nivel de probabilidad:** Función que se encarga de calcular el nivel de probabilidad de un riesgo.
- **Calculo del nivel de riesgo:** Función que se encarga de calcular el nivel final del riesgo.

eR ->enRealizacion ; c ->cerrada

Cuadro 7.1: Test de modelos

Entrada	Salida esperada	Salida obtenida
Fechas lista inspecciones: 1- 12-02-2020, 2- 11-02-2020, 3- 10-02-2020, 4- 13-02-2020	4-1-2-3	4-1-2-3
Estado lista inspecciones: 1- 12-02-2020 (eR), 2- 11-02-2020 (eR), 3- 13-02-2020(c)	1-2-3	1-2-3
Lista provincias: 1- Valladolid, 2- Badajoz, 3- Gerona	2-3-1	2-3-1
Ids lista evaluaciones: id 1- 4, id 2- 3, id 3- 5, id 4- 6	7	7
Ids lista riesgos: id 1- 4, id 2- 3, id 3- 5, id 4- 6	7	7
Ids lista inspecciones: id 1- 4, id 2- 3, id 3- 5, id 4- 6	7	7
nivel Deficiencia: 6, nivel Exposición: 2	12	12
nivel Probabilidad: 12, nivel Consecuencias: 100	1200	1200

Cuadro 7.2: Test de operaciones

Entrada	Salida esperada	Salida obtenida
Map<String, dynamic>mapa = new Map(); mapa['id'] = 1; mapa['nombre'] = 'a';	riesgo.id=1, riesgo.nombre='a'	riesgo.id=1, riesgo.nombre='a'
Map<String, dynamic>mapa = new Map(); mapa['id'] = 1; mapa['nombre'] = 'a';	evaluacion.id=1, evaluacion.nombre='a'	evaluacion.id=1, evaluacion.nombre='a'
Map<String, dynamic>mapa = new Map(); mapa['url'] = 'a';	fotoRiesgo.url='a'	fotoRiesgo.url='a'
Map<String, dynamic>mapa = new Map(); mapa['email'] = 'a'; mapa['motivo'] = 'a';	Usuario.email='a', Usuario.motivo='a'	Usuario.email='a', Usuario.motivo='a'
Map<String, dynamic>mapa = new Map(); mapa['id'] = 1; mapa['lugar'] = 'a';	Inspeccion.id=1, Inspeccion.lugar='a'	Inspeccion.id=1, Inspeccion.lugar='a'

7.4. Test de integración

Serán los encargados de probar la funcionalidad de los casos de uso. Para ello, vamos a realizar que tests que ejecuten el programa siguiendo una secuencia normal de cada uno de los casos de usado probando así su correcto funcionamiento.

Para ello, tendremos que hacer test de drivers, los cuales necesitan cargar previamente un contexto para ejecutarse. Para ello, la manera de proceder es crear dos clases por cada test, la primera de ellas la llamaremos por ejemplo login-registro.Dart y será la encargada de cargar el contexto y la segunda la llamaremos login-registro_test.Dart. De esta manera cuando ejecutemos el test:

Flutter drive –target=test_driver/login-registro.Dart

Antes de empezar a realizar las pruebas de la clase login-registro_test.Dart se ejecutará la clase login-registro.Dart cargando así el contexto en el emulador y permitiendo que se ejecuten dichos test.

Con cargar el contexto no solo nos referimos a cargar la pantalla en el emulador, sino que además cargaremos los Notifiers necesarios para la ejecución de las pruebas. Veamos un ejemplo de código para entender esto mejor:

```
enableFlutterDriverExtension();

runaplicación(
    MultiProvider(
        Providers: [
            ChangeNotifierProvider(
              create: (context) => AuthNotifier(),
            ),
            ChangeNotifierProvider(
              create: (context) => UsuarioNotifier(),
            ),
        ],
        child: Materialaplicación(
            home: Builder(
            builder: (context) {
                UsuarioNotifier userNotifier =
                Provider.of<UsuarioNotifier>(context, listen: false);
                Usuario u=new Usuario(
                    "a2@gmail.com",
                    "654321",
                    "333 333 333",
                    "222222222A",
                    "prueba prueba",
                    "https://FirebaseStorage.googleapis.com/v0/b/tfg-
   riesgos-laborales-f85a8.aplicaciónspot.com/o/3a25d5b9-9753-4705-
   b926-d6a2b691a6804804850712917333266.jpg?alt=media&token=d9dc2f8b
   -7241-4bc7-9cd4-c890817723d6",
                    "inspector",
                    "wdqwdqwdqwd");
                userNotifier.currentUser=u;
                return CambiarPass();
```

Listing 7.2: Cargar contexto driver test

Mencionar antes de empezar a ver los diferentes test, que en este caso no vamos a ver los resultados con tablas como hemos hecho con los unitarios, sino más bien con explicación de la utilidad y lo que se espera de cada uno, ya que al estar probando las propias pantallas algunos test, simplemente comprueban que se haya añadido bien contenido a un campo de texto.

Por ello, al no ser un resultado tan claro no creo que sea una buena manera de mostrarlo a través de resultado esperado y resultado obtenido, por lo que vamos a explicar que se está probando en cada uno de los casos. Además, importante mencionar que todos los test probados a continuación se han ejecutado de manera correcta.

Una vez entendido el funcionamiento de estos test, vamos a pasar a ver que funcionalidades de la aplicación se han probado con ellos:

- **Prueba de inicio de sesión (registro y login):**
  - Inicio de sesión correcto del usuario administrador.
  - Inicio de sesión correcto del usuario inspector.
  - Navegabilidad login-registro y registro-login.
  - Funcionalidad de restablecer contraseña.
  - Error por no rellenar todos los campos del login.
  - Error por no rellenar todos los campos del registro.
  - Error por no rellenar con el mismo contenido el campo de contraseña y contraseña repetida.
  - Error por intentar registrar un email ya existente.
  - Registro exitoso.
- **Prueba de cambio de contraseña:**
  - Error por no rellenar los tres campos (son necesarios).
  - Error por no introducir la contraseña actual correcta.
  - Error por no introducir el mismo valor en el campo de nueva contraseña y confirmar nueva contraseña.
  - Error por introducir la nueva contraseña igual a la actual.
  - Error por introducir contraseñas de menos de seis caracteres.
  - Cambio de contraseña correcto.
- **Prueba de moficar perfil:**
  - Comprobar que se han cargado los datos del usuario.
  - Cambiar dirección de manera exitosa.
  - Cambiar email de manera exitosa.
  - Cambiar telefono de manera exitosa.
  - Probar botón de cancelar

- **Prueba de añadir inspección**
  - Error por no rellenar todos los campos necesarios de la inspección.
  - Agregar inspección de manera exitosa.
- **Pruebas de evaluar un riesgo:**
  - Se muestra el riesgo cargado en el contexto.
  - Error por no rellenar todos los campos necesarios de la evaluación.
  - Evaluación realizada de manera correcta
- **Prueba de dar de baja inspector**
  - Comprobar que se han cargado los inspectores de manera correcta.
  - Comprobar que se ha cargado la información del inspector de manera correcta.
  - Dar de baja de manera exitosa.

7.5. Pruebas end-to-end

Por último, para asegurarnos del correcto funcionamiento de la aplicación, de manera previa al envío y tras haber realizado los últimos cambios, probamos que los cambios realizados no han afectado al funcionamiento de lo implementado correctamente antes de la entrega final, y que las nuevas funcionalidades se han implementado de manera correcta:

- **Inicio:**
  - Introducir email: a@gmail.com contra: 111111 ->Se muestre en un mensaje que el email y/o la contraseña son incorrectos.
  - Introducir email: a@gmail.com contra: ”vació” ->Se muestre en un mensaje que se deben rellenar todos los campos para iniciar sesión.
  - Acceder a la opción de restablecer contraseña, introducir en el cuadro de texto a.romeropacho30@gmail.com y enviar ->Se muestre en un mensaje que el email ha sido enviado y llegue al correo introducido.
  - Acceder a la opción de registro, rellenar todos los campos salvo uno ->Se muestre en un mensaje que se deben rellenar todos los campos para registrarse.
  - Rellenar todos los campos introduciendo una contraseña diferente en contraseña y en repetir contraseña ->Se muestre en un mensaje que el campo de contraseña y confirmar contraseña deben coincidir.
  - Rellenar todos los campos introduciendo en el campo de email a@gmail.com ->Se muestre en un mensaje que el email ya está registrado.
  - Rellenar todos los campos introduciendo en el campo de email otro correo diferente. Ej : av2@gmail.com ->Se registre correctamente
  - Volver a la sección de login y introducir email: a@gmail.com contra:654321 ->Acceda correctamente redireccionando a la pantalla de página principal

- **Gestión de usuario:**
  - Desplegar el menú y ir a la pantalla de cambio de contraseña, rellenar los campos: Contraseña actual: 654321 Nueva contraseña: 123456, el último campo en blanco ->Se muestre en el campo de texto que esta sin rellenar que es necesario rellenarlo.
  - Rellenar los campos: Contraseña actual: 123456 Nueva contraseña: 654321 Repita la nueva contraseña: 654321 ->Se muestre en el campo de contraseña actual que la contraseña no coincide con la de el usuario y en el campo de nueva contraseña que la contraseña nueva debe de ser diferente a la anterior.
  - Rellenar los campos: Contraseña actual: 654321 Nueva contraseña: 123456 Repita la nueva contraseña: 1234567 ->Se muestre en el campo de contraseña actual que los campos de nueva contraseña y repita la nueva contraseña deben de coincidir.
  - Rellenar los campos: Contraseña actual: 654321 Nueva contraseña: 123456 Repita la nueva contraseña: 123456 ->Se muestre en un mensaje que la contraseña ha sido cambiada con éxito ->Desplegar menú y ir a la opción de cerrar sesión, volver a iniciar sesión pero esta vez email: a@gmail.com y contraseña: 123456 ->Inicie sesión correctamente.
  - Desplegar menú y ir a la pantalla de perfil, una vez alli pulsar en el icono de la cámara y seleccionar la opción de ”Camera”, sacar una foto y pulsar el tick para confirmar ->Ver como la imagen tomada hace unos instantes se muestra en el circulo.
  - Pulsar el icono de la cámara y seleccionar la opción de ”Gallery” ->Ver como la imagen cargada de la galería se muestra en el circulo.
  - Pulsar el botón de modificar perfil para ir a la pantalla de modificar perfil, una vez ahí hacer como en los casos anteriores para probar que la imagen se cambia correctamente desde aquí también.
  - Cambiar uno de los campos y darle a cancelar, esto nos redireccionará a la pantalla de perfil, si entramos nuevamente en la pantalla de modificar perfil los cambios que realizamos no deberían de verse reflejados.
  - Cambiar uno de los que no sea el email y darle a guardar, esto nos redireccionará a la pantalla de perfil y si volvemos a entrar nuevamente en la de modificar perfil los cambios deberían verse reflejados.
  - En el campo de email cambiar lo escrito actualmente por av2@gmail.com, darle a guardar y cerrar sesión y acceder con email: av2@gmail.com y contra: 123456 ->Inicie sesión correctamente.

- **Añadir nueva inspección:**
  - Ir a la pantalla de página principal y pulsar el botón de añadir inspección, esto nos redireccionará a la pantalla de añadir una nueva inspección, si pulsamos el botón de crear nueva inspección se mostrará un cuadro con diferentes campos, pulsamos en el botón cancelar ->Esconderá el cuadro.
  - Pulsar el botón de crear nueva inspección y darle a confirmar ->Muestre mensajes debajo de cada campo de texto indicando que se tienen que rellenar para crear la inspección.
  - Pulsar el icono del mapa, esto desplegará un mapa centrado en nuestra ubicación. Seleccionar un punto del mapa y ver como esta dirección se muestra en el cuadro de texto de debajo y pulsar el botón de confirmar ->El cuadro de texto de Dirección se rellenará automáticamente con la dirección seleccionada en el mapa.
  - Rellenar el resto de campos y darle a confirmar ->Esto nos llevará a la pantalla de lista de riesgo evaluados teniendo como titulo en la parte superior el nombre que le hemos dado a dicha inspección.
- **Lista de inspecciones:**
  - Desde la pantalla de página principal ir a la opción de inspecciones, la cual mostrará una lista con las inspecciones de ese usuario ->La que acabamos de añadir debería de salir la primera (están ordenadas de más reciente a más antigua).
  - Darle al icono del lapicero en la inspección y añadir un nuevo riesgo, volver atrás y volver a pulsar el icono del lapicero ->El riesgo que se ha añadido debería de estar ahí.
  - Entrar en la inspección (pulsando el icono del lapicero) y pulsar el botón de finalizar inspección ->si volvemos a la lista de inspecciones esta estará la primera de las que tienen un recuadro gris después de todas las que tienen un recuadro azul. Además, el icono del lapicero ya no aparecerá sobre la carta.
  - Pulsar en el icono del documento de la inspección ->Se muestra un mensaje en el que dice que se ha creado el fichero csv en la carpeta downloads del teléfono, si vamos a la carpeta de downloads el fichero csv debería de estar ahí, en caso de que queramos validar su contenido tendremos que pasarlo al ordenador y abrirlo con excel.

- **Añadir un riesgo:**
    - Una vez en la pantalla de lista de riesgos evaluados vamos a pulsar el botón de agregar riesgo y seleccionar una de las cartas, lo que nos llevará a otra pantalla en la que se muestran nuevas cartas, al seleccionar una de estas pasaremos a la pantalla de evaluar riesgo ->Comprobar que en la parte superior debajo de Evaluación de Riesgo aparece el nombre de la carta seleccionada.
    - Darle a guardar ->Muestre mensajes debajo de cada campo de texto indicando que se tienen que rellenar para completar la evaluación.
    - Pulsar el botón de GPS de al lado de los campos de altitud, longitud y latitud ->Ver como se rellenan los tres campos.
    - Rellenar el resto de campos sin mover los sliders de los niveles y pulsar el botón de añadir foto, añadir una nueva foto desde la cámara ->Ver como la imagen se muestra debajo del botón.
    - Pulsar nuevamente el botón de añadir foto, añadir una nueva foto desde la galería ->Ver como se muestra también debajo del botón al lado de la primera y pulsar el botón de guardar ->Nos llevará a la vista de lista de riesgos por evaluar y se mostrará una tarjeta con la foto del riesgo y el nombre del riesgo, además rodeando la foto habrá un marco de color verde.
    - Pulsar el botón de la papelera y ver como la carta desaparece.
    - Volver a añadir un riesgo pero esta vez cambiando el nivel de deficiencia a 2 en lugar de 0 y darle a guardar ->Se mostrará una carta similar a la anterior pero con el marco de color amarillo.
    - Ahora en vez de añadir un nuevo riesgo vamos a modificar el que acabamos de añadir, para ello, tendremos que pulsar en la carta y modificar alguno de sus campos, el titulo por ejemplo y darle a guardar ->Volver a entrar en la carta y comprobar que el cambio se ha realizado con éxito.
    - Volver a modificar esa evaluación, en este caso cambiando el nivel de deficiencia a 10 y el de exposición a 2 ->el marco ha cambiado a color naranja.
    - Modificar nuevamente esa evaluación, cambiar el nivel de consecuencia a 25 y darle a guardar ->el marco ha cambiado a color rojo.

- **Dar de baja inspector**
    - Cerrar sesión y iniciar nuevamente introduciendo email: súper@gmail.com y contraseña: 123456 ->se deberá mostrar la pantalla de página principal pero en lugar de tener dos botones tendrá solo uno (inspectores).
    - Comprobar que la funcionalidad de gestionar usuario (cambio de contraseña, cambio de email, cambio de foto...) funciona bien al igual que lo hace con los usuarios de tipo inspector ->funcione correctamente.
    - Pulsar en el botón de inspectores ->debería de mostrar una lista con todos los usuarios de tipo inspector registrados en la aplicación.
    - Pulsar una de las cartas para entrar en la información de uno de los inspectores ->se debería de mostrar una pantalla con la información de ese usuario (incluida la imagen que este tiene cargada).
    - Añadir un texto en el recuadro de motivo baja y pulsar el botón de dar de baja inspector ->al cerrar sesión y intentar iniciar con el usuario que ha sido dado de baja no nos dejará.
    - Pulsar el botón de dar de baja inspector ->al igual que en el anterior caso tampoco dejará iniciar sesión con ese usuario (no es necesario añadir un motivo para dar de baja).

Cabe destacar, que todas las pruebas end-to-end se han probado:

- En modo debug: tanto en el emulador (fallando las relacionadas con la ubicación a causa de que el ordenador no tiene acceso a esta herramienta), como en un dispositivo móvil físico. En estas últimas, las pruebas relacionadas con la ubicación han resultado exitosas.
- En modo release: tras generar la apk en un dispositivo físico. El resultado obtenido de estas últimas no solo ha sido bueno puesto que las ha pasado todas, sino que también ha demostrado que el rendimiento de la aplicación en modo release es casí perfecto, teniendo sus únicos retrasos debido a las cargas de base de datos.

Capítulo 8

Conclusiones y mejoras futuras

8.1. Conclusiones

Tras finalizar el proyecto, se puede considerar que se ha realizado con éxito puesto que se han alcanzado todos los objetivos establecidos en un principio y se han realizado todas las fases del proyecto en un orden adecuado y de la manera que se harían en un proyecto real.

Se ha logrado crear satisfactoriamente un programa para la realización de inspecciones de riesgos laborales con una interfaz intuitiva y práctica. Además, se ha logrado realizar con una base de datos en la nube, lo que permite su ejecución desde cualquier dispositivo manteniendo los datos guardados y actualizados de manera gratuita en todo momento.

En cuanto a los objetivos personales se ha cumplido todo lo esperado:

- He adquirido conocimientos relativamente avanzados (todo lo posible habiendo realizado solo un proyecto) acerca del Framework Flutter, del que en un principio no sabía nada y actualmente soy capaz de desarrollar una aplicación partiendo desde cero.

- He aprendido a realizar test de integración en una aplicación con interfaz gráfica, en este caso pese a que nunca antes había realizado este tipo de test no creo que partiera de cero, al menos conceptualmente, ya que durante la carrera hemos tratado el ámbito de las pruebas en diferentes asignaturas.

- He reforzado los conocimientos sobre el uso de la base de datos Firebase, y muchas de las librerías para la realización de aplicaciones móviles (uso de ubicación, cámara, galería...).

- He tenido la oportunidad de realizar por primera vez un proyecto software desde cero siguiendo todas sus fases: planificación inicial, análisis, diseño, implementación y pruebas.

8.2. Líneas futuras

Pese a que el proyecto es completo, existen ciertos puntos que podrían ser objeto de mejora en un futuro:

- En el informe descargado de la inspección, sería una buena mejora que diera la opción de descargarlo en diferentes formatos, por ejemplo, estaría bien que se pudiera decidir si descargarlo en PDF o Word además del csv. Además, sería interesante que en caso de descargar un documento de texto en vez de incluir el enlace a las imágenes se incluyeran las mismas imágenes.
- También, sería una buena mejora que el escalado de la aplicación fuera más óptimo, de esta manera podría ser utilizada en móviles de tamaños exagerados y en tablets. Actualmente, algunas de las pantallas no estas correctamente escaladas, por lo que al cambiar el tamaño del dispositivo algunos de los elementos del diseño se descuadran.
- Posibles cambios en las interfaces en caso de que el cliente lo requiera, actualmente están realizadas de manera correcta pero es posible que el cliente quiera cambiar algo una vez lo pruebe.
- Aumentar la funcionalidad de gestión de bajas, una posible mejora sería dar la posibilidad al súper usuario de dar de alta a un usuario que se encuentra dado de baja y que cuando un usuario dado de baja intente iniciar sesión se le notifique que está dado de baja junto a un mensaje que indique el motivo.
- También podría ser una buena mejora estudiar la posibilidad de alojar la base de datos en un servidor, utilizando Firebase no hay problemas a la hora de funcionar, puesto que pese a ser gratuita da un buen servicio, pero tiene un retraso para cargar datos cuando se está utilizando desde un dispositivo móvil en lugar del emulador. Además, una vez pasan los meses del periodo de prueba no hay una forma gratuita de mantener los datos seguros, puesto que Firebase hace pública tu base de datos.

Apéndice A

Manual de usuario

En este apartado se explica el proceso de generación de una APK, tanto para Android como para iOS y un pequeño tutorial de como se utiliza la aplicación.

A.1. Android

Para generar la apk de Android se utiliza el comando:

flutter build apk –build-name=1.0.1 –build-number=1

El proceso por lo general suele tardar unos minutos. Al final se mostrará el directorio donde se ha generado la APK.

A.2. iOS

Para generar la APK de IOs se usa el comando:

Flutter build ios –release

El resultado será el mismo que para Android.

A.3. Funcionamiento de la aplicación

La aplicación permite la documentación de las inspecciones para la evaluación y Prevención de Riesgos Laborales que se realicen en distintos lugares, así como la generación de un informe resumen de la evaluación y la posibilidad de acceder a la pantalla de registro.

Cuando entras en la aplicación te encuentras con un Login, en el cual además se puede mostrar una pestaña que pide introducir tu email para recuperar la contraseña:

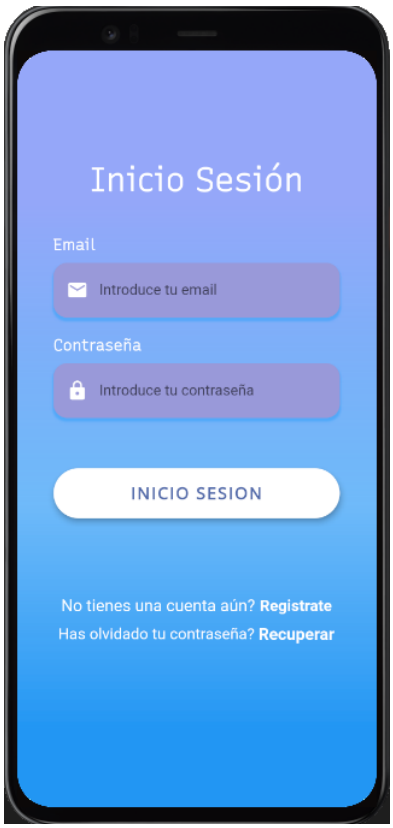


Figura A.1: pantalla de login

Una vez iniciado sesión, se le enviará a la pantalla de página principal (en caso de que se quiera acceder con el usuario administrador tendrá que solicitar uno para que se añada manualmente a la base de datos), en caso de que se inicie sesión como usuario administrado, la funcionalidad será diferente a la que se tiene con un usuario registrado. En caso de que el usuario no recuerde la contraseña, podrá recuperarla introduciendo su email, siempre y cuando se haya registrado con un correo real:

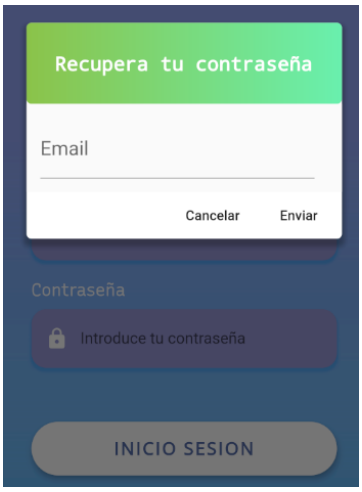


Figura A.2: recuperar contraseña con email

En caso de que el usuario decida restablecer su contraseña, tendrá que introducir el correo electrónico con el que se registro, y en caso de que este email fuera real le llegará el siguiente correo:

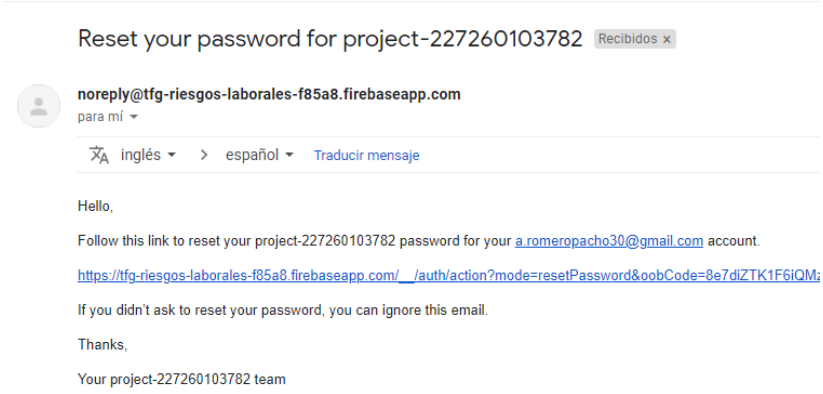


Figura A.3: correo de recuperación de contraseña

Si el usuario tarda demasiado en acceder al link para restablecer su contraseña se le mostrará el siguiente mensaje:

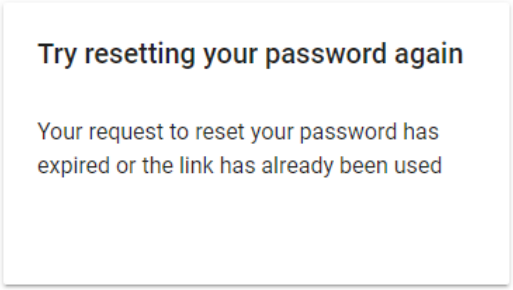


Figura A.4: mensaje de notificación de expiración

En caso contrario:

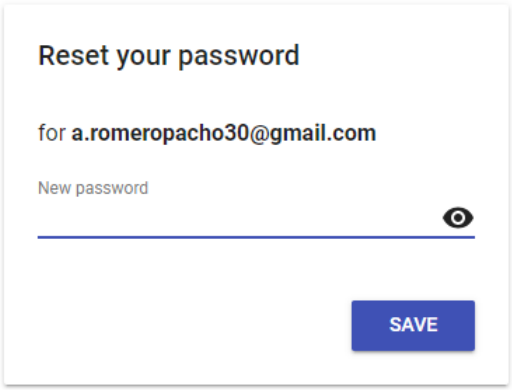


Figura A.5: pantalla de restablecimiento de contraseña

En caso de que el usuario aún no tenga una cuenta, tendrá que acceder a la página de registro y introducir toda la información para crearse una cuenta:

Figura A.6: pantalla de registro

en caso de que acceda con un usuario registrado, las funcionalidades son gestionar su lista de inspecciones y añadir una nueva:

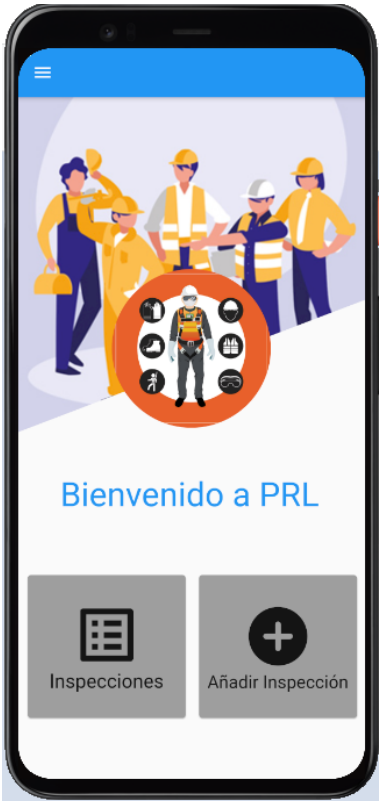


Figura A.7: pantalla de página principal usuario inspector

En caso de que se inicie sesión con el usuario administrado, la funcionalidad será la de gestionar inspectores (teniendo la posibilidad de darlos de baja):

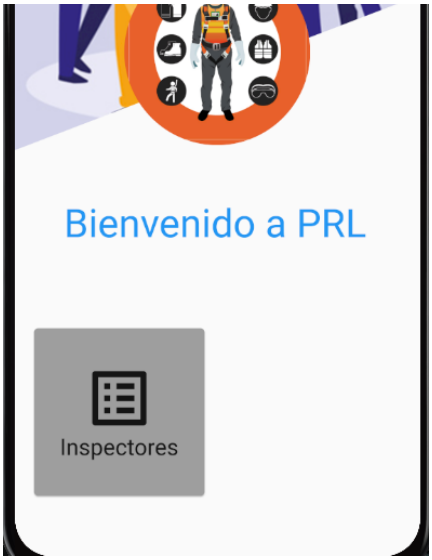


Figura A.8: pantalla de página principal usuario administrador

Si el usuario decide agregar una nueva inspección, se le mostrará una pantalla con un botón de agregar inspección, el cual al presionarlo se le mostrará un cuadro con todos los campos por rellenar para crear la inspección:

Figura A.9: pantalla de añadir inspección

Para rellenar los datos vamos a necesitar:

- **Nombre:** Un texto escrito. (obligatorio)
- **Dirección:** Se puede introducir una dirección manualmente o acceder al mapa a través del icono de la derecha del cuadro de texto. (obligatorio)
- **Empresa:** Un texto escrito. (obligatorio)
- **Provincia:** Seleccionar una opción. (obligatorio)
- **Nombre:** Un texto escrito. (obligatorio)

En caso de que se pulse el icono del mapa, se redireccionará a una nueva pantalla:

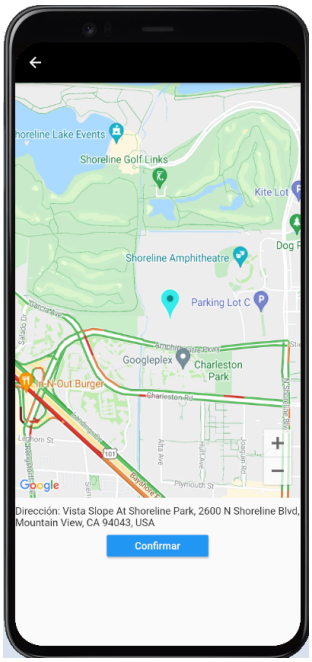


Figura A.10: pantalla de mapa

Una vez creada la inspección, se redreccionará al usuario a la pantalla que muestra la lista de riesgos evaluados, en la que se podrá finalizar la inspección o añadir un nuevo riesgo. En caso de que ya haya algún riesgo evaluado se podrá acceder a su evaluación para modificarla:

Figura A.11: pantalla de lista de riesgos evaluados

En caso de que se quiera añadir un nuevo riesgo, se le mostrará primero una pantalla para elegir la categoría que más se adecue al riesgo encontrado (podrá seleccionarla pulsando en una de las cartas):

Figura A.12: pantalla de selección de categoría

Una vez elegida la categoría, se mostrarán los riesgos de esa categoría (podrás seleccionar uno pulsando en su carta):

Figura A.13: pantalla de selección de riesgo

Una vez el usuario seleccione un riesgo, se le mostrará una pantalla con unos campos a rellenar para evaluar dicho riesgo:

Figura A.14: pantalla de evaluar riesgo

Para rellenar los datos vamos a necesitar:

- **Titulo:** Un texto escrito. (obligatorio)
- **Tipo Factor:** Seleccionar una opción. (obligatorio)
- **Nivel de Deficiencia:** Barra para indicar el nivel de deficiencia. (obligatorio)
- **Nivel de Exposición:** Barra para indicar el nivel de exposición. (obligatorio)
- **Nivel de Consecuencias:** Barra para indicar el nivel de consecuencias. (obligatorio)
- **Acción Correctora:** Un texto escrito. (obligatorio)
- **Añadir foto:** Un botón que al pulsarlo da la opción de cargar una imagen desde la cámara o desde la galería. (opcional)

Una vez visto esto, volvamos a la página principal para ver el funcionamiento de la otra opción del menú (inspecciones). Al acceder a ella muestra la lista de inspecciones de ese usuario:

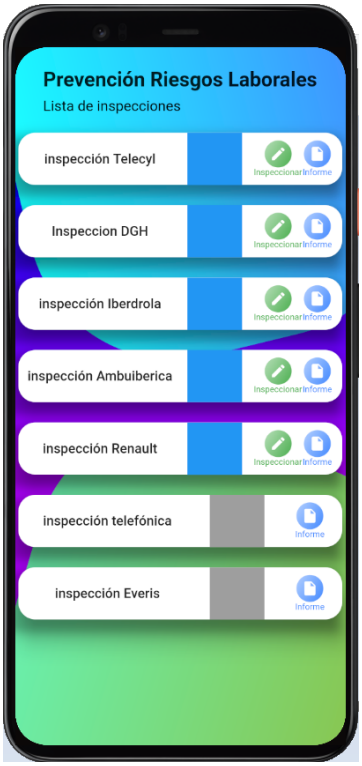


Figura A.15: pantalla de lista de inspecciones

En esta pantalla, se podrá decidir entre acceder a la inspección que aun está en realización para poder añadir nuevos riesgos o generar un informe con la inspección. Dicho informe se descargará en la carpeta de descargas del teléfono.

Por último, antes de pasar a ver la funcionalidad del usuario administrador vamos a ver el menú lateral con sus diferentes funciones:

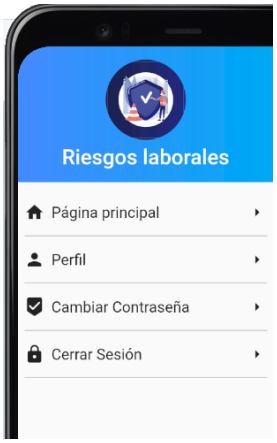


Figura A.16: pantalla de menú

En este menú, solo nos quedarían dos funcionalidades por ver, la de cambio de contraseña:

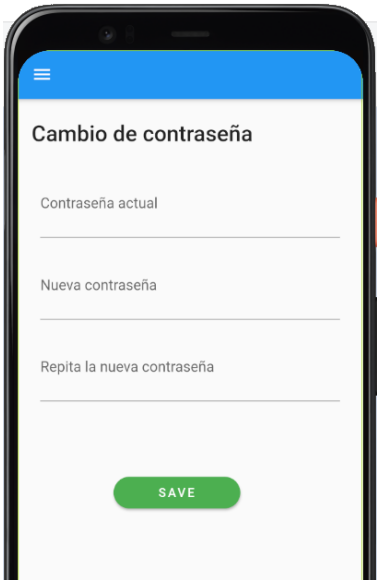


Figura A.17: pantalla de cambio de contraseña

Y la de modificar perfil:

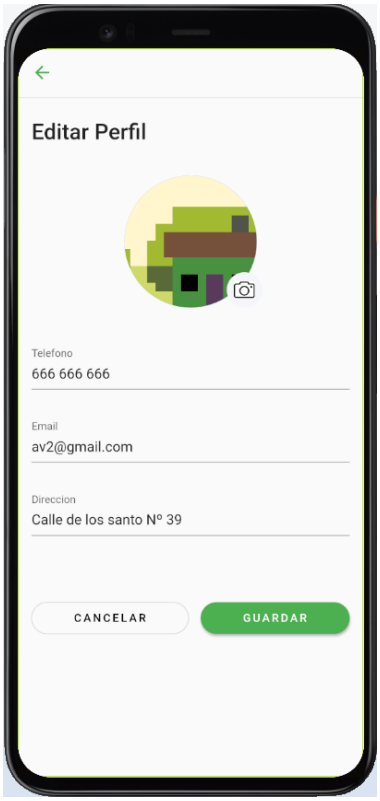


Figura A.18: pantalla de modificación de perfil

Por último, queda de ver la parte del súper usuario, que se diferencia con respecto a la del usuario inspector por su página principal y a su vez por consiguiente por la funcionalidad a la que este tiene acceso. Si pulsa el botón de inspectores de la página principal, le redireccionará a la lista de inspectores:

Figura A.19: pantalla de lista de inspectores

Una vez entre en una de las cartas de los inspectores, se le mostrará la información junto con la posibilidad de dar de baja al usuario y un cuadro de texto para introducir un motivo:

Figura A.20: pantalla de información de inspector

Una vez el usuario este dado de baja no podrá acceder a la aplicación.

Bibliografía

[34] Definir el alcance del proyecto en 5 pasos. https://www.recursosenprojectmanagement.com/definir-el-alcance/#Pasos_para_definir_el_alcance_de_un_proyecto. Accesed: 2021-02-18.

[4101] Método de william t. fine. https://upcommons.upc.edu/bitstream/handle/2099.1/18520/ANEXO%20I.William%20T.Fine.PDF?sequence=3&isAllowed=y, 2001. Accesed: 2021-02-14.

[4319] El modelo en cascada: desarrollo secuencial de software. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/, 2019. Accesed: 2021-02-19.

[AA20] Osama Al-Atraqchi. fetching data from firestore and display it in the flutter using streambuilder. https://www.youtube.com/watch?v=4-84lwBta08&ab_channel=OsamaAl-Atraqchi, 2020. Accesed: 2021-04-02.

[AEN21] AENOR. Iso 45001 de seguridad y salud en el trabajo. https://www.aenor.com/certificacion/riesgos-y-seguridad/seguridad-salud-trabajo-45001, 2021. Accesed: 2021-02-11.

[And18] Code With Andrea. Flutter layouts walkthrough: Pageview, listview, gridview, slivers, customscrollview. https://www.youtube.com/watch?v=-zJ6CnOVndE&ab_channel=CodeWithAndrea, 2018. Accesed: 2021-03-14.

[ATL] ATLASSIAN. Flujo de trabajo de gitflow. https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow. Accesed: 2021-05-12.

[Bar17] Marc Bara. Roles, eventos y artefactos en la metodología scrum. https://www.obsbusiness.school/blog/roles-eventos-y-artefactos-en-la-metodologia-scrum, 2017. Accesed: 2021-02-19.

[Bee15] Equipo BeeDIGITAL. Riesgo laboral en el trabajo: definición y cómo evitarlo. https://www.beedigital.es/prevencion-riesgos/que-es-el-riesgo-laboral-definicion-y-como-evitarlo/, 2015. Accesed: 2021-02-10.

[Bha20a] Sujan Bhattarai. firebase_auth. https://github.com/sujanbhattaraiofficial/firebase_auth/tree/master/firebaseAuth/lib, 2020. Accesed: 2021-03-04.

[Bha20b] Sujan Bhattarai. Flutter - firebase authentication with email password. https://www.youtube.com/watch?v=yj3plSl3qNs&ab_channel=SujanBhattarai, 2020. Accesed: 2021-03-04.

[Bla] José Antonio Blanes. ¿qué es react native? https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html. Accesed: 2021-03-15.

[Ble20] BleylDev. Reading lists from firestore using streambuilder in flutter. https://medium.com/quick-code/reading-lists-from-firestore-using-streambuilder-in-flutter-eda590f461ed, 2020. Accesed: 2021-05-25.

[Cod19a] Cheetah Coding. How to use firebase in flutter - part 1 (email auth). https://www.youtube.com/watch?v=bjMw89L61FI&ab_channel=CheetahCoding, 2019. Accesed: 2021-03-10.

[Cod19b] Cheetah Coding. How to use firebase in flutter - part 2 (retrieving data). https://www.youtube.com/watch?v=730eVDuN5zc&ab_channel=CheetahCoding, 2019. Accesed: 2021-03-14.

[Cod20] Bourbon Code. Blog app con flutter y firebase storage - firestore (parte 2). https://www.youtube.com/watch?v=n9UtE4RJ5C8&ab_channel=BourbonCode, 2020. Accesed: 2021-04-24.

[Cod21] Peter Coding. Using cloud firestore in flutter. https://petercoding.com/firebase/2020/04/04/using-cloud-firestore-in-flutter/, 2021. Accesed: 2021-02-25.

[Coh20] Mike Cohn. Agile testing. https://www.autentia.com/wp-content/uploads/2020/04/Agile-Testing.pdf, 2020. Accesed: 2021-06-15.

[Con19] Jesús Conde. Crear entorno de trabajo para desarrollar con flutter. https://www.youtube.com/watch?v=2jZ6BIavxjU&ab_channel=Jes,%C3%BAsConde, 2019. Accesed: 2021-02-20.

[con20] CTMA consultores. Riesgos laborales más frecuentes en las empresas. https://ctmaconsultores.com/riesgos-laborales/, 2020. Accesed: 2021-02-11.

[Cru20] Andrés Cruz. 20 onwillpop: Interceptar al darle cick al botón retroceso en flutter para ir a la página anterior. https://www.youtube.com/watch?v=DfgGP0OvhpE&ab_channel=Andr%C3%A9sCruz, 2020. Accesed: 2021-05-16.

[Dar21] Pinkesh Darji. Adding google maps to a flutter app. https://blog.logrocket.com/adding-google-maps-to-a-flutter-app/#:~:text=Adding%20Google%20Maps%20to%20Flutter%20app%20(Android)&text=First%2C%20open%20your%20Flutter%20project,xml%20.&text=Replace%20the%20value%20%22YOUR%20KEY,Then%2C%20add%20the%20location%20permission., 2021. Accesed: 2021-05-15.

[dev19] Quality devs. Qué es ionic y por qué te interesa conocerlo si eres desarrollador web. https://www.qualitydevs.com/2019/05/31/que-es-ionic-desarrollador-web/, 2019. Accesed: 2021-03-15.

[dev20] developers. Cómo ejecutar apps en android emulator. https://developer.android.com/studio/run/emulator?hl=es-419, 2020. Accesed: 2021-05-11.

[Div] Victor Diví. ¿qué es el lenguaje de programación dart? https://inlab.fib.upc.edu/es/blog/que-es-el-lenguaje-de-programacion-dart. Accesed: 2021-05-11.

[dSySL] Osalan Instituto Vasco de Seguridad y Salud Laborales. Qué es la prevención de riesgos laborales. https://www.osalan.euskadi.eus/a-quien-nos-dirigimos/-/que-es-la-prevencion-de-riesgos-laborales/. Accesed: 2021-02-11.

[Ebe21] Ebergementwebs. Las ventajas y desventajas de flutter para el desarrollo de aplicaciones. https://www.hebergementwebs.com/nuevo/las-ventajas-y-desventajas-de-flutter-para-el-desarrollo-de-aplicaciones, 2021. Accesed: 2021-05-09.

[EBF19] EBF. Ventajas y desventajas de las metodologías agile (ágiles). https://ebf.com.es/blog/ventajas-y-desventajas-de-las-metodologias-agiles-y-su-aplicacion-en-el-t
2019. Accesed: 2021-02-19.

[Esc20] Javier Escacena. Flutter visto con gafas de programador web. https://www.paradigmadigital.com/dev/flutter-visto-con-gafas-programador-web/, 2020. Accesed: 2021-05-09.

[Fer19] Pau Fernández. Flutter 40: Conectar a firebase. https://www.youtube.com/watch?v=osp1WL7W9Wo&t=2295s&ab_channel=PauFern%C3%A1ndez, 2019. Accesed: 2021-02-26.

[Fer20a] Jorge Sánchez Fernández. El patrÓn bloc junto a clean architecture en flutter. https://xurxodev.com/el-patron-bloc-junto-a-clean-architecture-en-flutter/, 2020. Accesed: 2021-05-10.

[Fer20b] Jorge Sánchez Fernández. IntroducciÓn al patrÓn bloc. https://xurxodev.com/introduccion-al-patron-bloc/, 2020. Accesed: 2021-05-10.

[Fir20] Firebase. Getting started with firebase on flutter - firecasts. https://www.youtube.com/watch?v=EXp0gq9kGxI&t=696s&ab_channel=Firebase, 2020. Accesed: 2021-02-25.

[Flua] Flutter. Adding interactivity to your flutter app. https://flutter.dev/docs/development/ui/interactive. Accesed: 2021-03-08.

[Flub] Flutter. Duration class. https://api.flutter.dev/flutter/dart-core/Duration-class.html. Accesed: 2021-04-20.

[Fluc] Flutter. Flutter. https://flutter.dev/. Accesed: 2021-05-09.

[Flud] Flutter. An introduction to integration testing. https://flutter.dev/docs/cookbook/testing/integration/introduction. Accesed: 2021-06-04.

[Flue] Flutter. Probando apps en flutter. https://esflutter.dev/docs/testing. Accesed: 2021-06-15.

[Fluf] Flutter. scroll method. https://api.flutter.dev/flutter/flutter_driver/FlutterDriver/scroll.html. Accesed: 2021-03-27.

[Flug] FlutterFire. Cloud firestore. https://firebase.flutter.dev/docs/firestore/usage/. Accesed: 2021-02-25.

[Fon] Google Fonts. https://fonts.google.com/specimen/Recursive?preview.text_type=custom. Accesed: 2021-03-02.

[Gar] Guillermo García. Stateful y stateless widgets. https://flutter.dev/docs/development/ui/interactive. Accesed: 2021-03-08.

[HAG20] ASÍ LO HAGO. Master flutter dart de cero a experto - flutter primeros pasos - 1 primera aplicación. https://www.youtube.com/watch?v=wet3Jbr3ky0&list=PLWi0C_awWivGdhrDC-ZitZljRTOkgK9ST&index=38&ab_channel=AS%C3%8DLOHAGO, 2020. Accesed: 2021-03-01.

[Her20] Luis Herazo. ¿qué es una aplicación móvil? https://anincubator.com/que-es-una-aplicacion-movil/, 2020. Accesed: 2021-03-14.

[ION20a] IONOS. Dart de google: Una introducción al lenguaje dart. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/lenguaje-de-programacion-dart-de-google/, 2020. Accesed: 2021-05-11.

[ION20b] IONOS. Flutter: introducción al framework multiplataforma. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-flutter/, 2020. Accesed: 2021-05-09.

[ISO15] ISOTools. Riesgo laboral: definición y conceptos básicos. https://www.isotools.org/2015/09/10/riesgo-laboral-definicion-y-conceptos-basicos/, 2015. Accesed: 2021-02-10.

[Jua20] JuanyBabas. Corriendo flutter en visual studio code, para principiantes, bien explicado. https://www.youtube.com/watch?v=Qkr3sRJXbls&ab_channel=JuanyBabas, 2020. Accesed: 2021-02-25.

[Lie18] Zell Liew. Managing your git branches with git flow. https://zellwk.com/blog/git-flow/, 2018. Accesed: 2021-05-12.

[Loh19] Damodar Lohani. flutter_auth. https://github.com/lohanidamodar/flutter_auth/blob/master/test/ui/pages/login_test.dart, 2019. Accesed: 2021-03-20.

[Lui20] Pramish Luitel. Comes let us write some tests for our flutter application. https://medium.com/@pramish.luitel/flutter-testing-68dafdfd841e, 2020. Accesed: 2021-06-04.

[ma01a] manifiesto agil. Manifiesto por el desarrollo Ágil de software. https://agilemanifesto.org/iso/es/manifesto.html, 2001. Accesed: 2021-02-19.

[ma01b] manifiesto agil. Principios del manifiesto Ágil. https://agilemanifesto.org/iso/es/principles.html, 2001. Accesed: 2021-02-19.

[Mar19] Ricardo Markiewicz. flutter_infinite_list. https://github.com/Gazer/flutter_infinite_list/blob/master/lib/main.dart, 2019. Accesed: 2021-04-13.

[mat] Flutter material. Icons class. https://api.flutter.dev/flutter/material/Icons-class.html. Accesed: 2021-03-06.

[Mon21] Noé Montes. Patrón provider en flutter. https://noemontes.dev/?p=234, 2021. Accesed: 2021-04-05.

[Mor20] Pablo Lanza Moreno. Patrón provider en flutter. https://noemontes.dev/?p=234, 2020. Accesed: 2021-04-05.

[Nin19] The Net Ninja. Flutter firebase app tutorial 17 - firestore user records. https://www.youtube.com/watch?v=EA7973HI93E&ab_channel=TheNetNinja, 2019. Accesed: 2021-03-14.

[Pag19] Alex Pagoada. Curso de dart and flutter con firebase interfaz registros de usuarios. https://www.youtube.com/watch?v=PpSI9M3lv5o&ab_channel=AlexPagoada, 2019. Accesed: 2021-02-25.

[pro20] profesxorgeek. ¿qué es xamarin? https://docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin, 2020. Accesed: 2021-03-15.

[pub20] pub.dev. address_search_text_field 1.3.5+1. https://pub.dev/packages/address_search_text_field, 2020. Accesed: 2021-05-27.

[R20] Girish R. Top technologies used to develop mobile app. https://www.fingent.com/blog/top-technologies-used-to-develop-mobile-app/, 2020. Accesed: 2021-03-14.

[Roj21] Felipe Rojas. Gitflow: ¿qué és y cómo aplicarlo sin morir en el intento? https://gfourmis.co/gitflow-sin-morir-en-el-intento/, 2021. Accesed: 2021-05-12.

[ROS21] ROSEPAC. Los 14 mejores framework de desarrollo de aplicaciones multiplataforma (web y móvil) 2021. https://ciberninjas.com/mejores-sdk-multiplataforma-2019-20/#2-ionic, 2021. Accesed: 2021-03-15.

[Saf19] SafetYA. Gtc 45 y el nivel de deficiencia en la evaluación de riesgos. https://safetya.co/gtc-45-y-el-nivel-de-deficiencia/#:~:text=Definici%C3%B3n%20del%20nivel%20de%20deficiencia&text=%E2%80%9CNivel%20de%20deficiencia%20(ND),en%20un%20lugar%20de%20trabajo%E2%80%9D., 2019. Accesed: 2021-02-12.

[Sel20] Chris Sells. Updates on flutter testing. https://medium.com/flutter/updates-on-flutter-testing-f54aa9f74c7e, 2020. Accesed: 2021-06-12.

[Sir17] Sandamal Siripathi. Lenguajes de desarrollo para móvil. https://code.tutsplus.com/es/articles/mobile-development-languages--cms-29138, 2017. Accesed: 2021-03-15.

[Sof] Softcorp. Definición y cómo funcionan las aplicaciones móviles. https://servisoftcorp.com/definicion-y-como-funcionan-las-aplicaciones-moviles/#Que_es_una_aplicacion_movil. Accesed: 2021-03-14.

[Stu19] RetroPortal Studio. Flutter - creating pop-up dialog in flutter! (beginners). https://www.youtube.com/watch?v=FGfhnS6skMQ&ab_channel=RetroPortalStudio, 2019. Accesed: 2021-03-12.

[Vet] Giuseppe Vetri. Que es un unit test. https://www.codingpizza.com/que-es-un-unit-test/. Accesed: 2021-06-15.

[Way20] The Flutter Way. Welcome, login, signup page - flutter ui - speed code. https://www.youtube.com/watch?v=ExKYjqgswJg&ab_channel=TheFlutterWay, 2020. Accesed: 2021-03-02.

[wha19a] whatsupcoders. Flutter tutorial - flutter charts. https://www.youtube.com/watch?v=4gkt5qDBq4w&ab_channel=whatsupcoders, 2019. Accesed: 2021-03-14.

[wha19b] whatsupcoders. Flutter tutorial - upload images using firebase storage. https://www.youtube.com/watch?v=7uqmY6le4xk&ab_channel=whatsupcoders, 2019. Accesed: 2021-03-26.

[Wik] Wikipedia. Flutter (software). https://es.wikipedia.org/wiki/Flutter_(software). Accesed: 2021-05-09.

[Wik19] Wikipedia. Iso 45001. https://es.wikipedia.org/wiki/ISO_45001, 2019. Accesed: 2021-02-11.

[Wik21a] Wikipedia. Git. https://es.wikipedia.org/wiki/Git, 2021. Accesed: 2021-05-12.

[Wik21b] Wikipedia. Latex. https://es.wikipedia.org/wiki/LaTeX, 2021. Accesed: 2021-05-13.

[Wik21c] Wikipedia. Nativescript. https://es.wikipedia.org/wiki/NativeScript, 2021. Accesed: 2021-03-15.

[Wik21d] Wikipedia. Visual studio code. https://es.wikipedia.org/wiki/Visual_Studio_Code, 2021. Accesed: 2021-05-11.

[Yog18] Raja Yogan. Flutter - reading data from firestore. https://www.youtube.com/watch?v=2KknXzalXfg&ab_channel=RajaYogan, 2018. Accesed: 2021-02-26.

[yXC] Franco Rosa y Xiara Cavallero. Tipos de sistemas operativos móviles. https://sites.google.com/site/sistemasoperativosfranco9/tipos-de-sistemas-operativos-moviles. Accesed: 2021-03-16.

[Zar20] José Luis Zarabanda. Flutter | cambiar fuente al texto. https://www.youtube.com/watch?v=vtzAEWTWfgU&ab_channel=Jos%C3%A9LuisZarabanda, 2020. Accesed: 2021-03-02.