MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MÁSTER

DISEÑO & BACKEND DE UNA APP CONCEBIDA PARA SIMPLIFICAR LA GESTIÓN DE RIESGOS DE EMPRESAS EN LA COMUNICACIÓN DE SINIESTROS A SU COMPAÑÍA ASEGURADORA.

Autor: Miguel Enrile Fernández de Arévalo

Director: Pablo Collado Puerta

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Diseño & backend de una app concebida para simplificar la gestión de riesgos de empresas en la comunicación de siniestros a su compañía aseguradora en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2020/21 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

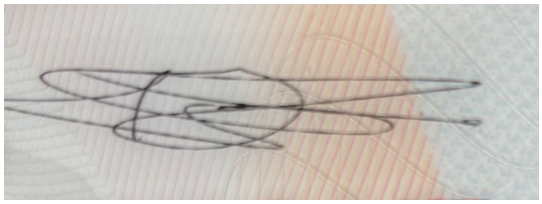
tomada de otros documentos está debidamente referenciada.

Fdo.: Miguel Enrile Fernández de Arévalo Fecha: 17/ 05/ 2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo: Pablo Collado Puerta Fecha: 17/ 06/ 2021



AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor Ilmo.D. Miguel Enrile Fernández de Arévalo

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: Diseño & backend de una app concebida para simplificar la gestión de riesgos de empresas en la comunicación de siniestros a su compañía aseguradora, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 17 de Junio de 2021

ACEPTA

Fdo MIGUEL ENRILE FERNÁNDEZ DE ARÉVALO

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIONES

TRABAJO FIN DE MASTER

DISEÑO & BACKEND DE UNA APP CONCEBIDA PARA SIMPLIFICAR LA GESTIÓN DE RIESGOS DE EMPRESAS EN LA COMUNICACIÓN DE SINIESTROS A SU COMPAÑÍA ASEGURADORA.

Autor: Miguel Enrile Fernández de Arévalo

Director: Pablo Collado Puerta

Madrid

Agradecimientos

A mis padres y a mi familia por el apoyo y la confianza a lo largo de estos años.

DISEÑO & BACKEND DE UNA APP CONCEBIDA PARA SIMPLIFICAR LA GESTIÓN DE RIESGOS DE EMPRESAS EN LA COMUNICACIÓN DE SINIESTROS A SU COMPAÑÍA ASEGURADORA.

Autor: Enrile Fernández de Arévalo, Miguel
Director: Collado Puerta, Pablo
Entidad Colaboradora: Insurance Manager SL (IMeureka)

RESUMEN DEL PROYECTO

Desarrollo de una aplicación móvil para los sistemas operativos más comunes con el objetivo de realizar una adaptación digital al proceso de la gestión de riesgos y la declaración de siniestros de los clientes de la correduría de seguros IMeureka. A fecha de escritura de este ensayo, la aplicación se puede encontrar en las tiendas de aplicaciones de Android y Apple bajo el nombre de IMeureka – Clientes.

Palabras clave: Transformación digital, insurtech, IMeureka, gestión de riesgos.

1. Introducción

La empresa Insurance Manager SL [1], ubicada en Madrid, es la propietaria de la plataforma de gestión de riesgos www.imeureka.com. Sus objetivos son dos, por un lado habilitar un entorno transparente de libre comercio, abierto a la contraoferta, en el contexto de la intermediación de pólizas de seguros, para ello, proporcionan su plataforma de gestión de riesgos e intermediación. Por otro lado, facilitar la comunicación entre la compañía aseguradora y la empresa.

La aplicación desarrollada permite al usuario declarar siniestros de una manera sencilla e interactiva, de manera que para el proceso de la declaración sea ágil y rápido, con una menor necesidad de intermediación por parte del corredor, beneficioso para todos los integrantes de la cadena de valor (cliente – correduría - aseguradora)

2. Definición del proyecto

2.1 La aplicación será desarrollada en un framework multiplataforma, concretamente Flutter, para facilitar que ésta pueda ser utilizada por el máximo número de usuarios.

2.2 Debe ser una aplicación *Friendly* y fácil de utilizar

2.3 La aplicación deberá estar completamente integrada en la plataforma de IMeureka a través de API suficientemente generales para que puedan ser usado por otros proyectos que requieran información de la DB.

2.4 La aplicación debe ser uniforme independientemente del tipo de siniestro a declarar, compañía aseguradora … Actualmente cada compañía dispone de su método de declaración de siniestros, se va a unificar ese proceso.

2.4 Las funcionalidades básicas serán:

2.5.1 Declaración de siniestros: Es el objetivo principal alrededor del cual se engloba la aplicación. Debe ser un proceso intuitivo y que recoja:

- Localización del siniestro

- Archivos del siniestro (Imágenes, vídeo, docs)

- Título, Descripción, Fecha

2.5.2 Almacenamiento de siniestros en la base de datos de IMeureka

2.5.3 Visualización de los siniestros declarados: Se podrán ver todos los detalles de un siniestro, tanto si ha sido declarado a través de la app cómo si no.

2.5.4 Visualización de las empresas/pólizas de un usuario (en caso de que el usuario tenga más de una)

3. Descripción del modelo/sistema/herramienta

La aplicación tiene una pantalla de bienvenida, desde la cual el usuario se puede autenticar. Desde la aplicación un usuario no podrá registrarse, ya que previamente debe tener cuenta en la plataforma principal de IMeureka.

Tras acceder a la aplicación, se podrá directamente navegar por el sistema descrito en el apartado anterior. La siguiente ilustración muestra el flujo entre pantallas principales de la aplicación, no limitado a estas mismas el resultado final.

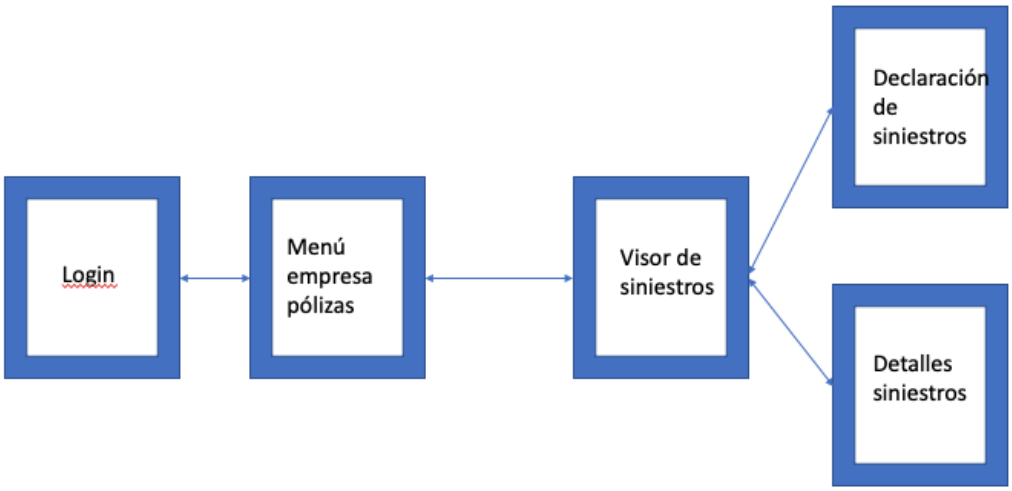
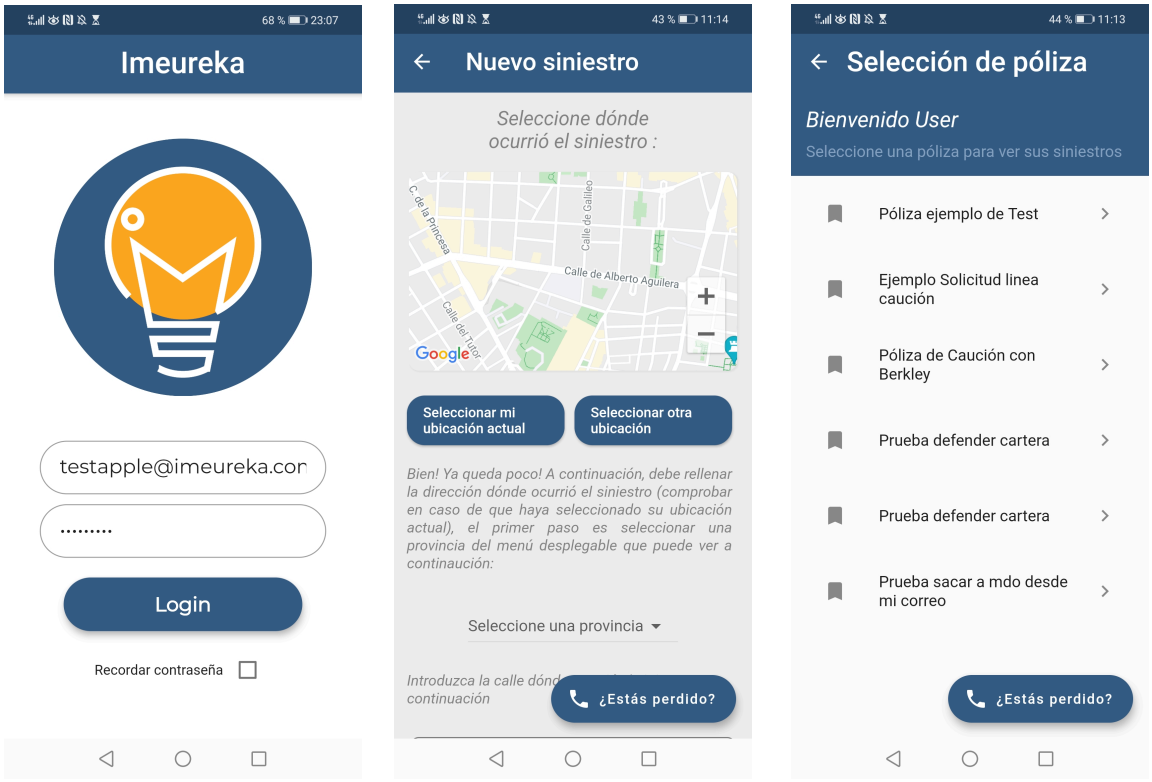


Ilustración 1 Diagrama de flujo de la aplicación

4. Resultados

Se ha conseguido el comportamiento deseado para la aplicación en ambos sistemas operativos. Se ha subido a los marketplaces de Ios y Android y se han cumplido los objetivos especificados en la sección objetivos.



Ilustraciones 2,3,4 – Login, Ubicación del siniestro, Selección de póliza

A través de la aplicación el usuario es capaz de realizar los servicios que son requeridos por la misma de manera sencilla e interactiva. La aplicación está integrada con la BD de IMeureka. Cuenta con un botón de ¿Estás perdido? Que facilita la usabilidad de la misma.

5. Conclusiones

Con esta aplicación se consigue aportar valor tanto para el usuario de la aplicación (que tiene un único lugar, claro y ágil dónde declarar todos sus siniestros, independientemente de compañía aseguradora, póliza, tipo de siniestro…) como para la correduría de seguros, que ahorrará una buena parte del costo de trámite de los siniestros, ya que los usuarios sabrán directamente declarar siniestros correctamente.

6. Referencias

[1] IMeureka, Página web y contacto. https://www.imeureka.com/

DESIGN & BACKEND OF AN APP DESIGNED TO SIMPLIFY THE RISK MANAGEMENT OF COMPANIES IN THE COMMUNICATION OF CLAIMS TO YOUR INSURANCE COMPANY.

Author: Miguel Enrile Fernández de Arévalo
Supervisor: Pablo Collado Puerta
Collaborating Entity: Imeureka

ABSTRACT

Development of a mobile application for the most common operating systems with the aim of making a digital adaptation to the risk management process and the declaration of claims of the clients of the IMeureka insurance brokerage. As of the writing of this essay, the app can be found in the Android and Apple app stores under the name IMeureka - Clients.

Keywords: Digital transformation, insurtech, IMeureka, risk management.

1. Introduction

The company Insurance Manager SL [1], located in Madrid, is the owner of the risk management platform www.imeureka.com. Its objectives are twofold, on the one hand to enable a transparent environment of free trade, open to counter offer, in the context of the intermediation of insurance policies, for this, they provide their risk management and intermediation platform. On the other hand, facilitate communication between the insurance company and the company.

The developed application allows the user to declare claims in a simple and interactive way, so that the declaration process is agile and fast, with less need for intermediation by the broker, beneficial for all members of the value chain (client - brokerage - insurer)

2. Definición del proyecto

2.1 The application will be developed in a multiplatform framework, specifically Flutter, to facilitate that it can be used by the maximum number of users.

2.2 It should be a Friendly application and easy to use

2.3 The application must be fully integrated into the IMeureka platform through sufficiently general APIs so that they can be used by other projects that require information from the DB.

2.4 The application must be uniform regardless of the type of claim to be declared, insurance company ... Currently each company has its own claim declaration method, this process will be unified.

2.4 The basic functionalities will be:

2.5.1 Claim declaration: It is the main objective around which the application is encompassed. It should be an intuitive process and that collects:

- Location of the claim

- Loss files (Images, video, docs)

- Title, Description, Date

2.5.2 Saving claims in the IMeureka database

2.5.3 Viewing declared claims: You will be able to see all the details of a claim, whether it has been declared through the app or not.

2.5.4 Viewing the companies / policies of a user (in case the user has more than one)

3. Description od the system

The application has a welcome screen, from which the user can authenticate. From the application, a user will not be able to register, since they must previously have an account on the main IMeureka platform.

After accessing the application, you can directly navigate through the system described in the previous section. The following illustration shows the flow between main screens of the application, the final result is not limited to these.

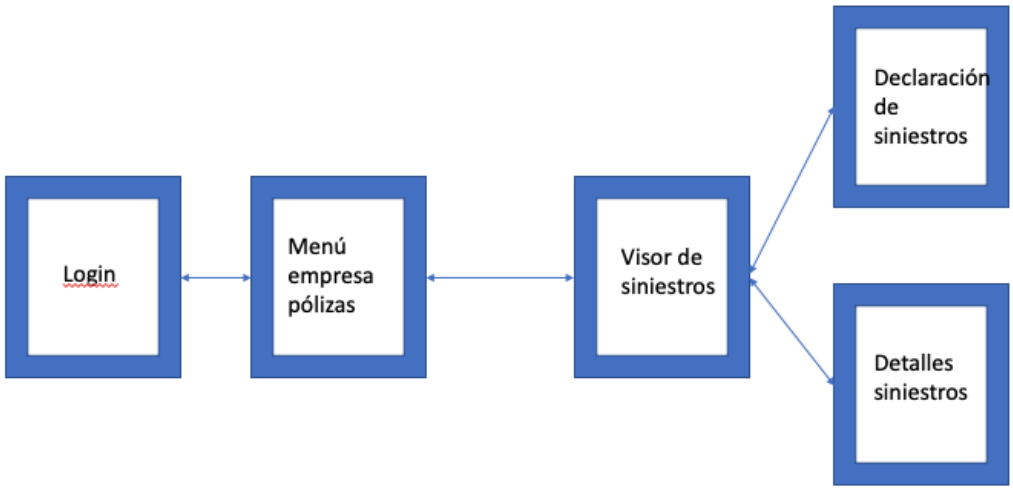
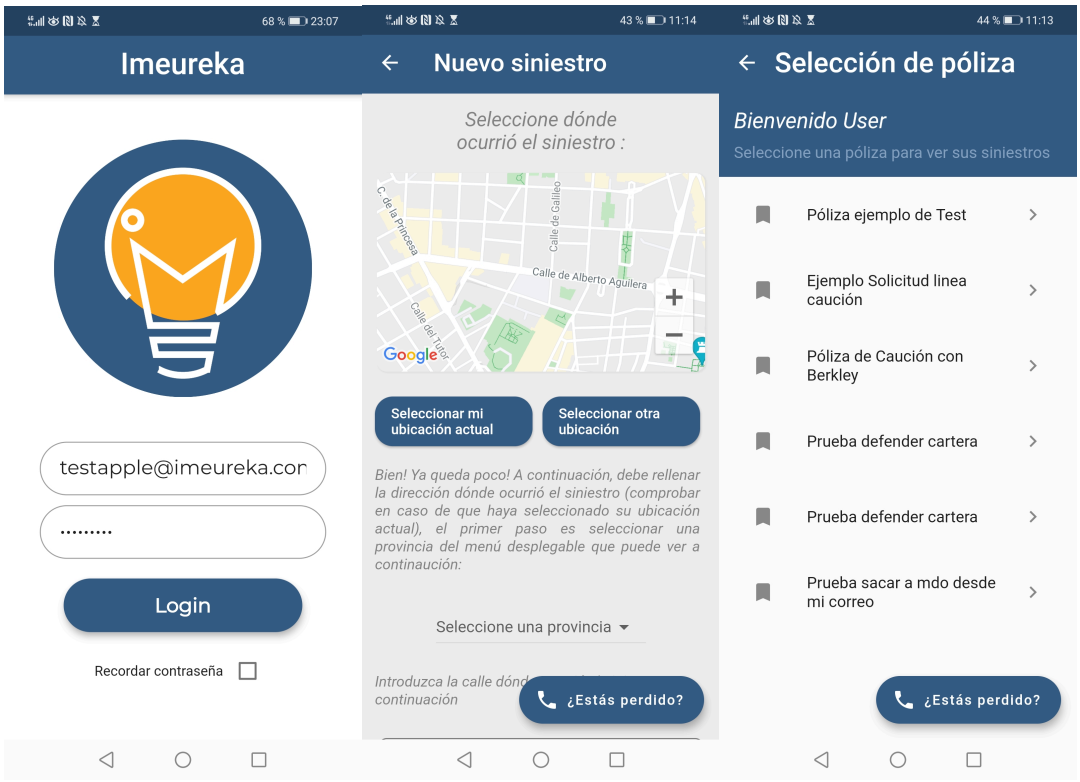


Ilustración 3 Diagrama de flujo de la aplicación

4. Results

The desired behavior for the application has been achieved on both operating systems. Here are some screens of it.



Illustrations 2,3,4 - Login, Loss location, Policy selection

Through the application, the user is able to perform the services that are required by it in a simple and interactive way. The application is integrated with the IMeureka database. It has a button for Are you lost? That facilitates the usability of it.

5. Conclusions

With this application it is possible to add value both for the user of the application (which has a single, clear and agile place where to declare all its claims, regardless of insurance company, policy, type of claim ...) and for the insurance brokerage, which You will save a good part of the cost of processing claims, since users will know how to directly declare claims correctly.

6. Referencias

[1] IMeureka, Página web y contacto. https://www.imeureka.com/

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

La empresa Insurance Manager SL, ubicada en Madrid, es la propietaria de la plataforma de gestión de riesgos www.imeureka.com. Sus objetivos son dos, por un lado, habilitar un entorno transparente de libre comercio, abierto a la contraoferta, en el contexto de la intermediación de pólizas de seguros. Para ello, proporcionan su plataforma de gestión de riesgos e intermediación. Por otro lado, facilitar la comunicación entre la compañía aseguradora y la empresa.

Ilustración 4: Logo de IMeureka

IMeureka es una correduría de seguros centrada en la transformación digital, gracias a la tecnología, están colaborando en la evolución del sector de los seguros hacia modelos más transparentes y eficientes. Ofrecen servicios como la verificación de riesgos, la gestión de siniestros, Asesoría jurídica, Ciberseguridad ...

El sector asegurador en España tiene un tamaño bastante considerable (cerca de 60.000 millones de euros anuales), disfruta de un crecimiento constante y sostenible y se ha mantenido extremadamente resiliente a través de los ciclos económicos. Sin embargo, el sector ha demostrado ser difícil de penetrar tecnológicamente para una variedad de razones (que explicaremos en detalle en este documento) y ahora está listo para tecnología disruptiva

para traer las eficiencias muy necesarias y traer de vuelta el centro de atención el principal activo de la industria: los datos.

Esta combinación de elementos (mercado grande y segmentado, crecimiento constante y rentable, altos márgenes, bajas barreras de entrada, gran estructura heredada de grandes jugadores, etc.) presenta una oportunidad única para que surja un jugador de Marketplace en línea, y nos trae a la declaración de la visión de IMeureka: "Convertirnos en el Marketplace de referencia en el sector asegurador. Tecnología, transparencia, eficiencia y calidad de servicio, serán la base de cualquier relación de confianza en los negocios." [1]

Las deficiencias existentes en la intermediación de pólizas a menudo encarecen el precio de las primas de seguro manera innecesaria. Además, el bróker, debido a sus intereses económicos y facilitado por la opacidad del proceso, tiende a cerrar la operación con la compañía aseguradora que le genera mayor ingreso. Esto es malo para el cliente, ya que encarece la prima que paga, pero es tremendamente negativo para el mercado asegurador, que pierde el tiempo cotizando riesgos que nunca va a ganar ya que el bróker no lo va a permitir. La comisión del bróker es, en algunos casos de malas prácticas, arbitraria, orientado a conseguir "el máximo beneficio sin perder el cliente". Algunas aseguradoras incluso permiten encarecer de manera artificial las pólizas; al terminar una cotización, la aseguradora le indica al bróker un precio mínimo, y éste puede subirlo a lo que quiera para aumentar su comisión. Por otro lado, existen muchas pólizas que se van renovando año a año sin volverlas a cotizar. En muchos casos estas pólizas antiguas tienen sobreprecio debido a que las condiciones de los riesgos hace años no eran las mismas, ni la competitividad de cotización entre aseguradoras era tan agresiva como lo es actualmente.

Para solucionar esto, se creó el Marketplace de IMeureka, en el cual los usuarios (gestores de riesgos de las empresas y clientes particulares) pueden gestionar su programa de seguros (acceder a la información de sus riesgos digitalizada, comunicar siniestros y contratar pólizas, pedir cotización pólizas existentes) y las compañías aseguradoras pueden cotizar las pólizas que quieran (lanzar ofertas) sin que el bróker se interponga por intereses económicos.

Recientemente, la empresa ha abierto otra línea de negocio, y es que todas las ventajas competitivas que estaba adquiriendo a través de la innovación en el sector las pondrá a disposición de las correduríias actuales existentes del mercado español. De esta manera IMeureka abre otro frente, aparte de la correduría de seguros orientada simultáneamente al liderazgo en costes y a la diferenciación, inicia un nuevo modelo de negocio con el objetivo de transformarse en el "idealista de los seguros". La empresa pondrá a disposición de las correduríias actuales las herramientas que vayan desarrollando, de tal manera que éstas no la vean como la competencia si no como todo lo contrario, un apoyo en su negocio.

Paralelamente al trabajo en este ensayo, la empresa está trabajando en temas de NLP, reconocimiento de imágenes, extracción de datos de fuentes externas, lectura e interpretación automática de PDF … Todo ello orientado a aportar valor a todos los clientes de la empresa, tanto clientes de la correduría como los clientes del Marketplace explicado anteriormente.

Es importante resaltar la diferencia entre IMeureka y un simple comparador de seguros, en palabras de IMeureka "No somos un comparador. Ni mucho menos. Los comparadores son herramientas que ofrecen tarifas pre acordadas con distintas compañías. Precios fijos. IMeureka es una herramienta de gestión de riesgos, un Marketplace donde las compañías tendrán acceso a tu información y podrán ir mejorando sus ofertas para tu beneficio. Precio dinámico." [2]

En resumen, la misión de IMeureka es generar valor para los clientes [2] proporcionando las herramientas indicadas para ello.

La correduría de seguros ha cerrado ya acuerdos con decenas de aseguradoras, confirmando así el interés del nicho de mercado que quieren explotar. Algunas referentes del sector como Mapfre, Generali, Plus Ultra, Axa, Helvetia , Cáser Seguros… La idea es que, al sacar un riesgo al Marketplace, sean todas las aseguradoras las que compitan (tanto en precio como en coberturas ofrecidas) entre ellas, en un entorno anónimo entre ellas para garantizar la transparencia.

En el contexto previamente explicado surge la idea de la aplicación en cuestión, no existe hoy en día una manera uniforme de declara los siniestros, más bien, cada caso "es un mundo". Hay compañías aseguradoras a las que hay que llamar por teléfono, otras declararlo a través de su aplicación, otras enviar un email…

Por otro lado, en la parte de correduría de seguros, los corredores no hacen todo el esfuerzo posible en la declaración del siniestro. Lógicamente, un corredor intenta que sus pólizas no tengan siniestros, ya que de esa manera tendrá menos trabajo. Un siniestro supone para el corredor un gasto de tiempo y recursos del que no va a obtener beneficio directo, únicamente le añade carga de trabajo de gestión.

El proceso tradicional para la realización de una comunicación de siniestro desde el usuario hasta la aseguradora es el siguiente (en caso de que la póliza tenga un corredor como intermediador). Primero, ocurre el siniestro. El usuario se pone en contacto con su bróker de seguros (en todos los pasos el método es mediante email o teléfono). El bróker informa de la situación a la compañía de seguros. La compañía de seguros asigna un tramitador que se encargue de este siniestro. El tramitador contacta con el bróker y le pide los datos necesarios. El bróker le reenvía esta petición al usuario. El usuario responde al bróker...

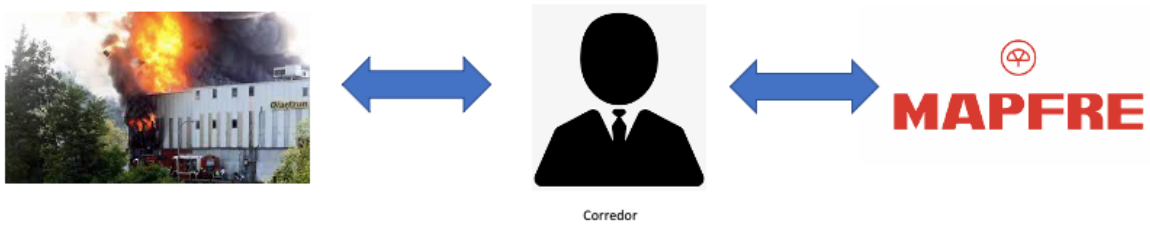


Diagrama 2: Declaración de siniestros

Este proceso es así para cada comunicación, en cada siniestro tienen lugar varias comunicaciones de ida y vuelta sin ningún valor añadido, únicamente agregando carga de trabajo al bróker. Este proceso tan ineficiente causa que, en la actualidad, un siniestro tarda en resolverse, de media, unos 50 días. En este tiempo, se habrán enviado entre 20 y 30 emails a través del intermediario. En un futuro, IMeureka plantea un tiempo máximo de resolución de siniestros de 15 días.

La atención tan personalizada que brinda el bróker a la hora de la contratación de la póliza se termina en el momento en que hay un siniestro, en muchos casos, el bróker ofrece un formulario tal que así al asegurado.

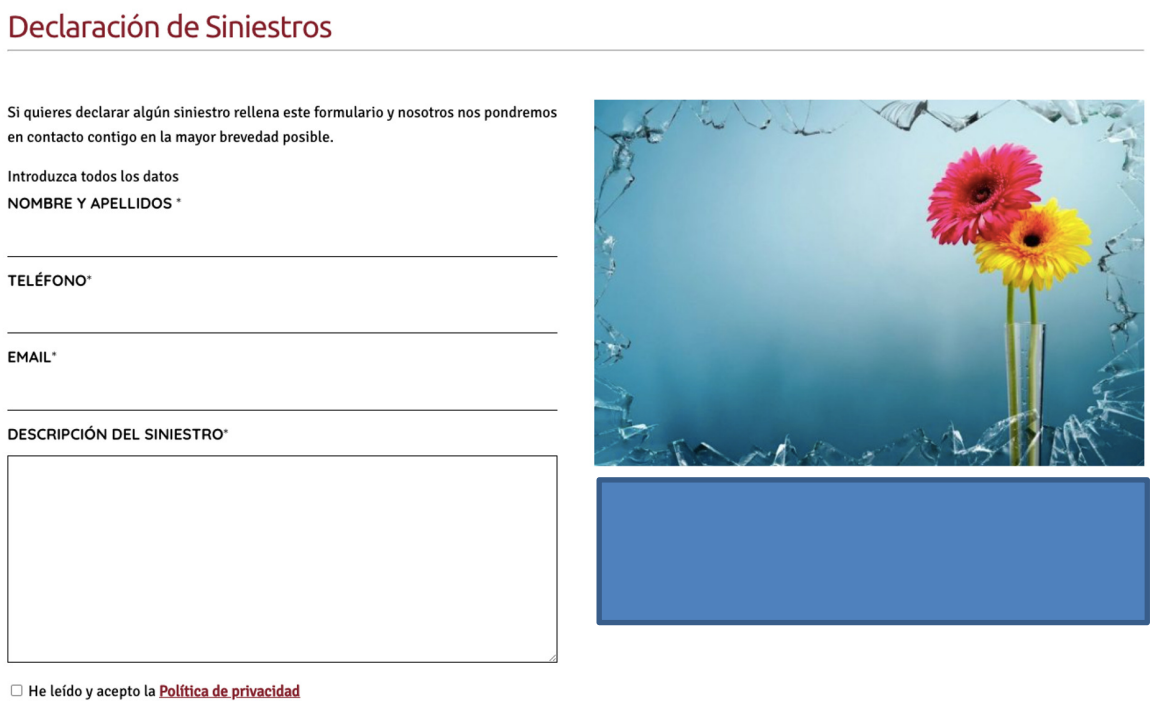


Ilustración 3: Declaración de siniestros a través de corredor

Que no contiene todos los datos necesarios para declarar un siniestro, le pide datos que el bróker ya conoce (ya que el bróker conoce a sus clientes, sabe su email y teléfono). Este formulario inevitablemente dará paso al proceso descrito anteriormente.

De aquí surge la necesidad de la agilización de este proceso, mediante una aplicación para declarar siniestros, las comunicaciones entre usuario y aseguradora podrían realizarse sin apenas intermediación por parte del bróker. Mediante la interfaz adecuada, el usuario le entregará a la aseguradora directamente los datos que esta necesita, y la aseguradora podrá tramitar el siniestro en menos tiempo, resultando en un beneficio para todas las partes implicadas.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Para el desarrollo de la aplicación se han usado cuatro tecnologías principalmente.

Android Studio & Flutter SDK [3] – Entorno de desarrollo más usado para la creación de aplicaciones, incluye diversas funcionalidades como el manejo automático de maquinas virtuales, la ejecución sencilla de las aplicaciones creadas sobre terminales físico, previsualización de diseños de pantallas, etc. Además, cuenta con un plugin para flutter. Al usar Android Studio sobre un ordenador Mac permite ejecutar tanto máquinas virtuales Android como Ios, de manera que se programa una vez y se ejecuta en ambos sistemas operativos.

Postman [4] – Herramienta para simplificar el desarrollo e implementación de Apis. Con esta herramienta se pueden realizar peticiones http/https de todo tipo, fue usada el desarrollo para depurar la API y la conexión con el backend.

GPS [5] – "Global Positioning System" es un sistema de navegación por radio que permite a los usuarios conocer su posición y su velocidad. En la aplicación fue usada para la agilización del proceso de declaración de siniestros.

JSON [6] – Es un formato ligero de intercambio de datos. En la aplicación fue usado para la comunicación entre la BD y la aplicación.

Git & Bitbucket [7 , 8] – Git es un software de control de versiones. En la aplicación fue utilizada para tener versiones del código externas a la propia máquina en la que se desarrollaba. Bitbucket permite integrarse con Git para almacenar el código en un repositorio en línea

PHP Storm [9] – IDE para la programación de la API en PHP, agiliza el desarrollo.

XAMPP [10] – "XAMPP es un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para

lenguajes de script PHP y Perl. El nombre es en realidad un acrónimo: X, Apache, MariaDB/MySQL, PHP, Perl." XAMPP fue usada para la gestión de bases de datos MYSQL en IMeureka, y la ejecución del código en localhost.

Capítulo 3. ESTADO DE LA CUESTIÓN

La solución planteada para el problema de la declaración de los siniestros a través de una app móvil. Esta idea es revolucionaria en el sector, por lo que no se puede comparar con otras empresas. Actualmente, las aseguradoras tienen números de atención al cliente gratuitos, emails... A través de los cuales gestionan los siniestros. Hay algunas aseguradoras que cuentan con app para la consulta de pólizas y la declaración de siniestros, pero si un cliente tiene pólizas de seguro con distintas compañías, el proceso de declaración de siniestro no sería uniforme. (Tendría que bajarse una app para Mapfre, llamar a un tlf para Allianz...).

Imeureka se ha propuesto que desde una sola plataforma se puedan comunicar siniestros a cualquier compañía aseguradora desde cualquier dispositivo.

Con el objetivo en mente de facilitar la declaración de siniestros por parte de usuarios finales de empresa, se planteó el desarrollo de una app móvil que cumpliese con este objetivo. Para ello, se debe hacer un breve estudio del estado actual del mercado de los dispositivos móviles.

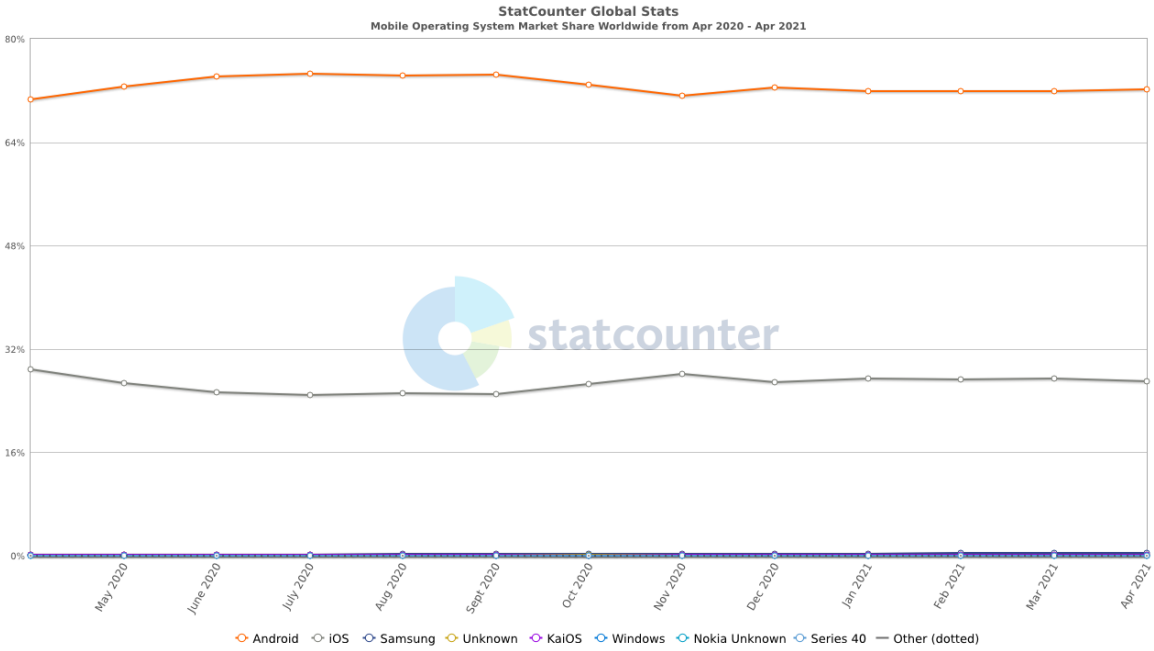


Ilustración 4: Cuotas de mercado de los principales OS móvil [11]

En la imagen anterior observamos que, a nivel mundial, el 99,5% de los dispositivos móviles usan el sistema operativo Android o IOS. Por lo tanto, lo lógico será centrarse en estos dos sistemas operativos, ya que el resto están en desuso.

La siguiente cuestión para plantear es el método de programación empleado para desarrollar la app, hasta hace poco tiempo la respuesta era clara: Una app nativa para cada OS. Esto tenía problemas debido a que un mismo desarrollador debía conocer la implementación y programación de dos plataformas y lenguajes diferentes, que no son parecidas, y requiere cada una mucho tiempo de formación. Para empresas pequeñas y startups hay otras tecnologías que permiten reducir los costos de desarrollo y los tiempos.

Para solucionar este problema surgieron frameworks como React Native o Flutter. Son soluciones multi-plataforma, se programa una vez en su código (Dart, en el caso de Flutter, JavaScript, reactNative) y al compilarse se implementa nativamente para cada sistema operativo. Los dos frameworks son extensamente usados. Flutter nace en 2017 de la mano de Google y React Native en 2015 de la mano de Facebook.

Se ha decidido usar flutter para el desarrollo front-end de la app. Estos frameworks mencionados son útiles en apps “típicas”, ya que contienen centenares de plugins para hacer prácticamente de todo sin apenas implementar funcionalidad nueva. La aplicación usa 19 de estos plugins, que han sido suficientes para toda la funcionalidad que se pretendía implementar.

Los datos introducidos a través de la app se guardarán en el back-end de la empresa, de tal manera que sean accesibles desde la plataforma por los usuarios de la plataforma web. Para ello, será necesaria la creación de una API que conecte la base de datos actual de la empresa con la app móvil.

La plataforma ya usa PHP+MYSQL con el framework de fat-free, por lo que se desarrollará una API usando este lenguaje para que se integre en los sistemas actuales de la manera más sencilla posible.

Por otro lado, se encuentra el estado de la cuestión de la declaración de siniestros. Existen dos maneras principales de declarar siniestros, directamente a la aseguradora o a través del corredor que intermedió la póliza.

En ambos casos la manera de declarar el siniestro es diferente. Las aseguradoras cuentan con aplicaciones, números de teléfono e emails, pero no está estandarizado. Esto significa que alguien que tenga seguros con varias compañías tendrá que realizar un proceso diferente en la declaración de siniestro cada vez, con el inconveniente que esto supone.

Mapfre pone a su disposición teléfonos….

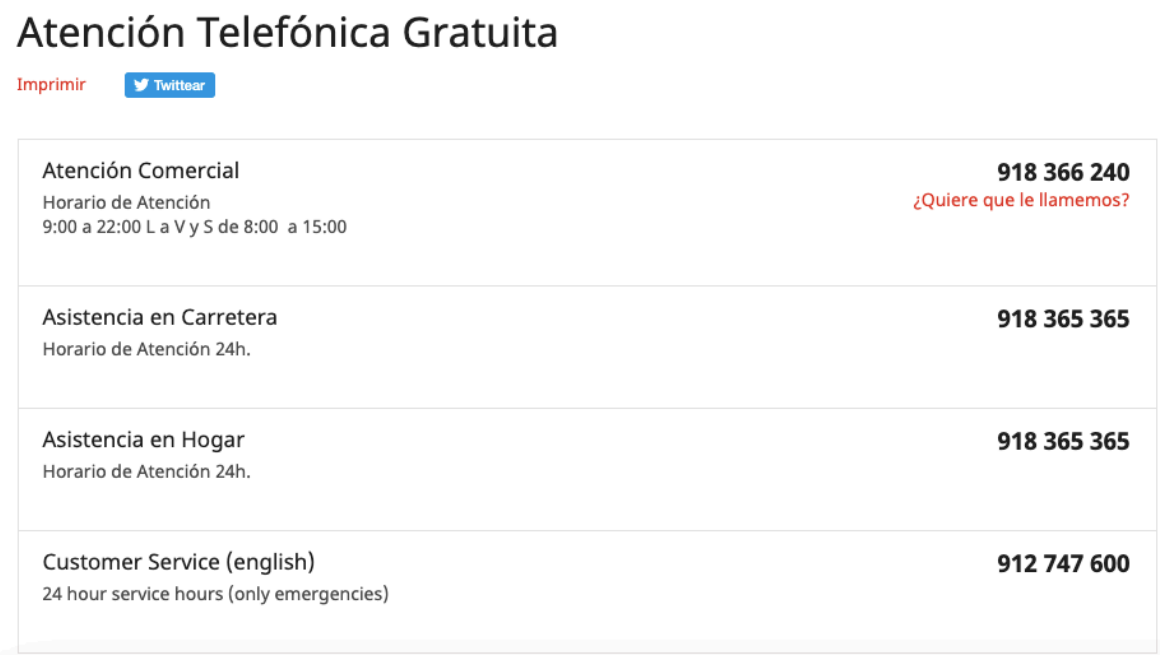


Ilustración 5: Declaración de siniestros a través de Mapfre

Allianz permite descargarse su app, declarar el siniestro online o por teléfono.

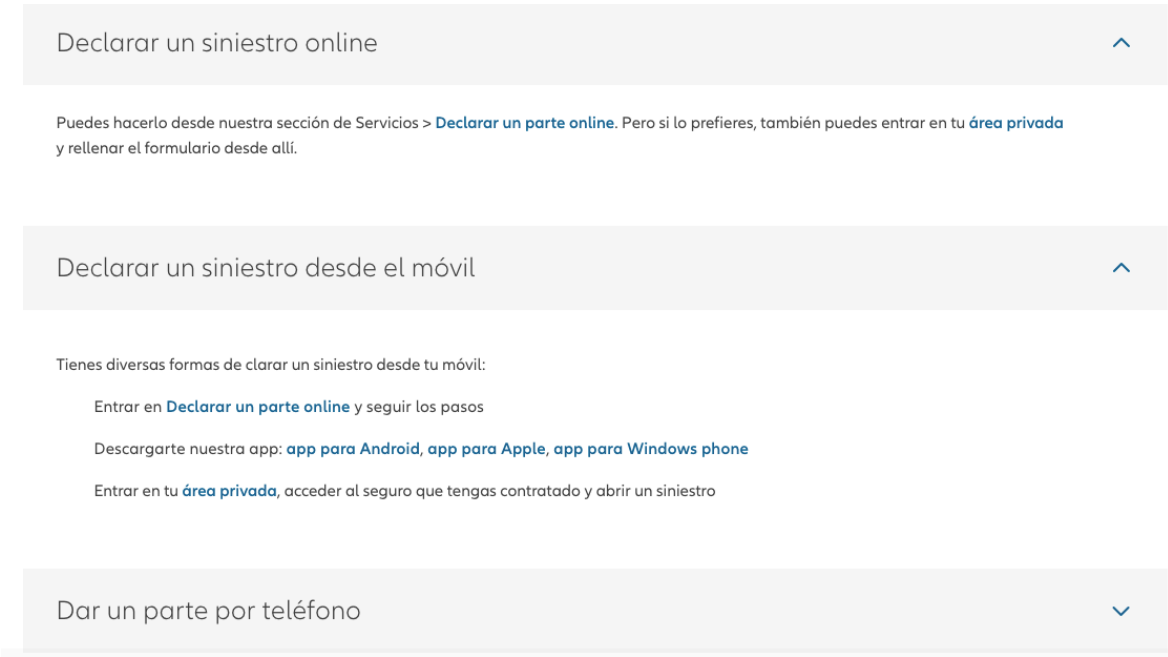


Ilustración 6: Declaración de siniestros a través de Allianz

En España hay actualmente más de 200 aseguradoras, por lo que poder unificar este proceso le ayudaría tanto a las aseguradoras como a los clientes. A las aseguradoras porque sus clientes aprenderán el proceso para todas, y les costará menos trabajo declarar el siniestro, ahorrando tiempo de gestión. A los clientes porque tendrán una app ubicua en la cuál podrán declarar sus siniestros sin preocuparse de los detalles concretos de su aseguradora.

El mercado asegurador es maduro y con alta competencia, goza de grandes economías de escala, por lo que diferenciarse es fundamental. En la siguiente imagen podemos ver el número de aseguradoras privadas en España por año:

Ilustración 7: Número de empresas de seguros vs año

El número de empresas aseguradoras lleva cayendo 10 años de manera consecutiva (salvo en 2017), por lo que podría pensarse que es un mercado que está muriendo. Nada más lejos de la realidad como demuestra la siguiente imagen:

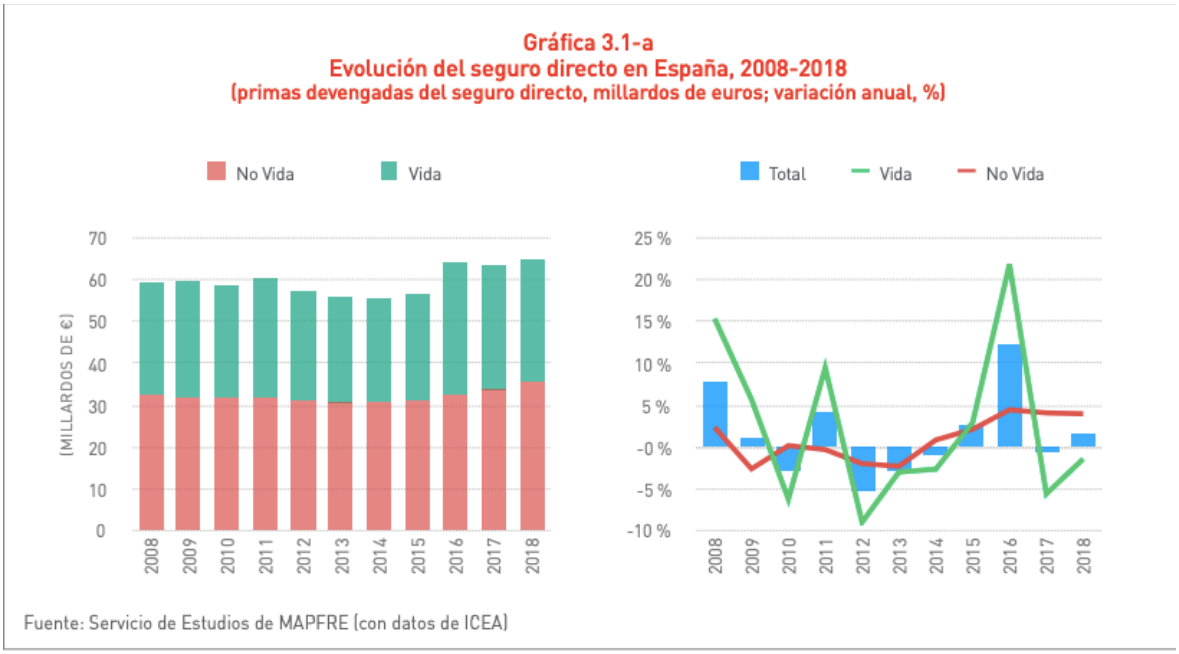


Ilustración 8: Facturación seguros españoles[13]

Podemos observar que, a pesar de que el volumen de las primas se mantiene estable e incluso crece en años recientes, la cantidad de aseguradoras disminuye. Cada vez, las aseguradoras grandes son más grandes, pueden ofrecer mejores precios y se van creando enormes barreras de entrada que expulsa a los más ineficientes o menos innovadores.

El índice de Herfindahl [14] es una medida empleada en economía que informa sobre la concentración de un mercado. Un índice alto >1000 supone que el mercado se reparte entre muy pocas empresas grandes. Un índice bajo lo contrario.

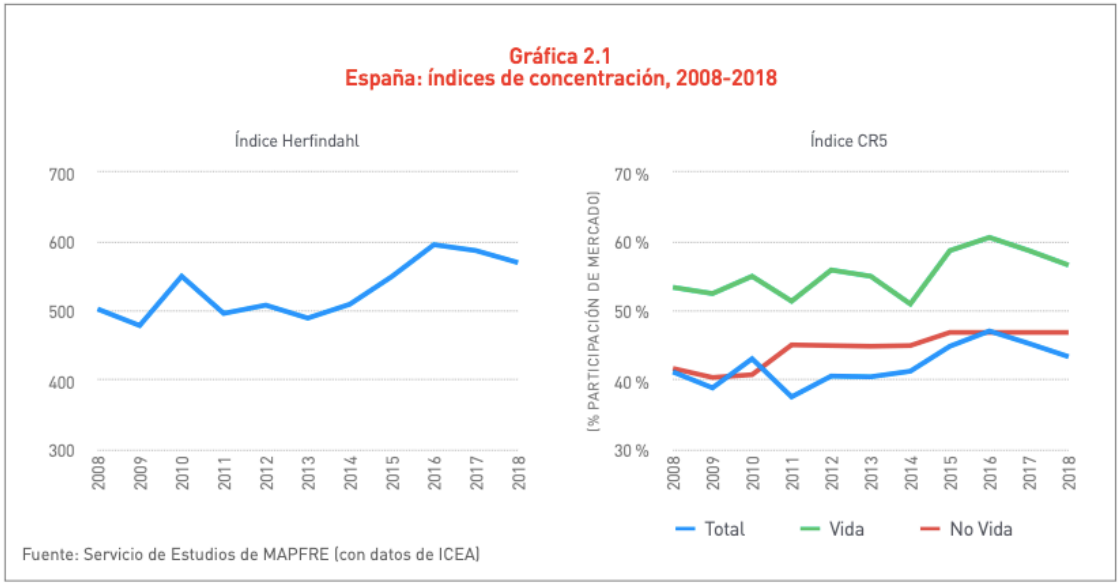


Ilustración 9: El índice de Herfindahl del mercado asegurador [13]

El índice está aumentando, y esto demuestra lo expuesto antes, que el mercado se está concentrando en cada vez menos manos.

Parece observarse una tendencia alcista en el índice de Herfindahl, lo que significa que el mercado se está volviendo cada vez menos competitivo (cuadra con lo visto de la disminución del número de empresas, cada vez las empresas que sobreviven tienen más cuota de mercado)

Por tanto, ante una industria madura y de decreciente competitividad (antes eran más competitivos, pero se está concentrando en pocas manos), se hace necesaria aportar valor diferencial más allá de la reducción en costes. Es por ello por lo que IMeureka decide llevar adelante este proyecto y muchos otros, las nuevas tecnologías serán su factor diferencial para entrar en el mercado de las corredurías.

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

El proyecto se va a llevar a cabo debido a que se estima que existe un gran nicho de mercado tanto para clientes de IMeureka como para corredurías de seguros. La idea de la aplicación es que sirva de catalizador para facilitar a los comerciales la venta de nuevas pólizas, se intenta que el cliente entienda que hay diferencias entre corredurías, que IMeureka y los brókeres asociados, además del seguro, ofrece otras muchas ventajas que inclinarían al cliente, a igualdad de precio, a decidirse por IMeureka. Hay que tener en cuenta que el mercado asegurador es un mercado muy maduro, por lo que cada cliente que contrata una póliza con la correduría de IMeureka es un cliente que ha perdido otra correduría en muchas ocasiones. Se entienden así las agresivas políticas de retención (si un corredor / aseguradora tiene información de que un cliente está viendo precios para su póliza, en algunas ocasiones llaman ofreciendo descuentos). Conseguir nuevos clientes no es fácil ya que la gente es reacia al cambio en general. Ofreciéndoles más ventajas aparte de la diferenciación en costes se pretende conseguir que acepten este cambio.

Para las corredurías que utilicen los servicios de IMeureka, la ventaja es clara. Más tiempo para buscar negocio y menos tiempo “desaprovechado” en tareas de gestión.

4.2 OBJETIVOS

El objetivo del proyecto es diseñar una aplicación multiplataforma para el front-end y su integración en el back-end que permita la declaración de siniestros, acelerando el proceso descrito anteriormente, y liberando de trabajo repetitivo a los brókers de Imeureka y de las corredurías que contraten el servicio.

Para que la app pueda ser utilizada por los usuarios, éstos se la deben poder descargar, por lo que el punto de partida será la cumplimentación de los requisitos para que sea aceptada en los Marketplace de Android e IOS (Google Play y APP Store).

El diseño y flujo de la aplicación debe ser el mismo en ambos sistemas operativos, de tal manera que a nivel de usuario no se aprecien diferencias significativas.

La aplicación debe ser muy sencilla y auto explicativa, ya que en principio los usuarios no son previamente entrenados para su uso. Un usuario medio debería poder declarar un siniestro sin necesidad de contactar con su bróker (salvo dudas especificas). Se accederá a la aplicación mediante el correo electrónico, que deberá ser previamente registrado en la plataforma de Imeureka. La aplicación está concebida para clientes que ya tienen póliza con IMeureka, con lo que ya tienen una cuenta en la plataforma. Es por ello que no se puede registrar alguien descargando la aplicación de los markets.

Las funcionalidades básicas que tendrá la aplicación serán las siguientes:

- Declaración de siniestros: Es el objetivo principal, la aplicación se engloba en el contexto de este objetivo. La declaración de siniestros debe ser muy sencilla, los campos que se podrán rellenar serán
    - Localización: para este campo, se tratará de que sea lo más sencillo para el usuario posible, para ello, el usuario podrá geolocalizarse con la ubicación

del móvil, y será éste el que automáticamente rellene toda su dirección. Esto puede hacerse a través de geocoding de google Maps.

- Archivos del siniestro (todo lo que el usuario considere necesario; imágenes, videos, documentos...): Se deberá poder acceder a cualquier archivo del almacenamiento interno del teléfono. En principio, el usuario puede subir cualquier tipo de archivo, ya que no es estándar. Los archivos estarán limitados en tamaño para evitar abusos / errores que el usuario pueda cometer
- Fecha y hora en la que ocurrió el siniestro: Similar a lo anterior, se tratará de que la usabilidad sea máxima, mediante la implementación de algún reloj o similar user-friendly
- Titulo
- Descripción

Es muy importante que las diferencias que puedan existir en la tramitación de los seguros sean transparentes para el usuario, ya que ese es el problema que se intenta tratar.

- Los siniestros deberán quedar almacenados en la BD de IMeureka, de tal manera que sean accesibles desde la plataforma. Deberán tener los mismos campos que ya tenían en la plataforma de tal manera que sea la aplicación la que se adapta a la BD existente, y no al revés. Los archivos asociados serán guardados en el cloud de Google, siguiendo el esquema de rutas predefinido por la plataforma.

- Visualización de los siniestros declarados: Se podrán ver todos los detalles asociados a un siniestro, tanto si ha sido declarado a través de la app como si no.

- Visualización de las empresas de un usuario: En caso de que un usuario tenga más de una empresa con IMerueka, podrá visualizarlas y seleccionarlas para ver las pólizas asociadas a una empresa. (La declaración de un siniestro va asociado a una póliza concreta)

Debido a la confidencialidad de los datos con los que trata la aplicación, se hace un especial hincapié al principio del proyecto en la seguridad de esta. Deberán establecerse mecanismos de seguridad y cortafuegos para que un usuario malicioso no pueda de manera sencilla obtener todos los datos de la base de datos de IMeureka a través de lo que concierne a la app.

Por otro lado, la API será desarrollada en paralelo con los métodos que vayan siendo necesarios para la cumplimentación de los objetivos de la app. Los métodos de la API deben ser suficientemente generales de tal manera que no solo puedan ser usados por la aplicación, si no por cualquiera que necesite su información a futuro (otra empresa, otra plataforma, otra app...). La generalidad de los métodos es necesaria porque al ser una startup la base de datos puede ser cambiada, por lo que al programarlos lo más genérico posible, se ahorran futuras incompatibilidades.

La API deberá tener al menos los siguientes métodos:

- Login: Permite el acceso a la aplicación , siempre con credenciales
- Get_empresa : Consigue las empresas de un usuario
- Get_pólizas : Consigue las pólizas de una empresa
- Get_siniestros_poliza : Consigue los siniestros existentes asociados a una póliza
- Declarar_siniestro : Sirve para subir los datos de un siniestro a la plataforma
- Modificar_siniestro : Modifica los datos de un siniestro existente
- Get_datos_siniestro : Descarga los datos de un siniestro, se usará en la visualización del mismo
- Storage_list : Lista los archivos almacenados por un usuario en el storage de Google Cloud
- Storage_download : Descargar archivos de un usuario

- Storage_delete : Borra archivos

- Storage_upload : Sube los archivos de un usuario

A priori, se estima que con estos métodos serán suficientes para llevar a cabo la funcionalidad de la aplicación.

La comunicación entre la aplicación y la API será mediante archivos JSON.

4.3 METODOLOGÍA

En primer lugar, se comenzará por el aprendizaje del estado del arte actual de las corredurías de seguros, su antiguo modelo de negocio, las novedades en el modelo que plantea implementar IMeureka, el lenguaje propio de la intermediación de seguros ...

Tras la introducción a la empresa, se aprenderá el lenguaje requerido para desarrollar la aplicación, Dart. Se descargará el IDE y se realizará un curso empezar a crear la aplicación.

Una vez se conozca el lenguaje de Dart, se empezará a crear la API, integrada con la plataforma actual de IMeureka. Se podrán hacer pequeñas modificaciones a la API en caso de que fuese necesario en cualquier momento.

Posteriormente, se comenzará con el desarrollo de la app propiamente dicho. Se empezará con el planteamiento del flujo de la aplicación y de las pantallas básicas. Posteriormente se estudiarán librerías especificas (como Google Maps para Flutter) para añadir funcionalidad especifica a la aplicación. La aplicación debe estar terminada mínimo 1 mes antes de la presentación, para disponer de tiempo suficiente para terminar con la documentación (que se ha ido realizando paralelamente) en todo momento.

4.4 PLANIFICACIÓN Y ESTIMACIÓN ECONÓMICA

La planificación temporal a lo largo del curso académico es la siguiente:

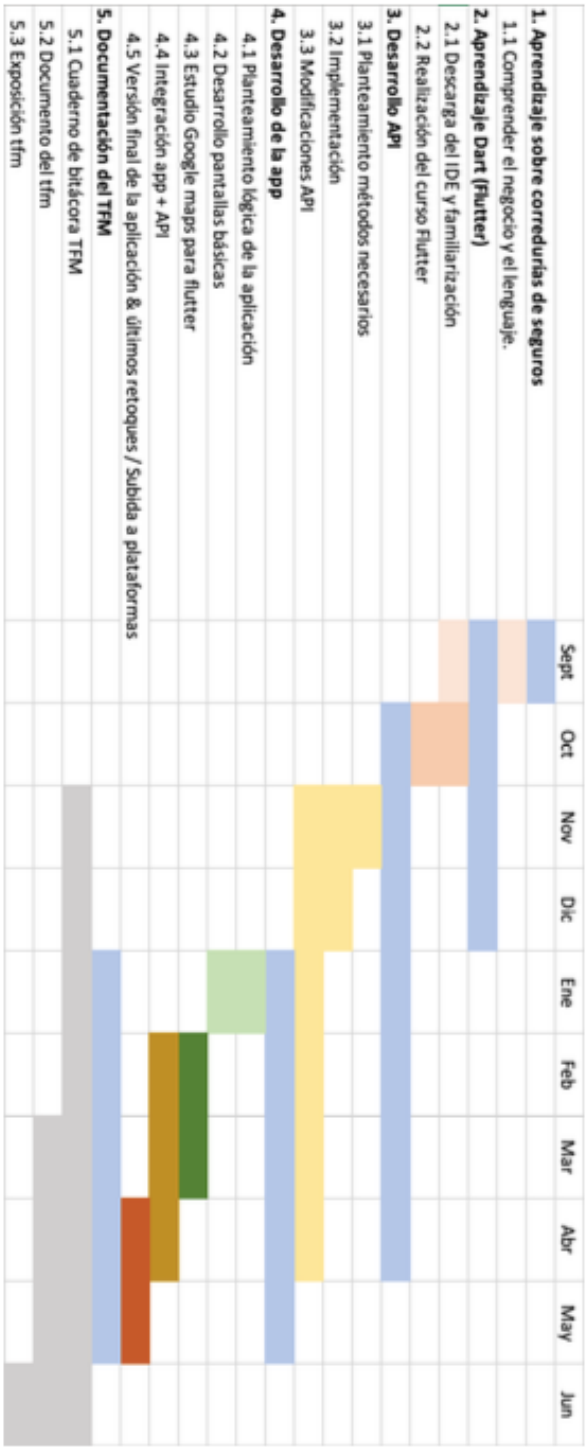


Ilustración 10: Planificación

Donde se puede ver que el grueso del trabajo se encuentra en la API y en el desarrollo de la app, lógicamente.

Por otro lado, se le debe dar una justificación económica a la aplicación.

Para los clientes de IMeureka la app es gratuita, sin embargo, podría llegar a monetizarse directamente para los clientes de otras correduría. Aún no está decidido. Es gratuita en el Marketplace y toda su funcionalidad es gratuita. Sin embargo, esta app no se hace sin tener en cuenta el factor económico, ya que hay tres maneras concretas en las que se va a rentabilizar.

Incremento de clientes de la correduría: La app será un factor diferencial que ayudará a los comerciales a vender la correduría como un servicio diferenciado de la competencia, ayudando en el proceso de venta.

Incrementos de clientes del Marketplace: Abriendo la aplicación al Marketplace en general, cualquier correduría podrá utilizarla con sus propios clientes, lo que les ayudará a reducir trámites innecesarios y en consecuente a aumentar su rentabilidad. Las corredurías pagarán un canon por usar los servicios desarrollador por IMeureka, que aún está por determinar.

Disminución del tiempo de gestión de siniestros de la correduría propia de IMeureka.

Los costes estimados para la empresa serán:

Producto	Coste estimado
Becario 6 meses	1800 €
Licencias Apple	100 €
Licencias Android	30 €

Ya que el resto de los costes (dominio / servidores) no son imputables a la aplicación en sí porque ya existían antes de ésta, e iban a seguir haciéndolo. La aplicación sólo se apoya en ellos, pero no le supone un costo extra.

Los beneficios económicos estimados de la aplicación prefieren no ser revelados por IMeureka. Pero al realizar el proyecto, se entiende que se la va a sacar una rentabilidad positiva. Además, es difícil de calcular porque, como hemos explicado, en la parte de la correduría de seguros los números son difícilmente adjudicables (¿Porqué se ha decidido un cliente, ha sido por el comercial, por las ventajas, precio, descontento con su situación anterior …?)

Sin embargo, lo que sí se puede calcular es el ahorro en tiempo que supondrá para un gestor de la correduría el uso de la aplicación. Si nos atenemos a los 20-30 emails que se estimaron en la introducción (usaremos la media, 25).

Si el tiempo de redacción de cada email por el gestor es de unos 5 minutos, esto significa que en media por cada siniestro el intermediario usa unas 2h de su tiempo. Si el coste para la empresa total del trabajador es de 2000€/mes (sueldo neto de 1250€ aprox) el

coste por hora del trabajador es de 12.5€/h , por lo que cada la gestión de cada siniestro, en términos económicos, le cuesta a las correduría unos 25€. Con la app de se va a disminuir mínimo en un 50% los emails. Con lo que el ahorro será aproximadamente de 12,5€ por cada siniestro declarado por los clientes de media.

Tras el desarrollo inicial de la app, será necesario un mantenimiento y actualización del código para implementar las nuevas funcionalidades vayan siendo necesarias conforme se conozcan mejor las necesidades de clientes y aseguradoras.

Capítulo 5. SISTEMA/MODELO DESARROLLADO

5.1 INTERFAZ Y EXPERIENCIA DE USUARIO

La interfaz de usuario es la primera entrada a la aplicación, en el subconsciente, una buena interfaz invita a usar la misma. Si al entrar a la aplicación, esta es compleja y poco atractiva, la interactividad de los usuarios con la aplicación será menor, con lo que no conseguirá sus objetivos de agilización.

En este sentido, podemos identificar tres características principales de las interfaces de usuario que aumentarán la posibilidad de que tengamos una aplicación de éxito.

- Simplicidad

  Es vital para una aplicación móvil que contengan interfaces lo más claras posibles, poco recargadas. Esto aumenta la usabilidad de la aplicación por los usuarios medios. Cada pantalla y botón deben tener su función concreta, y deben estar limitadas de tal manera que sólo haya funciones importantes. De esta manera, al descargarse la aplicación, el usuario sabrá usarla directamente.

- Consistencia

  La navegación a través de la aplicación debe ser consistente e idempotente, es decir, un botón siempre realiza la misma función. Un ejemplo de no consistencia sería que al usuario que recientemente haya creado un siniestro se le lleve a ver el siniestro directamente, ya que es probable que quiera ver los detalles. Esto sería una inconsistencia en el flujo de la aplicación explicado, que inevitablemente dificultaría su uso.

- Navegación intuitiva

  La navegación debe ser lo suficientemente intuitiva para que un usuario medio sea capaz de usar las funciones de la aplicación sin conocimientos extra, únicamente con lo que ya conocía sobre su dispositivo.

Finalmente, se debe hablar de la forma en la que se sostiene el móvil, ya que cambia la interfaz totalmente. Los móviles se encuentran la mayoría del tiempo en vertical, mientras que las tabletas se suelen poner más en horizontal. La orientación horizontal ayuda a aprovechar más la pantalla, mientras que la vertical favorece la usabilidad, ya que en móviles se podrán usar con una mano.

En el caso de esta aplicación, se ha optado por un diseño que cumpla los tres patrones anteriores con una orientación vertical que no cambia al poner el terminal en horizontal por dos razones; la primera es que para usar la aplicación es necesario conexión continua a internet, y esto no suele ser común en tabletas, que suelen usar más wifi que conexiones tipo 3G/4G. La aplicación está enfocada a la productividad del cliente, además, está pensada para ser usada en el lugar del siniestro (recoger información, sacar fotos …) por lo que debe ser ágil.

Vamos a ver la UI que se ha implementado y a justificar porqué cumple con los patrones, por ejemplo, en la pantalla de login.

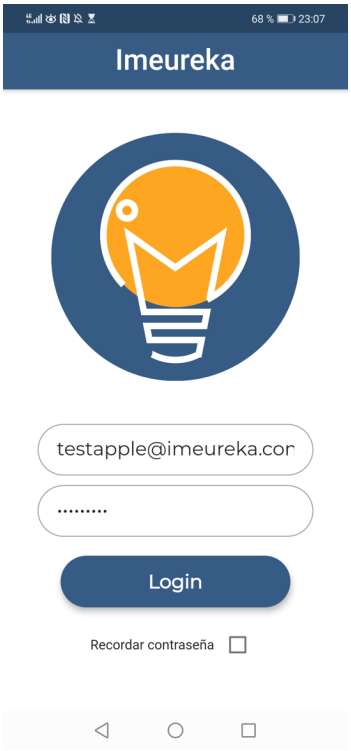


Ilustración 11: Pantalla de login

Se puede ver en la imagen que la pantalla de login cumple con lo especificado anteriormente, tiene las mínimas funciones necesarios con la mayor claridad posible. Además, es muy sencillo prever que ocurrirá al pulsar cada botón. Al pulsar sobre los recuadros de texto, nos aparecerá el teclado del teléfono para escribir. Una vez rellenos email y contraseña, podremos pulsar en Login para acceder a la aplicación. Si marcamos "recordar contraseña" nuestra contraseña será recordada en futuros logeos.

Ejemplo de una mala interfaz sería por ejemplo incluir más texto del necesario, como una descripción de la empresa en el login (¿Qué es IMeureka?) … Para facilitar la usabilidad a los usuarios, tienen el botón de "recordar contraseña", de manera que recuerde las últimas credenciales correctas que se introdujeron en la app.

Para el diseño visual, se decidió realizar las interfaces desde 0. Usando clases integradas en ciertos plugins de flutter, se consiguen botones, campos de texto, mapas… Luego, todos

estos son modificables programáticamente para adaptarlos a los tamaños y colores decididos por la empresa.

En ciertas ocasiones, los desarrollos visuales son creados por un diseñador o desarrollador fron-end. En este caso no fue así. En conjunto con la empresa se decidieron una serie de "pautas de diseño". En general se buscaron ejemplos de aplicaciones en internet y se fue viendo qué encajaba con la imagen de la empresa y qué no.

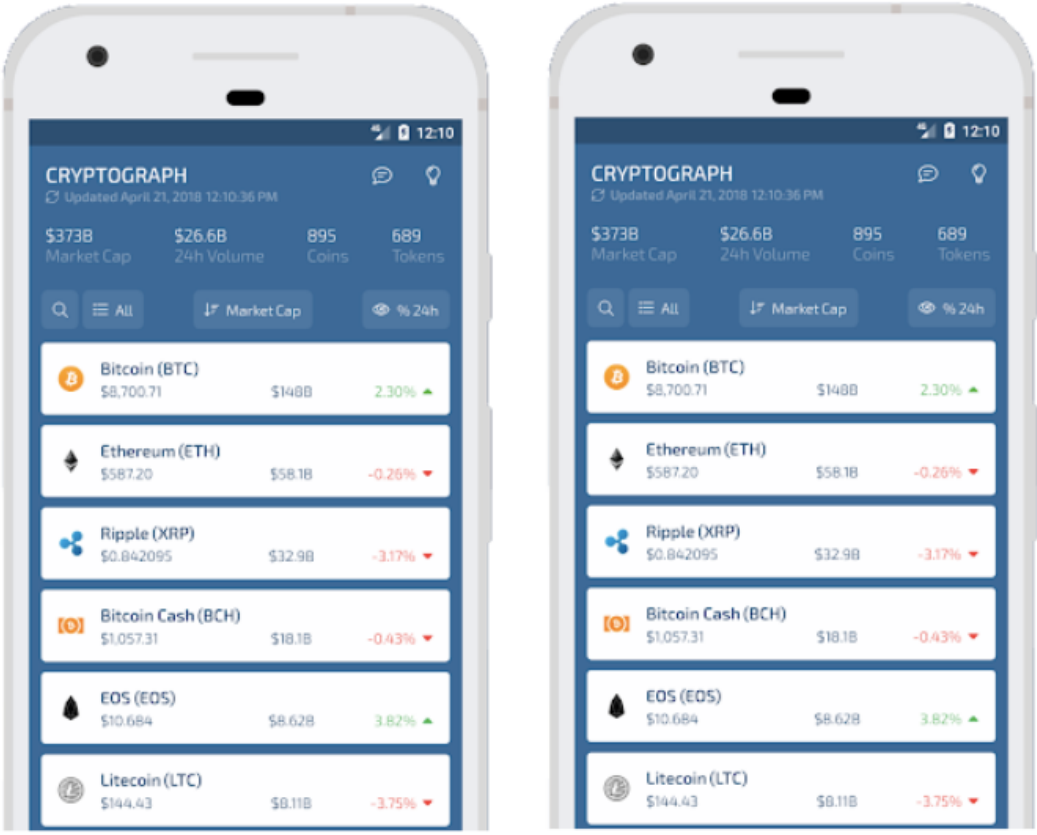


Ilustración 12: App ejemplo

Se decidió que el diseño fuera similar al de esta aplicación, los colores… las listas… Esto representó el "punto de partida" sobre el que luego se iteró múltiples veces.

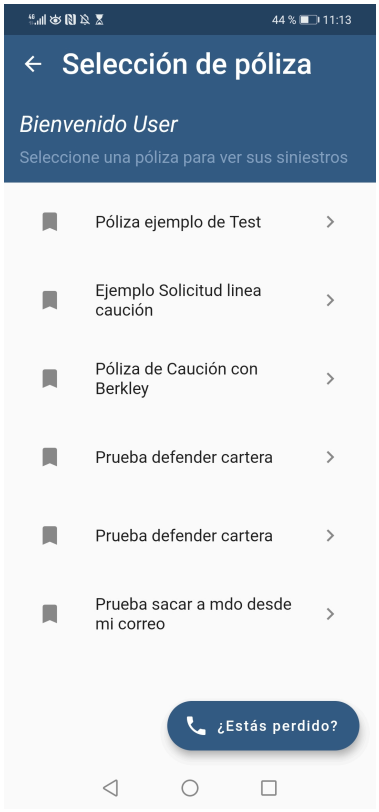


Ilustración 13: Pantalla de selección de póliza

Este es un ejemplo de pantalla que cumple también con los patrones explicados. Dado que su objetivo es la selección de pólizas, éstas se encuentran en el centro de la pantalla, y la flechita lateral indica que al pulsarla se cambiará de pantalla. La flechita de arriba a la izquierda indica que tras pulsarla se irá atrás, a la pantalla anterior. Se le da un pequeño mensaje de bienvenida al usuario. Es importante destacar que se le presupone al usuario un conocimiento básico del funcionamiento de su dispositivo, de esta manera, se asume que sabrán que una flecha junto al título significa "acceder aquí". Esto hace que la interfaz quede más bonita.

Una posible crítica a esta pantalla es el botón de ¿Estás Perdido? Que es posible que no cumpla completamente con los objetivos de diseño, ya que las primeras versiones del desarrollo este botón no incluía el icono del teléfono, y además no tiene relación con esta pantalla en concreto. Aún así, se ha decidido poner ya que ese botón conecta por llamada directamente con la persona responsable de esa cuenta. El botón se decidió que se encontrase

en todas las pantallas de la aplicación, para solucionar cuánto antes los problemas que pudieran tener los usuarios.

5.2 CASO DE USO ESPERADO

En esta sección se va a explicar el caso de uso esperado para la aplicación, el usuario típico no conoce la aplicación por casualidad o por publicidad, sino que es el resultado de contratar alguna póliza con la correduría de IMeureka, es por ello que la propia aplicación no cuenta con registro. El usuario medio, actualmente, conoce primero la plataforma de IMeureka a través de los medios de la empresa de captación de clientes, y un comercial le introduce a la aplicación.

Es de esperar que no sea una aplicación en la que los usuarios entren a diario, ya que únicamente accederán cuando tengan un siniestro. Hay que tener extremo cuidado con el funcionamiento de la aplicación y testearla bien por dos motivos. El primero y más importante es que el cobro del siniestro es una parte fundamental del seguro en el cuál el cliente espera que todas las promesas que se le hicieron a la hora de contratar el seguro se hagan realidad y de manera rápida, por ello, debe hacerse hincapié en la consistencia de la aplicación. Los datos se guardan bien, la aplicación no "colapsa" … El segundo motivo es que en general, cuando se usa la aplicación, se tiene a estar de mal humor desde un punto de vista psicológico. Esto es debido a que a nadie le alegra tener que entrar en el proceso de tramitar un siniestro, es molesto para todos los integrantes del proceso, por lo que debe simplificarse al máximo.

El usuario entra a la aplicación, y gracias al botón de guardar contraseña, tiene directamente escritos los campos de email y contraseña. Al acceder, entrará directamente a la pantalla de selección de empresas (en caso de que el usuario tenga sólo una empresa, esta pantalla se omite). Posteriormente el usuario selecciona la póliza (en caso de que sólo tenga una, esta pantalla se omite) y se llega a la pantalla principal de la aplicación; la de siniestros.

En este sencillo proceso se le ha ahorrado al usuario mucho tiempo, no ha tenido que buscar en su email la póliza de seguro que quizás firmó hace años, hacer una llamada de teléfono o email al corredor y éste le indica los pasos a seguir… Con la aplicación, todo esto es directo.

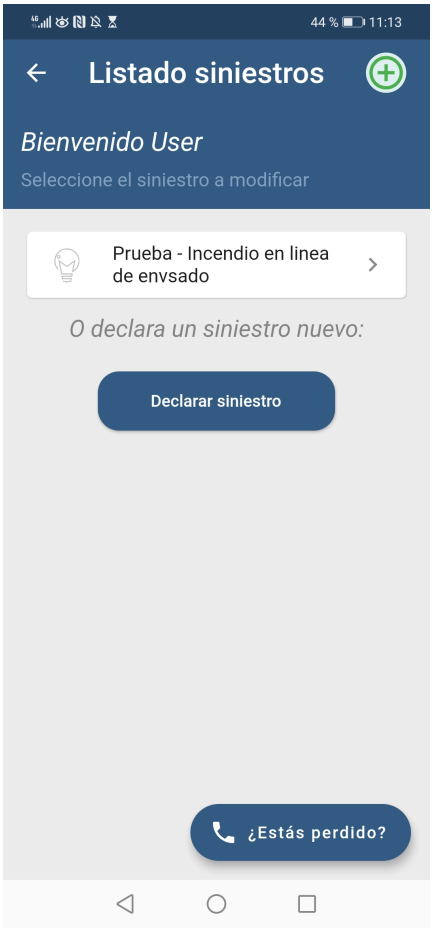


Ilustración 11: Listado de siniestros

En esta pantalla, el usuario declara un nuevo siniestro o ve la información de uno antiguo. Se entiende que lo más habitual será la declaración del siniestro.

La pantalla de declara siniestro será un formulario sobre el que hacer scroll:

Ilustración 15: Declaración de siniestros - ubicación

El usuario tiene dos opciones, seleccionar su ubicación actual o seleccionar otra ubicación, y pinchar sobre el mapa la ubicación del siniestro. Al hacer esto, automáticamente se le rellenan el resto de los campos de la dirección (calle, provincia, código postal…) aunque puede cambiarlo en caso de que algún dato se haya obtenido mal.

Finalmente, se declara el siniestro tras rellenar todos los datos necesarios y empieza el proceso.

5.3 DISEÑO DEL BACKEND

El backend queda dividido en tres partes diferenciadas, la primera el desarrollo de la API, la segunda la comunicación entre app y API y la tercera el desarrollo de la app (frontend) en sí. Gracias a este modelo planteado, la app debería ocuparse sólo de la parte visual y de captar la información del usuario. A la API le llegarán los datos siempre con el formato correcto y dará respuestas preestablecidas.

5.3.1 LÓGICA DE LA API

Los métodos para el caso de uso básico de la aplicación serán los siguientes:

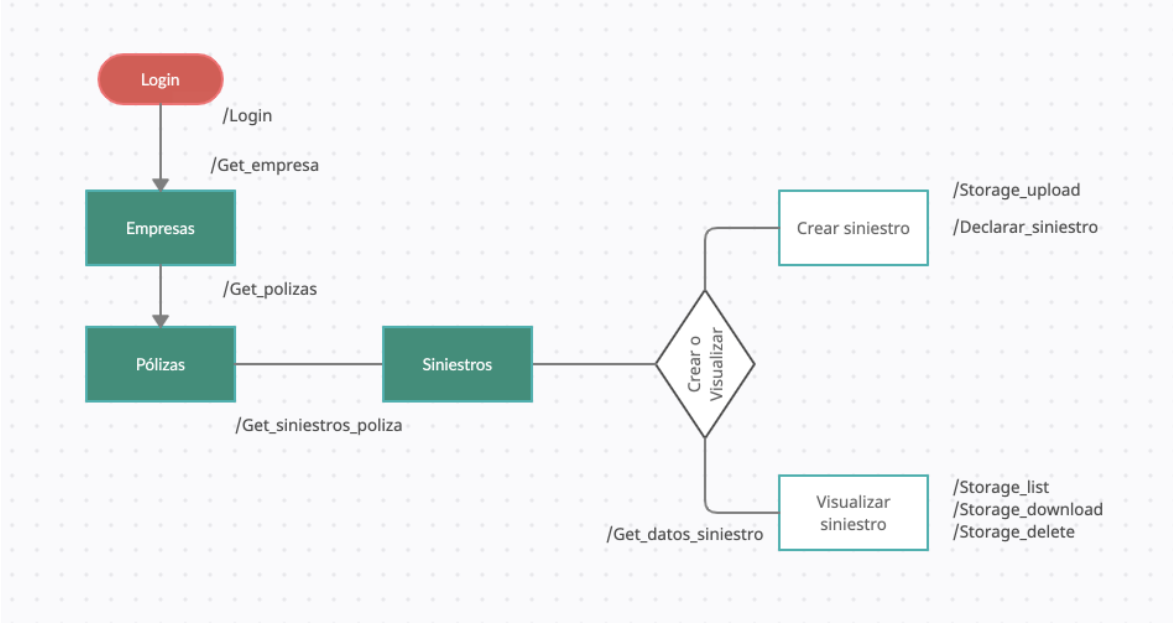


Diagrama 16: Lógica de la api

Desde la pantalla de login se establece una navegación forward en la aplicación. Aunque se dispone de botón de atrás, en el caso ideal el usuario realiza una "navegación hacia delante" hasta terminar con o declarando un siniestro o viendo un siniestro.

Este diagrama muestra la lógica que se sigue en la navegación a través de la aplicación, a la pantalla de login se accede directamente tras iniciar la app. Tras introducir correctamente usuario y contraseña se accede a la pantalla de empresas, en caso de que únicamente haya una empresa, esta pantalla tendría un único botón seleccionable, por lo que se omite (se selecciona automáticamente la empresa sin que el usuario la vea). Después, se accede a la pantalla de pólizas, aquí el usuario selecciona la póliza sobre la que interactuar con los siniestros. Tras esta selección, se va a la pantalla de siniestros, en la cuál puede seleccionar un siniestro existente o crear uno nuevo.

En caso de selección, se le permite ver el siniestro y modificar archivos. Finalmente se decidió no poder modificar los datos del siniestro por temas de negocio.

En caso de creación, permite crear desde cero los siniestros.

5.3.2 CÓDIGO API

Finalmente, se crearon dos archivos en el repositorio de IMeureka para gestionar los temas de la aplicación, API_SiniestrosController.php, que permite interactuar con a la app con la base de datos y API_StorageController.php permite almacenar archivos en el storage de Google Cloud.

Se van a explicar ciertos métodos de la api:

Get_siniestros_polizas

```
function get_siniestros_polizas() {
    //Recibe por POST la póliza de la que sacar los siniestros
    //Instanciamiento del framework
    $f3 = F3::instance();
    //Instancia del objeto "user" que contiene los datos del usuario (  nombre,
tlf…)
    $user = $f3->get('user');
    //Extracción del parámetro ID_póliza de la request. Sirve para
       realizar peticiones sobre los siniestros asociados a esa póliza
    $ID_poliza    = $f3->clean($f3->get("POST.ID_poliza"));
    //Creación del objeto póliza para conexión con BD (POO)
    $poliza = new Poliza();


       //Si el usuario no puede ver esa póliza …
    if(!$poliza->puede_ver(array('ID_user'=>$user['ID_user'],'ID_poliza'=>
$ID_poliza))) {
       //Se devuelve directamente un error
         $result = array("status" => "ERROR", "message" => "OPERACION NO
PERMITIDA");
       //Error devuelto como JSON

         $this->json($result);
    }else {
       //Creación del objeto siniestros
         $siniestro = new Siniestro();
       //Query, devuelve los siniestros asociados a una póliza
         $res = $siniestro->get_siniestros_poliza($ID_poliza);
         if (!empty($res)) {
            Si la póliza tiene siniestros, los devolvemos como JSON
             $result = array("status" => "OK", "message" => "SE HAN OBTENIDO SUS
SINIESTROS ASOCIADOS A LA PÓLIZA", "result" => $res);
             $this->json($result);
         } else {
             $result = array("status" => "ERROR", "message" => "NO HAY
SINIESTROS", "result" => $res);
```

```
            $this->json($result);
        }
    }
}
```

Puede_ver es un método que le aporta seguridad a la aplicación, ya que una vez que el usuario hace login, cuenta con un token que le permite hacer todas las peticiones que quiera a través de la app. Para usuarios normales que utilizan la aplicación sin malas intenciones, la función puede_ver (que analiza si el usuario tiene relación con esa póliza/siniestro que está pidiendo, y por tanto decide si tiene acceso a ello) no sería necesaria. Sin embargo, un atacante fácilmente podría identificar las peticiones que se realizan y cambiar los parámetros a su antojo. Incluso podría extraer toda la base de datos de IMeureka mediante la aplicación de bucles en sus peticiones. Para prevenir esta hipotética brecha de seguridad directamente se creó la función puede_ver.

Esta consideración de seguridad es muy importante, sin ir más lejos, el 10 de junio de 2021 salió a la luz esta noticia: "Un error de programación permitía acceder a los datos personales de los ciudadanos introduciendo códigos de identificación sanitaria aleatorios en el sistema. La Comunidad de Madrid ha cerrado la brecha de seguridad tras el aviso de elDiario.es"[15].

En esencia esta brecha de seguridad que le ha ocurrido a la comunidad de Madrid es la que se ha prevenido en este epígrafe del trabajo.

Por otro lado, se muestra un ejemplo de query sencilla a la BD de Imeureka:

```
        //API -> get siniestros by ID poliza, método de la clase Siniestro
    public function get_siniestros_poliza($ID_poliza){
        $f3     = F3::instance();
        $db     = $f3->get("DB");
        $sql = "";
        $params = array(1 => $ID_poliza);
       //La query en este caso es sencilla, un SELECT datos WHERE la póliza la
que ha seleccionado el usuario.
        $query = "SELECT ID_sin , Asunto , Descripcion FROM siniestro where
ID_poliza = ?;";
        $result = $db->exec($query, $params);
        return $result;
    }
```

Como se puede observar, se ha hecho mucho hincapié en la orientación a objetos del código. Esto es debido a que facilita la integración y la reusabilidad con otras partes del código de IMeureka.

La petición resultante, a través de Postman, es la siguiente:

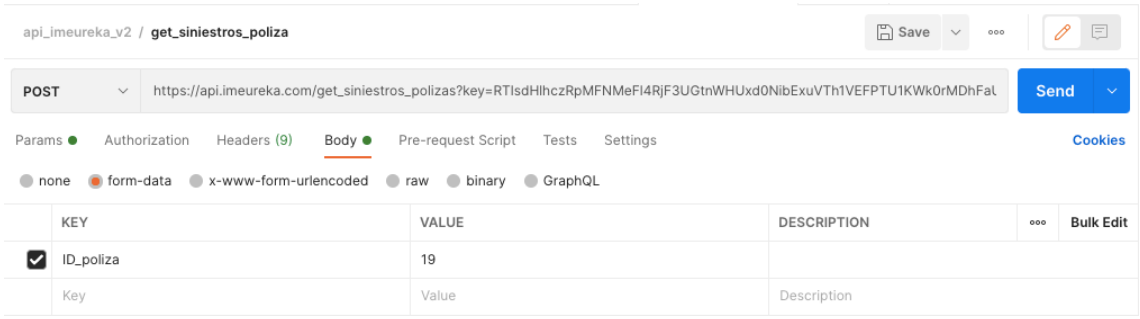


Ilustración 17: Postman petición get_siniestros_póliza

En esta imagen se puede observar que la petición utiliza el método POST , lleva una key en la URL y en el body el parámetro ID_poliza. El JSON resultante es el siguiente:

```
{
    "status": "OK",
    "message": "SE HAN OBTENIDO SUS SINIESTROS ASOCIADOS A LA PÓLIZA",
    "result": [
        {
            "ID_sin": "7",
            "Asunto": "Prueba - Incendio en linea de envsado",
            "Descripcion": "Se quemo la linea 3 de envase pet"
        }
    ]
}
```

Con lo que queda explicada los métodos que interacción con la base de datos MYSQL de IMeureka, en concepto el resto son similares.

Los archivos no son guardados en la base de datos MYsql, si no en Google Storage. La conexión se realiza a través de la api de google.

storage_list

```
static function storage_list($data = array()) {
      // Esta función tiene dos maneras de utilizarse, o pasándole en $data un
array de clave-valor, o con los datos que almacena el framework de la última
```

```
request
    $f3 = F3::instance();
    if(!empty($data)){
       // Si viene con datos, los copiamos a la request.
        $_REQUEST = $data;
    }
     // Extraemos los datos necesarios de la REQUEST
         $tipo_archivo = REQUEST['Tipo_arch'];
    $id = $_REQUEST['ID_tipo_arch'];
    $esJSON = isset($_REQUEST['JSON_file']) ? $_REQUEST['JSON_file'] : 0;
    // Comprobamos que efectivamente los datos no eran null
    if(empty($tipo_archivo) || empty($id)){
        $respuesta = array( 'status'  => '1',
                            'message' =>'Dato no introducido',
                            'results' => array());
    }else{
    // Se instancia el Objeto StorageClient que nos permite conectarnos a
       BD
        $storage = new StorageClient(['projectId' => $f3-
>get('STORAGE_PROJECT_ID'), 'keyFilePath' => $f3->get('storageCredentials')]);
    // Un bucket es una "carpeta" en google cloud
        $bucket = $storage->bucket($f3->get('BUCKET'));

        $res=array();
        $contador=0;
        $objects = $bucket->objects(['prefix' =>
self::getPrefix($tipo_archivo,$id,$esJSON)]);
        //Recorre el array de objetos creado buscando nombres, si hay nombre es
que hay objeto y le saca los datos
        foreach($objects as $object){
            $info=$object->info();
            $res[$contador] = array('Nombre_archivo' => basename($object-
>name()),
                                    'Fecha_creacion' => $info['timeCreated'],
                                    'Tamaño'         => $info['size'].' Bytes',
                                    'Tipo'           => $info['contentType']
                                );
            $contador++;
        }
        //Si no se han sacado resultados se envia el error
        if(count($res)==0){
            $respuesta = array( 'status'  => '2',
                                'message' => 'Ningun documento adjunto',
                                'results' => $res);
        }else{
            $respuesta = array( 'status'  => '0',
                                'message' => 'Todo correcto!',
                                'results' => $res);
        }
    }
    return $respuesta;
}
```

Para el Storage también se han tenido en cuenta consideraciones de ciberseguridad, dos principalmente.

La primera es que el tipo de archivo que se permite subir (por ello la variable tipo_arch) está limitada al tipo de archivo que subirán los usuarios de la plataforma. Las extensiones típicas son .jpg, .docx, .pdf, .mp4 … La lista es más extensa, pero todos los archivos que se encuentren fuera de ella serán rechazados, estos incluyen pero no se limitan a .sh,.py…

Otro método de seguridad es el tamaño máximo del archivo que se permite subir, esto ya no es tanto un riesgo de ciberseguridad si no errores de los usuarios o mala fe. Debido a que el Storage es un servicio de pago por uso de Google, hay que aprovechar el servicio usándolo lo mínimo posible. Es seguro que un archivo relacionado con un siniestro no va a ocupar más de 1GB, por lo que se establece este límite.

En este punto, ya se tiene una visión completa de la estructura de la API en el servidor de IMeureka. El siguiente paso es la descripción de cómo se realiza una petición desde la APP. En Dart, el método getEmpresas :

```
      //La asincronía es muy utilizada en funciones en las que se dependen de
recursos externos, no se puede "congelar" la aplicación hasta que responda la
API, si no que debe ir en paralelo.
static Future<EmpresasObject> getEmpresas() async {
  debugPrint('Ejecutando getEmpresa');

     //Las keys se guardan en una clase global, esto porque de esta manera si al
key cambia sólo hay que modificarla en un lugar del código, y no en cada método.
  String url = 'https://api.imeureka.com/get_empresa?' +
      'key=' +
      Globals.userCompleteObject.token;
  debugPrint('LA URL' + url);
  Map<String, dynamic> body = {};
    //Se usa el paquete http de flutter para realizar la petición, cumple con lo
que se especificó anteriormente, usa el método POST, recibe un JSON
  var response = await http.post(
    Uri.parse(url),
    body: body,
    headers: {
      "Accept": "application/json",
      "Content-Type": "application/x-www-form-urlencoded"
    },
  );
```

```
  debugPrint('Post realizado, ha devuelto' + '' + response.body);

  if (response.statusCode == 200) {
    // Si el servidor responde con un 200 y todo ha ido correcto, se desearliza
el JSON en un Objeto del tipo EmpresasObject() y se devuelve. Esto concuerda con
POO

    EmpresasObject empresasObject = new EmpresasObject();
    empresasObject = empresasObjectFromJson(response.body);

    return empresasObject;
  } else {
    // No es habitual que se llegue a este punto, puede ser debido a acciones
maliciosas (petición sin token) o a problemas puntuales del terminal (falta de
cobertura)
    if (response.statusCode == 401) {
      print('Unauthorized en empresas');
    }
    debugPrint("Ha fallado");
  }
}
```

La particularidad de los métodos de storage en la app es que los archivos se deben guardar temporalmente en la memoria del teléfono para poder mostrarlos.

Finalmente, se muestra un ejemplo de petición HTTP:

```
HTTP: POSTMAN
POST
/get_siniestros_polizas?key=bjNjT09vbThYSHVVUUFlZzAvK2RlRzlXTWxkRWg3RXFQUzFvYW5GM
1ZiRDNiVlR2eE9zQWU5clRYUEg3aktZWnRYMlpwNXpPV3ZIOFFSYk5rZU9ubURxenlUWUNFY056T2pWRW
d6R2VTcDU5ZGxEUWhrcldpanB0K082QzFJQkk== HTTP/1.1
Host: api.imeureka.com
Content-Length: 129
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="ID_poliza"

71
----WebKitFormBoundary7MA4YWxkTrZu0gW
```

Vemos que el método es POST, en general nunca es recomendable que los parámetros se pasen en la URL ya que serían muy fácilmente modificables. Las keys son aleatorias y están asociadas por usuario , caducan con el tiempo, aquí se muestra una de ejemplo.

Las keys son devueltas por el método de login, y serán usadas a lo largo de toda la aplicación para proporcionar seguridad, ya que cada petición se puede asociar a un usuario concreto.

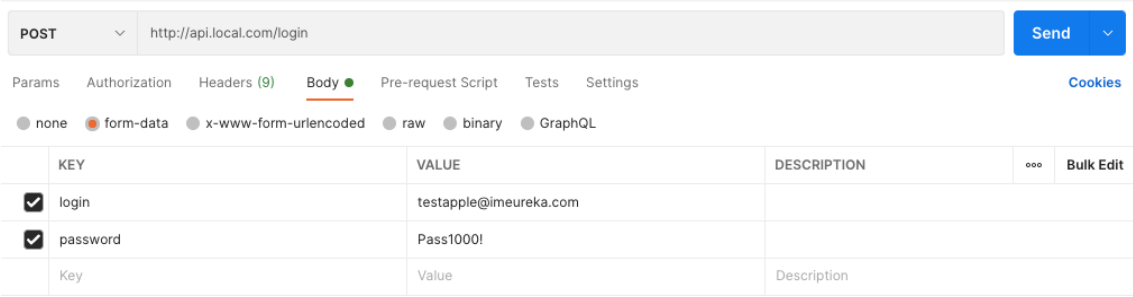


Ilustración 18: Postman login

Response:


```
{
    "status": "OK",
    "message": "Login Correcto",
    "token":
"RTlsdHlhczRpMFNMeFl4RjF3UGtnWHUxd0NibExuVTh1VEFPTU1KWk0rMDhFaUxDRVNzY3oxOUlIVHNB
RDdFU0tOMWljMCs1RytkOU1wSStENmV4THEydW44NWRWMkVlVlp0dmoyV3NBaHJSeVZVWWFtdzk5Z0lRa
zBBdk1yUjducTVWSTAzYm9hM0UxZnA3SUh3UkxBPT0=",
    "result": {
        "ID_user": 18,
        "Nombre": "User",
        "Apellidos": "Apple",
        "NombreGestor": "Pablo",
        "ApellidosGestor": "Collado Puerta",
        "Telefono": "XXXXXXXX"
    }
}
```

5.3.3 BASE DE DATOS

La base de datos es común tanto para la plataforma de IMeureka como para la app, por lo que cuenta con más campos (algunos no son necesarios en la declaración a través de la app). Se va a explicar los campos de la tabla siniestros,

ID_sin : Es un campo numérico , clave primaria, que le asigna una identificación única a cada siniestro. Este campo es un autoincremental.

ID_poliza: Es un campo numérico, relaciona el siniestro con la póliza , ID_póliza es clave externa de la tabla pólizas , lo que significa que todo siniestro debe ir relacionado a una póliza previamente existente.

ID_rel_user_clientes: Es un campo numérico, relaciona el siniestro con el usuario que lo ha declarado.

ID_user_tramitador: Es un campo numérico, relaciona el siniestro con el tramitador asignado.

ID_gabinete: Es un campo numérico, relaciona el siniestro con el gabinete pericial asignado. Un gabinete pericial es "Un gabinete pericial es un grupo de peritos profesionales y especialistas, con capacidad para elaborar informes periciales, así como dar servicios de peritaciones judiciales."

Fec_sin: Es un campo tipo TIMESTAMP, almacena la fecha en la que ocurrió el siniestro.

Fec_Intro: Es un campo tipo TIMESTAMP, almacena la fecha en la que el usuario introdujo el siniestro.

Fec_cierre: Es un campo tipo TIMESTAMP, almacena la fecha en la que se da por concluidas las gestiones relacionadas con el siniestro.

Estado: Es un campo numérico, almacena el estado (abierto, cerrado, en revisión…)

Indemnización : Es un campo numérico, en caso de que haya habido una indemnización por parte de la aseguradora, lo almacena.

Asunto: Es un campo de texto, breve resumen del siniestro "titular".

Descripción: Es un campo de texto, con detalles del siniestro.

Estado, Comentario gabinete : Son dos campos, sirven para controlar el estado de la gestión del siniestro por parte de los gabinetes periciales.

ID_prov : Campo numérico, almacena la provincia en la cual sucedió el siniestro.

Ciudad: Campo de texto, almacena el nombre de la población en la cuál ocurrió el siniestro.

Más datos de dirección: Campos de número, código postal… etc

Latitud y longitud : Dirección exacta almacenadas como numéricos.

Estos datos se rellenan en cada siniestro, al ir populando la base de datos de siniestros reales, se conseguirá información que le permitirá a la empresa a corto o medio plazo aplicar técnicas de inteligencia artificial en la predicción de siniestros. Aunque es un tema interesante y en el que se está trabajando actualmente en la empresa, no pertenece a este ensayo.

5.3.4 FUNCIONALIDAD & FRONTEND

Como se comentó, la app lleva un flujo unidireccional, desde el login hasta las acciones finales que pueden realizarse. A continuación, se va a realizar un estudio de cada pantalla y sus particularidades principales, ya que muchas tienen features que no estaban directamente reflejadas en los objetivos.

5.3.4.1 Pantalla de login

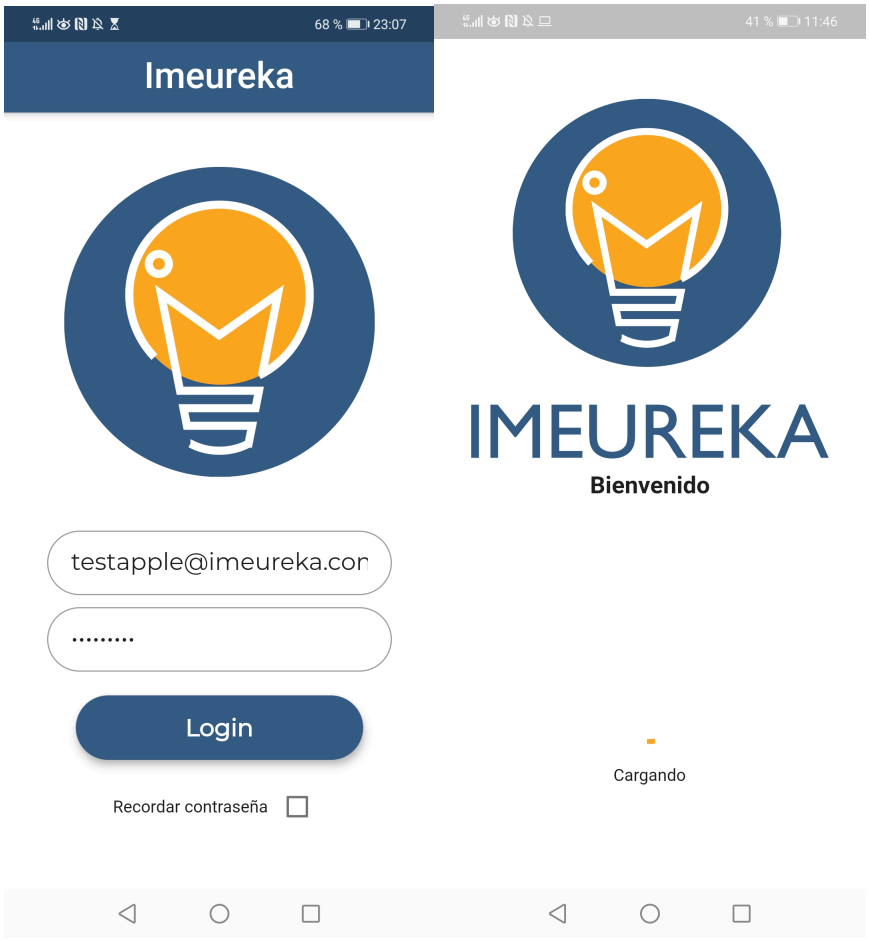


Ilustración 19: Izquierda Login, Dcha Pantalla de carga

La primera pantalla que ve el usuario es una temporal, que sirve de espera en lo que carga la aplicación. La pantalla de login aparecerá con los campos con unas indicaciones de "email" y contraseña. En caso de que el usuario seleccione "recordar contraseña" y acceda correctamente a la aplicación, la próxima vez que la abra ya le aparecerán sus datos escritos, de manera que sólo tendrá que pulsar en login.

Esto se consigue con la librería shared preferences de flutter, que se puede usar para guardar en el almacenamiento interno del teléfono parejas de clave-valor. [18]

```
if (savePwd == true) {
  prefs.setString("email", user_basic.email);
  prefs.setString("pwd", user_basic.password);
}
```

Nota: Al realizar la documentación y el estudio exhaustivo del código se descubrió una vulnerabilidad que deberá atajarse en la próxima actualización.

La vulnerabilidad reside en que la librería shared preferences por sí no utiliza ningún método de encriptación, si no que guarda archivos xml en el dispositivo que luego recupera. Aunque no es directamente accesible, con acceso root al dispositivo móvil sería fácilmente obtenible este archivo xml *que almacena contraseñas en texto plano*. [20]

Por ello, se deja como el próximo trabajo futuro (cercano) la implementación de una librería que sí encripte los datos de login, se usará flutter_secure_storage[19] la cual tiene un funcionamiento a nivel de desarrollador exactamente igual pero a nivel interno utiliza encriptación para guardar los datos. Con lo cual alguien con acceso root podrá ver unos archivos xml ilegibles, no en texto plano. La encriptación usada es Keychain en IOS, AES para Android y libsecret para Linux.

5.3.4.2 Pantalla de empresas/pólizas

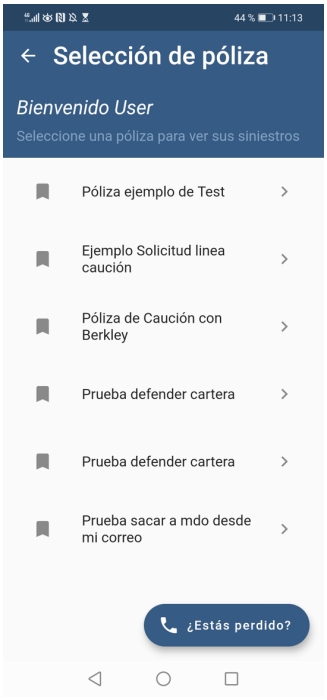


Ilustración 20: Pantalla de selección de póliza

Las pantallas de selección de pólizas / empresas son idénticas, lo que cambia es lo que se selecciona. Lo más destacable de esta pantalla, que se extiende a lo largo de toda la aplicación, es el botón de ¿Estás Perdido? Con éste, en caso de que el usuario no sepa cómo usar la app en algún punto, podrá ponerse en contacto con el servicio técnico de IMeureka para recibir ayuda. Cada persona tiene un teléfono asignado. Gracias a que en el login se devuelve el teléfono asociado, no es necesario realizar una petición.

Para dale un aspecto más cercano, bienvenido "User" es parametrizable, y cada usuario será el nombre con el que se registró.

Cabe destacar que, a nivel visual, los colores usados son los corporativos de IMeureka.

5.3.4.3 Pantalla de siniestros

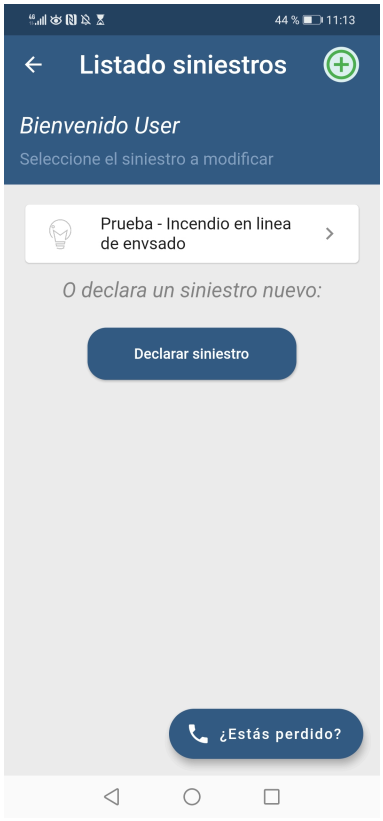


Ilustración 21: Listado de siniestros

En esta pantalla se puede elegir entre declarar un siniestro nuevo, mediante el botón azul o el + de la esquina superior derecha, o ver un siniestro existente. Podemos observar que los datos son los que se muestran en el JSON de la página 44.

5.3.4.4 ¿Estás perdido?

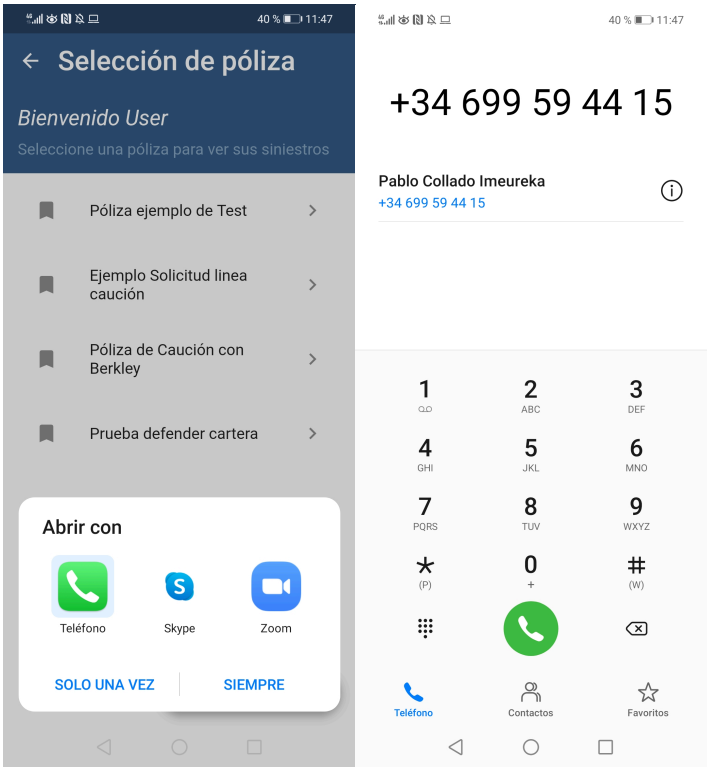


Ilustración 22: Botón de estás perdido

Este botón permite realizar llamadas con la persona gestora de la póliza con sólo pulsarlo. Es especialmente útil para clientes particulares que no hayan contratado la póliza ellos directamente (pej : deportistas federados). En caso de que tengan cualquier duda, la opción de una atención personalizada está ahí. Al pulsar en el botón, aparece una nube en la parte inferior de la pantalla en la que pone “Llamando a – Nombre del Agente --”.

Para la implementación de esta funcionalidad se ha usado el paquete de flutter url_launcher [21] que incluye la funcionalidad para realizar llamadas en el dispositivo. Cabe destacar que la decisión última de llamar (pulsar el botón) es del usuario. La aplicación nunca realizará llamadas sin el permiso explícito del usuario.

```
//La función launchURL permite utilizar la interfaz del teléfono en el que se
//está ejecutando la aplicación para realizar una llamada a través de los métodos
//disponibles en el dispositivo
static launchURL(context) async {
       //Se construte el teléfono con el prefijo +34 ya que de momento IMeureka
       //solo funciona en España
  String telefono = 'tel:+34' + userCompleteObject.result.telefono;

  debugPrint('EL telefono :' + telefono);

  Toast.show(
      "LLamando a su administrador : " +
          userCompleteObject.result.nombreGestor +
          ' ' +
          userCompleteObject.result.apellidosGestor,
      context,
      duration: Toast.LENGTH_LONG,
      gravity: Toast.BOTTOM);

  //Esto para esperar en el cambio de pantalla
  await Future.delayed(const Duration(seconds: 2)); //recommend

  if (await canLaunch(telefono)) {
    await launch(telefono);
  } else {
    Toast.show(
      "Su dispositivo no puede realizar llamadas"
       );
  }
}
```

La última condición en el código (if canLaunch) permite comprobar si el dispositivo tiene método de realizar llamadas. Debido a que el desarrollo es multiplataforma y para el 99.5% de los dispositivos, es posible que se use en tablets que no tengan forma de realizar llamadas. En ese caso, esta función le muestra una burbuja al usuario que le indica que no puede realizar llamadas y no hace nada más.

5.3.4.5 Declaración de siniestros

La declaración de siniestros es el principal objetivo de la aplicación, al principio de la misma, se le pregunta al usuario por un título y una descripción del siniestro. En la descripción el usuario explica con la extensión que considere qué ha ocurrido.

En todos los siniestros hay que especificar la ubicación en la que ocurrió, por lo que en este formulario es lo siguiente que se pide. Los mapas se han implementado usando la librería de google maps para flutter[22] que permite usar la API de google maps de manera simple.

Ilustración 23: Declaración de siniestros

La api de Google Maps es un servicio cloud de pago [23], por lo que es necesario obtener una KEY que permite a Google cobrar por el servicio, y a IMeureka usarlo.

Este caso sirve para ilustrar que, aunque en la mayoría de las ocasiones la programación es igual para Android que para IOS, en algunos no. Las keys son diferentes en caso de que el dispositivo que se esté usando sea Android o IOS, se implementaron mapas para ambos sistemas operativos, por lo que se tienen dos keys diferentes.

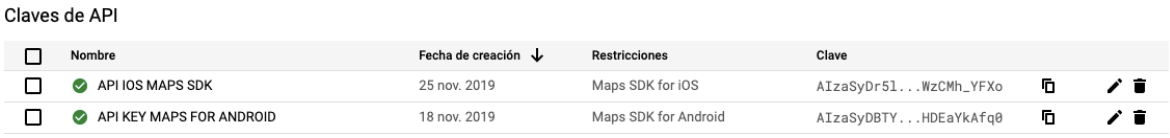


Ilustración 24: Keys google maps diferentes

Lógicamente no se muestran las keys completas.

La keys se incluyen de maneras diferentes en función del sistema operativo, por lo que hay que tocar el código autogenerado por Flutter. Para Android, hay que incluir en un archivo denominado manifest.xml …

```
      <meta-data android:name="com.google.android.geo.API_KEY"
   android:value="AIzaSynoenseñarkeysprivadasYkAfq0"/>
```

Para IOS se pone en el archivo AppDelegate.m

```
      GMSServices.provideAPIKey("AIzaSyDnoleasestofnneWzCMh_YFXo")
```

Las keys mostradas no son reales.

Continuando la declaración del siniestro, el usuario tiene dos opciones, seleccionar la ubicación actual (útil en los casos en los que se declara un siniestro en el acto) o seleccionar cualquier ubicación.

En ambos casos, al seleccionar una ubicación, mediante la API geocoding de google, se transforman las parejas de latitud/longitud automáticamente en la dirección que hay que escribir en la parte siguiente del formulario.

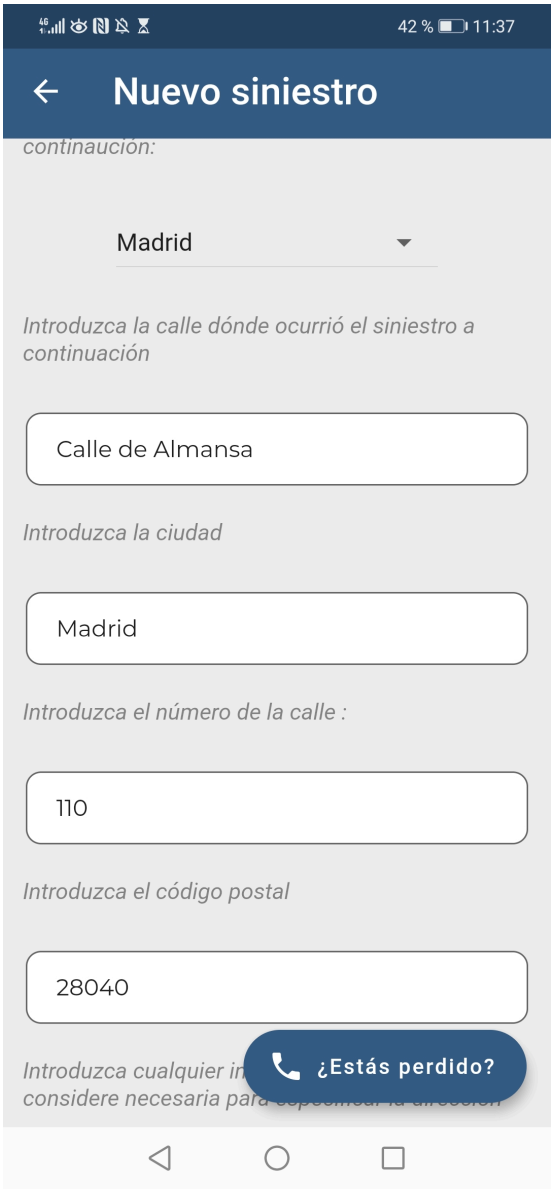


Ilustración 25: Declaración de siniestros - ubicación

Geocoding inverso [24] es el proceso por el cual se convierten coordenadas (37.43,-122.2) a una dirección concreta. Es implementado a través del paquete de geocoding para Dart [25]

Para prevenir que la ubicación actual sea claramente errónea, se hace una pequeña comprobación. Todos los siniestros deben haber ocurrido en España, por lo que el siniestro la provincia debe pertenecer al array de provincias.

```
      //Al pasarle a la api de geocoding a través del método
placemarkFromCoordinates una latitud y longitud, responde con la dirección del
lugar.
      List<Placemark> placemark = await
placemarkFromCoordinates(center.latitude, center.longitude);
      //Extraemos los datos, código postal ciudad…
direccionSiniestro.text = placemark[0].thoroughfare;
numeroSiniestro.text = placemark[0].subThoroughfare;
codigoPostal.text = placemark[0].postalCode;
ciudad.text = placemark[0].locality;

//Es necesario algo de lógica para comprobar si la provincia que saca el geocoder
se encuentra en el array de provincias

for (var i = 0; i < provincias.length; i++) {
  if (provincias[i] == placemark[0].subAdministrativeArea) {
    province_id = i;
    debugPrint("Provincia encontrada!! es : " + provincias[i]);
    break;
  } else {
    debugPrint("Provincia NO encontrada!! es : " + provincias[i]);
  }
}
```

Al pulsar sobre la ubicación, se rellena automáticamente. El usuario siempre tiene la opción de cambiar la ubicación.

Si por el contrario decide no usar este servicio, tendrá que rellenar en el formulario la localización a mano.

Ilustración 26: Declaración de siniestros – ubicación vacía

Otro dato clave es la fecha y hora en la que ocurrió el siniestro, para ello, se ha utilizado el paquete Material [26] que incluye utilidades de botones, entre ellos un selector de fechas:

```
      Future<Null> _selectDate(BuildContext context) async {
  final DateTime picked = await showDatePicker(
      context: context,
      initialDate: selectedDate,
      firstDate: DateTime(2000, 8),
      lastDate: DateTime.now());
  if (picked != null && picked != selectedDate)
    setState(() {
      //Date que se manda a base de datos
      selectedDate = picked;

      var newFormat = DateFormat("yy-MM-dd");
      fecha_mostrada = newFormat.format(picked);
    });
}
```

La fecha inicial que se puede seleccionar no importa, se ha seleccionado el año 2000 por ejemplo. En cambio, la última fecha que aparece en el selector sí es clave, y es la fecha actual (por mucho que el usuario sea previsor, no se le permite declarar un siniestro a futuro).

Para la hora el procedimiento es similar:

```
      final TimeOfDay selectedTime24Hour = await showTimePicker(
  context: context,
  initialTime: TimeOfDay(hour: 10, minute: 47),
  builder: (BuildContext context, Widget child) {
    return MediaQuery(
      data: MediaQuery.of(context).copyWith(alwaysUse24HourFormat: true),
      child: child,
    );
  },
);
```

El calendario del DateTimePicker es así visualmente:

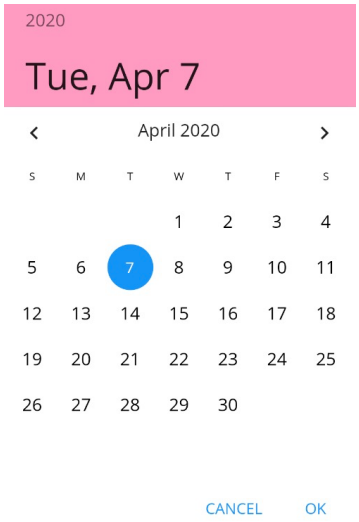


Ilustración 27: Selector de fechas

Tras esto se entra en la parte de la subida de archivos, en la cuál el usuario puede seleccionarlos del almacenamiento interno de su dispositivo.

Ilustración 28: Selección de archivos

Al seleccionar un archivo, se puede verlo tocando sobre el nombre. De esta manera puede comprobar que todo es correcto. Tras rellenar todos estos datos, el siniestro es subido.

Las opciones que se ven en el menú seleccionable son las siguientes:

```
      items: <DropdownMenuItem>[
  new DropdownMenuItem(
    child: new Text('Archivo de audio'),
    value: FileType.audio,
  ),
  new DropdownMenuItem(
    child: new Text('Imagen'),
    value: FileType.image,
  ),
  new DropdownMenuItem(
    child: new Text('Video'),
    value: FileType.video,
  ),
  new DropdownMenuItem(
    child: new Text('Ver todos & PDF'),
    value: FileType.any,
  ),
],
```

Principalmente se le permite escoger entre audio (si quiere describir el siniestro con voz, imágenes, vídeos y finalmente cualquier archivo que considere relevante). Hay que recordar que los archivos que no se permiten subir son bloqueados a nivel de backend.

Esto se ha implementado con el paquete file_picker [27] de flutter, que permite usar el sistema explorador de archivos nativo del dispositivo para seleccionar uno o múltiples elementos, con soporte para extensiones.

Para abrir los archivos al tocarlos en la imagen anterior, se ha usado el paquete open_file [28], que permite usar los visores de archivos nativos para visualizar los mismos, su uso es muy sencillo conociendo el path:

```
      OpenFile.open(path),
```

Capítulo 6. ANÁLISIS DE RESULTADOS

El objetivo principal de diseñar una aplicación multiplataforma e integrarla en el back-end de IMeureka se ha cumplido. Inclusive, la aplicación se encuentra subida en los marketplaces de apps de IOS y Android [29]

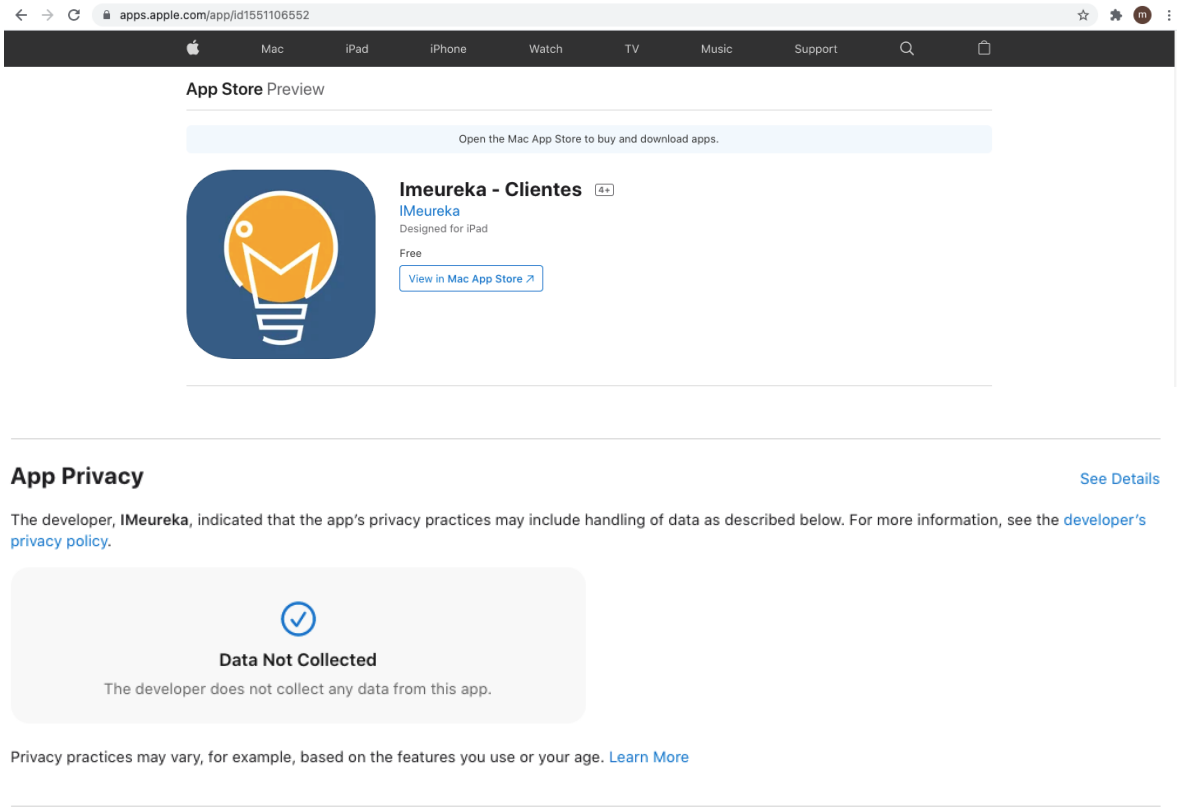


Ilustración 29: IMeureka – Clientes en app store

La app cuenta con los requisitos para ser subida a ambos marketplaces.

Se ha conseguido, por los motivos explicados anteriormente que la app sea sencilla y auto explicativa. Además, en caso de que el usuario tenga problemas por alguna peculiaridad de su caso, siempre puede llamar e inmediatamente se le pone en contacto con la persona a cargo.

Las funcionalidades básicas de declaración de siniestros, localización, archivos del siniestro, fecha y hora, título y descripción se han cumplido. Los siniestros quedan almacenados en la base de datos de IMeureka y se pueden visualizar tanto las empresas, como las pólizas, como los siniestros asociados.

Se han implementado de manera general los métodos de las API que se especificaron como objetivos, y la comunicación entre API y app es mediante JSON.

Queda pendiente como objetivos ver si la app realmente agiliza el proceso de declaración de siniestros cómo se indicó (máximo 15 días, importante reducción en el número de emails). Por desgracia, aún no hay clientes que nos permitan sacar estadísticas en este aspecto.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Hay mucho margen de mejora en la digitalización del sector asegurador, ya que es sorprendente que una app como esta sea "innovación" cuando debiera ser algo básico. La falta de uniformidad entre las compañías causa este desajuste que causa que cada compañía haga un proceso uniforme (la declaración del siniestro) de manera distinta.

La app aún puede ser mejorada con el siguiente trabajo a corto plazo:

Implementar flutter_secure_storage: De manera inmediata se debe focalizar en la securización de las contraseñas de los usuarios. Será parcheado en la próxima versión que se suba a los marketplaces.

Email de siniestro declarado: Actualmente cuando el usuario declara su siniestro no recibe confirmación de que está todo correcto… Sería una buena idea enviar por mail un recordatorio de que el siniestro se ha declarado correctamente, para que estén tranquilos de que todo ha ido correctamente y lo tengan por escrito

Enfocándonos en el trabajo a largo plazo, aparecen varias líneas de trabajo:

Contrastar la hipótesis de que esta herramienta efectivamente reduce la carga de gestión. Al no haber sido probada aún por suficientes usuarios para hacer una estadística, no se puede saber.

Implementar más funcionalidad para otros players del ecosistema: Se verá si en esta misma aplicación u otra… Hacer apps para aseguradoras, corredurías … Aunque es necesario el visto bueno de negocio.

Capítulo 8. BIBLIOGRAFÍA

[1] Imeureka.com. 2021. *¿Qué es IMeureka? | Imeureka.com*. [online] Available at: <https://www.imeureka.com/quienes-somos>.

[2] Imeureka.com. 2021. *¿Qué es IMeureka? | Imeureka.com*. [online] Available at: <https://www.imeureka.com/quienes-somos>.

[3] Flutter.dev. 2021. *Install*. [online] Available at: <https://flutter.dev/docs/get-started/install>.

[4] Postman.com. 2021. [online] Available at: <https://www.postman.com/> [Accessed 16 June 2021].

[5] Es.wikipedia.org. 2021. *GPS - Wikipedia, la enciclopedia libre*. [online] Available at: <https://es.wikipedia.org/wiki/GPS> [Accessed 16 June 2021].

[6] Es.wikipedia.org. 2021. *JSON - Wikipedia, la enciclopedia libre*. [online] Available at: <https://es.wikipedia.org/wiki/JSON#:~:text=JSON%20(acr%C3%B3nimo%20de%20JavaScript%20Object,para%20el%20intercambio%20de%20datos.&text=Una%20de%20las%20supuestas%20ventajas,sint%C3%A1ctico%20(parser)%20para%20%C3%A9l.> [Accessed 16 June 2021].

[7] Git-scm.com. 2021. *Git*. [online] Available at: <https://git-scm.com/> [Accessed 16 June 2021].

[8] Bitbucket. 2021. *Bitbucket | The Git solution for professional teams*. [online] Available at: <https://bitbucket.org/product/> [Accessed 16 June 2021].

[9] Jetbrains.com. 2021. *PhpStorm: el IDE rápido e inteligente para programación en PHP de JetBrains*. [online] Available at: <https://www.jetbrains.com/es-es/phpstorm/> [Accessed 16 June 2021].

[10] Es.wikipedia.org. 2021. *XAMPP - Wikipedia, la enciclopedia libre*. [online] Available at: <https://es.wikipedia.org/wiki/XAMPP> [Accessed 16 June 2021].

[11] StatCounter Global Stats. 2021. *Mobile Operating System Market Share Worldwide | StatCounter Global Stats*. [online] Available at: <https://gs.statcounter.com/os-market-share/mobile/worldwide> [Accessed 16 June 2021].

[12] Statista. 2021. *Aseguradoras privadas operativas en España 2011-2019 | Statista*. [online] Available at: <https://es.statista.com/estadisticas/526728/numero-de-companias-de-seguros-en-espana/> [Accessed 16 June 2021].

[13] Análisis del mercado asegurador por mapfre https://documentacion.fundacionmapfre.org/documentacion/publico/i18n/catalogo_imagenes/grupo.cmd?path=1099983

[14] Es.wikipedia.org. 2021. *Índice de Herfindahl - Wikipedia, la enciclopedia libre*. [online] Available at: <https://es.wikipedia.org/wiki/%C3%8Dndice_de_Herfindahl#:~:text=El%20%C3%8Dndice%20de%20Herfindahl%20o,muy%20concentrado%20y%20poco%20competitivo.> [Accessed 16 June 2021].

[15] Carlos del Castillo, F., 2021. *Madrid deja al descubierto datos personales de cientos de miles de pacientes con su sistema de autocita para la vacuna*. [online] ElDiario.es. Available at: <https://www.eldiario.es/madrid/madrid-deja-descubierto-datos-personales-pacientes-sistema-autocita-vacunacion_1_8013044.html> [Accessed 16 June 2021].

[16] Mysql.com. 2021. *MySQL :: MySQL Workbench*. [online] Available at: <https://www.mysql.com/products/workbench/> [Accessed 16 June 2021].

[17] Perito Judicial GROUP©. 2021. *El Gabinete Pericial, el mejor aliado en el juicio ⚖️?⚖️*. [online] Available at: <https://peritojudicial.com/gabinete-pericial/#:~:text=Un%20gabinete%20pericial%20es%20un,dar%20servicios%20de%20peritaciones%20judiciales.> [Accessed 16 June 2021].

[18] Dart packages. 2021. *shared_preferences | Flutter Package*. [online] Available at: <https://pub.dev/packages/shared_preferences> [Accessed 16 June 2021].

[19] Dart packages. 2021. *flutter_secure_storage | Flutter Package*. [online] Available at: <https://pub.dev/packages/flutter_secure_storage> [Accessed 16 June 2021].

[20] datas?, I., Kazaman, A., Żygłowicz, K. and Amiya, I., 2021. *Is Shared Preferences safe for private datas?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/34994807/is-shared-preferences-safe-for-private-datas> [Accessed 16 June 2021].

[21] Dart packages. 2021. *url_launcher | Flutter Package*. [online] Available at: <https://pub.dev/packages/url_launcher>

[22] Dart packages. 2021. google_maps_flutter | *Flutter Package*. [online] Available at: <https://pub.dev/packages/google_maps_flutter >

[23] Google Cloud. 2021. *Geo-location APIs | Google Maps Platform | Google Cloud*. [online] Available at: <https://cloud.google.com/maps-platform/.> [Accessed 16 June 2021].

[24] Google Developers. 2021. *Overview | Geocoding API | Google Developers*. [online] Available at: <https://developers.google.com/maps/documentation/geocoding/overview> [Accessed 16 June 2021].

[25] Dart packages. 2021. geocoding | *Flutter Package*. [online] Available at: <https://pub.dev/packages/geocoding >

[26] Dart packages. 2021. flutter_datetime_picker | *Flutter Package*. [online] Available at: <https://pub.dev/packages/flutter_datetime_picker >

[27] Dart packages. 2021. file_picker | *Flutter Package*. [online] Available at: <https://pub.dev/packages/ file_picker >

[28] Dart packages. 2021. open_file | *Flutter Package*. [online] Available at: <https://pub.dev/packages/ open_file >

[29] App Store. 2021. *Imeureka - Clientes*. [online] Available at: <https://apps.apple.com/app/id1551106552> [Accessed 16 June 2021].

ANEXO ODS

La app ayuda en el ODS 8 para el trabajo decente y el crecimiento económico. La parte de crecimiento económico mediante el aumento de eficiencias que aumentan los beneficios de todos los usuarios implicados, y el trabajo decente mediante la disminución de trabajo repetitivo que tienen que realizar los intermediadores, aumentando su rentabilidad y su calidad de vida.

También colabora en el apartado 9 con la innovación, ya que es puntera en muchos ámbitos del sector asegurador.

Finalmente, también contribuye con el ODS 15 ya que al digitalizar el sector asegurador se disminuye el uso de papel, protegiendo así los bosques del plantea.