UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

APLICACIÓN PARA TABLETS QUE PERMITA A LOS COMERCIALES DE UNA COMPAÑÍA DE CRÉDITO LA GESTIÓN DEL PROCESO DE NEGOCIO POR EL QUE LOS COMERCIOS SE ADHIEREN AL SISTEMA QUE ACEPTA LOS MEDIOS DE PAGO DE DICHA COMPAÑÍA

Álvaro Pérez Alonso
Tutor: José Antonio Clavijo Blázquez
Ponente: Simone Santini

JUNIO DE 2019

APLICACIÓN PARA TABLETS QUE PERMITA A LOS COMERCIALES DE UNA COMPAÑÍA DE CRÉDITO LA GESTIÓN DEL PROCESO DE NEGOCIO POR EL QUE LOS COMERCIOS SE ADHIEREN AL SISTEMA QUE ACEPTA LOS MEDIOS DE PAGO DE DICHA COMPAÑÍA

AUTOR: Álvaro Pérez Alonso
TUTOR: José Antonio Clavijo Blázquez
PONENTE: Simone Santini

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2019

Resumen

El avance de las aplicaciones móviles hoy en día es imparable, desplazando a muchas aplicaciones web y provocando que muchas compañías desarrollen sus propios servicios para esta plataforma con el objetivo de acercar su producto al usuario de una manera más cómoda y rápida.
Este es el caso desarrollado en este Trabajo de Fin de Grado en el que un cliente, en representación de una compañía de crédito, ha propuesto a Delonia Software SL cambiar el método de adhesión de diferentes comercios al sistema que acepta el medio de pago de dicha compañía.
La solución propuesta y descrita en este documento se basa en la realización de una aplicación para *tablets* Android que unificará y mejorará el actual sistema que el cliente tiene para la incorporación de comercios a su red.
Esta aplicación contará con un formulario en el que los agentes comerciales de la compañía rellenarán la información del cliente que se va a unir al sistema. Además, facilita escanear documentos a través de la cámara de la *tablet*, adjuntar la firma del cliente con el panel táctil de la propia *tablet* e incluir otros archivos desde el buzón de correo del agente. El agente podrá crear solicitudes de adhesión de una manera más simple y minimizando el tiempo necesario para incorporar a los clientes. Todo esto garantizando la confidencialidad y seguridad de sus datos.
En este proyecto se pretende mejorar y dar un nuevo enfoque, tanto tecnológico como comercial, al trabajo anteriormente realizado por un compañero de la misma compañía con la que he realizado este trabajo.
En esta memoria se describe todo el proceso seguido para la creación de la aplicación desde el análisis y diseño inicial hasta el desarrollo completo y prueba de la solución entregada. Este proyecto no solo abarca la aplicación en sí, sino que incluye la construcción del servidor y base de datos necesarias para el correcto funcionamiento de la aplicación.
Un ingeniero y arquitecto software ha supervisado todo el ciclo de vida del proyecto para garantizar la calidad de la solución final y que ha sido entregada al cliente.

Palabras clave

Aplicación, móvil, *tablet*, Android, .Net, Xamarin.Forms, C#, XAML, software, MVVM, *binder*, agente, comercio, administrador, base de datos, interfaz, servidor, front-end, back-end.

Abstract

The advance of mobile aplication today is unstoppable, displacing many web applications and causing many companies to develop their own services for this platform in order to bring their product to the user in a more convenient and faster way.

This is the case developed in this End-of-Degree Project in which a client, on behalf of a credit company, has proposed to Delonia Software SL change the method of accession of different businesses to the system that accepts the means of payment of said company.

The solution proposed and described in this document is based on the realization of an application for Android tablets that will unify and improve the current system the client has for the incorporation of clients to their network.

This application will have a form and in which the commercial agents of the company will fill with the information of the client that is going to join the system. In addition, it facilitates scanning documents through the camera of the tablet, attaching the signature of the client with the touch panel of the tablet itself and including other files from the agent's mailbox. The agent can create adhesion requests in a simpler way and minimizing the time necessary to incorporate the clients. All this guaranteeing the confidentiality and security of your data.

This project aims to improve and give a new focus, both technological and commercial, to the previous work carried out by a colleague of the same company with whom I have done this work.

This report describes the entire process followed for the creation of the application from the initial analysis and design to the complete development and testing of the delivered solution. This project not only covers the application itself, but includes the construction of the server and database necessary for the proper functioning of the application.

An engineer and software architect has supervised the entire life cycle of the project to guarantee the quality of the final solution and that has been delivered to the client.

Keywords

Aplication, phone, tablet, Android, .Net, Xamarin.Forms, C#, XAML, software, MVVM, binder, agent, commerce, administrator, data base, interface, server, front-end, back-end.

Agradecimientos

A mi abuelo, por haberme dado todo. Por cuidarme y guiarme desde que nací. Y por sus "estudia y trabaja, que nadie te va a dar nada" que han logrado que llegue hasta aquí.

A mis padres, por su educación y apoyo durante todos estos años. Por haber confiado en mí y, sobre todo, por su infinita paciencia que han conseguido que llegue a ser la persona que soy ahora.

A mi hermano, con el que tantos buenos momentos de diversión he pasado y ha sido capaz de aguantarme durante tantos años. Por haberme confiado en mí y dado la oportunidad de enseñarte.

A mis compañeros del colegio, a los que debido a los estudios no nos vemos tanto, son mis primeras amistades y con los que he crecido y aprendido.

A mis amigos de Cantalojas, con los que empecé pasando los veranos y que ahora se han convertido en una familia. Con los que he pasado tantos buenos momentos que ya ni se pueden enumerar. Y, en especial, por todas esas noches de fiesta.

A mis compañeros de carrera, que durante estos años hemos sufrido tanto por estudios y prácticas, pero entre todos nos hemos apoyado para sacar todas las asignaturas adelante y siempre hemos encontrado un hueco para desconectar y vivir la vida universitaria.

A Delonia Software SL, por haberme dado la oportunidad de trabajar con ellos realizando este proyecto y acogerme como uno más dentro de su equipo.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1 Introducción

En este primer capítulo se hará una visión global del proyecto realizado, el plan de trabajo seguido durante el desarrollo del mismo y, por último, se explicará la estructura de esta memoria.

1.1 Motivación

Este nuevo proyecto tiene su base en otro proyecto realizado anteriormente, 1718_066_ISSITI, que fue realizado por Pablo Yus Domínguez "Aplicación móvil sobre Tablets para la correcta gestión y soporte del proceso de negocio de Solicitudes de Tarjetas Face to Face que se procesan en 'Hubs' de transporte".

La nueva aplicación usará parcialmente los componentes y arquitectura ya creados, pero se adaptarán para desarrollar nuevas funcionalidades que están apoyadas en servicios creados desde cero. De esta manera, se permitirá a la compañía encargada de la venta de tarjetas de crédito incorporar nuevos comercios a su red mediante los diferentes procesos que se desarrollarán en esta nueva aplicación.

La solución propuesta por Delonia consiste en un cambio tanto tecnológico como de proceso de negocio, pues más allá de la tecnología existen dificultades y desafíos en el funcionamiento actual de agentes que generan y gestionan las solicitudes en papel, así como de los procesos de negocio posteriores que se llevan a cabo en la compañía para la validación y proceso de estas solicitudes.

1.1.1 Motivación tecnológica

Actualmente el proceso de negocio de contratación es mayormente analógico, requiriendo un formulario en papel y un posterior proceso con herramientas intermedias como Excel, lo cual además de dificultar la agilidad y movilidad de agentes, supone una fuente de potenciales problemas a la hora de garantizar la privacidad de la información, seguridad de acceso y automatización / control de los datos.

Con estos datos, se propone un cambio orientado al uso de las nuevas tecnologías, simplificando aquellos procesos que actualmente pueden generar errores o inconsistencias.

1.1.2 Motivación de negocio

La solución que se propone intenta dar soporte tanto al actual modelo de negocio como las posibles futuras modificaciones o adaptaciones. Esta solución facilita todo el proceso de negocio y la vinculación Grupo Empresarial, Cliente, Agente y Productos reduciendo la necesidad de realizar modificaciones futuras cuando aparezca un nuevo cambio en los actuales procesos de venta, redes comerciales o gestión de clientes y grupos de empresas de la compañía.

Otro de los aspectos existentes en el proceso de simplificación y normalización del actual sistema técnico de los agentes en la venta (gestión de contratos): el modelo actual requiere de una serie de elementos (papel, equipo PC, Escáner para digitalizar…) que

aumenta la inconsistencia del sistema ante posibles problemas en la infraestructura, en la pérdida de información física, en la dependencia de personas físicas, etc.

La solución propuesta es capaz de simplificar en un solo elemento el formulario, la captura de imágenes y la recogida de firmas gracias al uso de *tablets*.

1.1.3 Motivación por normativas

Como ya se ha indicado anteriormente, una de las mayores motivaciones que planteaba el desarrollo de esta solución es que el actual modelo de negocio de la compañía no cumplía todas las normativas de privacidad y seguridad de la información. Un punto crucial de potencial incumplimiento es debido que los diferentes datos del cliente (formulario, DNI digitalizado, firma u otros documentos) están tanto en papel como alojadas en equipos informáticos de personas, a veces en disposición de agencia y por tanto con posibles usos no homologados.

La solución basada en *tablet* nos permite controlar esta información. Para ello, se creará un entorno seguro que será gestionado por la nueva aplicación que se desarrolle. De esta forma, toda la información de los clientes quedará clasificada correctamente en el dispositivo *tablet*, enviada al servidor y finalmente eliminada evitando así que pueda ser extraída para otros usos.

Al no tener que usar ningún otro dispositivo, nos permite lograr el aislamiento de información que implica un almacenamiento seguro de todos los datos y todo ello mejorado con las diferentes medidas de seguridad y cifrado que tiene la propia tablet. Por último, sería posible separar los datos de los agentes y de la agencia (información no relacionada con el cliente) de los datos críticos y que son propiedad de la compañía (información relacionada con el cliente) usando el dispositivo portátil.

1.2 Objetivos y enfoque

El objetivo final de este proyecto es el de resolver una necesidad existente desarrollando una nueva aplicación para *tablets* Android. Esta solución contará con todo el análisis, diseño, desarrollo y pruebas tanto de la aplicación en sí, como de la base de datos y el servidor que se han sido necesario desarrollar para lograr el objetivo final.

Con esta solución se pretende dar un enfoque mejor al proceso de creación de solicitudes por parte de una compañía, unificándolo y haciéndolo seguro de acuerdo con la normativa actual. El desarrollo de esta aplicación para *tablets* facilitará el trabajo de los agentes dentro de la compañía y reducirá el tiempo actual en la creación de solicitudes.

1.3 Organización de la memoria

La memoria presentada se ha dividido en una serie de capítulos de acuerdo con las diferentes etapas que presenta un proyecto software. Se describen a continuación:

En el *Capítulo 1* se hace una explicación general del proyecto. Motivación que ha llevado a la realización del mismo y como se ha estructurado.

En el *Capítulo 2* se presentan las últimas tecnologías y conocimientos de las aplicaciones móviles, así como las diferentes tecnologías existentes para su desarrollo.

En el *Capítulo 4* se presenta la fase inicial del proyecto. En este capítulo se incluye el alcance que va a tener el proyecto y el listado completo de requisitos que debe cumplir la aplicación.

En el *Capítulo 5* se describe el diseño y la arquitectura de los diferentes componentes del proyecto.

En el *Capítulo 6* se explica todo el proceso de desarrollo de la aplicación. Las tecnologías usadas y las herramientas que se han usado.

En el *Capítulo 7* se presenta la fase final del proyecto en el que se comprueba que la aplicación satisface todas las necesidades del cliente.

En el *Capítulo 8* hace un resumen final del proyecto y se plantean posibles trabajos futuros relacionados con el mismo.

Tras los capítulos, se incluyen las referencias bibliográficas usadas para la realización del proyecto. La memoria se completa con dos anexos. El primero incluyen los diagramas usados en la base de datos y las maquetas de la aplicación. En el segundo, un manual del programador con el código de los elementos más importantes de la aplicación y el servidor.

2 Estado del arte

2.1 Introducción

En este capítulo se dará una visión global de la situación actual de los dispositivos móviles, sus sistemas operativos y los diferentes tipos de aplicaciones que se pueden crear.

Con este apartado, se pretende explicar los conocimientos que existen en la materia que estamos trabajando y explicar porque se ha decidido usar unos u otros.

2.2 Sistemas Operativos móviles

Un sistema operativo o SO es un software encargado de gestionar los procesos y recursos del sistema en el que está alojado. Esto incluye tanto las aplicaciones e instrucciones como el hardware. Sin un SO no podría funcionar ningún dispositivo [1].

Si nos centramos en los dispositivos móvil, los dos sistemas operativos más utilizados son Android y iOS. Estos dos SSOO son relativamente recientes y antes de su llegada los sistemas que dominaban el mercado eran Symbian y, en menor medida, Windows Mobile.

Symbian

Nacido en 1997, Symbian era el SO rey de la época. El sistema destacaba por su peculiar escritorio que contenía todas las aplicaciones, el uso del GPS y la creación de multitareas para saltar fácilmente entre las aplicaciones [2].

Android

Android es el sistema operativo desarrollado inicialmente por Android Inc. pero fue comprada por Google en 2005. Está basado en el Kernel de Linux y actualmente es el SO móvil más usado del mundo. La primera versión de Android sale a la luz a finales de 2007 pero no fue hasta el año siguiente cuando apareció el primer móvil que incluía el SO.

Desde entonces ha ido evolucionando y ha dado el salto a nuevas plataformas. Es un sistema operativo con una gran comunidad y el cual prácticamente todas las compañías de desarrollo de móviles usan [3, 4].

iOS

iOS es el segundo SO más usado del mundo desarrollado por Apple Inc. y usado únicamente en sus dispositivos. Este sistema operativo no llega a alcanzar las cuotas de Android debido al elevado coste de los dispositivos y a que la mayoría de las aplicaciones disponibles son de pago. A cambio, tenemos un sistema operativo más seguro que Android y con más actualizaciones periódicas para mejorarlo [5].

2.3 Aplicaciones móviles

Los móviles actualmente son indispensables en nuestra vida diaria, esto se traduce en más de 5.000 millos de dispositivos en el mundo [6]. Por esta razón, todas las compañías crean sus propias aplicaciones móviles para usar sus servicios.

Las aplicaciones móviles se han convertido en las reinas del consumo móvil. Los usuarios gastan alrededor del 80% de su tiempo con el móvil usando alguna de ellas.

Con el auge de las aplicaciones móviles ha surgido una nueva forma de venta online y en la que el móvil ha desplazado en ventas al ordenador. Cerca del 60% de las ventas online en España ya se realizan desde un dispositivo móvil por lo que estar bien posicionado dentro de este sector es una tarea muy importante para todas las compañías [7].

Los usuarios pueden acceder a estas aplicaciones a través de las tiendas que crean los desarrolladores de sistemas operativos u otras compañías. Actualmente, las más usadas son Google Play de Android y AppleStore de Apple.

Las aplicaciones subidas a estas tiendas son desarrolladas usando diferentes herramientas o implementaciones. La elección de unos u otros depende de la función que vaya a realizar la aplicación móvil. Las dos formas más destacadas de crearlas se describen a continuación:

2.3.1 Aplicaciones móviles nativas

Las aplicaciones móviles nativas son desarrolladas específicamente para un sistema operativo, Android o iOS. Si se quiere que la aplicación funcione en ambos SO será necesario crear dos aplicaciones. Una escrita en Java o Kotlin para Android y la otra escrita en Objetive-C o Swift para iOS de manera que el código no puede ser reutilizado entre uno y otro. Realizar aplicaciones nativas es más costoso de desarrollar y mantener.

El hecho de crear dos aplicaciones diferentes nos permite usar por completo todas las funcionalidades que traen los SO presentando un mejor rendimiento y experiencia de usuario [8].

2.3.2 Aplicaciones móviles híbridas

Como ya hemos visto, las aplicaciones móviles nativas tienen un gran problema, es necesario crear una para cada sistema operativo.

El desarrollo de aplicaciones *cross-platform* permite desarrollar con un mismo lenguaje una aplicación para todos los SO. Este código base se traduce automáticamente al código nativo logrando un rendimiento casi similar al de las aplicaciones nativas. El uso de las aplicaciones *cross-platform* está muy extendido ya que permite abaratar costes de un proyecto y reduce el tiempo necesario para desarrollar la aplicación [9].

Por estas razones, la aplicación tablet que se ha desarrollado se ha realizado con esta tecnología ya que en un futuro se pretende dar el salto a iOS.

Existen varias herramientas para el desarrollo de aplicaciones *cross-platform* descritas a continuación:

Appcelerator Titanium

Titanium utiliza el lenguaje de JavaScript para desarrollar las aplicaciones, propio de páginas web pero que se traduce a lenguaje nativo cuando se compila en el dispositivo.

La interfaz de programación o IDE está basada en Eclipse y garantiza un entorno fácil, intuitivo y visual para facilitar la programación.

Está en continuo desarrollo por lo que cada cierto tiempo aparecen nuevos recursos y posibilidades para desarrollar [10].

React Native

A igual que Titanium, usa JavaScript para desarrollar sus aplicaciones móviles. Es un proyecto de código abierto creado inicialmente por Facebook.

La principal ventaja y que hace que se diferencie del resto es que es un lenguaje interpretado y permite visualizar los cambios que se hacen al instante. Tiene una curva de aprendizaje mala pero una vez aprendido cumple todas las propiedades que debe tener una aplicación cross-platform [11].

Xamarin

A diferencia de los anteriores, Xamarin se escribe en lenguaje C# antes de ser traducido al lenguaje nativo. Desarrollado por Microsoft, es muy popular dentro de los desarrolladores de .Net.

Xamarin permite crear aplicaciones nativas para Android e iOS pero también aplicaciones *cross-platform* a través del *framework* Xamarin.Forms. Para su desarrollo, se usa el IDE creado por Microsoft, Visual Studio [12].

Otra de las ventajas es que, si se desea desarrollar código nativo para implementar una funcionalidad específica del sistema operativo, se puede realizar e incluirlo fácilmente al código común.

Cuenta con una gran comunidad detrás que crea numerosas librerías y componentes que ayudan en la creación de aplicaciones.

3 Metodología y plan de trabajo

3.1 Introducción

Antes de comenzar a desarrollar la aplicación y sus componentes, se debe establecer el plan de trabajo que se va a seguir durante los diferentes meses que va a durar el proyecto. Para el desarrollo de este producto software se ha decidido usar la metodología ágil de Scrum y que es muy común en los proyectos de este tipo

3.2 Scrum

Scrum es un proceso que nos ayuda a seguir unas buenas prácticas que mejoran el trabajo en equipo para conseguir un mejor resultado al finalizar el proyecto. Scrum se basa en entregas parciales del proyecto final y de forma regular. De esta forma, Scrum está indicado para entornos con requisitos cambiantes o que están poco definidos.

Aunque Scrum está indicado para grupos más grandes, hemos decidido usarlo entre el jefe de proyecto asignado por Delonia Software SL y yo para garantizar el cumplimiento de todos los requisitos y los plazos propuestos por el cliente.

Scrum se ejecuta sobre el proyecto en periodos cortos (*sprints*) con una duración fija (para este proyecto se han usado ciclos de 1 semana). Al finalizar cada *sprint*, se debe poder entregar un incremento del proyecto final completamente funcional. Inmediatamente después de terminar un *sprint* comienza el siguiente.

La planificación de estos *sprint* o *sprint planning meeting* se realiza en dos partes, una selección de los requisitos de la aplicación y una estimación del esfuerzo para realizar los requisitos seleccionados. Cada miembro del equipo elegirá un requisito que actuará como tarea a realizar [13].

Durante el tiempo que dura el proyecto, se deben realizar reuniones diarias o *daily meeting*. Estas se realizan normalmente sobre un tablero de tareas dirigidas por el *Scrum Master* (jefe de proyecto) y cada miembro deberá contar al resto de sus compañeros la tarea que está realizando y los posibles problemas que le han surgido. De esta manera, todos los miembros buscarán una solución al problema surgido evitando así retrasos en la entrega final del producto.

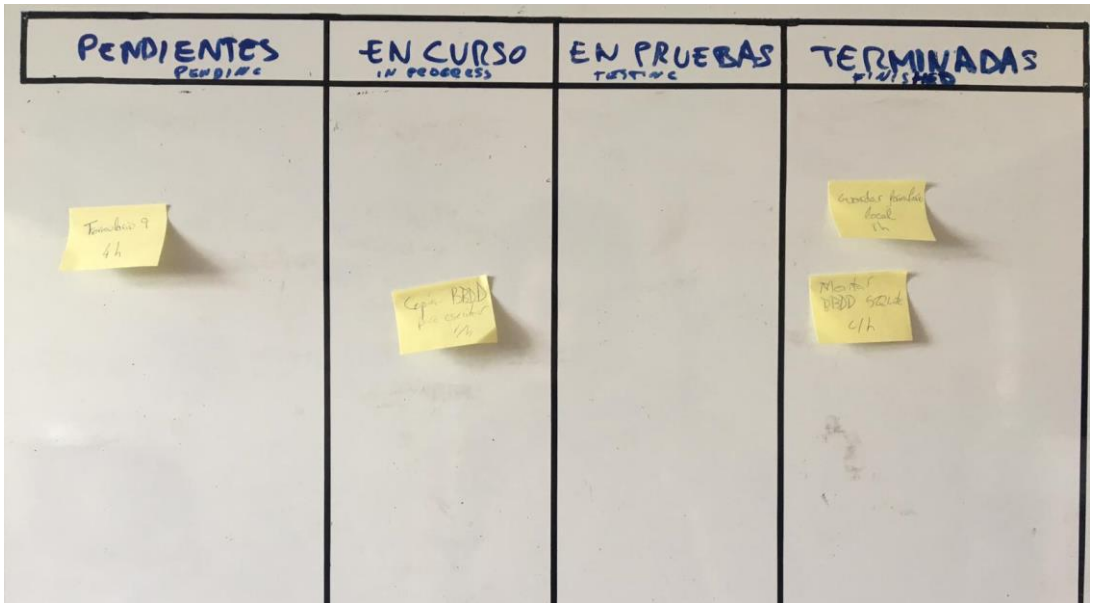

Figura 3-1: Tablero Scrum de tareas. En él se pueden ver las tareas pendientes, las que se están realizando, en pruebas y las ya finalizadas.
Fuente: Propia

Al finalizar cada sprint, se realiza una revisión o *sprint review* para valorar el incremento realizado y ver si se han cumplido con todas las tareas. Se realiza de una de forma informal con el objetivo de buscar mejoras en la forma de trabajar y soluciones a los problemas encontrados y que no se han podido resolver durante el *sprint*.

Junto con el tablero de tareas, se ha ido realizando un diagrama de quemado que consiste en la representación gráfica de las tareas que quedan por hacer (estimación en horas) en el tiempo.

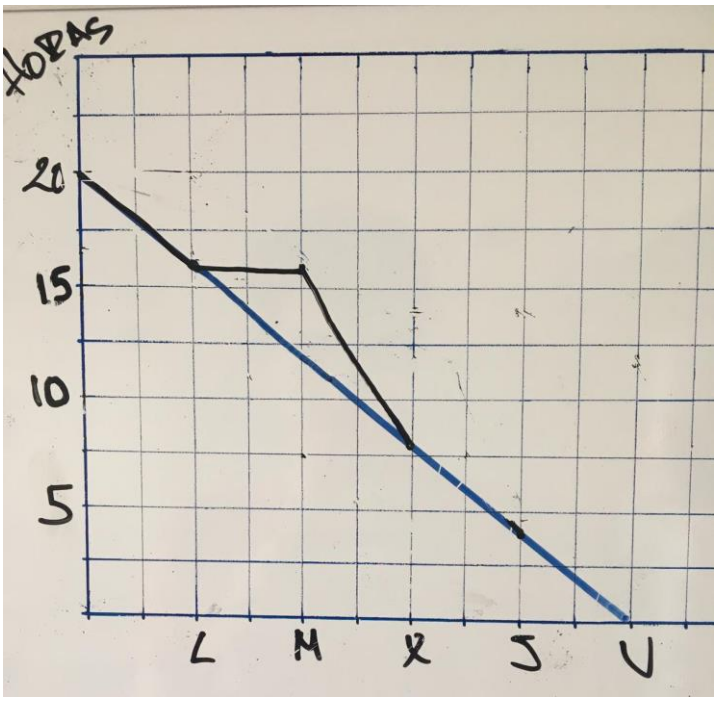

Figura 3-2: Diagrama de quemado. La gráfica corresponde con la misma semana del tablero de la figura anterior.
Fuente: Propia

3.2.1 Roles dentro del equipo Scrum

Dentro del equipo Scrum se pueden definir varios roles que cumplen diferentes funciones dentro del proyecto [14].

Scrum *Master*

Persona encargada del proyecto y su gestión. Tiene que asegurarse que todo el equipo entienda su trabajo, quedando a disposición de este para cuando sea necesario. Además, debe asegurarse de que haya un ambiente laboral adecuado.

El jefe de proyecto ha sido el encargado de realizar esta función.

Equipo de Desarrollo

Encargados de desarrollar el proyecto. No existe jerarquía ya que se autogestionan. Este modelo de trabajo facilita la productividad, la flexibilidad y la creatividad.

Este ha sido mi rol dentro del proyecto.

Product Owner

Persona que actúa como la voz del cliente. Organiza todas las tareas que deben realizarse y la gestión de las características del producto final.

El jefe de proyecto también ha sido el encargado de realizar las tareas del *product owner*.

3.3 Control de Versiones

Se ha realizado también un control de versiones del proyecto y los diferentes documentos que lo componen a través del software Git. Con cada cambio realizado sobre los diferentes componentes del proyecto, se ha creado un *commit*. Con esta metodología hemos logrado tener un buen control de versiones que nos permitía retroceder cuando surgían problemas. El repositorio donde se ha guardado todo el proyecto ha sido proporcionado por Delonia Software.

Para la gestión de versiones, ramas y demás complementos se ha optado por la herramienta visual Sourcetree [15]. Con ella, se puede controlar todos los cambios que se han producido en los ficheros y prepararlos para subir al repositorio. Se puede preparar tanto el fichero entero como fragmentos del mismo fichero que funcionarán como diferentes versiones. Todo ello en una herramienta visual muy completa e intuitiva disponible para dispositivos Mac con el que se ha desarrollado este proyecto.

4 Análisis

4.1 Introducción

La primera etapa de un proyecto de ingeniería de software es el análisis. Se debe estudiar por completo el problema y establecer cuáles van a ser las especificaciones que va a tener el sistema software que va a resolver ese problema. Es importante tener una buena descripción del producto que se ha solicitado el cliente, así como sus requisitos para poder ajustar el producto final para evitar retrasos en la entrega o cambios en la aplicación una vez que ha comenzado el proyecto.

4.2 Alcance de la aplicación

La aplicación surge con el objetivo de satisfacer la necesidad de resolver las dificultades y desafíos que existen actualmente a la hora de crear y gestionar solicitudes. Esta nueva aplicación permite automatizar y simplificar el negocio de contratación que actualmente es mayormente analógico, y que requiere de un formulario en papel y un procesamiento a través de herramientas intermedias lo que supone una fuente potencial de problemas a la hora de garantizar la privacidad de la información, seguridad de acceso y la automatización y el control de los datos.

Esta solución fue planteada por Delonia al cliente y una vez que fue aceptada se comenzó con el diseño y desarrollo de la aplicación. Esta versión de la aplicación está orientada a dispositivos *tablets* Android. En un futuro también se planea una ampliación a dispositivos iOS.

4.3 Roles

Antes de comenzar a plantear los requisitos que debe cumplir la nueva aplicación se debe establecer los diferentes roles que van a tener los usuarios dentro del sistema:

- Agente: único usuario de la aplicación. Será el encargado de hablar con los representantes de los comercios para la creación de la solicitud.

- Representante del comercio: persona que actuará en nombre de la compañía y que dará la información necesaria al agente para la creación de las solicitudes.

- Administrador: encargado de validar todas las solicitudes que crean los agentes desde la aplicación.

4.4 Análisis de Requisitos

El análisis de requisitos en una etapa crucial tanto para el usuario como para el equipo de desarrollo. Unos requisitos ambiguos o incompletos pueden provocar grandes inconvenientes en futuras etapas del proyecto. Para evitar estos problemas es necesario

establecer una buena comunicación con el cliente de manera que se pueda comprender la totalidad del problema y su entorno [16].

4.4.1 Requisitos Funcionales

En los requisitos funcionales se define la forma en el que el sistema debe reaccionar ante una entrada determinada, así como los servicios que este debe proporcionar al usuario. Los requisitos funcionales requeridos para la aplicación se enumeran a continuación:

RF1. Un agente podrá iniciar sesión en la aplicación a través de un usuario y una contraseña.

RF2. Un agente podrá cambiar la contraseña de inicio de sesión cuando lo desee.

RF3. Un agente podrá cerrar sesión cuando lo desee.

RF4. Un agente podrá visualizar su nombre y foto de usuario.

RF5. Un agente podrá iniciar una nueva solicitud.

RF6. Un agente podrá finalizar y enviar una solicitud.

RF7. Un agente podrá visualizar una solicitud ya finalizada.

RF8. Un agente podrá guardar una solicitud para modificarla más adelante.

RF9. Un agente podrá guardar una solicitud en el dispositivo cuando no disponga de conexión a internet.

RF10. Un agente podrá enviar las solicitudes guardadas en el dispositivo cuando vuelva a disponer de conexión a internet.

RF11. Un agente podrá cancelar una solicitud antes de que esta se haya finalizado o guardado para más adelante.

RF12. Un agente podrá ver una lista con todas las solicitudes tanto finalizadas o no finalizadas como guardadas en el dispositivo o en el servidor remoto.

RF13. Un agente podrá filtrar la lista de solicitudes por finalizadas o no finalizadas.

RF14. Un agente podrá filtrar la lista de solicitudes por guardadas en el dispositivo o guardadas en el servidor remoto.

RF15. Un agente podrá buscar una solicitud a través del nombre de la compañía.

RF16. Un agente podrá modificar una solicitud previamente creada.

RF17. Un agente podrá borrar una solicitud solo cuando esté guardada en el dispositivo.

RF18. Un agente podrá hacer una foto a un documento, por ejemplo, un DNI e incluirlo como archivo adjunto a una solicitud.

RF19. Un agente podrá incluir imágenes adjuntas que se encuentren en la galería del teléfono a una solicitud.

RF20. Un agente podrá adjuntar la firma del representante del comercio (captura desde pantalla) e incluirla en una solicitud.

RF21. Un agente podrá iniciar sesión en su correo electrónico.

RF22. Un agente podrá visualizar la bandeja de entrada de su correo electrónico.

RF23. Un agente podrá ver tanto el mensaje de un correo electrónico como los archivos adjuntos incluidos en él.

RF24. Un agente podrá incluir los archivos adjuntos que haya recibido por correo electrónico a una solicitud.

RF25. Un agente podrá eliminar archivos adjuntos incluidos previamente en una solicitud.

RF26. Un agente podrá cambiar el nombre de un archivo adjunto incluido en una solicitud.

4.4.2 Requisitos No Funcionales

A diferencia de los funcionales, estos requisitos describen las características del funcionamiento y restricciones del sistema, tales como la seguridad, la usabilidad o la eficiencia.

A continuación, se muestra la lista de todos los requisitos no funcionales de la aplicación:

RNF1: La aplicación deberá ajustarse a cualquier resolución de pantalla.

RNF2: La aplicación podrá ser ejecutada en cualquier Tablet Android siempre y cuando tengan una versión Android 6.0 (versión de API 23) o superior.

RNF3: La aplicación deberá guardar las credenciales de la última sesión de un agente de manera que este no tenga que volver a iniciar sesión la próxima vez que use la aplicación.

RNF4: La aplicación almacenará de forma segura las credenciales de un agente a través de un cifrado AES.

RNF5: La aplicación tendrá una base de datos para guardar solicitudes cuando la *tablet* no disponga de conexión a internet.

RNF6: La aplicación deberá actualizar automáticamente el nombre de un agente y/o su foto cuando estos hayan sido modificado desde el servidor web.

RNF7: La aplicación enviará un informe al servidor con las acciones que ha realizado un agente (iniciar sesión, crear solicitud, cerrar sesión…) y la información del dispositivo con el cual se han realizado dichas acciones.

RNF8: Todas las comunicaciones que se realicen con el servidor deberán estar verificadas y firmadas.

RNF9: La aplicación deberá mostrar un *spinner* de carga cuando esta demore demasiado tiempo en mostrar una pantalla.

RNF10: La aplicación no podrá quedarse bloqueada a la espera de la respuesta de un servicio (a no ser que sea estrictamente necesario). Las conexiones, por tanto, deberán ser asíncronas.

RNF11: La solución final deberá estar comentada.

5 Diseño y Arquitectura

5.1 Introducción

Una vez definido los requisitos que debe cumplir la aplicación comienza el diseño de la aplicación. En esta etapa no solo debemos diseñar la interfaz gráfica que va a tener la aplicación, sino que tenemos que decidir la arquitectura que va a seguir la capa de presentación o *front-end*, el servidor o *back-end*, los servicios para conectarnos al servidor y la base de datos.

También se hablará de las diferentes herramientas que se han usado para la realización de todos estos diseños.

5.2 Patrón de Arquitectura

Para diseñar la aplicación se ha usado el patrón Modelo-Vista-VistaModelo o MVVM y que se caracteriza principalmente por desacoplar al máximo la lógica de la aplicación de la interfaz gráfica.

Una de las características principales es que los cambios que se realizan en la interfaz o en los datos se realiza automáticamente sin necesidad de tener un controlador y realizarlo manualmente [17].

Este diseño de arquitectura es una derivación del patrón MVC o Modelo Vista Controlador en la que los datos de la aplicación, la lógica y la interfaz están separados. Esta estructura obliga que la lógica de negocio tenga que actualizar la interfaz cuando se produzca una modificación [18]. En cambio, en MVVM el modelo incluye tanto la lógica como los datos de la aplicación y es el *ViewModel* el intermediario entre la interfaz y el modelo.

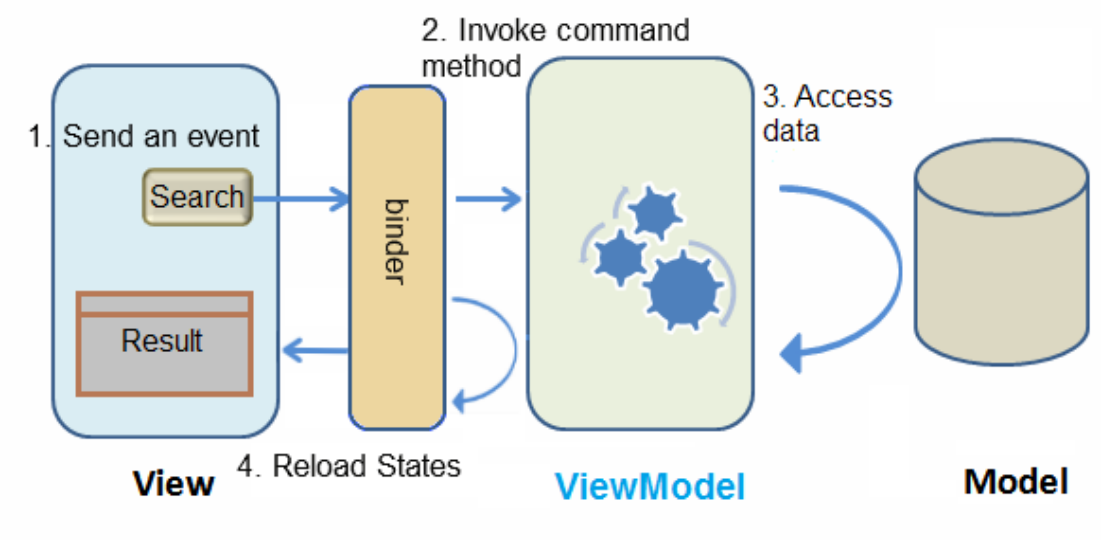

Figura 5-1: Estructura del modelo MVVM. El *binder* es el encargado de actualizar automáticamente la interfaz
Fuente: https://www.adictosaltrabajo.com

La arquitectura se basa en la estructura básica de una aplicación móvil. Por un lado, tenemos la capa de presentación o *front-end* la cual está formada por la propia aplicación y los servicios necesarios para hacer consultas o inserciones en la base de datos y por otro lado el servidor o *back-end* que procesa esas peticiones del usuario, busca en la base de datos y devuelve la respuesta correspondiente.

El servidor se ha decidido desarrollar en .Net Core, una tecnología *open-source* que nos permite desplegar el servidor en los sistemas operativos Windows, macOS y Linux. Esta desarrollada por Microsoft para .Net.

Unido a .Net Core, se ha usado la tecnología Uaithnet, creada para mejorar y simplificar las operaciones que se realizan, al igual que nos permite especificar el flujo de ejecución de estas y que ha sido desarrollada por la propia empresa Delonia. Para las conexiones con la base de datos se ha empleado *Entity Framework.*

5.3 Servicios

Los servicios para la conexión entre la aplicación el servidor se hará a través de una conexión HTTPS con la arquitectura REST lo que podremos tener conexiones seguras entre el *back-end* y el *front-end*.

Además, para la verificación de estas conexiones, se usará un *token authentication* generado con JWT y asignado a un agente cuando inicie sesión en la aplicación y que tendrá un tiempo limitado de uso. Por lo que será necesario volver a iniciar sesión en la aplicación cuando este caduque. De esta manera podremos garantizar que todos los usuarios estén verificados.

5.4 Base de Datos

Otro de los apartados que debemos diseñar antes de comenzar el desarrollo de la aplicación es la base de datos. Un buen diseño inicial facilitará las búsquedas posteriores en ella lo que se traducirá en una reducción de coste y tiempo.

Durante la realización de la solución, ha sido necesario crear dos bases de datos para satisfacer todos los requisitos. Una de ellas se encontrará en un servidor remoto a la que accederemos a través del *back-end* y donde se almacenará toda la información. La segunda, se encontrará en el dispositivo *tablet*. Esta última se encargará de almacenar todos los formularios que no puedan ser subidos a la base de datos remota cuando el usuario no tenga una conexión a internet. La base de datos local será, por tanto, igual al fragmento correspondiente al almacenamiento de solicitudes de la BD remota, incluyendo información adicional necesaria para poder sincronizar las dos bases de datos.

En el **Anexo A**, se puede ver los diseños de las dos bases de datos a partir de diagramas ER.

5.5 Maquetas

Crear maquetas permite enseñar al cliente cómo será la interfaz de la aplicación antes de que esta sea desarrollada por completo. De esta manera, un cambio en el diseño durante esta etapa causará menos problemas que un cambio de diseño con la aplicación ya creada. Es importante hacer varias versiones y compararlas entre ellas para decidir cuál es más intuitiva, más fácil de desarrollar, cual gusta más al cliente, etcétera.

Además de crear el diseño de las pantallas, también es importante indicar el flujo que van a seguir estas pantallas cuando el usuario navegue entre ellas ya que facilita mucho el trabajo de los desarrolladores.

En el **Anexo A** se incluyen algunos ejemplos de maquetas que se han realizado para el diseño de esta aplicación.

5.6 Herramientas utilizadas

A continuación, se mostrarán las diferentes herramientas usadas para la realización del diseño y la arquitectura de la aplicación.

5.6.1 Visual Paradigm

Para el diseño de la base de datos se ha optado por usar Visual Paradigm [19] ya que, aparte de ser una herramienta muy completa especializada tanto en el diseño de base de datos como los diferentes tipos de diagramas UML, permite obtener el código SQL para la generación de la base de datos.

Una vez creadas las diferentes tablas que componen la BD, con sus *primary keys* y los diferentes tipos de relaciones que existen entre ellas a través de *foreing keys*, se podrá crear el *script* SQL indicando el tipo de gestor de base de datos que se va a usar.

5.6.2 Adobe XD

Para el diseño de la interfaz gráfica y el flujo entre las diferentes pantallas se ha optado por Adobe XD [20] una herramienta perteneciente a Adobe Inc. Permite crear maquetas con grandes detalles que ayudan en la creación de la aplicación final e incluye multitud de componentes y kits de interfaces de usuario tanto de iOS como de Android. Además, facilita el desarrollo de la aplicación ya que nos permite visualizar el valor de los márgenes de la pantalla, los colores, el tamaño de letra, la distancia entre componentes…

5.6.3 Apicurio Studio

Apicurio Studio [21] ha sido elegido para el diseño de los servicios ya que es una herramienta de código abierto con multitud de opciones para diseñar y definir nuestra API. Se basa en el estándar Open API v3. Una vez definidos los servicios, podemos generar la documentación de estos en el que podemos incluir el tipo protocolo, el tipo de verificación y los valores de entrada y salida de las peticiones.

Al crear un estándar nos permite crear ejemplos de uso de los servicios que se utilizarán para generar un archivo YAML que podemos usar, con ayuda de otra herramienta APISprout, que se encuentra en el apartado 6.3.2, para simular las respuestas que nos daría el servidor cuando lo llamamos desde la aplicación. Pudiendo emular las respuestas del servidor, antes de estar este terminado

6 Desarrollo

6.1 Introducción

Una vez que se ha realizado las etapas de Análisis y Diseño se puede comenzar a desarrollar como tal la nueva aplicación. Un buen trabajo en las dos etapas anteriores ayuda reducir drásticamente el tiempo que se va a emplear en esta nueva etapa que, a priori, es la más larga de las tres.

El desarrollo inicial de la aplicación comenzó con la capa de presentación o *front-end* y la creación de los servicios necesarios para conectarla con el servidor. Una vez desarrollados, se creó la base de datos remota donde se almacenará todos los datos de la aplicación. Por último, se comenzó con el desarrollo del *back-end* permitiendo conectar la aplicación con la base de datos.

6.2 Front-End

Para el desarrollo de la aplicación se ha usado Visual Studio, el IDE desarrollado por Microsoft que permite el desarrollo con el *framework* Xamarins.Forms con el que podemos crear aplicaciones multiplataforma.

6.2.1 Visual Studio

Visual Studio [22] fue lanzado en 1997 pero no fue hasta el año 2002 cuando Microsoft presentó la plataforma .Net que permitió crear aplicaciones web, móviles y videoconsolas, entre otros, con un mismo lenguaje intermedio. Actualmente, cuenta con la versión Visual Studio 2019 con la que se ha desarrollado tanto el *front-end* como el *back-end*.

La elección de este IDE ha sido porque, con la desaparición de Xamarin Visual en mayo de 2017, es la herramienta oficial para el desarrollo con Xamarin.Forms, la tecnología con la que se ha creado la aplicación.

6.2.2 Xamarin.Forms

Xamarins.Forms [23] permite crear aplicaciones Android y iOS gracias a un completo kit de herramientas para desarrolladores .Net. La elección de este *framework* en lugar de otros también multiplataforma como puede ser React Native se debe a la implantación en Delonia Software durante los últimos años. Esta elección se debe a un compañero de la empresa que realizó un estudio y análisis decidiendo que framework de todos los existentes presentaba mejores características en el desarrollo de aplicaciones *cross-platform.* Este estudio se puede ver en el TFG que realizó [24].

Xamarin.Forms permite el desarrollo de la aplicación con lenguaje XAML y C#. Para el desarrollo de la interfaz de aplicación, se ha optado por usar archivos XML con lenguaje XAML ya que está orientado a tales tareas y nos permite separarlo de la lógica de la aplicación, que está programada en C# y en un archivo subyacente independiente. En el **Anexo B** se puede ver un ejemplo de interfaz desarrollada en XAML en el que se usan los diferentes componentes existentes.

C# además nos permite la programación asíncrona. Este tipo de programación nos permite que ciertas operaciones se realicen fuera del hilo principal de la aplicación. Al descargar este hilo principal de tareas, conseguimos que la aplicación sea más fluida y con menores tiempos de carga. Podemos conseguir así un aumento en el *throughput* haciendo que ciertas operaciones se ejecuten en segundo plano y devuelvan el control al hilo principal. Un ejemplo de uso se puede ver en el **Anexo B** en el que se explica su funcionamiento y su creación a través de los comandos *async* y *await*.

Otra de las ventajas que presenta Xamarin.Forms es el uso de *binders* ya que permiten enlazar dos objetos. Normalmente, uno de ellos es la vista o interfaz de usuario. Así, si en nuestra aplicación queremos mostrar una lista de elementos, pero no sabemos cuántos van a ser en el momento en el que se muestra la vista, podemos usar un *binder* para enlazar la lista y que esta se actualice automáticamente cuando los tengas. El uso de *binders* es típico del patrón MVVM, utilizado en el proyecto, facilitando mucho el trabajo del desarrollador. En el **Anexo B** hay un ejemplo de uso de *binders* explicando la tarea que realizan.

Xamarins.Forms cuenta también con una librería, Xamarin.Essentials, un paquete oficial que facilita el desarrollo de ciertas tareas como puede ser el almacenamiento seguro de información o la obtención de información del dispositivo y que han sido usadas en esta aplicación.

6.2.3 Android Studio

Android Studio [25] es el IDE oficial para el desarrollo Android. Aunque la herramienta como tal no ha sido usada ya que no permite el desarrollo de aplicaciones móviles multiplataforma, se ha decidido usar por el gran número de emuladores de *tablets* que presenta, por lo que podíamos comprobar que el diseño de nuestra *app* se adapte correctamente a diferentes resoluciones de pantalla y versiones de Android.

Además, esta herramienta permite acceder a todo el contenido del dispositivo, por lo que podíamos descargarnos la base de datos alojada en el dispositivo para controlar que el almacenamiento se realizaba correctamente.

6.3 Servicios

Como ya se ha comentado anteriormente, las comunicaciones con el servidor se harán con el protocolo HTTPS. De esta manera garantizamos que los datos vayan cifrados y que nadie los pueda modificar.

El patrón de diseños sigue una arquitectura REST. Es la arquitectura más famosa para la creación de servicios y presente en numerosos servicios tales como Twitter, Facebook, Youtube…

Los datos intercambiados con esta interfaz se realizan con formato JSON. En el **Anexo B** se puede ver un ejemplo de una petición POST.

6.3.1 JWT

Usando HTTPS y REST hemos garantizado la confidencialidad e integridad de los mensajes intercambiados. Para lograr la verificación de la información, se ha usado JWT

[26]. Este estándar nos permite crear un *token authentication* para propagar la integridad del mensaje.

Los JWT se componen de tres partes: el encabezado, que indica el algoritmo que se ha usado para crear la firma, el contenido en el que se incluyen los datos del *token* (privilegios del usuario, identificador del usuario…) y la firma, que se crea al codificar el encabezado y el contenido. El *token* final es la unión de las tres partes separadas por un punto.

Este *token* es creado por el servidor una vez que el usuario ha iniciado sesión en la aplicación y le sirve al usuario para usar el resto de los servicios de la aplicación. En el *token* también se añade una marca temporal que indica el tiempo que es válido. Una vez que caduca, es necesario volver a generar otro, en nuestro caso iniciar sesión, para poder seguir usando la aplicación.

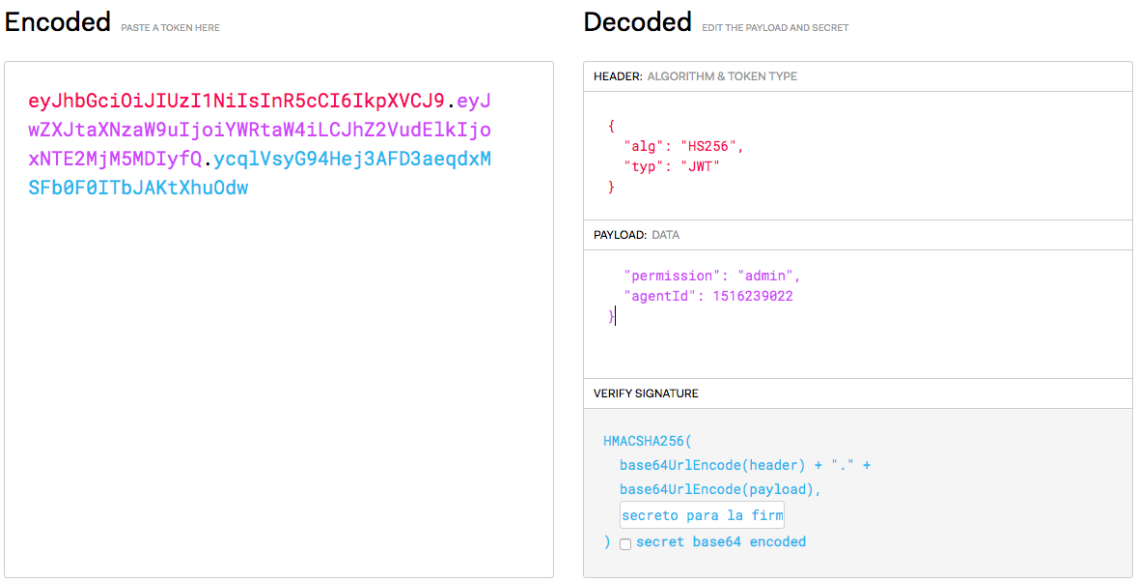

Figura 6-1: Ejemplo de un token generado con JWT
Fuente: https://jwt.io/

6.3.2 APISProut

APISprout [27] es una API multiplataforma que se encarga de simular un servidor que devuelve los ejemplos especificados en un documento [3]. De esta manera, podemos comprobar a través del archivo YAML generado con Apicurio Studio que las peticiones creadas envían correctamente los datos necesarios para cada servicio y actúan correctamente ante la respuesta generada.

Figura 6-2: Logo de Apisprout
Fuente: https://github.com/danielgtaylor/apisprout

6.4 Cliente de correo electrónico

Para poder adjuntar archivos a las solicitudes que llegaran por correo electrónico ha sido necesario crear un cliente de correo ya que se quería evitar el uso de aplicaciones externas para tales tareas. La razón era que, si se enviaba un documento comprometido por correo, como puede ser una fotocopia del DNI, al descargarlo con otra aplicación para posteriormente incluirlo a la solicitud, se quedaban almacenados en el dispositivo incumpliendo así uno de los requisitos del proyecto.

La solución, por tanto, ha sido crear un sencillo cliente de correo, conectado al servidor de correo del cliente, y que solo pudiera recibir correos e incluir los archivos adjuntos deseados al formulario. Se ha decidido usar el protocolo IMAP para el acceso al correo electrónico.

6.4.1 IMAP

IMAP es un protocolo que permite acceder al correo electrónico de un servidor de manera remota y no descargándolos como lo hace POP3, por lo que teníamos que almacenarlos en el dispositivo [28]. Al no tener que enviar mensajes no ha sido necesario implementar ningún otro protocolo de envío como SMTP.

Uno de los inconvenientes que se han encontrado al implementar este protocolo han sido las cadenas de correos, en las que un usuario responde a un correo, el usuario creador de correo vuelve a responder y así sucesivamente. El problema residía en que cada respuesta se presenta como un correo independiente, pero incluyendo todos los mensajes anteriores por lo que aparecían duplicados. Ha sido necesario analizarlo y dividirlo en partes correspondientes a cada correo.

En el **Anexo B** se muestra un fragmento del código del cliente de correo creado.

6.5 Base de datos

Como ya se ha comentado antes, ha sido necesario desarrollar dos bases de datos, una alojada en el dispositivo *tablet* y otra en un servidor remoto. A continuación, se hablará del tanto del gestor de base de datos como del administrador que se han elegido para el desarrollo de cada BD.

6.5.1 PostgreSQL y pgAdmin

Primero comenzaremos hablando de la base de datos alojado en el servidor remoto. Como SGBD se ha usado PostgreSQL [29], un gestor *OpenSource* y administrado con pgAdmin [30].

El uso de un administrador nos permite ver de forma fácil nuestra base de datos. Además, tener una interfaz gráfica permite modificar, en el caso de ser necesario, fácilmente nuestra base de datos sin la necesidad de introducir un *script* SQL. Otro de las ventajas es que podemos controlar que los datos introducidos y las relaciones entre ellos sean correctos.

Al haber generado anteriormente el *script* SQL, la creación de la base de datos ha sido relativamente sencilla. En el **Anexo B** se puede ver un fragmento del código generado automáticamente con Visual Paradigm.

6.5.2 SQLite y DB Browser

Para la BD local se ha optado por SQLite [31] un sistema de gestión de bases de datos de código abierto. La razón de usar SQLite como SGBD ha sido el gran número de librerías existentes desarrolladas en .Net que facilitan la conexión entre la aplicación móvil y la base de datos.

Al ser una base de datos muy sencilla no ha sido necesario un gran administrador para controlar todos los datos que se introducían. Por tanto, se ha decidido usar DB Browser [32], una herramienta diseñada para SQLite y que nos permite visualizar todos los datos alojados en nuestra BD.

6.6 Back-End

El *back-end* es el servidor intermedio que se encarga de conectar nuestra aplicación con la base de datos remota. Este servidor procesa las peticiones realizadas por el usuario y responderá con los datos solicitados buscándolos previamente en la base de datos.

El *back-end* se ha desarrollado también con el IDE Visual Studio usando las tecnologías .Net Core y Uaithne, desarrollada por Delonia Software.

6.6.1 .Net Core

Como ya se ha comentado antes, .Net Core se ha elegido por ser multiplataforma y porque está avalado por Microsoft.

El acceso a los datos de la BD se ha realizado con *Entity Framework* [33], una tecnología enfocada al desarrollo de aplicaciones orientada a datos. *Entity Framework* permite trabajar con los datos como si fueran objetos sin tener que pensar en la base de datos y en las tablas y columnas que la componen. De esta manera, el desarrollador puede trabajar en un nivel muy alto de abstracción al tratar con los datos logrando así reducir el código de la aplicación.

Para implementar *Entity Framework* se ha usado el método *Database First*. De esta manera, antes de comenzar con el servidor se debe crear la base de datos, con sus tablas y

estructuras. Una vez creada, se incorpora a la aplicación [34]. El uso de esta metodología se debe a que era más fácil crear la base de datos (puesto que ya teníamos el *script* SQL) que crear los modelos en la aplicación.

Otra de las tareas que realiza el servidor es la creación del *token* para la autenticación de los usuarios. En el **Anexo B** aparece el ejemplo de la creación de un *token* con JWT escrito en lenguaje C#.

6.6.1 Uaithne

Uaithne es una herramienta que funciona como *framework* y como generador de código. Como *framework* funciona uniendo una serie de disciplinas o hilos de ejecución que se unen en torno a una estructura que actúa como clase y que puede ser reutilizada más adelante.

Como generador de código funciona ahorrando línea de código repetidas que surgen con la implementación del acceso a la base de datos o por la verbosidad requerida por la arquitectura del *framework* [35].

La arquitectura que define se asemeja a un juego de LEGO©. Existen varias piezas que se intercambian para lograr el resultado deseado. Inicialmente fue diseñada para Java, pero con el tiempo se ha ido adaptando para poder usarlo con .Net Core.

El uso de esta herramienta se debe a que fue desarrollada por Delonia y, al ser usada actualmente en numerosos proyectos de la empresa, se mantiene la misma metodología.

En el **Anexo B** se incluye el código de uno de estos fragmentos reutilizables usados en el *back-end* del servidor

7 Pruebas Realizadas y Resultados

7.1 Introducción

Una vez finalizado el desarrollo de la aplicación, es necesario realizar una serie de pruebas para comprobar el buen funcionamiento del producto. Además, también es necesario verificar que se cumplen todos los requisitos recogidos en la fase de análisis y que fueron pedidos por el cliente [36, 37].

Estas pruebas han realizado tanto en la parte del *front-end* como en el *back-end* y los servicios.

7.2 Pruebas de caja blanca

Durante estas pruebas nos centramos en comprobar el funcionamiento interno y en la estructura de la aplicación mediante la examinación de la lógica interna.

Estas pruebas se realizaban al finalizar cada uno de los módulos de la aplicación y consistían en una comprobación de todas las decisiones lógicas, en recorrer todos los caminos independientes y en ejecutar todas las sentencias del módulo al menos una vez.

Una vez que un módulo pasaba todas las pruebas, se comenzaba a desarrollar el siguiente módulo.

7.3 Pruebas de caja negra

En estas pruebas nos hemos centrado en los datos de entrada y salida. A partir de unos datos introducidos en la aplicación se deben generar una salida que concuerde con las especificaciones sin importar los procedimientos software de la aplicación.

En estas pruebas han intervenido personas ajenas al proyecto permitiendo así crear diferentes casos de prueba y poder comprobar toda la aplicación. Los errores reportados se registraban en una tabla indicando el módulo al que pertenecía, el grado de importancia y una pequeña descripción para facilitar su resolución.

7.4 Pruebas de validación

Otra de las pruebas que ha pasado la aplicación ha sido la comprobación de la concordancia respecto a la lista de requisitos. Se ha repasado uno por uno la lista de requisitos para comprobar que todos estaban incluidos [38].

Al haber tenido una buena comunicación con el cliente, al que se le han ido entregando varias versiones de la aplicación para que él mismo validará la aplicación y los diferentes diseños de la interfaz gráfica y, unido al buen trabajo realizado durante la fase de análisis, el número de errores que han surgido durante estas pruebas han sido mínimos.

8 Conclusiones y trabajo futuro

Con la realización de este Trabajo de Fin de Grado se ha conseguido resolver un problema real que fue presentado a Delonia Software. Aunque en un primer momento fue dada una solución y que está recogida en el TFG de un compañero nombrado anteriormente, se ha mejorado y dado un nuevo enfoque de acuerdo con los cambios y las nuevas peticiones realizadas por el cliente.

La solución propuesta se ha basado en la creación de una aplicación para *tablets* que ayuda a los comerciales de la compañía en el proceso de adhesión de un comercio al sistema de pago de dicha compañía. La aplicación logra la integridad de todos los sistemas garantizando la seguridad y confidencialidad de la información de los clientes.

Durante el desarrollo de todo este proyecto he aprendido como funciona un proyecto software desde su inicio hasta la entrega final al cliente. Aunque la base de este trabajo se ha realizado sobre la etapa de desarrollo, sobre la cual he conseguido aplicar los conocimientos que he obtenido durante mi carrera, me ha servido también para adquirir nuevas ideas que podré usar en futuros proyectos.

El uso de la tecnología Xamarins.Forms y el *framework* .Net me han abierto un nuevo campo en el desarrollo software que actualmente está muy demandado y me ha parecido muy interesante y con el que espero seguir especializándome durante los próximos años.

En este resumen final quiero destacar también la buena comunicación que he tenido con el jefe de proyecto que ha sido asignado y que me ha proporcionado los conocimientos necesarios para sacar adelante este trabajo.

Para finalizar con este proyecto se presenta una lista de trabajos futuros relacionados:

- Desarrollo para la plataforma iOS: adoptar la aplicación sería desarrollada para *tablets* con sistema operativo iOS. Este trabajo estaría muy avanzado debido al uso de Xamarin.Forms para el desarrollo, ya que es multiplataforma y permite reutilizar un alto grado del código

- Creación de formularios dinámicos: la versión actual de la aplicación solo permite rellenar un tipo de formulario. En futuras versiones, se permitiría a los administradores crear sus propios formularios o modificar los ya existentes haciendo la aplicación mucho más completa.

- Plataforma web para la gestión de solicitudes: en este proyecto se mejoraría la versión actual web que la compañía tiene para gestionar las solicitudes creadas por los agentes. Adoptándola a los servicios desarrollados. Los administradores podrían ver las solicitudes subidas y podrán aceptarlas o no, dependiendo del criterio del negocio. Además, incluiría la creación de nuevos formularios que serían cargados desde la aplicación. Este proyecto ya se ha iniciado y actualmente es en el que me encuentro trabajando.

Referencias

[1] Anónimo. Sistemas Operativos. https://www.areatecnologia.com/sistemas-operativos.htm

[2] Anónimo. Symbian, el Sistema Operativo móvil. https://www.vix.com/es/btg/tech/2007/03/12/ symbian-el-sistema-operativo-movil

[3] Anónimo. Android. https://es.wikipedia.org/wiki/Android

[4] Google. Android. https://www.android.com/intl/es_es/

[5] Anónimo. iOS. https://es.wikipedia.org/wiki/IOS

[6] Fátima Martinez. Informe 2019 Usuarios Internet, Redes Sociales, Móvil e Ecommerce en España y en el Mundo. https://fatimamartinez.es/2019/02/01/informe-2019-usuarios-internet-redes-sociales-movil-e-ecommerce-en-espana-y-en-el-mundo/

[7] Ditrendia. Consumo, uso y tendencias del móvil: 'Informe Mobile en España y en el mundo 2018' http://mktefa.ditrendia.es/blog/consumo-uso-y-tendencias-del-m%C3%B3vil-informe-mobile-en-espa%C3%B1a-y-en-el-mundo-2018

[8] Solbyte. Tipos de aplicaciones móviles: nativas, webs, híbridas. https://www.solbyte.com/blog/2014/07/21/tipos-de-aplicaciones-moviles-nativas-webs-hibridas/

[9] HAFO. Desarrollo de aplicaciones híbridas. https://aplicacionesmovil.com/marketing-movil/desarrollo-de-aplicaciones-hibridas/

[10] Deustro Formación. Descubriendo Appcelerator Titanium. https://www.deustoformacion.com/blog/desarrollo-apps/descubriendo-appcelerator-titanium-i-definicion-caracteristicas-basicas

[11] Javier Escacena. Desarrollando aplicaciones móviles nativas con React Native. https://www.paradigmadigital.com/dev/desarrollando-aplicaciones-moviles-nativas-con-react-native/

[12] Luis Setfree. Conociendo Xamarin, la herramienta para desarrolladores multiplataforma. https://www.vix.com/es/btg/tech/13263/conociendo-xamarin-la-herramienta-para-desarrolladores-multiplataforma

[13] Anónimo. Qué es Scrum. https://proyectosagiles.org/que-es-scrum/

[14] Anónimo. Scrum: una manera de ejecutar los principios de la metodología Ágil. https://www.ticportal.es/glosario-tic/scrum-implementacion-proyectos

[15] Sourcetree. https://www.sourcetreeapp.com/

[16] Anónimo. https://www.infor.uva.es/~mlaguna/is1/apuntes/2-requisitos.pdf

[17] Shamlia. Understanding the basics of MVVM design pattern. https://blogs.msdn.microsoft.com/msgulfcommunity/2013/03/13/understanding-the-basics-of-mvvm-design-pattern/

[18] Anónimo. MVC y MVVM. https://www.adictosaltrabajo.com/2012/10/07/zk-mvc-mvvm/

[19] Visual Paradigm. https://www.visual-paradigm.com/

[20] Adobe XD. https://www.adobe.com/es/products/xd.html

[21] Apicurio Studio. https://www.apicur.io/

[22] Visual Studio. https://visualstudio.microsoft.com

[23] Xamarin.Forms. https://docs.microsoft.com/es-es/xamarin/xamarin-forms/

[24] Stanislav Mostovoy. Aplicación móvil multiplataforma, nativa y de código único para la publicación de contenidos gestionados desde web. https://repositorio.uam.es/handle/10486/668430.

[25] Android Studio. https://developer.android.com/studio

[26] JWT. https://jwt.io/

[27] APISProut. https://github.com/danielgtaylor/apisprout

[28] Anónimo. IMAP VS POP3: ¿Cuáles son sus diferencias? https://www.axarnet.es/blog/imap-vs-pop3-cuales-son-sus-diferencias/

[29] PostgreSQL. https://www.postgresql.org/

[30] pgAdmin. https://www.pgadmin.org

[31] SQLite. https://www.sqlite.org/

[32] DB Browser for SQLite. https://sqlitebrowser.org/

[33] Microsoft. Entity Framework. https://docs.microsoft.com/es-es/ef/

[34] Diego Bersano. Entity Framework: Database First. https://docs.microsoft.com/es-es/ef/

[35] Josué Fernández León. “Evolución y aseguramiento de la calidad en Uaithne e implantación en Unit Linked”. https://repositorio.uam.es/bitstream/handle/10486/662249/fernandez_leon_josue_tfg.pdf?sequence=1

[36] Juan Quijano. ¿Qué pruebas debemos hacerle a nuestro software y para qué? https://www.genbeta.com/desarrollo/que-pruebas-debemos-hacerle-a-nuestro-software-y-para-que

[37] Panel Testing - Centro de Excelencia. Software QA – ¿Cuáles son los tipos de pruebas software? https://www.panel.es/blog/software-qa-cuales-son-los-tipos-de-pruebas-software/

[38] José M. Drake y Patricia López. Verificación y Validación. https://www.ctr.unican.es/asignaturas/Ingenieria_Software_4_F/Doc/M7_09_VerificacionValidacion-2011.pdf

Glosario

API	Application Programming Interface
IDE	Integrated Development Environment
SO	Sistema Operativo
SSOO	Sistemas Operativos
APP	Aplicación
SGBD	Sistema de Gestión de Base de Datos
BD	Base de Datos
SQL	Structured Query Language
IMAP	Internet Message Access Protocol
POP3	Post Office Protocol version 3
SMTP	Simple Mail Transfer Protocol
ER	Entidad-Relación
HTTPS	Hypertext Transfer Protocol Secure
REST	Representational State Transfer
POST	Tipo de solicitud soportada por HTTPS
JWT	JSON Web Tokens
JSON	JavaScript Object Notation
YAML	Formato de serialización de datos legible por humanos
AES	Advanced Encryption Standard
DNI	Documento Nacional de Identidad
SL	Sociedad Limitada
TFG	Trabajo de Fin de Grado
MVVM	Modelo Vista VistaModelo

MVC	Modelo Vista Controlador

Anexos

A Diagramas y Maquetas

A.1 Diagramas ER

Diagramas Entidad-Relación usados para el diseño de las dos bases de datos que usa el proyecto y realizados con la herramienta Visual Paradigm.

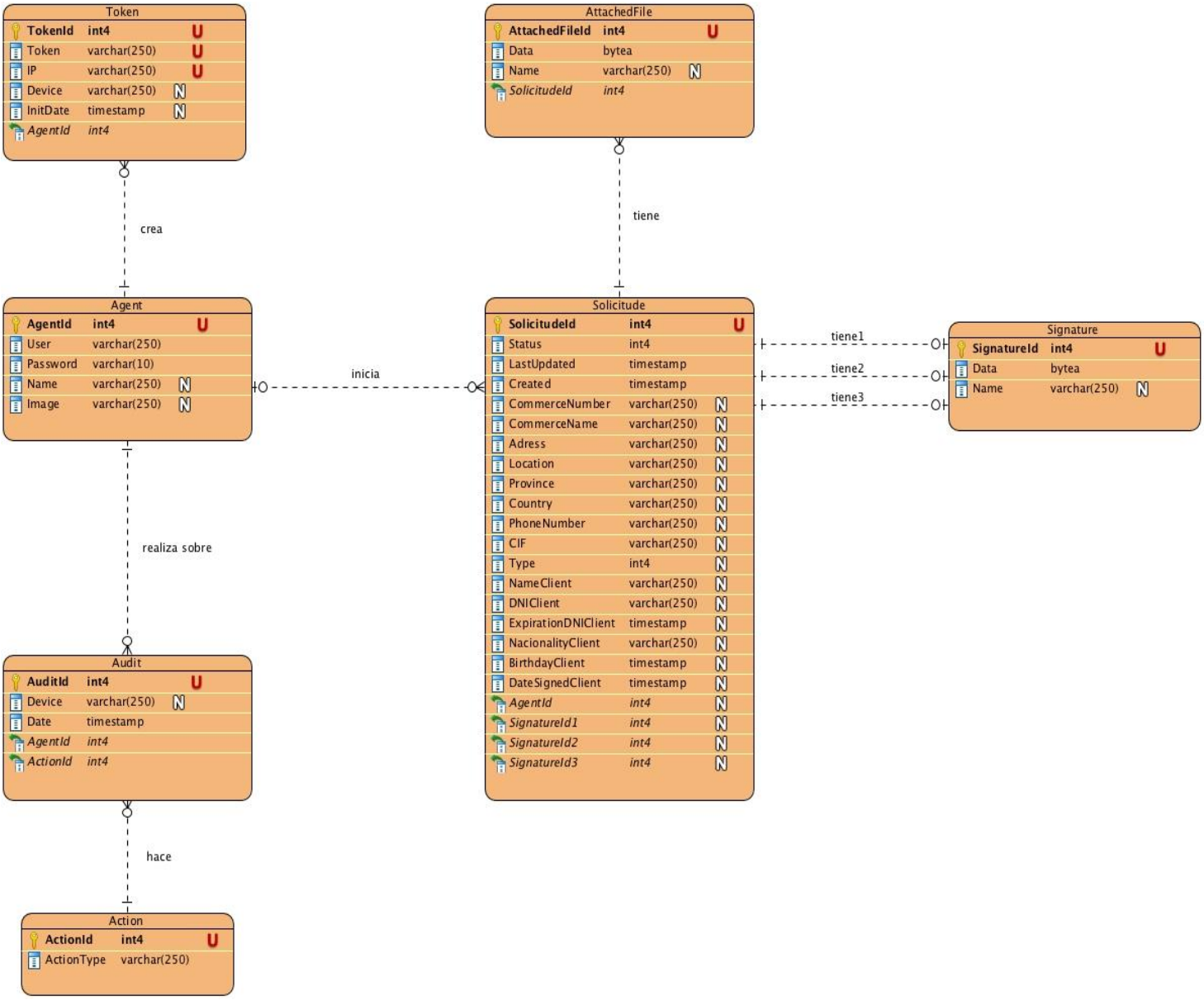

Figura A-1: Diagrama ER de la base de datos alojada en el servidor.
Fuente: Propia

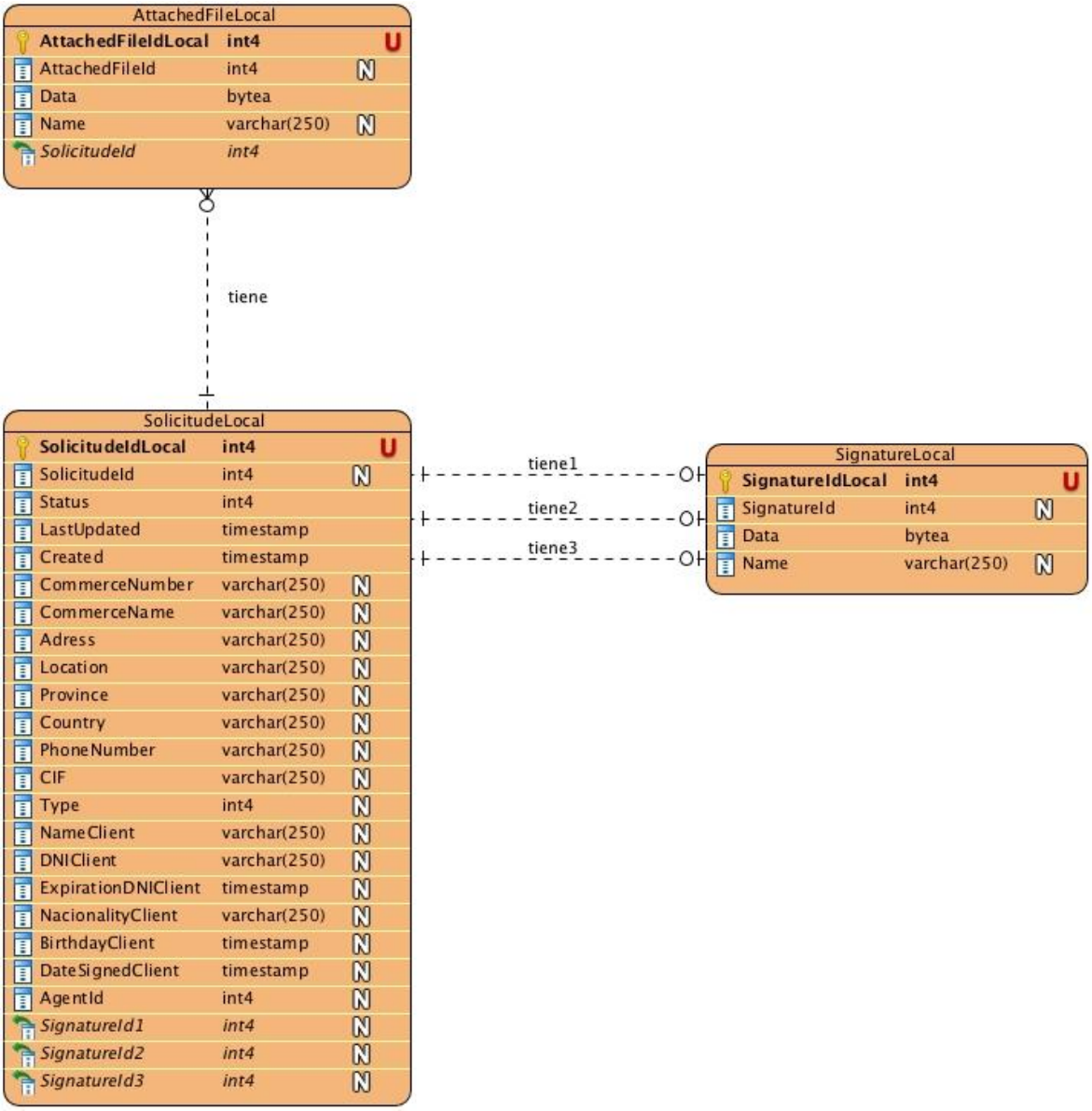

Figura A-2: Diagrama ER de la base de datos alojada en la *tablet*.
Fuente: Propia

A.2 Maquetas

En este apartado se muestran algunas de las maquetas diseñadas para desarrollar la aplicación con Adobe XD.

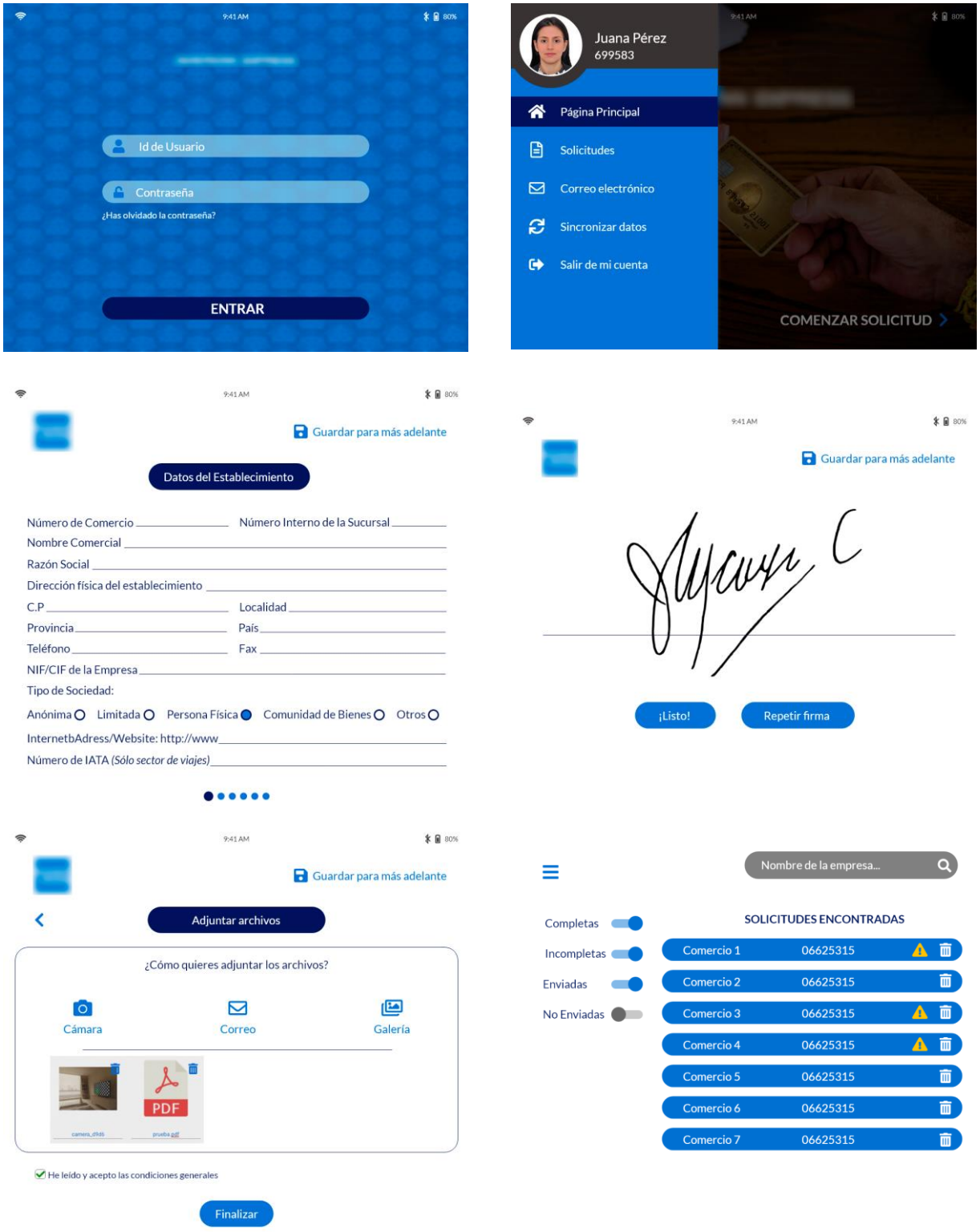

Figura A-3: Ejemplos de maquetas de la aplicación
Fuente: Propia

B Manual del programador

B.1 Componentes XAML y *binders*

Ejemplo de creación de una lista (*ListView*) de elementos con XAML. Cada elemento (*ViewCell*) está dividido en una cuadrícula (*Grid*) a la que se le añade texto (*Label*) e imágenes (*Image*).

Como el texto de cada elemento es dinámico, se usan los *binders* para que lo actualicen automáticamente liberando así al Modelo de tales tareas. Los *binders* no solo actualizan texto, sino que también pueden usarse para las funciones y los parámetros que se deben llamar cuando pulses en un botón.

```
<ListView x:Name="Table" ItemTapped="OnMenuItemSelected"
 SelectionMode="None" HasUnevenRows="true"
 HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand">
  <ListView.ItemTemplate>
     <DataTemplate>
          <ViewCell>
               <Frame BackgroundColor="#0070D2" CornerRadius="25"
                Padding="45,0,0,0" Margin="0,20,0,0"
                VerticalOptions="CenterAndExpand">
                    <Grid x:Name="Item">
                         <Grid.ColumnDefinitions>
                              <ColumnDefinition Width="*" />
                              <ColumnDefinition Width="350" />
                              <ColumnDefinition Width="80" />
                              <ColumnDefinition Width="80" />
                         </Grid.ColumnDefinitions>
                         <Grid.RowDefinitions>
                              <RowDefinition Height="50" />
                         </Grid.RowDefinitions>
                         <Label Text="{Binding CompanyName}"
                          Grid.Column="0" FontSize="25"
                          VerticalOptions="Center"
                          TextColor="white" />
                         <Label Text="{Binding CommerceNumber}"
                          Grid.Column="1" FontSize="25"
                          VerticalOptions="Center"
                          TextColor="white" />
                         <Image Source="error" Grid.Column="2"
                          VerticalOptions="Center"
                          IsVisible="{Binding Status}" />
                         <Image Source="icono_papelera_blanca"
                          Grid.Column="3" VerticalOptions="Center"
                          IsVisible="{Binding Status}">
                              <Image.GestureRecognizers>
                                   <TapGestureRecognizer
                                    BindingContext="{Binding
                                    Source= {x:Reference Table},
                                    Path=BindingContext}"
                                    Command="{Binding
                                    RemoveSolicitudeCommand}"
                                    CommandParameter="{Binding
```

```
                                                    Source={x:Reference Item}}"
                                                    />
                                            </Image.GestureRecognizers>
                                    </Image>
                            </Grid>
                    </Frame>
            </ViewCell>
      </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

B.2 Función asíncrona

El ejemplo de la función asíncrona siguiente se utiliza para obtener la lista de todas las solicitudes de la base de datos local. Las funciones asíncronas de declaran con *async* y siempre devuelve un objeto tarea (*Task*<*T*>).

La función del ejemplo se queda esperando a recibir los valores de la BD pero sin bloquear ningún proceso. Por último, la función devuelve una lista de objetos *SolicitudeInfo* que contiene los datos principales de una solicitud;

```
public async Task<List<SolicitudeInfo>> GetSolicitudesListLocal(int
  id)
    {
    var completeList = await
        Database.Connection.Table<SolicitudeLocal>().Where(solicitude
        => solicitude.AgentId == id).ToListAsync();

   return completeList.Select(
       s => new SolicitudeInfo
           {
             SolicitudeId = s.SolicitudeId,
             SolicitudeIdLocal = s.SolicitudeIdLocal,
             CommerceNumber = s.CommerceNumber,
             CompanyName = s.CommerceName,
             Status = s.Status,
             LastUpdate = (DateTime)s.LastUpdate
           }).ToList();
    }
```

B.3 Petición POST

A continuación, se presenta la función genérica para realizar una petición POST. Se debe pasar la ruta del servidor a la que se va a hacer la petición y los argumentos que van a tener. Estos se pasan en formato JSON previamente convertidos.

La función acepta cualquier tipo de respuesta y, además, comprueba que el *token* sea válido.

```
async Task<T> Post<T>(string source, object arguments)
{
    Uri _uri = new Uri(string.Concat(Constants.URL, source));
    try
```

```
{
    HttpContent sendData;
    Byte[] sendDataBody =
      Encoding.UTF8.GetAllBytes(JsonConvert.SerializeObject(argume
      nts, Formatting.None, new JsonSerializerSettings {
      NullValueHandling = NullValueHandling.Ignore,
      DateFormatHandling = DateFormatHandling.IsoDateFormat
      })).ToArray<Byte>();

    sendData = new ByteArrayContent(sendDataBody);
    sendData.Headers.Add("Content-Type", "application/json");

              HttpResponseMessage responseData = await
                _client.PostAsync(_uri, sendData);

              //Respuesta correcta
              if (responseData.IsSuccessStatusCode)
              {
                  var content = await
                    response.Content.ReadAsStringAsync();
                  return JsonConvert.DeserializeObject<T>(content);

              }
               //Usuario con token caducado
              else if (responseData.StatusCode ==
                   HttpStatusCode.Unauthorized)
              {
                  App.Current.ResetAll();

                  Await  DisplayAlert  ("ERROR",  "Su  sesión  ha
                     caducado. Inicie sesión de nuevo", "Ok");

                  //Volvemos a la página de iniciar sesión
                  App.Current.MainPage = new LogInPage();
                  return default;
              }
              await DisplayAlert("ERROR", "No hay conexión a
                internet", "Ok");
              return default;

          }
          //No internet
          catch (Exception ex)
          {
              await DisplayAlert("ERROR", "Sin conexión a
                 internet", "Ok");
              return default;
          }
     }
```

B.4 Cliente de correo

El código que se muestra a continuación corresponde al cliente de correo creado en la aplicación. Para crearlo, se ha usado la librería externa *MailKit* que es de código abierto y desarrollada en .Net.

La siguiente función muestra el método de autenticarse:

```
public async Task<bool> Authenticate(string userName, string
  password)
        {
            if (!_imapClient.IsConnected)
            {
                Debug.WriteLine("\t\tNo estás conectado al
                    servidor");
                await Connect();
            }
            try
            {
                //Inicio de sesion
                await _imapClient.AuthenticateAsync(userName,
                    password);
                //Abrimos tambien la carpeta
                await
                 _imapClient.Inbox.OpenAsync(FolderAccess.ReadOnly);
               //Empezamos a leer por el último que es el más
                 reciente
                startReading = _imapClient.Inbox.Count - 1;
                return true;
            }
            catch (Exception ex)
            {
                 //…
            }

        }
```

A continuación, se muestra la función para obtener los n primeros correos del servidor devueltos como una lista del objeto creado *Email*. Al obtener solo un número determinado de correos, reducimos el tiempo de espera de la aplicación.

```
public async Task<List<Email>> GetFirstEmails(int number)
        {
            var listOfEmail = new List<Email>();
            var inbox = _imapClient.Inbox;
            var total = inbox.Count - 1;
            try
            {
                for (int i = total; i > total - number; i--)
                {
                    listOfEmail.Add(await GetEmail(i));
                }
            }
            catch (Exception ex)
            {
                //…
            }
            return listOfEmail;
        }
```

Esta función llama a *GetEmail* que es la encargada de obtener el correo del servidor y crear el objeto *Email* con los datos recibidos.

```
async Task<Email> GetEmail(int index)
        {
            var message = await
               _imapClient.Inbox.GetMessageAsync(index);
            var email = new Email
            {
                Id = message.MessageId,
                Author = message.From[0],
                Subject = message.Subject,
                Text =
                  message.GetTextBody(MimeKit.Text.TextFormat.Plain),
                Attachments = new List<MimePart>()
            };

            foreach (var attachment in message.Attachments)
           //Añadimos todos los adjuntos convirtiendolos en File que
              es el que necesitamos para trabajar
            {
                email.Attachments.Add((File)attachment);
            }
            return email;
        }
```

B.5 Uaithne

Ejemplo del controlador de Uaithne de la funcionalidad relacionada con el usuario en el que existen varios hilos (*Stream*) de ejecución según la tarea a ejecutar (iniciar sesión, cerrar sesión u obtener la información).

Dentro de estos hilos se ejecutan los fragmentos reutilizables (leer de base de datos, crear un *token*, escribir en BD…) y que son la característica principal de Uaithne.

```
public UserController(Context Context, IConfiguration configuration)
{
     _configuration = configuration;
     _context = Context;//Información común para todos los Stream

    _logInStream = new DBReaderExecutor(_context)
         {
             Next = new CreateTokenExecutor
             {
                 Next = new DBWriterExecutor(_context)
                 {
                     Next = new LogInExecutor()
                 }
             }
         };
    _logOutStream = new DBTokenExecutor(_context)
         {
             Next = new DBWriterExecutor(_context)
          };
     _getMyInfoStream = new DataBaseReaderExecutor(_context);
}
```

B.6 Creación de un *token*

La creación de *tokens* se realiza con JWT. La función que se muestra a continuación es el *Executor* de crear el *token* del hilo (*Stream*) de inicio de sesión. Es uno de los fragmentos reutilizables creados con el método Uaithne.

El método recibe por argumentos una operación que contiene el identificador del agente, el valor que va a tener el token y el secreto para realizar la firma. Para crearel *token*, primero se crean el *payload* donde se incluye el ID y el valor del *token*. Después, se añade la cabecera que incluye el tipo de algoritmo que se ha usado para firmar y, por último, se crea el *token* como tal, incluyendo los datos anteriores.

La función devuelve una respuesta genérica con el objeto *LoginResponse* en el que se incluye el *token* y el resto de datos del usuario obtenidos en otro *Executor*.

```
public GenericResponse ExecuteOperation(LoginOperation operation,
  Context context)
{
   Contract.Ensures(Contract.Result<IActionResult>() != null);
            var claims = new Claim[]
            {
              new Claim("token", operation.Token),
              new Claim(ClaimTypes.NameIdentifier,
                    operation.AgentId.ToString()),
              new Claim(JwtRegisteredClaimNames.Nbf, new
                    DateTimeOffset(DateTime.UtcNow).
                    ToUnixTimeSeconds().ToString()),
              new Claim(JwtRegisteredClaimNames.Exp, new
                  DateTimeOffset(DateTime.UtcNow.AddMinutes(
                  operation.TimeToLiveToken)).
                  ToUnixTimeSeconds().ToString()),
            };
            var token = new JwtSecurityToken(
                new JwtHeader(new SigningCredentials(
                    new SymmetricSecurityKey(
                         Encoding.UTF8.GetBytes(operation.Secret)),
                         SecurityAlgorithms.HmacSha256)),
                new JwtPayload(claims));

            var response = new GenericResponse
            {
                StatusCode = StatusResponseEnum.OK,
                Data = new LoginResponse
                {
                    Token = new
                       JwtSecurityTokenHandler().WriteToken(token),
                    IdAgent = operation.AgentId,
                    Name = operation.Name,
                    Image = operation.Image
                }
            };

            return response;
        }
```

B.7 Código SQL

Fragmento del código SQL generado automáticamente con la herramienta Visual Paradigm de la base de datos remota una vez creado el diagrama ER.

```
CREATE TABLE Action (ActionId SERIAL NOT NULL, ActionType
  varchar(250) NOT NULL, PRIMARY KEY (ActionId));

CREATE TABLE Agent (AgentId SERIAL NOT NULL, "User" varchar(250) NOT
  NULL, Password varchar(10) NOT NULL, Name varchar(250), Image
  varchar(250), PRIMARY KEY (AgentId));

CREATE TABLE Audit (AuditId SERIAL NOT NULL, Device varchar(250),
  "Date" timestamp NOT NULL, AgentId int4 NOT NULL, ActionId int4 NOT
  NULL, PRIMARY KEY (AuditId));

CREATE TABLE Solicitude (SolicitudeId SERIAL NOT NULL, Status int4
  NOT NULL, LastUpdated timestamp NOT NULL, Created timestamp NOT
  NULL, CommerceNumber varchar(250), CommerceName varchar(250),
  Adress varchar(250), Location varchar(250), Province varchar(250),
  Country varchar(250), PhoneNumber varchar(250), CIF varchar(250),
  Type int4, NameClient varchar(250), DNIClient varchar(250),
  ExpirationDNIClient timestamp, NacionalityClient varchar(250),
  BirthdayClient timestamp, DateSignedClient timestamp, AgentId int4,
  SignatureId1 int4, SignatureId2 int4, SignatureId3 int4, PRIMARY
  KEY (SolicitudeId));

ALTER TABLE Token ADD CONSTRAINT crea FOREIGN KEY (AgentId)
  REFERENCES Agent (AgentId);

ALTER TABLE Audit ADD CONSTRAINT hace FOREIGN KEY (ActionId)
  REFERENCES Action (ActionId);

ALTER TABLE Solicitude ADD CONSTRAINT inicia FOREIGN KEY (AgentId)
  REFERENCES Agent (AgentId);

ALTER TABLE Audit ADD CONSTRAINT "realiza sobre" FOREIGN KEY
  (AgentId) REFERENCES Agent (AgentId);

ALTER TABLE AttachedFile ADD CONSTRAINT tiene FOREIGN KEY
  (SolicitudeId) REFERENCES Solicitude (SolicitudeId);

ALTER TABLE Solicitude ADD CONSTRAINT tiene1 FOREIGN KEY
  (SignatureId1) REFERENCES Signature (SignatureId);

ALTER TABLE Solicitude ADD CONSTRAINT tiene2 FOREIGN KEY
  (SignatureId2) REFERENCES Signature (SignatureId);

ALTER TABLE Solicitude ADD CONSTRAINT tiene3 FOREIGN KEY
  (SignatureId3) REFERENCES Signature (SignatureId);
```