Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería del Software

AppGenda: app Android como agenda virtual de las PYMEs de servicios de un Ayuntamiento

---

Alumno: Juan Carlos Garrote Gascón

Tutor: Yania Crespo González-Carvajal

A mi familia, que siempre ha estado ahí

Agradecimientos

A mi familia, que siempre me ha apoyado y ha aguantado mis altibajos estos meses.

A mis amigos, que también me han apoyado y animado a conseguir lo que me propusiese.

A mis compañeros de carrera que se han convertido en amigos, sin vosotros no habría sido lo mismo este paso por la universidad.

A los establecimientos de mi pueblo, por colaborar de manera desinteresada en este proyecto con lo que he necesitado.

A Samuel Alfageme, Fernando Javier Rodríguez, Alejandra Martínez y Silvia Garrote, que me han ayudado de una manera u otra con ciertas tareas en algunos puntos del proyecto.

A mi tutora Yania, que a pesar de trabajar de sol a sol, siempre ha sacado tiempo para atenderme y ayudarme cuando lo he necesitado.

Gracias a todos

Resumen

El objetivo de este proyecto es desarrollar una aplicación móvil que permita funcionar como un sustituto de la agenda en papel de los diversos negocios que trabajan con citas y/o pedidos. Además, la aplicación ofrece la posibilidad de establecer una interacción entre los clientes y los dueños de negocio mediante solicitudes y sus correspondientes respuestas, así como un mapa y un tablón de anuncios para facilitar la comunicación desde los establecimientos hacia los clientes.

El proyecto se ha desarrollado como una app para el sistema operativo Android, empleando Java como lenguaje de programación, siguiendo las pautas marcadas por el marco de trabajo ágil Scrum así como una guía de buenas prácticas y de estilo para conseguir un producto de alta calidad.

Abstract

The aim of this project is to develop a mobile application which allows to work as a substitute for the paper agenda of the various businesses that work with appointments and/or orders. Moreover, the application offers the possibility to establish an interaction between the clients and the business owners through requests and their corresponding replies, as well as a map and a noticeboard to facilitate communication from the establishments towards the clients.

The project has been developed as an app for Android operating system, using Java as programming language, following the guidelines set by Scrum agile framework as well as a guide of best practices and of style to get a high quality product.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Actualmente, las tecnologías móviles están en pleno auge. En concreto, a fecha de enero de 2021, en el mundo existen 5,22 mil millones de usuarios de dispositivos móviles, representando al 66,6 % de la población mundial. Es decir, 2 de cada 3 personas en el mundo tienen a su disposición un dispositivo móvil. Asimismo, del total de usuarios de Internet, 91,5 % de ellos acceden desde un teléfono inteligente o *smartphone* [127]. En un país como España, las cifras se disparan hasta un 97,8 % de ciudadanos desde los 16 hasta los 64 años con un dispositivo móvil inteligente en su posesión, a fecha de enero de 2021, y siendo un 92,1 % los usuarios que acceden a Internet desde un teléfono inteligente [126]. Ambas cifras van en aumento cada año, lo que refleja el gran uso cotidiano que se da a estas tecnologías.

Por otro lado, cada vez existen más innovaciones tecnológicas en el mundo de las empresas, lo que fomenta la introducción de nuevas tecnologías en las mismas, entre ellas, los smartphones, un elemento cada vez más habitual en cada uno de los aspectos del día a día como acabamos de ver, y por tanto cada vez más accesible a un público más general.

Además, la situación actual provocada por el COVID-19 ha impulsado a utilizar aún más ampliamente las tecnologías para facilitar la vida de las personas, que se ha visto muy afectada por las restricciones que este contexto sanitario implica. Las pequeñas y medianas empresas han sido uno de los sectores de la economía más afectados, y aunque están en proceso de recuperación, todavía queda un largo camino por recorrer, en el que todo apoyo es bien recibido.

AppGenda nace en medio de toda esta situación para aprovechar la extendida utilización de la tecnología móvil y constituir una herramienta de utilidad tanto para negocios y servicios como para los clientes de los mismos, permitiendo gestionar conjuntamente citas y pedidos y ofreciendo otras funcionalidades que una agenda de papel no podría llevar a cabo.

1.2. Motivación

La idea surge tras observar en el entorno cercano cómo la mayoría de negocios pequeños y medianos siguen utilizando la tradicional agenda de papel para gestionar sus citas y pedidos. En el caso de algunos negocios, sí que tienen su sistema de gestión de reservas informatizado, pero mayoritariamente éste se limita a que solamente el dueño de negocio pueda trabajar con los registros almacenados. Además, dichos sistemas suelen instalarse en un ordenador, normalmente en la zona de trabajo, limitando la interacción con los mismos cuando no se está físicamente en ese lugar, al igual que una agenda de papel. Aunque sean poco frecuentes los casos en los que se desea consultar o añadir nuevos datos al sistema cuando se está fuera del trabajo, resulta muy incómodo no poder hacerlo cuando ocurren, casi tanto como tener que transportar la agenda fuera del trabajo cada vez que termina la jornada.

Viendo cómo los dispositivos móviles inteligentes funcionan como herramienta en muchas situaciones cotidianas, lo accesibles que se han convertido y que siempre llevamos uno encima, y en vista de la situación producida por el COVID-19 y las complicaciones que la misma produce, se decidió llevar adelante este proyecto. El resultado buscado es una gran mejora en la comodidad respecto a las situaciones que se planteaban en el párrafo anterior, la promoción del trabajo de los negocios que se puedan beneficiar del proyecto y una alternativa atractiva, accesible, gratuita, completa y fácil de utilizar para todas las partes involucradas en el proceso de venta de un producto o servicio (principalmente responsable del servicio y cliente). Sin embargo, el resultado también busca suponer una innovación respecto a las alternativas existentes que se verán en la siguiente sección, ya que ninguna de ellas junta todas las características que se incluyen en este proyecto.

1.3. Estudio de alternativas

Aunque actualmente no existe una alternativa que suponga un claro competidor para la app móvil objetivo, se ha hecho una búsqueda de posibles apps similares o que se puedan utilizar para el mismo propósito, destacando sus ventajas y desventajas frente al presente proyecto, dado que se busca combinar las mejores características de cada una de estas alternativas a la vez que innovar para crear algo no disponible actualmente en el mercado. Las siguientes subsecciones analizarán algunas posibles alternativas, desde las más concretas a las más generales.

1.3.1. Sistema de reserva de citas

La app *Sistema de reserva de citas* [59] presenta una idea similar a lo que se pretende desarrollar en este proyecto en la parte de reserva de citas.

Ventajas:

- Es simple y cumple su función.

- Permite seleccionar la duración de la franja horaria que ocupa la cita, así como los horarios en los que se ofrecen dichas citas.
- Permite interacción con el cliente, ya que es el propio cliente el que pide la cita a través de una página web.
- El dueño de negocio también puede registrar nuevas citas.

Desventajas:

- No es demasiado popular, pues sólo posee alrededor de 10000 descargas.
- Interfaz de usuario no muy usable, a veces muestra mucha información por pantalla y su manejo es poco intuitivo.
- No traducida correctamente al español. Algunas opciones o mensajes del manejo de errores de la app no son del todo entendibles.
- El uso de esta app implica la utilización también de una aplicación web por parte de los clientes, es decir, no está todo unificado en una sola app.
- No permite almacenar información asociada a la cita más allá del nombre del cliente que la ha solicitado, y su email opcionalmente.
- Aunque el cliente pueda solicitar una cita, no se exige confirmación por parte del negocio, se registra directamente. Por tanto, si el negocio decide rechazar la cita o tiene que cancelarla, el cliente no recibirá ningún tipo de aviso.
- Sólo tiene soporte para citas, no para pedidos. Tampoco se puede hacer un uso de las citas imitando a pedidos si así se quisiera, ya que no se puede almacenar información adicional y ocuparía innecesariamente una franja horaria.
- La app se puede utilizar de forma gratuita para un máximo de 20 horas de reserva al mes, a partir de ese punto requiere suscripción.

Muchas otras apps similares a la descrita, como por ejemplo *Reservio* [57] o *Bookitit* [52], poseen las mismas desventajas, principalmente la carencia de popularidad, que se pone el foco sólo en citas y que se ofrece una aplicación web para los clientes. En el caso de la segunda, también se puede destacar su interfaz poco usable (las principales quejas de los usuarios en las reseñas es por este motivo) y que posee un período gratuito de prueba de 15 días, a partir del cual se requiere suscripción de pago. Otras apps como *SalonAppy* [58], *ReBox* [56] o *WodBuster* [61], aparte de centrarse sólo en citas, están más dirigidas a ciertos tipos de negocios, como son los salones de belleza en el caso de la primera app y los gimnasios o centros deportivos en el caso de las dos últimas, por lo que no son fácilmente ampliables a un uso general.

1.3.2. Evernote

La app de *Evernote* [53], aunque posee una funcionalidad más general, ya que es un organizador de notas, se puede utilizar como gestor de agenda de trabajo.

Ventajas:

- Es muy popular, posee más de 100 millones de descargas.
- Su interfaz de usuario es muy usable, cumple con muchos principios de usabilidad y posee una gran aceptación entre sus usuarios.
- Permite guardar información relacionada con las notas (que funcionarían como citas o pedidos), desde texto hasta archivos multimedia.
- Permite tener una agenda compartida, por lo que todos los empleados de un mismo negocio podrían tener acceso a la agenda virtual del mismo.

Desventajas:

- Aunque se pueda utilizar como tal, no es una app que se centre en la funcionalidad que cumple una agenda, sino para guardar notas con distintos propósitos.
- No existe interacción con el cliente. La funcionalidad de agenda compartida no es adecuada ni viable para el propósito de reservar citas o pedidos.
- Aunque fuera viable compartir la agenda con los clientes, esta característica es de pago.

1.3.3. Google Calendar

Google Calendar [55] es posible que constituya la alternativa más popular actualmente entre los dueños de negocio para gestionar su agenda de trabajo.

Ventajas:

- Gran popularidad, con más de 1000 millones de descargas.
- Interfaz muy usable, siguiendo los principios de *Material Design* [74].
- Totalmente gratuita.
- Permite organizar eventos genéricos (que podrían constituir citas o pedidos) con información asociada, como una descripción textual o cualquier tipo de archivo.

Desventajas:

- La funcionalidad de la app es mucho más amplia y no se ciñe sólo a citas y pedidos, sino que constituye un gestor de calendario.
- No hay interacción con el cliente. Se trata de un gestor personal, sin capacidad para realizar solicitudes de creación de eventos.

1.3.4. WhatsApp Business

La última alternativa tenida en cuenta ha sido *WhatsApp Business* [60]. Esta aplicación móvil permite una interacción personalizada entre cliente y negocio mediante un servicio de chat, similar a su famoso hermano *WhatsApp*.

Ventajas:

- Muy popular y cada vez más utilizada por empresas importantes, que dan pie a que las empresas más pequeñas hagan uso de la aplicación también.
- Fácil de utilizar, ya que la gran mayoría de usuarios han tenido previamente contacto con la app *WhatsApp*.
- Se integra con *WhatsApp*, la aplicación de mensajería más utilizada mundialmente actualmente.
- Su principal pilar es la interacción de los negocios con sus clientes. Los servicios ofrecidos son todos los que el chat permite: atención al cliente, servicio técnico, etc.
- Totalmente gratuita.

Desventajas:

- La app no proporciona funcionalidad para gestionar citas o pedidos. Para ello habría que utilizar otra app de terceros.
- No es posible tener cuenta en *WhatsApp Business* y en *WhatsApp* simultáneamente con el mismo número de teléfono, por lo que sería necesario un número específico para el negocio.

Existen otras apps muy populares, como es el caso de *Facebook* [54], que integran algunas funcionalidades dirigidas especialmente a negocios, como por ejemplo: creación de páginas y eventos específicamente para un negocio registrado, facilitación del contacto de los usuarios de la red social con el negocio, consulta de su ubicación en el mapa y creación de publicaciones con información sobre novedades para los clientes. Sin embargo, todas esas características son una pequeña parte de una red social cuyo propósito es mucho más general que ese.

Como conclusión del estudio de alternativas, se podría decir que la idea a desarrollar es una buena decisión en cuanto a innovación y al intento de juntar varias herramientas útiles para negocios y clientes en una misma app, dado que como se ha visto, existen apps no muy populares que ofrecen algunas de las características que se pretenden abordar en el proyecto, pero no todas en un mismo sitio, y con aspectos que se pueden mejorar notablemente.

1.4. Nicho de mercado y usuarios objetivo

El principal segmento de mercado al que se dirige esta app se encuentra en los dueños de negocio que tengan un sistema de citas o pedidos o ambos, aunque para los casos menos comunes en los que no se disponga ninguno de estos sistemas, también sería útil para dar a conocer la ubicación del establecimiento del negocio y publicar anuncios con información de interés para los clientes.

Por tanto, el grupo de usuarios objetivo está compuesto por los propios dueños de negocios que utilizarán la app con ese rol, además de los clientes de dichos negocios o cualquier otro usuario que tenga interés en conocer novedades sobre los diversos negocios registrados en la app. El objetivo es que la app sea accesible y fácil de utilizar, para así poder ampliar lo máximo posible este grupo de usuarios finales.

1.5. Objetivos

1.5.1. Objetivos de desarrollo

El principal objetivo que se persigue con el desarrollo de este proyecto es la creación de una app móvil para Android, usable, que disponga principalmente de las siguientes características o funcionalidades:

- Gestión de las citas de un negocio.
- Gestión de los pedidos de un negocio.
- Interacción entre el propio negocio y sus clientes, mediante solicitudes de citas o pedidos y la respuesta a las mismas.
- Mapa interactivo con las ubicaciones de los establecimientos de los negocios registrados.
- Tablón de anuncios formado por publicaciones creadas por los distintos negocios registrados.

Otras características interesantes incluidas en la idea inicial, pero que debido al alcance del proyecto no se podrán incluir, son:

- Notificación a los clientes de retraso en citas previamente registradas.
- Gestión de usuarios con rol “dueño de negocio” mediante un rol de “administrador”.

1.5.2. Objetivos personales

Dado que el presente proyecto, enmarcado en un Trabajo de Fin de Grado, se encuentra en un contexto académico, también existen ciertos objetivos de formación personal:

- Descubrir cómo es un proyecto de desarrollo software "real" partiendo desde cero, pasando por todas las fases del mismo, desde elicitación de requisitos hasta el despliegue de la aplicación.
- Poner en práctica y entender mejor un proceso de desarrollo ágil basado en Scrum, cada vez más popular en las empresas del sector.
- Mejorar mi técnica y nivel de programación en Android, y aprender a utilizar nuevos componentes y elementos de Android que no había utilizado anteriormente.
- Aprender a realizar una app con una interfaz de usuario usable y que ofrezca una buena experiencia de usuario.
- Aplicar buenas prácticas en las distintas fases del desarrollo del proyecto.

1.6. Estructura de la memoria

El presente documento se estructura de la siguiente forma:

Capítulo 1: Introducción. En este capítulo se describe el origen de la idea del proyecto, qué motivación hay para desarrollarla, algunas alternativas ya existentes y qué objetivos se pretenden conseguir con el proyecto.

Capítulo 2: Planificación y requisitos. En este capítulo se presenta el método de planificación elegido y cómo se adapta al proyecto. También se expondrán el plan de riesgos y los presupuestos asociados al proyecto, pertenecientes a su planificación. Para finalizar se presentan los requisitos iniciales del proyecto, obtenidos mediante la metodología de planificación escogida.

Capítulo 3: Tecnologías utilizadas. En este capítulo se enumeran y explican las distintas tecnologías y herramientas empleadas en el desarrollo del proyecto a través de todas sus fases.

Capítulo 4: Análisis. En este capítulo se expone la fase de análisis llevada a cabo inicialmente a partir de los requisitos iniciales, recogidos y explicados en el capítulo de planificación y requisitos.

Capítulo 5: Diseño. En este capítulo se presentan las decisiones de diseño tomadas, y los diagramas y modelos elaborados que mejor representan el sistema creado.

Capítulo 6: Implementación y pruebas. En este capítulo se explicarán algunos aspectos relacionados con la implementación del sistema, una explicación de la integración

continua que se ha llevado a cabo y las distintas pruebas que se han realizado para la aplicación. También se expondrán las principales dificultades en la implementación surgidas a lo largo del proyecto.

Capítulo 7: Seguimiento del proyecto. En este capítulo se presenta cómo ha avanzado el proyecto, qué incidencias y eventos han ocurrido y un detalle de los distintos pasos seguidos para el desarrollo del mismo.

Capítulo 8: Conclusiones. En este capítulo se expondrán las conclusiones finales obtenidas, y algunas propuestas que se podrían llevar a cabo en el futuro del proyecto.

Finalmente, se incluyen unos anexos con otro contenido como el manual de usuario, despliegue, y otros enlaces de interés.

Capítulo 2

Planificación y requisitos

2.1. Scrum

El marco de trabajo elegido para la realización de este proyecto ha sido **Scrum**. Scrum se enmarca dentro de los procesos ágiles de desarrollo de software, que tal y como se declaró en 2001 en el *Manifiesto ágil* [69], se caracterizan por cuatro valores fundamentales:

- Individuos e interacciones por encima de procesos y herramientas.
- Software funcionando por encima de documentación extensiva.
- Colaboración con el cliente por encima de negociación de contrato.
- Respuesta ante el cambio por encima de seguir un plan.

Más a fondo, Scrum [29] es un marco de referencia en el que un equipo único y reducido desarrolla, entrega y mantiene productos complejos a través de un enfoque iterativo e incremental en bloques temporales cortos y fijos, gestionando en lo posible la incertidumbre y por tanto el riesgo asociado a la misma. El proceso se basa en tres pilares [68]:

- **Transparencia.** El trabajo debe ser visible para los que lo realizan así como para los que lo reciben, dado que muchas decisiones importantes se basan en la percepción del mismo.
- **Inspección.** El progreso del trabajo y el cumplimiento de objetivos debe ser revisado frecuentemente, para evitar problemas o sesgos indeseados en el producto
- **Adaptación.** Si el trabajo, tanto el proceso en sí como el producto, se desvía de lo deseado, se debe ajustar tan pronto como sea posible para evitar más desviación.

Es por ello que Scrum es adecuado para proyectos con un carácter exploratorio y/o con una alta incertidumbre, es decir, que pueden ser propensos a sufrir muchos cambios, y que requieren de continua retroalimentación por tratarse de proyectos complejos. Estos proyectos suelen implicar también equipos multifuncionales altamente cualificados, un entorno de desarrollo hipercompetitivo y que se esté trabajando con productos tecnológicos de vanguardia.

Dentro de Scrum, existen tres componentes esenciales para su desarrollo, que se explicarán en las siguientes subsecciones: **roles**, **eventos** y **artefactos**.

2.1.1. Roles

El equipo Scrum es un grupo pequeño auto-organizado (internamente eligen la mejor forma de realizar su trabajo) y multifuncional (disponen de todas las habilidades necesarias para crear valor, sin depender de alguien externo al equipo). Dentro del equipo, se distinguen tres roles [68]:

- **Product Owner.** Es el propietario de la iniciativa y el responsable de maximizar el valor del producto. Toda la organización debe respetar sus decisiones, que se reflejan en el *Product Backlog*, artefacto que se explicará en la subsección correspondiente (Sección 2.1.3).
- **Scrum Master.** Es el que se encarga de impulsar todas las prácticas Scrum en el equipo y el responsable de la eficacia del mismo. Además, es el punto de comunicación entre el equipo y toda la organización.
- **Equipo de desarrollo.** Son el resto de personas del equipo Scrum, responsables de desarrollar el producto para aportar valor en cada *incremento*, artefacto que se explicará en la Sección 2.1.3, dedicada a los artefactos.

En la Figura 2.1 se puede observar de manera más gráfica los distintos roles involucrados en el equipo Scrum y las relaciones entre ellos.

2.1.2. Eventos

En el marco de trabajo de Scrum existen diversos eventos, que tienen lugar con el objetivo de cumplir con los tres pilares principales explicados anteriormente. Estos eventos, de periodicidad y duración definida, tratan de minimizar la necesidad de reuniones adicionales no definidas en el marco original. Son los siguientes [68]:

- **Sprint.** Es el contenedor para el resto de eventos de Scrum. Los sprints son períodos de tiempo de duración fijada, entre 1 y 4 semanas, que ocurren secuencialmente uno

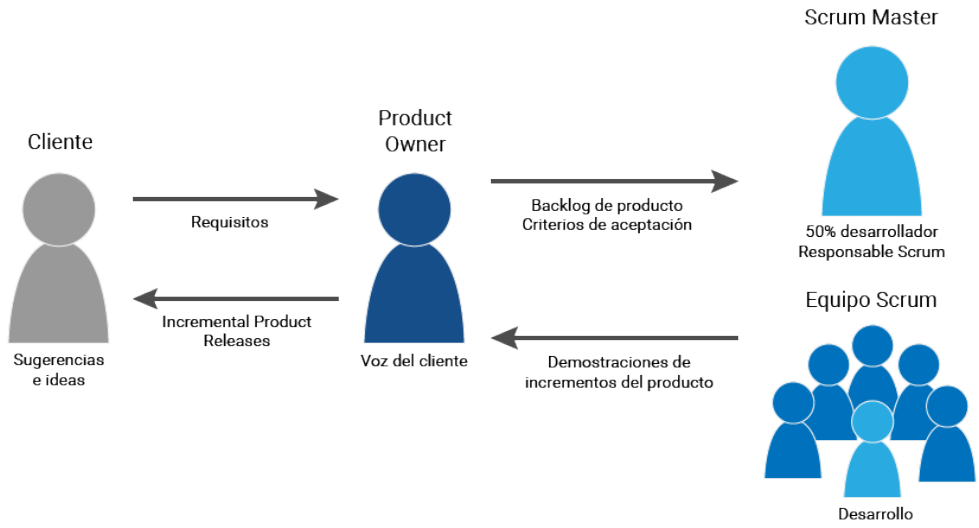

Figura 2.1. Roles en el equipo Scrum y sus relaciones [129]

tras otro a lo largo de todo el proceso de desarrollo del proyecto. Durante un sprint, los requisitos no se deben cambiar, pero pueden servir para aprender y hacer cambios en sprints futuros. El artefacto resultante del desarrollo de un sprint es un *incremento* (Sección 2.1.3).

- **Sprint Planning.** Es el evento que da comienzo a un sprint y en el que se planifica cuál será el contenido y los objetivos del sprint. En esta reunión participan todos los componentes del equipo Scrum, y se seleccionan los ítems del *Product Backlog* (Sección 2.1.3), ya priorizados, que se pretenderán realizar en el sprint que se está planificando. Para ello, a las distintas tareas se les asigna una estimacion basada en puntos de historia, que a su vez se traduce en tiempo estimado de trabajo. El resultado de este evento es el *Sprint Backlog* (Sección 2.1.3).

- **Daily Scrum.** Es una reunión diaria de alrededor de 15 minutos en la que cada componente del equipo de desarrollo informa a los demás sobre su progreso desde la última reunión Daily Scrum, y el plan de trabajo hasta la próxima Daily Scrum. El objetivo es mejorar la comunicación y solventar cualquier impedimento que pueda ocurrir a los miembros del equipo de desarrollo.

- **Sprint Review.** Es una reunión que ocurre al final del sprint, en la que el equipo Scrum al completo y los clientes discuten sobre los objetivos que se han cumplido en el sprint, para poder decidir qué hacer en el siguiente. Si es necesario, puede haber ajustes en el *Product Backlog* (Sección 2.1.3).

- **Sprint Retrospective.** Es el evento que finaliza el sprint. En él, se evalúa el sprint al que dará fin, de manera que se saquen conclusiones sobre qué ha ido bien, qué problemas se han encontrado y qué cambios se pueden hacer para mejorar en sprints futuros.

2.1.3. Artefactos

El resultado de las distintas actividades en Scrum se materializa en artefactos, que representan trabajo o valor. El objetivo de los mismos es maximizar la transparencia de la información clave para todas las partes involucradas. Los artefactos considerados en Scrum son [68]:

- **Product Backlog.** Es una lista ordenada por prioridad de las características y mejoras que se desean añadir al producto, en forma de historias de usuario. El principal responsable del mismo es el *Product Owner*, como ya se ha visto, y es variable a lo largo del proyecto. El objetivo que se persigue es el *Product Goal*, que define un estado futuro del producto al que se desea llegar.
- **Sprint Backlog.** Es la lista de historias de usuario pertenecientes al *Product Backlog* en las que el equipo de desarrollo pretende trabajar en un sprint. Los responsables de este artefacto son los componentes del equipo de desarrollo, y es el que utilizan para actualizar su situación en las *Daily Scrum*. La meta a alcanzar en este artefacto es el *Sprint Goal*, que define el objetivo a cumplir en el sprint, pero que a su vez es flexible en términos del trabajo realizado para conseguirlo.
- **Incremento.** Es el resultado del trabajo realizado en un sprint. Cada incremento es acumulativo con los incrementos de sprints anteriores, fusionando todos ellos para acercarse al objetivo final actual del proyecto, el *Product Goal*. Los responsables del mismo son los miembros del equipo de desarrollo. El incremento debe cumplir con las condiciones establecidas para ser utilizable, dado que será lo que se entregue al cliente al final del sprint. Estas condiciones conforman la *Definition of Done* o *Definición de Hecho*, que es el objetivo a perseguir con la realización de cada incremento.

En la Figura 2.2 aparece un resumen gráfico de los eventos y artefactos involucrados en el proceso de Scrum.

2.2. Adaptación de Scrum al proyecto

La decisión de utilizar Scrum como marco de trabajo para este proyecto se basa en varios factores. Primero, como se trata de un proyecto que surge de una idea con unos requisitos no muy definidos desde el principio, la flexibilidad que proporciona Scrum es ideal para los cambios que puedan producirse a lo largo del mismo. También, el seguimiento del proyecto por parte de la tutora será constante semanalmente, asegurando una retroalimentación que permitirá mejorar según se progrese. Por otro lado, Scrum es ahora mismo una de las formas de desarrollo ágil más aplicadas en las empresas del sector, por lo que aplicarlo también a este proyecto permitirá poner en práctica y aprender una de las formas de trabajo más demandadas en la actualidad.

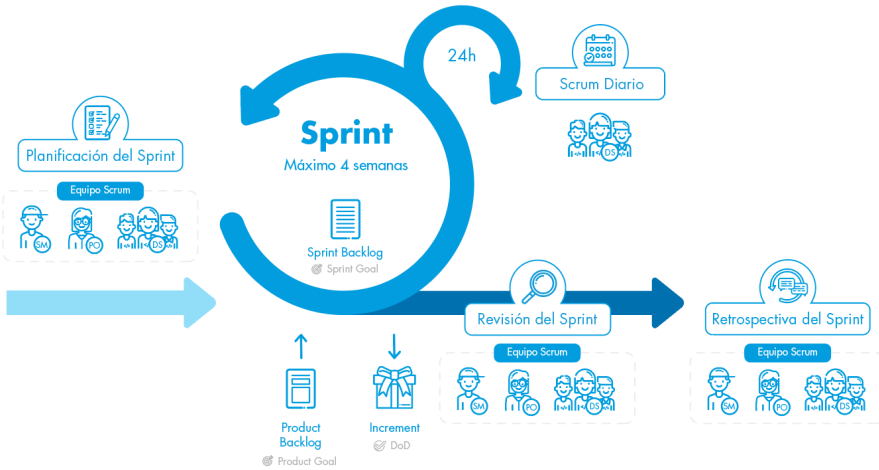

Figura 2.2. Eventos y artefactos en Scrum [83]

La flexibilidad de Scrum también permite aplicarlo como marco de referencia en diversos contextos, poniéndose de manifiesto su adaptabilidad. En este proyecto, debido a sus características especiales (como la limitación de personal), se ha tenido que aplicar una adaptación de Scrum.

En cuanto a roles, dado que sólo dos miembros estarán involucrados en el desarrollo del proyecto (estudiante autor del TFG y tutora), la adaptación ha sido la siguiente: el estudiante cumplirá simultáneamente con los papeles de *Product Owner* y equipo de desarrollo, mientras que la tutora cumplirá con el papel de *Scrum Master*. De esta manera, el estudiante se encargará de la idea del producto y la priorización de sus características, así como del desarrollo del mismo, y la tutora se encargará de revisar que se cumple con las prácticas Scrum de forma correcta y de asesorar al estudiante en distintos aspectos.

En cuanto a eventos, los sprints del proyecto tendrán una duración de 2 semanas. Las reuniones de *Sprint Planning*, *Sprint Review* y *Sprint Retrospective* tendrán lugar cada 2 semanas, justo en el día en el que se termina un sprint y se comienza el siguiente. El día elegido para hacerlo han sido los miércoles. Las reuniones *Daily Scrum* se han convertido en *Weekly Scrum*, que se realizarán una vez por semana, en las semanas que no se empiece ni finalice un sprint. Es decir, todos los miércoles se celebrará una reunión entre el estudiante y la tutora, y dependiendo del punto temporal del sprint en el que se encuentre, dicha reunión tendrá un propósito u otro.

En cuanto a artefactos, tanto *Product Backlog* como *Sprint Backlog* como incremento, estarán a cargo del estudiante, ya que asume todos los roles responsables de los mismos.

2.3. Planificación inicial

Se ha establecido una planificación inicial con un calendario de sprints. Dada la fecha de inicio del proyecto y que según la guía docente [115] la realización del Trabajo de Fin de Grado se corresponde con 12 ECTS, es decir, 300 horas, se ha decidido elaborar un plan inicial basado en **8 sprints**. Como cada sprint tiene una duración fija (2 semanas) pero la carga de trabajo asociada al mismo puede ser variable entre sprints, se prevé dedicar aproximadamente de 15 a 25 horas semanales, lo que resulta en **sprints de 30 a 50 horas**, en función del tiempo disponible debido a otros motivos académicos. Esto se decide en la reunión *Sprint Planning* de preparación de cada sprint, y el método que se ha utilizado para ello son los puntos de historia. A cada historia de usuario perteneciente al *Product Backlog* se le asignará una estimación de puntos de historia para poder ajustar el *Sprint Backlog* correspondiente al tiempo disponible. En este caso, la equivalencia utilizada ha sido de **1 punto de historia $\simeq$ 5 horas**.

La planificación inicial incluye un **sprint 0**, que se diferencia de los demás por tener una duración distinta. El objetivo de este sprint es dedicar las primeras semanas del proyecto a labores de preparación y planificación, para en el sprint 1 dar paso al desarrollo del proyecto como tal. Asimismo, se incluyen al final **2 sprints extra** de refuerzo, a modo de ofrecer un margen en caso de que el trabajo deseado no se haya podido completar en el transcurso de los 8 sprints principales, y para hacer retoques finales tanto en documentación como en material presentado (código, modelos, etc.).

Dado que los sprints tendrán una carga de trabajo de 30 a 50 horas, dependiendo de las circunstancias en el sprint, si tomamos como promedio que los sprints tendrán una carga de 40 horas cada uno, en total se sumarían 320 horas. Añadiendo además el tiempo empleado en el sprint 0, que es en el que se ha desarrollado esta planificación y por tanto se sabe que tuvo una carga de alrededor de 20 horas, la estimación inicial queda en **340 horas**, que satisfaría las 300 horas requeridas. En caso de necesidad, se utilizaría uno o los dos sprints extra de refuerzo, que considerando también un promedio de 40 horas para cada uno, elevaría la estimación inicial a **380 horas** en caso de utilizar uno y a **420** horas en caso de utilizar los dos, cumpliendo con creces con la cantidad de horas requeridas y dejando un margen muy amplio para los inconvenientes que puedan surgir a lo largo del proyecto.

En la Tabla 2.1 se puede observar una calendarización de los sprints y eventos planificados para el transcurso del proyecto.

2.4. Plan de riesgos

En la fase de planificación, también se ha elaborado un plan de riesgos. Como en todo proyecto, tras hacer un plan inicial, hay un componente de incertidumbre que hay que gestionar, pues todo se basa en suposiciones y pueden darse situaciones que hagan que esas suposiciones se vuelvan incorrectas.

Un **riesgo** [29] es un suceso relacionado con el futuro que involucra causa y efecto. En este contexto, la ocurrencia del riesgo afectará al desarrollo del proyecto. Es por ello que es

Nº Sprint/Evento	Fecha inicio	Fecha fin	Anotaciones
Sprint 0	19/01/2021	10/02/2021	
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	10/02/2021		
Sprint 1	10/02/2021	24/02/2021	
Scrum Weekly	17/02/2021		
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	24/02/2021		
Sprint 2	24/02/2021	10/03/2021	
Scrum Weekly	03/03/2021		
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	10/03/2021		
Sprint 3	10/03/2021	24/03/2021	
Scrum Weekly	17/03/2021		
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	24/03/2021		
-	24/03/2021	06/04/2021	Período de vacaciones de Semana Santa
Sprint 4	06/04/2021	21/04/2021	
Scrum Weekly	14/04/2021		
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	21/04/2021		
Sprint 5	21/04/2021	05/05/2021	
Scrum Weekly	28/04/2021		
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	05/05/2021		
Sprint 6	05/05/2021	19/05/2021	
Scrum Weekly	12/05/2021		
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	19/05/2021		
Sprint 7	19/05/2021	02/06/2021	
Scrum Weekly	26/05/2021		
Sprint Review, *Sprint Retrospective* & *Sprint Planning*	02/06/2021		
Sprint 8	02/06/2021	16/06/2021	
Scrum Weekly	09/06/2021		
Sprint Review & *Sprint Retrospective*	16/06/2021		También se realizará *Sprint Planning* en caso de utilizar el próximo sprint
Sprint extra 1	16/06/2021	30/06/2021	
Scrum Weekly	23/06/2021		
Límite solicitud defensa convocatoria ordinaria	28/06/2021		Los tribunales se realizarán **antes del 12/07/2021**
Sprint Review & *Sprint Retrospective*	30/06/2021		Se podrá adelantar en caso de presentar el TFG en convocatoria ordinaria. También se realizará *Sprint Planning* en caso de utilizar el próximo sprint
Sprint extra 2	30/06/2021	14/07/2021	
Scrum Weekly	07/07/2021		
Sprint Review & *Sprint Retrospective*	14/07/2021		
Límite solicitud defensa convocatoria extraordinaria	19/07/2021		Los tribunales se realizarán **antes del 28/07/2021**

Tabla 2.1. Planificación inicial de sprints

importante la elaboración de un plan de riesgos, cuyo objetivo es controlar en lo posible la ocurrencia y las consecuencias de los mismos.

En este caso, para la identificación y análisis de los riesgos, se han tenido en cuenta cuatro elementos asociados a cada riesgo:

- **Probabilidad.** Se trata de la posibilidad de que el riesgo se pueda materializar. Se le asigna uno de los siguientes valores: {baja, media, alta}.
- **Impacto.** Se trata del nivel del efecto que produciría el riesgo en caso de materializarse. Se le asigna uno de los siguientes valores: {bajo, medio, alto}.
- **Plan de mitigación.** Se trata de una serie de acciones que se pueden llevar a cabo para reducir la probabilidad de que el riesgo ocurra.
- **Plan de contingencia.** Se trata de una serie de acciones que se pueden llevar a cabo una vez que el riesgo se presenta, para minimizar en lo posible su impacto.

De esta manera, además de identificar cada riesgo, se dispone de una lista de acciones a llevar a cabo a priori y a posteriori para gestionar la ocurrencia de los mismos y así minimizar el efecto final que puedan producir en el correcto desarrollo del proyecto.

Los riesgos identificados se han dividido en dos categorías: **riesgos generales**, que podrían ocurrir en cualquier contexto, independientemente del proyecto que se llevará a cabo, y **riesgos particulares**, que tienen en cuenta las características específicas de este proyecto.

Los riesgos generales se presentan en las Tablas 2.2 a 2.8, y los riesgos particulares en las Tablas 2.9 a 2.10.

Riesgo R01	
Título	Ausencia temporal del estudiante por enfermedad
Descripción	El estudiante no puede dedicar tiempo a trabajar en el TFG durante un período determinado debido a causas médicas que le impidan trabajar con normalidad
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	▪ Llevar una vida saludable ▪ Evitar actividades que supongan un riesgo para la salud
Plan de contingencia	▪ Si la duración de la ausencia es unos días, tratar de recuperar el tiempo perdido distribuyéndolo entre los siguientes días ▪ Si la duración es más prolongada, pasar tareas al siguiente sprint ▪ Si ocurre en los últimos sprints y las dos anteriores no se pueden realizar, hacer uso de los sprints extra de refuerzo

Tabla 2.2. Riesgo R01: Ausencia temporal del estudiante por enfermedad

Riesgo R02	
Título	Fallos técnicos en el equipo de trabajo del estudiante
Descripción	El equipo de trabajo del estudiante sufre algún fallo técnico y no se puede continuar trabajando con normalidad
Probabilidad	Baja
Impacto	Medio
Plan de mitigación	▪ Hacer una copia de seguridad de los documentos importantes cada poco tiempo ▪ Mantener una copia de los documentos importantes en algún repositorio online ▪ Trabajar en lo posible de forma online con los documentos (Overleaf, Microsoft Teams...)
Plan de contingencia	▪ Si se ha realizado copias de seguridad de los documentos, recuperarlos desde otro dispositivo para seguir trabajando ▪ Si no se ha realizado copias de seguridad, tratar de recuperar los documentos con los mecanismos de recuperación que ofrecen los propios programas

Tabla 2.3. Riesgo R02: Fallos técnicos en el equipo de trabajo del estudiante

Riesgo R03	
Título	Ausencia temporal de la tutora por enfermedad
Descripción	La tutora no puede continuar con el seguimiento del TFG durante un período determinado debido a causas médicas que le impidan trabajar con normalidad
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	▪ Tener contacto asiduamente para poder saber si el riesgo se materializa en algún momento
Plan de contingencia	▪ Seguir trabajando en el TFG hasta poder tener la siguiente reunión con la tutora, e ir apuntando dudas o problemas que surjan ▪ Si es posible comunicarse mediante otros medios que no sean videoconferencia (chats, emails...), enviar dudas o un resumen del contenido que se vería en la reunión

Tabla 2.4. Riesgo R03: Ausencia temporal de la tutora por enfermedad

Riesgo R04	
Título	Confinamiento domiciliario debido a COVID-19
Descripción	Debido al empeoramiento de la situación sanitaria actual producida por el COVID-19, se produce una orden de confinamiento domiciliario
Probabilidad	Baja
Impacto	Bajo
Plan de mitigación	▪ Estar familiarizado con herramientas de comunicación telemática (video-conferencia, chats...)
Plan de contingencia	▪ Realizar el seguimiento con la tutora mediante videoconferencia u otras herramientas de comunicación en remoto

Tabla 2.5. Riesgo R04: Confinamiento domiciliario debido a COVID-19

Riesgo R05	
Título	Mala estimación de tiempo y trabajo de algunas actividades
Descripción	La estimación producida para algunas actividades o tareas a realizar en los sprints resulta ser muy sesgada y el tiempo empleado realmente no se asemeja a lo estimado
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	▪ Establecer uno o más sprints de refuerzo al final del proceso de desarrollo del proyecto que puedan cubrir estas necesidades de tiempo adicional ▪ Ser realista en las estimaciones. Para ello, mejorar con la práctica ▪ Dejar los detalles pequeños para el final y centrarse más en las partes importantes del proyecto
Plan de contingencia	▪ Hacer uso de los Sprints de refuerzo ▪ Si no hay disponibilidad de más sprints de refuerzo, recortar el alcance del proyecto en la medida de lo posible

Tabla 2.6. Riesgo R05: Mala estimación de tiempo y trabajo de algunas actividades

Riesgo R06	
Título	Desconocimiento del manejo de las herramientas que hay que utilizar
Descripción	Algunas de las herramientas que hay que utilizar no se conocen y por tanto hay que aprender a utilizarlas
Probabilidad	Media
Impacto	Medio
Plan de mitigación	▪ Familiarizarse lo antes posible (en el sprint 0 idealmente) con las herramientas que no se sepan manejar
Plan de contingencia	▪ Buscar documentación oficial o tutoriales donde se explique el manejo de dichas herramientas ▪ Tratar de buscar alternativas factibles y ya conocidas para utilizar en lugar de la nueva herramienta

Tabla 2.7. Riesgo R06: Desconocimiento del manejo de las herramientas que hay que utilizar

Riesgo R07	
Título	Menor tiempo empleado que el previsto debido a otros asuntos académicos
Descripción	En algún sprint no se puede dedicar tanto tiempo como se ha estimado inicialmente en la reunión de planificación debido a otros eventos académicos que se solapen con la duración del sprint (prácticas en empresa, exámenes, entregas...)
Probabilidad	Alta
Impacto	Medio
Plan de mitigación	▪ Tratar de planificar cada sprint sabiendo qué eventos académicos sucederán durante el tiempo que dure el sprint
Plan de contingencia	▪ Pasar tareas sin finalizar al sprint siguiente ▪ Recuperar el tiempo no empleado en futuros sprints, cuando se disponga de más tiempo

Tabla 2.8. Riesgo R07: Menor tiempo empleado que el previsto debido a otros asuntos académicos

Riesgo R08	
Título	Cambio en los requisitos del proyecto
Descripción	Los requisitos del proyecto cambian por parte del cliente (los dueños de negocio) o por parte del estudiante, que asume el cargo de *Product Owner*, cuando el desarrollo ya ha sido iniciado
Probabilidad	Media
Impacto	Bajo
Plan de mitigación	▪ Intentar aclarar desde un principio cuáles son las funcionalidades propuestas y tener en cuenta las opiniones al respecto de los clientes ▪ Hacer una entrega continua y validación por parte de los usuarios, y en este caso por parte de la tutora también
Plan de contingencia	▪ Si es un cambio estético, de IU o similar, se puede llevar a cabo ▪ Si es un cambio que supone mucho trabajo, en principio no se efectuaría ya que el objetivo final del TFG no es comercializarlo en principio

Tabla 2.9. Riesgo R08: Cambio en los requisitos del proyecto

Riesgo R09	
Título	Necesidad de aprendizaje de nuevas herramientas debido a actualizaciones en las tecnologías utilizadas
Descripción	Durante el transcurso del proyecto algunas de las herramientas o tecnologías utilizadas (API Android, programas utilizados...) reciben una actualización y cambian su aspecto y/o comportamiento
Probabilidad	Baja
Impacto	Bajo
Plan de mitigación	▪ Definir desde un principio las versiones de las herramientas y tecnologías que se van a utilizar. En el Capítulo 3, dedicado a las tecnologías empleadas en el proyecto, se detallan las versiones de los distintas herramientas utilizadas
Plan de contingencia	▪ Si se llega a actualizar la herramienta a una nueva versión con características que supongan problemas, volver a la versión inicial

Tabla 2.10. Riesgo R09: Necesidad de aprendizaje de nuevas herramientas debido a actualizaciones en las tecnologías utilizadas

2.5. Plan de presupuestos

Como última parte de la fase de planificación, se ha realizado un plan presupuestario del proyecto. Al tratarse de un contexto académico y no estar orientado al mundo real, como podría ser el caso de cualquier proyecto en cualquier empresa del sector, se han elaborado dos presupuestos de manera diferenciada: un **presupuesto simulado**, que trata de estimar los costes que supondría este proyecto si se tratase de un proyecto real dirigido al mercado, y un **presupuesto real**, en el que se estiman los costes que realmente tendrán lugar al tratarse de un Trabajo de Fin de Grado.

2.5.1. Presupuesto simulado

El primer elemento que se ha tenido en cuenta para calcular el presupuesto es el propio trabajador. Consultando en dos fuentes diferentes [64] [114], cada una de las cuales contrasta la información contra otras tantas fuentes, se ha obtenido un sueldo anual similar para un desarrollador Android en España en la fecha de consulta. Por ello, y a modo de estimación, se ha considerado un sueldo de 2300€ al mes para un desarrollador Android estándar (no refiriéndose a un puesto *Junior* ni *Senior*). Como en un mes aproximadamente hay 22 días laborables, cada uno con 8 horas de trabajo, lo que supone 176 horas/mes, la hora de un desarrollador Android es pagada a 13,07€. Como en la planificación inicial antes mostrada (Sección 2.3) se estima que el trabajo constará en principio de 340 horas, el precio total a pagar por el desarrollador será de **4443,80€**. De esos 4443,80€ (sueldo neto) se ha descontado previamente un 33,4 % aproximadamente [70] que va dirigido a la Seguridad Social del trabajador, a cargo de la empresa. El sueldo bruto por tanto sería de 6672,37€, y el total que va destinado a la Seguridad Social del empleado es de **2228,57€** (diferencia entre ambas cifras).

Siguiendo por el equipo del trabajador, se ha escogido un *MacBook Pro 2020* [23]. Si el precio del equipo completo es 2129€ y se estima que su vida útil es de 4 años (48 meses), el precio aproximado del equipo es de 44,35€/mes. Como la duración estimada del proyecto es de 5 meses, el coste amortizado que supondrá es de **221,75€**. Asimismo, para realizar pruebas de la app en un dispositivo físico, será necesario un smartphone Android. El dispositivo elegido ha sido *Samsung Galaxy S10+* [88]. Estimando su vida útil en 48 meses también, y si el precio total es de 649€, su coste es de 13,52€/mes. Para los 5 meses de duración del proyecto, el coste amortizado será de **67,60€**.

Para el lugar de trabajo, se ha elegido un espacio de trabajo coworking en Valladolid [35], cuya tarifa se sitúa en 165€/mes. Para cubrir con los 5 meses de trabajo, el precio a pagar será de **825€**.

Finalmente, para las licencias de algunas de las aplicaciones de pago a utilizar, en el caso de *Microsoft 365* [78], se considerará la más básica para empresas, que incluye *Microsoft Teams*, herramienta utilizada para la comunicación entre el equipo de trabajo, y supone 4,12€/mes, es decir, **20,60€** para cubrir los meses de trabajo. En el caso de *Balsamiq Cloud* [27], aplicación usada para la creación de los prototipos de la app, la licencia tiene un precio de 7,42€/mes, resultando en un total de **37,10€**. Por último, la aplicación que se utilizará

para la creación de los distintos modelos es *Astah Professional* [25], que ofrece una licencia anual por 140€, de la que se obtiene que el precio por un mes es 11,67€y por tanto el coste total para los 5 meses de trabajo es de **58,35€**. El resto de programas y herramientas utilizadas no tienen coste, o se utilizará su versión gratuita.

Sumando todo lo anterior, el coste total asciende a **7902,77€**. A modo de colchón, se elevará dicha cifra un 25 %, para poder hacer frente a gastos imprevistos como alguna herramienta nueva que haya que utilizar a lo largo del proyecto y requiera licencia, o simplemente el retraso de la fecha de finalización del proyecto y el aumento en el coste de cada uno de los elementos que eso conlleva, pues todos son calculados en función del tiempo. El presupuesto total final resultante es de **9878,46€**.

En la Tabla 2.11 se refleja un resumen del presupuesto simulado con los elementos desglosados por filas.

Concepto	Precio unitario	Cantidad	Total
Hora de trabajo de desarrollador Android	13,07€/hora	340 horas	4443,80€
Seguridad Social del empleado	6,55€/hora	340 horas	2228,57€
Equipo del empleado (*MacBook Pro 2020*)	44,35€/mes	5 meses	221,75€
Smartphone de pruebas (*Samsung Galaxy S10+*)	13,52€/mes	5 meses	67,60€
Espacio de trabajo coworking	165€/mes	5 meses	825€
Licencia *Microsoft 365 Empresa Básico*	4,12€/mes	5 meses	20,60€
Licencia *Balsamiq Cloud*	7,42€/mes	5 meses	37,10€
Licencia *Astah Professional*	11,67€/mes	5 meses	58,35€
TOTAL			7902,77€
+25 % de normalización			**9878,46€**

Tabla 2.11. Presupuesto simulado

2.5.2. Presupuesto real

En el presupuesto real, a diferencia del simulado, no se aplican los elementos relacionados con el sueldo del empleado, ya que se trata de un estudiante realizando el Trabajo de Fin de Grado. Tampoco se tendrá en cuenta el espacio de trabajo, ya que el desarrollo del proyecto se llevará a cabo desde casa en su totalidad, o posiblemente puntualmente en el edificio de la Escuela de Ingeniería Informática, al que se tiene acceso como estudiante.

Sin embargo, al trabajar desde casa, sí que se tendrá en cuenta el gasto de la electricidad que produce el equipo de trabajo del estudiante, ya que es un gasto que se produce realmente. Si el consumo medio de un portátil en 1 hora es de 0,11 kWh [34], en 340h será de 37,4 kWh. Considerando que se trabajará normalmente de 16h a 20h cada día, el precio medio del kWh en ese período es de aproximadamente 0,065€/kWh [113], resultando en un coste de **2,43€**.

El equipo del estudiante, un *HP Notebook* [63], tiene un coste total de 749€. Si se estima que su vida útil es de 4 años (48 meses), el precio aproximado del equipo es de 15,60€/mes. Para cubrir los 5 meses de duración estimada del proyecto, el coste amortizado será de

78€. De la misma manera que en el presupuesto simulado, se utilizará un smartphone para pruebas. Se utilizará también un *Samsung Galaxy S10+*, cuyo precio ya ha sido calculado en el presupuesto simulado, ascendiendo a **67,60€**.

En cuanto a licencias de programas utilizados, no se incluye en el presupuesto ninguna de ellas, dado que tanto en el caso de *Microsoft 365*, como en el de *Balsamiq*, como en el de *Astah Professiomal*, la licencia es ofrecida de forma gratuita a los estudiantes a través de la universidad.

Tampoco se incluye en este contexto un colchón al coste total, dado que es relativamente bajo y no se pretende invertir en elementos adicionales. Por tanto, el presupuesto total final es de **148,03€**.

En la Tabla 2.12 se muestra un resumen del presupuesto real con el desglose de elementos.

Concepto	Precio unitario	Cantidad	Total
Electricidad	0,065€/kWh	37,4 kWh	2,43€
Equipo del estudiante (*HP Notebook*)	15,60€/mes	5 meses	78€
Smartphone de pruebas (*Samsung Galaxy S10+*)	13,52€/mes	5 meses	67,60€
TOTAL			**148,03€**

Tabla 2.12. Presupuesto real

2.6. Product Backlog inicial

Tal y como se explicaba en la Sección 2.2, es el alumno el que en este caso asume el papel de *Product Owner*, y por tanto, el encargado de elaborar el *Product Backlog*. Mediante el desarrollo del concepto inicial de la app con aportaciones o sugerencias de la tutora, y diversas entrevistas a potenciales usuarios reales (se explicará más a fondo en el Capítulo 7 de seguimiento del proyecto), se ha elaborado el *Product Backlog* presente en la Tabla 2.13.

Al estar trabajando con Scrum, los requisitos tienen forma de historias de usuario, como ya se ha explicado. Estas historias tienen la forma: "Como <*stakeholder*> quiero <*objetivo*> para <*razón que aporte valor como stakeholder*>", donde el término *stakeholder* hace referencia a una entidad interesada en el proyecto. Sin embargo, al tratarse de una lista de requisitos iniciales y muy generales, se ha optado por escribirlos en forma de **épicas**, que no son más que historias de usuario muy grandes que se pueden subdividir a su vez en más historias de usuario. La división de las distintas épicas en historias de usuario se llevará a cabo durante el transcurso del proyecto, siguiendo ciertas recomendaciones, guías y ejemplos [80] [82], como una de las tareas de análisis.

Número	Épica
1	Como dueño de negocio quiero gestionar las citas de los clientes en mi establecimiento para poder tener una gestión centralizada del mismo junto a los pedidos en un solo sitio
2	Como dueño de negocio quiero gestionar los pedidos de los clientes en mi establecimiento para poder tener una gestión centralizada del mismo junto a las citas en un solo sitio
3	Como cliente quiero hacer solicitudes de citas a los establecimientos registrados para facilitar la tarea de pedir cita y poder tener centralizados todos los establecimientos en un solo sitio
4	Como cliente quiero hacer solicitudes de pedidos a los establecimientos registrados para facilitar la tarea de hacer pedidos y poder tener centralizados todos los establecimientos en un solo sitio
5	Como cliente quiero ver si mis solicitudes de cita o pedido han sido confirmadas por el establecimiento para poder asegurar el servicio que he solicitado
6	Como dueño de negocio quiero poder publicar en el mapa interactivo de establecimientos la ubicación del mío para permitir a los usuarios encontrarlo más fácilmente
7	Como cliente quiero ver en el mapa interactivo los distintos establecimientos registrados y su ubicación para poder llegar a ellos más fácilmente
8	Como dueño de negocio quiero poder hacer publicaciones sobre mi negocio en el tablón de anuncios para proporcionar información a los clientes
9	Como cliente quiero consultar anuncios publicados por los establecimientos en el tablón de anuncios para recibir distintas informaciones de los mismos
10	Como dueño de negocio quiero notificar a los clientes con citas previas si existe algún retraso para evitar esperas innecesarias de los clientes y aglomeraciones o saturación en el establecimiento
11	Como cliente quiero que se me notifique cuando exista un retraso en mis citas en los establecimientos para evitar esperas innecesarias
12	Como administrador quiero aprobar las solicitudes de registro de nuevos establecimientos para incluir nuevos negocios a la app
13	Como administrador quiero gestionar establecimientos para ofrecerles todas las funcionalidades de la app

Tabla 2.13. *Product Backlog* inicial

2.7. Product Backlog final

Tras diversos cambios durante el transcurso de los sprints del proyecto, y como bien permite Scrum modificar el *Product Backlog* inicial entre sprint y sprint, el aspecto final del *Product Backlog* es el que se muestra en la Tabla 2.14. En este caso se ha asignado un código identificativo a las épicas para poder referenciarlas más tarde en el documento, ya que son las que se utilizarán durante el desarrollo del proyecto.

Uno de los cambios llevados a cabo consiste en la introducción de una nueva épica (**EP06**), ya que previamente no existía una que se dirigiera expresamente a la gestión de usuarios y se decidió que era mejor tratarlo como un requisito independiente en lugar de considerarlo implícito en otras épicas.

El otro cambio consiste en el cambio de prioridad de las épicas relativas al tablón de

anuncios y al mapa interactivo (**EP07-10**). En el *Product Backlog* inicial, se dio más importancia a la característica del mapa, pero en el *Product Backlog* final se decidió dar más prioridad al tablón de anuncios, dado que si al final hubiera que recortar funcionalidad, era preferible ofrecer el tablón de anuncios antes que el mapa, ya que es más complejo y da más valor a la app. Asimismo, entre las dos épicas dedicadas a cada una de esas características, también se ha cambiado el orden de prioridad para facilitar el desarrollo: primero se llevará a cabo la tarea desde el lado del cliente y después desde el lado del dueño de negocio, ya que esta segunda tarea consiste en añadir funcionalidades a la primera, por lo que se trata de un orden más coherente.

Dado el contexto académico del proyecto y la limitación de tiempo, se decidió limitar el alcance del mismo al desarrollo de las épicas desde **EP01** a **EP10**.

Código	Épica
EP01	Como dueño de negocio quiero gestionar las citas de los clientes en mi establecimiento para poder tener una gestión centralizada del mismo junto a los pedidos en un solo sitio
EP02	Como dueño de negocio quiero gestionar los pedidos de los clientes en mi establecimiento para poder tener una gestión centralizada del mismo junto a las citas en un solo sitio
EP03	Como cliente quiero hacer solicitudes de citas a los establecimientos registrados para facilitar la tarea de pedir cita y poder tener centralizados todos los establecimientos en un solo sitio
EP04	Como cliente quiero hacer solicitudes de pedidos a los establecimientos registrados para facilitar la tarea de hacer pedidos y poder tener centralizados todos los establecimientos en un solo sitio
EP05	Como cliente quiero ver si mis solicitudes de cita o pedido han sido confirmadas por el establecimiento para poder asegurar el servicio que he solicitado
EP06	Como usuario quiero iniciar sesión en la app para identificarme como cliente o dueño de negocio y poder acceder a las características disponibles en función de mi rol
EP07	Como cliente quiero consultar anuncios publicados por los establecimientos en el tablón de anuncios para recibir distintas informaciones de los mismos
EP08	Como dueño de negocio quiero poder hacer publicaciones sobre mi negocio en el tablón de anuncios para proporcionar información a los clientes
EP09	Como cliente quiero ver en el mapa interactivo los distintos establecimientos registrados y su ubicación para poder llegar a ellos más fácilmente
EP10	Como dueño de negocio quiero poder publicar en el mapa interactivo de establecimientos la ubicación del mío para permitir a los usuarios encontrarlo más fácilmente
EP11	Como dueño de negocio quiero notificar a los clientes con citas previas si existe algún retraso para evitar esperas innecesarias de los clientes y aglomeraciones o saturación en el establecimiento
EP12	Como cliente quiero que se me notifique cuando exista un retraso en mis citas en los establecimientos para evitar esperas innecesarias
EP13	Como administrador quiero aprobar las solicitudes de registro de nuevos establecimientos para incluir nuevos negocios a la app
EP14	Como administrador quiero gestionar establecimientos para ofrecerles todas las funcionalidades de la app

Tabla 2.14. *Product Backlog* final

Capítulo 3

Tecnologías utilizadas

3.1. GitLab

GitLab [50] es un servicio web de código abierto utilizado para el desarrollo de software colaborativo. Entre sus principales objetivos se encuentra integrar múltiples funcionalidades para que sea la única aplicación necesaria durante el ciclo de vida completo del desarrollo de software, mejorando así la eficiencia y la velocidad en el trabajo. Es por ello que en este proyecto *GitLab* ha cumplido una triple función: sistema de control de versiones, herramienta de gestión del proyecto y herramienta de integración continua.

3.1.1. Git

El núcleo de funcionamiento de *GitLab* se basa en *Git*. ***Git*** [44] es un sistema de control de versiones distribuido de código abierto y gratuito, cuyo propósito es llevar un registro de los cambios que se producen en los ficheros de código fuente y configuración de un proyecto, principalmente software.

Git se basa en un sistema de repositorio y ramas. Los **repositorios** son espacios de almacenamiento virtuales para los distintos ficheros de un proyecto. Las **ramas** son copias exactas de todo el proyecto que se crean desde otra rama, en el estado que se encontrase dicho punto de creación al momento de crearla. De esta manera, si tenemos un código base de nuestro proyecto y queremos añadir una nueva característica al mismo, *Git* nos permite crear una nueva rama en nuestro repositorio (es decir, hacer una copia del proyecto) y añadir la nueva funcionalidad ahí, asegurando así que siempre tenemos una rama con una versión estable del proyecto. Esta rama original y estable se suele denominar `master`. En la Figura 3.1 se puede observar de manera más gráfica.

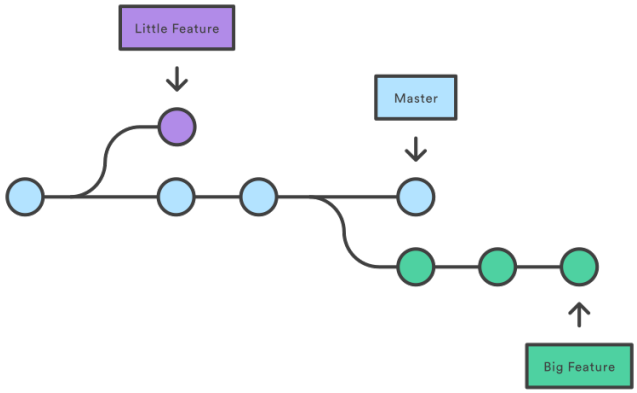

Figura 3.1. Sistema de ramas en *Git* [26]

Las operaciones básicas con ramas consisten en crear nuevas ramas y, una vez que se ha terminado de trabajar en ellas, fusionarlas de nuevo con la rama de la que se han originado. Esta fusión se denomina `merge` en términos de *Git*.

Además, *Git* permite **trabajo en local**, aparte de en remoto. Para ello, existe la posibilidad de clonar repositorios en el almacenamiento local, realizar cambios sobre ellos (incluida la gestión de ramas), y grabar lo que se ha trabajado en local en el repositorio remoto. De esa manera, los cambios estarán disponibles para otros miembros si se trata de trabajo colaborativo, o para poder acceder desde cualquier otro entorno local, en definitiva. Las operaciones utilizadas para ello son: `pull`, que trae los últimos cambios producidos en el repositorio remoto al local; `commit`, que guarda los cambios realizados en el proyecto en el repositorio local; y `push`, que lleva los últimos `commits` del repositorio local al remoto. Algo que hace diferente a *Git* sobre esto, es que dispone de una zona de preparación o *staging area* [44], que supone una zona intermedia donde los cambios realizados en local se pueden supervisar y corregir en caso de necesidad antes de completar la operación de `commit`. Aquí entra la operación `add`, que pasa los cambios locales seleccionados al *staging area*. En la Figura 3.2 se puede observar el proceso de grabación de cambios en el repositorio local pasando por el *staging area*.

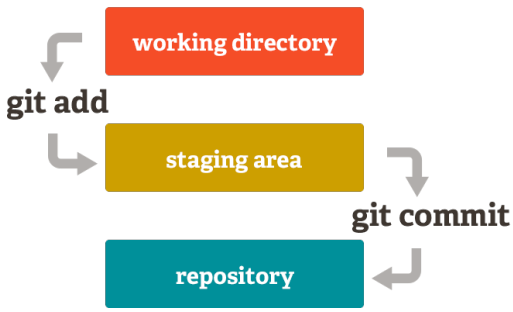

Figura 3.2. Guardando cambios en local con *Git* [44]

Las ramas que se han utilizado en este proyecto, su nomenclatura y usos han sido:

- `master`: rama original y con una versión estable de la app. Como hasta que no se termine con el desarrollo de los objetivos del TFG no se considera que la app esté en una versión inicial estable candidata a entrar en producción, solamente se fusionará el contenido de `develop` en esta rama al final de la realización del TFG. Si después se quisiera seguir añadiendo más funcionalidades a la app, se realizaría una fusión a `master` por cada nueva característica que se añadiera.
- `develop`: rama secundaria creada a partir de `master` y con una versión generalmente estable de la app, en proceso de desarrollo. Si se detecta algún bug en el código contenido en esta rama, se sacará una nueva rama a partir de ésta para solucionarlo y fusionarla de nuevo.
- <`número_HU`>-<`resumen_HU`>: cada una de las ramas creadas a partir de `develop` dedicadas a añadir nueva funcionalidad. Se creará una por cada historia de usuario siguiendo dicho formato. Por ejemplo, si la historia de usuario fuera "HU01: Como usuario quiero introducir mis credenciales para iniciar sesión en mi cuenta", entonces su funcionalidad se añadiría en la rama `1-introducir-credenciales`.
- `refactor-`<`resumen_refactorización`>: si es necesario realizar en algún punto del proyecto una refactorización del código, se creará una rama a partir de `develop` con dicho formato. Por ejemplo, si hubiera que hacer una refactorización para cambiar una librería X utilizada actualmente en el proyecto por una librería Y, la rama que se crearía para ello sería `refactor-cambio-libreria-x-por-libreria-y`.
- `bug-`<`resumen_bug`>: si se encuentra algún bug o fallo en la app, se creará una rama a partir de `develop` con esta nomenclatura. Por ejemplo, si no funciona un click sobre un botón .[el]iminar", la rama creada sería `bug-click-boton-eliminar`.

3.1.2. GitLab Issue Board

GitLab Issue Board [48] es una herramienta para gestión de proyectos software, constituida por tableros en los que se pueden encontrar varias columnas de tarjetas que representan *issues*. Una *issue* es una incidencia que debe llevarse a cabo en el proyecto. Las tarjetas de las *issues* llevan asociadas información como: el nombre de la incidencia, una descripción de la misma, archivos multimedia asociados, usuario de *GitLab* al que se asigna la incidencia para trabajar en ella, un seguimiento del tiempo en el que se indica el tiempo estimado para la incidencia y el tiempo realmente empleado en ella, una lista de incidencias relacionadas y un número que representa el peso de la incidencia. Las tarjetas de *GitLab* poseen más campos con otra información distinta relacionada, pero en el caso de este proyecto, los campos utilizados son los que se acaban de enumerar.

De esta manera, para la gestión del proyecto y para facilitar el seguimiento del mismo por parte de la tutora, pudiendo acceder en cualquier momento para ver el estado exacto del proyecto, se ha utilizado un tablero único dividido en 6 columnas distintas, que son:

- **Product Backlog.** En esta columna entrarán todas las tarjetas que representan las distintas historias de usuario en las que se dividen las épicas del *Product Backlog*.
- **Sprint Backlog.** A esta columna se moverán todas las tarjetas de historias de usuario de la columna "Product Backlog" que se decide que se van a desarrollar en el presente sprint en la reunión de planificación. Es decir, esta columna representa, como su propio nombre indica, al *Sprint Backlog* del sprint actual.
- **Doing.** En esta columna se encontrarán las tarjetas de las *issues* que estén en progreso, las cuales se irán cogiendo de la columna "Sprint Backlog" según se empiece a trabajar en ellas.
- **Under review.** En esta columna aparecerán las tarjetas de las incidencias ya finalizadas y que están esperando a ser revisadas en la próxima reunión con la tutora. Por tanto, las tarjetas aquí presentes provienen de la columna "Doing".
- **Done.** Las tarjetas de las incidencias que ya hayan sido revisadas y aprobadas por la tutora, pasarán de la columna "Under review" a esta columna. Si todas las tarjetas relacionadas con una historia de usuario (se explicará su organización a continuación) están en esta columna, entonces se procederá a cerrar cada una de ellas.
- **Closed.** En esta columna aparecerán cada una de las tarjetas que se han ido cerrando en el transcurso del proyecto.

Para poder identificar más fácilmente la naturaleza de cada tarjeta del tablero, se ha decidido seguir una pequeña guía de estilo aplicada a los títulos de las tarjetas: si se trata de una historia de usuario, el título seguirá el formato **"[<número_HU>] <descripción_HU>"**; si se trata de una tarea de refactorización, el título será de la forma **"[REFACTOR] <descripción_refactorización>"**; y si se trata de una tarea de documentación, el título seguirá la plantilla **"[DOC] <descripción_documentación>"**. Además, para cada tarjeta de historia de usuario, se crean otras 5 tarjetas a mayores para especificar las distintas tareas de dicha historia de usuario (que se explicarán más a fondo en el Capítulo 7 de seguimiento del proyecto), en concreto:

- **"[<número_HU>-AN] <descripción_HU>".** Tarea de análisis.
- **"[<número_HU>-DIS] <descripción_HU>".** Tarea de diseño.
- **"[<número_HU>-IMPL] <descripción_HU>".** Tarea de implementación.
- **"[<número_HU>-REV_IU] <descripción_HU>".** Tarea de revisión de interfaz de usuario.
- **"[<número_HU>-TEST] <descripción_HU>".** Tarea de testeo.

En la tarjeta general de historia de usuario será en la que se listen estas 5 tarjetas como incidencias relacionadas, y también será la única en la que se establezca un peso, concordando con los puntos de historia que se ha decidido dar a dicha historia de usuario.

En las Figuras 3.3 y 3.4 se muestra como ejemplo el estado del tablero en medio de un sprint.

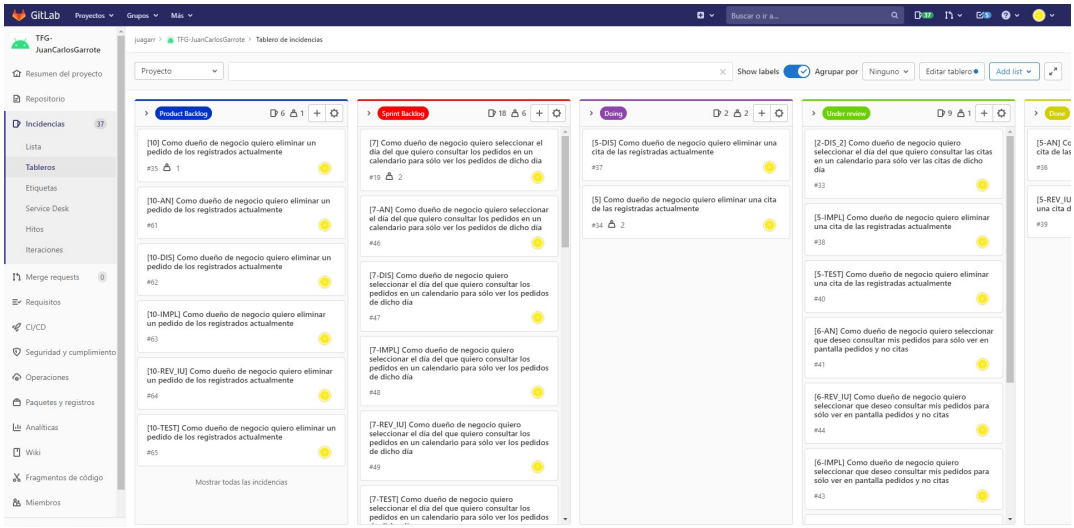

Figura 3.3. Tablero *GitLab Issue Board* (parte 1)

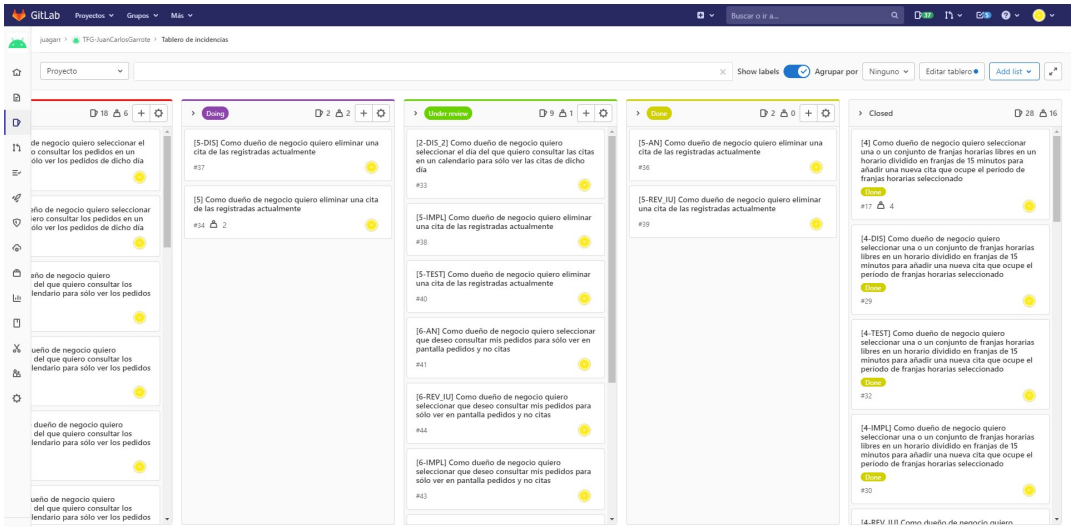

Figura 3.4. Tablero *GitLab Issue Board* (parte 2)

3.1.3. GitLab CI/CD

GitLab CI/CD [47] es una herramienta integrada en *GitLab*, que se ha utilizado para la integración continua en este proyecto. Su funcionamiento consiste en que con cada operación de `push`, se ejecuta un script almacenado en el proyecto (comúnmente con el nombre `gitlab-ci.yml`) que pone en marcha trabajos organizados en *pipelines*, con el objetivo de compilar, testear y desplegar todos los cambios realizados. Esto es muy útil a la hora de evaluar si los cambios que se han introducido en nuevas ramas siguen manteniendo el código en una versión estable, y por tanto se pueden fusionar con su rama de origen.

En el Capítulo 6, donde se detallará la implementación realizada, se explicará más a fondo el contenido del archivo `gitlab-ci.yml` empleado en este proyecto.

3.2. Android

Android [118] es un sistema operativo propiedad de *Google* basado en una versión modificada del kernel Linux, dirigido principalmente a dispositivos móviles inteligentes. Actualmente, es el sistema operativo móvil más utilizado en el mundo, con más de 2500 millones de dispositivos que lo utilizan [1].

En este proyecto, la app objetivo se construirá para dispositivos Android. Para ello, se ha utilizado la **API 26** de Android, lo cual quiere decir que la versión mínima de Android que deben tener los dispositivos para poder ejecutar la app es **Android 8.0 Oreo**. En el momento de creación del proyecto, la utilización de dicha versión cubre el 60,8 % de dispositivos del mercado [22].

La decisión sobre el nivel de API a utilizar se basó en varios factores. Por una parte, se quería utilizar una versión no muy antigua, para poder hacer uso de los componentes más nuevos de Android y darle un aspecto moderno a la app. Por otra parte, uno de los objetivos principales de la app es que sea accesible, por lo que tampoco se podía elegir la versión más nueva disponible, dado que el porcentaje de dispositivos que podrían ejecutarla sería relativamente bajo. Por ello, había que encontrar una versión que cumpliera estas dos premisas de manera equilibrada. Como factor decisivo se decidió hacer una pequeña investigación en el entorno cotidiano, y preguntar sobre la versión Android de sus dispositivos a dueños de negocio que posiblemente constituyeran potenciales candidatos a participar en una futura fase de testing en el proyecto. De los resultados obtenidos, la versión más antigua utilizada era Android 8.0 Oreo, por lo que comprobando el dato de distribución de versiones mencionado anteriormente (la API 26 funciona en el 60,8 % de dispositivos), se consideró una versión adecuada para utilizar en el proyecto.

Como lenguaje de programación se ha utilizado **Java**, con el **JDK 8**. Actualmente, en Android, los principales lenguajes de programación utilizados son Java y Kotlin, cada vez migrándose más a este segundo. Dada la práctica adquirida a lo largo de todo el grado con Java y su aplicación en diseño, se decidió utilizar este para que la curva de aprendizaje en este aspecto fuera baja.

El entorno de desarrollo oficial para Android, y el que se ha utilizado en el proyecto, es **Android Studio** [11], en su versión **4.1.2**, que era la última versión estable en el momento de creación del proyecto.

El gestor de automatización y de dependencias utilizado en Android es ***Gradle*** [62]. *Gradle* es una herramienta de código abierto que se encarga de ejecutar las tareas necesarias para compilar una aplicación de forma eficiente, puesto que sólo ejecuta los pasos necesarios (aquellos en los que haya habido cambios), y de forma flexible, puesto que puede compilar cualquier tipo de software al realizar pocas suposiciones sobre qué es lo que se quiere compilar o cómo hacerlo.

Android supone un framework completo para el desarrollo de apps móviles, pero dentro del mismo, se han utilizado diversos componentes y estilos. Los más importantes se detallarán en las siguientes subsecciones.

3.2.1. Material Design

Material Design [74] es un sistema de guías de estilo, componentes y herramientas que utiliza las mejores prácticas (*best practices*) del diseño de interfaz de usuario (IU). Entre las principales características del diseño presentado en estas guías, se busca que los elementos de la interfaz de usuario simulen papel real, y se fomenta la utilización de tarjetas y sombras para representar y organizar información en pantalla. Fue creado por *Google*, y comenzó siendo utilizado en sus aplicaciones y webs, pero más tarde se puso a disposición de todos los desarrolladores para poder utilizarlo en todas las aplicaciones que se deseara. Aunque principalmente es una filosofía dirigida a las apps Android, también es extrapolable a las aplicaciones web.

En este proyecto se hace uso de este sistema de diseño, pues es una forma completa y atractiva de seguir buenas prácticas a la hora de diseñar interfaces de usuario, y adecuada, ya que el objetivo es una app Android, justo para lo que fue originalmente desarrollado *Material Design*. Asimismo, todos los iconos que se han utilizado en la app también pertenecen a *Material Design*, de código abierto y uso libre para cualquier usuario [76].

3.2.2. Fragments

Los componentes principales de la lógica de una app Android son las *Activities* o actividades y los *Fragments* o fragmentos. Una ***Activity*** [15] se puede entender como una pantalla de la app. Como la mayoría de las apps contienen varias pantallas, por lo general suele haber varias actividades presentes, una de ellas la actividad principal, con la que arranca la app. Un ***Fragment*** [13] es una parte de la interfaz de usuario de la app. Los fragmentos siempre deben estar alojados en una actividad, y aunque poseen ciclo de vida propio, éste se ve directamente afectado por el ciclo de vida de la actividad anfitriona. Una actividad puede contener varios fragmentos, y cada fragmento se puede reutilizar en actividades distintas, permitiendo así modularizar la IU.

Esto ofrece mucha flexibilidad entre dispositivos con distinto tamaño de pantalla. Un ejemplo de ello se puede ver en la Figura 3.5, en la que se observa que con dos fragmentos se puede construir una IU para dos dispositivos con distinto tamaño de pantalla (tablet y dispositivo de mano).

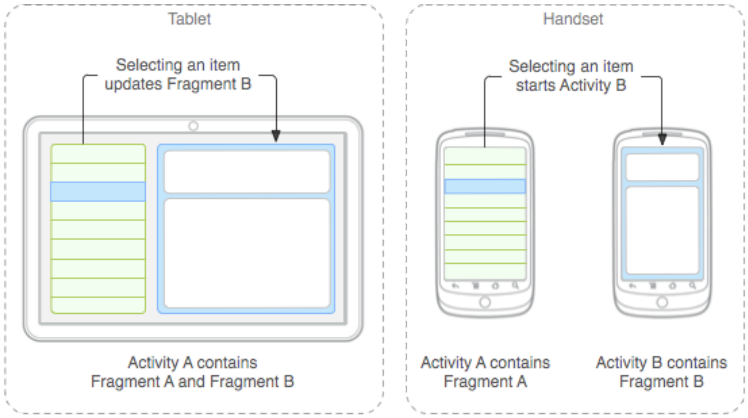

Figura 3.5. Uso de fragmentos y su flexibilidad en tamaños distintos de pantallas [13]

El ciclo de vida de un *Fragment* se puede observar en la Figura 3.6, y en la Figura 3.7 se observa cómo el ciclo de vida de la *Activity* anfitriona afecta sobre el ciclo de vida de sus fragmentos.

En este proyecto, se ha hecho uso de la ***single-activity architecture*** [84], o arquitectura de actividad única, recomendada también por *Google*. Esto quiere decir que la app consiste en una única actividad que servirá como contenedor para todos los fragmentos que muestren las distintas pantallas. Algunas de las ventajas que esto supone son internas, ya que al hacer uso de esta arquitectura se pueden emplear componentes como *Navigation* y *Safe Args* (que se explicarán en las siguientes subsecciones); pero las más notables son externas, ya que la experiencia de usuario se ve mejorada al incrementar el rendimiento de la app y añadir otros elementos visuales como animaciones y mejoras en la integración de elementos en pantalla como la barra de app, el menú lateral desplegable y el menú inferior de navegación.

3.2.3. Navigation

Debido al uso de la *single-activity architecture*, se puede hacer uso del componente *Navigation* de Android. ***Navigation*** [17] permite implementar la navegación entre los distintos fragmentos por los que está formada la app, integrando a su vez la navegación con la barra de

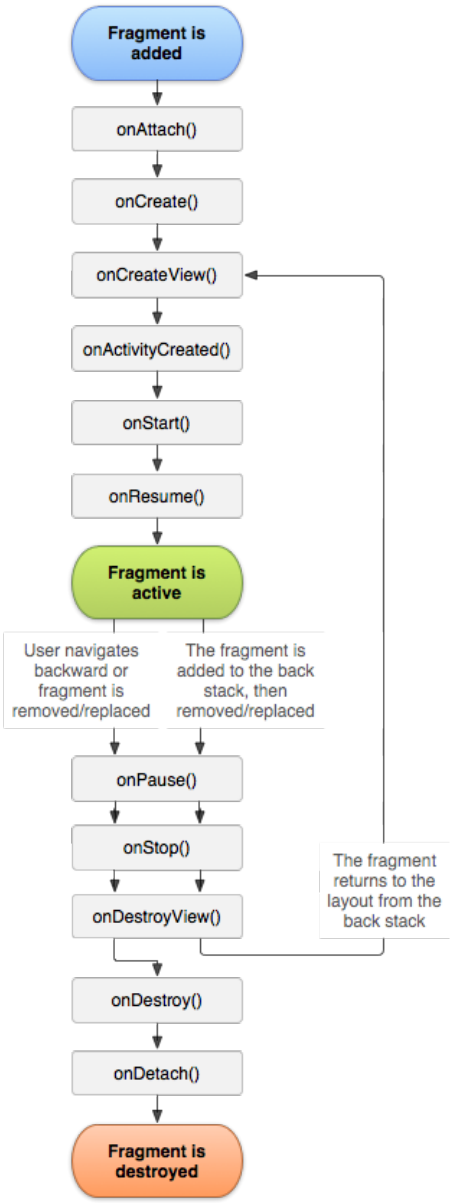

Figura 3.6. Ciclo de vida de un fragmento [13]

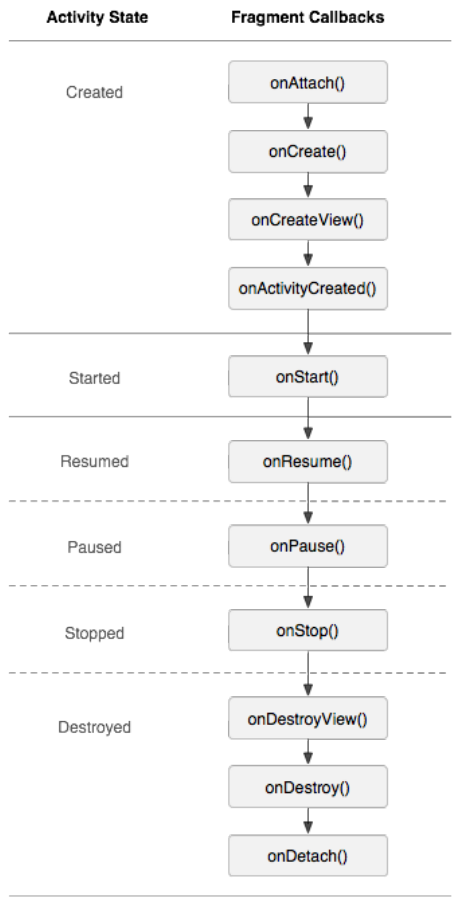

Figura 3.7. Efecto del ciclo de vida de la actividad sobre el ciclo de vida de sus fragmentos [13]

app, el menú lateral desplegable y el menú inferior de navegación. Este componente consta de tres partes clave:

- **Gráfico de navegación.** Recurso XML que se traduce en un grafo que representa todos los posibles caminos que un usuario puede tomar navegando entre fragmentos en la app. Un ejemplo se muestra en la Figura 3.8.
- ***NavHost.*** Contenedor vacío dentro de una actividad que alojará y mostrará los distintos fragmentos (destinos) hacia los que se vaya navegando durante la ejecución de la app.
- ***NavController.*** Objeto que gestiona la navegación dentro de un *NavHost*, intercambiando el fragmento que contiene el mismo cuando se realiza una acción de navegación.

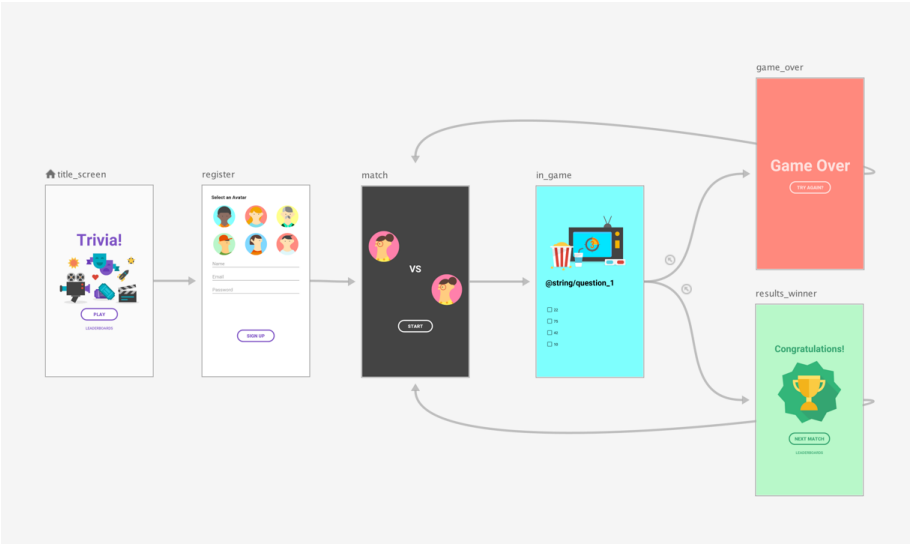

Figura 3.8. Gráfico de navegación del componente *Navigation* [17]

Utilizando *Navigation*, la gestión de la navegación entre las pantallas de una app se vuelve mucho más sencilla, por lo que se ha elegido como componente a utilizar en este proyecto.

3.2.4. Safe Args

Safe Args [9] es un complemento de *Gradle* para *Navigation* que genera clases de objeto simples en tiempo de compilación para permitir una navegación que garantiza la seguridad de tipo en los argumentos que se pasan entre fragmentos, así como el acceso a los mismos de forma segura. Las clases generadas son las siguientes: por cada destino de origen (nodo de

donde sale una flecha en el grafo de navegación), una clase con el nombre de dicho fragmento origen y sufijo `-Directions`, que tiene un método para cada acción definida en ese destino; por cada acción, una clase interna con su mismo nombre, que recibe por parámetros los argumentos a pasar al fragmento destino, y a su vez se pasa como argumento al método correspondiente de la clase `-Directions`; y por cada destino de recepción (nodo donde llega una flecha en el grafo de navegación), una clase con el nombre de dicho fragmento destino y sufijo `-Args`, que dispone de métodos para recuperar de forma segura los argumentos recibidos.

Dado que se utiliza el componente *Navigation* en el proyecto, para prevenir fallos en tiempo de ejecución y seguir las recomendaciones establecidas, se ha decidido utilizar también *Safe Args*.

3.2.5. Espresso

Espresso [12] es un framework para Android que proporciona una API que permite crear tests de interfaz de usuario, simulando la interacción de usuarios con la app. Fue lanzado por *Google*, por lo que constituye el framework oficial para pruebas de IU en Android. Las pruebas con Espresso buscan ejecutarse con relativa rapidez y eficiencia, a pesar de que siguen siendo de los tipos de tests que más recursos consumen para su ejecución.

La API está compuesta por métodos de interacción con las vistas (como `onView()` y `onData()`), `ViewMatchers` que filtran o seleccionan una vista dentro de la jerarquía de vistas (como `withText()`), `ViewActions` que ejecutan acciones sobre la vista previamente seleccionada (como `click()`), y `ViewAssertions` que realizan comprobaciones sobre el estado de la vista seleccionada (como `isDisplayed()`).

Dado que en este proyecto se han realizado tests unitarios y tests de IU, para estos segundos se ha utilizado *Espresso*.

3.3. Firebase

Firebase [40] es una plataforma en la nube que integra varias herramientas pensadas para el desarrollo y despliegue de aplicaciones web y móvil. Actualmente, la plataforma es propiedad de *Google*, por lo que se integra perfectamente con el desarrollo de apps Android. El objetivo de *Firebase* es ofrecer servicios de backend, monitorización y analíticas sin necesidad de administrar servidores, así como ofrecer escalabilidad en los mismos para poder brindar asistencia a numerosos usuarios de las apps.

Aunque la plataforma ofrece numerosos servicios, en este proyecto se han utilizado los que se detallan en las siguientes subsecciones.

3.3.1. Realtime Database

Firebase ofrece dos alternativas para bases de datos alojadas en la nube: *Realtime Database* y *Cloud Firestore*. Realizando un pequeño cuestionario [38] en el que se comparan las consideraciones clave de cada una, y consultando otros aspectos menores, se concluyó que para este proyecto la mejor solución era utilizar *Realtime Database*.

Realtime Database [42] es una base de datos NoSQL alojada en la nube. Los datos se almacenan como un gran árbol JSON, y se sincronizan en tiempo real con todos los clientes conectados. Su latencia es extremadamente baja, y no está pensada para realizar consultas o transacciones muy complejas, por lo que se ajusta a lo deseado para este proyecto. Se ha utilizado el plan gratuito *Spark*, que incluye una cuota de uso de la base de datos que no se planea alcanzar, por lo que constituye una tarifa adecuada para el contexto donde se va a utilizar.

3.3.2. Authentication

Authentication [41] es el servicio de autenticación en línea que proporciona *Firebase*. Sus funciones principales son la gestión de usuarios y de las sesiones, ofreciendo una API simple para utilizar desde el código de la app. Entre los distintos métodos de autenticación ofrecidos se encuentran: acceso con correo electrónico y contraseña, acceso mediante proveedores de identidad federada (*Google*, *Apple*, *Facebook*...), acceso mediante número de teléfono y acceso mediante integración con sistemas de autenticación personalizados ya existentes. En el caso de este proyecto, se ha utilizado el acceso mediante correo electrónico y contraseña para la autenticación de usuarios.

3.3.3. Cloud Storage

Cloud Storage [36] es un servicio de almacenamiento de archivos en la nube, que permite la carga y descarga de contenido con filtros de seguridad de *Google* integrados. Además, el servicio destaca por su escalabilidad, ya que es posible mantener la misma infraestructura en una app prototipo que en una app en producción con un elevado uso. En este proyecto, se ha utilizado *Cloud Storage* de *Firebase* para almacenar las fotos de perfil tanto de clientes como de establecimientos, y cargarlas o descargarlas mediante el uso de su API cuando sea necesario.

3.4. Herramientas para análisis, diseño y documentación

3.4.1. Astah Professional

Astah es una herramienta de modelado, empleada para realizar los distintos diagramas y modelos que representan un sistema software en sus etapas de análisis y diseño. En este

proyecto, se ha utilizado la versión ***Astah Professional 8.2.0*** [24] para elaborar todos los diagramas que se presentarán en capítulos posteriores. La licencia de la herramienta ha sido cedida por la Escuela de Ingeniería Informática de la Universidad de Valladolid, que ofrece a sus alumnos un pack de licencias de software de pago de manera gratuita.

3.4.2. Balsamiq Wireframes

Balsamiq Wireframes [28] es la herramienta que se ha utilizado para crear *wireframes*, *mockups* o prototipos de baja fidelidad de la interfaz de usuario de las distintas pantallas de la app. Estos prototipos se mostrarán en el Capítulo 5, en el que se tratará sobre el diseño del sistema software. Se ha utilizado tanto la versión web de la aplicación (***Balsamiq Cloud***), como la versión de escritorio (***Balsamiq Wireframes 4.2.1***), cuya licencia también ha sido ofrecida por la universidad.

3.4.3. Overleaf

Overleaf [86] es un editor online de LaTeX colaborativo, en el que se ha desarrollado el presente documento. Permite la edición de varios usuarios simultáneamente y almacena un historial del documento con las distintas versiones por las que ha ido pasando. Es por ello que constituye una herramienta muy útil para las revisiones de la tutora sobre la documentación, permitiendo añadir comentarios e indicaciones de revisión.

3.5. Herramientas para la comunicación

3.5.1. Microsoft Teams

Microsoft Teams [79] es una plataforma desarrollada por *Microsoft* dedicada al trabajo en equipo y colaborativo. Sus principales características son canales de chat, videoconferencias y compartición de archivos. En este proyecto se ha utilizado la herramienta en las primeras semanas de trabajo para editar y compartir con la tutora un documento donde se tomarán apuntes sobre el proyecto que posteriormente se trasladarán a *Overleaf*.

3.5.2. Rocket.Chat

Rocket.Chat [93] es una plataforma de código abierto basada en un chat en equipo. En el proyecto, *Rocket.Chat* ha supuesto la principal vía de comunicación con la tutora cuando era necesario realizar alguna pregunta o comentario puntual que no podía esperar a la siguiente reunión semanal. Además, a través de unos comandos, se pueden crear salas de videoconferencia de *Jitsi*, explicado a continuación.

3.5.3. Jitsi

Jitsi [67] es una aplicación de videoconferencia, también de código abierto, utilizada a través de la web. Mediante esta herramienta se han realizado las reuniones de los distintos eventos Scrum con la tutora, y se pretende que sea la herramienta de videoconferencia con la que se realice la defensa ante el tribunal del TFG.

3.5.4. Vysor

Vysor [117] es una aplicación que permite visualizar y controlar en la pantalla del ordenador la pantalla de un dispositivo móvil físico conectado mediante cable. En este proyecto, se ha utilizado la herramienta para mostrar a la tutora en las reuniones semanales los avances y el aspecto de la app Android en desarrollo, pues consume muchos menos recursos que un emulador y permite un mejor funcionamiento del resto de programas abiertos en el ordenador. También se pretende utilizar en la defensa del TFG para realizar una "demo" completa de la app desarrollada.

Capítulo 4

Análisis

Como parte de la fase de análisis del proyecto, llevada a cabo parcialmente tanto en el sprint 0 como también una de las tareas de cada una de las historias de usuario desarrolladas, se han desglosado los requisitos del proyecto como historias de usuario, para posteriormente dar lugar a un modelo de dominio inicial y un modelo de proceso de negocio. Los modelos y sus diagramas se han representado íntegramente en inglés, de manera que se respeta el requisito no funcional de desarrollar la documentación interna en inglés, que también se presentará en este Capítulo.

4.1. Requisitos funcionales como historias de usuario

Los requisitos funcionales se han obtenido desglosando las épicas que se especifican en el *Product Backlog* final (Tabla 2.14) en distintas historias de usuario. En este caso, también se les ha asignado un código a cada una de ellas para poder referenciarlas más tarde de manera más fácil. Asimismo, como parte de la planificación del proyecto, otra tarea que se ha hecho es asignar a cada historia un número de puntos de historia, tal y como se explica en el Capítulo 2. Esta estimación aparece también junto a cada historia de usuario.

Como en el alcance del proyecto se desarrollarán las épicas desde **EP01** hasta **EP10**, sólo se han dividido éstas en distintas historias de usuario. En el caso de las épicas que sólo se han desglosado en una, el objetivo es especificar más y crear una historia de usuario que esté al mismo nivel que el resto, para poder tratar a todas de la misma manera.

En las Tablas 4.1 a 4.10 se muestran las historias de usuario divididas por su épica de origen.

Código	Historia de usuario	Puntos
HU01	Como dueño de negocio quiero seleccionar que deseo consultar mis citas para sólo ver en pantalla citas y no pedidos.	5
HU02	Como dueño de negocio quiero seleccionar el día del que quiero consultar las citas en un calendario para sólo ver las citas de dicho día.	2
HU03	Como dueño de negocio quiero seleccionar una franja horaria ocupada en un horario dividido en franjas de 15 minutos para ver los detalles de la cita ya establecida en esa franja.	5
HU04	Como dueño de negocio quiero seleccionar una o un conjunto de franjas horarias libres en un horario dividido en franjas de 15 minutos para añadir una nueva cita que ocupe el período de franjas horarias seleccionado.	4
HU05	Como dueño de negocio quiero eliminar una cita de las registradas actualmente.	2

Tabla 4.1. Historias de usuario de la épica **EP01**

Código	Historia de usuario	Puntos
HU06	Como dueño de negocio quiero seleccionar que deseo consultar mis pedidos para sólo ver en pantalla pedidos y no citas.	1
HU07	Como dueño de negocio quiero seleccionar el día del que quiero consultar los pedidos en un calendario para sólo ver los pedidos de dicho día.	2
HU08	Como dueño de negocio quiero seleccionar una franja horaria en un horario dividido en franjas de 15 minutos para ver todos los pedidos ya registrados para esa franja.	2
HU09	Como dueño de negocio quiero añadir un nuevo pedido en la franja horaria que me encuentro actualmente para registrarlo en la app y gestionarlo junto con los demás pedidos.	2
HU10	Como dueño de negocio quiero eliminar un pedido de los registrados actualmente.	1

Tabla 4.2. Historias de usuario de la épica **EP02**

Código	Historia de usuario	Puntos
HU11	Como cliente quiero seleccionar uno de los establecimientos de la lista para poder interactuar con él.	2
HU12	Como cliente quiero ver los detalles del establecimiento seleccionado y seleccionar que deseo pedir una cita para realizar una solicitud de cita al mismo.	1
HU13	Como cliente quiero seleccionar el día para el que quiero pedir una cita a un establecimiento para ver las posibilidades de reserva de ese día.	1
HU14	Como cliente quiero ver las franjas horarias en las que es posible solicitar una cita y los huecos disponibles actualmente en un horario dividido en franjas de 15 minutos para informarme sobre la disponibilidad del establecimiento.	1
HU15	Como cliente quiero seleccionar una o un conjunto de franjas horarias libres en un horario dividido en franjas de 15 minutos para pedir una cita que ocupe el período de franjas seleccionado y que el establecimiento reciba mi solicitud de cita.	1

Tabla 4.3. Historias de usuario de la épica **EP03**

Código	Historia de usuario	Puntos
HU16	Como cliente quiero ver los detalles del establecimiento seleccionado y seleccionar que deseo hacer un pedido para realizar una solicitud de pedido al mismo.	0,5
HU17	Como cliente quiero seleccionar el día para el que quiero hacer un pedido a un establecimiento para ver las posibilidades de reserva de ese día.	0,5
HU18	Como cliente quiero ver las franjas horarias en las que es posible solicitar un pedido en un horario dividido en franjas de 15 minutos para informarme sobre la disponibilidad del establecimiento.	1
HU19	Como cliente quiero seleccionar una franja horaria disponible en un horario dividido en franjas de 15 minutos para hacer un pedido en la franja horaria seleccionada y que el establecimiento reciba mi solicitud de pedido.	1

Tabla 4.4. Historias de usuario de la épica **EP04**

Código	Historia de usuario	Puntos
HU20	Como dueño de negocio quiero ver la lista de solicitudes de cita que se han realizado a mi establecimiento para poder dar una respuesta al cliente.	2
HU21	Como dueño de negocio quiero aceptar una de las solicitudes de cita que se han realizado a mi establecimiento para informar al cliente solicitante.	0,5
HU22	Como dueño de negocio quiero rechazar una de las solicitudes de cita que se han realizado a mi establecimiento para informar al cliente solicitante.	0,5
HU23	Como cliente quiero ver la lista de solicitudes de cita pendientes, aceptadas, rechazadas y confirmadas que he realizado a los distintos negocios.	2
HU24	Como cliente quiero confirmar la asistencia a una cita solicitada y aceptada por un establecimiento.	0,5
HU25	Como dueño de negocio quiero ver la lista de solicitudes de pedido que se han realizado a mi establecimiento para poder dar una respuesta al cliente.	1
HU26	Como dueño de negocio quiero aceptar una de las solicitudes de pedido que se han realizado a mi establecimiento para informar al cliente solicitante.	1
HU27	Como dueño de negocio quiero rechazar una de las solicitudes de pedido que se han realizado a mi establecimiento para informar al cliente solicitante.	0,5
HU28	Como cliente quiero ver la lista de solicitudes de pedido pendientes, aceptadas y rechazadas que he realizado a los distintos negocios.	2

Tabla 4.5. Historias de usuario de la épica **EP05**

Código	Historia de usuario	Puntos
HU29	Como usuario quiero acceder a la pantalla de inicio de sesión e introducir mis credenciales para entrar en mi cuenta de cliente o de dueño de negocio.	2
HU30	Como usuario quiero acceder a la pantalla de registro para crear una nueva cuenta de cliente.	1

Tabla 4.6. Historias de usuario de la épica **EP06**

Código	Historia de usuario	Puntos
HU31	Como cliente quiero acceder a la sección de "Tablón" para ver el conjunto de anuncios publicados hasta la fecha.	1

Tabla 4.7. Historias de usuario de la épica **EP07**

Código	Historia de usuario	Puntos
HU32	Como dueño de negocio quiero tener una opción de añadir un nuevo anuncio al tablón para ver el formulario que tengo que rellenar.	0,5
HU33	Como dueño de negocio quiero añadir un nuevo anuncio al tablón para publicar información sobre mi negocio que puedan ver los clientes.	1

Tabla 4.8. Historias de usuario de la épica **EP08**

Código	Historia de usuario	Puntos
HU34	Como cliente quiero acceder a la sección de "Mapa" para ver las ubicaciones de los distintos establecimientos registrados en la app.	1

Tabla 4.9. Historias de usuario de la épica **EP09**

Código	Historia de usuario	Puntos
HU35	Como dueño de negocio quiero proporcionar mi ubicación en el momento del registro para poder ver mi ubicación en el mapa interactivo.	0,5

Tabla 4.10. Historias de usuario de la épica **EP10**

4.2. Requisitos de información como historias de usuario

Aunque los requisitos de información forman parte de los requisitos funcionales, se ha decidido tratarlos por separado, ya que a partir de éstos es de donde se obtendrá el modelo de dominio inicial.

En la Tabla 4.11 se observan los requisitos de información en forma de historias de usuario.

4.3. Requisitos no funcionales como historias de usuario

De la misma manera que hay requisitos funcionales, en todo proyecto software existen unos requisitos no funcionales que hay que cumplir de manera transversal.

Para su obtención, se ha seguido la guía FURPS [120], que recorriendo cada una de las categorías que describe el acrónimo inglés (funcionalidad -que se ha cubierto con los requisitos funcionales-, usabilidad, fiabilidad, rendimiento y soporte), se garantiza que se cubren los aspectos básicos que deben cumplir los requisitos no funcionales.

En la Tabla 4.12 se pueden observar los requisitos no funcionales en forma de historias de usuario. En este caso también se les ha asignado un código, para poder referirnos a ellas a lo largo del documento.

Historia de usuario
Como equipo de desarrollo quiero guardar los siguientes datos de un **cliente**: nombre, email, contraseña, fecha de nacimiento, localidad y foto de perfil, para manejar la información básica necesaria en la app.
Como equipo de desarrollo quiero guardar los siguientes datos de un **establecimiento**: nombre, email, contraseña, categoría, descripción, número de teléfono, ubicación, los servicios que ofrece entre "citas" y "pedidos", el máximo de clientes que recibe para citas por franja horaria y foto de perfil, para manejar la información básica necesaria en la app.
Como equipo de desarrollo quiero guardar los siguientes datos de una **cita**: día, franja horaria, motivo, cliente solicitante si es que lo hay o nombre del cliente solicitante en caso contrario, establecimiento al que va dirigida la cita, estado de la cita, si el cliente ha confirmado su asistencia después de aceptar la cita, minutos de retraso estimados en caso de que lo haya y respuesta del establecimiento si se rechaza la cita, para manejar la información básica necesaria en la app.
Como equipo de desarrollo quiero guardar los siguientes datos de un **pedido**: día, hora, detalle, cliente solicitante si es que lo hay o nombre del cliente solicitante en caso contrario, establecimiento al que va dirigido el pedido, estado del pedido, minutos de retraso estimados en caso de que lo haya y respuesta del establecimiento si se rechaza el pedido, para manejar la información básica necesaria en la app.
Como equipo de desarrollo quiero guardar los siguientes datos de un **anuncio**: título, descripción, imagen, categoría, fecha de publicación y establecimiento que lo ha publicado, para manejar la información básica necesaria en la app.
Como equipo de desarrollo quiero guardar los siguientes datos de una **solicitud de registro de nuevo establecimiento**: nombre, email, categoría, descripción, ubicación, los servicios que ofrece entre "citas" y "pedidos", el máximo de clientes que recibe por franja horaria, foto de perfil y respuesta del administrador si se rechaza la solicitud, para manejar la información básica necesaria en la app.

Tabla 4.11. Requisitos de información como historias de usuario

Código	Historia de usuario
HUNF01	Como usuario quiero que la aplicación de gestión de citas y pedidos se materialice en una app Android para poder tenerla siempre a mano en mi dispositivo móvil y hacer uso de ella cuando lo necesite.
HUNF02	Como usuario quiero que la interfaz de usuario de la app tenga un aspecto atractivo, haciendo que sus elementos simulen en lo posible papel real, y que sea intuitiva y fácil de utilizar, de manera que todo usuario que maneje apps similares sea capaz de hacer una tarea en menos de 1 minuto, para dotar a la app de buenos atributos de usabilidad.
HUNF03	Como usuario quiero que la transición entre las distintas pantallas de la app sea fluida y animada para poder disfrutar de una buena experiencia de usuario.
HUNF04	Como usuario quiero que la app muestre un mensaje de error genérico cuando ocurra un fallo interno en la app, pero que ésta siga funcionando, para que la app sea robusta y posea buen manejo de errores.
HUNF05	Como equipo de desarrollo quiero que la app genere logs durante su ejecución en caso de que ocurra un fallo interno, indicando en qué punto del código ha ocurrido y la razón del fallo, para poder depurar y corregir el código de manera mucho más fácil.
HUNF06	Como usuario quiero recibir actualizaciones de los datos de la app en tiempo real y con una latencia menor a 1 segundo, para que la realización de tareas en la app sea fluida.
HUNF07	Como usuario quiero que la interfaz de usuario se muestre en español para poder comprender adecuadamente toda la información mostrada.
HUNF08	Como equipo de desarrollo quiero que el código y su documentación interna se desarrolle en inglés para poder internacionalizarlo más fácilmente.
HUNF09	Como usuario quiero tener mi propia cuenta en la app para que los datos que se almacenan relacionados conmigo (información personal, solicitudes, citas y pedidos si soy dueño de negocio) no sean visibles por cualquier otro usuario.

Tabla 4.12. Requisitos no funcionales como historias de usuario

4.4. Restricciones como historias de usuario

En el sistema a construir también se ha decidido incluir algunas restricciones que la aplicación tendrá que respetar, basadas en los objetivos principales a cumplir. En la Tabla 4.13 se puede ver la lista de restricciones en forma de historias de usuario.

Historia de usuario
Como usuario quiero que la app sea gratuita para poder utilizarla libremente sin tener que realizar ninguna inversión económica.
Como equipo de desarrollo quiero distribuir la app a través de un archivo APK o si se llega a desplegar, a través de *Google Play*, para hacer llegar la app a los usuarios finales de manera sencilla.
Como usuario quiero que mis datos se traten acorde al RGPD para poder garantizar la privacidad de los mismos.

Tabla 4.13. Restricciones como historias de usuario

4.5. Modelo de dominio inicial

A partir de los requisitos funcionales de información (Tabla 4.12), se ha elaborado un modelo de dominio inicial, que se representa en el diagrama de clases que se puede observar en la Figura 4.1.

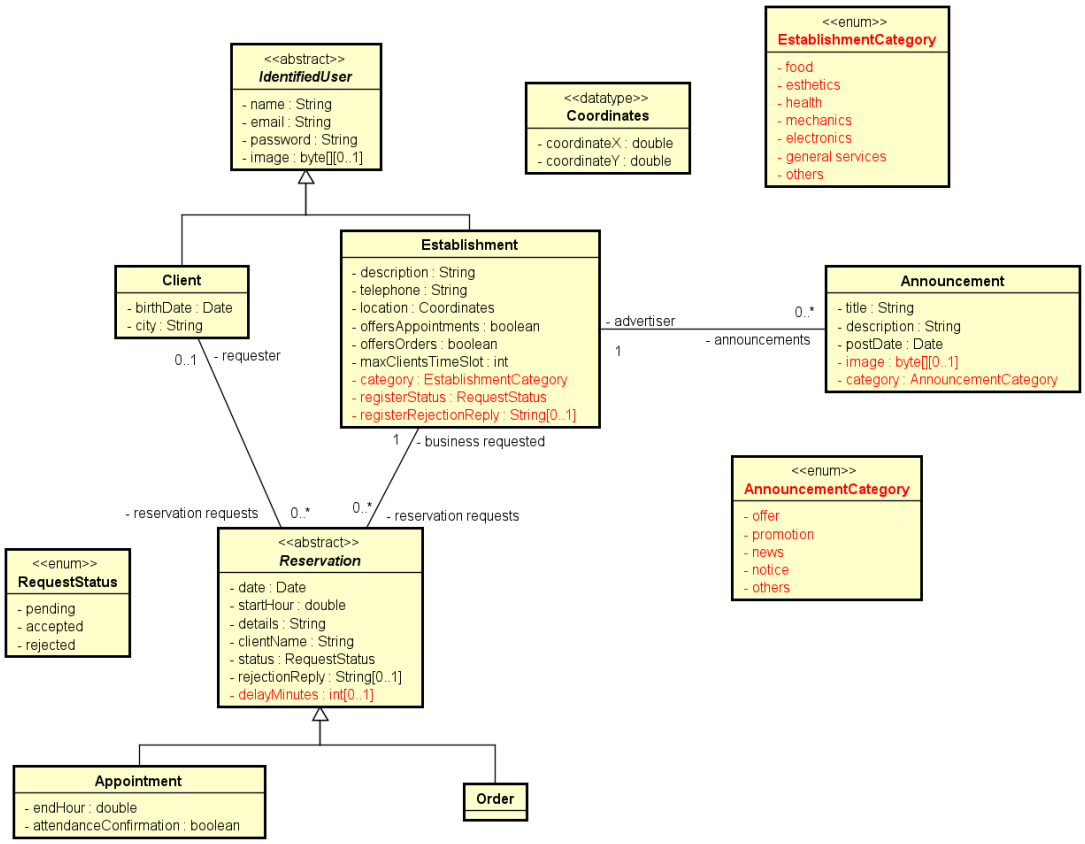

Figura 4.1. Modelo de dominio inicial

Las clases y atributos que aparecen en rojo son los que no entran dentro del alcance del proyecto, aunque al formar parte del sistema y sus requisitos, deben aparecer en el modelo de dominio. En concreto, estos son: las clases *enum* que indican la categoría de establecimientos y anuncios y que en un futuro se podría utilizar para filtrarlos; la imagen de los anuncios, ya que no se ha llegado a incluir en la versión de la app desarrollada en el proyecto; los minutos de retraso en una reserva de cita o pedido, que está dentro del contexto de las épicas **EP11** y **EP12** y por tanto fuera del alcance del proyecto; y los atributos relacionados con la solicitud de registro de un establecimiento, que pertenece a la funcionalidad de las épicas **EP13** y **EP14**, también fuera del alcance del proyecto.

4.6. Modelo de proceso de negocio

Para completar el análisis del sistema, se ha realizado un diagrama de actividades mostrando el principal proceso de negocio que se llevará a cabo a través de la app: **pedir cita a un establecimiento**. En la Figura 4.2 se puede observar dicho diagrama.

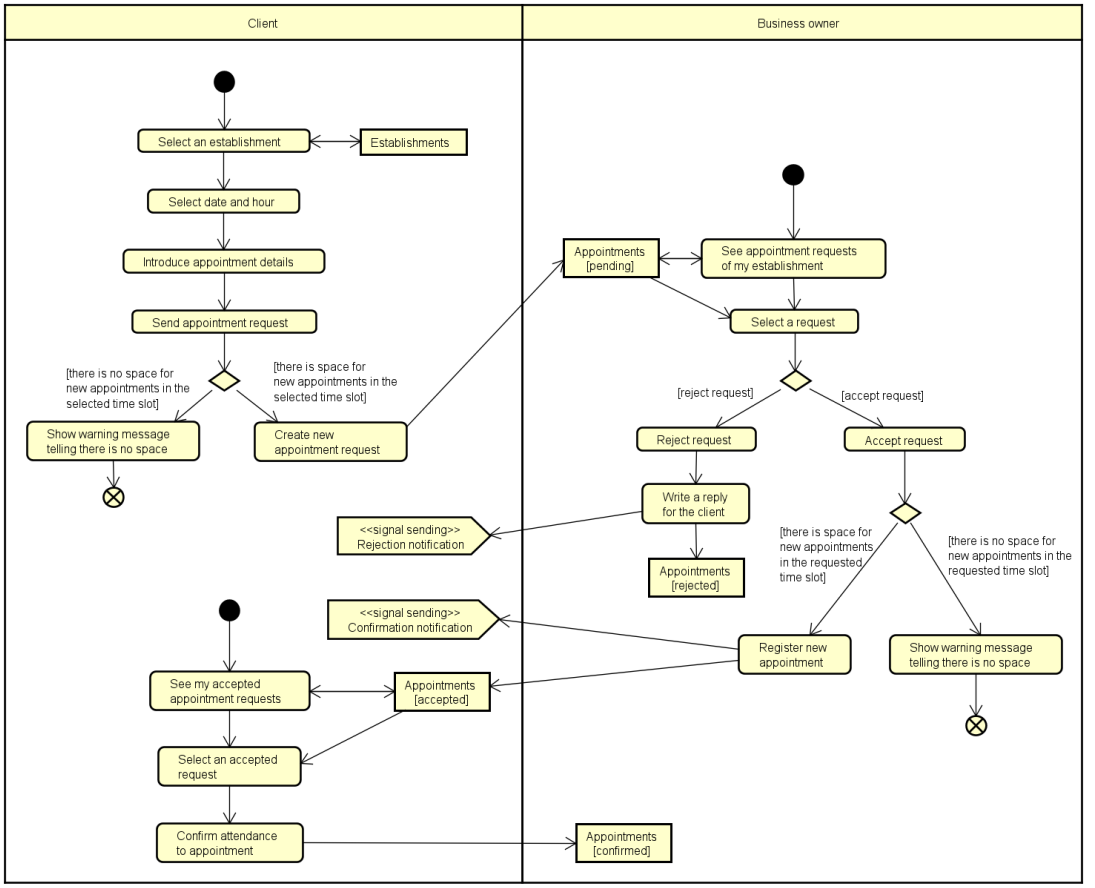

Figura 4.2. Modelo de dominio inicial

Como se observa, son dos actores los implicados en el proceso de negocio: cliente y dueño de negocio. El proceso es comenzado por el cliente, de tal manera que introduce los parámetros de la cita que desea solicitar (a qué establecimiento va dirigida, fecha y hora y detalles de la cita), y la envía. Si no hay espacio en la franja horaria y día seleccionados, se informa al cliente y el flujo de la actividad termina, y si hay espacio libre, se crea una nueva cita en estado "pendiente". El dueño de negocio comenzará la actividad de revisar las solicitudes de cita pendientes, y puede o bien rechazar la solicitud de cita escribiendo una respuesta

al cliente, lo que genera una notificación de rechazo para el cliente y pasa la cita a estado "rechazada", o bien aceptarla, donde se mostrará un mensaje en caso de que no haya espacio libre para la cita y terminará el flujo de la actividad, o se registrará la nueva cita generando una notificación de confirmación para el cliente y pasando la cita a estado "aceptada" en caso contrario. Finalmente, el cliente iniciará la actividad de revisar sus solicitudes de cita aceptadas y confirmará su asistencia a la cita ya aceptada por el establecimiento, pasando el estado de la misma a "confirmada".

En el sistema, como se define en sus objetivos principales, se desea gestionar las citas y pedidos de los establecimientos. Es por ello que el otro proceso de negocio principal de la app es **hacer pedido a un establecimiento**, pero se ha prescindido de su diagrama de actividades ya que es completamente análogo al ya presentado, eliminando la parte de la confirmación de cita, por lo que el pedido solamente podría pasar por los estados "pendiente", "rechazado" y "aceptado".

Capítulo 5

Diseño

5.1. Decisiones de diseño

A lo largo del proyecto, se han tenido que tomar diversas decisiones de diseño para satisfacer los requisitos del mismo. En esta sección, se pondrán en correspondencia los requisitos no funcionales en forma de historias de usuario de la Tabla 4.12 con las decisiones tomadas para satisfacerlos.

- **HUNF01.** Para la construcción de la app, se ha utilizado la **API 26** de Android, equivalente a **Android 8.0 Oreo**. Las razones principales son las que se explican en la Sección 3.2.
- **HUNF02.** Para la elaboración de la interfaz de usuario, se han seguido las guías de estilo de ***Material Design***, explicado en la Sección 3.2.1.
- **HUNF03.** Para poder ofrecer una buena experiencia de usuario en la transición entre las distintas pantallas de la app, se ha aplicado una **arquitectura *single-activity*** con fragmentos (Sección 3.2.2) junto con el componente ***Navigation*** de Android (Sección 3.2.3). Esto permite una navegación entre destinos fluida y que se puede dotar de animaciones.
- **HUNF04.** Para que la aplicación sea robusta, se ha decidido que cuando ocurra un error interno (principalmente relacionado con la base de datos, que es independiente de la app) se reciba un `null` como respuesta, y que dicho caso se maneje **manteniendo el funcionamiento** de la app y mostrando un **mensaje de error** genérico al usuario mediante un `Snackbar`.
- **HUNF05.** Para facilitar la depuración del código de la app, se ha utilizado la clase `Log` [16], que imprime un mensaje en el log de la ejecución indicando en qué clase ha ocurrido el error y la razón del mismo.

- **HUNF06.** Para poder recibir actualizaciones de datos en tiempo real con una latencia extremadamente baja, se ha utilizado como base de datos ***Realtime Database*** de *Firebase*, explicada en la Sección 3.3.1.
- **HUNF07.** Para mostrar la información en español, se ha utilizado para todos los textos mostrados en la interfaz de usuario los **recursos de strings** [19] que proporciona Android, en este caso escritos en castellano, de tal manera que también se pueda traducir a cualquier otro idioma de forma directa simplemente traduciendo esos recursos, sin necesidad de tocar el código de la aplicación.
- **HUNF08.** Para facilitar la internacionalización del proyecto, el código y su documentación se ha escrito íntegramente en **inglés**. Esto implica que los nombres y descripción de variables, clases, métodos, documentación interna, modelos y diagramas se han desarrollado completamente en inglés.
- **HUNF09.** Para gestionar usuarios y sesiones en la app, se ha utilizado el servicio ***Authentication*** de *Firebase*, explicado en la sección 3.3.2.

5.2. Patrón arquitectónico MVVM

Model-View-ViewModel [94] [77] o Modelo-Vista-Modelo de vista (MVVM) es un patrón arquitectónico de software que se caracteriza por el principio de separación de responsabilidades. En concreto, se trata de desacoplar lo máximo posible la interfaz de usuario de la lógica de la aplicación. Este patrón deriva del patrón arquitectónico *Model-View-Presenter* (Modelo-Vista-Presentador, MVP), definido por Martin Fowler, que a su vez es una derivación del patrón arquitectónico *Model-View-Controller* (Modelo-Vista-Controlador, MVC), ampliamente conocido y utilizado.

Las tres partes presentes en el patrón son:

- **Modelo.** Representa la información con la que el sistema trabaja, es decir, la lógica de negocio.
- **Vista.** Presenta la información del Modelo por pantalla mediante la interfaz de usuario. También captura los eventos de interacción del usuario con la interfaz y puede ejecutar lógica de IU.
- **Modelo de vista.** Es un actor intermediario entre el Modelo y la Vista que contiene toda la lógica de presentación.

Además, una particularidad de este patrón arquitectónico es que se puede establecer un *data binding* o enlace de datos entre Vista y Modelo de vista, de tal manera que este último notifique a la Vista cuando ocurre algún cambio en el Modelo para actualizarla automáticamente. Entre Modelo de vista y Modelo se puede implementar lo mismo, de forma que el Modelo notifique al Modelo de vista.

En la Figura 5.1 se puede observar de manera gráfica las relaciones entre las distintas partes presentes en el patrón. La gran ventaja es el desacoplamiento entre las distintas partes, que se puede ver dado que la Vista sólo conoce al Modelo de vista, y el Modelo de vista sólo conoce al Modelo.

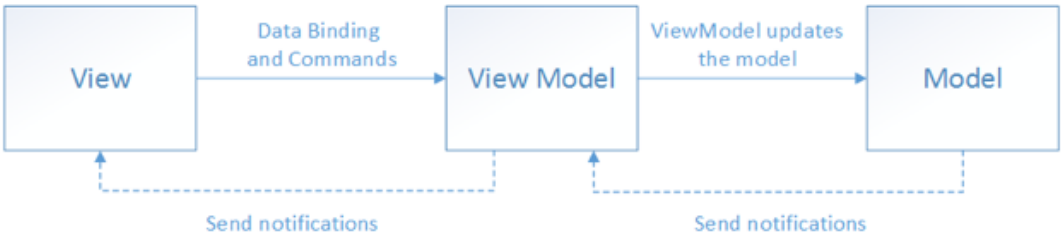

Figura 5.1. Relaciones entre partes en el patrón arquitectónico MVVM [77]

5.2.1. MVVM en este proyecto

El patrón MVVM se ha adaptado al proyecto de tal manera que la arquitectura está compuesta por distintos paquetes de clases, en concreto: **views**, **viewmodels**, **daos** y **entities**, de los cuales los tres primeros hacen referencia respectivamente a las partes de Vista, Modelo de vista y Modelo del patrón, y el último funciona como capa transversal que contiene los DTO, patrón que se explicará en la siguiente sección. Es decir, en este proyecto los DTOs son entidades. De DAOs y de DTOs se hablará en la Sección 5.3.1. La arquitectura del sistema se verá con más detalle en la Sección 5.5.

En este caso, no se han implementado *data bindings*, dado que en principio no han sido necesarios con el diseño que se ha dado a las distintas partes de la aplicación, aunque podría ser una posible modificación para aplicar un patrón MVVM más puro. Al no aplicar enlaces de datos, la complejidad de las relaciones es aún menor, puesto que el envío de notificaciones se omite. Por contra, las actualizaciones en la interfaz de usuario no son automáticas y hay que consultar al Modelo de vista desde la Vista.

Al ser MVVM una de las arquitecturas recomendadas por Android, se ofrecen varios componentes arquitectónicos de gran utilidad para implantarla. El componente por excelencia y el que se ha utilizado en este proyecto, es la clase `ViewModel` [10]. `ViewModel` fue diseñada con el objetivo de almacenar y administrar datos de una actividad o fragmento de manera optimizada para los ciclos de vida. Si el sistema destruye o recrea un controlador de IU (actividad o fragmento), se perderán todos los datos almacenados allí. Utilizando `ViewModel`, si ocurre algún cambio de configuración (como una simple rotación de pantalla) que haga cambiar la etapa del ciclo de vida de su controlador de IU, los datos se conservan tal y como estaban. En la Figura 5.2 se puede observar el ciclo de vida de un `ViewModel` junto a su controlador de IU.

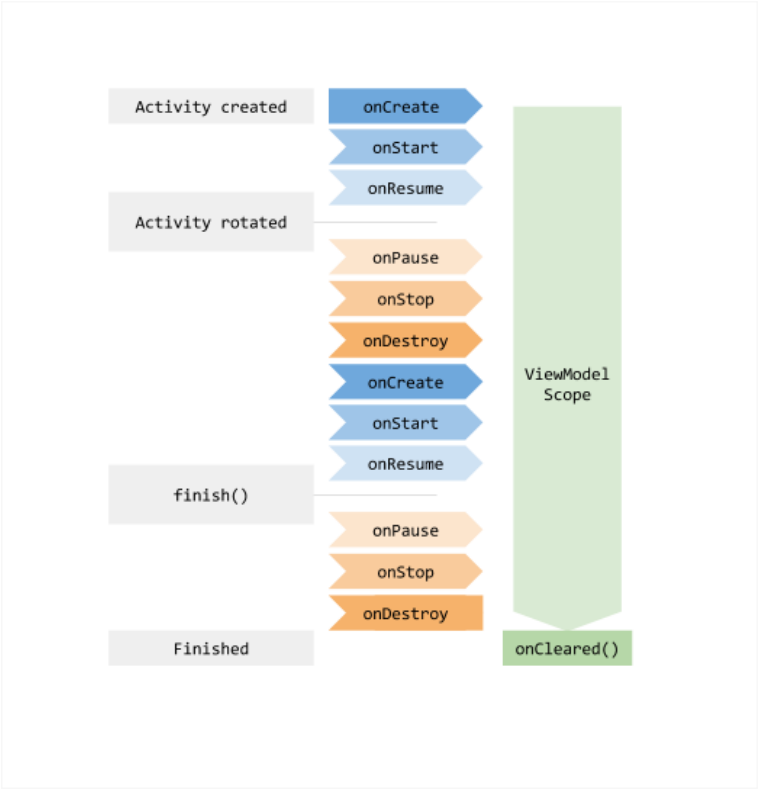

Figura 5.2. Ciclo de vida de un `ViewModel` [10]

Como se observa, a pesar de que la actividad asociada al `ViewModel` sufre varios cambios de configuración, el `ViewModel` conserva el mismo estado, hasta que se destruye finalmente junto a la actividad. Con los fragmentos ocurre lo mismo, con su ciclo de vida análogo.

Para utilizarlo, simplemente hay que crear una clase propia que herede de `ViewModel`, de manera que podemos crear los métodos que sean necesarios en un determinado contexto, siempre manteniendo las propiedades que `ViewModel` proporciona. Estas clases, por convención suelen llevar el sufijo "-ViewModel", y comparten prefijo con la actividad o fragmento del que almacenan información.

5.3. Patrones de diseño utilizados

A lo largo del desarrollo del proyecto se han utilizado distintos patrones de diseño para resolver varios problemas que se han presentado. En esta sección se detallarán los principales patrones utilizados.

5.3.1. Patrón DAO/DTO

El patrón DAO/DTO en realidad se refiere a dos patrones de diseño distintos usados en conjunto.

Por una parte, el **patrón DAO (*Data Access Object*)** [85] u Objeto de Acceso a Datos, es un patrón de diseño que propone separar la lógica de negocio de la lógica de acceso a datos, proporcionando en una clase (el DAO) la interfaz de operaciones que se pueden realizar con la información contenida en la fuente de los datos.

Por otra parte, el **patrón DTO (*Data Transfer Object*)** [121] u Objeto de Transferencia de Datos, es un patrón que proporciona un objeto (el DTO) que transporta información entre procesos o capas, con el fin de reducir el número de llamadas y por tanto mejorar la eficiencia.

En la Figura 5.3 se muestra una combinación de ambos patrones de diseño.

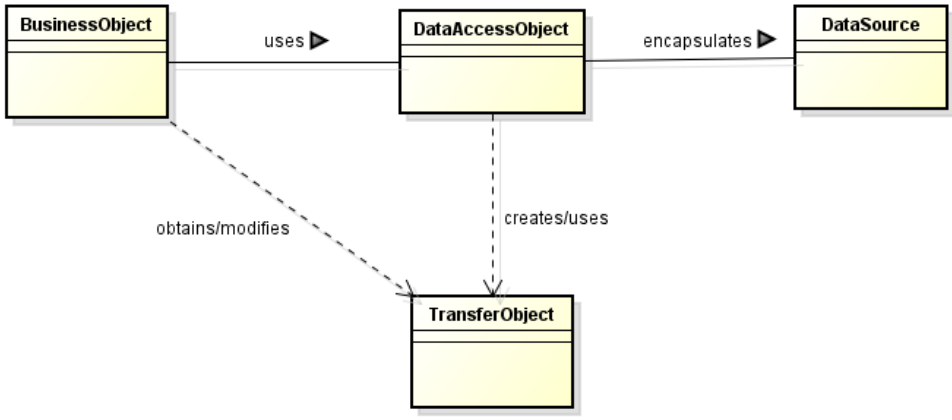

Figura 5.3. Patrón DAO/DTO [71]

Uso en el proyecto

En la aplicación, existe un paquete llamado **daos**, que contiene las clases DAO con las operaciones para cada tipo de entidad que se guarda en la base de datos. A su vez, estas clases suponen el acceso a la parte de Modelo del patrón MVVM, pues los ViewModels accederán a la información de la base de datos a través de las operaciones de las distintas clases DAO. También, existe un paquete llamado **entities**, en el que se encuentran las entidades del dominio en forma de clases POJO (*Plain Old Java Object*, clases con atributos, getters,

setters y constructores). Este paquete supone una capa transversal accesible por todos los demás paquetes, ya que estas entidades funcionan como DTO y son la manera en la que se transmite la información entre todas las capas. Desde el momento en que se recupera información de la base de datos a través de los DAO, ésta es transformada en un objeto de una clase del paquete **entities**, y viaja como DTO hasta los fragmentos y actividades, pasando por los ViewModels correspondientes.

5.3.2. Método Factoría

El **método Factoría** [91] [125] es un patrón de diseño creacional (destinado a la creación de objetos) cuya intención es definir una interfaz para crear un objeto mediante un método fábrica, en lugar de utilizar el operador `new`. El objetivo es crear el tipo adecuado de objeto (una de las subclases de la clase fábrica) a través de una única interfaz (la superclase o clase fábrica) cuando no se pueda anticipar el tipo de objeto que se quiere crear.

En la Figura 5.4 se puede ver la estructura general del patrón. En este caso, la clase factoría es "Creador", que contiene un método fábrica o factoría, y que a su vez tiene una subclase "CreadorConcreto" que se encarga de crear el tipo adecuado de objeto, en este caso "ProductoConcreto".

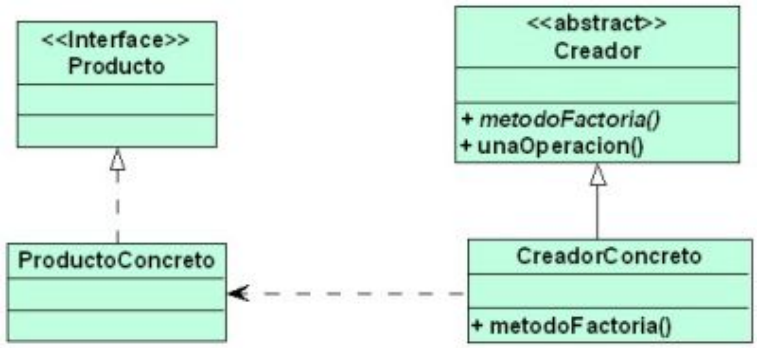

Figura 5.4. Patrón Factoría [65]

Uso en el proyecto

En Android, puesto que para ser seguros ante los cambios del ciclo de vida de su controlador de IU los ViewModels se deben crear a través de una clase `ViewModelProvider`, proporcionada por Android también, y mediante esta clase sólo se pueden crear ViewModels con su constructor por defecto (sin parámetros), para crear ViewModels con datos inicializados desde su creación es necesario utilizar el método factoría.

La solución que ofrece Android se basa en crear una subclase que herede de la clase `ViewModelProvider.Factory`, en la que se creará mediante un método factoría el ViewModel correspondiente con sus parámetros en el constructor. Esta nueva clase factoría se pasará ahora al `ViewModelProvider`, y Android se encargará internamente de instanciar el ViewModel con los datos iniciales indicados.

5.3.3. Patrón Adaptador

El **patrón Adaptador** [90] [125], también llamado envoltorio o *wrapper*, tiene como objetivo convertir la interfaz de una clase en otra esperada por los clientes. Un adaptador envuelve un objeto para ocultar la complejidad de las conversiones necesarias para adaptar una interfaz a otra, de manera que un cliente que quiera utilizar una interfaz no compatible, lo pueda hacer a través del mencionado adaptador.

En la Figura 5.5 se puede observar la estructura del patrón Adaptador. La clase "Target" haría referencia a la interfaz esperada por el cliente, mientras que la clase "Adapter" es el adaptador, que permite utilizar los métodos de la clase "Adaptee", los que realmente se desean utilizar.

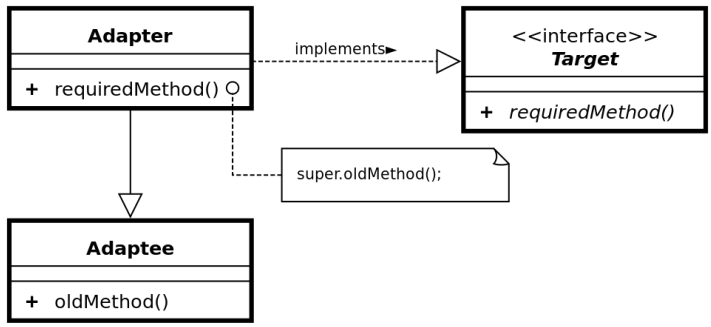

Figura 5.5. Patrón Adaptador [123]

Uso en el proyecto

Junto con el componente Android `RecyclerView`, utilizado para mostrar una lista de elementos que se añaden dinámicamente, las clases clave que se han de crear son: un contenedor de vistas, clase que extiende `RecyclerView.ViewHolder`, que representará cada uno de los elementos de la lista dinámica, y un adaptador que extiende `RecyclerView.Adapter`. Esta última clase es la que aplica el patrón Adaptador, ya que su función es convertir la lista de objetos entidades que se le pasa por parámetro a objetos ViewHolder que se mostrarán por

pantalla, y adaptar los métodos que se quieran realizar sobre estos elementos. El procedimiento es tan simple como crear un adaptador y asignárselo al `RecyclerView`, Android se encargará internamente del resto.

5.3.4. Patrón Proxy

El **patrón Proxy** [92] proporciona un objeto que actúa como sustituto de otro objeto o servicio real utilizado por un cliente. Este objeto proxy recibe solicitudes del cliente, realiza parte del trabajo y finalmente pasa la solicitud al objeto o servicio real que corresponde. Al poseer la misma interfaz que el objeto o servicio real, éstos son totalmente intercambiables.

En la Figura 5.6 se observa la estructura que sigue el patrón. La clase "Proxy" es la que actúa como sustituto y delega algunas tareas en "RealSubject", que es el objeto o servicio real. Ambas poseen la misma interfaz, "Subject", que es la conocida por el cliente.

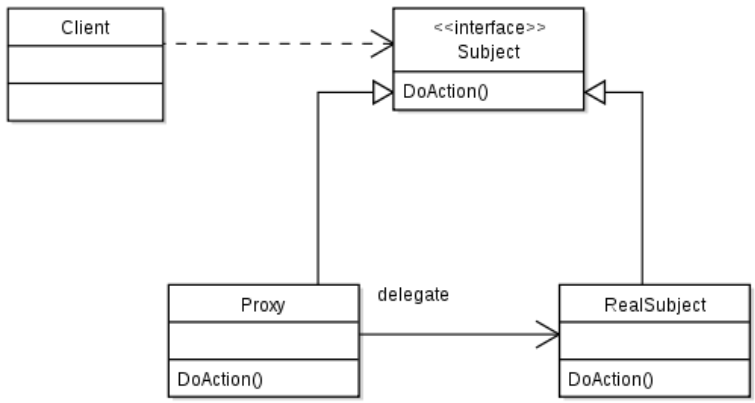

Figura 5.6. Patrón Adaptador [89]

Uso en el proyecto

Debido a la asincronía que proporciona la base de datos en *Firebase* y la estructura dada a los datos en los nodos de la base de datos, era difícil poder obtener de forma completa objetos entidades que a su vez tuvieran dependencia con otras entidades, dado que había que consultar en varios nodos distintos y la asincronía complicaba este trabajo. Por ejemplo, una cita está relacionada con un establecimiento pero también puede estarlo con un cliente, por lo que al recuperar una cita también habría que consultar los nodos de establecimiento y probablemente de cliente.

La solución que se propuso a este problema para simplificarlo, es una adaptación del patrón Proxy, de manera que cuando se recupere un objeto que tenga atributos entidades, sólo

se pase a los ViewModel estos atributos con contenedores vacíos, excepto por su ID, a modo de objeto proxy. Por ejemplo, al consultar una cita, sus atributos de tipo `Establishment` y `Client` serán contenedores vacíos que simplemente contengan el ID de esos objetos, ninguna información más.

Esta forma de uso del patrón Proxy se aplica en contextos donde se busca una recuperación perezosa.

El patrón, sin embargo, sólo se aplica entre los paquetes **viewmodels** y **daos**, puesto que con los contenedores vacíos que llegan a los ViewModels se realiza una nueva consulta con el ID que contengan para obtener la entidad de tipo correspondiente. Así, cuando desde un ViewModel se hace una consulta de cita y se recibe un objeto `Appointment` con sus proxys establecimiento y cliente, posteriormente se realizarán dos consultas a los DAO correspondientes, primero al de `Establishment` y después al de `Client`, en cada caso con el ID del proxy correspondiente. De esta manera, se garantiza que en la capa de interfaz de usuario (paquete **views**) los objetos lleguen en su estado natural, con sus dependencias completas, puesto que todo el trabajo para ello se ha realizado en capas inferiores.

5.4. Diseño de la interfaz de usuario

Previo al comienzo de la implementación de las historias de usuario, se realizaron durante la fase de planificación inicial en el sprint 0 (se explicará el seguimiento del proyecto en el Capítulo 7) varios *mockups* o prototipos que muestran un boceto de las distintas pantallas de la app. El objetivo era facilitar la fase de elicitación de requisitos, mostrando a los clientes potenciales entrevistados prototipos de la aplicación para dar una visión más representativa de la idea, y posteriormente durante el desarrollo de las historias de usuario, tener una base sobre la que comenzar el desarrollo de la interfaz de usuario.

Las Figuras 5.7 a 5.46 muestran los *mockups* que se realizaron. El pie de foto de cada uno de ellos sigue el formato: "Descripción de la pantalla - *Rol de usuario*".

Figura 5.7. Vista principal de "Sitios" - *Cliente*

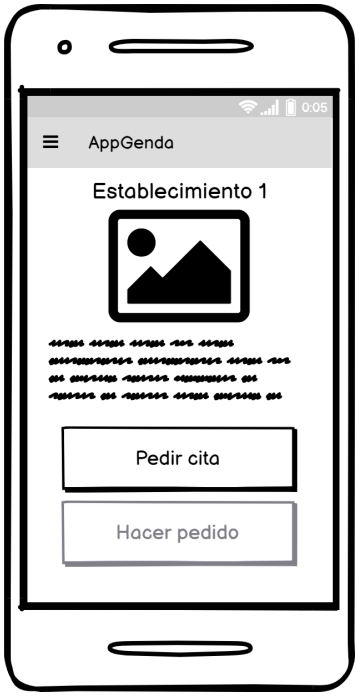

Figura 5.8. Establecimiento seleccionado - *Cliente*

Figura 5.9. Pedir cita (seleccionar día) - *Cliente*

Figura 5.10. Pedir cita (seleccionar franja horaria) - *Cliente*

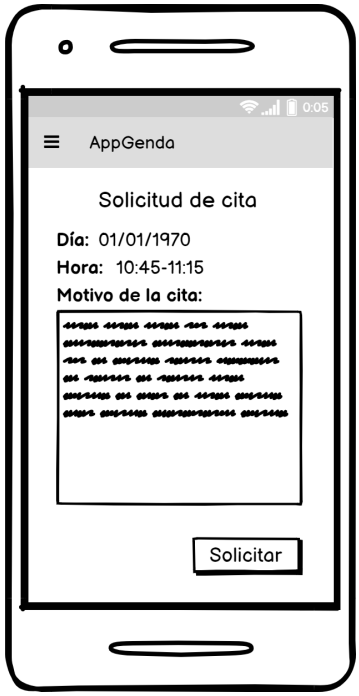

Figura 5.11. Pedir cita (detalles de la cita) - *Cliente*

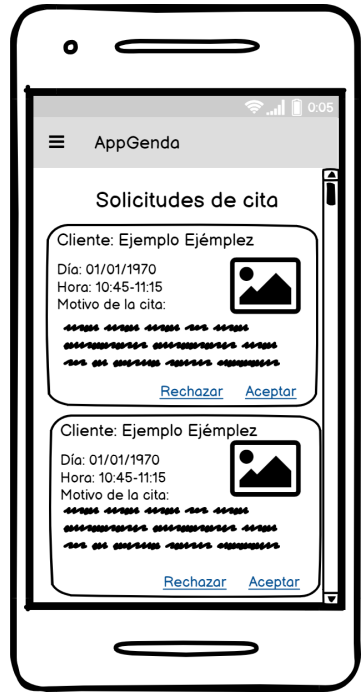

Figura 5.12. Solicitudes de cita - *Dueño de negocio*

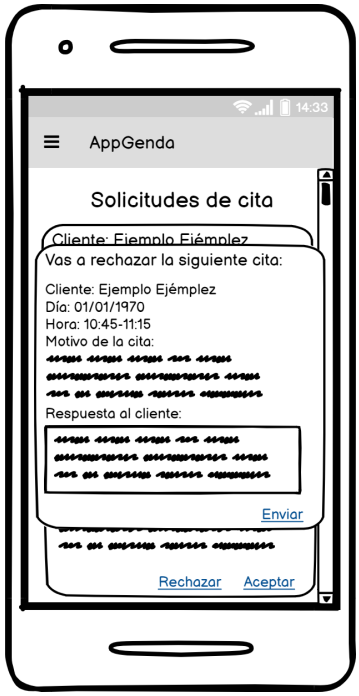

Figura 5.13. Rechazo de cita - *Dueño de negocio*

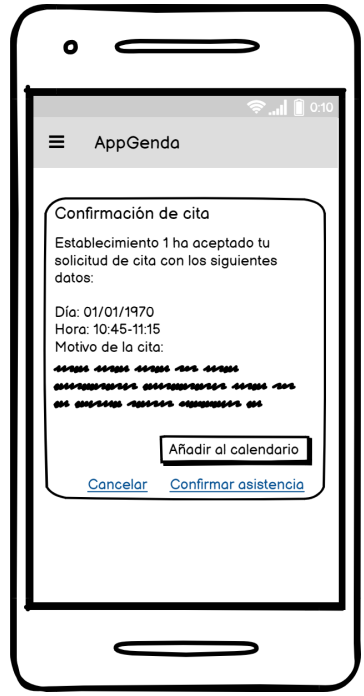

Figura 5.14. Confirmación de cita - *Cliente*

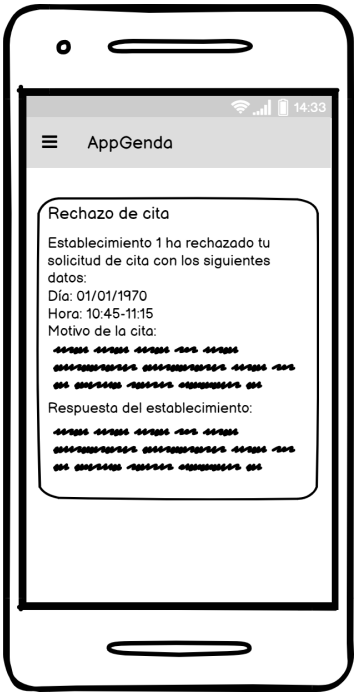

Figura 5.15. Rechazo de cita - *Cliente*

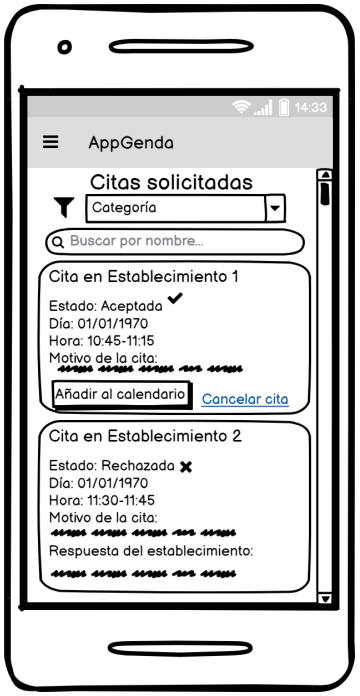

Figura 5.16. Citas solicitadas - *Cliente*

Figura 5.17. Vista principal de "Agenda" - *Dueño de negocio*

Figura 5.18. Mis citas (calendario) - *Dueño de negocio*

Figura 5.19. Mis citas (horas) - *Dueño de negocio*

Figura 5.20. Mis citas (detalles de una cita) - *Dueño de negocio*

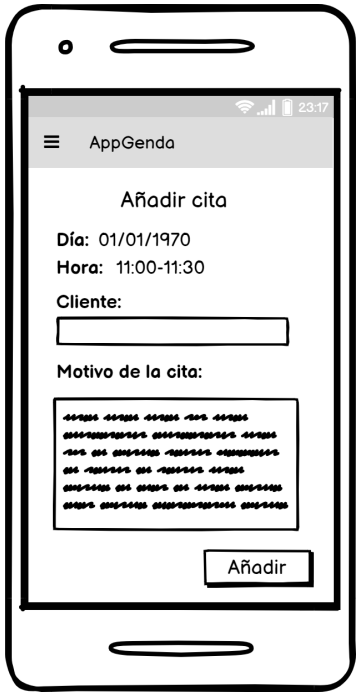

Figura 5.21. Añadir nueva cita - *Dueño de negocio*

Figura 5.22. Hacer pedido (seleccionar día) - *Cliente*

Figura 5.23. Hacer pedido (seleccionar franja horaria) - *Cliente*

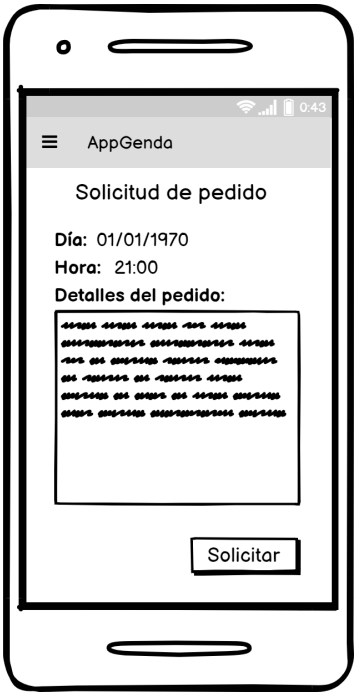

Figura 5.24. Hacer pedido (detalles del pedido) - *Cliente*

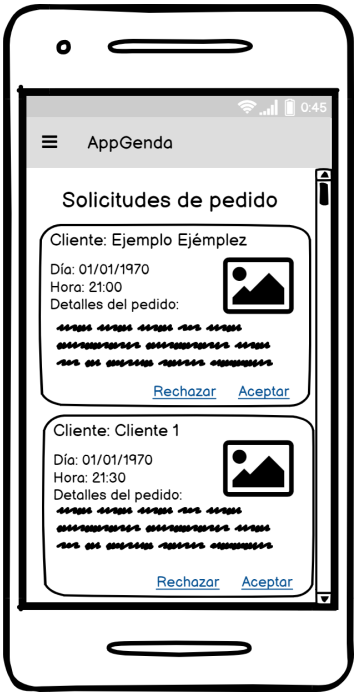

Figura 5.25. Solicitudes de pedido - *Dueño de negocio*

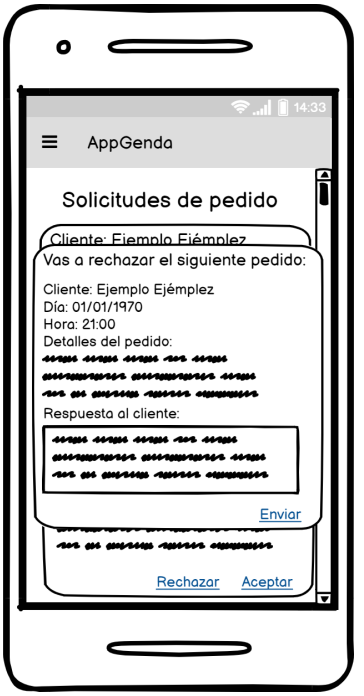

Figura 5.26. Rechazo de pedido - *Dueño de negocio*

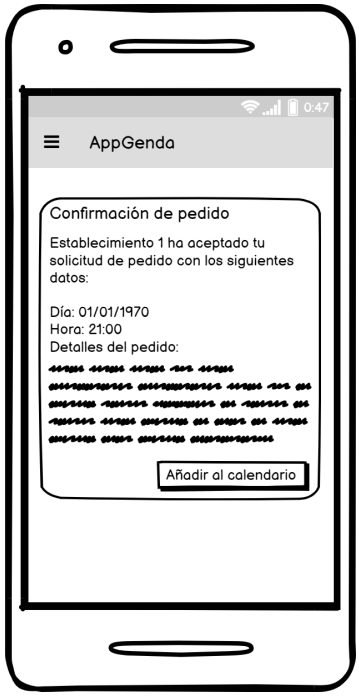

Figura 5.27. Confirmación de pedido - *Cliente*

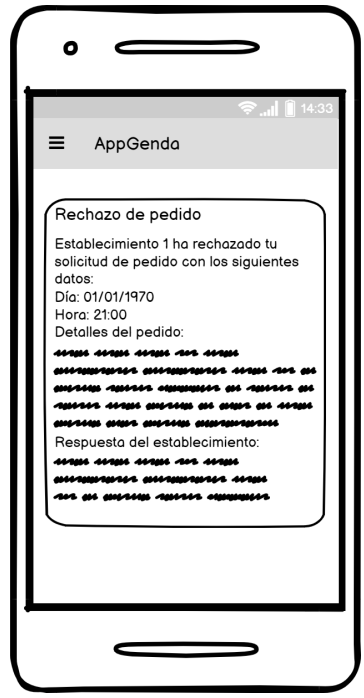

Figura 5.28. Rechazo de pedido - *Cliente*

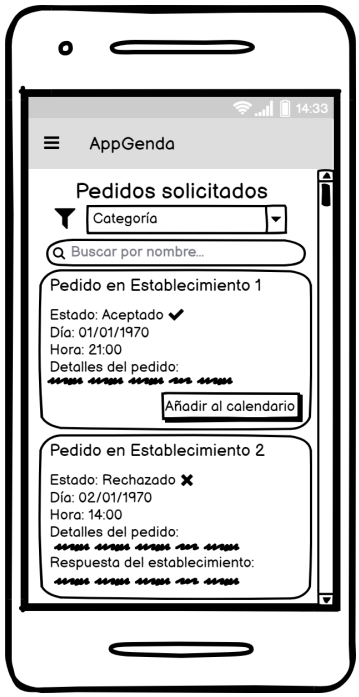

Figura 5.29. Pedidos solicitados - *Cliente*

Figura 5.30. Mis pedidos (calendario) - *Dueño de negocio*

Figura 5.31. Mis pedidos (horas) - *Dueño de negocio*

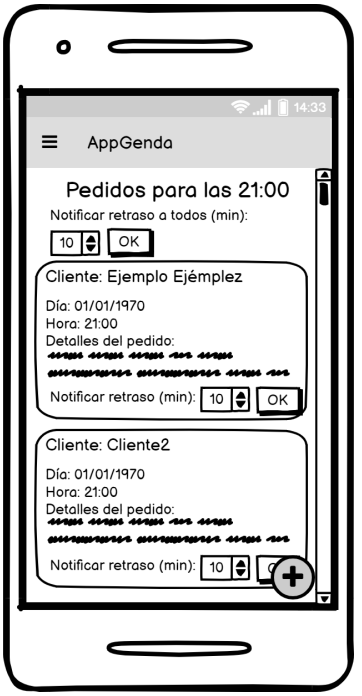

Figura 5.32. Mis pedidos (detalles de una franja horaria) - *Dueño de negocio*

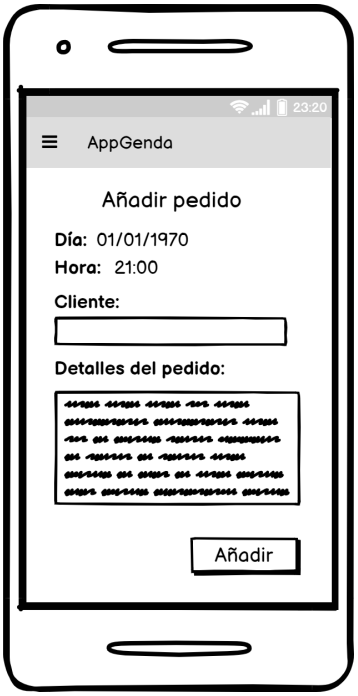

Figura 5.33. Añadir nuevo pedido - *Dueño de negocio*

Figura 5.34. Vista principal de "Mapa" - *Cliente*

Figura 5.35. Vista principal de "Mapa" - *Dueño de negocio*

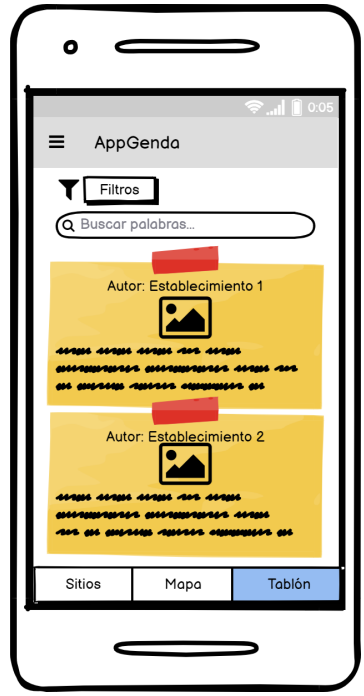

Figura 5.36. Vista principal de "Tablón" - *Cliente*

Figura 5.37. Vista principal de "Tablón" - *Dueño de negocio*

Figura 5.38. Crear nuevo anuncio - *Dueño de negocio*

Figura 5.39. Login de la app - *Usuario no identificado*

Figura 5.40. Elección de registro - *Usuario no identificado*

Figura 5.41. Registro de nuevo cliente - *Usuario no identificado*

Figura 5.42. Registro de nuevo establecimiento - *Usuario no identificado*

Figura 5.43. Solicitudes de registro de nuevos establecimientos - *Administrador*

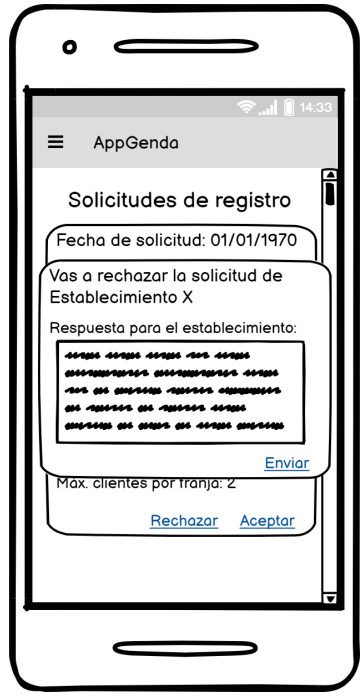

Figura 5.44. Rechazo de nuevo establecimiento - *Administrador*

Figura 5.45. Gestión de establecimientos - *Administrador*

Figura 5.46. Gestión de un establecimiento - *Administrador*

Posteriormente, en la fase de mejora de IU de los últimos sprints, se crearon unos diseños personalizados para mejorar el aspecto de la app (agradecimientos a Silvia Garrote Gascón por la ayuda prestada en el diseño de éstos). Estos diseños son los que se pueden ver en las Figuras 5.47 a 5.50.

Figura 5.47. Logo de **AppGenda**

Figura 5.48. Icono de la app

Figura 5.49. Avatar de usuario por defecto

Figura 5.50. Imagen de anuncio por defecto

Como se menciona en la Sección 5.1 de decisiones de diseño, se ha intentado respetar en todo momento las guías de diseño marcadas por *Material Design*, por lo que también se ha intentado que los diseños personalizados sigan esa misma línea.

5.5. Diseño arquitectónico

La arquitectura general de la aplicación consiste en 4 paquetes principales, alguno de ellos dividido en subpaquetes. Todos ellos están englobados dentro del paquete general **es.appgenda**. Éstos son:

- **views:** en este paquete se encuentran las clases relacionadas con la interfaz de usuario, y constituye la parte correspondiente a la Vista del patrón MVVM. A su vez, se divide en varios subpaquetes:

  - **activities:** las actividades o clases *Activity* se encuentran aquí. En este proyecto, al seguir la arquitectura *single-activity*, sólo se ha creado una actividad, pero si aumentara la complejidad del sistema y fuera necesaria otra actividad con sus respectivos fragmentos, pertenecería a este paquete.
  - **fragments:** los fragmentos o clases *Fragment* pertenecen a este paquete. Como excepción, también pertenece a este paquete la clase `AuthUtils`, con métodos de utilidad para el sistema de autenticación de la aplicación y que se ha decidido encajar en este paquete ya que sólo los fragmentos y la actividad lo utilizan.
  - **uicomponents:** los componentes de IU creados programáticamente en lugar de con XML y los adaptadores de los *RecyclerView*, se hallan dentro de este paquete.

- **viewmodels:** en este paquete se encuentran los ViewModels y sus factorías. Como su propio nombre indica, este paquete corresponde con la parte de Modelo de vista del patrón MVVM.

- **daos:** en este paquete se encuentran las clases DAO que poseen métodos para operar con la información de la base de datos. El paquete por tanto constituye el acceso a la lógica de negocio, constituyendo la parte de Modelo del patrón MVVM.

- **entities:** en este paquete se encuentran las clases de las entidades del dominio en forma de *POJO*. Se trata de un paquete transversal, ya que las entidades se han utilizado como DTO en el sistema.

Además, en los paquetes de **fragments**, **viewmodels** y **daos** se pueden encontrar tres interfaces: `AuthCallback`, `VMCallback` y `DAOCallback`, respectivamente. Estas interfaces son parte de la solución al manejo de la asincronía presente en la app, como se explicará en la siguiente sección de este Capítulo.

En la Figura 5.51 se puede observar un diagrama *Decomposition&Uses style* [87] presentando la arquitectura general de la aplicación.

Si se hiciera una correspondencia entre paquetes y capas, se podría hablar de un sistema de capas [124]. En este caso, se trataría de un **sistema de capas relajado** o arquitectura de capas abierta, pues los componentes solamente interactúan con los de su capa y los de capas inferiores (considerando la capa **entities** como la más inferior), proporcionando de esa manera cohesión entre los componentes de una misma capa y disminuyendo el acoplamiento entre capas en lo posible. Además, aunque se observa una dependencia del subpaquete **fragments** hacia el paquete **daos**, ésta proviene del uso de algún DAO en la clase de utilidad `AuthUtils` solamente, por lo que en ese caso el acoplamiento es débil.

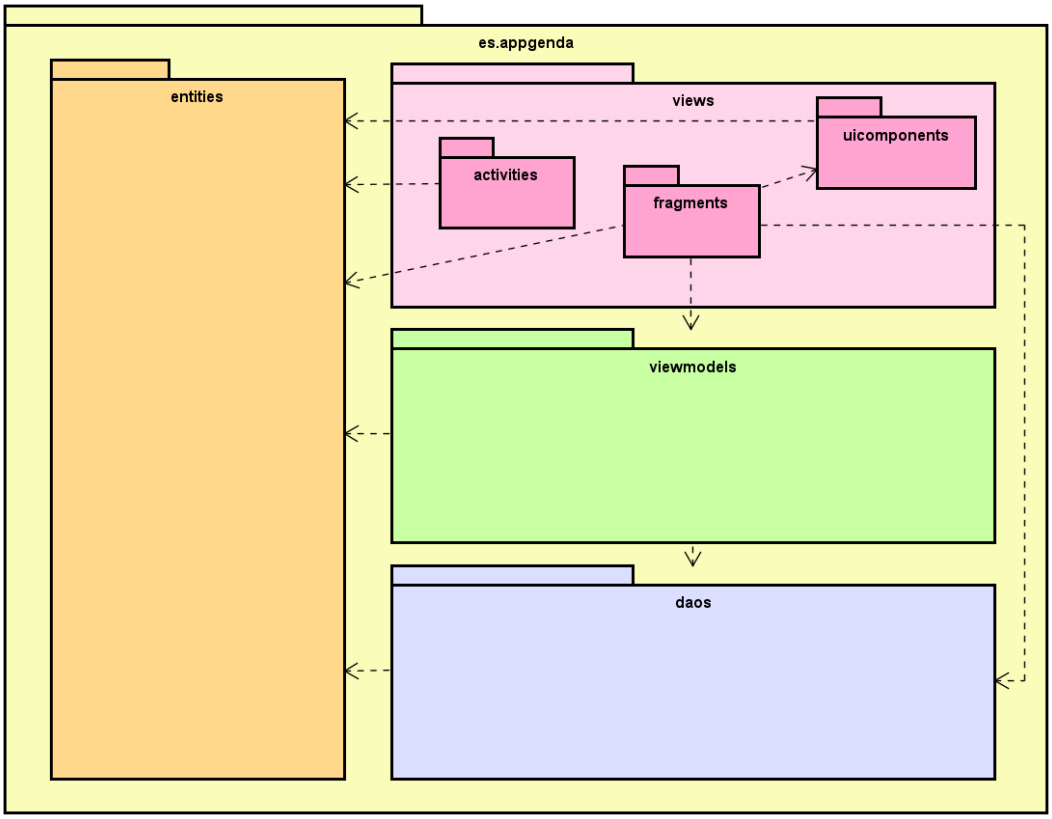

Figura 5.51. Arquitectura general de la aplicación

En las Figuras 5.52 a 5.55 aparecen los diagramas *Decomposition&Uses style* de cada uno de los paquetes principales.

Para no aumentar la complejidad de estos diagramas, se ha optado por realizar un diagrama *Decomposition&Uses style* por cada historia de usuario que utilice componentes de más de un paquete para mostrar las relaciones entre ellos. Asimismo, se ha realizado un diagrama de este tipo también para mostrar las relaciones entre distintos paquetes de la actividad principal y de la clase `AuthUtils`. Los diagramas mencionados se encuentran en el repositorio cuyo enlace se puede encontrar en el Apéndice B, y no se mostrarán en este documento por brevedad, excepto el de la **HU02**, que es la historia de usuario que se desarrollará en la siguiente sección, y cuyo diagrama *Decomposition&Uses style* se muestra en la Figura 5.62 a modo de ejemplo.

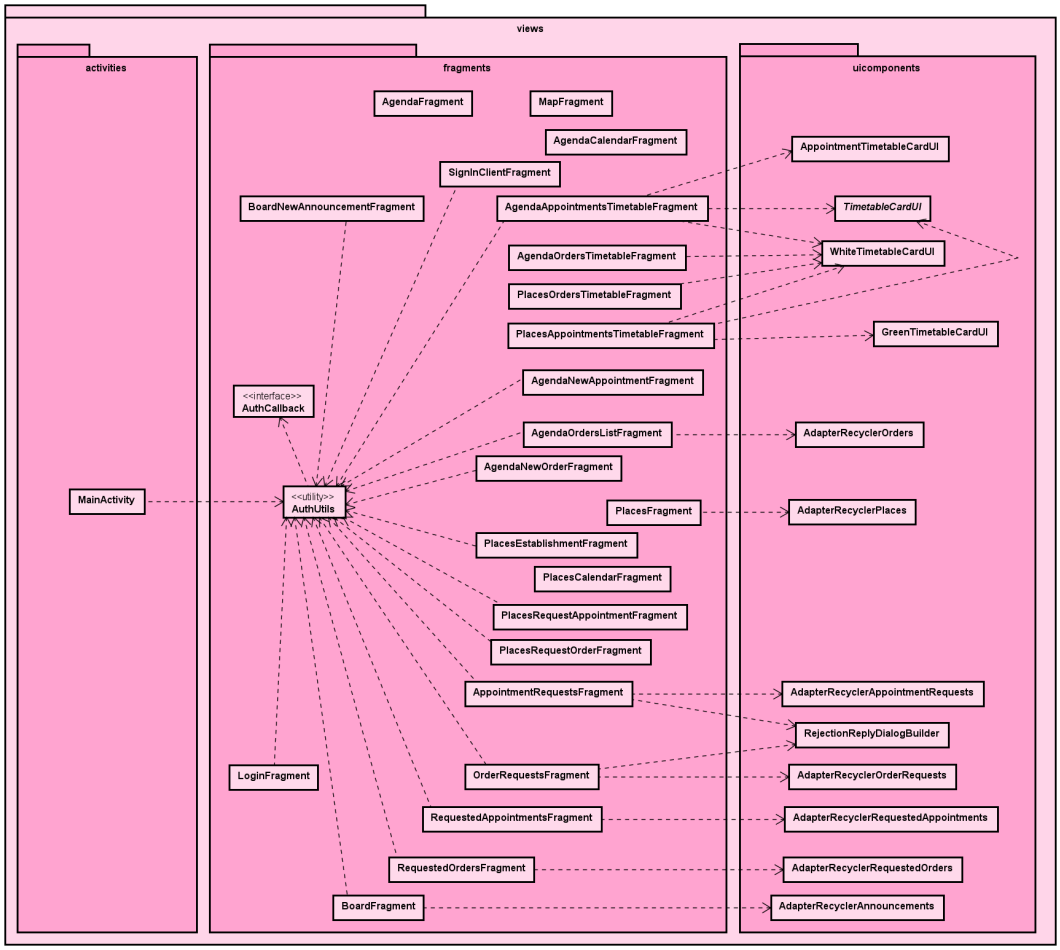

Figura 5.52. *Decomposition&Uses style* del paquete **views**

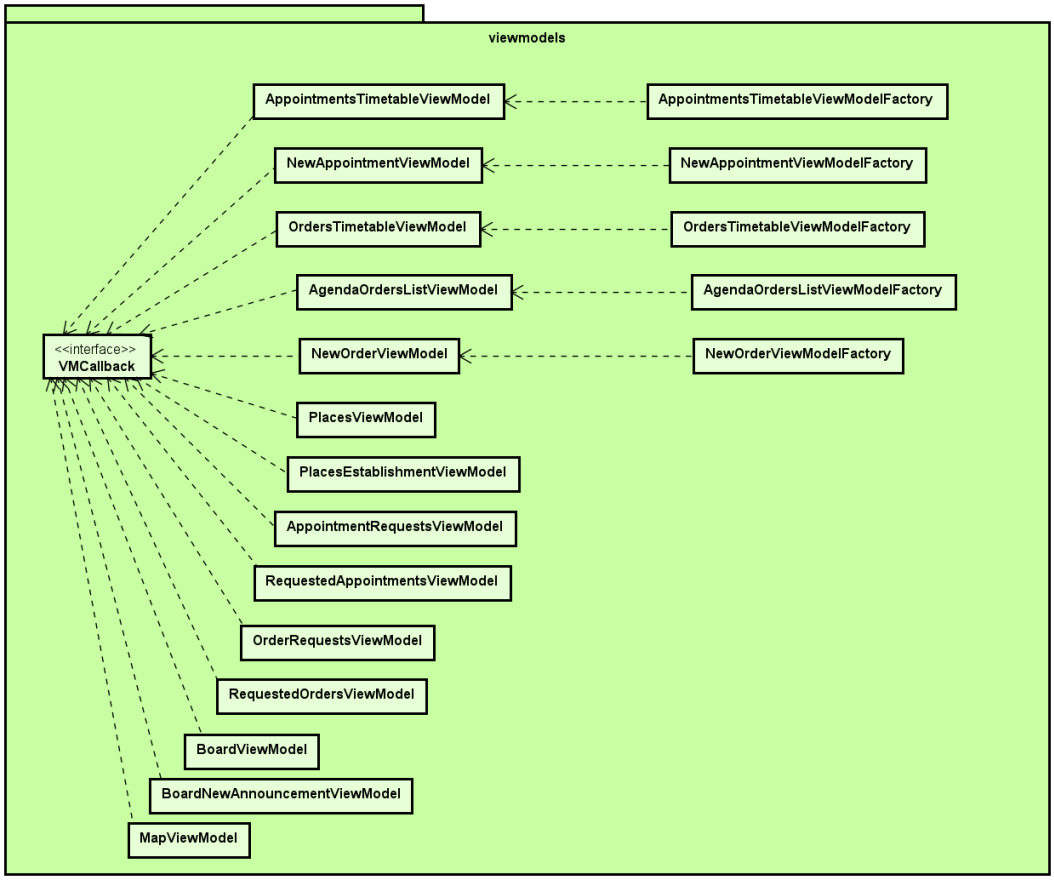

Figura 5.53. *Decomposition&Uses style* del paquete **viewmodels**

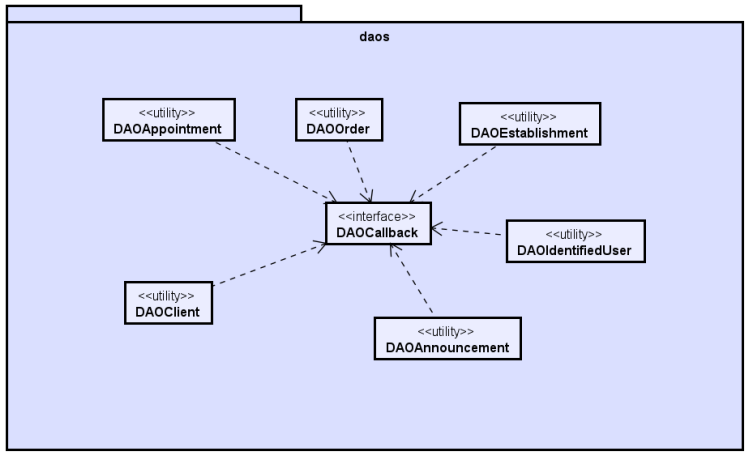

Figura 5.54. *Decomposition&Uses style* del paquete **daos**

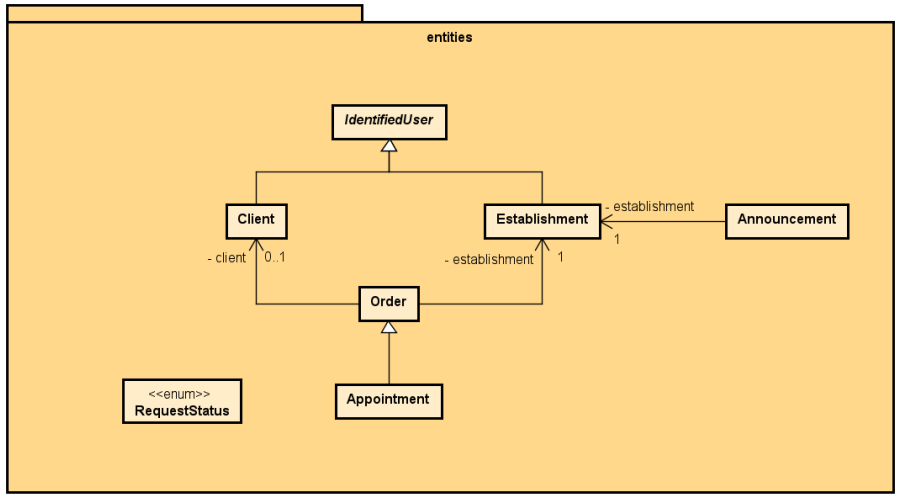

Figura 5.55. *Decomposition&Uses style* del paquete **entities**

5.6. Diseño de la comunicación entre objetos

Para explicar el diseño de comunicación entre objetos, se desarrollará y explicará el diagrama de secuencia de la **HU02: Como dueño de negocio quiero seleccionar el día del que quiero consultar las citas en un calendario para sólo ver las citas de dicho día**. Alguno de los subdiagramas de secuencia menos importantes que se omitirán por brevedad en este documento se pueden encontrar en el archivo *Astah* del repositorio que se puede encontrar en el Apéndice B. También, con el objetivo de reducir la complejidad, se obviará la obtención de ciertas variables como el contexto de la actividad principal y el uso de algunos componentes de IU como `progressBarTimetable`. Se recomienda observar el *mockup* correspondiente (Figura 5.19) para visualizar el resultado que se desea conseguir con el siguiente desarrollo, que puede resultar un poco complejo debido a la cantidad de clases de distintos paquetes utilizadas y la variedad de técnicas aplicadas.

En la siguiente explicación, se utilizará informalmente el nombre de las clases en castellano para facilitar la comprensión por parte del lector, aunque en los diagramas éstas aparezcan con su nombre original, en inglés.

El diagrama de la secuencia principal de dicha historia de usuario se puede ver en la Figura 5.56. En el comienzo del diagrama vemos como el actor (dueño de negocio) primero ha realizado la **HU01**. Una vez en la pantalla del calendario, el usuario elige una fecha (en este ejemplo 14 de marzo del 2021) y el objeto `listenerDatePickerCalendar`, que es de una clase activa porque está esperando a que ocurra un evento, realiza las operaciones correspondientes englobadas en el nodo ref "Seleccionar una fecha del calendario" (desarrollado en el archivo *Astah*). La principal operación ejecutada es navegar hacia el fragmento `AgendaAppointmentsTimetableFragment`, tras lo que Android crea automáticamente dicho fragmento. El primer método ejecutado, según el ciclo de vida de los fragmentos, es `onCreate`, que en este caso simplemente invoca al mismo método de la clase padre. El segundo método se trata de `onCreateView`, donde sucederá todo lo restante del diagrama a partir de ahora. En este método, la primera operación realizada es crear una referencia a las vistas del XML correspondiente, obtener los argumentos del fragmento y crear el ViewModel correspondiente, englobado bajo el nodo ref "Encontrar vistas, argumentos y crear ViewModel".

En la Figura 5.57 se desarrollan estas operaciones. Primero, la vista que se devolverá se "infla" desde su XML correspondiente (`R.layout.fragment_timetable`), y después, para encontrar los componentes de IU en esa vista, se invoca sobre ella el método `findViewById`, pasando el ID correspondiente de la vista buscada. Tras encontrar los tres componentes buscados en este caso (y por tanto creados como objetos), se obtendrán los argumentos recibidos en el fragmento mediante la clase generada mediante *SafeArgs* `AgendaAppointmentsTimetableFragmentArgs`, en este caso día, mes, año y un boolean que indicará si hay que mostrar en rojo las franjas del horario completamente llenas. Finalmente, para crear el ViewModel, se crea un objeto factoría al que se le pasan los parámetros necesarios para su inicialización (día, mes y año), y creando un `ViewModelProvider` al que se le pasa en su constructor la factoría ViewModel que se acaba de crear, se obtiene finalmente el objeto de tipo `AppointmentsTimetableViewModel`.

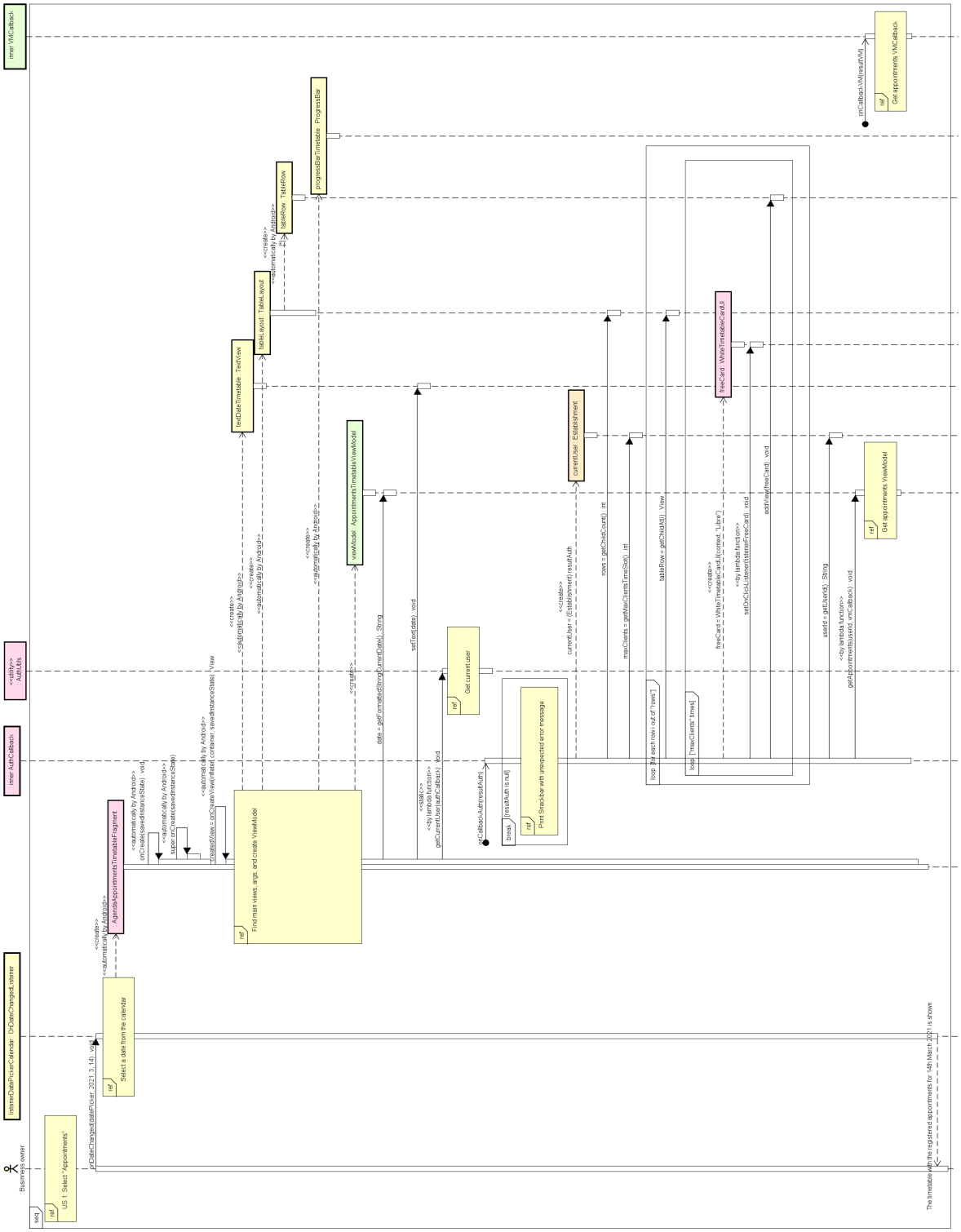

Figura 5.56. Diagrama de secuencia principal de la **HU02**

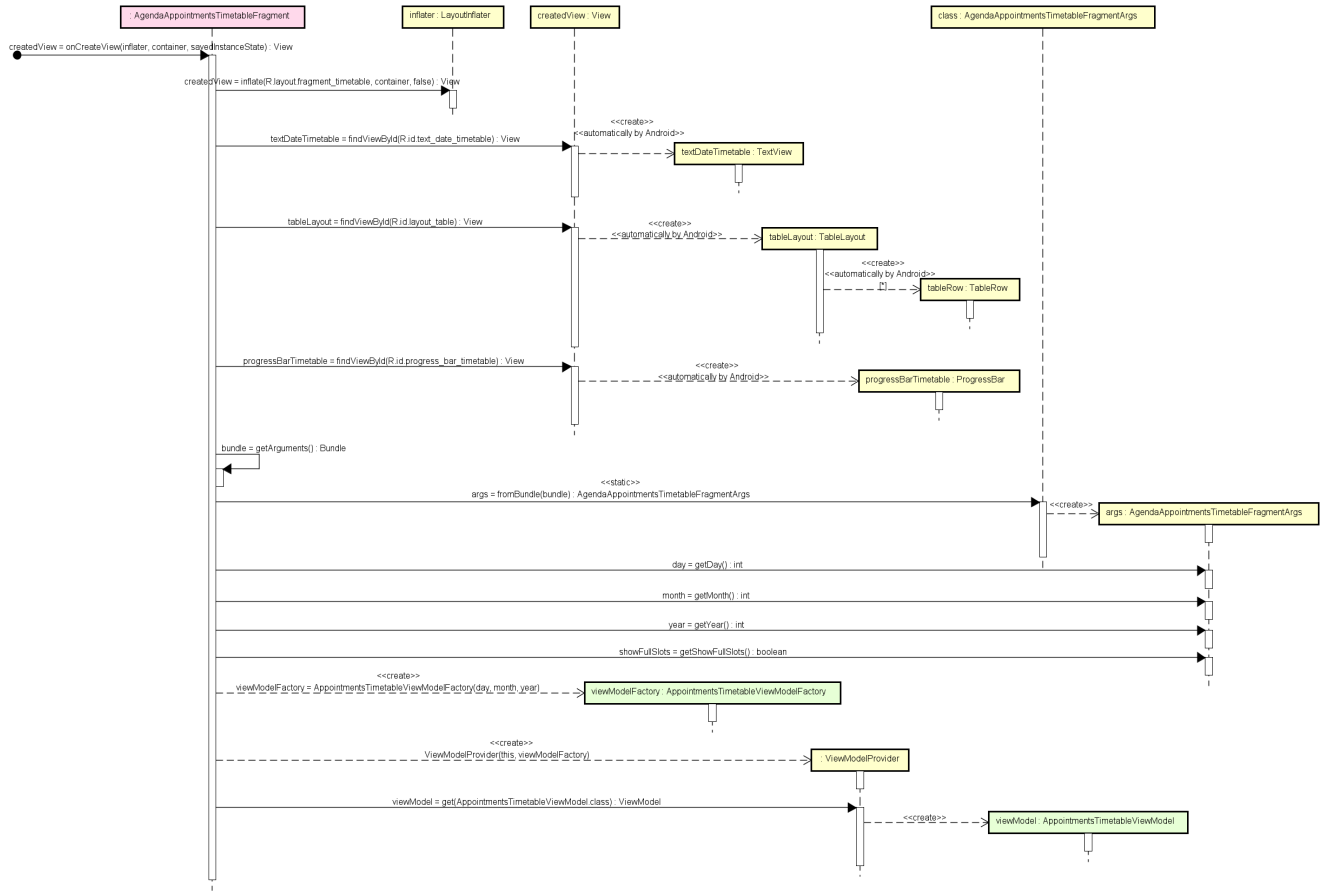

Figura 5.57. Diagrama de secuencia de "Encontrar vistas, argumentos y crear ViewModel"

Volviendo a la Figura 5.56, vemos que, como resultado de las acciones ocultas por el *interaction use* (nodo ref) desarrollado, se crean (mediante el estereotipo «`create`») los tres componentes de IU y el ViewModel. Lo siguiente que se hace es pedirle al ViewModel la fecha en formato DD/MM/AAAA para establecérsela al texto del título de la pantalla.

El siguiente paso es obtener el usuario actual en la sesión, que al encontrarse en esta pantalla, sólo se puede tratar de un usuario tipo Establecimiento. Para ello, se invoca al método `getCurrentUser` de la clase de utilidades `AuthUtils`. En este punto ocurre algo importante, puesto que se trata de la manera que se ha manejado la asincronía que proporciona *Firebase*, tanto en esta parte de la aplicación como en todas las demás que así lo requieran.

La solución aplicada se basa en **callbacks**, que no son más que interfaces de un único método que devuelve `void` y posee un parámetro de tipo `Object`. El funcionamiento se basa en pasar a los métodos asíncronos como argumento un callback, (`authCallback` en este caso),

que al ser una interfaz, hay que implementar. Desde el método asíncrono, una vez que se han terminado todas las tareas y se esté preparado para devolver el control a la clase que lo ha invocado, en lugar de hacer `return` como se haría típicamente en métodos síncronos, se invoca al método del callback recibido por parámetro con el resultado que se quiera devolver. Como el parámetro del método del callback es de tipo `Object`, se podrá hacer casting a cualquier tipo. La implementación del método del callback se desarrollará en la propia clase que se invoca al método asíncrono mediante una función lambda (de ahí el estereotipo «`by lambda function`»), por tanto como clase interna o *inner class*. La ejecución de este código comenzará cuando desde el método asíncrono se invoque al método del callback, asegurando de esta manera que el método asíncrono ha terminado de ejecutarse y el resultado del mismo está en el argumento de tipo `Object`, al que se hará casting al tipo que proceda.

Por ello, tras la invocación del método `getCurrentUser` y lo que ocurre en su interior (que no se desarrollará puesto que se verá más tarde con otro ejemplo), se invoca de manera asíncrona al método `onCallbackAuth` de la clase interna `inner AuthCallback`, que corresponde a la implementación del callback pasado por parámetro a `getCurrentUser`, y desde donde ocurrirán el resto de operaciones a partir de este momento. La primera comprobación corresponde a si el parámetro recibido en el método del callback es `null`, puesto que por la decisión de diseño tomada (Sección 5.1), significará que ha ocurrido un error, y se imprimirá un mensaje y se parará el flujo de ejecución (de ahí el bloque `break`, que pararía el bloque `seq` que engloba todo el diagrama de secuencia). De no ser `null`, entonces se hará un casting del parámetro a tipo `Establishment`, resultando en el objeto `currentUser`.

Lo siguiente que ocurre son dos bucles anidados, que llenan el horario de tarjetas libres (`WhiteTimetableCardUI`) en todas las casillas del mismo. Esto se hace un total de `rows` (número de filas del tablero) x `maxClients` (máximo de clientes por franja del establecimiento) veces, para llenar el horario completamente. Posteriormente, se sustituyen las tarjetas libres correspondientes con las citas ya registradas. Para ello, se invoca el método `getAppointments` del ViewModel pasando como parámetro el ID del usuario actual. Como llegará hasta el DAO (donde las operaciones se realizan contra la base de datos de *Firebase*), se realiza de manera asíncrona, por lo que se vuelve a utilizar de nuevo un callback, en este caso `vmCallback`. El contenido de este *interaction use* (nodo ref) “Obtener citas ViewModel” se desarrollará en otro diagrama de secuencia.

En la Figura 5.58 se observa el desarrollo del *interaction use* mencionado. Tras hacer unas comprobaciones iniciales sobre los parámetros, se invoca al método `getAcceptedAppointmentsForDateAndEstablishment` de `DAOAppointment`, cuyo desarrollo se puede ver en la Figura 5.59. Tras otras comprobaciones iniciales, se crea una lista de `Appointments` que servirá para ir guardando las citas que se obtengan de la base de datos. Como resultado de la consulta, se obtiene un objeto de tipo `Task<DataSnapshot>`, al que se asocian dos listeners, uno en caso de que no haya errores (`OnSuccessListener`) y otro en caso de que ocurra algún error (`OnFailureListener`). Aquí nace la raíz de la asincronía, pues estos listeners son invocados de manera asíncrona, y por eso pertenecen a clases activas. Una vez se recuperan los datos de la base de datos correctamente, se invoca de manera asíncrona el listener `OnSuccessListener` (Figura 5.60)

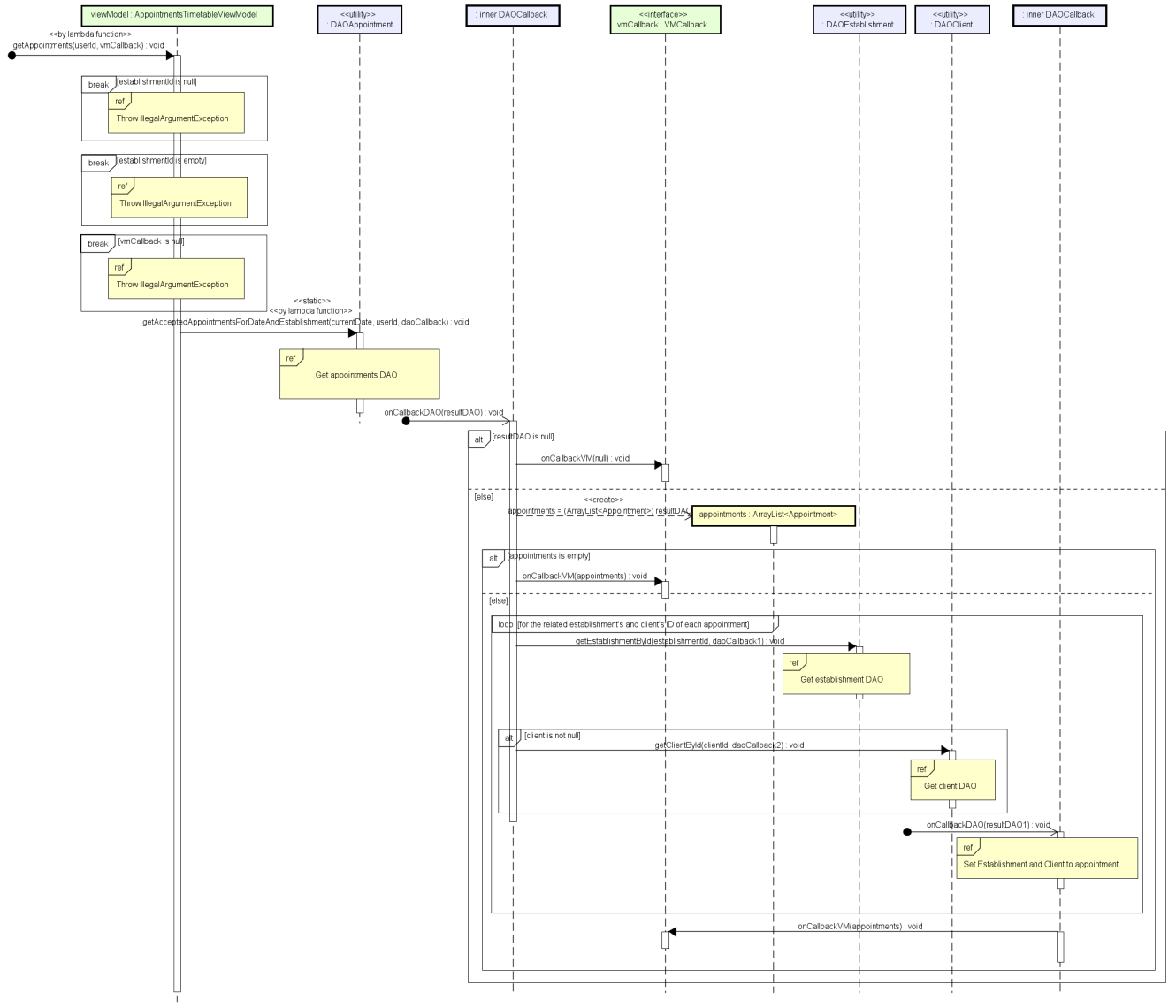

Figura 5.58. Diagrama de secuencia de "Obtener citas ViewModel"

En dicho listener, se recorren todos los nodos hijos del nodo `appointments` de la base de datos. Convirtiendo la fecha, ID del establecimiento asociado y estado de cada una de las citas a los tipos necesarios, se comparan con los recibidos como argumento en el método del DAO, y en caso de coincidir tanto en fecha como en ID de establecimiento y estar en estado "aceptado", se construye el objeto Cita correspondiente que funcionará como DTO, seteando

sus atributos con los datos del nodo de la base de datos, y creando contenedores vacíos (con el ID) para los atributos de tipo Establecimiento y Cliente, aplicando así el patrón Proxy (nodo ref "Construir cita", desarrollado en el archivo *Astah*), tras lo que se añadirá a la lista creada para almacenar las citas resultantes de la consulta. Una vez recorridos todos los nodos de la base de datos y finalizado el bucle, se invocará al método del callback `onCallbackDAO` pasándole como argumento la lista de citas, lo que nos lleva de vuelta al diagrama de la Figura 5.58, en la parte de la invocación asíncrona a `onCallbackDAO`.

Tras realizar la comprobación de si `resultDAO` (el resultado de la consulta en el DAO, en este caso la lista de citas) es `null`, en cuyo caso habría ocurrido un error y se devolvería al callback del ViewModel otro `null`, se comprueba si la lista de citas (ya casteada a su tipo correspondiente) está vacía, en cuyo caso se devolvería al callback del ViewModel. En caso contrario, para cada cita de la lista, se obtendrá su atributo de tipo Establecimiento mediante el ID que nos ha proporcionado el objeto proxy, invocando al método `getEstablishmentById` de `DAOEstablishment`, que no se desarrollará ya que sigue el mismo mecanismo que la obtención de citas. En caso de que el atributo de tipo Cliente no sea `null` (la cita está asociada a un cliente), se seguirá el mismo procedimiento, invocando a `getClientById` de `DAOClient`. Tras setear a la cita su atributo Establecimiento (y Cliente si se ha realizado la consulta) en un nuevo nivel de callback que se puede ver a la derecha del diagrama, se devuelve el control al fragmento invocando al método `onCallbackVM` de `vmCallback`, volviendo a la Figura 5.56.

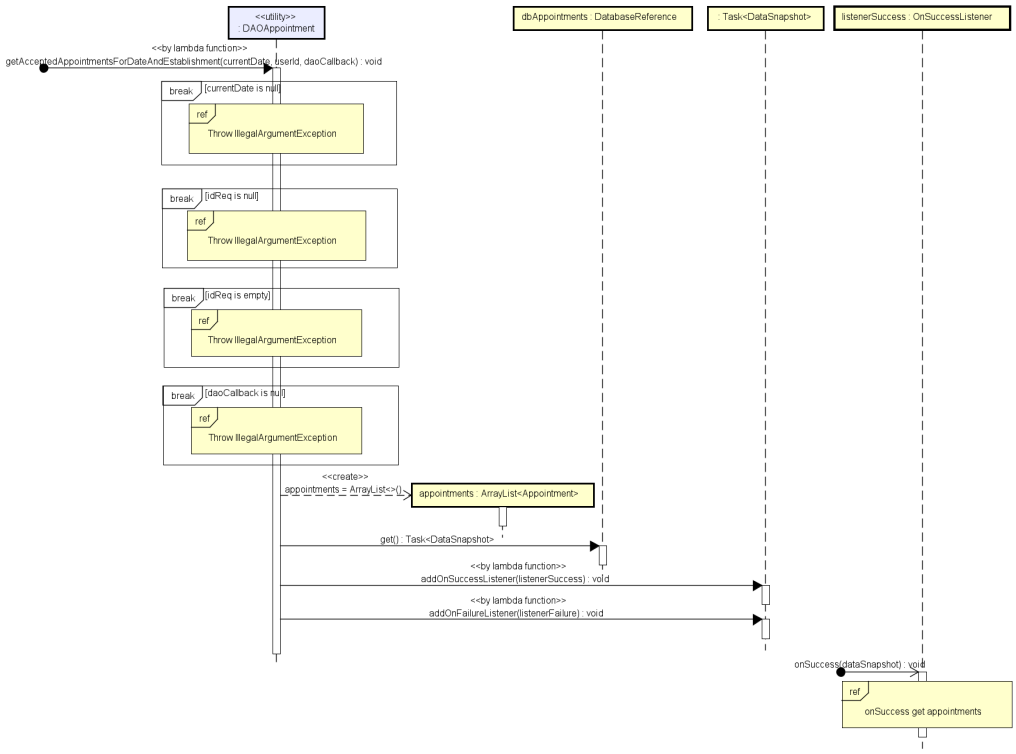

Figura 5.59. Diagrama de secuencia de "Obtener citas DAO"

Tras haber recuperado el control en el fragmento, nos hallamos en la invocación asíncrona de `onCallbackVM`, como se ve en la parte más a la derecha de la Figura 5.56. Esta última parte del diagrama se desarrolla en la Figura 5.61.

Lo primero que se hace es la comprobación de si el resultado recibido es `null`, en cuyo caso se imprimiría un mensaje de error al usuario y se rompería el flujo del diagrama. De no ser así, se hace un casting al resultado recibido para convertirlo en una lista de citas. Para cada una de esas citas, se construirá una tarjeta de cita (`AppointmentTimetableCardUI`), seteando su listener (que se encargará de mostrar los detalles de la cita en una ventana de diálogo). Para insertar la tarjeta de cita en la franja horaria correspondiente, se recorrerá en bucle todas las franjas hasta encontrar la que marque la hora de comienzo de la cita, donde se insertará la tarjeta. Si además resulta que la cita ocupa más de una franja horaria de 15 minutos, se calculará cuántas franjas más tiene que ocupar, y para cada una de estas franjas adicionales, se creará una tarjeta de cita adicional similar a la primera creada, y se insertará en ella. La razón de crear una nueva tarjeta y no utilizar la original es que en Android, cada componente de IU sólo puede poseer una vista padre, por lo que se hace necesario crear más tarjetas para poder introducirlas en filas distintas del horario. La acción de insertado de tarjeta en este documento se omite, pero se desarrolla en el archivo *Astah*.

Finalmente, siguiendo en la Figura 5.61, se invoca al método `showFullSlots`, que en función del boolean recibido por argumento al principio de la construcción del fragmento, mostrará las tarjetas de las franjas sin huecos libres en rojo. Esto tiene utilidad cuando la navegación proviene de la pantalla de añadir una nueva cita y se elige un horario para la cita que ya está completo, de manera que se da una respuesta más visual al usuario, pero en la historia de usuario aquí desarrollada no es importante. El desarrollo de "Mostrar franjas llenas" se puede encontrar en el archivo *Astah*.

Para terminar con el diagrama de secuencia, y volviendo al diagrama principal (Figura 5.56), tras la invocación asíncrona de `onCallbackVM`, el método `onCreateView` del fragmento que se está construyendo termina, concluyendo así su construcción y mostrando por pantalla el horario de citas para el día 14 de marzo de 2021 del establecimiento que ha iniciado sesión, con las casillas del horario libres ocupadas por una tarjeta de hueco libre y las casillas del horario reservadas ocupadas por una tarjeta de cita que muestra sus detalles.

Para ilustrar mejor las relaciones entre clases de distintos paquetes que participan en la solución de esta historia de usuario, en la Figura 5.62 se puede ver el diagrama *Decomposition&Uses style* asociado a esta historia de usuario.

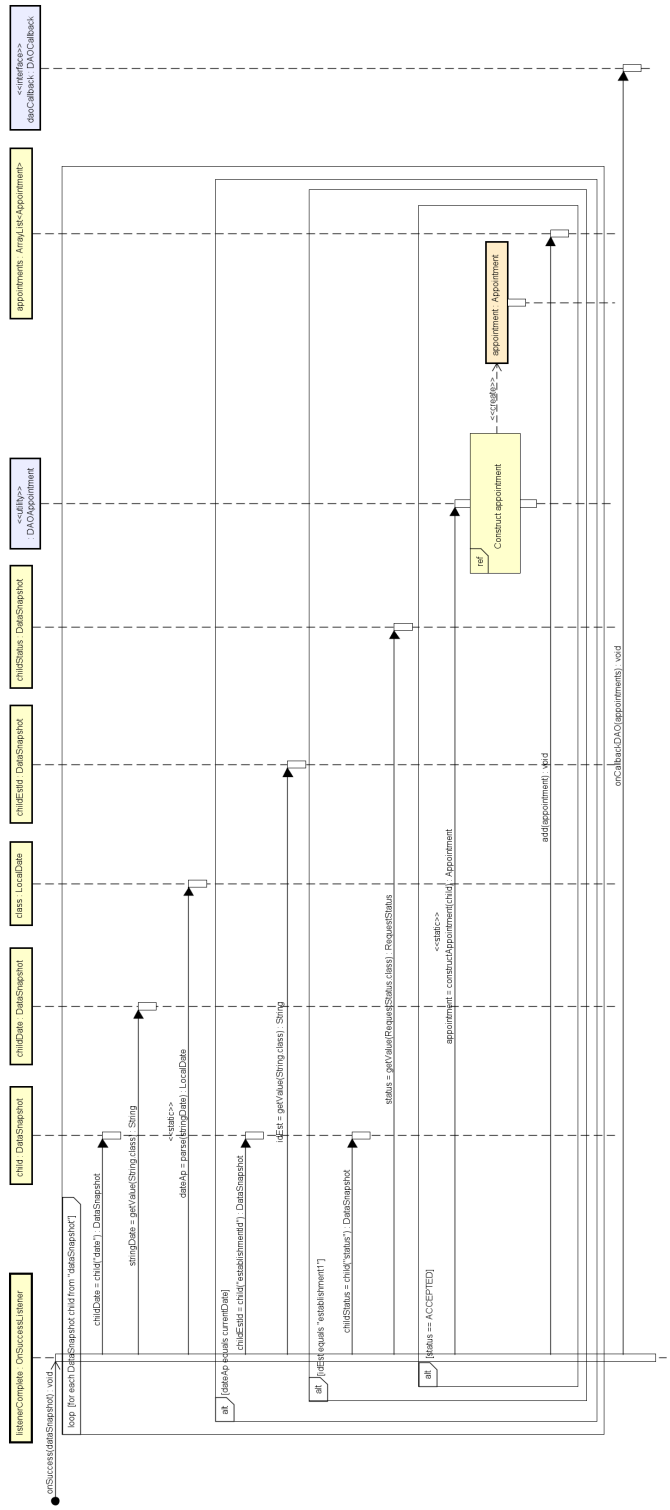

Figura 5.60. Diagrama de secuencia de "Obtener citas **onSuccess**"

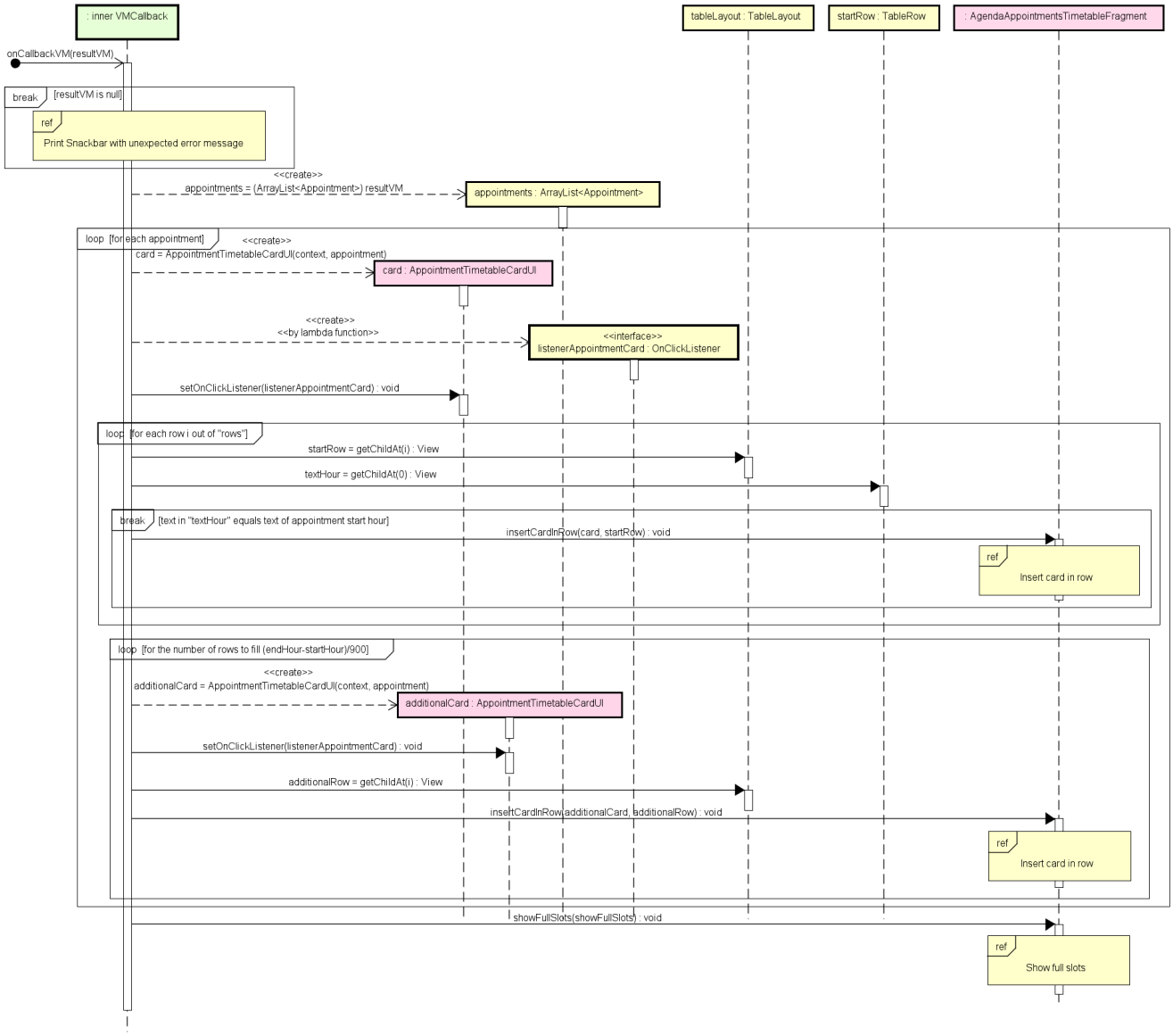

Figura 5.61. Diagrama de secuencia de "Obtener citas VMCallback"

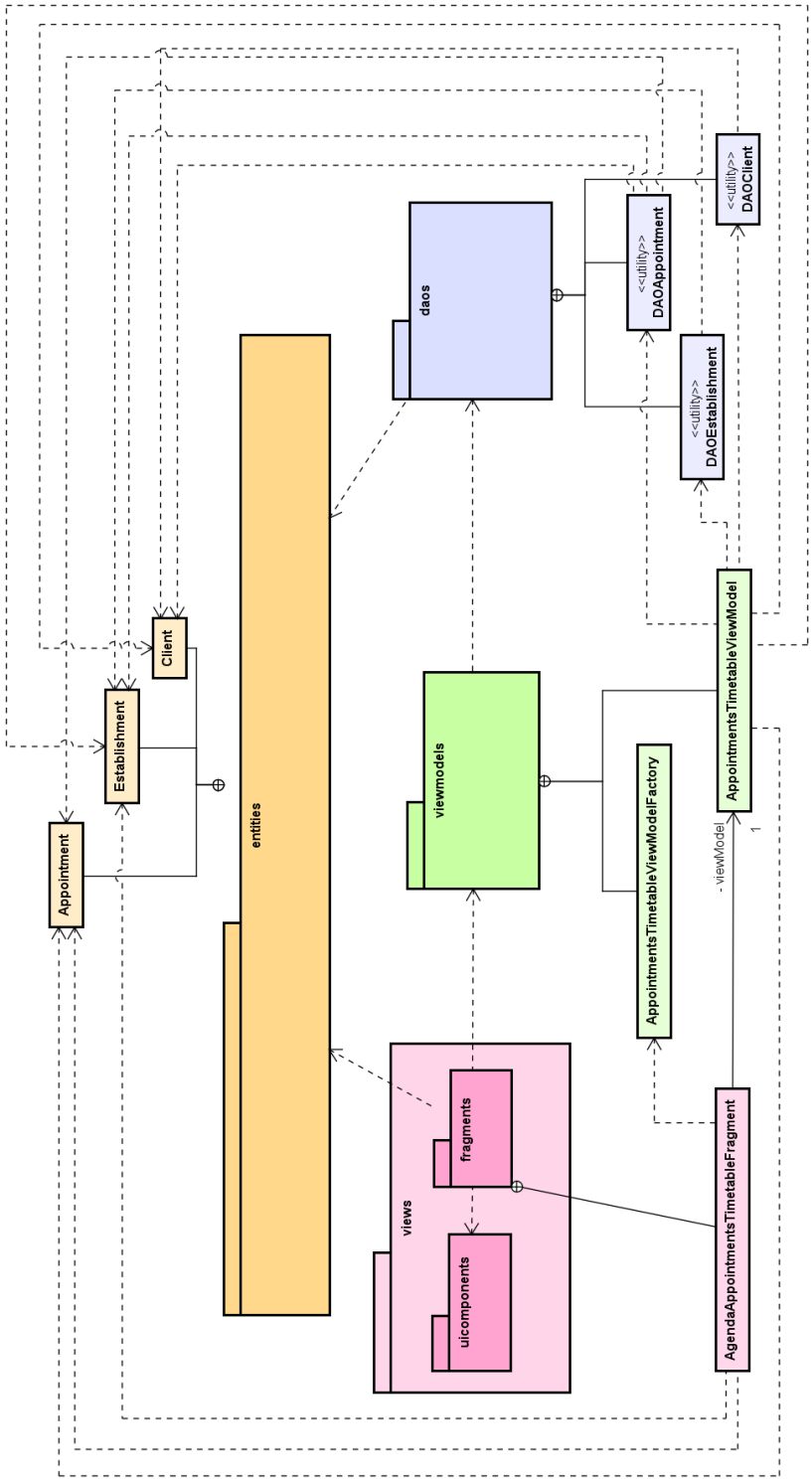

Figura 5.62. *Decomposition&Uses style* de la **HU02**

5.7. Diseño detallado

Los diagramas de clases con el diseño detallado de las mismas y algún diagrama de herencia y realización complementarios se encuentran en el archivo *Astah* en el repositorio del proyecto, cuyo enlace se encuentra en el Apéndice B. No se mostrarán en el presente documento por brevedad.

5.8. Diseño del despliegue

El despliegue de la app en este caso es muy sencillo. Como no se tienen dependencias con servicios de backend más que con los servicios proporcionados por *Firebase*, simplemente se deben representar esos componentes.

En la Figura 5.63 se puede observar el diagrama de despliegue de la aplicación.

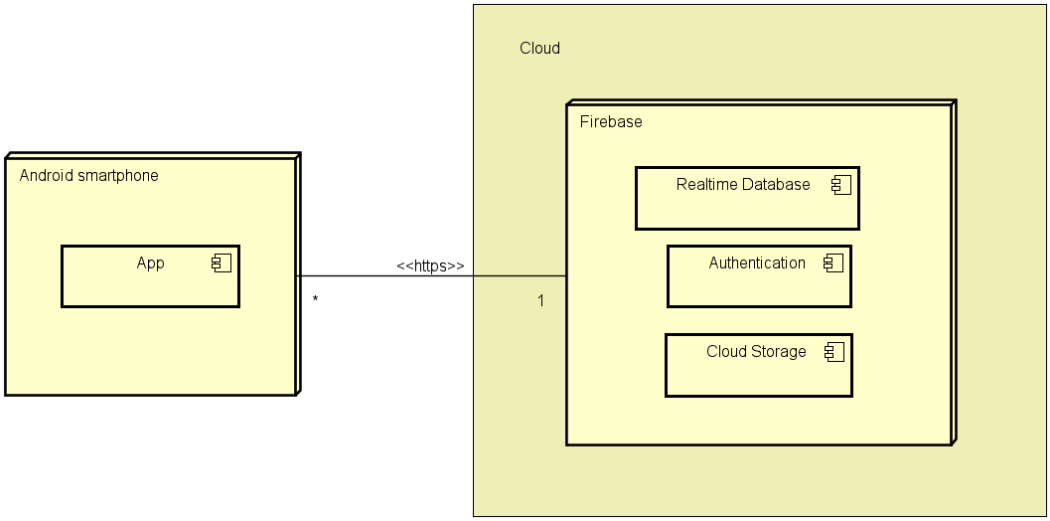

Figura 5.63. Diagrama de despliegue

Capítulo 6

Implementación y pruebas

6.1. Guía de estilo personal

Puesto que se prevé que en la aplicación vaya a haber una gran cantidad de componentes de interfaz de usuario, clases e incluso fragmentos similares, para facilitar la comprensión del código y la tarea de programación se elaboró al comienzo del proyecto una guía de estilo personal que se iría completando a lo largo del mismo. Esta guía establece la convención de los nombres que se han dado a los distintos elementos de la aplicación.

Se recuerda que debido a una de las decisiones de diseño tomadas (Sección 5.1), la totalidad del código se encuentra en **inglés**, por lo que la guía de estilo se ha adaptado a esta decisión también.

En la guía aparecen reglas para los elementos que aparecen más frecuentemente en la aplicación. Por supuesto, a lo largo de la aplicación aparecerán más tipos de elementos, pero al tener menos apariciones, no se ha considerado necesario crear reglas de estilo para los mismos. En cualquier caso, las reglas de nomenclatura de esos elementos serán similares a las aquí expuestas.

XML

En Android, una de las partes claramente diferenciadas es la parte que utiliza XML como lenguaje de representación. Los **recursos** se representan principalmente en XML. Esto incluye: imágenes que aparecen en la app, cadenas de texto, colores, layout de actividades, fragmentos y componentes de IU, entre otros. Para distinguirlos del código y sus elementos, se ha decidido darles el formato `snake_case` [122]. La nomenclatura para XML incluida en la guía de estilo ha sido:

Para los **nombres de los archivos** de recursos:

- `icon_(descripción):` iconos que se utilicen en la app. En la descripción puede aparecer la sección a la que representan o su significado. Por ejemplo: `icon_board` para el icono de la sección de "Tablón" de la barra de navegación inferior, o `icon_add` para el icono "+"que representa la acción de añadir.
- `activity_(nombre):` layout de una actividad, que conservará el mismo nombre que su respectivo archivo de código. Por ejemplo: `activity_main` para la clase `MainActivity`.
- `fragment_(descripción):` layout de un fragmento, que conservará el mismo nombre que su respectivo archivo de código. Por ejemplo: `fragment_board_new_announcement` para la clase `BoardNewAnnouncementFragment`.
- `item_recycler_(descripción):` layout de uno de los elementos de un *RecyclerView*, que posteriormente se inflará en su correspondiente ViewHolder (Sección 5.3.3). Por ejemplo: `item_recycler_request` para uno de los elementos de una lista dinámica de solicitudes.

En algunas ocasiones, si se ha utilizado uno de los archivos para más de un caso (un mismo XML de layout de fragmento para varias clases de fragmento distintas), se le ha dado excepcionalmente un nombre más general que pueda abarcar todos los sitios donde se utiliza, siempre manteniendo un nombre descriptivo. Por ejemplo: el layout `fragment_new_order` se utiliza tanto para la clase `AgendaNewOrderFragment` como para `PlacesRequestOrderFragment`.

Para los **ID de los elementos** de los archivos de recursos:

- `label_(ID_elemento_donde_aparece):` cadena de texto que aparece en otro componente de IU. Por ejemplo: en el botón `button_login` aparece una cadena de texto cuyo ID es `label_button_login`.
- `snackbar_(descripción):` cadena de texto que aparecerá mostrada en un Snackbar. Por ejemplo: para mostrar un error en el registro, se muestra en un Snackbar un mensaje con ID `snackbar_error_sign_in`.
- `(descripción)_(color):` color que se utiliza con cierto propósito en la app. Por ejemplo: para mostrar franjas horarias llenas de citas, se utilizará el color `no_space_red`.
- `item_(nombre_sección):` item contenido en un menú. Por ejemplo: `item_map` para el item de la barra de navegación inferior que representa la sección "Mapa".
- `action_(ID_salida)_to_(ID_destino):` en el grafo de navegación, acción que permite navegar desde el fragmento `ID_salida` hacia el fragmento `ID_destino`. Por ejemplo: la acción que permite navegar desde el fragmento principal de la sección "Agenda" hacia el fragmento de calendario es `action_item_agenda_to_fragment_agenda_calendar`.
- `(tipo_componente)_(descripción)_(ID_fragmento_sin_'fragment'):` para el resto de componentes de IU de Android, su nomenclatura se basa en el nombre del tipo del componente, una descripción y el fragmento al que pertenece, para evitar confusiones entre IDs de elementos de distintos fragmentos muy similares. Por ejemplo: para el componente de texto editable del email en la pantalla de login, el ID es `edit_text_email_login`.

Java

La otra parte diferenciada en Android es la parte del código. En este caso, se ha utilizado Java como lenguaje de programación, e incluye todas las clases y sus métodos y variables. En este caso, para hacer una distinción con el XML, se ha decidido dar a los nombres el formato `CamelCase` [119]. La nomenclatura para el código Java de la guía de estilo es:

Para el **nombre de las clases**:

- `(Descripción)Activity:` actividad de la app. Por ejemplo: para la actividad principal, el nombre es `MainActivity`.
- `(Sección)(Descripción)Fragment:` fragmento de la app. Para especificar con el propio nombre de que fragmento se trata, se incluirá la "ruta" o sección en la que se encuentra el fragmento. Por ejemplo: para el fragmento que muestra un horario con los pedidos en la sección "Agenda", el nombre es `AgendaOrdersTimetableFragment`.
- `(Descripción)UI:` componente Android creado programáticamente en lugar de con XML. Por ejemplo: el nombre de la clase de las tarjetas que representarán citas en el horario es `AppointmentTimetableCardUI`.
- `AdapterRecycler(Descripción):` adaptadores para los *RecyclerView* (Sección 5.3.3). Por ejemplo: el adaptador del *RecyclerView* para la lista dinámica de anuncios del tablón es `AdapterRecyclerAnnouncements`.
- `(NombreFragmentoSinFragment)ViewModel:` ViewModel de uno de los fragmentos de la app. Por ejemplo: el ViewModel del fragmento `BoardFragment` es `BoardViewModel`.
- `(NombreViewModel)Factory:` factoría del ViewModel al que hace referencia, cuando es necesario inicializarlo con parámetros iniciales. Por ejemplo: la clase factoría de `NewAppointmentViewModel` es `NewAppointmentViewModelFactory`.
- `DAO(NombreEntidad):` DAO que contiene métodos para operar con datos de la base de datos de la entidad a la que hace referencia. Por ejemplo: el DAO que opera con datos de clientes es `DAOClient`.
- `(NombreClase)Test:` clase de tests que testea la clase referenciada en el nombre. Sirve tanto para tests unitarios como para tests de IU. Por ejemplo: para los tests unitarios de la clase `PlacesViewModel`, se crea la clase `PlacesViewModelTest`, y para los tests de IU del fragmento `AgendaNewOrderFragment`, se crea `AgendaNewOrderFragmentTest`.

Para el **nombre de variables y métodos**:

- `idDelComponenteXml:` variables que referencian componentes de IU presentes en los XML. Como el XML ya proporciona un nombre explicativo, se mantiene el mismo nombre, de manera que se hace la correspondencia fácilmente. Por ejemplo: para el botón con ID `button_add_board_new_announcement`, el nombre de la variable será `buttonAddBoardNewAnnouncement`.

- `test(NombreMétodo)Ok:` test unitario que prueba un caso correcto del método referenciado. Por ejemplo: el método `testGetStringOrderHourOk`
- `test(NombreMétodo)(RazónDeError):` test unitario que prueba un caso incorrecto del método referenciado. Por ejemplo: el método `testCreateNewOrderClientNameIsNull`, que espera que el método lance una excepción debido a que el nombre del cliente que se pasa por parámetro es `null`.
- `test(Descripción):` test de IU que prueba el caso explicado en el nombre del test. Por ejemplo: el método `testAddNewAppointmentWithSpace` testea la acción de añadir una nueva cita a la agenda cuando hay espacio.

6.2. CI/CD

Para la **integración continua** de la aplicación, a lo largo de todo el proyecto se ha desarrollado un *pipeline* en *GitLab CI/CD* con distintos trabajos que se ejecutan en cada *push* de código al repositorio.

La intención inicial era añadir tres trabajos al *pipeline*: compilación, tests unitarios y tests de IU, divididos en dos etapas o *stages*: compilación, a la que pertenecería el primer trabajo, y tests, a la que pertenecerían los dos últimos. Finalmente, sólo fue posible incluir los dos primeros trabajos, puesto que a lo largo del proyecto se han presentado múltiples dificultades y limitaciones para el trabajo de tests de IU, las cuales se expondrán en la Sección 6.2.2.

El contenido del fichero `gitlab-ci.yml`, encargado de la ejecución del *pipeline* de integración continua, tras varias optimizaciones, se presenta a continuación:

```
image: androidsdk/android-30:latest

before_script:
  # Not necessary, but just for surety
  - chmod +x ./gradlew

stages:
  - build
  - test

# Make Project
assembleDebug:
  interruptible: true
  stage: build
  script:
    - ./gradlew assembleDebug
  artifacts:
    paths:
      - app/build/outputs/
```

```
# Run all unit tests, if any fails, interrupt the pipeline (fail it)
runUnitTests:
  interruptible: true
  stage: test
  script:
    - ./gradlew app:testDebug
  artifacts:
    paths:
      - app/tests/unit/
```

6.2.1. Explicación del *pipeline*

Abordando línea a línea el contenido del fichero, se puede realizar una descripción de las operaciones realizadas.

Preparación

```
image: androidsdk/android-30:latest
...
```

En esta línea, se define la imagen que se va a utilizar en el contenedor Docker creado en el runner. En este caso, puesto que se trata de un proyecto Android, la imagen elegida es la última versión de `androidsdk/android-30` [33]. Esta imagen cuenta con herramientas pre-instaladas para poder ejecutar trabajos de Android en su versión SDK 30 (con la que se compila el proyecto), incluyendo la compilación y los tests, por lo que es muy adecuada para este propósito. Además, dado que las herramientas y comandos básicos vienen ya instalados en la imagen, se puede prescindir totalmente de una fase `before_script`.

```
...
before_script:
  # Not necessary, but just for surity
  - chmod +x ./gradlew
...
```

Aunque como se ha mencionado, esta fase de preparación es totalmente prescindible, para evitar errores en la ejecución que a veces no indican su causa, se ha decidido otorgar permisos de ejecución al fichero `./gradlew`, que es el encargado de ejecutar todos los comandos relacionados con trabajos de Android. *Gradle* es el sistema de automatización en la compilación de código Android, y por tanto también el responsable de este tipo de operaciones.

```
...
stages:
  - build
  - test
...
```

El bloque `stages` indica las distintas etapas por las que está compuesto un *pipeline*. En este caso, como ya se ha explicado, el proceso está compuesto por dos etapas: `build`, donde se ejecutará el trabajo de compilación, y `test`, donde se ejecutarán los trabajos de tests.

Compilación

```
...
# Make Project
assembleDebug:
  interruptible: true
  stage: build
  script:
    - ./gradlew assembleDebug
  artifacts:
    paths:
      - app/build/outputs/
...
```

El primer trabajo que se ejecutará en el *pipeline* se denomina `assembleDebug`. El objetivo del mismo es compilar la app Android en versión *debug* o de depuración, de manera que se genere un archivo .apk para instalar la app en un dispositivo móvil compatible. La línea `interruptible: true` indica que si ocurre algún error durante la ejecución de este trabajo, el *pipeline* se dará por fallido y finalizará. La siguiente línea nos indica que este trabajo se ejecutará en la etapa `build`, la primera de las dos definidas.

El bloque `script` constituye el bloque realmente interesante, pues aquí es donde se ve realmente qué operaciones se van a ejecutar. Cada una de las líneas de este bloque son los comandos que se deberán ejecutar sobre el contenedor Docker que aloja este *pipeline*. En este caso, para compilar la app solamente es necesario un comando, `./gradlew assembleDebug`. *Gradle* se encargará por debajo de todo lo demás. Finalmente, el bloque `artifacts` indica la ruta (`paths`) donde se deberán guardar los artefactos generados por el trabajo y que se podrán descargar más tarde desde el repositorio de *GitLab*, en este caso el apk de la app en versión de depuración.

Se ha decidido no restringir este trabajo a unas ramas específicas (como podrían ser `master` o `develop`) mediante directivas `only` o `except`, puesto que debido al método de planificación del proyecto explicado en el Capítulo 2, las nuevas características se desarrollan en ramas nuevas (Sección 3.1.1), y puede ser interesante conseguir un fichero .apk de la app con la nueva característica antes de fusionar la rama con `develop`.

Tests unitarios

```
...
# Run all unit tests, if any fails, interrupt the pipeline (fail it)
runUnitTests:
  interruptible: true
  stage: test
  script:
    - ./gradlew app:testDebug
  artifacts:
    paths:
      - app/tests/unit/
```

El segundo y último trabajo presente en el *pipeline* se trata de `runUnitTests`. El objetivo del mismo es ejecutar todos los tests unitarios del proyecto, de manera que se comprueba automáticamente si con el nuevo código añadido en el *push* realizado, todos los tests siguen pasando correctamente. Los elementos de este trabajo son los mismos que en el anterior, de manera que con la línea `interruptible: true` se indica que el *pipeline* detendrá su ejecución y fallará si la ejecución del trabajo falla en algún punto. El trabajo pertenece a la etapa `test`, como se indica en la siguiente línea.

El bloque `script` sólo contiene un comando nuevamente, `./gradlew app:testDebug`, pues gracias a *Gradle* las dependencias del comando son gestionadas internamente. Los artefactos generados en el trabajo se almacenarán en la ruta indicada en el bloque `artifacts`. En este caso, si la ejecución es correcta, el trabajo no genera ningún artefacto, pero si falla en algún punto, se generará un reporte con las causas del fallo o los tests fallidos.

Este trabajo tampoco se ha querido restringir a sólo algunas ramas en concreto, pues se considera que es importante conocer si los tests siguen pasando correctamente en cualquier rama donde se haga un *push*. Además, el tiempo de ejecución de este trabajo es relativamente bajo (alrededor de 2 minutos), y al ser el último no entorpecerá al resto de trabajos.

6.2.2. Limitaciones con tests de IU

Para comenzar esta sección, quiero expresar mis agradecimientos a Fernando Javier Rodríguez Aparicio, técnico de la Escuela de Ingeniería Informática de la Universidad de Valladolid, y a Samuel Alfageme Sainz, antiguo alumno de la Escuela y actualmente ingeniero de software en el CERN (Organización Europea para la Investigación Nuclear), con gran conocimiento sobre *GitLab* y sus herramientas, entre ellas *GitLab CI/CD*. Durante todo el proceso de investigación del trabajo de tests de IU me han ayudado proporcionándome fuentes de información útiles y consejos y en el caso de Javier, configurando la máquina virtual que actúa de runner para *GitLab CI/CD* en mi proyecto.

Como se ha mencionado, la intención inicial era introducir en el *pipeline* un trabajo de tests de IU en la etapa `test` para complementar a los tests unitarios ya presentes. Sin embargo, a lo largo del proyecto han surgido diversas complicaciones y limitaciones al respecto.

Aunque no se incluye en el *pipeline* presente en el repositorio del proyecto, el aspecto final del trabajo de tests de IU que se ha conseguido es el siguiente:

```
# Run all UI tests, if any fails, interrupt the pipeline (fail it)
runUITests:
  interruptible: true
  stage: test
  script:
    - echo no | avdmanager create avd -n VirtualDevice -k "system-images;android-
    30;google_apis;x86_64" --force
    - emulator -avd VirtualDevice -no-window -no-audio -debug-init &
    - adb wait-for-device
    - adb devices
    - ./gradlew connectedAndroidTest
  artifacts:
    paths:
      - app/tests/ui
```

El trabajo **no es funcional en el runner del proyecto de *GitLab*** que contiene el código de la app. En el bloque `script` del trabajo, lo que se hace es crear un emulador mediante un AVD (*Android Virtual Device*), arrancarlo e intentar ejecutar los tests de IU en el mismo.

El problema raíz proviene de la documentación inexistente en Internet al respecto, junto con la falta de conocimiento sobre el tema. A lo largo de todo el proyecto se visitaron numerosas páginas web con explicaciones y ejemplos acerca de tests de IU de Android en integración continua, ninguna de las cuales ha servido para poder poner en marcha este trabajo. Tras intentarlo haciendo avances sin prácticamente ayuda de documentación útil, las principales dificultades que han ido surgiendo y su estado son:

- **Falta de espacio en el dispositivo - RESUELTO.** En los primeros intentos de ejecución del trabajo, puesto que consume muchos recursos ya que hay que crear un emulador completo con su propia imagen de sistema, los runners compartidos de *GitLab* se quedaron sin espacio en el dispositivo, denegando también la ejecución de otros trabajos que previamente sí funcionaban. Se avisó a Javier sobre esto, y duplicó el espacio de 10 GB que poseen normalmente estos runners para permitir el espacio de más trabajos, con lo que se solucionó.
- **Aceleración de hardware no permitida porque la virtualización está desactivada - RESUELTO.** Para ejecutar los tests de IU, es necesaria tener activada la aceleración de hardware en la BIOS del sistema. Se contactó de nuevo con Javier, y tras informar de que los runners están alojados en máquinas virtuales, aplicó la virtualización anidada para salvar este error.
- **Aceleración de hardware no permitida porque no está cargado el módulo KVM - RESUELTO.** Tras activar el soporte anidado de virtualización en la máquina virtual del runner, aún no permitía la aceleración de hardware, en este caso por falta

del módulo KVM del kernel del sistema. Javier comprobó que efectivamente el módulo está cargado en la máquina virtual, y en teoría los contenedores Docker creados poseen las propiedades de la máquina en la que se alojan por defecto. Tras realizar algunas pruebas, se descubrió que era problema de los permisos del dispositivo de aceleración KVM en la máquina virtual. Se otorgaron los permisos de ejecución necesarios y se continuó haciendo pruebas.

- **Falta de espacio en el dispositivo de nuevo - RESUELTO.** Al tratarse de runners compartidos con todos los proyectos de *GitLab* de la instancia de la Escuela, el hecho de realizar pruebas volvió a sobrecargar el sistema y a quedarse de nuevo sin espacio. Por tanto, se optó por solicitar a Javier una máquina virtual personal para establecerla como runner privado en el proyecto, de manera que también fuera más sencillo hacer pruebas desde la máquina directamente. Se creó una nueva máquina virtual con 50 GB para evitar nuevos problemas de falta de espacio y se estableció como runner dedicado del proyecto, deshabilitando los runners compartidos.

- **Aceleración de hardware no permitida por no arrancar en modo privilegiado - RESUELTO.** Cuando se consiguió realizar pruebas de nuevo, se comprobó que a pesar de haber otorgado permisos al módulo KVM, la aceleración hardware seguía sin funcionar. Samuel me informó sobre el arranque en modo privilegiado de los contenedores Docker para la ejecución del *pipeline*. Tras realizar algunos cambios en el fichero de configuración de Docker en la máquina virtual del proyecto, el contenedor arrancaba ahora en modo privilegiado y la aceleración por fin funcionaba.

- **Comandos del script no encontrados - RESUELTO.** A pesar de que la aceleración ya se ponía en marcha, llegado a cierto punto del script del trabajo, algunos de los comandos (como `adb`) no se encontraban en la variable `PATH` del sistema. La solución que se tomó fue cambiar la imagen del contenedor Docker que se estaba utilizando de `openjdk:8-jdk` a `androidsdk/android-30`. Esto fue una gran decisión, puesto que a partir de este punto se pudo prescindir del bloque `before_script`, foco de problemas, y al tener dicha imagen algunas herramientas Android pre-instaladas, los comandos se encontraban sin problema.

- ***Daemon* de *Gradle* desaparece de forma inesperada - RESUELTO.** Cuando por fin se pudo empezar a poner en marcha el trabajo, en medio de la ejecución del último comando, el que realmente ejecuta los tests de IU (`./gradlew connectedAndroidTest`), surgía una excepción en tiempo de ejecución que informaba sobre que el proceso o *daemon* de *Gradle* desaparecía de forma inesperada. Tras investigar sobre esto, se dedujo que podía ser problema de la RAM de la máquina virtual. Informando a Javier sobre esto último, subió la RAM de 2 GB a 6 GB, solventando el problema.

- `InstallException: Unknown failure` **- NO RESUELTO.** Al ejecutar finalmente el comando de tests de IU con aceleración de hardware y espacio de sobra tanto en el dispositivo como en RAM, la ejecución se detenía tras unos minutos con la excepción mencionada. Tras esto, Samuel intentó imitar la secuencia de comandos utilizada para un trabajo de tests de IU de Android del repositorio *GitLab* de *F-Droid* [46], muy complejo aunque visiblemente funcional en sus ejecuciones, pero sin éxito de nuevo. Por la limitación de tiempo del TFG, se decidió no llevarlo más allá y concluirlo aquí, no incluyendo este trabajo en el *pipeline*.

Como conclusión, y haciendo una valoración técnica, se puede decir que quizás no merece la pena incluir tests de IU en el *pipeline* de *GitLab CI/CD* debido a todos los problemas y pérdidas de tiempo que esto supone, así como la gran cantidad de recursos que consume sólo la preparación del trabajo, y el aumento de tiempo que supondría para la ejecución completa del *pipeline*, pues sólo este trabajo tiene una duración aproximada de 30 minutos, dependiendo de las características del runner. La manera de proceder entonces será **ejecutar los tests de IU de manera local**.

6.3. Pruebas

Junto a la implementación de la aplicación, se han incluido pruebas para testear el código realizado y el funcionamiento de la app a más alto nivel. Se han realizado tests automatizados, como son los **tests unitarios** y los **tests de IU**, y tests con usuarios, como son los **tests de usabilidad**.

6.3.1. Tests unitarios

El primer tipo de tests automatizados realizados se trata de **tests unitarios**. Los tests unitarios [116] verifican el funcionamiento de cierto componente de la aplicación en aislamiento. En este caso, con tests unitarios, nos referimos a tests que se pueden ejecutar en la JVM (*Java Virtual Machine*), sin necesidad de tener que ejecutarlos en un dispositivo Android. El framework utilizado para crear los tests (y el que se utiliza en Java por defecto) ha sido **JUnit 4**.

En este proyecto, los tests unitarios realizados se han empleado para probar todas las clases ViewModel, por eso se encuentran en el paquete `es.appgenda.viewmodels` del módulo `test`. Se ha realizado un total de **173 tests unitarios**.

No se han realizado tests unitarios de las clases del subpaquete `entities` puesto que al tratarse de clases *POJO*, sólo cuentan con *getters* y *setters*. Por otro lado, la intención inicial era realizar tests unitarios de las clases del subpaquete `daos`, pero surgió un problema al respecto. Al intentar realizar tests en aislamiento de las mismas, independientes del uso de la API de *Firebase* (puesto que los DAO son el punto de acceso a la base de datos), se utilizó *Mockito*, el framework recomendado oficialmente por Android para realizar *mocks* u objetos simulados, con el fin de simular las entidades relacionadas con *Firebase* y suplantar el resultado que sus métodos devuelven. El problema surge debido a la técnica de manejo de la asincronía utilizada en la aplicación: los callbacks, como se explica en la Sección 5.6. Dado que los métodos asíncronos son de tipo `void`, y el resultado se devuelve mediante la llamada al método del callback que reciben por parámetro en lugar de con un `return` como tradicionalmente, esto hace que haya que implementar el método de la interfaz callback *in-situ*, mediante una clase interna. Para poder simular clases internas, hay que utilizar la característica `mockito-inline` del framework, pero una vez que se intenta ejecutar el test, se produce un error y se informa de que **la característica `mockito-inline` no está disponible para Android**, por lo que resulta imposible realizar pruebas de este tipo.

Por esa misma razón, los métodos de los ViewModel que tengan dependencias con métodos de los DAO no se han podido testear, limitándose a realizar tests para el resto de métodos en sus casos válidos y no válidos, y también tests para los casos no válidos de los métodos con dependencias de los DAO, puesto que en estos casos no se llega a ejecutar la invocación al método del DAO.

En cuanto a la **cobertura** de estos tests, debido a la condición especial explicada, se ha decidido que **no** sería realmente **representativa** en este caso, por lo que no se ha medido. Los casos no válidos de los métodos de los ViewModel (y los válidos en los métodos que no invocan métodos de los DAO) tienen una cobertura del 100 %, pero la cobertura global es más baja por los casos que no es posible testear, y además en cada ViewModel esta cobertura será distinta, en función del número de métodos no testeables y su longitud en líneas.

En la Figura 6.1 se puede observar el resultado de ejecución de los tests unitarios, indicando que se ha tardado un total de 167 ms en ejecutarlos.

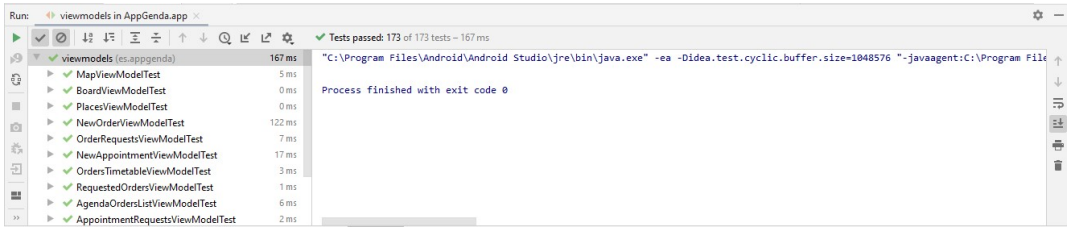

Figura 6.1. Resultado de la ejecución de los tests unitarios

6.3.2. Tests de IU

Los **tests de IU** o tests instrumentados son el segundo tipo de tests automatizados realizados. Los tests de IU [116] permiten comprobar el funcionamiento de la aplicación considerando tanto los componentes Android como el ciclo de vida de las actividades y fragmentos anfitriones. Por ello, estos tests requieren ser ejecutados en un sistema Android (emulador o dispositivo físico), de manera que se tenga acceso a sus recursos. El framework utilizado para crear los tests ha sido ***Espresso***, el oficial de Android. Este framework ofrece soporte para simular de forma automática interacciones del usuario con la app instalada en el dispositivo.

En este proyecto, los tests de IU realizados se han empleado como tests de aceptación, para verificar el aspecto inicial de los componentes IU, su comportamiento, y los casos de interacción con la app correctos e incorrectos de cada uno de los fragmentos. Por ello, los tests se localizan en el paquete `es.appgenda.views.fragments` del módulo `androidTest`. Se ha realizado un total de **75 tests de IU**.

Puesto que en estos tests se simula una interacción real con la app, se testean indirectamente los casos válidos de los métodos de los DAO y de los ViewModel que dependen de

algún DAO, complementando así a los tests unitarios previamente explicados.

Las características especiales que poseen estos tests son dos. Por un lado, puesto que como se ha explicado anteriormente no ha sido posible crear *mocks* para los métodos asíncronos (entre los que se incluyen los métodos de los DAO, responsables de manejar la información de la base de datos) y por tanto no se ha podido trabajar con datos que no pertenecieran a la base de datos, se ha decidido recurrir a las etiquetas `@BeforeClass` y `@AfterClass`. De esta manera, se ha creado un **entorno de test** controlado, haciendo que antes de cada clase de test se creen los datos convenientes que se manejarán durante la ejecución de los tests, insertándolos en la base de datos, y tras la finalización de su ejecución, se eliminen, consiguiendo así mantener los datos presentes en la base de datos y su estado intacto. Para que estos datos nuevos creados no interfieran con el contenido ya alojado en *Firebase*, la fecha que se ha utilizado siempre es el *Epoch* (01/01/1970), y en el servicio de autenticación de *Firebase* se ha registrado un usuario de ejemplo `test@mail.com`, que funcionará como cliente o dueño de negocio, según lo que se pretenda testear en cada caso.

Por otro lado, debido a la asincronía, hay ocasiones en las que es necesario esperar a que *Firebase* devuelva la respuesta solicitada (datos, iniciar sesión o recuperar archivos) para poder continuar con el test. En estos casos, la solución que se ha adoptado es una **espera activa**, que comprobará continuamente el valor de una variable boolean atómica utilizada como bandera, y no permitirá continuar con el test hasta el momento que esta variable sea `true`, momento en el *Firebase* ya ha terminado de responder a la solicitud realizada. En los casos en los que por diversos motivos no es posible detectar la finalización del trabajo de *Firebase*, se han realizado esperas activas con eventos que ocurren en la interfaz de usuario, como por ejemplo el cambio de un fragmento a otro, o la inserción de los elementos en un *RecyclerView*. Esta solución supone una mejor alternativa que utilizar una espera arbitraria con `Thread.sleep`, dado que con esta última puede suceder que se perjudique al tiempo de ejecución por una espera excesiva, o que en ocasiones no se deje finalizar la solicitud a *Firebase* si la espera es insuficiente, causando que los tests fallen, y por tanto no garantizando que será una solución efectiva en la totalidad de los casos.

En este caso, puesto que no se está testando código de forma directa, **tampoco es representativa la cobertura** de los tests.

En la Figura 6.2 se puede observar el resultado de ejecución de los tests de IU, indicando que se ha tardado un total de 1 m 56 s 16 ms en ejecutarlos.

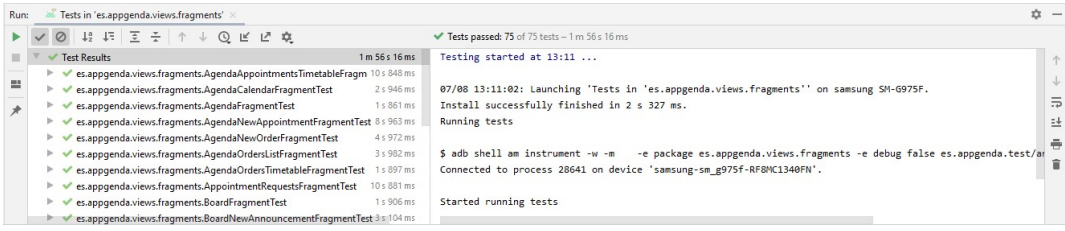

Figura 6.2. Resultado de la ejecución de los tests de IU

6.3.3. Tests de usabilidad

Para completar la fase de testing de la aplicación, y puesto que se ha hecho hincapié en la usabilidad de la interfaz de usuario a lo largo de todo el proyecto, se han realizado unos breves **tests de usabilidad** a 3 de los usuarios potenciales entrevistados al comienzo del proyecto (como dueños de negocio) y a 2 personas usuarias de un smartphone Android (como clientes). También, se comprobará si la app consigue satisfacer el requisito no funcional **HUNF02** de la Tabla 4.12.

Las pruebas realizadas han consistido en una pequeña introducción a la app (a modo de presentación, con una breve descripción y recorrido por algunas pantallas), similar a la descripción que se encontraría en *Google Play*, la plataforma donde se encontraría la app en su fase de producción, y tras esto, la realización por parte del usuario tester de 4 tareas simples con la app instalada en un smartphone Android que se le ha proporcionado. Mientras el usuario lleva a cabo la tarea, se han tomado notas sobre su comportamiento frente a la app, qué dificultades se han encontrado y otros comentarios, así como el tiempo final que ha tomado la realización de la tarea (descontando el tiempo que se tarda en teclear textos, por la variabilidad de práctica y velocidad de escritura entre los distintos usuarios tester).

Aunque es interesante conocer la opinión de los usuarios sobre la aplicación, en este caso no se ha recogido información sobre ello, puesto que el objetivo principal de los tests de usabilidad es **observar** el comportamiento del usuario y analizarlo, sacando conclusiones sobre cómo se podría mejorar la interfaz de usuario para proporcionar más usabilidad en la app y así mejorar la experiencia de usuario con la misma.

Las cuatro tareas han sido distintas según el rol que se está testeando, y dentro de los dueños de negocio, ligeramente distintas para adaptarlas a su situación (su negocio ofrece citas o pedidos) y hacerlo más realista. Las tareas elegidas no cubren la totalidad de las características de la app, pero se ha intentado centrarse en las más importantes. La información común que se ha recogido de todos los usuarios tester ha sido: **rango de edad** (0-20, 21-30, 31-40, 41-50, 51-60, 61-70, >70) y **familiarización con el uso de las TICs** (del 1 al 10).

Dueños de negocio

Las 4 tareas elegidas han sido:

- **Tarea 1.** Si el establecimiento ofrece citas, *consultar quién ha pedido cita el 14/03/2021 a las 17:00 y el motivo de la cita*, y si ofrece pedidos, *consultar quién ha hecho un pedido el 14/03/2021 a las 17:00 y el motivo del pedido.*

- **Tarea 2.** Si el establecimiento ofrece citas, *añadir una cita para el 15/07/2021 de 9:00 a 9:45 para <nombre_cliente> con el motivo <motivo_cita>*, y si ofrece pedidos, *añadir un pedido para el 15/07/2021 a las 10:00 para <nombre_cliente> con los detalles <detalles_pedido>.*

- **Tarea 3.** Si el establecimiento ofrece citas, *rechazar la solicitud de cita pendiente con la respuesta <respuesta_rechazo>*, y si ofrece pedidos, *rechazar la solicitud de pedido pendiente con la respuesta <respuesta_rechazo>*.
- **Tarea 4.** Indistintamente si ofrece citas o pedidos, la tarea ha sido *crear un nuevo anuncio en el tablón con título <título_anuncio> y descripción <descripción_anuncio>*.

El texto entre <> ha sido adaptado a cada persona participante en los tests de usabilidad.

El resumen de las pruebas se puede ver en la Tabla 6.1.

Tarea	Taller de coches	Peluquería	Confitería
-	Rango de edad: 51-60 Familiarización TICs: 5/10	Rango de edad: 41-50 Familiarización TICs: 7/10	Rango de edad: 61-70 Familiarización TICs: 6/10
Tarea 1	Completada **Tiempo:** 1:10 min	Completada **Tiempo:** 1:25 min	Completada **Tiempo:** 0:49 min
Tarea 2	No completada **Tiempo:** 2:20 min	No completada **Tiempo:** 0:59 min	Completada **Tiempo:** 0:38 min
Tarea 3	Completada **Tiempo:** 1:47 min	Completada **Tiempo:** 1:40 min	Completada **Tiempo:** 2:55 min
Tarea 4	Completada **Tiempo:** 0:58 min	Completada **Tiempo:** 0:27 min	Completada **Tiempo:** 0:59 min

Tabla 6.1. Resultados del test de usabilidad en dueños de negocio

Las posibles mejoras en la interfaz de usuario obtenidas a partir de las observaciones en los tests han sido:

- En la segunda tarea, los dueños de negocio de establecimientos con citas han tenido problemas a la hora de añadir una cita que ocupase más de una franja horaria de 15 minutos, pues no comprendían cómo podían hacerlo. En la app, aparte de aparecer la hora en otro color, indicando que ésta es modificable, se explica con un texto en la parte superior de la pantalla que pulsando sobre la hora, se puede cambiar. Estos resultados han mostrado claramente que la tendencia es no leer las instrucciones por pantalla, aunque sean mínimas. Por ello, una posible solución podría ser **colocar un texto bajo los parámetros modificables** de la cita que indique que tocándolo, se pueden modificar. También se podría buscar **otra manera de representar los parámetros modificables**, por ejemplo con algún *widget* interactivo como un selector de fecha y hora, que sugieran desde un principio que son modificables. Asimismo, se ha observado que las franjas de 15 minutos han resultado confusas a veces, puesto que al mostrar en el horario sólo la hora en la que se inician esas franjas, no se tenía muy claro cuántas franjas debe rellenar una cita de 45 minutos o en qué franja debe empezar esa cita. Para aclarar más esto, se puede **indicar la hora de inicio y fin de cada franja** en lugar de sólo el inicio, a pesar de que sea redundante (la hora de fin de cada franja será siempre la de comienzo de la siguiente).
- La tercera tarea, se ha completado en todos los casos, pero siempre con un exceso de tiempo. Esto es debido a que la sección objetivo de esta tarea se halla en el menú lateral, oculto a no ser que se pulse el botón superior izquierdo desde la pantalla principal

o se haga el gesto de arrastrar desde la parte izquierda de la pantalla, lo que ha ocasionado una pérdida de tiempo hasta encontrar la sección. Una posible solución sería **reorganizar las distintas secciones de la app**, moviendo al menú lateral las que se consideren menos importantes de las que se hallan en el menú de navegación inferior ahora mismo, y moviendo al menú inferior las más importantes del menú lateral.

- También en la tercera tarea, se ha observado que no siempre es intuitivo que la respuesta de rechazo que aparece por defecto es editable. Además, editarlo resulta algo incómodo, puesto que hay que eliminar varias líneas de texto. La mejora para solventar este problema consistiría en **mostrar una pista en el cuadro de texto** que muestre el mensaje por defecto, y que si se acepta el rechazo de la solicitud sin haber escrito nada como respuesta, se envíe esa respuesta por defecto.

Clientes

Las 4 tareas elegidas han sido:

- **Tarea 1.** *Hacer un pedido al establecimiento <nombre_establecimiento> para el 10/07/2021 a las 14:00 con los detalles <detalles_pedido>.*
- **Tarea 2.** *Pedir una cita al establecimiento <nombre_establecimiento> de duración 30 minutos lo más pronto posible el 14/03/2021, con motivo <motivo_cita>.*
- **Tarea 3.** *Consultar si tengo alguna solicitud de pedido rechazada y el mensaje de rechazo del establecimiento.*
- **Tarea 1.** *Confirmar la asistencia de la cita solicitada al <nombre_establecimiento> que ya me ha aceptado.*

Al igual que con los dueños de negocio, el texto entre <> ha sido adaptado a cada persona participante en los tests de usabilidad.

En la Tabla 6.2 se puede ver un resumen de las pruebas realizadas.

Tarea	Cliente 1	Cliente 2
-	Rango de edad: 51-60 Familiarización TICs: 6/10	Rango de edad: 21-30 Familiarización TICs: 9/10
Tarea 1	Completada **Tiempo:** 1:08 min	Completada **Tiempo:** 0:32 min
Tarea 2	No completada **Tiempo:** 3:50 min	Completada **Tiempo:** 0:58 min
Tarea 3	Completada **Tiempo:** 1:25 min	Completada **Tiempo:** 0:23 min
Tarea 4	Completada **Tiempo:** 1:40 min	Completada **Tiempo:** 0:37 min

Tabla 6.2. Resultados del test de usabilidad en clientes

Las posibles mejoras en la interfaz de usuario obtenidas a partir de las observaciones en los tests han sido:

- En la segunda tarea, al igual que en los tests con dueños de negocio, no ha sido intuitivo desde el primer momento en algunos casos la modificación de los parámetros de las citas o pedidos, por los que las soluciones propuestas son las mismas.
- En la tercera tarea, también de forma similar a los dueños de negocio, las secciones del menú lateral no siempre se ha encontrado fácilmente, puesto que el usuario se olvida de dicho menú. Como posibles soluciones también se proponen las mismas.
- En la cuarta tarea, se ha observado una confusión entre los términos "aceptada" y "confirmada" para el estado de las citas. Una posible solución es buscar **nueva terminología** más clara y descriptiva para los estados de una solicitud de cita, de manera que el usuario no confunda los mismos.

Una mejora válida para ambos casos, tanto dueños de negocio como clientes, consistiría en proporcionar formación a los usuarios para aplanar la curva de aprendizaje del uso de la app. Esta formación podría darse en forma de un breve **videotutorial**, en la propia página de descarga de *Google Play* junto con el resto de imágenes descriptivas de la app, o bien con un **asistente *wizard*** que vaya indicando los pasos sólo la primera vez que se realice una acción concreta desde la app, de manera que se asegura que los usuarios lo vean. También sería una buena opción un **manual de usuario** como el que se incluye en el Apéndice A.2, para consultar dudas sobre cómo realizar ciertas acciones en la app.

En cuanto al requisito no funcional **HUNF02**, las tareas no se han completado en menos de 1 minuto en el 50 % de los casos. Sin embargo, salvo excepciones, el tiempo total de las tareas no ha sobrepasado 1:45 minutos, por lo que quizás el requisito establecido es ligeramente restrictivo y se podría cambiar a la cantidad de minutos mencionada, que se sigue considerando aceptable. Otra opción, dado que no todas las tareas suponen la misma complejidad (por ejemplo, las relacionadas con el tablón de anuncios se han completado sin problemas en todos los casos), es categorizar las tareas por su complejidad en distintos niveles, y establecer un requisito diferente con distintos tiempos para cada nivel, de manera que se adapta más a cada situación concreta.

Como conclusión final, se ha de decir que el factor edad y la experiencia con las tecnologías sí afectan al uso que se da a la app, pero como el conjunto de usuarios objetivo en este caso no se restringe a personas más jóvenes o familiarizadas con las tecnologías, sería beneficioso aplicar las mejoras mencionadas y seguir testeando para encontrar cada vez un diseño más dirigido al usuario.

6.4. Principales dificultades y retos

En la Sección 7.2 se irán describiendo todos los problemas surgidos durante el desarrollo del proyecto, pero aquí se enumerarán las principales dificultades y retos que se han ido afrontando a lo largo del mismo en cuanto a implementación:

- **Tests de IU en el *pipeline* de *GitLab CI/CD*.** Esta ha sido la principal dificultad afrontada en el proyecto. Ha sido la tarea que más problemas ha dado y a su vez la más larga (desde febrero que se comenzó hasta el final del proyecto en julio, que por problemas constantes no se pudo finalizar). En la Sección 6.2.2 se describen paso a paso todos los problemas surgidos y de qué manera se ha actuado en cada caso.

- **Tests de IU como concepto.** Antes de comenzar el proyecto, no se conocían los tests instrumentados de Android ya que no se habían estudiado ni practicado a nivel académico ni a nivel profesional en las prácticas de empresa. Esta tarea por tanto supuso un esfuerzo de formación previa, que a lo largo del proyecto se fue consolidando y mejorando. Puesto que la tarea de testeo es una de las más importantes en el proceso de creación de un producto software y sobre todo en metodologías ágiles, este reto superado se considera de los más valiosos del proyecto.

- **Uso de componentes de IU Android desconocidos.** Como antes de comenzar con el TFG sólo se había trabajado con Android en una ocasión, en una asignatura del grado, y a nivel básico, no se conocían ni se tenía práctica utilizando componentes de IU más avanzados, como *RecyclerView*, o menos comunes, como los selectores de fechas y horas en una ventana de diálogo. Sin embargo, Android posee una documentación oficial online muy completa que ha facilitado mucho esta tarea. Por tanto, aparte de aprender a utilizar nuevos componentes de IU, también se ha adquirido práctica en buscar documentación de Android, y algunos sitios donde se puede encontrar información de calidad. Asimismo, se ha tenido que lidiar en varias ocasiones con componentes de IU que presentan bugs no documentados de manera oficial, sino por otros usuarios que han experimentado el mismo error, lo cual también ha incrementado la complejidad de realizar algunas tareas por la investigación que ha conllevado.

- **Trabajar con asincronía.** Otro de los grandes retos presentes ha sido manejar la asincronía que los servicios de *Firebase* introducían a la aplicación. Esto ha supuesto numerosos quebraderos de cabeza, ya que aunque se había trabajado con *Firebase* en Android anteriormente, fue a nivel muy básico. Para conseguir un código lo más limpio posible en este caso, se ha hecho un esfuerzo extra buscando y probando distintas soluciones, que ha repercutido también en otras tareas relacionadas como el diseño por no haber trabajado totalmente con sincronía, como se estaba acostumbrado.

- ***Mocks* en los tests de Android.** Relacionado con el punto anterior, la asincronía implicaba tener que utilizar nuevas técnicas de testeo, no tan básicas como las ya conocidas. Tras haber investigado y haber dado con la solución para testear los callback (técnica utilizada para manejar la asincronía en la app), se descubrió que en Android no está disponible esta característica, como se explica en la Sección 6.3.1. Sin embargo, esto ha servido también como formación personal, para cuando se presente ese problema fuera de este contexto.

6.5. Licencia

Se ha dotado al proyecto de una licencia **Creative Commons Atribución-NoComercial-SinDerivadas 4.0 Internacional (CC BY-NC-ND 4.0)** [32]. Esto significa que el material se puede compartir y distribuir mediante cualquier medio, pero se debe dar crédito debidamente e indicar si se han realizado modificaciones, junto con un enlace a la licencia. Además, no se puede hacer uso del material con propósitos comerciales, y no se podrá distribuir material derivado del presente proyecto (remezclado, transformado o creado a partir de él). El texto legal de esta licencia se encuentra en el repositorio *GitLab* del proyecto del Apéndice B.

Capítulo 7

Seguimiento del proyecto

7.1. Introducción

En este Capítulo se presentará el seguimiento del proyecto realizado sprint a sprint. Los sprints están compuestos de distintas tareas, principalmente relacionadas con las historias de usuario, aunque también hay tareas de otra índole que se han llevado a cabo en los mismos. Es por ello que para cada sprint, se presentará una tabla compuesta por las siguientes columnas: **Historia de usuario**, que indicará el código de la historia de usuario con la que se relacionan las tareas de la siguiente columna; **Tareas**, que enumerará las actividades a realizar para dicha historia de usuario; **Tiempo estimado**, que indicará el tiempo que previsiblemente se ha asignado a la historia de usuario, traducido en puntos de historia en la fase de planificación; **Tiempo empleado**, que registrará el tiempo que se ha dedicado realmente al desarrollo de la historia de usuario; y **Estado**, que indicará la situación en la que se encuentra la historia de usuario en el momento de finalización del sprint entre "no comenzado", "en progreso" y "finalizado".

Para las tareas que no están relacionadas con ninguna historia de usuario (como pueden ser tareas de documentación, refactorización, etc.), en la columna de **Historia de usuario** aparecerá un "-", en la columna **Tareas** la descripción de la tarea a realizar y en la columna **Tiempo estimado** el tiempo que se ha planificado para llevarla a cabo. El resto de columnas seguirán cumpliendo con la misma función.

La realización de cada historia de usuario supone completar cinco tareas distintas (correspondientes con las cinco tarjetas creadas en el tablero *GitLab Issue Board*, como se explica en el Capítulo 3). Estas tareas son:

- **Análisis.** En esta tarea, se comprobará si es necesario hacer algún cambio en el modelo de dominio inicial para poder desarrollar correctamente la historia de usuario en cuestión. También, se podrán desglosar épicas del *Product Backlog* en historias de usuario, que puede ser adecuado en ciertos momentos debido al conocimiento ya adquirido.

- **Diseño.** En esta tarea, se completarán los distintos diagramas existentes actualmente con los nuevos elementos y detalles que se introduzcan en el desarrollo de la historia de usuario.
- **Implementación.** En esta tarea, se escribirá el código correspondiente al diseño creado para añadir a la app la funcionalidad descrita por la historia de usuario.
- **Revisión de IU.** En esta tarea, se revisará si el diseño de interfaz de usuario desarrollado se ajusta a lo planeado previamente en los *mockups* o prototipos.
- **Testeo.** En esta tarea, se desarrollarán tests unitarios y tests de IU para las nuevas características añadidas con la historia de usuario.

7.2. Seguimiento de los sprints realizados

7.2.1. Sprint 0 (19/01/21 - 10/02/21)

Como ya se comentaba en el Capítulo 2 de planificación, este primer sprint es algo diferente respecto al resto. El objetivo es dedicar unas semanas a labores de preparación y planificación del proyecto, para en el sprint 1 poder dar comienzo al desarrollo de las historias de usuario. También, la duración del sprint es distinta a la duración del resto, habiendo dedicado al mismo finalmente un total de 3 semanas y 1 día.

Puesto que en este sprint aún no se desarrollan tareas relacionadas con las historias de usuario, en lugar de presentarlas en forma de tabla, se enumerarán y explicarán las tareas desarrolladas:

- **Elaboración del *Product Backlog* inicial.** Partiendo de la idea inicial, se creó una versión preliminar del *Product Backlog*, que se completaría más tarde mediante entrevistas a posibles usuarios de la app. Para ello, se escribieron los requisitos de usuario u objetivos de la app en forma de épicas, dedicando un tiempo a formación en elaboración de historias de usuario mediante una guía sobre cómo escribirlas correctamente y ejemplos reales [82].
- **Creación de *mockups* de la app.** Se crearon diversos *mockups* o prototipos de bajo coste que muestran el aspecto de las distintas pantallas que habrá en la app. También, en función de las sugerencias de la tutora y de las entrevistas con los usuarios potenciales, se realizaron varias iteraciones de refinamiento de los mismos. Estos diseños son los que se han presentado en el Capítulo 5.
- **Realización de entrevistas a posibles usuarios potenciales.** Para darle al proyecto un aspecto más real, se decidió hacer parte de la elicitación de requisitos con entrevistas a dueños de pequeños negocios reales y al alcalde de un ayuntamiento. El objetivo de estas entrevistas era, una vez propuesta la idea inicial y enseñados los *mockups* a modo de prototipo, recoger ideas que hicieran de la app más útil para ellos, y concluir en líneas generales si consideran el proyecto viable para utilizar la app en un futuro. Las entrevistas que se realizaron y los resultados obtenidos fueron:

- **Taller de coches.** No se han propuesto nuevas funcionalidades para el sistema. El dueño de negocio sí considera viable la idea y utilizaría la app.
- **Peluquería.** Se han propuesto nuevas características que se traducen en las siguientes historias de usuario:
  - Como dueño de negocio quiero tener la posibilidad de añadir más de un cliente por franja horaria en la gestión de citas para mejorar la eficiencia de mi trabajo
  - Como dueño de negocio quiero que las franjas horarias estén divididas en 15 minutos para poder gestionar mejor el tipo de citas que mi negocio ofrece

  La dueña de negocio sí considera viable la idea y utilizaría la app.
- **Confitería.** No se han propuesto nuevas funcionalidades para el sistema. El dueño de negocio sí considera viable la idea y utilizaría la app.
- **Ayuntamiento.** En este caso, no se trata de un negocio, pero su función de gestión de citas para servicios a la ciudadanía es análoga a la gestión de citas de un establecimiento. Se ha propuesto una nueva característica que se traduce en la siguiente historia de usuario:
  - Como dueño de negocio quiero desglosar mi establecimiento en distintos departamentos para saber en qué área están interesados en pedir cita los clientes

  El ayuntamiento sí considera viable la idea y utilizaría la app.

En el caso de la característica sugerida por el ayuntamiento, una alternativa que se le propuso se basa en crear distintas cuentas de establecimiento en la app, una por departamento, que se aceptó como solución a su necesidad. Todas las ideas propuestas fueron recogidas y añadidas a la lista de requisitos iniciales, dando como resultado el *Product Backlog* inicial, que se puede ver en la Tabla 2.13.

- **Descripción de requisitos de información y modelo de dominio inicial.** Se escribieron y refinaron los requisitos funcionales de información del sistema en forma de historias de usuario, resultando en los mostrados en la Tabla 4.11. A partir de ellos, se elaboró el modelo de dominio inicial, mostrado en la Figura 4.1.
- **Calendarización inicial.** Como parte de la planificación inicial del proyecto, se realizó un calendario con las fechas de inicio y fin de los sprints del proyecto y de los eventos importantes a lo largo del proyecto (eventos Scrum, fechas límite de entrega, etc.). Este calendario es el que se puede ver en la Tabla 2.1.
- **Plan de riesgos.** Se realizó el plan de riesgos ya presentado en la Sección 2.4.
- **Plan de presupuestos.** Para culminar con la planificación inicial, también se realizó y refinó el plan de presupuestos mostrado en la Sección 2.5.
- **Búsqueda de información inicial.** Para poder empezar el desarrollo del proyecto, se dedicó tiempo a la búsqueda de información y a la formación de algunas de las tecnologías y herramientas que se iban a utilizar, como *GitLab CI/CD* para proyectos Android [49] y tests en Android [18].
- **Arquitectura inicial.** Para tener una base sobre la que comenzar la arquitectura de la aplicación, se realizó un diagrama *Decomposition&Uses style* que muestra la arquitectura general inicial del sistema. Este diagrama se mostró en el Capítulo 5, dedicado al diseño.

- **Preparativos iniciales.** Finalmente, se llevaron a cabo otros preparativos iniciales como la creación del tablero de *GitLab Issue Board* con sus distintas columnas, y la elaboración de un listado de versiones de los programas y herramientas que se utilizarán a lo largo del proyecto y que ya se han enumerado en el Capítulo 3.

En este sprint, el total de tiempo empleado entre todas las tareas asciende a **20 horas** aproximadamente.

7.2.2. Sprint 1 (10/02/21 - 24/02/21)

Este sprint es el que da comienzo al desarrollo del proyecto y sus historias de usuario. En un principio, se decidió establecer la carga de trabajo por sprint en 40 horas, tal y como sugiere la planificación inicial (20 horas/semana), cantidad que será ajustable en próximos sprints en función de cómo se trabaje en este.

En la primera historia de usuario del *Product Backlog*, incluida en el *Sprint Backlog* de este sprint, se ha incluido una tarea extra respecto a las demás historias de usuario, consistente en formación y búsqueda inicial de información, de manera que se complemente lo necesario con la tarea de la misma índole llevada a cabo en el sprint 0.

En la Tabla 7.1 se pueden consultar las tareas llevadas a cabo en el sprint 1.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
HU01	▪ Formación y búsqueda inicial de información ▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	25 horas	30 horas 15 minutos	Formación, análisis, implementación y revisión de IU finalizados, diseño en progreso, testeo no comenzado
HU02	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	4 horas	Análisis, implementación y revisión de IU finalizados, diseño en progreso, testeo no comenzado
HU03	▪ Análisis	5 horas	0 horas	No comenzado

Tabla 7.1. Tareas del sprint 1

Finalmente, el tiempo empleado al sprint fue de **34 horas 15 minutos**, debido a dificultades para conllevarlo con la asignatura que se cursaba este cuatrimestre y sobre todo con las prácticas de empresa, comenzadas en la primera semana de este sprint.

En la tarea de formación incluida en la **HU01**, se consultó diversa documentación sobre estilos y colores de *Material Design* [75] [73] [76], que se aplicó a la app desde el principio del desarrollo, y sobre algunos componentes de Android que se introdujeron, como la navegación inferior [72] y el componente *Navigation* [17].

Las tareas de diseño de las historias de usuario no fueron finalizadas por dudas que se consultaron a la tutora. Tras consultarlo, se decidió que el diseño de las clases de actividades y fragmentos de Android se importaría desde *Astah* tras implementarlos por comodidad, ya que parte del código de los mismos es auto-generado y supondría una gran pérdida de tiempo hacerlo para todos los fragmentos que se creen a lo largo del proyecto. El testeo no se pudo comenzar en ninguno de los casos por formación insuficiente aún, por lo que se seguirá trabajando sobre ello en el próximo sprint. Debido a la falta de tiempo, la tercera historia de usuario tampoco se pudo comenzar.

Por ello, pasan al siguiente sprint las tareas de diseño y testeo de las **HU01** y **HU02**, y la tarea de análisis de la **HU03**.

7.2.3. Sprint 2 (24/02/21 - 10/03/21)

A pesar de que en el sprint anterior no se pudo lograr dedicar el tiempo estimado por falta de tiempo por otros motivos académicos, en este sprint se volvió a intentar cumplir con las 40 horas estimadas.

Como aún quedaban varias tareas pendientes del sprint anterior y la **HU03** se estimó en 5 puntos (25 horas) debido a que sería la primera en la que había que introducir técnicas como el uso de la arquitectura MVVM o herramientas como *Firebase*, que serviría de "plantilla" de aquí en adelante, se decidió no añadir nuevas historias de usuario en el *Sprint Backlog*. Además, en la implementación de las historias de usuario ya realizadas, se decidió hacer un cambio para convertir todas las pantallas a fragmentos y seguir la arquitectura *single-activity* [84] en lugar de utilizar múltiples actividades.

El desglose de tareas del sprint se puede observar en la Tabla 7.2.

De la misma manera que en el sprint anterior, no se pudieron alcanzar las horas propuestas para el sprint por falta de tiempo, dedicando un total de **34 horas 45 minutos**.

Para completar la formación sobre diversos componentes de Android, sobre todo *Navigation* y la arquitectura MVVM, se realizaron algunas partes de dos cursos oficiales de Android [5] [6].

En el caso de la **HU01**, el tiempo empleado fue mayor de lo estimado debido a problemas con el framework de pruebas en la tarea de testeo, que finalmente fueron subsanados. De la misma manera, en la **HU02**, el tiempo iba a ser menor al estimado, pero tras varios problemas con los tests de IU en *GitLab CI/CD*, no se pudo reducir. De momento, en el *pipeline* de *GitLab CI/CD* no se incluye el trabajo de ejecución de tests de IU, solamente se ejecutarán los trabajos de compilación de la app y de ejecución de tests unitarios.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
HU01	▪ Cambios en la implementación ▪ Diseño ▪ Testeo	7 horas 30 minutos	9 horas	Finalizado
HU02	▪ Cambios en la implementación ▪ Diseño ▪ Testeo	7 horas 30 minutos	7 horas 30 minutos	Finalizado
HU03	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	25 horas	18 horas 15 minutos	Análisis finalizado, implementación en progreso, diseño, revisión de IU y testeo no comenzados

Tabla 7.2. Tareas del sprint 2

Un error desconocido relacionado con el componente *Navigation* fue resuelto mediante las respuestas a una pregunta de *Stack Overflow* [108]. Para elaborar la vista del horario de citas, había dos componentes Android adecuados: *GridView* y *TableLayout*. Vista la anatomía y las características de cada uno [14] [21], se escogió *TableLayout*. También se investigó sobre cómo dibujar bordes alrededor de la tabla del horario, dando con la solución de nuevo en *Stack Overflow* [102].

El problema más grande que se encontró en este sprint era que las vistas que se añadían dinámicamente a la tabla no eran visibles. Tras horas de investigación, se dio con el problema, que consistía en el tipo de parámetros que se asignaba a las vistas a añadir [98] [99].

Finalmente, pasan al siguiente sprint las tareas de diseño, implementación, revisión de IU y testeo de la **HU03**.

7.2.4. Sprint 3 (10/03/21 - 24/03/21)

A partir de este sprint, y tras hablarlo con la tutora, se ha hecho un reajuste en las horas planificadas por sprint para evitar la sobrecarga de trabajo. Desde este momento, los sprints en principio constarán de una carga de 30 horas en lugar de 40. Puesto que existen sprints de refuerzo, se recuperarán más adelante esas horas, a partir del momento que se finalicen las prácticas en empresa.

Dado que se pretende realizar el diagrama de secuencia de las **HU03** y **HU04**, y es algo que lleva bastante tiempo, se ha decidido incluir solamente estas historias de usuario en el sprint, para completar todas sus tareas y dejar hecha la parte más trabajosa de diseño del proyecto.

En la Tabla 7.3 se puede consultar el conjunto de tareas que se han llevado a cabo en este sprint.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
HU03	▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	21 horas 45 minutos	Implementación y revisión de IU finalizadas, diseño y testeo en progreso
HU04	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	20 horas	15 minutos	Análisis finalizado, implementación y revisión de IU en progreso, diseño y testeo no comenzados

Tabla 7.3. Tareas del sprint 3

A pesar de la reducción en el número de horas estimadas, no se ha conseguido llegar al objetivo debido a la carga de trabajo impuesta por la asignatura de este cuatrimestre, que requería una práctica con fecha de entrega en medio de este sprint. Por ello, se ha incluido un nuevo riesgo al plan de riesgos que se ha materializado y no estaba presente en la lista, el **R07** (Tabla 2.8). El total de tiempo empleado en este sprint fue de **22 horas**.

Como en la **HU03** se ha introducido el uso de *Firebase* como base de datos, se ha dedicado tiempo a realizar un cuestionario sobre cuál de los dos tipos de base de datos que ofrece *Firebase* utilizar [38], también a leer documentación sobre consejos de la estructura de los datos en la base de datos, que es NoSQL [39], y sobre cómo utilizar en Android la base de datos elegida, *Realtime Database* [43].

En esta historia de usuario también se ha dedicado más tiempo del estimado debido a un bloqueo surgido durante la implementación del primer DAO. Al crear el proyecto en *Firebase*, se decidió que el servidor de la base de datos esté alojado en Europa y no en Estados Unidos como lo está por defecto, con el fin de reducir la latencia lo máximo posible (aunque sea prácticamente inapreciable). Resulta que si se aloja en otro sitio que no sea Estados Unidos, hay que especificar la URL de la base de datos a la hora de obtener la instancia en el código, y ese detalle sólo lo menciona en algunos apartados de la documentación, no en todos los que debería, por lo que esta inconsistencia en la documentación ha hecho dedicar un exceso de tiempo en investigar sobre el problema. Para terminar con la implementación de esta historia de usuario, ha sido necesario también consultar información sobre cómo dar formato a los recursos de cadenas de caracteres en Android [19].

Por último, se ha intentado añadir de nuevo el trabajo de ejecución de tests de IU a *GitLab CI/CD* durante la tarea de testeo de la **HU03** mediante un ejemplo encontrado [45], pero no se ha conseguido por problemas durante la ejecución.

7.2.5. Sprint 3 (ampliación) (24/03/21 - 06/04/21)

A pesar de que en la planificación inicial este periodo estaba reservado para vacaciones de Semana Santa, se ha decidido realizar una ampliación del sprint 3 para recuperar las horas que no se habían podido dedicar en los últimos sprints.

Al tratarse de una ampliación y no de un sprint como tal, no se ha estimado tiempo para las tareas, y tampoco se han incluido más tareas que las que quedaron pendientes al final del sprint 3. La única diferencia, es que ahora se ha añadido la tarea de diseño de la **HU02** porque resulta que el diagrama de secuencia que se estaba elaborando correspondía en realidad a esa historia de usuario y no a la **HU03**, en la que no existe un acceso a la base de datos. Por tanto, las tareas son las mismas, pero ahora correctamente asignadas a su historia de usuario correspondiente.

En la Tabla 7.4 se pueden consultar las tareas llevadas a cabo en esta ampliación del sprint 3.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
HU03	▪ Diseño ▪ Testeo	-	6 horas	Finalizado
HU04	▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	-	14 horas 45 minutos	Implementación, revisión de IU y testeo finalizados, diseño en progreso
HU02	▪ Diseño	-	45 minutos	En progreso

Tabla 7.4. Tareas de la ampliación del sprint 3

El total de tiempo dedicado a esta ampliación de sprint ha sido de **21 horas 30 minutos**.

Como parte de la tarea de testeo de la **HU03**, se siguió intentando añadir los tests instrumentados o de IU al *pipeline* de *GitLab CI/CD*, sin éxito. En este punto se decidió solicitar a los técnicos de la Escuela una máquina virtual para que funcione como runner en el proyecto de este TFG en *GitLab*, con el objetivo de tener más espacio y no entorpecer las ejecuciones de los runners compartidos con los demás proyectos, además de poder hacer pruebas en la propia máquina virtual, por lo que de momento las tareas de testeo no incluirán la ejecución de los tests de IU en integración continua.

Durante la realización de la **HU04**, la implementación tuvo que añadir un detalle respecto al *mockup* realizado para esta pantalla, dado que en un principio se planeó poder añadir una cita seleccionando varias tarjetas libres disponibles pero finalmente, por simplicidad, se hará click en una sola tarjeta, por lo que el formulario de nueva cita ahora también contendrá un campo “Hora de finalización”. Este formulario contiene selectores para elegir fecha y hora, por

lo que se consultó la documentación de Android correspondiente [20]. Dadas las restricciones de las horas a seleccionar (en franjas de 15 minutos y de 8:00 a 23:00), se intentó crear un selector personalizado, pero debido a la complejidad de crearlo, ya que es un componente que ofrece Android y no es fácilmente customizable [111] [107], de momento se ha optado por mostrar un mensaje de retroalimentación al usuario cuando la selección sea incorrecta. Para los tests de esta misma historia de usuario, se añadió un *matcher* customizado obtenido de una pregunta en *Stack Overflow* [100], y se buscó información sobre cómo comprobar que se muestra un *Snackbar* con el mensaje requerido por pantalla [110].

Las tareas que pasan al siguiente sprint son las de diseño de las **HU03** y **HU04**, de las que se planea realizar diagramas de secuencia.

7.2.6. Sprint 4 (06/04/21 - 21/04/21)

Para este sprint, en lugar de 30 horas, se ha estimado que se podría dedicar un total de 25 horas, debido a un examen que tendrá lugar en la primera semana y a la entrega de una práctica planificada para la segunda semana del mismo, de tal manera que no ocurra como en los sprints anteriores y no se pueda alcanzar el tiempo estimado.

A parte de continuar con las tareas pendientes del anterior sprint, se han introducido las dos historias de usuario siguientes del *Product Backlog*.

En la Tabla 7.5 se puede observar el desglose de tareas del sprint.

Esta vez, sí que se ha cumplido con la estimación e incluso se ha rebasado mínimamente, con **25 horas 15 minutos** empleadas en total.

En las tareas de testeo, se han podido solucionar finalmente los "problemas" en los tests de IU relacionados con la asincronía de los métodos de la base de datos, que obligaban a introducir una espera arbitraria con `Thread.sleep` hasta que se crearan las nuevas entradas para el test. Esto no es considerado una buena práctica, por lo que la solución ideada se basa en una técnica similar a una espera activa, con variables de tipo boolean de acceso atómico. Además, para poder crear un entorno de test más fácilmente, ahora los métodos "create" de los DAO devuelven el ID del objeto creado, para poder eliminarlos por ID al finalizar el test.

También en relación con los tests, tras numerosos intentos por testear los métodos de los ViewModel que usan métodos del DAO, se ha llegado a la conclusión de que no es posible, dado que en Android no es posible *mockear* o simular mediante *mocks* un método estático que a su vez utiliza un *callback*, técnica que se está utilizando en la aplicación para manejar la asincronía, como se ha explicado en el Capítulo 5. *Mockito* [81], que es el framework de *mocks* recomendado por Android, imprime un mensaje en el log informando de que la funcionalidad de *mock inline*, la necesaria para simular *callbacks*, no está soportada en Android. Durante el tiempo que se ha dedicado investigando sobre ello antes de llegar a esta conclusión, se ha leído diversa documentación y preguntas [66] [109] [97] [106]. Tras comentarlo con la tutora en la reunión *Scrum weekly*, se ha acordado no realizar tests de los métodos de los DAO, dado que indirectamente ya se testean con los tests de IU.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
HU04	▪ Diseño ▪ Cambios en la implementación ▪ Cambios en los tests	5 horas	10 horas 25 minutos	Finalizado
HU02	▪ Diseño	5 horas	9 horas 30 minutos	Finalizado
HU05	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	2 horas 50 minutos	Finalizado
HU06	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	2 horas 25 minutos	Finalizado

Tabla 7.5. Tareas del sprint 4

También, la URL de la base de datos ahora ha pasado a ser una variable del fichero de propiedades de *Gradle*, el sistema de compilación de Android, en lugar de estar en el propio código. Esto es considerado mejor práctica ya que mejora la seguridad del código. El procedimiento para ello fue consultado online [105].

En cuanto al diseño, para representar correctamente en el diagrama de secuencia el orden de las operaciones invocadas en un fragmento, se ha consultado la documentación sobre su ciclo de vida [4].

En la **HU05**, en el prototipo inicial no existía un botón para eliminar citas porque no se había valorado la opción de que el dueño de negocio pudiera borrar una cita, sino que fuese el propio cliente el que la cancelase. En su implementación se ha decidido incluir dicho botón, ya que sí tiene sentido al menos en los casos en los que el dueño de negocio registra nuevas citas, sin estar asociadas a un cliente de la app.

Tras la reunión final de este sprint con la tutora, se ha acordado no realizar más diagramas de secuencia que el de la **HU02**, dado que ha resultado ser muy complejo, entre otras cosas debido a la asincronía que implica el acceso a la base de datos de *Firebase*, y completo en el sentido de que se muestran operaciones en todos los paquetes de la aplicación. Por tanto, la explicación de este diagrama ya hecho y todos sus subdiagramas correspondientes cubrirán la necesidad de crear más diagramas de secuencia, y se dan por finalizadas las otras dos historias de usuario de este sprint de las que inicialmente se iba a realizar diagrama de secuencia. Así pues, todas las tareas de este sprint han sido finalizadas y no pasa ninguna al siguiente sprint.

7.2.7. Sprint 5 (21/04/21 - 05/05/21)

En este sprint se vuelve a la estimación inicial de 30 horas de trabajo.

Como no hay tareas pendientes del sprint anterior, se ha continuado con el desarrollo de las siguientes historias de usuario.

En la Tabla 7.6 se puede ver el conjunto de tareas llevadas a cabo en este sprint.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
HU07	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	5 horas 40 minutos	Finalizado
HU08	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	10 horas 10 minutos	Finalizado
HU09	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	4 horas 20 minutos	Finalizado
HU02	▪ Cambios en el diseño	-	1 hora	Finalizado
-	▪ Refactorización de ViewModels	-	3 horas 10 minutos	Finalizado
-	▪ Refactorización de fragmentos	-	5 horas 45 minutos	En progreso

Tabla 7.6. Tareas del sprint 5

En este sprint, el total de tiempo dedicado asciende a **30 horas 5 minutos**, cumpliendo con lo planificado. Además, como las tareas planificadas se terminaron antes de lo previsto, fue posible introducir otras tareas de corrección en el diagrama de secuencia en el caso de la **HU02** y de refactorización, que se explicarán a continuación.

En la **HU07**, se vio que el fragmento que se iba a crear era muy similar a otro utilizado en historias de usuario anteriores, por lo que en lugar de crear uno nuevo se decidió reutilizar

el ya existente. Como se prevé que esto mismo ocurra más veces según se vaya avanzando en el desarrollo de la app, se ha decidido que a partir de este momento, si se puede utilizar un fragmento ya existente, se utilice. Para mantener la consistencia, en los casos que esto ocurra, se hará un renombrado de las variables, componentes y clases implicadas para mostrar una descripción más general (adaptando los nombres a todos los fragmentos en los que se vaya a utilizar, no sólo al que se utilizaba originalmente) y mantener la consistencia de la app.

En la **HU08** se ha introducido el concepto *RecyclerView* a la aplicación, para elementos que se añaden dinámicamente a una lista y que se deben mostrar por pantalla. Para ello, se realizaron unos pequeños cursos y tutoriales [3] [128]. En la tarea de revisión de IU de esta misma historia, se decidió eliminar el campo "Hora" de cada uno de los pedidos mostrado en el *mockup*, ya que dicha hora se muestra siempre en la parte superior de la pantalla y sería redundante. En la tarea de testeo, se vio que no tenían del todo sentido algunos métodos de algunos ViewModels que definen los datos iniciales del mismo, y que sería mejor aplicar el patrón factoría para ello. Esto en los ViewModels se hace de una manera especial dado que poseen unas clases e interfaces dedicadas a ello, nativas de Android, por lo que se abrió una nueva tarea de refactorización de ViewModels, en la que se utilizó parte de un curso [2]. También, para poder hacer tests de IU con los *RecyclerView*, se tomó un *matcher* customizado obtenido de una pregunta de *Stack Overflow* [103].

Para la **HU09** se añadió un botón flotante para añadir nuevos pedidos, siguiendo las guías de *Material Design*, por lo que se consultó documentación relacionada para darle el comportamiento deseado [7] [101].

Como en la última revisión con la tutora se habló el tema de captura de excepciones de los ViewModels en los fragmentos, se ha incluido en el sprint una nueva tarea de refactorización de fragmentos para investigar sobre ello y ver cuál podría ser la mejor solución. Finalmente, y visto que capturar excepciones `IllegalArgumentException` no está visto como buena práctica, ya que realmente sirven para informar al desarrollador en el momento del desarrollo y no están pensadas para que se lancen en una aplicación en producción [112], se ha optado por no capturar las excepciones, y para evitar que la aplicación se rompa debido a una excepción en la base de datos (que son independientes del código y previamente se lanzaban como `IllegalStateException`), se han manejado los casos en los que se lanza un `OnFailureListener` desde los DAO, pero en lugar de utilizar excepciones, en función del resultado que devuelvan dichos métodos. Se ha optado por seguir la regla de que cuando se reciba `null` como resultado de alguna operación de los DAO, significa que ha ocurrido algún error en la base de datos, en cuyo caso se mostrará un mensaje de error genérico al usuario. En cualquier otro caso, el resultado es válido y la app seguirá su flujo.

También, se observó que en las escrituras en la base de datos la instancia de la base de datos se queda esperando indefinidamente hasta volver a tener conexión a Internet, al contrario que en la lectura, que falla si no se tiene conexión. Por ello, se hace necesario comprobar desde los fragmentos si se dispone de conexión o no para continuar con el flujo normal de la app o mostrar el mensaje genérico de error, lo cual se ha hecho mediante un método obtenido de un fragmento de código online [104].

Con casi todas las tareas del sprint finalizadas, pasa al siguiente la tarea de refactorización de fragmentos para el manejo de errores de la base de datos.

7.2.8. Sprint 6 (05/05/21 - 19/05/21)

En el transcurso de este sprint, justo al finalizar su primera semana, se darán por finalizadas las prácticas en empresa, por lo que al poder dedicar más tiempo la segunda semana, la carga de trabajo planificada se ha subido hasta las 40 horas.

En el *Sprint Backlog* se han incluido las siguientes historias de usuario del *Product Backlog* hasta completar 30 horas junto con 10 horas de documentación, para no arrastrar toda la elaboración de la memoria al final del proyecto, y se ha dejado la refactorización de fragmentos pendiente del sprint anterior como tarea extra para cuando se disponga de tiempo, ya que es menos prioritaria.

En la Tabla 7.7 se puede observar el desglose de tareas del presente sprint.

El total de tiempo dedicado en este sprint ha sido de **34 horas 55 minutos**. No se pudo alcanzar el objetivo de 40 horas debido a la realización de una práctica de la asignatura del cuatrimestre y de la memoria de prácticas en empresa. En concreto, se pudieron hacer alrededor de 5 horas menos, lo que supone el trabajo de un día.

En la **HU10**, se decidió en la tarea de revisión de IU añadir en cada tarjeta de pedido un botón "Eliminar", para poder borrar pedidos análogamente a las citas. Para establecer el listener de este botón surgieron algunos problemas, ya que el listener tenía que poder acceder al ViewModel correspondiente, elemento sólo accesible desde el fragmento, pero a su vez se necesitaba el ID del pedido, sólo accesible desde el ViewHolder (clase interna del adaptador que transforma los objetos de pedidos en las tarjetas de pedido para mostrarlas en el *RecyclerView*). La solución que se adoptó finalmente es establecer una etiqueta (*tag*) al botón, que corresponda con el ID del pedido que muestra la tarjeta que contiene dicho botón, y colocar el listener en el fragmento para tener acceso al ViewModel. En la tarea de testeo de esta historia de usuario, se volvió a intentar introducir en *GitLab CI/CD* los tests de IU utilizando otra guía [31], pero sin éxito nuevamente por errores en la ejecución.

Durante el desarrollo de la **HU12**, al tener que hacer un método en el DAO para recuperar un establecimiento por ID, se ha visto que, como en el caso de que no exista ninguno con dicho ID se devuelve `null`, al igual que cuando hay un error en la base de datos, es necesario distinguir los casos de alguna manera. Al usuario se le seguirá mostrando el mismo error genérico que hasta ahora, pero para facilitar la tarea al desarrollador, se han introducido los logs con mensajes descriptivos del error [16].

Con la implementación de la **HU12**, también se ha realizado la implementación de la futura **HU16**, y con la implementación de la **HU13**, se ha realizado la de la **HU17**.

En la implementación de la **HU14** se introdujo un atributo `establishmentId` en la clase `Order`, por lo que consecuentemente hubo que adaptar el resto del código y tests a este cambio, así como el diagrama de secuencia.

Al finalizar este sprint, se decidió hacer un cambio en el *Product Backlog* inicial con el fin de simplificar el desarrollo, cambiando de orden de prioridad algunas épicas e introduciendo una nueva, como se explica en el Capítulo 2, dando lugar al *Product Backlog* final (Tabla 2.14). Como todas las tareas de este sprint han sido finalizadas, no pasa ninguna al siguiente.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
-	▪ Documentación del Capítulo 1	10 horas	12 horas 10 minutos	Finalizado
HU10	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	4 horas 45 minutos	Finalizado
HU11	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	4 horas 5 minutos	Finalizado
HU12	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	4 horas 25 minutos	Finalizado
HU13	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	1 hora 50 minutos	Finalizado
HU14	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	5 horas 35 minutos	Finalizado
-	▪ Refactorización de fragmentos	-	2 horas 5 minutos	Finalizado

Tabla 7.7. Tareas del sprint 6

7.2.9. Sprint 7 (19/05/21 - 02/06/21)

Aunque en un principio se pretendía dedicar un total de 50 horas al sprint puesto que ya se acabaron las prácticas en empresa, se ha decidido planificar un sprint de 45 horas, debido al tiempo que consumirán el último examen de la asignatura del cuatrimestre y la finalización de la memoria de prácticas.

Como no hay tareas pendientes del sprint anterior, se ha planificado seguir con las historias de usuario del *Product Backlog* y dedicar 15 horas a la documentación.

En la Tabla 7.8 se pueden ver las tareas llevadas a cabo en el sprint 7.

El tiempo total dedicado del sprint ha sido de **46 horas 40 minutos**, por lo que se ha cumplido y sobrepasado con la planificación de 45 horas.

Tras consultarlo con la tutora, se vio que el atributo `establishmentId` de `Order` introducido en el sprint anterior no es la solución más apropiada, ya que de esa manera las clases de las entidades del dominio no tendrían ninguna dependencia ni relación entre ellas, y al no ser muy complejas, la idea es que conserven esas relaciones. Por ello, se crea una nueva tarea de refactorización de código para sustituir el atributo `establishmentId` por uno de tipo `Establishment`, con todos los cambios que eso conlleva. Tras investigar cómo se podría hacer el cambio, se ha visto que no es posible crear dependencias entre los DAO, ya que debido a la asincronía de la base de datos de *Firebase*, no se pueden anidar varias llamadas "get" a la base de datos utilizando esperas activas, que eran necesarias en este caso para cargar los establecimientos en cada una de las citas que se recuperan. Por tanto, la decisión que se ha tomado es utilizar una adaptación del patrón Proxy, consistente en recuperar las citas y pedidos con su atributo de tipo `Establishment` como un objeto de dicho tipo vacío (excepto por su ID), y en el ViewModel, antes de devolver las citas o pedidos al fragmento, recuperar el establecimiento correspondiente mediante su ID. De esa manera, al fragmento le llegará el objeto `Appointment` o `Order` completo, y es el ViewModel el que se encarga de hacer de intermediario en este trabajo.

Durante la realización de la **HU20**, debido a que dicho fragmento es accedido desde el menú lateral, se ha descubierto un bug que existía a la hora de navegar con el mismo. La modificación del menú lateral para arreglarlo conllevaba a su vez la modificación de la barra superior de acción de la app para poder funcionar conjuntamente, por lo que se sustituyó la `ActionBar` por la `Toolbar`, manteniendo el mismo aspecto pero con funcionalidad correcta [30]. Tras el cambio, el menú lateral seguía sin funcionar ya que sus elementos no eran clickables, por lo que investigando sobre ello y con la ayuda de una pregunta en *Stack Overflow* [96] se llegó a la conclusión de que el orden en el que aparecen los elementos en el XML del fragmento sí influye, y colocando el menú lateral el último de ellos, funciona correctamente.

Se han finalizado todas las tareas planificadas y las añadidas a lo largo del sprint, por lo que no pasa ninguna al siguiente sprint.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
-	▪ Documentación del Capítulo 2	15 horas	19 horas 30 minutos	Finalizado
HU15	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	6 horas	Finalizado
HU16	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	35 minutos	Finalizado
HU17	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	50 minutos	Finalizado
HU18	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	1 hora 20 minutos	Finalizado
HU19	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	2 horas	Finalizado
HU20	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	7 horas 25 minutos	Finalizado
-	▪ Refactorización de código	-	9 horas	Finalizado

Tabla 7.8. Tareas del sprint 7

7.2.10. Sprint 8 (02/06/21 - 16/06/21)

Puesto que la asignatura del cuatrimestre fue finalizada en el sprint anterior y la memoria de prácticas también, a partir de este sprint se puede dar dedicación completa a la realización del TFG, planificando un sprint de 50 horas para compensar con los de 30 horas durante la realización de prácticas en empresa.

Además de incluir las siguientes historias de usuario, se ha planificado 20 horas para avanzar en la documentación y 2 horas 30 minutos para incluir los tests de IU en el *pipeline* de integración continua.

La Tabla 7.9 muestra las tareas llevadas a cabo en este sprint.

El tiempo total dedicado al sprint fue de **53 horas 30 minutos**, sobrepasando las 50 horas planificadas.

Como la tarea de documentación del Capítulo 3 se finalizó sin consumir todas las horas planificadas para ello, se añadió al sprint una nueva tarea para continuar con la memoria del TFG, en este caso con el Capítulo 4.

En cuanto a la mejora de *GitLab CI/CD*, puesto que se llevaba varios meses intentando incluir los tests de IU en el *pipeline* sin éxito, se decidió recurrir a Samuel Alfageme, antiguo alumno de la Escuela con gran conocimiento sobre *GitLab*. La tarea no fue finalizada puesto que se quedó pendiente una reunión y más pruebas para una fecha perteneciente al sprint siguiente.

En la **HU21** se ha invertido más tiempo del estimado dado que en un principio se pensaba incluir la funcionalidad de notificaciones *push*. Tras unas horas de investigación a fondo, se ha llegado a la conclusión de que para poder gestionar notificaciones debidas a cambios en nodos de la base de datos, es necesario un script *Node.js* para el backend, por lo que acordándolo con la tutora se ha decidido dejar las notificaciones *push* fuera del alcance del proyecto.

También, en la **HU22** se ha dedicado más tiempo del planificado debido a problemas con el formato del texto editable de la ventana de diálogo personalizada de respuesta de rechazo. Tras un tiempo investigando y probando por qué todo el texto aparecía en una sola línea y no en varias, se ha deducido con ayuda de una pregunta de *Stack Overflow* [95] y más pruebas, que se trata de un bug de Android, ya que estableciendo como tipo de entrada `InputType.TYPE_CLASS_TEXT` funciona, pero si se establece `InputType.TYPE_TEXT_VARIATION_LONG_MESSAGE`, que es una variación del anterior y debería funcionar igual (en el XML lo hace), no funciona.

En la tarea de revisión de IU de la **HU23**, se ha decidido simplificar la interfaz de usuario a base de incluir en la pantalla de "Citas solicitadas" también las pantallas de respuesta de rechazo de cita y la confirmación de cita, puesto que muestran la misma información y tiene sentido agruparlas. Lo mismo se hará con las historias de usuario análogas de pedidos. Asimismo, se ha decidido en esta versión de la app no ofrecer la opción de cancelar la cita, sino que sea el propio dueño de negocio el que tenga que eliminarla de la agenda para cancelarla. Para comenzar con la implementación de esta historia de usuario, se ha hecho una refactorización para añadir un atributo a la clase `Order` (y por tanto a `Appointment` que

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
-	▪ Documentación del Capítulo 3	20 horas	15 horas 5 minutos	Finalizado
-	▪ Mejora de *GitLab CI/CD*	2 horas 30 minutos	5 horas 50 minutos	En progreso
HU21	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	6 horas 15 minutos	Finalizado
HU22	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	3 horas 35 minutos	Finalizado
HU23	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	13 horas 10 minutos	Finalizado
HU24	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	2 horas 55 minutos	Finalizado
HU25	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	1 hora 45 minutos	Finalizado
HU26	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	0 horas	No comenzado
-	▪ Documentación del Capítulo 4	-	4 horas 55 minutos	En progreso

Tabla 7.9. Tareas del sprint 8

hereda de ella) de tipo `Client`, para asociar cada cita y pedido con un cliente si este atributo no es `null` (puede ocurrir que no esté asociado con ninguno si es el propio dueño de negocio el que registra la cita o pedido, en cuyo caso tendrá sólo el atributo que indica el nombre del cliente). También, se ha creado una nueva clase `IdentifiedUser` que generalizará a las clases `Client` y `Establishment`, las cuales tienen atributos en común.

Pasan al siguiente sprint todas las tareas de la **HU26**, que no se ha podido comenzar, y las tareas de documentación del Capítulo 4 y de mejora de *GitLab CI/CD*, ambas en progreso.

7.2.11. Sprint extra 1 (16/06/21 - 30/06/21)

Con este sprint se comienzan los dos sprints de margen que se habían establecido en la planificación inicial. Como aún quedan tareas por hacer, se hará uso de ellos. El objetivo de este sprint es finalizar la implementación de la app con todas las historias de usuario restantes, por lo que se ha estimado para el mismo unas 55 horas, que según los puntos de historia restantes y la tarea de mejora de la integración continua, es el tiempo que puede llevar finalizarlo.

En este caso, como lo prioritario consiste en acabar el desarrollo de las historias de usuario, la tarea de documentación proveniente del sprint anterior se ha dejado como tarea extra por si sobra tiempo.

En la Tabla 7.10 se pueden ver todas las tareas realizadas en este penúltimo sprint.

El tiempo total dedicado al sprint fue de **61 horas 15 minutos**, superando lo planificado incluso aunque se estimara por lo alto.

La tarea de añadir los tests de IU a la integración continua sigue en progreso una vez acabado el sprint, pues se han realizado diversas pruebas y durante el curso del siguiente sprint se tendrá una reunión final con Samuel para concluir si se puede y si merece la pena añadir finalmente estos tests al *pipeline* o no, puesto que da muchos problemas durante la ejecución.

En las historias **HU26** y **HU28** parece que se ha sobreestimado el tiempo planificado, ya que se disponía del código base desarrollado en historias de usuario anteriores, lo que ha permitido acelerar considerablemente el ritmo limitando las tareas a realizar algunas adaptaciones. De la misma manera, el contenido de la base de datos ya estaba adaptado desde historias de usuario anteriores y no ha hecho falta manipularlo en estos casos.

Para la implementación del sistema de autenticación en la **HU29**, se ha introducido al proyecto *Firebase Authentication*, consultando su documentación para aprender a utilizarlo [41]. A pesar de la estimación de 10 horas de esta historia de usuario, se ha empleado un número considerablemente mayor de horas dado que se tuvo que probar con varias soluciones distintas. Finalmente, se decidió crear una clase de utilidad `AuthUtils` con los métodos necesarios para la gestión de usuarios y sesiones. Después de esto, hubo que adaptar el resto del código del proyecto y sus tests al nuevo sistema de autenticación, que suponía una gran cantidad de código dado que toda la parte de gestión de citas y pedidos ya estaba implementada.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
-	▪ Mejora de *GitLab CI/CD*	2 horas 30 minutos	3 horas	En progreso
HU26	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	50 minutos	Finalizado
HU27	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	50 minutos	Finalizado
HU28	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	2 horas	Finalizado
HU29	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	10 horas	21 horas 55 minutos	Finalizado
HU30	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	2 horas 30 minutos	Finalizado
HU31	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	4 horas 45 minutos	Finalizado
HU32	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	55 minutos	Finalizado

(Continúa en la página siguiente)

(Empieza en la página anterior)

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
HU33	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	2 horas	Finalizado
HU34	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	5 horas	4 horas 40 minutos	Finalizado
HU35	▪ Análisis ▪ Diseño ▪ Implementación ▪ Revisión de IU ▪ Testeo	2 horas 30 minutos	25 minutos	Finalizado
-	▪ Mejora de IU	-	15 horas	En progreso
-	▪ Revisión de la documentación de los Capítulos 1-3	-	2 horas 10 minutos	Finalizado
-	▪ Documentación del Capítulo 4	-	15 minutos	En progreso

Tabla 7.10. Tareas del sprint extra 1

Además, en los tests de IU surgieron problemas al introducir esta nueva característica, ya que la técnica utilizada hasta ahora de espera activa para manejar la asincronía no se podía usar, puesto que no se puede hacer un seguimiento del proceso de inicio de sesión ejecutado por *Firebase* y no ocurre ningún evento que indique que se ha iniciado sesión para poder continuar con el test. La solución adoptada ha sido introducir una espera, en este caso establecida en 1 segundo, que ha sido el mínimo tiempo que se ha comprobado empíricamente que tarda aproximadamente en iniciar sesión el servidor, de tal manera que no perjudique significativamente al tiempo total de ejecución de los tests.

Para la **HU34** se ha introducido al proyecto la API de *Google Maps*, para lo que se ha consultado su documentación oficial [51]. En la fase de testeo, se ha observado que *Espresso*, el framework oficial para tests de IU en Android, no posee funcionalidades para testear marcadores dentro de un mapa interactivo, por lo que los tests de IU de esta historia y la siguiente se han limitado a comprobar que efectivamente se muestra correctamente el mapa

por pantalla. Con el desarrollo de esta historia de usuario, la mayor parte de la **HU35** también queda hecha, por lo que habrá que dedicar tiempo solamente a retocar algunos detalles en implementación y tests.

Como se consiguió finalizar el desarrollo de las historias de usuario antes de lo previsto, se añadieron al sprint una tarea de mejora de IU y tareas de documentación.

En la tarea de mejora de IU se introdujo *Firebase Cloud Storage* para almacenar en la nube las imágenes de perfil de los usuarios identificados. Para ello, se consultó su documentación oficial [37]. También se añadió la funcionalidad de explorar en el almacenamiento interno del dispositivo para elegir la foto de perfil en el registro de un cliente. Para ello es necesario empezar una nueva actividad, y como `startActivityForResult` (el método tradicional) ha sido deprecado hace relativamente poco, se ha utilizado el nuevo método recomendado por Android, `registerForActivityResult` [8].

Pasan al siguiente sprint las tareas de mejora de IU y de *GitLab CI/CD* por reuniones pendientes que tendrán lugar durante las próximas semanas, y la tarea de documentación.

7.2.12. Sprint extra 2 (30/06/21 - 14/07/21)

En este último sprint, de duración de 2 semanas también, se ha decidido no estimar tiempo para las tareas pendientes para finalizar el proyecto, dado que se dedicará todo lo necesario para darlas por finalizadas. También es posible que el proyecto se finalice antes de la fecha de fin de sprint, por lo que lo más lógico era no dar una estimación.

Las tareas que se incluyen en el sprint son las pendientes del sprint anterior y todo el resto de la documentación.

En la Tabla 7.11 se muestra el desglose de tareas llevadas a cabo en el sprint.

El tiempo total dedicado a este sprint finalmente fue de **64 horas 40 minutos**. Además, no se ha tenido en cuenta la preparación de la presentación para la defensa del TFG, puesto que la presente memoria se debe entregar antes de elaborar dicha presentación.

En la tarea de mejora de IU, para darla por finalizada, se tuvo una reunión con Alejandra Martínez, docente de la Escuela de Ingeniería Informática de la Universidad de Valladolid con especialidad en interacción persona-computadora, para hacer una revisión final de la interfaz de usuario de la app y recibir sugerencias sobre cómo hacer más usable la app u otros comentarios. Algunas de las sugerencias fueron aplicadas y otras, que se explicarán en el Capítulo 8, quedarán como líneas futuras de trabajo para una nueva iteración de mejora de IU.

Para concluir con la tarea de mejora de *GitLab CI/CD*, se tuvo una última conversación con Samuel, en la que se decidió finalmente no incluir los tests de IU al *pipeline* de integración continua en *GitLab*. Las distintas limitaciones encontradas y conclusiones aparecen documentadas en la Sección 6.2.2.

Historia de usuario	Tareas	Tiempo estimado	Tiempo empleado	Estado
-	▪ Mejora de *GitLab CI/CD*	-	30 minutos	Finalizado
-	▪ Mejora de IU	-	1 hora 45 minutos	Finalizado
-	▪ Documentación del Capítulo 4	-	3 horas 5 minutos	Finalizado
-	▪ Documentación del Capítulo 5	-	17 horas 45 minutos	Finalizado
-	▪ Documentación del Capítulo 6	-	10 horas 45 minutos horas	Finalizado
-	▪ Documentación del Capítulo 7	-	13 horas 15 minutos	Finalizado
-	▪ Documentación del Capítulo 8	-	1 hora 10 minutos	Finalizado
-	▪ Documentación de Anexos	-	5 horas 10 minutos	Finalizado
-	▪ Revisión de toda la documentación	-	6 horas 15 minutos	Finalizado
-	▪ Preparación de tests de usabilidad	-	45 minutos	Finalizado
-	▪ Realización de tests de usabilidad	-	2 horas 30 minutos	Finalizado
-	▪ Solución de algunos bugs	-	1 hora 45 minutos	Finalizado

Tabla 7.11. Tareas del sprint extra 2

Para completar la documentación del Capítulo 6 (Implementación y pruebas), se han realizado tests de usabilidad a 3 de los usuarios potenciales de la app entrevistados al principio del proyecto, y a 2 usuarios con rol de cliente, por lo que se añadieron dos tareas relacionadas con esto: una de preparación y otra de realización de los tests.

Realizando los tests de usabilidad se descubrieron algunos pequeños bugs, por lo que se abrió una nueva tarea en el sprint para solventarlos.

Finalmente, tras terminar la documentación y su revisión, el proyecto se dio por finalizado dos días antes de terminar el sprint, el **12 de julio**.

7.3. Resumen de la ejecución del proyecto

Una vez finalizado el proyecto, es interesante hacer un contraste entre la planificación del mismo y cómo se ha desarrollado realmente, por lo que en esta sección se compararán diversos aspectos.

7.3.1. Calendarización planificada y real

La calendarización inicial, que se puede ver en la Tabla 2.1, ha sido en su mayor parte respetada. Finalmente se ha hecho uso de los dos sprints extra, con su duración planificada, y la gran mayoría los eventos planificados han tenido lugar con normalidad. La carga de trabajo de los sprints ha sido adaptada a las circunstancias del momento, pero eso ya se tuvo en cuenta en la planificación inicial, y el total de horas requeridas se ha podido cumplir, como se verá en la siguiente subsección.

Entre los detalles a destacar, se pueden mencionar dos *Scrum Weekly*, que por diversos motivos como por ejemplo una semana de trabajo intensivo (periodo de corrección de exámenes y prácticas antes del cierre de actas), la tutora no tuvo disponibilidad para realizar la reunión por videoconferencia y se hizo la reunión por escrito. Se redactaron los puntos principales a mencionar en la reunión, dejando para la siguiente semana la presentación en directo de nuevas características en la app, y cuando la tutora tuvo disponibilidad, se realizaron también por escrito los comentarios necesarios sobre ello. Esto no supuso ningún problema puesto que en realidad la esencia de la reunión (informar sobre avances y preguntar dudas importantes) siguió siendo la misma, sólo que en lugar de realizarla de manera síncrona, se realizó por escrito y de manera asíncrona.

Otro detalle es que, a modo de recuperación de horas de trabajo, el período de vacaciones de Semana Santa planificado inicialmente, se convirtió en una ampliación del sprint 3 (el que estaba en curso en ese momento).

Finalmente, en el último sprint del proyecto (sprint extra 2), a pesar de haber mantenido la duración de 2 semanas establecida, se pudo finalizar antes, y por tanto los últimos días del sprint no se trabajó sobre ello.

En conclusión, el proyecto se desarrolló desde el **19 de enero** hasta el **12 de julio** del 2021, resultando en una duración aproximada de 6 meses.

7.3.2. Tiempo estimado y empleado

El tiempo requerido para la realización del TFG según la guía docente son 300 horas (explicado en la Sección 2.3), y para tener un margen amplio por posibles retrasos en la fecha de finalización, se planificó un total de 340 horas de trabajo con los 8 sprints principales y el sprint 0 de preparación, y un total de 420 horas con los sprints extra.

Realizando la suma del tiempo dedicado de cada uno de los sprints desarrollados en la sección anterior de este Capítulo, el cómputo total de tiempo dedicado al proyecto es de **448 horas 30 minutos**, por lo que se ha cumplido con creces las 300 horas requeridas para la elaboración del TFG.

7.3.3. Costes finales

Teniendo en cuenta una duración aproximada de 6 meses para el proyecto, y 448 horas de dedicación al mismo, se recalculará el coste total y se contrastará con los presupuestos simulados y reales planificados en la Sección 2.5.

Coste simulado final

Los elementos son los mismos ya explicados en el prespuesto simulado. Con el tiempo y duración dedicados realmente al proyecto, el cálculo del coste simulado final queda como se ve en la Tabla 7.12.

Concepto	Precio unitario	Cantidad	Total
Hora de trabajo de desarrollador Android	13,07€/hora	448 horas	5855,36€
Seguridad Social del empleado	6,55€/hora	448 horas	2934,40€
Equipo del empleado (*MacBook Pro 2020*)	44,35€/mes	6 meses	266,10€
Smartphone de pruebas (*Samsung Galaxy S10+*)	13,52€/mes	6 meses	81,12€
Espacio de trabajo coworking	165€/mes	6 meses	990€
Licencia *Microsoft 365 Empresa Básico*	4,12€/mes	6 meses	24,72€
Licencia *Balsamiq Cloud*	7,42€/mes	6 meses	44,52€
Licencia *Astah Professional*	11,67€/mes	6 meses	70,02€
TOTAL			**10266,24€**

Tabla 7.12. Coste simulado final

Con la normalización del 25 % que se había aplicado al presupuesto simulado a modo de colchón, la estimación era de un coste total simulado de 9878,46€, que al final ha resultado ser de **10266,24€**, por lo que la estimación inicial se ha quedado relativamente cerca, pero no habría cubierto todo el coste simulado final, debido al mayor número de horas empleado en el proyecto y el sobrecoste que esto supone.

Coste real final

En este caso los elementos también son los mismos que los explicados en el presupuesto real. Con la cantidad de tiempo y duración que se han dedicado realmente al proyecto, el cálculo del coste real final es el que se muestra en la Tabla 7.13.

Como el consumo medio de un portátil en 1 hora es de 0,11 kWh (como se explica en el presupuesto real), en las 448 horas dedicadas ha sido de 49,28 kWh.

Concepto	Precio unitario	Cantidad	Total
Electricidad	0,065€/kWh	49,28 kWh	3,20€
Equipo del estudiante (*HP Notebook*)	15,60€/mes	6 meses	93,60€
Smartphone de pruebas (*Samsung Galaxy S10+*)	13,52€/mes	6 meses	81,12€
TOTAL			**177,92€**

Tabla 7.13. Coste real final

El total estimado en el presupuesto real era de 148,03€, que debido al mayor tiempo dedicado, al final ha resultado ser de **177,92€**.

Capítulo 8

Conclusiones y líneas futuras

8.1. Conclusiones

Tras la finalización del proyecto, se puede concluir definiendo el mismo como trabajoso, pero a su vez satisfactorio y gratificante, ya que ver cómo las numerosas horas invertidas en él se han convertido en la aplicación resultante, **AppGenda**, además de todo lo aprendido, hacen que sienta que ha merecido la pena. A lo largo del proyecto, se ha hecho un recorrido por la gran mayoría de aspectos aprendidos en el grado, con especial hincapié en los que conciernen a la mención correspondiente, en este caso, Ingeniería del Software.

En cuanto a los objetivos de desarrollo establecidos en la Sección 1.5.1:

- Se ha conseguido que la app sea capaz de gestionar las citas de un negocio.
- Se ha conseguido que la app sea capaz de gestionar los pedidos de un negocio.
- Se ha conseguido realizar una interacción entre cliente y dueño de negocio, a través de las solicitudes de citas y pedidos y sus respectivas respuestas realizadas mediante la app.
- Se ha conseguido que la app posea una sección dedicada a un mapa interactivo para indicar la ubicación de los establecimientos registrados en la app.
- Se ha conseguido que la app posea un tablón de anuncios básico que permita consultar y publicar anuncios.

Además, para hacer la app más usable, se ha realizado una fase de mejora de interfaz de usuario que también la ha convertido en más atractiva y agradable al uso.

En cuanto a los objetivos personales establecidos en la Sección 1.5.2:

- Se ha llevado a cabo un proyecto de desarrollo software de principio a fin, pasando por todas sus fases y las complicaciones de cada una, lo que implica un gran aprendizaje al poder haberlo vivido en primera persona.
- Se ha aplicado el marco de trabajo ágil Scrum, aprendiendo las nociones básicas sobre cómo aplicarlo y poniéndolo en práctica durante los 6 meses de duración del proyecto.
- Se ha desarrollado gran cantidad de código Android en Java, utilizando técnicas, componentes y elementos que no se conocían y permitiendo adquirir un amplio conocimiento al respecto.
- Se ha desarrollado una interfaz de usuario siguiendo la guía de buenas prácticas *Material Design*, realizando posteriormente tests de usabilidad para verificar cómo de usable ha resultado ser realmente.
- Se han aplicado buenas prácticas a lo largo de todo el proyecto como: uso de una guía de estilo personal para el código, aplicación de las pautas de *Material Design*, realización de tests automatizados del código, realización de tests de usabilidad ante un experto y ante usuarios finales e integración continua de la aplicación.

8.2. Líneas de trabajo futuras

Entre las principales posibles líneas de trabajo futuras en la app, destacan las siguientes:

- **Nuevas funcionalidades.** Debido a la limitación de tiempo en el proyecto, no se ha podido abordar todos los requisitos planteados inicialmente como parte de la concepción del producto, pero sí los principales. Estos requisitos, mencionados en las épicas no desarrolladas del *Product Backlog* final (Tabla 2.14, **EP11-14**), incluyen la funcionalidad de **retrasos en las citas**, de manera que se pueda notificar al cliente si va a existir un retraso en su cita, y un **rol de administrador** de la app, encargado de gestionar los establecimientos presentes en la misma. Además, algunas de las funcionalidades actualmente en la app están ya preparadas para añadir nuevas características como **añadir imágenes personalizadas en los anuncios**, o una **gestión de usuarios más avanzada**, pudiendo modificar los datos proporcionados en el momento del registro.
- **Notificaciones *push*.** La otra mejora evidente que está presente en la gran mayoría de las apps es la introducción de notificaciones push. En este proyecto, debido a que la utilización de notificaciones push con la base de datos de *Firebase* implicaba desarrollo de código de backend, se ha decidido no trabajar sobre ello.
- **Mejora del *pipeline* de integración continua.** Como se detalla en la Sección 6.2.2, los tests de IU no se han conseguido añadir al *pipeline* en *GitLab CI/CD*. Si se consiguieran añadir y que el script de integración continua en su conjunto se ejecute en un tiempo aceptable, supondría una gran mejora en el proyecto.

- **Base de datos local.** La introducción de una base de datos local complementaria a la utilizada de *Firebase* permitiría mejorar el rendimiento y el uso de datos de datos de la app, incluso puede que utilizarla en modo offline, puesto que funcionaría a modo de caché y no habría que realizar tantas solicitudes remotas vía Internet.
- **Nueva iteración de mejora de IU.** Aunque se ha realizado una primera iteración de mejora de IU, ésta aún es mejorable. Una característica detectada en los tests de usabilidad, es que en dispositivos con pantallas pequeñas, la interfaz no se mostraba de forma óptima, por lo que es un punto a mejorar. Además, para afianzar este aspecto, se celebró una reunión con Alejandra Martínez Monés, docente en la Escuela con especialidad en el campo de la interacción persona-computadora (agradecimientos por la colaboración), y se recopilaron las siguientes mejoras que se pueden llevar a cabo:
  - Colocar el avatar del usuario en la barra superior de la app, de manera que es visible sin tener que abrir el menú lateral, oculto por defecto.
  - Cambiar la organización de las secciones para hacer más intuitivo el acceso a algunas de ellas más importantes localizadas en el menú lateral (conclusión que también se ha obtenido de los tests de usabilidad, Sección 6.3.3).
  - Eliminar mensajes de texto con instrucciones y dejar que los componentes de IU "hablen por sí mismos", presentándolos de manera más intuitiva (de nuevo, conclusión obtenida en los tests de usabilidad realizados).
  - Mantener en todo momento el flujo de la app para mantener la experiencia de usuario (por ejemplo, tras iniciar sesión, volver a la pantalla en la que se estaba).

Bibliografía

[1] Android. Qué es Android. https://www.android.com/intl/es_es/what-is-android/. Accedido el 04/06/2021.

[2] Android Developers. Android Kotlin Fundamentals: 5.1 ViewModel. https://developer.android.com/codelabs/kotlin-android-training-view-model?index=..%2F..android-kotlin-fundamentals. Accedido el 27/04/2021.

[3] Android Developers. Android Kotlin Fundamentals: RecyclerView fundamentals. https://developer.android.com/codelabs/kotlin-android-training-recyclerview-fundamentals?index=..%2F..android-kotlin-fundamentals. Accedido el 26/04/2021.

[4] Android Developers. Ciclo de vida de los fragmentos. https://developer.android.com/guide/fragments/lifecycle. Accedido el 13/04/2021.

[5] Android Developers. Codelabs de Conceptos básicos de Kotlin para Android. https://developer.android.com/courses/kotlin-android-fundamentals/toc. Accedido el 28/02/2021.

[6] Android Developers. Codelabs de Conceptos básicos para desarrolladores de Android. https://developer.android.com/courses/fundamentals-training/toc-v2. Accedido el 01/03/2021.

[7] Android Developers. Cómo agregar un botón de acción flotante. https://developer.android.com/guide/topics/ui/floating-action-button. Accedido el 02/05/2021.

[8] Android Developers. Cómo obtener un resultado de una actividad. https://developer.android.com/training/basics/intents/result. Accedido el 29/06/2021.

[9] Android Developers. Cómo usar Safe Args para pasar datos con seguridad de tipo. https://developer.android.com/guide/navigation/navigation-pass-data#Safe-args. Accedido el 21/02/2021.

[10] Android Developers. Descripción general de ViewModel. https://developer.android.com/topic/libraries/architecture/viewmodel. Accedido el 04/07/2021.

[11] Android Developers. Download Android Studio and SDK tools. https://developer.android.com/studio. Accedido el 04/06/2021.

[12] Android Developers. Espresso. https://developer.android.com/training/testing/espresso. Accedido el 07/06/2021.

[13] Android Developers. Fragmentos. https://developer.android.com/guide/components/fragments. Accedido el 05/06/2021.

[14] Android Developers. GridView. https://developer.android.com/reference/android/widget/GridView. Accedido el 07/03/2021.

[15] Android Developers. Introducción a las actividades. https://developer.android.com/guide/components/activities/intro-activities. Accedido el 05/06/2021.

[16] Android Developers. Log. https://developer.android.com/reference/android/util/Log. Accedido el 11/05/2021.

[17] Android Developers. Navigation. https://developer.android.com/guide/navigation. Accedido el 21/02/2021.

[18] Android Developers. Prueba tu app. https://developer.android.com/studio/test. Accedido el 07/02/2021.

[19] Android Developers. Recursos de strings. https://developer.android.com/guide/topics/resources/string-resource. Accedido el 16/03/2021.

[20] Android Developers. Selectores. https://developer.android.com/guide/topics/ui/controls/pickers. Accedido el 02/04/2021.

[21] Android Developers. TableLayout. https://developer.android.com/reference/android/widget/TableLayout. Accedido el 07/03/2021.

[22] Android Studio. Android Platform/API Version Distribution. Consultado el 20/02/2021.

[23] Apple. MacBook Pro de 13 pulgadas - Gris espacial. https://www.apple.com/es/shop/buy-mac/macbook-pro/13-pulgadas-gris-espacial-procesador-intel-core-i5-de-cuatro-n%C3%BAcleos-a-2-ghz-con-intel-iris-plus-graphics-512gb. Accedido el 09/02/2021.

[24] Astah. Astah Professional: UML, ER, DFD & Flowchart Software. https://astah.net/products/astah-professional/. Accedido el 07/06/2021.

[25] Astah. Team Licensing Options. https://astah.net/pricing/team/. Accedido el 09/02/2021.

[26] Atlassian. Git Branch. https://www.atlassian.com/git/tutorials/using-branches. Accedido el 03/06/2021.

[27] Balsamiq. Balsamiq Cloud Plans & Pricing. https://balsamiq.com/buy/#cloud. Accedido el 09/02/2021.

[28] Balsamiq. Balsamiq Wireframes. https://balsamiq.com/wireframes/. Accedido el 07/06/2021.

[29] Bob Hughes y Mike Cotterell. Software Project Management, 5ª edición, McGraw-Hill Higher Education. Consultado el 30/05/2021.

[30] CodePath. Using the App Toolbar. https://guides.codepath.com/android/using-the-app-toolbar. Accedido el 29/05/2021.

[31] Computer Science Blog. Using Gitlab to set up a CI/CD workflow for an Android App from scratch. https://blog.mi.hdm-stuttgart.de/index.php/2020/02/24/using-gitlab-to-set-up-a-ci-cd-workflow-for-an-android-app-from-scratch/. Accedido el 09/05/2021.

[32] Creative Commons. **Developing Android unit and instrumentation tests - Tutorial**. https://creativecommons.org/licenses/by-nc-nd/4.0/. Accedido el 11/07/2021.

[33] Docker Hub. androidsdk/android-30. https://hub.docker.com/r/androidsdk/android-30. Accedido el 11/06/2021.

[34] Ecorresponsabilidad. Uso del ordenador portátil vs ordenador de sobremesa. http://www.ecorresponsabilidad.es/fichas/portatil.htm. Accedido el 09/02/2021.

[35] El Coworking del Pop Up. Tarifas Coworking Valladolid. https://elcoworkingdelpopup.com/tarifas-2/. Accedido el 09/02/2021.

[36] Firebase. Cloud Storage for Firebase. https://firebase.google.com/docs/storage. Accedido el 28/06/2021.

[37] Firebase. Comienza a usar Cloud Storage en Android. https://firebase.google.com/docs/storage/android/start. Accedido el 28/06/2021.

[38] Firebase. Elige una base de datos: Cloud Firestore o Realtime Database. https://firebase.google.com/docs/database/rtdb-vs-firestore. Accedido el 14/03/2021.

[39] Firebase. Estructura tu base de datos. https://firebase.google.com/docs/database/android/structure-data. Accedido el 14/03/2021.

[40] Firebase. Firebase. https://firebase.google.com/. Accedido el 07/06/2021.

[41] Firebase. Firebase Authentication. https://firebase.google.com/docs/auth. Accedido el 19/06/2021.

[42] Firebase. Firebase Realtime Database. https://firebase.google.com/docs/database. Accedido el 07/06/2021.

[43] Firebase. Instalación y configuración en Android. https://firebase.google.com/docs/database/android/start. Accedido el 14/03/2021.

[44] Git. About. https://git-scm.com/about. Accedido el 03/06/2021.

[45] GitHub Gist. Archivo .gitlab-ci.yml del usuario *illuzor*. https://gist.github.com/illuzor/988385c493d3f7ed7193a6e3ce001a68. Accedido el 24/03/2021.

[46] GitLab. Archivo .gitlab-ci.yml del proyecto *F-Droid*. https://gitlab.com/fdroid/fdroidclient/-/blob/master/.gitlab-ci.yml#L74-103. Accedido el 06/07/2021.

[47] GitLab. GitLab CI/CD. https://docs.gitlab.com/ee/ci/. Accedido el 07/02/2021.

[48] GitLab. Issue Boards. https://docs.gitlab.com/ee/user/project/issue_board.html. Accedido el 09/02/2021.

[49] GitLab. Setting up GitLab CI for Android projects. https://about.gitlab.com/blog/2018/10/24/setting-up-gitlab-ci-for-android-projects/. Accedido el 07/02/2021.

[50] GitLab. What is GitLab? https://about.gitlab.com/what-is-gitlab/. Accedido el 03/06/2021.

[51] Google Developers. Descripción general del SDK de Maps para Android. https://developers.google.com/maps/documentation/android-sdk/overview. Accedido el 26/06/2021.

[52] Google Play. Bookitit Calendario - SOLO EMPRESAS. https://play.google.com/store/apps/details?id=com.nubesis.bookititmobile2. Accedido el 30/05/2021.

[53] Google Play. Evernote - Organizador de notas. https://play.google.com/store/apps/details?id=com.evernote. Accedido el 17/05/2021.

[54] Google Play. Facebook. https://play.google.com/store/apps/details?id=com.facebook.katana. Accedido el 17/05/2021.

[55] Google Play. Google Calendar. https://play.google.com/store/apps/details?id=com.google.android.calendar. Accedido el 17/05/2021.

[56] Google Play. ReBox — Sistema de reservas para gimnasios. https://play.google.com/store/apps/details?id=genapp.rebox.app. Accedido el 17/05/2021.

[57] Google Play. Reservio - Sistema de cita previa online. https://play.google.com/store/apps/details?id=com.reservio. Accedido el 17/05/2021.

[58] Google Play. SalonAppy - Citas, Recordatorios, Ventas y Más. https://play.google.com/store/apps/details?id=com.kolayrandevu.isletme. Accedido el 17/05/2021.

[59] Google Play. Sistema de reserva de citas. https://play.google.com/store/apps/details?id=com.calengoo.booking. Accedido el 17/05/2021.

[60] Google Play. WhatsApp Business. https://play.google.com/store/apps/details?id=com.whatsapp.w4b. Accedido el 17/05/2021.

[61] Google Play. WodBuster. https://play.google.com/store/apps/details?id=santi.wodbuster. Accedido el 17/05/2021.

[62] Gradle. Gradle Build Tool. https://gradle.org/. Accedido el 11/07/2021.

[63] HP. HP Notebook. https://store.hp.com/SpainStore/Merch/Product.aspx?id=2R7Z4EA&opt=ABE&sel=NTB. Accedido el 09/02/2021.

[64] Indeed. ¿Cuánto se gana en España de Android developer? https://es.indeed.com/career/android-developer/salaries. Accedido el 09/02/2021.

[65] InformaticaPC. Factory Method. https://informaticapc.com/patrones-de-diseno/factory-method.php. Accedido el 05/07/2021.

[66] Javadoc.io. Stubbing with callbacks, Mockito 3.11.2 API. https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html#answer_stubs. Accedido el 12/04/2021.

[67] Jitsi. Jitsi Meet. https://meet.jit.si/. Accedido el 07/06/2021.

[68] Ken Schwaber y Jeff Sutherland. The 2020 Scrum Guide. https://scrumguides.org/scrum-guide.html. Accedido el 30/05/2021.

[69] Kent Beck, Mike Beedle, Arie van Bennekum y 14 autores más. Manifesto for Agile Software Development. https://agilemanifesto.org/. Accedido el 30/05/2021.

[70] Loentiendo. ¿Cuánto cuesta un trabajador? https://loentiendo.com/cuanto-cuesta-un-trabajador/. Accedido el 09/02/2021.

[71] Manh Phan. DAO pattern in Java. https://ducmanhphan.github.io/2019-02-15-DAO-pattern-in-java/. Accedido el 04/07/2021.

[72] Material Design. Bottom navigation. https://material.io/components/bottom-navigation/android. Accedido el 21/02/2021.

[73] Material Design. Color Tool. https://material.io/resources/color. Accedido el 21/02/2021.

[74] Material Design. Design. https://material.io/design. Accedido el 05/06/2021.

[75] Material Design. Designing a Material Theme: Color. https://material.io/blog/design-material-theme-color. Accedido el 21/02/2021.

[76] Material Design. Material Icons. https://material.io/resources/icons. Accedido el 21/02/2021.

[77] Medium. Create Android app with MVVM pattern simply using Android Architecture Component. https://medium.com/hongbeomi-dev/create-android-app-with-mvvm-pattern-simply-using-android-architecture-component-529d983eaabe. Accedido el 04/07/2021.

[78] Microsoft. Compara todas las ofertas de planes de Microsoft 365. https://www.microsoft.com/es-ww/microsoft-365/business/compare-all-microsoft-365-business-products?market=bz. Accedido el 09/02/2021.

[79] Microsoft. Microsoft Teams. https://www.microsoft.com/es-es/microsoft-teams/group-chat-software/. Accedido el 07/06/2021.

[80] Mike Cohn. User Stories Applied: For Agile Software Development (Addison-Wesley Signature Series (Beck)). Consultado el 28/01/2021.

[81] Mockito. Mockito framework site. https://site.mockito.org/. Accedido el 11/04/2021.

[82] Mountain Goat Software. User Stories. https://www.mountaingoatsoftware.com/agile/user-stories. Accedido el 28/01/2021.

[83] Netmind. Scrum: el pasado y el futuro. https://netmind.net/es/scrum-el-pasado-y-el-futuro/. Accedido el 31/05/2021.

[84] OOZOU. Reasons to use Android Single-Activity Architecture with Navigation Component. https://oozou.com/blog/reasons-to-use-android-single-activity-architecture-with-navigation-component-36. Accedido el 06/06/2021.

[85] Oscar Blancarte. Data Access Object (DAO) Pattern. https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/. Accedido el 04/07/2021.

[86] Overleaf. Overleaf, Editor de LaTeX online. https://es.overleaf.com/. Accedido el 07/06/2021.

[87] Paul Clements, Felix Bachmann, Len Bass y 6 autores más. Documenting Software Architectures: Views and Beyond, 2ª edición, Addison-Wesley Professional. Consultado el 06/07/2021.

[88] Phone House. Samsung Galaxy S10+ 128GB+8GB RAM. https://www.phonehouse.es/movil/samsung/galaxy-s10-plus-128gb-8gb-ram.html. Accedido el 06/07/2021.

[89] Programacion.net. Patrones de Diseño (XIII): Patrones Estructurales - Proxy. https://programacion.net/articulo/patrones_de_diseno_xiii_patrones_estructurales_proxy_1016. Accedido el 05/07/2021.

[90] Refactoring Guru. Adapter. https://refactoring.guru/es/design-patterns/adapter. Accedido el 05/07/2021.

[91] Refactoring Guru. Factory Method. https://refactoring.guru/es/design-patterns/factory-method. Accedido el 05/07/2021.

[92] Refactoring Guru. Patrones de diseño: Proxy en Java. https://refactoring.guru/es/design-patterns/proxy/java/example. Accedido el 05/07/2021.

[93] Rocket.Chat. Rocket.Chat - La plataforma de comunicación definitiva. https://rocket.chat/es/. Accedido el 07/06/2021.

[94] Solid Gear. Componentes de arquitectura de Android, de MVC a MVVM, por David González Verdugo. https://ahorasomos.izertis.com/solidgear/componentes-de-arquitectura-de-android-de-mvc-a-mvvm/. Accedido el 04/07/2021.

[95] Stack Overflow. Android EditText Multiline does not work as it should. https://stackoverflow.com/questions/25294498/android-edittext-multiline-does-not-work-as-it-should. Accedido el 12/06/2021.

[96] Stack Overflow. Button inside DrawerLayout is not clickable. https://stackoverflow.com/questions/25320805/button-inside-drawerlayout-is-not-clickable. Accedido el 29/05/2021.

[97] Stack Overflow. Calling callbacks with Mockito. https://stackoverflow.com/questions/13616547/calling-callbacks-with-mockito. Accedido el 12/04/2021.

[98] Stack Overflow. Dynamically add TableRow to TableLayout. https://stackoverflow.com/questions/3200691/dynamically-add-tablerow-to-tablelayout. Accedido el 09/03/2021.

[99] Stack Overflow. Dynamically added table rows not appearing. https://stackoverflow.com/questions/7541766/dynamically-added-table-rows-not-appearing. Accedido el 09/03/2021.

[100] Stack Overflow. Espresso match first element found when many are in hierarchy. https://stackoverflow.com/questions/32387137/espresso-match-first-element-found-when-many-are-in-hierarchy. Accedido el 05/04/2021.

[101] Stack Overflow. FloatingActionButton hide on list scroll. https://stackoverflow.com/questions/31617398/floatingactionbutton-hide-on-list-scroll. Accedido el 02/05/2021.

[102] Stack Overflow. How to add border around TableLayout? https://stackoverflow.com/questions/18003021/how-to-add-border-around-tablelayout. Accedido el 07/03/2021.

[103] Stack Overflow. How to assert inside a RecyclerView in Espresso? https://stackoverflow.com/questions/31394569/how-to-assert-inside-a-recyclerview-in-espresso. Accedido el 12/04/2021.

[104] Stack Overflow. How to check internet access on Android? InetAddress never times out. https://stackoverflow.com/questions/1560788/how-to-check-internet-access-on-android-inetaddress-never-times-out. Accedido el 05/05/2021.

[105] Stack Overflow. How to generate buildConfigField with String type. https://stackoverflow.com/questions/30796533/how-to-generate-buildconfigfield-with-string-type. Accedido el 12/04/2021.

[106] Stack Overflow. How to mock just one static method in a class using Mockito? https://stackoverflow.com/questions/63840898/how-to-mock-just-one-static-method-in-a-class-using-mockito. Accedido el 12/04/2021.

[107] Stack Overflow. How to set a custom minutes interval in TimePickerDialog in Android. https://stackoverflow.com/questions/7713538/how-to-set-a-custom-minutes-interval-in-timepickerdialog-in-android. Accedido el 04/04/2021.

[108] Stack Overflow. IllegalArgumentException: navigation destination xxx is unknown to this NavController. https://stackoverflow.com/questions/51060762/illegalargumentexception-navigation-destination-xxx-is-unknown-to-this-navcontr. Accedido el 01/03/2021.

[109] Stack Overflow. Java unit testing - how to unit test an async method (uses callback) with mockito answer. https://stackoverflow.com/questions/25099500/java-unit-testing-how-to-unit-test-an-async-method-uses-callback-with-mockit. Accedido el 12/04/2021.

[110] Stack Overflow. Testing Snackbar show with Espresso. https://stackoverflow.com/questions/33111882/testing-snackbar-show-with-espresso. Accedido el 05/04/2021.

[111] Stack Overflow. Timepicker with 15 minute intervals. https://stackoverflow.com/questions/32397884/timepicker-with-15-minute-intervals. Accedido el 04/04/2021.

[112] Stack Overflow. What is the best way to catch an IllegalArgumentException. https://stackoverflow.com/a/2117559. Accedido el 04/05/2021.

[113] Tarifaluzhora. ¿Cuánto cuesta el kilovatio hora de luz (kWh) en España? https://tarifaluzhora.es/info/precio-kwh. Accedido el 09/02/2021.

[114] Tokio School. ¿Cuál es el sueldo de un programador Android? https://www.tokioschool.com/noticias/cual-es-el-sueldo-de-un-programador-android/. Accedido el 09/02/2021.

[115] Universidad de Valladolid. Guía docente del Trabajo de Fin de Grado (mención Ingeniería de Software). https://alberguweb1.uva.es/guia_docente/uploads/2020/545/46976/1/Documento.pdf. Accedido el 31/05/2021.

[116] Vogella. Developing Android unit and instrumentation tests - Tutorial. https://www.vogella.com/tutorials/AndroidTesting/article.html. Accedido el 08/07/2021.

[117] Vysor. Vysor. https://www.vysor.io/. Accedido el 07/06/2021.

[118] Wikipedia. Android. https://es.wikipedia.org/wiki/Android. Accedido el 04/06/2021.

[119] Wikipedia. Camel case. https://en.wikipedia.org/wiki/Camel_case. Accedido el 11/07/2021.

[120] Wikipedia. FURPS. https://en.wikipedia.org/wiki/FURPS. Accedido el 01/07/2021.

[121] Wikipedia. Objeto de transferencia de datos. https://es.wikipedia.org/wiki/Objeto_de_transferencia_de_datos. Accedido el 04/07/2021.

[122] Wikipedia. Snake case. https://en.wikipedia.org/wiki/Snake_case. Accedido el 11/07/2021.

[123] Wikiwand. Adaptador (patrón de diseño). https://www.wikiwand.com/es/Adaptador_(patr%C3%B3n_de_dise%C3%B1o). Accedido el 05/07/2021.

[124] Yania Crespo González-Carvajal. Apuntes del Tema 3 de la asignatura Diseño de Software. Consultado el 05/07/2021.

[125] Yania Crespo González-Carvajal. Apuntes del Tema 5 de la asignatura Diseño de Software. Consultado el 05/07/2021.

[126] Yi Min Shum. Situación digital, Internet y redes sociales España 2021. https://yiminshum.com/redes-sociales-espana-2021/. Accedido el 16/05/2021.

[127] Yi Min Shum. Situación Global Móvil 2021. https://yiminshum.com/mobile-movil-mundo-2021/. Accedido el 16/05/2021.

[128] YouTube. 73. RecyclerView Personalizado en Android, del usuario *Cristian Henao*. https://developer.android.com/codelabs/kotlin-android-training-recyclerview-fundamentals?index=..%2F..android-kotlin-fundamentals. Accedido el 26/04/2021.

[129] Zoraida Ceballos de Mariño. SCRUM: roles y responsabilidades del SCRUM TEAM. https://xn--zoraidaceballosdemario-4ec.info/scrum/scrum-roles-y-responsabilidades-del-scrum-team/. Accedido el 31/05/2021.

Apéndice A

Manuales

A.1. Manual de instalación

Para instalar la app se requiere un **smartphone o tablet Android** con versión mínima de **Android 8.0 Oreo**.

El proceso de instalación se basa en simplemente acceder al repositorio de *GitHub* cuyo enlace se encuentra en el Apéndice B, **descargar el archivo .apk** en el dispositivo Android, y ejecutarlo desde el propio dispositivo para iniciar la instalación, que será guiada por un asistente del sistema operativo desde dicho momento.

Nota: es posible que previo a la instalación (o durante el proceso en algunas versiones) sea necesario activar la opción de instalar apps de fuentes desconocidas desde la configuración del dispositivo, puesto que es una instalación externa a *Google Play*, pero que en este caso no supone ningún peligro. Para ello la ruta a seguir en general es Ajustes > Seguridad > **Instalar apps desconocidas**, activando dicha opción solamente para esta instalación. Se recomienda volver a desactivar la opción después de instalar la app, por motivos de seguridad.

A.2. Manual de usuario

La primera vez que se accede a la app, se hace como usuario no identificado. Como usuario no identificado, las funcionalidades que están disponibles consisten en **consultar información de los establecimientos**, **acceder al mapa interactivo** para ver las ubicaciones de los establecimientos registrados en la app, y **consultar los anuncios del tablón**, publicados por establecimientos. Para más información sobre estas funcionalidades, consultar la sección del manual dedicada a los clientes.

Para **iniciar sesión** o **registrarse como cliente**, se accederá desde el menú lateral, como se muestra en las Figuras A.1, A.2 y A.3.

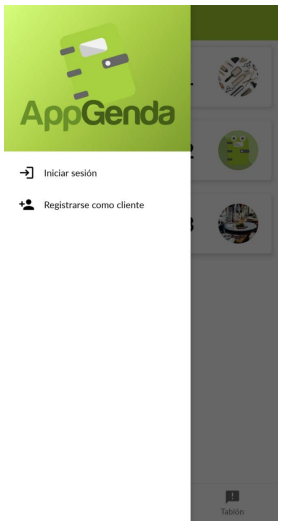

Figura A.1. Menú lateral

Figura A.2. Iniciar sesión

Figura A.3. Registro

Puesto que en la app existen dos roles diferenciados con funcionalidades distintas, **dueño de negocio** y **cliente**, a partir de este punto se explicarán por separado.

Dueño de negocio

La pantalla principal cuando se inicia sesión como dueño de negocio es la **sección "Agenda"** (Figura A.4). Desde aquí, podemos acceder a nuestra agenda de citas o de pedidos. En cualquiera de los casos, el siguiente paso es seleccionar el día para el que se desea consultar la agenda (Figura A.5).

Accediendo a la **agenda de citas** de un día, podemos ver los huecos disponibles y reservados (Figura A.6). El número de columnas de la tabla indica el número máximo de clientes que un establecimiento puede recibir en cada franja horaria de 15 minutos, y las tarjetas de cita siguen el siguiente código de colores: *verde* indica que el cliente ha confirmado su asistencia a la cita, y *amarillo* indica que está pendiente de confirmación. Además, para las citas solicitadas a través de la app, aparecerá la foto de perfil del cliente junto a la cita. Para ver los detalles de una cita, pulsa sobre ella y aparecerá una ventana de diálogo con la fecha, hora y motivos de la cita, así como si ha sido confirmada o no (Figura A.7). En esta misma ventana, también se ofrece la opción de eliminar la cita si así se desea. Para registrar una nueva cita en la agenda, pulsa sobre un hueco libre, que llevará a un formulario de nueva cita (Figura A.8), donde la fecha, hora de inicio y hora de fin que aparecen corresponden con los del hueco libre pulsado, pero son totalmente modificables pulsando sobre ellas.

Figura A.4. Sección "Agenda"

Figura A.5. Selección de día

Figura A.6. Horario de citas

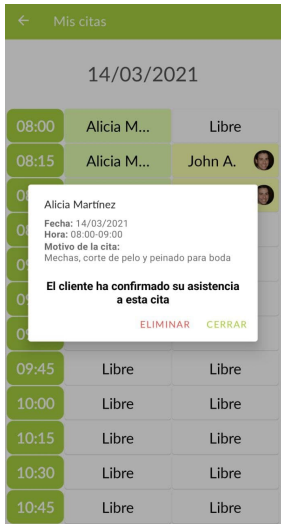

Figura A.7. Detalles de una cita

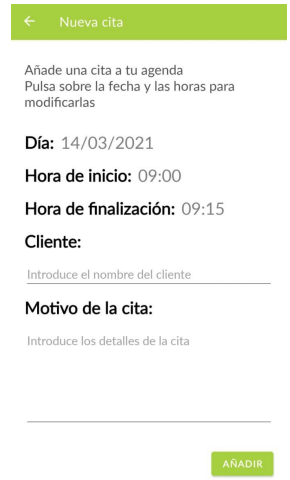

Figura A.8. Crear nueva cita

Si desde la pantalla inicial se accede a la **agenda de pedidos** y se selecciona un día en el calendario, aparecerá un horario (Figura A.9) indicando el número de pedidos por franja horaria. Pulsando en una franja, accederemos a la lista de pedidos para esa franja (Figura A.10), que de manera similar a las citas, tendrán la foto del cliente si ha sido solicitado a través de la app. Para eliminar un pedido se puede utilizar el botón "Eliminar" de cada una de las tarjetas de pedido. Para añadir un nuevo pedido, pulsa sobre el botón flotante con un

"+" en la parte inferior derecha, que conducirá a un formulario (Figura A.11) en el que, al igual que las citas, se puede personalizar la fecha y hora.

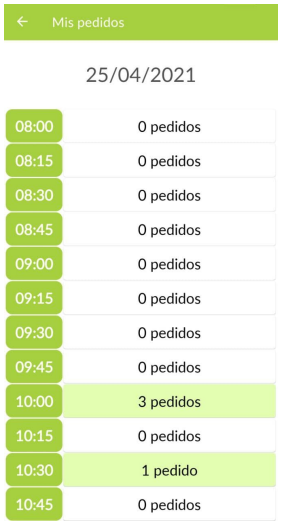

Figura A.9. Horario de pedidos

Figura A.10. Lista de pedidos

Figura A.11. Crear nuevo pedido

Para ver las **solicitudes de cita o pedido pendientes**, se accede al menú lateral y, en función de si tu establecimiento ofrece citas y/o pedidos, aparecerá sólo una de las dos secciones, o ambas (Figura A.12). En ambas secciones el comportamiento es similar. Accediendo al menú de solicitudes de cita, podremos ver las tarjetas con las solicitudes de cita pendientes por responder (Figura A.13). Viendo los detalles, podremos o bien aceptarlas o bien rechazarlas, pudiendo enviar una respuesta al cliente en este último caso (Figura A.14). En la sección de solicitudes de pedido, el funcionamiento es el mismo.

En la **sección "Mapa"** (Figura A.15) podemos acceder a un mapa interactivo, en el que se indican las ubicaciones de los establecimientos registrados en la app. Si pulsamos sobre uno de los marcadores del mapa, aparecerá el nombre y número de contacto del establecimiento. Además, pulsando sobre los botones que aparecen en la esquina inferior derecha, se nos redirigirá a la app *Google Maps* para ver más detalles de la ubicación, o bien, encontrar una ruta para llegar.

En la **sección "Tablón"** (Figura A.16) podremos ver los anuncios publicados por nuestro establecimiento y los otros registrados en la app, por orden cronológico (más recientes primero). Para crear un nuevo anuncio que aparezca en el tablón, pulsa sobre el botón flotante con un "+" en la parte inferior derecha, que te llevará a un formulario de nuevo anuncio (Figura A.17), que una vez completado y enviado, hará que aparezca tu nuevo anuncio en el tablón.

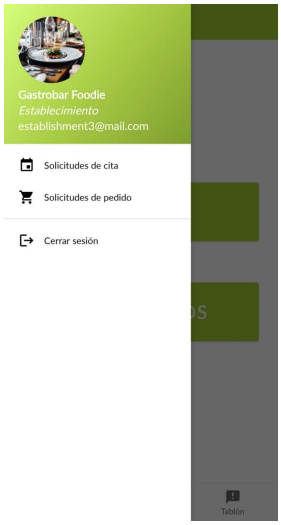

Figura A.12. Menú lateral de dueño de negocio

Figura A.13. Solicitudes de cita

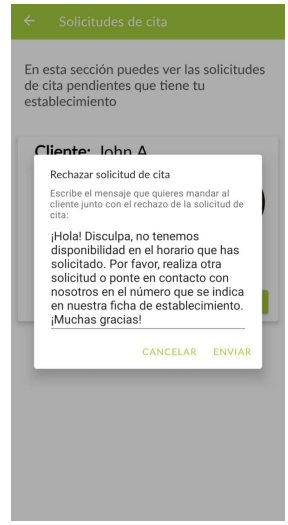

Figura A.14. Rechazo de solicitud de cita

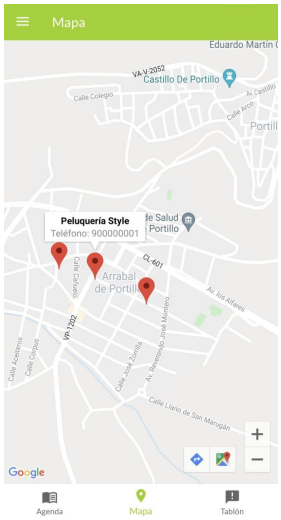

Figura A.15. Sección "Mapa"

Figura A.16. Sección "Tablón"

Figura A.17. Crear nuevo anuncio

Cliente

La pantalla principal cuando se inicia sesión como dueño de negocio es la **sección "Sitios"** (Figura A.18). En esta sección, aparecerá la lista de establecimientos registrados en la app, a los que podemos realizar solicitudes de cita y pedido. Para ello, se selecciona uno de los mostrados en la lista, que nos llevará a la pantalla de su ficha de establecimiento, donde se muestran sus detalles (Figura A.19). En esta pantalla, se puede ver qué opciones de solicitud existen para dicho establecimiento. Si alguno de los botones aparece deshabilitado, entonces el establecimiento no ofrece ese servicio a través de la app.

Si seleccionamos **pedir cita**, se nos conducirá a una pantalla de calendario para seleccionar día (similar a Figura A.5), y posteriormente podremos ver el horario con los huecos ocupados y libres para ese día (Figura A.20). El número de columnas indica el máximo de clientes que el negocio admite por franja horaria de 15 minutos. Para solicitar una cita, pulsamos sobre un hueco vacío, y se nos llevará a un formulario de solicitud de cita (similar a Figura A.8), con parámetros también modificables si se pulsa sobre ellos. Una vez completado, la solicitud se envía al establecimiento, a la espera de que sea respondida.

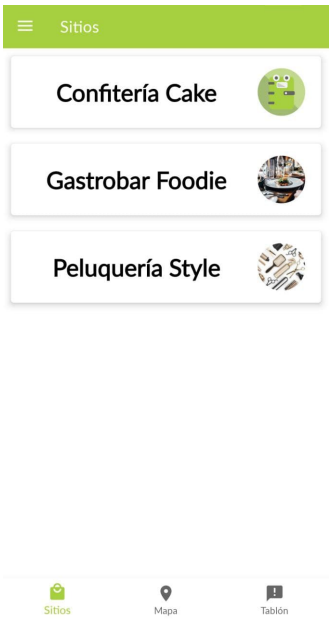

Figura A.18. Sección "Sitios"

Figura A.19. Ficha de establecimiento

Figura A.20. Solicitar una cita

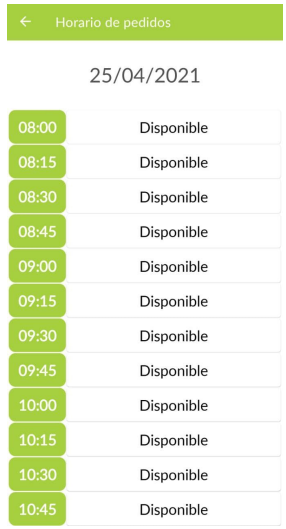

Figura A.21. Solicitar un pedido

Si seleccionamos **hacer pedido**, tras seleccionar en un calendario el día deseado para el pedido, se mostrará un horario con las horas disponibles para hacer pedidos ese día (Figura A.21). Pulsando sobre la hora deseada, se mostrará un formulario de solicitud de pedido (similar a Figura A.11), que una vez completado quedará a la espera de la respuesta por parte del establecimiento.

Para poder ver las **citas y pedidos solicitados**, tenemos que acceder al menú lateral, donde aparecerán estas dos secciones (Figura A.22).

En el caso de citas solicitadas, nos aparecerán todas las solicitudes de cita que hemos realizado, su estado y posible respuesta del establecimiento (Figura A.23). También podremos confirmar nuestra asistencia a las citas que ya hayan sido aceptadas por el establecimiento mediante el botón que aparece en las citas que estén en este estado.

En el caso de pedidos solicitados, de manera similar a las citas solicitadas, nos aparecerán todas las solicitudes de pedido realizadas a través de la app, indicando sus detalles, estado, y si hay, respuesta de rechazo del establecimiento (Figura A.24).

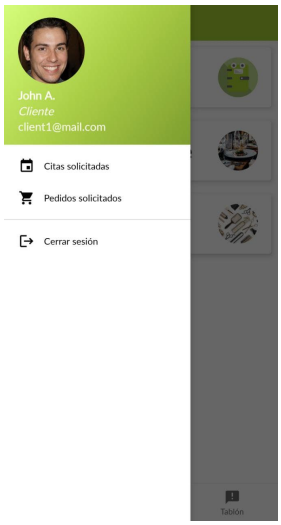

Figura A.22. Menú lateral de cliente

Figura A.23. Citas solicitadas

Figura A.24. Pedidos solicitados

En la **sección "Mapa"** podemos acceder a un mapa interactivo en el que se indican las ubicaciones de los establecimientos registrados en la app. De la misma manera que para los dueños de negocio, podremos redirigirnos a *Google Maps* para obtener más detalles de la ubicación y ver distintas rutas para llegar hacia los distintos establecimientos (Figura A.15).

Finalmente, en la **sección "Tablón"**, podremos consultar los anuncios publicados por los establecimientos, ordenados cronológicamente, mostrándose más arriba los más recientes (Figura A.16).

Apéndice B

Enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código de la app y su diseño en un archivo .asta: `https://gitlab.inf.uva.es/juagarr/tfg-juancarlosgarrote`
- Repositorio con el archivo .apk y el manual de usuario de la app: `https://github.com/JuancaG05/AppGenda`