UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

APP móvil Fitocampo

Autor/es

David Íñiguez Comps

Director/es

LAUREANO LAMBAN PARDO

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2021-22

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

App móvil Fitocampo

Realizado por:

David Íñiguez Comps

Tutelado por:

Laureano Lambán Pardo

Logroño, julio, 2022

Resumen

Este Trabajo de Fin de Grado se corresponde con un proyecto de la empresa SDi Digital Group, la cual posee un sistema de gestión de trabajo de campo en el sector fitosanitario. La empresa observó que era necesario crear una *app* para acercar estos servicios al trabajador.

El proyecto ha consistido en desarrollar desde cero una aplicación que permita a los trabajadores registrar sus visitas y actuaciones sobre las fincas y conectar todo eso con la base de datos de SDi para todas las empresas del sector. La *app* hace uso de dos APIs y se desarrolla a partir de un diseño, todo esto proporcionado por la empresa. Empleando la tecnología de desarrollo móvil híbrido, se crea un código fuente que puede ser exportado de forma sencilla tanto a Android como a iOS.

Así, el objetivo del proyecto fue crear la mencionada aplicación, que permita con un alto grado de usabilidad para los trabajadores de campo, registrar sus visitas, prescripciones y pedidos. Para ello, simplemente han de seleccionar en los distintos pasos de cado proceso entre las opciones disponibles, las cuales se ajustan a la empresa concreta en la que trabajan, para mayor comodidad. Por último, se añade la posibilidad de exportar las prescripciones a PDF para generar documentos que presentar a su empresa.

Abstract

This Final Degree Project corresponds to a project of the company SDi Digital Group, which has a fieldwork management system in the phytosanitary sector. The company realized that it was necessary to create an *app* to bring these services closer to the worker.

The project consisted of developing from scratch an application that allows workers to record their visits and actions on the farms and connect all this with the SDi database for all companies in the sector. The *app* makes use of two APIs and is developed from a design, all provided by the company. Using hybrid mobile development technology, a source code is created that can be easily exported to both Android and iOS.

Then the goal of the project was to create the aforementioned application, which allows field workers to record their visits, prescriptions and orders with a high degree of usability. To do this, they simply have to select in the different steps of each process among the available options, which are adjusted to the specific company in which they work, for convenience. Finally, there is the possibility of exporting the prescriptions to PDF to generate documents to present to their company.

Índice

1 Introducción

Esta sección contiene la información previa al inicio de la planificación, análisis y desarrollo de la aplicación. Pretende aclarar el contexto, características y objetivos de la *app* que se pretende desarrollar, así como las tecnologías y tipo de metodología a emplear en su desarrollo.

1.1 Entorno

Este proyecto es una propuesta de SDi [1] (empresa donde he realizado mis prácticas curriculares), para sus clientes del mundo de la agricultura, más concretamente del sector de servicios fitosanitarios. Surgió para facilitar la creación y distribución de informes acerca de los productos a utilizar por los empleados en las diferentes parcelas. La idea inicial es proporcionar una *app* que permita la creación de informes de peritación y de actuación sobre las fincas de forma rápida y sencilla.

SDi es una empresa que apuesta por las soluciones escalables, que sean útiles para más de un ámbito o empresa. Soluciones generales, pero que se adapten correctamente a las necesidades concretas de cada cliente. Esta aplicación seguirá la misma filosofía.

En este caso se trata de un proyecto con una orientación concreta, creado únicamente para el sector fitosanitario. *FitocampoApp* sirve para cualquier cliente en contrato con la empresa, y que por tanto tenga conectada su base de datos con la API interna producida por SDi para este servicio (la cual no es competencia de este proyecto, pero se detallará brevemente en un apartado posterior).

La *app* pretende servir por tanto para todas las empresas fitosanitarias que tengan relación con SDi. La idea es crear una única aplicación que ofrezca un servicio personalizado y particular tanto a cada una de las distintas empresas que tienen contrato actualmente, como las que puedan unirse en un futuro.

Además, esta *app* se crea con la posibilidad de ser útil en otros ámbitos con modificaciones menores en su estructura. Por ejemplo, con algún cambio referente a los datos y campos de los menús, es una *app* que puede funcionar perfectamente para compañías de seguros, facilitando mucho la comunicación de peritos con los profesionales de la construcción o restauración de desperfectos habituales como fontaneros o electricistas.

Esta *app* pretende suplir necesidades relevantes y facilitar en gran medida el trabajo de estos empleados del sector fitosanitario. Además, se buscará conseguir una buena usabilidad y comodidad para el usuario.

1.2 Ámbito

1.2.1 Tipo de app

FitocampoApp pretende ser una aplicación móvil para la creación de informes tanto de visita como de prescripción para un tratamiento fitosanitario. Tendrá un servicio de login que permitirá identificarse a los empleados de las distintas empresas fitosanitarias, y una sección para cada función (visita, prescripción...). Estas secciones permitirán escoger entre la información disponible por pasos y añadir lo necesario para generar el informe correspondiente en el caso de la prescripción, el cual se puede enviar, firmar o descargar.

Estará para la plataforma Android, aunque la exportación a iOS será también posible y sencilla.

1.2.2 Estudio de mercado. Alternativas

Para comprobar la viabilidad de este producto, he realizado un estudio de mercado en busca de aplicaciones que ofrezcan el mismo servicio actualmente.

Como resultado de esta investigación se han localizado ciertas *apps* que ofrecen este servicio o similar de distintas formas.

La primera es *Fitosanitarios [2]*, de *AGER TECHNOLOGY SL.* Esta *app* permite crear registros de productos fitosanitarios, y generar un archivo PDF con la información. Pero no permite realizar informes de visitas, ni tiene login, ni está, en general, enfocada al mismo objetivo que tiene *FitocampoApp.* Además, tiene un diseño algo anticuado y parece ser que no se encuentra disponible actualmente para Android ni algunas versiones de iOS. Por tanto, esta aplicación no debería suponer un obstáculo en el mercado para la nuestra.

Hay otras alternativas que agrupo por ser similares: *TankCalc [3]* de *Syngenta*, la *app* de gestión de inventario fitosanitario de *VisionAgro [4]*, y la *app* de *aGROSlab Agricultor [5]* de *aGROSlab*. Las dos primeras parecen de baja calidad, y la segunda no está ni en PlayStore ni AppStore. Ambas tienen una interfaz anticuada, y no cubren las mismas necesidades que buscamos. Por tanto, no se tienen en cuenta. La tercera, en cambio, realiza una función muy similar a la que pretende cubrir FitocampoApp, además parece funcionar correctamente y añadir el tratamiento de forma fácil (de cero o a partir de una prescripción), generar prescripciones, vista de mapa de las fincas y llevar un diario de tratamientos. Aun así, FitocampoApp sigue siendo viable ya que se enfoca a aligerar el trabajo dentro de las empresas, ofrece firma y generación de archivo PDF, lo que esta no permite, y una interfaz personalizada para cada empresa, además de resultar mucho más atractiva, cómoda para el usuario y moderna.

Otra aplicación es *PlantCare Pro [6]* de *XG Soft.* Esta aplicación dispone de una base de datos muy completa sobre cultivos, plagas, productos y fabricantes. Permite localizar el producto concreto necesario para tratar una plaga específica y generar el archivo PDF pertinente. Está más enfocada a información que a un uso laboral, sólo se encuentra disponible en iOS y posee un diseño más enfocado a iPad que a móvil con una interfaz poco atractiva y algo anticuada. Por estas razones, pese a que es una *app* competente, no cubre las necesidades que se piensan satisfacer con este proyecto luego podemos seguir adelante con el mismo.

Finalmente, como última alternativa relevante a tener en cuenta está *FitosGEST [7]* de *locatec.es*. La *app* no se encuentra en las tiendas de *apps,* sino que se debe solicitar una demo, u obtenerla por medio de su página web. Tiene versión de PC, Tablet y móvil. No ofrecen información relativa a la plataforma, y dispone de tres planes de compra del servicio alrededor de 500, 1000 y 1800 euros. Permite extraer información de los PDF de los productos fitosanitarios del Ministerio de Agricultura, los registros no vigentes, controlar los tratamientos, envío de prescripciones por mail o WhatsApp, dictado por voz en la sección de visitas, mapas, calendarios... Es una aplicación muy completa y que ofrece una gran variedad de servicios, con lo cual es una alternativa bastante relevante.

Una vez hecho el análisis de mercado, se aprecia que solo hay una aplicación realmente potente, pero considero que sigue teniendo sentido el desarrollar *FitocampoApp*, ya que trabaja expresamente con las empresas y ofrece un servicio más enfocado, con sus productos concretos. Esto ofrece una experiencia más cómoda y directa, sin que pueda abrumar con demasiada información u opciones. Genera de forma rápida e intuitiva los informes ofreciendo un procedimiento en definitiva más personalizado. Considero también que en el apartado estético y de diseño *FitocampoApp* puede destacar frente al resto de alternativas, ya que se plantea con un manejo mucho más moderno y sencillo de usar. Además, será escalable y sus funciones podrían ir aumentando en un futuro conforme se soliciten por parte de los clientes, o se consideren rentables.

Este proyecto busca cubrir necesidades concretas, las más básicas, lo que permite a los clientes pagar sólo por lo que van a usar, sin obtener con el paquete funciones que no van a rentabilizar.

En definitiva, del análisis se concluye que el proyecto es viable por su enfoque concreto, procurando un buen diseño y funciones de utilidad, y con buena proyección a futuro por su posibilidad de ampliación, o capacidad de emplear la estructura para producir aplicaciones que cubran esa misma función o similar en otros ámbitos (por ejemplo, compañías de seguros).

1.2.3 Conveniencia de uso

El uso de una aplicación como *FitocampoApp* para una empresa de productos fitosanitarios resulta muy conveniente ya que ofrece una forma cómoda de realizar un trabajo algo pesado, ofreciendo un soporte que permite elaborar los informes sobre la marcha, en el propio campo y sin ofrecer más información de la necesaria lo que agiliza la elaboración de los mismos. Además, al ser una aplicación móvil, es una herramienta que siempre irá con los empleados, y al no ocupar mucho espacio de almacenamiento no supone ningún problema en ese sentido, y es usable por cualquiera en cualquier momento.

Esto agilizará el proceso de redacción y paso de informes, lo que puede llegar a permitir a los empleados hacer más visitas o tratamientos cada día ya que se les libera tiempo que dedicaban a redactar los distintos ficheros.

1.3 Objetivos

Teniendo en cuenta las conclusiones anteriores, he decidido desarrollar esta aplicación y centrarme fundamentalmente en hacerla escalable y extrapolable (mediante un código general que tenga posibilidad de adaptarse a otros usos), y con un buen diseño de interfaz de usuario. Con esto se conseguiría diferenciar bien la aplicación de las que hay actualmente en el mercado, y ofrecer grandes ventajas tanto a los clientes como a la propia SDi, ya que este proyecto puede seguir creciendo, como servir de base para desarrollar alternativas similares para otros sectores de forma mucho más rápida.

FitocampoApp debe permitir a los empleados generar sus informes de prescripciones y visitas de forma rápida y sencilla, mediante selección de opciones, mostrándoles solo las correspondientes a la empresa donde trabajan. También debe, como ya he mencionado, mostrar una interfaz de usuario limpia, moderna, atractiva e intuitiva.

En cuanto al aspecto de la programación, debe ser un código ordenado y bien estructurado, empleando componentes y páginas lo más generales posibles, pensando siempre en la posibilidad de añadir nuevos en un futuro, y la de emplearlos para otro fin. Para conseguir esto la estructura de los componentes debe basarse en una aplicación para generar informes, no en una para generar informes fitosanitarios.

La aplicación dispondrá de un sistema de login, que permitirá a los empleados acceder a las funcionalidades de la aplicación, y mediante el cual se pueden obtener los datos de estos, para mostrarles la vista adaptada a su empresa en particular.

Una vez dentro se ha de mostrar un menú principal donde seleccionar la explotación y el año, y donde se muestren las diversas opciones disponibles en la *app* (prescripción, visita...). Desde esta página también se podrá acceder a un histórico de los informes realizados.

En cuanto a los procesos de visita y prescripción, serán similares e irán por pasos, guiando al usuario dejando marcados las etapas del proceso completadas. Deberá permitir indicar de forma fácil

seleccionándolas desde un filtro la variedad concreta de cultivo, la parcela que se trata y la plaga en caso de la visita o el aplicador responsable y el equipo en el caso de la prescripción.

Una vez hecho eso en el caso de la prescripción debe poder seleccionar la cantidad de producto, la superficie donde se aplica y qué producto, fertilizante o alternativa no química, se va a emplear.

Finalmente, FitocampoApp ofrecerá la posibilidad de, una vez terminados estos pasos y generado el informe, descargarlo en formato PDF.

1.4 Tecnologías

Para desarrollar este proyecto, haré uso de las siguientes tecnologías:

- Ionic [8]: se trata de una tecnología de código abierto que permite el desarrollo de aplicaciones móviles híbridas. Estas se caracterizan por ser creadas de forma similar a una página web, y luego se exportan a iOS, Android o una aplicación web como tal. Permite crear una aplicación móvil para cada plataforma partiendo de una misma base de código. Se basa en Angular [9] y emplea HTML [10] y CSS [11] para la parte visible, y TypeScript [12] para la lógica. También permite implementar una gran variedad de *plugins* para hacer uso de las funciones nativas del teléfono (cámara, email...).
- Angular: es el *framework* de *frontend* del que hace uso Ionic. Es gratuito, de código abierto y permite el desarrollo multiplataforma. Funciona de forma reactiva, lo que permite mejor rendimiento e interacción con el usuario.
- HTML: es el lenguaje de marcas usado en páginas web para estructurar el contenido de estas.
- CSS: lenguaje de estilos que permite definir el aspecto de la aplicación.
- TypeScript: lenguaje ampliación de JavaScript que permite implementar la lógica de las aplicaciones programadas en Ionic mediante contenido dinámico.
- Bootstrap [13]: *framework* de CSS que facilita el crear interfaces adaptables al tamaño de pantalla (responsive) y mejorar su usabilidad. Es de código abierto.
- Capacitor [14]: tecnología que permite exportar un proyecto Ionic a un proyecto de Android o iOS.
- Visual Studio Code [15]: IDE empleado para el desarrollo del proyecto de Ionic.
- AdobeXD [16]: programa de diseño gráfico donde he ido consultando el diseño de la aplicación.
- Odoo [17]: herramienta de gestión de proyectos usada en SDi que permite llevar el control de las horas dedicadas a cada tarea, así como tener una visión general y ordenada de las mismas. Posee una lista Kanban para comprobar de forma sencilla en todo momento el estado de las distintas tareas.
- Word [18]: editor de textos de Microsoft empleado para elaborar esta misma memoria.
- Outlook [19]: gestor de correo electrónico de Microsoft usado para la comunicación dentro de SDi, así como con el tutor académico.
- GitLab [20]: servicio de control de versiones utilizado en SDi, por tanto, es el que he empleado para tener un repositorio con el proyecto. Esto me permite guardar de forma segura todos los cambios y la posibilidad de volver a una versión anterior si fuera necesario.
- Laravel [21]: tecnología empleada para construir la API. No es competencia de este proyecto, pero de ahí es donde se obtienen los datos que pueblan la *app*.
- PostgreSQL [22]: es la tecnología en la que está desplegada la base de datos, tampoco es parte de este proyecto, pero es importante mencionar que ahí se almacenan los datos que se usan.

1.5 Metodología

De entre todas las metodologías existentes para el desarrollo de un proyecto informático, me he decidido por una variación de Scrum [23]. Hay varias razones para esto. La primera es que es la metodología empleada en SDi, luego resulta más sencillo y útil adaptarse a ella. Además, ya tengo conocimiento de la misma porque es parte del temario del Grado en Ingeniería Informática. Otra razón es porque una metodología ágil es muy conveniente para este tipo de proyectos, ya que se va desarrollando añadiendo cosas al proyecto y es susceptible a variaciones desde el inicio. También conviene tener siempre una versión relativamente funcional para poder ir mostrando cómo va el desarrollo y hacer pruebas.

La metodología Scrum se basa en iteraciones, llamadas *sprints*, en este caso de 2 o 3 semanas cada uno. Esto se hace para que al final de cada sprint tengamos un producto funcional con las características implementadas hasta el momento.

Dentro de un *sprint* se distinguen distintas fases. La primera es la planificación (*Sprint Planning*), donde elegimos qué tareas del *Product Backlog* (total de tareas del proyecto) se harán en ese sprint en concreto (formando el *Sprint Backlog*). Algo imprescindible en un proyecto de informática es el seguimiento del mismo, y para ello Scrum incluye en cada sprint reuniones para la planificación (*Sprint Planning*), seguimiento diario (*Dailys*) y comprobar los resultados (*Sprint Review*). Al estar empleando esta metodología en un proyecto realizado únicamente por una persona, se omiten ciertas reuniones pensadas para trabajos en equipo como las diarias o la retrospectiva y no se harán reuniones como tal, sino unas fases de reflexión o trabajo acerca de lo que se trataría en la reunión correspondiente. Se mantiene eso sí una sesión de planificación al inicio del sprint y una de revisión de resultados al final, donde veré qué he conseguido respecto al plan, y qué debo incluir en el siguiente sprint cuando toque planificarlo.

Esas sesiones pueden convertirse en reuniones si en algún caso veo conveniente contar con la opinión o ayuda del tutor de la empresa. Como norma general será trabajo individual.

El uso de esta metodología facilita el añadir nuevas características a la aplicación si el desarrollo transcurriese en menos tiempo del previsto, bastaría con añadir las tareas al *Product Backlog* y planificar un sprint extra. Además, permite adaptarse a los cambios, mover tareas, o finalizarlas de forma controlada en otro sprint si requieren más tiempo del previsto.

Por otro lado, al ser una metodología ágil iterativa e incremental, posee una característica que implica que al final de cada iteración se posee un prototipo o versión del producto con la funcionalidad que haya sido implementada hasta el momento. Esto es muy útil para mantener actualizados a los tutores (académico y de prácticas) del avance del proyecto de una forma muy gráfica, rápida y sencilla. Esto hará más ágiles las reuniones, mejora el entendimiento y permite tener una visión temprana relativamente clara de lo que será el resultado final.

2 Plan de proyecto

Realizaré una planificación general del proyecto al inicio de este. El objetivo es obtener una idea del tiempo a dedicar a cada sección del trabajo, e incluir una previsión de los riesgos que puedan hacer que me desvíe del tiempo previsto.

Se usarán ciertas técnicas del PMBOK [22] (Guía de los Fundamentos para la Dirección de Proyectos, del *Project Management Institute*) que se han practicado en algunas asignaturas del grado, como pueden ser la EDT (estructura de descomposición del trabajo), una tabla de estimación de horas, un diagrama de Gantt y la mencionada sección de posibles riesgos.

2.1 Tareas (EDT)

Emplearé la EDT para descomponer el trabajo en paquetes de trabajo, donde se puede ver más claramente a cuál de ellos pertenecerá cada tarea. Este diagrama está orientado principalmente a los entregables. En el caso de este TFG, se consideran entregables el código de FitocampoApp, el *apk* generada y esta misma memoria. He decidido descomponer el proyecto en los siguientes bloques: gestión inicial, gestión continua, desarrollo y defensa.

La gestión inicial comprende aquellas tareas que solo se realizan al principio del proyecto. Este bloque está formado por la planificación, la instalación inicial, la captura de requisitos y el *product backlog*. A continuación, está el bloque de gestión continua, que comprende los aspectos de gestión que se realizan a lo largo de todo el tiempo de vida del proyecto, véase esta misma memoria, la investigación que haya que ir realizando y el seguimiento y control. Después, el bloque de desarrollo, conformado en este caso por 5 sprints, y el bloque de defensa, que contempla la creación de la presentación y los ensayos de la exposición.

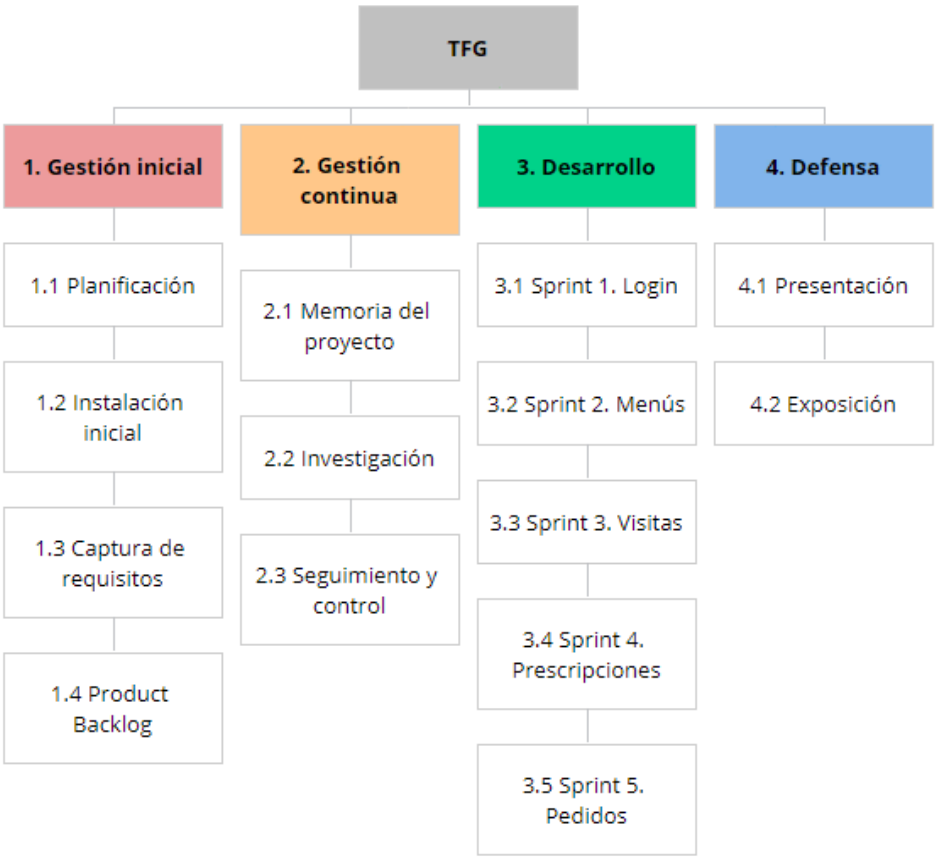


Figura 1 – EDT del proyecto

2.2 Estimación temporal

En la siguiente tabla se representa la estimación inicial de tiempos para las distintas tareas antes mencionadas como parte de cada paquete de trabajo. La estimación inicial la haré suponiendo que emplearé 300 horas en la realización del total del proyecto. La estimación de tiempos en las tareas de cada sprint se detallará en sus respectivas planificaciones.

Código	Nombre	Descripción	Tiempo estimado
1	**Gestión inicial**		**12h**
1.1	Planificación	Dividir el proyecto en diferentes tareas, planificar todos los aspectos de la realización del mismo y estimar tiempos.	6h
1.2	Instalación inicial	Instalar y preparar todos los programas y software necesario para la realización del proyecto.	1h
1.3	Captura de requisitos	Recogida de los requisitos funcionales y no funcionales.	2h
1.4	Product Backlog	Elaboración del Product Backlog a partir de los requisitos.	3h
2	**Gestión continua**		**68h**
2.1	Memoria del proyecto	Redacción de la memoria que documente todo el proceso de trabajo en el proyecto.	54h
2.2	Investigación	Búsqueda de información necesaria para el desarrollo de cualquier fase del proyecto.	6h
2.3	Seguimiento y control	Reuniones para comprobar avance, revisiones al finalizar los sprints y correcciones si fueran necesarias debido a desviaciones de tiempo.	8h
3	**Desarrollo**		**200h**
3.1	Sprint 1. Login	Planificación e implementación del primer sprint. Pantalla de login y estructura básica de la *app*.	20h
3.2	Sprint 2. Menús	Planificación e implementación del segundo sprint. Menú principal, secundario y de historial.	20h
3.3	Sprint 3. Visitas	Planificación e implementación del tercer sprint. Proceso de visitas.	30h
3.4	Sprint 4. Prescripciones	Planificación e implementación del cuarto sprint. Procesos de prescripciones.	80h
3.5	Sprint 5. Pedidos	Planificación e implementación del quinto sprint. Proceso de pedidos y mejoras generales.	50h
4	**Defensa**		**20h**
4.1	Presentación	Elaboración de la presentación que acompañará la defensa del TFG.	10h
4.2	Exposición	Preparación y ensayo de la propia defensa.	10h
Totalidad del TFG		**Horas totales empleadas**	**300h**

Tabla 1 – Estimación temporal

2.3 Diagrama de Gantt

Diagrama de Gantt - 1										
		Febrero		Marzo				Abril		
		21-25	28-04	7-11	14-18	21-25	28-1	4-8	11-15	
1.1	Planificación									
1.2	Instalación inicial									
1.3	Requisitos									
1.4	Product backlog									
2.1	Memoria									
2.2	Investigación									
2.3	Seguimiento									
3.1	Sprint 1									
3.2	Sprint 2									
3.3	Sprint 3									
3.4	Sprint 4									

Tabla 2.1 – Parte 1 del diagrama de Gantt. Meses de febrero, marzo y abril.

Diagrama de Gantt - 2										
		Abril		Mayo					Junio	
		18-22	25-29	2-6	9-13	16-20	23-27	30-3	6-10	
2.1	Memoria									
2.2	Investigación									
2.3	Seguimiento									
3.4	Sprint 4									
3.5	Sprint 5									

Tabla 2.2 – Parte 2 del diagrama de Gantt. Meses de abril, mayo y junio.

Diagrama de Gantt - 3								
		Junio			Julio			
		13-17	20-24	27-1	4-8	11-15	18-22	25-29
2.1	Memoria							
2.2	Investigación							
2.3	Seguimiento							
4.1	Presentación							
4.2	Exposición							

Tabla 2.3 – Parte 3 del diagrama de Gantt. Meses de junio y julio.

2.4 Riesgos

En esta sección recopilaré la información necesaria de algunos posibles riesgos, junto con las medidas que podría adoptar para prevenirlos y solucionarlos en caso de que se den.

El primer riesgo sería la pérdida del almacenamiento en el PC: datos, código, ficheros... Esto tendría un impacto medio debido a las medidas ya tomadas, aunque la probabilidad de que pase es baja. La causa de esto podría ser un fallo crítico en el ordenador, o algún problema con un corte de luz. La forma de prevenirlo es tener todo en la nube, con copias de seguridad, cosa que hacemos desde el principio con GitLab, Drive y Microsoft 365. Para solucionarlo simplemente se recuperarían las copias de seguridad.

Otro podría ser la falta de disponibilidad de alguien a consultar. El impacto en general es bajo, pero depende del momento puede variar. En este caso la probabilidad de que suceda es bastante alta ya

que el tutor académico y sobre todo el de la empresa suelen estar ocupados. La forma de prevenir esto es tener trabajo con el que poder seguir avanzando mientras espero, y la medida a tomar en caso de que suceda sería hacer ese trabajo y elaborar una lista de dudas para que la consulta sea lo más ágil posible y sin olvidar nada.

Finalmente hay que tener otro riesgo en cuenta, que serían los cambios en los requisitos. Esto tiene rango medio tanto en impacto como en probabilidad. Se debe a que el cliente quiera en un momento dado durante el desarrollo del proyecto, introducir nuevas funcionalidades o cambiar alguna característica ya acordada. También puede producirse porque el desarrollo del proyecto sea rápido y decidamos añadir nuevas funcionalidades de los requisitos opcionales. La medida de prevención sería simplemente trabajar de forma rápida para disponer de más tiempo, y la solución en caso de suceder es aprovechar el tiempo del que disponga para estas nuevas implementaciones, o denegar los cambios si no diera tiempo a realizarlos.

3 Análisis

3.1 Requisitos

El primer paso para definir el alcance del proyecto y analizar las tareas que habrá que realizar es recoger los requisitos. Una buena captura de requisitos es fundamental para que el producto final se corresponda justamente con lo que quería el cliente o con las expectativas iniciales. También ayuda a no olvidar ningún aspecto, ni a trabajar de más.

Para poder hacer esta captura de requisitos me reuní con el tutor de la empresa. Definimos así una lista de requisitos obligatorios, ordenados aproximadamente por orden de realización. Todos ellos son imprescindibles para el correcto funcionamiento de la *app*.

Los requisitos opcionales son aquellos que podrían ser mejoras para la aplicación, y que en caso de terminar el desarrollo antes del tiempo previsto, serían los siguientes a abordar.

Finalmente, los requisitos no funcionales se refieren a la estructura de la aplicación y estilo de programación.

3.1.1 Requisitos funcionales

Requisitos obligatorios:

- RFOb1: Tendrá una pantalla de login para poder acceder a sus contenidos.
- RFOb2: Debe estar conectada a la API disponible y hacer uso de la misma para obtener toda la información necesaria.
- RFOb3: Solo se debe mostrar la información relevante para cada usuario, las opciones de los filtros son las que tiene disponibles en su empresa.
- RFOb4: Generar un PDF al terminar el informe.
- RFOb5: Poder mostrar el histórico de informes desde el menú principal.
- RFOb6: Ofrecer un menú principal con acceso a toda la funcionalidad
- RFOb7: Existirá un menú lateral que permite navegar por la *app* desde cualquier pantalla.
- RFOb8: Tendrá una pantalla para visita donde seleccionar diversas opciones relacionadas con la misma y terminar creando una visita. Los campos a completar aquí son: especie, parcela, plaga y observaciones (siendo estos dos últimos opcionales).
- RFOb9: Tendrá diversas vistas para permitir elaborar un informe de prescripción, seleccionando lo necesario a lo largo del proceso. En la primera fase se seleccionan especie, parcela, aplicador y equipo. En la segunda el caldo, superficie, fitosanitarios, fertilizantes y/o alternativas no químicas. En esta última han de mostrarse los datos ya seleccionados en el anterior paso y la fecha de validez.
- RFOb10: Habrá una vista de detalle para cada entrada del historial.
- RFOb11: Incluir la funcionalidad de la sección de pedidos de material fitosanitario.

Requisitos opcionales:

- RFOp1: Incluir servicio de recuperar contraseña

3.1.2 Requisitos no funcionales

- RNF1: Estructurar el proceso de creación de informes en pasos de selección de opciones.
- RNF2: Estructurar bien el código de forma que sea sencillo de escalar y adaptar a otros usos.
- RNF3: Documentar el código (funciones y clases), no con un estándar para generar documentación, sino con el propósito de que quede claro el código para hacer más sencilla su comprensión en un futuro si se vuelve a trabajar con él.

3.2 Product Backlog (Historias de usuario)

Una vez que se han recogido los requisitos, hay que ordenarlos por relevancia y crear las historias de usuario que van a componer el *product backlog*. Estas son un componente muy importante en la metodología de Scrum.

Cada una va a corresponderse con las tareas necesarias para cumplir un determinado requisito. Además, en cada "tarjeta" hay un texto que expone lo que se pretende poder hacer desde el punto de vista de un implicado, como puede ser el usuario, y una explicación de lo que se hará para proporcionárselo. En la parte superior derecha de la "tarjeta" vemos además un número, los puntos de historia, que estarán entre el 1 y el 5 como máximo e indicarán el volumen de tiempo necesario para completarla.

Historia de usuario	
RFOb1- Página y servicio de login	2 puntos
Como usuario, quiero disponer de un sistema de login para que se me identifique únicamente y pueda disfrutar de las funcionalidades ajustadas a mi empresa y mis datos.	
Validación: Lo primero que se muestra al usuario al acceder a la *app* es la pantalla de login, donde podrá introducir sus credenciales (DNI y contraseña) para empezar a utilizarla. Esas credenciales estarán guardadas y el usuario estará registrado en el sistema.	

Historia de usuario	
RFOb2- Conexión y uso de la API	1 punto
Como usuario, quiero que se me muestren todas las opciones de productos fitosanitarios y otros datos relevantes de los que disponga mi empresa.	
Validación: La aplicación tomará los datos de las bases de datos de las empresas mediante una API proporcionada, para hacer uso de estos dispondrá de unos servicios que obtendrán dichos datos para mostrarlos en la *app*.	

Historia de usuario	
RFOb3- Filtros personalizados	3 puntos
Como usuario, quiero que únicamente se me muestren las opciones de las que dispone mi empresa a la hora de seleccionar los productos u otras opciones mediante filtros.	
Validación: En los filtros solo se mostrará aquello de lo que disponga la empresa correspondiente a cada usuario, obteniendo dicha información mediante la API y los datos del login.	

Historia de usuario	
RFOp1- Recuperar contraseña	1 punto
Como usuario, quiero poder recuperar mi contraseña en caso de que se me olvide o haya algún problema.	
Validación: Estará disponible la opción de recuperar contraseña en la pantalla de login, por ese medio el usuario podrá recuperarla.	

Historia de usuario	
RFOb4- Generar PDF del informe	2 puntos
Como usuario, quiero poder obtener un fichero en formato PDF con el informe que he realizado.	
Validación: Al terminar el proceso, el usuario puede obtener descargado en su dispositivo el PDF del informe con solo pulsar un botón.	

Historia de usuario	
RFOb5- Histórico de informes	2 puntos
Como usuario, quiero poder acceder al historial de informes que he realizado, y ver detalles de los mismos.	
Validación: El usuario podrá acceder al historial de informes mediante un botón del menú principal, ahí puede ver todos los que ha realizado y detalles sobre ellos.	

Historia de usuario	
RFOb6- Menú principal	2 puntos
Como usuario, quiero disponer de una página principal desde la que acceder a toda la funcionalidad de la aplicación.	
Validación: La *app* tendrá una vista principal una vez completado el login que será el menú principal, en ella se verán las opciones de procedimientos que permite la *app*, así como el acceso al histórico de informes.	

Historia de usuario	
RFOb7- Menú lateral	1 puntos
Como usuario, quiero tener siempre un menú lateral desde el que navegar a cualquier punto de la *app* cómodamente.	
Validación: El usuario podrá acceder al menú lateral siempre desde la esquina superior derecha.	

Historia de usuario	
RFOb8- Visita	4 puntos
Como usuario, quiero poder crear una visita de forma sencilla seleccionando especies, parcelas y plagas desde listas.	
Validación: El usuario podrá crear la visita seleccionando las opciones que considere de entre los filtros a su disposición para cada campo.	

Historia de usuario	
RFOb9- Prescripción	5 puntos
Como usuario, quiero poder crear un informe de prescripción con todos los datos que son necesarios.	
Validación: El usuario podrá crear un informe de prescripción, en el que figurarán todos los datos que ha ido seleccionando durante el proceso.	

Historia de usuario	
RFOb10- Vista detalle	3 puntos
Como usuario, quiero tener acceso a una vista detalle de la visita, prescripción o pedido que quiera ver del histórico.	
Validación: El usuario podrá acceder a esta vista detalle seleccionando cualquier entrada del histórico.	

Historia de usuario	
RFOb11- Pedidos	3 puntos
Como usuario, quiero acceder a una sección en la que realizar pedidos de material fitosanitario.	
Validación: El usuario accederá a esta sección desde el menú y generará un pedido con el material que seleccione.	

En total suman 28 puntos de historia obligatorios (más 1 opcional). Como contamos con 5 sprints según la planificación, dividiré estas historias buscando tener 5 puntos por sprint. Aun así, en ocasiones es mejor que queden 4 puntos en un sprint pero que todo lo que se haga en el mismo esté relacionado y sirva de base para lo que haya que programar después.

Para hacer una primera idea general de distribución, empezaré con el login y los servicios de la API, después los menús, luego la visita, a continuación, la prescripción y, para terminar, las acciones finales del proceso. Si sobra tiempo, en el *sprint* 5 se tratará la sección de pedidos.

4 Desarrollo

A lo largo de este apartado iré explicando el trabajo realizado en cada uno de los sprints: planificación, desarrollo e implementación de la parte correspondiente de la aplicación y revisión final.

4.1 Sprint 1

En el primero de los sprints, el objetivo es preparar la estructura del proyecto Ionic, programar los servicios de conexión con la API (al menos lo básico y añadir más si fuera necesario en el futuro), hacer el sistema de seguridad del login y programar su interfaz.

4.1.1 Sprint backlog

Vistos cuales son los objetivos de este sprint, es fácil saber qué tareas son las que se han de abarcar aquí.

El sprint ocupará una semana de trabajo en la empresa, que equivale a 20h. Empezará el jueves 3 de marzo y terminará el jueves 10 de marzo. Hay que incluir las tareas del login, API, y hacer la estructura base en el proceso.

En la siguiente tabla se muestran las tareas a realizar junto con las horas previstas para cada una.

Tareas del Sprint 1	
Tarea a realizar	**Horas**
Estructura general	1h
Conexión y uso de la API	3h
Página y servicio de login	10h
Total	**14h**

Tabla 3 -Tareas del sprint 1

4.1.2 Diseño

En este apartado me ocuparé en primer lugar del diseño de la estructura del propio proyecto de Ionic, explicando el porqué del mismo, y después me dedicaré al diseño del funcionamiento del login y de la página visible de este.

En cuanto a la estructura del proyecto, se parte de la configuración que aparece en la Figura 2 y el objetivo es acabar con la de la Figura 3.

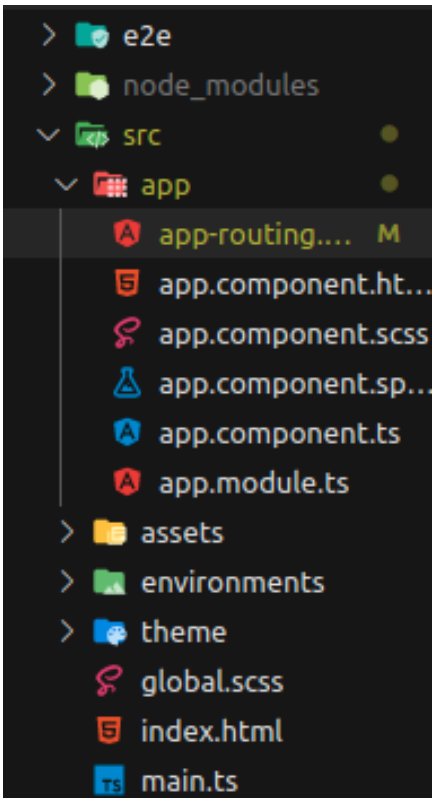


Figura 2 - Estructura inicial del proyecto

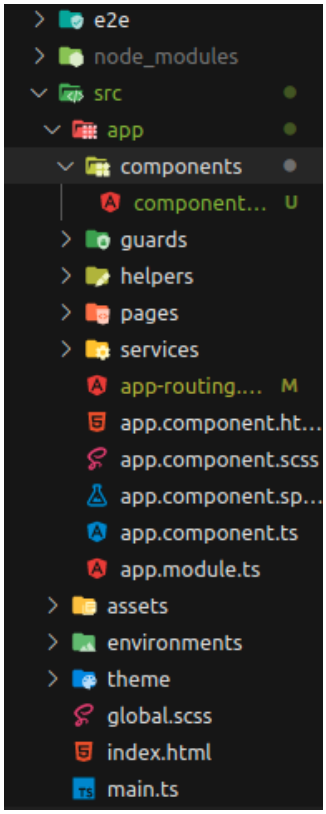


Figura 3 - Estructura final del proyecto

Como se observa en las figuras anteriores, los cambios son esencialmente en la carpeta de aplicación (*app*). La idea de esta estructura es separar las funcionalidades por componentes y los procedimientos que ofrece la *app* por páginas. Con esto se consigue la modularidad que se busca, permitiendo implementar de forma sencilla funciones concretas en distintas vistas, y permitiendo un alto grado de escalabilidad a la aplicación.

Se separan en carpetas los componentes (funciones), páginas (vistas), servicios (conexión a la API) y *guards* y *helpers* (seguridad).

Respecto al diseño de la página de login, he de lograr que se vea así:

Figura 4 – Diseño de la página de login

4.1.3 Implementación

4.1.3.1 Servicio para usar la API para login

Crearé una variable con la parte común a todas las llamadas a la API en el fichero de *enviroment.ts*, con objetivo de hacer más cómoda la programación y facilitar posibles modificaciones en un futuro.

El login será el estándar en cualquier *app*. Se hará mediante un usuario (finalmente será un email y no un DNI) y contraseña. Para el envío de la petición se emplea formato JSON, y después se envían los datos del usuario *loggeado* desde la API. Además, se almacena el token de la sesión, empleado para mantener la seguridad de la aplicación.

4.1.3.2 Guards para la autenticación.

El *guard* se usa para saber si se puede continuar en una redirección. Comprueba si el usuario tiene la sesión iniciada cuando intenta acceder a una página y, en caso contrario, lo redirige al login.

4.1.3.3 Página de login.

El HTML es muy sencillo y mediante el CSS y el framework de Ionic conseguimos que se vea prácticamente igual al diseño original. En cuanto a la funcionalidad, simplemente se aplican unas comprobaciones sencillas a los campos del formulario y la función de login que está en el servicio para ejecutarse al pulsar el botón de "Iniciar sesión".

Se añaden ciertos aspectos que mejoran la experiencia de usuario como mensajes cuando falla la autenticación, y *placeholders* en los campos del formulario.

Al instalar el *plugin* del "*Ionic Storage*" [24] surgieron problemas con la compatibilidad del módulo y no funcionaba. Tras un buen tiempo de búsqueda de soluciones y reinstalaciones encontré que para implementarlo de forma correcta en Ionic con el uso de capacitor, hay que hacer varias instalaciones relacionadas con el plugin y sus adaptaciones. Son una serie de seis instalaciones, que si se hacen en orden funciona correctamente. Estas quedan apuntadas para no perder ese tiempo en futuros usos de este *plugin*.

Finalmente hago la página responsive mediante el uso de media *queries* de CSS y el framework de Ionic.

4.1.4 Comprobación de requisitos

Tareas del sprint 1			
Tarea a realizar	**Horas planeadas**	**Horas reales**	**Desviación**
Estructura general	1h	0.5h	-50%
Conexión y uso de la API	3h	6h	100%
Página y servicio de login	10h	12h	20%
Total	**14h**	**18.5h**	**32,14%**

Tabla 4 – Comparación de tiempos de las tareas del sprint 1

Se ha invertido más tiempo del previsto en la API ya que hubo que hacer configuraciones externas al proyecto en sí, relativas a la configuración de la API. Además, se incluye el tiempo de estudio y pruebas para su correcto uso.

El exceso de tiempo en el login se debe a los problemas con el "Ionic Storage" y por ser la primera implementación del proyecto. Las próximas páginas se harán con un ritmo más ágil.

Cabe mencionar que no se han hecho todos los servicios para uso de la API, sino solo el de *login*. El resto se hará cada uno cuando se necesiten. El apartado de recuperar contraseña por su parte queda aplazado porque aún no hemos conseguido concretar el procedimiento para esta operación.

4.2 Sprint 2

En este segundo sprint me centraré en desarrollar las vistas relacionadas con menús. Esto incluye tanto el menú principal como el lateral, y la vista de histórico.

4.2.1 Sprint backlog

Las tareas a realizar son las correspondientes a los objetivos mencionados antes.

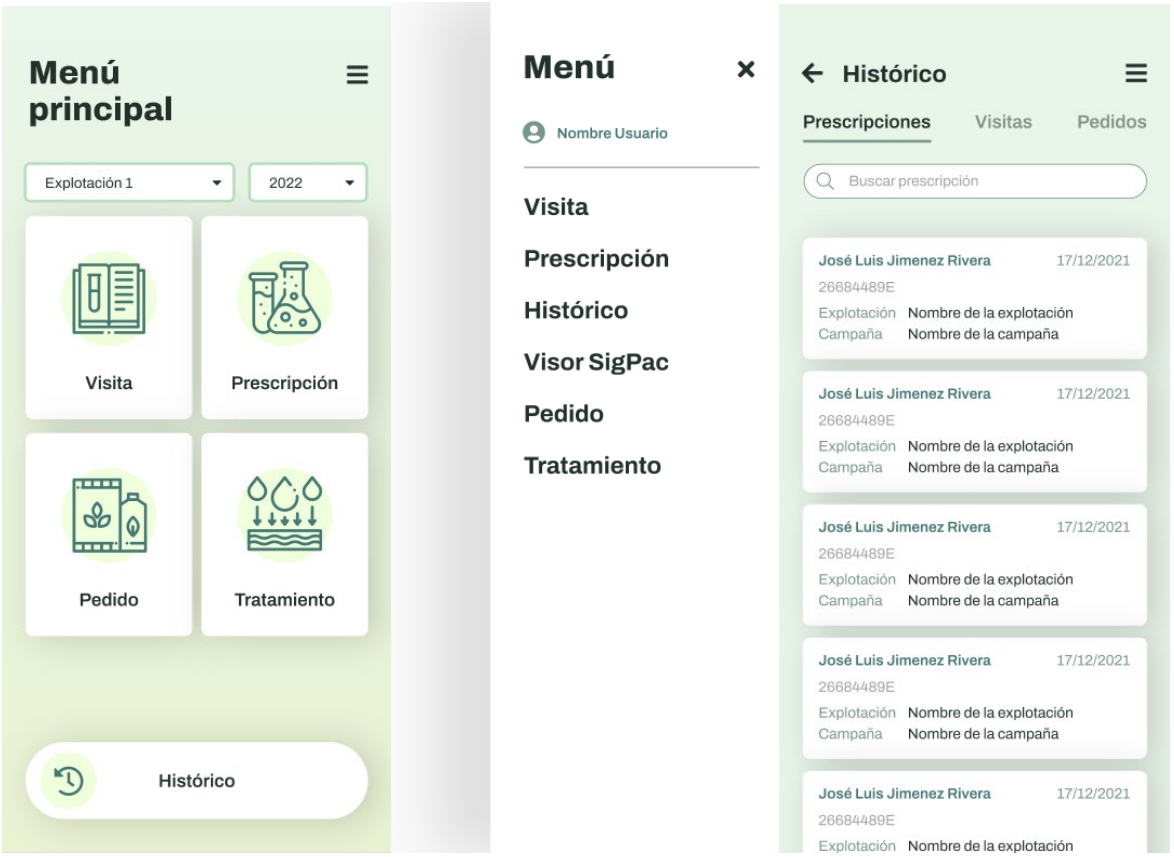
El sprint ocupará una semana de trabajo en la empresa, que equivale a 20h. Empezará el martes 15 de marzo y terminará el lunes 21 de marzo. En la siguiente tabla se muestran las tareas a realizar junto con las horas previstas para cada una.

Tareas del Sprint 2	
Tarea a realizar	**Horas**
Menú principal (home)	3h
Menú lateral	6h
Historial	4h
Servicios de la API	8h
Total	**21h**

Tabla 5 - Tareas del sprint 2

4.2.2 Diseño

En este apartado se ve el diseño de las páginas a realizar en este sprint. El objetivo es que queden lo más similares posibles.



Figuras 5, 6 y 7 - Diseños de las páginas menú principal, menú lateral e histórico

4.2.3 Implementación

4.2.3.1 Menú principal (home)

Tanto en esta como en el resto de páginas de la *app*, usaré el componente *grid* [25] de Ionic. Este permite distribuir bien los elementos por la página y asegura que sea responsive.

Otra cosa que se usará en todas las páginas y que se introduce ahora es el *header*. Es un componente que se colocará en todas las páginas y que es diferente si es el de menú principal o cualquier otro.

La implementación de esta página es sencilla, tan solo usar el *grid* para distribuir los elementos y enlazarlos con su parte correspondiente. En lógica se obtienen las explotaciones y campañas correspondientes para los *select* de la parte superior.

El apartado de tratamiento no lleva a ningún sitio, simplemente es una función que podría añadirse en un futuro.

4.2.3.2 Menú lateral

Implementado de forma sencilla en el *app.component*, con lo que el menú lateral es accesible desde cualquier punto de la aplicación, y a partir de él se puede navegar a los distintos apartados.

4.2.3.3 Historial

En esta página se pueden ver las distintas entradas almacenadas para cada una de las tres secciones: prescripciones, visitas y pedidos. Se ven unos datos principales y si pulsamos en alguna se muestra la vista de detalle, con el resto de información relevante.

4.2.3.4 Servicios de la API

Simplemente he implementado las llamadas a la API que se iban a usar, como obtener campañas, explotaciones, prescripciones, etc. Para no tener que hacer esto en todos los sprints, implemento ya todos los métodos de la API en el servicio *fitocampo*.

4.2.4 Comprobación de requisitos

Tareas del sprint 2			
Tarea a realizar	**Horas planeadas**	**Horas reales**	**Desviación**
Menú principal (home)	3h	5h	66,67%
Menú lateral	6h	6.5h	8,33%
Historial	4h	12h	200%
Conexión y uso de la API	8h	6h	-25%
Total	**21h**	**29.5h**	**40,48%**

Tabla 6 - Comparación de tiempos de las tareas del sprint 2

Al final toda la funcionalidad se ha conseguido de forma correcta para el final del sprint 2, a excepción del historial. Por el momento, respecto a este apartado solo he programado la estructura de la página y visualización de las entradas. En un sprint posterior abordaré la página de detalle para estas.

4.3 Sprint 3

El tercer sprint corresponde al apartado referente a visitas. Se implementará la página relacionada con estas, los componentes necesarios y las operaciones pertinentes para poder crear un registro de visita cómodamente desde la *app*.

4.3.1 Sprint backlog

Las tareas a realizar son las correspondientes a los objetivos mencionados antes.

El sprint ocupará una semana y media de trabajo en la empresa, que equivale a 30h. Empezará el 22 de marzo y terminará el 31 de marzo. En la siguiente tabla se muestran las tareas a realizar junto con las horas previstas para cada una.

Tareas del Sprint 3	
Tarea a realizar	**Horas**
Nuevos servicios API	2h
Vista (HTML y CSS)	12h
Funcionalidad	12h
Total	**26h**

Tabla 7 - Tareas del sprint 4

4.3.2 Diseño

En este apartado se ve el diseño de la página a realizar en este sprint. El objetivo es que quede lo más similar posible.

Figura 8 – Diseño de la página de visita

4.3.3 Implementación

4.3.3.1 Más servicios de la API

Es necesario crear un servicio nuevo para separar las llamadas correspondientes a información de elementos fitosanitarios como especies, plagas, etc. Se crea para separarlo de la API principal de la app ya que la URL es diferente.

4.3.3.2 Vista (HTML y CSS)

Haré uso del *header* y la organización *grid*. Para igualar el diseño, cada opción a seleccionar no son *selects*, sino filas del *grid* coloreadas, con los elementos de títulos, selección e iconos distribuidos en sus columnas.

Para mostrar la selección, he implementado un sistema que cambia el contenido de la página dependiendo de una variable. Cuando esta variable es *false*, el contenido que se muestra es el que se ve en el diseño, la página principal de visita. Pero al seleccionar uno de los selectores de especie, parcela o plaga, la variable pasa a ser *true*, y lo que se muestra en la página es el selector correspondiente. Este selector mostrará la lista de opciones para el apartado elegido, y para saber qué lista mostrar se comprueba una variable del *storage* del teléfono que se actualiza cuando pulsamos en un selector para ponerse con su valor correspondiente.

Por tanto, estamos siempre en la misma página y lo que cambia es lo que se muestra como contenido, así podemos tener los filtros como componentes y no hace falta ir saltando entre páginas.

4.3.3.3 Funcionalidad

La funcionalidad es relativamente sencilla, consiste únicamente en métodos para obtener los datos de cada selector, el tratamiento de las variables para mostrar los distintos filtros (listas de opciones de cada selector) y el método de crear visita que finaliza el proceso.

La idea es que el usuario simplemente tenga que seleccionar entre las opciones dadas, a excepción de las observaciones que es texto libre. Por ello hay una serie de selectores, algunos simples y otros algo más complejos (que se explican a continuación), donde marcar el dato elegido. Esos datos se recogen al finalizar el proceso y se crea el objeto visita que se guarda en la base de datos y que podremos ver luego en la pestaña de historial registrado.

Todo son utilizaciones básicas de servicios de la API y tratamiento de variables. En el caso del método para guardar la visita, simplemente crearemos una variable con los datos que hemos obtenido y se pasa el objeto construido al método post que la añade a la base de datos.

En el caso del filtro, que es un componente aparte, tiene funcionalidades algo diferentes dependiendo de si es de parcela o no. Ambos tienen un buscador, pero el de parcelas permite selección múltiple. Los métodos de este componente simplemente son los referentes a mostrar los datos correspondientes, o añadir lo seleccionado al *storage*.

A los selectores de especie, parcela y plaga, han sido referidos como filtros ya que las opciones que muestran están filtradas, dependiendo de la campaña y explotación seleccionadas en el menú principal, o de la especie seleccionada en la visita en el caso de la plaga.

4.3.4 Comprobación de requisitos

Tareas del sprint 3			
Tarea a realizar	**Horas planeadas**	**Horas reales**	**Desviación**
Nuevos servicios API	2h	1.5h	-25%
Vista (HTML y CSS)	12h	14h	16,66%
Funcionalidad	12h	13.5h	12,5%
Total	**26h**	**29h**	**11,54%**

Tabla 8 - Comparación de tiempos de las tareas del sprint 3

La vista costó más de lo planeado ya que probé distintas formas de presentar los filtros y pasar la información entre ellos, hasta que llegué a la opción adecuada. Llevó algo de tiempo el gestionar el cambio de los componentes, y la diferenciación entre selector de parcela u otros, pero no presentaron problemas muy relevantes. Simplemente la implementación final era más laboriosa que la idea que se tenía en la planificación, por eso resulta en más horas de trabajo reales.

Se dejaron 4h de margen en el *sprint* para las 30h, ya que era previsible que el desarrollo de la página ocupara más de lo previsto. Aun así, nos quedamos dentro del tiempo asignado para el *sprint* 3.

4.4 Sprint 4

Este *sprint* comprende lo relacionado con las prescripciones. Será un *sprint* mucho más largo que los anteriores ya que es el apartado más complejo de la aplicación. Aun así, me parece correcto englobar todas las tareas referentes a esto dentro de un mismo *sprint* ya que pertenecen a la misma operación.

4.4.1 Sprint backlog

Las tareas a realizar son las correspondientes a los objetivos mencionados antes.

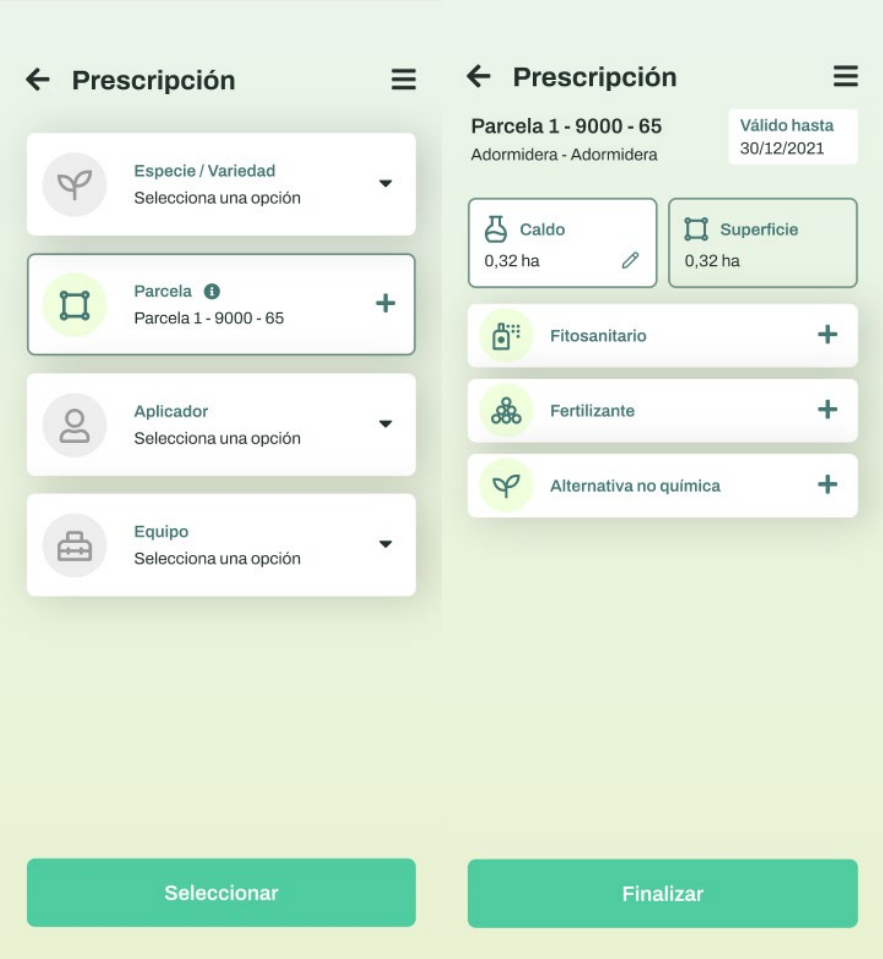
El *sprint* ocupará cuatro semanas de trabajo en la empresa, que equivalen a 80h. Empezará el 1 de abril y terminará el 6 de mayo. En la siguiente tabla se muestran las tareas a realizar junto con las horas previstas para cada una.

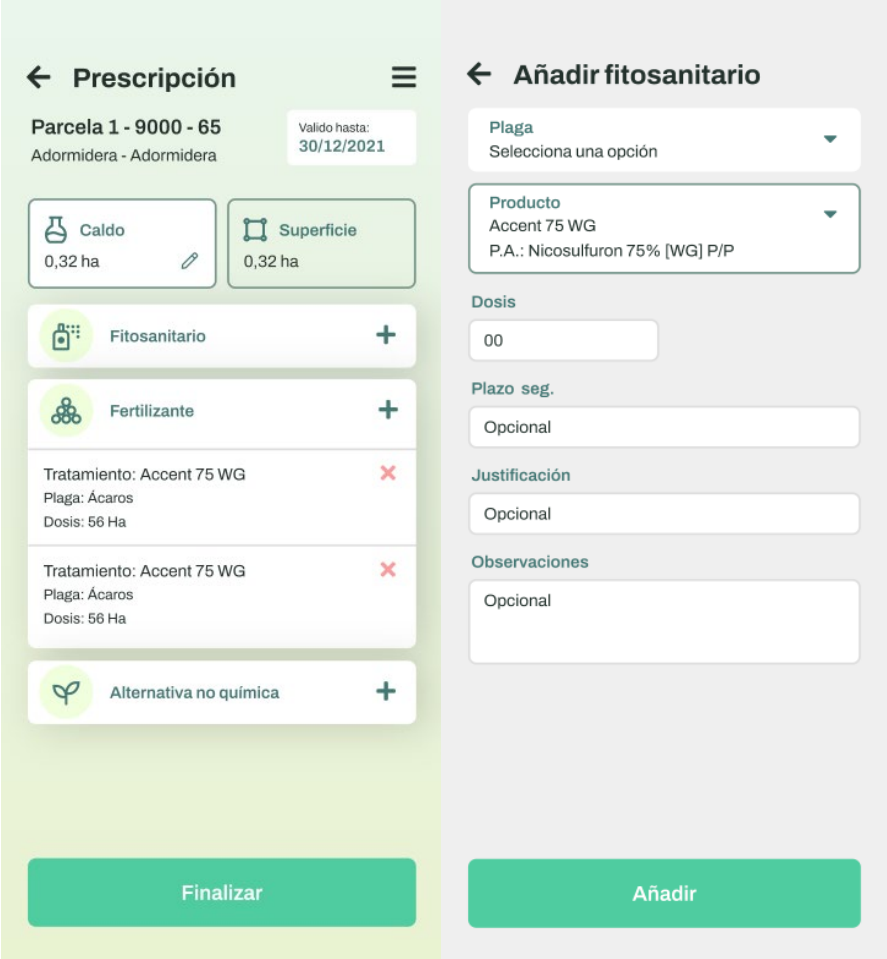
Tareas del Sprint 4	
Tarea a realizar	**Horas**
Vista fase 1	12h
Funcionalidad fase 1	10h
Filtros fase 1	6h
Vista fase 2	16h
Formularios fase 2	4h
Funcionalidad fase 2	14h
Filtros fase 2	10h
Visualización y descarga PDF	10h
Total	**82h**

Tabla 9 -Tareas del sprint 4

4.4.2 Diseño

En este apartado veremos rápidamente el diseño de la página a realizar en este sprint. El objetivo es que quede lo más similar posible al que se muestra a continuación.





Figuras 9, 10, 11 y 12- Diseños de las páginas prescripción fase 1, fase 2, fase 2 con selecciones y añadir fitosanitario.

4.4.3 Implementación

4.4.3.1 Vista fase 1

Esta vista es muy similar a la de Visita, no hay mayores dificultades ni diferencias a la hora de programarla.

4.4.3.2 Funcionalidad fase 1

De igual manera, la funcionalidad también es similar a la de Visita. Cambian ciertas llamadas y lo que se muestra en algunos *selects*, pero el funcionamiento es el mismo. En este caso hay un selector de variedad que va en función de la especie elegida, así como estaba el de plaga en la de visita.

4.4.3.3 Filtros fase 1

De nuevo, es muy similar a los filtros de Visita. No es el mismo componente por comodidad en el paso de información, pero es muy similar. En total esta primera fase de Prescripción es como la de Visita a grandes rasgos. Las diferencias vendrán ahora en la segunda fase.

4.4.3.4 Vista fase 2

Esto es nuevo y exclusivo del proceso de prescripción. A partir de la especie elegida en la fase anterior, se añaden elementos para seguir seleccionando datos. Disponemos de un selector de fecha (implementado por el componente de *Ionic* predeterminado para ello) y dos botones que nos llevarán a unos formularios para introducir la cantidad de caldo y superficie. Además, dispone de

tres selectores para agregar productos a utilizar. Cada uno permite acceder a su pantalla correspondiente, un formulario a rellenar con varios campos y selecciones con sus propios filtros.

La implementación de la parte visual no difiere en mucho en el resto de cosas ya realizadas y no supone ningún reto a mencionar. Lo único, señalar disponemos de varias vistas (filtros y formularios) entre las cuales hay que pasar ciertos datos, pero se hace de la forma estándar como hasta ahora y sin dificultad.

4.4.3.5 Formularios fase 2

En esta tarea se incluye el desarrollo de todos los formularios que aparecen al interactuar con los distintos elementos de la vista. Son todo componentes que se muestran en lugar del HTML de la vista de fase 2 al cambiar un booleano. Se trata del mismo funcionamiento visto para los filtros de visita.

En el caso del caldo y superficie, es el mismo componente consistente en un formulario con una serie de campos y botón de aceptar y cancelar. Para distinguir entre si mostrar la información de uno u otro se envía un booleano desde la vista anterior en el momento en el que se interactúa con alguno de los campos.

Para los selectores de Fitosanitario y Fertilizante se emplea el mismo componente, ya que el formulario es igual, solo cambia lo que se muestra en el filtro interno. Por ello se envía según corresponda la información desde la vista anterior como en los anteriores casos.

En el caso de la alternativa no química, es un componente único y con un formulario similar al mencionado para Fitosanitarios, salvo que, sin selección de producto, solo de plaga.

En todos los casos, al completar los campos del formulario y pulsar finalizar, los datos de los campos se incluyen en un objeto correspondiente dentro de uno de los tres arrays contenidos en el array pedido de prescripción. Así se pueden mostrar las opciones seleccionadas en cada selector y borrarlas si se desea.

4.4.3.6 Funcionalidad fase 2

A modo de visión general, la función de esta página es de nuevo seleccionar más datos relevantes para la prescripción. Se muestra la especie seleccionada en el paso anterior, y se ha de escoger una fecha de validez en un sencillo selector, y otro tipo de datos en formularios y filtros como los vistos anteriormente. En estos filtros se selecciona una plaga y/o producto, en función de la especie ya elegida anteriormente. Se pueden añadir varias entradas para cada selector, se pueden necesitar varios fitosanitarios para un mismo tratamiento.

Los datos seleccionados se van guardando, y al finalizar el proceso (que termina en esta pantalla), se generará el objeto prescripción que se guardará en la base de datos y se podrá ver en el historial registrado.

La idea de esta fase es, como ya se ha mencionado, almacenar una serie de datos y opciones escogidas por el usuario. Para ello he pensado una estructura de código muy similar a las usadas anteriormente. Como particularidad en este caso, se emplea el ya mencionado array de arrays “pedido”, donde ir guardando los objetos resultantes de cada selección de productos. Se declaran aquí las variables que se emplearán para construir esos objetos, y se programan los métodos para ello.

Como métodos más relevantes están los dos para abrir el modal de caldo o superficie según corresponda, donde se abre el *modal* con los datos iniciales y se guardan los seleccionados al cerrarlo.

Por otro lado, para cada opción de fitosanitario, fertilizante y alternativa no química, se programan dos métodos.

- *addForm*, el cual resetea los valores de las variables donde se guardan los datos seleccionados para que al abrir el formulario no haya valores residuales en los campos y coloca la variable booleana correspondiente a true para que se muestre el HTML del componente correspondiente.
- *addProduct*, donde se comprueba que el formulario está en orden y completo y se crea el objeto a añadir en su array correspondiente dentro de pedido. Después se vuelven a resetear los valores de las variables y se pone el booleano del componente a false para volver a mostrar el HTML de la vista de la segunda fase de prescripción.

Además, se añaden otros métodos para: inicializar la vista con los datos de la fase 1, borrar una entrada de un selector, gestionar los mensajes "*toast*" y, por supuesto, finalizar (donde se obtienen todos los valores almacenados hasta ahora durante todo el proceso en el *storage* y las variables de la fase actual). Con ellos se crea el objeto prescripción, el cual debe guardarse en la base de datos y se reinician los valores de todos los campos empleados para la prescripción. Contempla por supuesto que se agregue al menos un producto y que estén los campos obligatorios rellenados antes de dar por finalizado el proceso.

Cuando se termina, se pone a true un booleano que marca que se ha finalizado el proceso, mostrando el componente *end-form* que muestra la pantalla de finalización con opciones de generar pedido (redirige al apartado de pedido que será tratado en el sprint 5), ver PDF (ver en el siguiente punto), y volver al menú principal.

4.4.3.7 Filtros fase 2

Los filtros de esta fase funcionan igual que el resto, componentes que muestran un listado de lo que hay que seleccionar (plagas o productos en este caso), con su buscador, y que se sustituye por el HTML anterior cuando cierta variable tiene el valor pertinente, permitiendo así el uso de componentes que aumenta el escalado y granularidad de la aplicación, igual que en el resto de casos. Lo que se muestra en el filtro, al igual que en otros casos, viene predefinido por los datos seleccionados anteriormente como la especie o campaña, así como por la empresa para la que trabaja el usuario.

4.4.3.8 Visualización y descarga PDF

Tanto en el *end-form* como en la vista de detalle se añade el componente *prescripción-pdf* con una clase CSS que hace que no se muestre en pantalla. Este componente consiste en un HTML, con todos los datos de la prescripción distribuidos con la estructura que queremos que tenga el PDF. Se realiza usando las etiquetas de *Ionic* y, en este caso particular, los estilos en línea (queda más claro y se trata de un maquetado de un documento, el cual es más fácil de reutilizar si está en un mismo fichero).

La parte de TypeScript de este componente es muy sencilla, simplemente consiste en obtener los datos que se van a mostrar en el HTML, pero la realización de este último resultó algo costosa. No por complejidad, sino por ser una maquetación completa del documento, con muchas columnas, filas, líneas de estilo, variables y pruebas para ver que todo se ve como se espera.

Para generar un documento PDF a partir de la información de la prescripción y el HTML del componente anterior, se emplean una serie de módulos que Ionic y *ngx* tienen para ello. Estos son html2canvas [26], jsPDF [27], ionic-native file-opener [28] y ionic-native file [29].

El proceso de conversión consiste en tratar el HTML con la información mediante DOM y a partir de eso generar un archivo PDF. Esto lo harán los módulos internamente una vez se les pasa el elemento raíz del documento HTML. En cuanto a programación simplemente es necesario establecer ciertas propiedades de los módulos como las dimensiones del documento, nombre del fichero, formato concreto, etc.

La carga del PDF es algo lenta, pero funciona perfectamente y muestra los datos de la prescripción como debería.

Al realizarlo, supuso un pequeño problema el comprobar que el PDF se veía de forma adecuada, ya que por temas de tamaños de pantalla y ajustes que hay para el responsive no se veía bien el HTML del componente y no podía ver si iba por buen camino. Por ello lo probé en una vista aparte donde se podía ver bien y fui elaborando el HTML con la estructura que me indicaron en la empresa.

Otro aspecto que produjo un cierto retraso fue el uso de los módulos, ya que era la primera vez que los necesitaba y tuve que buscar información en sus páginas. Pero nada que supusiera un inconveniente real.

En última instancia cuando ya estaba todo programado quedaba probarlo, pero como se trata de un módulo nativo del dispositivo no se puede probar su funcionalidad en el simulador web de Ionic y era necesario crear el proyecto de Android para probarlo con el emulador de Android Studio. Cuando lo hice vi que no mostraba nada el PDF, así que estuve un tiempo comprobando si el HTML del componente estaba correcto y revisando el método de convertir a PDF. Al final me di cuenta de que no era que no lo mostrara, sino que se había activado el modo oscuro en el HTML, con lo cual la fuente era blanca y al pasarlo al PDF (que es de fondo blanco) no se veía nada. Intenté solucionarlo con reglas CSS, pero estaba resultando algo complicado, así que terminé por desactivar el modo oscuro que viene en la aplicación por defecto ya que no lo iba a necesitar. Y todo funcionó correctamente.

El formato del PDF podría mejorarse más, pero consumiría una cantidad de horas que a efectos de este proyecto considero que están mejor invertidas en las funcionalidades restantes y mejorar aspectos generales al terminar.

4.4.4 Comprobación de requisitos

Tareas del sprint 4			
Tarea a realizar	**Horas planeadas**	**Horas reales**	**Desviación**
Vista fase 1	12h	10h	-16,67%
Funcionalidad fase 1	10h	8.5h	-15%
Filtros fase 1	6h	5.5h	-8,33%
Vista fase 2	16h	18.5h	15,625%
Formularios fase 2	4h	4h	0%
Funcionalidad fase 2	14h	16.5h	17,86%
Filtros fase 2	10h	8h	-20%
Visualización y descarga PDF	10h	20h	100%
Total	**82h**	**91h**	**10,98%**

Tabla 10 - Comparación de tiempos de las tareas del sprint 4

Se aprecia que ha habido en general disminuciones en los tiempos debido a que ciertas partes del código eran muy similares a las de tareas anteriores, por lo que se ha podido reutilizar parte o simplemente se ha ido más ágil, ya que se sabía cómo había que hacerlo.

En la fase 2, que era lo diferente, vemos dos tareas con algo más de tiempo real invertido que el planificado, pero tampoco algo muy destacable. Donde sí que se ha producido una gran desviación es en la tarea referente a la vista en PDF. Esto se ha debido a que era algo muy nuevo para mí, había que trabajar con diversos módulos, una estrategia concreta (HTML en módulo invisible que luego se maquetaría) y, además, que escribir el HTML que representa al PDF lo cual llevó su tiempo. Por último, los problemas que surgieron con las pruebas llevaron más tiempo del esperado.

4.5 Sprint 5

Este sprint comprende la funcionalidad del apartado "pedido", el cual es relativamente sencillo, y la vista detalle que se quedó pendiente cuando se programó el historial. Además, se incluyen aquí las tareas relativas a mejora del código y la solución de errores que se detectaron.

4.5.1 Sprint backlog

Las tareas a realizar son las correspondientes a los objetivos mencionados antes.

El *sprint* ocupará dos semanas de trabajo en la empresa, que equivale a 40h. Empezará el 9 de mayo y terminará el 23 de mayo. En la siguiente tabla se muestran las tareas a realizar junto con las horas previstas para cada una. En un principio se planearon 50h, pero como se han acumulado ciertos retrasos en sprint anteriores el tiempo total es menor, con un límite de 40h para permanecer en torno a las 200h de desarrollo. Aun así, no habrá problema en terminar con el alcance y funcionalidad que se tenía previsto conseguir desde un principio.

Tareas del Sprint 5	
Tarea a realizar	**Horas**
Vista pedido	8h
Funcionalidad pedido	6h
Vista detalle	6h
Arreglos y mejoras	20h
Total	**40h**

Tabla 11 -Tareas del sprint 5

4.5.2 Diseño

En este apartado veremos rápidamente el diseño de la página a realizar en este *sprint*. El objetivo es que quede lo más similar posible a el que aparece a continuación:

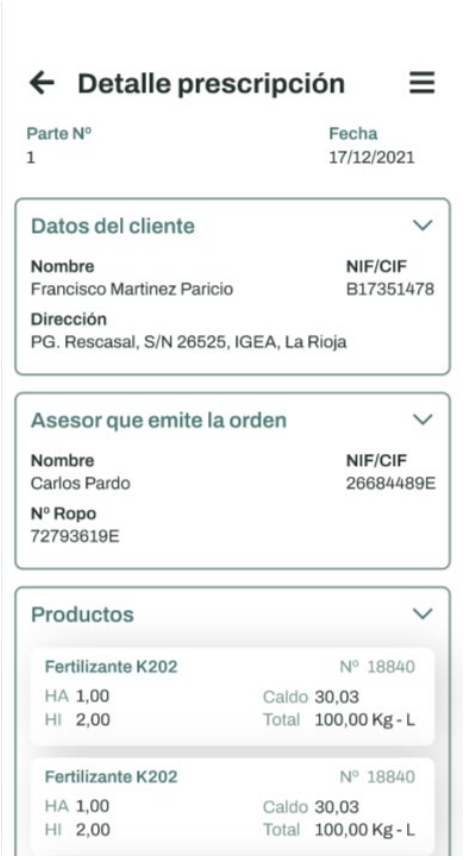


Figura 13 - Diseño de la página detalle.

En este caso no cuento con diseño de la página pedido, la desarrollaré conforme a la estética del resto de páginas ya que todos sus componentes están en algún otro lugar.

4.5.3 Implementación

4.5.3.1 Vista pedido

Esta vista es muy sencilla y no tiene nada que no se haya visto antes. Consta de un *select* en la parte superior para elegir el tipo de producto a pedir, y un selector que abre un filtro como los que vimos anteriormente mostrando los productos del tipo elegido.

Los tipos de producto, y los productos de este van en función a la empresa del usuario (información que está en su perfil en la base de datos). De esta manera, simplemente tiene que seleccionar y nunca puede elegir un producto que no proporcione su empresa por error.

Después hay un campo donde introducir el número de unidades deseadas y un botón para añadir ese producto al pedido. Se pueden añadir todos los que se quieran y eliminar de uno en uno con un botón, al igual que con las parcelas o las opciones de la fase 2 de prescripción. Se permite también editar cada entrada para cambiar sus unidades, y cuando se accede a la edición aparece un mensaje informativo sobre el tipo de distribución del producto (si se vende por envases, capacidad u otros datos relevantes). Al terminar queda una lista con todos los productos a pedir.

Finalmente, en el pie de página se encuentra el botón de finalizar para terminar la operación y guardar el pedido. Tiene varios componentes para mostrar en lugar del HTML principal, como se viene haciendo en todas las páginas, siendo estos los filtros, o la pantalla de fin de proceso (que en esta ocasión no tiene ninguna interacción más que volver a la página principal).

4.5.3.2 Funcionalidad pedido

Hay dos formas de entrar aquí, desde el menú para hacer un pedido particular nuevo, o desde una prescripción para pedir los productos que se necesitan. En este último caso, se recoge primero la información de la prescripción y con ella se construyen las entradas de productos que irán en el pedido. Para esto último se necesita un método que calcula las unidades de producto en función a la cantidad de caldo y superficie teniendo en cuenta las medidas y dosis necesarias para cada producto.

Se incluyen también métodos para aplicar cambios en el *select* de tipo de productos a las variables que se pasan a los filtros, el manejo de mensajes *toast*, redirección al menú, selección, borrado, edición y añadido de productos, etc. Aunque ninguno de estos métodos supone complicación adicional.

Finalmente está el método de finalización de pedido, el cual crea el objeto pedido, lo coloca en el *storage* y lo guarda también en la base de datos mediante la llamada correspondiente a la API. Finalmente reinicia los valores seleccionados en este proceso.

4.5.3.3 Vista detalle

Esta vista no tiene complejidad más allá de la disposición de la misma. Se trata de un extenso HTML con todos los datos de la entrada en concreto. Se vale de distintas variables para mostrar los datos y secciones correspondientes dependiendo del tipo de entrada. Estos datos se recogen al iniciarse la vista comprobando si los campos están en el detalle traído del *storage*, y se guardan en las variables correspondientes para poder mostrarse en el HTML. Se emplea el componente *ion-accordion* para mostrar los datos por secciones que puedan ampliarse o recogerse para una visualización más cómoda de los datos.

En el caso de tratarse de una prescripción, se le añade un botón en la esquina inferior derecha para poder guardarla en PDF. Por tanto, también se incluye el componente *prescripción-pdf* (invisible). El funcionamiento de todo lo relativo al PDF es exactamente igual a lo mencionado en el sprint anterior.

Más allá de lo mencionado no presenta ninguna dificultad.

4.5.3.4 Arreglos y mejoras

He invertido tiempo en diversas tareas una vez terminado el desarrollo para mejorar la calidad del producto.

Por ejemplo, he comentado los métodos para que se sepa qué hace cada uno. En un principio, pensé en escribir tanto comentarios como el propio código en inglés, pero vi que resultaba más confuso, ya que las llamadas a la API eran en español, lo mostrado por pantalla también y la traducción de algunos términos no resultaba cómoda. Por ello decidí volver a traducir al español todo lo hecho hasta el *sprint* 3 aproximadamente, lo cual me llevó algo de tiempo.

He trabajado también en el responsive de las páginas, para que se vea adecuadamente en distintos dispositivos, lo cual me llevó unas horas. Y, algo que me consumió tiempo en este aspecto, son los botones inferiores. En un principio los colocaba por CSS, pero al avanzar en el desarrollo me di cuenta que es más cómodo dejarlos como *footer*, así que los puse todos de esta manera, pero aparecía una pequeña línea que quería quitar. Invertí unas horas en intentarlo, pero, al parecer, es algo intrínseco del *footer* de Ionic, y no conseguí quitarla por ninguno de los métodos que encontré, ni usando CSS al componente, ni al DOM generado, ni con métodos en el TypeScript.

Por otro lado, invertí también algo de tiempo en la realización de las pruebas de los distintos procesos, para comprobar que funcionaban correctamente y arreglando errores muy puntuales. También dediqué un tiempo a poblar (con algunos datos de prueba de plantaciones, productos, aplicadores, etc.) la base de datos para poder hacer pruebas y demostraciones correctamente.

4.5.4 Comprobación de requisitos

Tareas del sprint 5			
Tarea a realizar	**Horas planeadas**	**Horas reales**	**Desviación**
Vista pedido	8h	6h	-25%
Funcionalidad pedido	6h	4.5h	-37,5%
Vista detalle	6h	8.5h	41,67%
Arreglos y mejoras	20h	24.5h	-22,5%
Total	**40h**	**43.5h**	**8,75%**

Tabla 12 - Comparación de tiempos de las tareas del sprint 5

5 Seguimiento y control

En este apartado se comparan las horas reales invertidas con las planificadas en un inicio para cada sección del proyecto, y se explican las desviaciones más destacables con sus causas.

5.1 Horas dedicadas

Código	Nombre	Descripción	Tiempo estimado		Tiempo invertido	Desviación
1	**Gestión inicial**		**12h**		**13h**	**8,33%**
1.1	Planificación	Dividir el proyecto en diferentes tareas, planificar todos los aspectos de la realización del mismo y estimar tiempos.	6h		6,5h	8,33%
1.2	Instalación inicial	Instalar y preparar todos los programas y software necesario para la realización del proyecto.	1h		3h	200,00%
1.3	Captura de requisitos	Recogida de los requisitos funcionales y no funcionales.	2h		1,5h	-25,00%
1.4	*Product Backlog*	Elaboración del *Product Backlog* a partir de los requisitos.	3h		2h	-33,33%
2	**Gestión continua**		**68h**		**71,5h**	**5,15%**
2.1	Memoria del proyecto	Redacción de la memoria que documente todo el proceso de trabajo en el proyecto.	54h		59,5h	10,19%
2.2	Investigación	Búsqueda de información necesaria para el desarrollo de cualquier fase del proyecto.	6h		7h	16,67%
2.3	Seguimiento y control	Reuniones para comprobar avance, revisiones al finalizar los sprints y correcciones si fueran necesarias debido a desviaciones de tiempo.	8h		4h	-50,00%
3	**Desarrollo**		**200h**		**202,5h**	**1,25%**
3.1	Sprint 1	Planificación e implementación del primer sprint.	20h	14h	18,5h	-7,50%
3.2	Sprint 2	Planificación e implementación del segundo sprint.	20h	21h	29,5h	47,50%
3.3	Sprint 3	Planificación e implementación del tercer sprint.	30h	26h	29h	-3,33%
3.4	Sprint 4	Planificación e implementación del cuarto sprint.	80h	82h	91h	13,75%
3.5	Sprint 5	Planificación e implementación del quinto sprint.	50h	40h	43,5h	-13,00%
4	**Defensa**		**20h**		**20h**	**0%**
4.1	Presentación	Elaboración de la presentación que acompañará la defensa del TFG.	10h		-	-
4.2	Exposición	Preparación y ensayo de la propia defensa.	10h		-	-
Totalidad del TFG		**Horas totales empleadas**	**300h**		**316h**	**5,33%**

Tabla 13 - Comparación de tiempos reales y estimados en el proyecto.

Al ser la entrega de la memoria anterior a la realización de las tareas de la sección *Defensa,* se dejan los espacios de horas invertidas y desviación vacíos en ese apartado.

En la columna de tiempo estimado de los sprints se muestran dos datos, primero la estimación realizada en la planificación inicial del proyecto y luego la prevista en la fase de planificación de cada sprint concreto. Esta es distinta ya que se hace en base a las tareas a realizar y la experiencia obtenida a lo largo del proyecto. Aun así, la comparativa de desviación se hace respecto de las horas planificadas al inicio del proyecto, como con el resto de secciones, ya que la comparación con las horas planeadas en los sprint está al final de cada uno de ellos.

5.2 Desviaciones

Se analizan a continuación las causas de las desviaciones más relevantes en el proyecto.

- **Instalación inicial**: La desviación es muy grande pero realmente solo supuso 2h más de retraso. Resultó perfectamente asumible y se produjo por temas de compatibilidad entre los sistemas, versiones, tiempos de descarga… No hubo realmente ningún problema, simplemente tomó algo más de lo esperado.
- **Memoria**: Ha tomado unas horas más de lo esperado simplemente por la repetición de la redacción de algún apartado y cambios de enfoque en la redacción de los mismos tras algunas revisiones con el tutor. Aunque tampoco ha supuesto una desviación muy grave.
- **Seguimiento y control**: Este apartado ha supuesto menos tiempo que lo planeado inicialmente, ya que en un principio se pensó en hacer reuniones con el tutor de la empresa cada final de sprint, pero no se hizo así finalmente por conveniencia para el proyecto. Al poder comprobar yo mismo si todo estaba correcto con el funcionamiento de la *app* y comparando con el diseño, y no surgir ningún problema muy grave, la revisión final del sprint se realizaba en muy poco tiempo. De igual manera las consultas de revisión del proyecto que se realizaron fueron muy rápidas ya que no hubo problemas mayores. Finalmente, la redacción de esta sección (*5 Seguimiento y control*) ha sido también rápida.
- **Sprint 2**: En este caso fue por un fallo de estimación temporal en cuanto al trabajo en la página del historial. Terminó costando más de lo planificado, pero no por algún problema en particular sino más bien por un fallo en la estimación ya que era más trabajo del pensado inicialmente.
- **Sprint 4**: La desviación en este Sprint no es muy grande debido a que ocupa un gran volumen de horas de trabajo, pero sí es una cantidad suficiente para comentar. Principalmente se debe a invertir más horas en todas las tareas con elementos nuevos hasta el momento. En las tareas con elementos similares a los ya realizados se ahorró tiempo, que se pudo invertir en las relativas a la *fase 2* del proceso de prescripción. Finalmente, en la realización del proceso de conversión a PDF se tuvo que invertir el doble de tiempo del estimado, ya que los módulos supusieron tiempo extra de instalación, pruebas y configuración como ya se comentó en el apartado correspondiente (*4.4.3.8 Visualización y descarga PDF*).

Teniendo todo esto en cuenta, se ve que no ha habido desviaciones excesivas en ningún apartado, aunque si algo de tiempo extra invertido sobre todo en el desarrollo. La desviación total, como indica la tabla del apartado anterior, es del 5,33%. Teniendo en cuenta que el alcance del proyecto se ha cumplido por completo, se considera una desviación asumible.

6 Conclusión

Con la finalización del proyecto, se ha obtenido la aplicación de *FitocampoApp*, la cual permite realizar las tareas de registros de visitas, prescripciones y pedidos. Todo ello con una buena experiencia de usuario y un buen diseño de pantallas.

La *app* cumple con todos los requisitos pensados originalmente, y se ha codificado de forma ordenada y coherente. Aun así, opino que se podría haber enfocado de forma algo distinta, con una programación mucho más general para obtener un producto base primero, y luego con pequeños cambios conseguir *FitocampoApp*. De esta forma la aplicación base puede servir (como mencionaba en la introducción) para elaborar aplicaciones de funcionalidad similar, como una de peritaciones de seguros, de forma extremadamente rápida y eficiente. La forma de conseguirlo sería simplemente dedicar más tiempo a pensar una forma generalizada de programar usando nombres genéricos y módulos estructurados de forma que sea sencillo personalizarlos. Está construida con esta idea, pero llevarla a cabo completamente hubiera llevado aún más tiempo, también de planificación, y por tema de calendario y alcance real del proyecto no era viable en esta ocasión.

También se podría añadir alguna mejora a la aplicación, como un sistema de guardar plantillas para completar muy rápido procesos de visita o prescripción si son muy similares. Por ejemplo, que ya vengan seleccionados los mismos datos de plaga, aplicador y fitosanitario entre otros, si el mismo trabajador va a aplicar idéntico tratamiento en varias fincas.

Por otro lado, trabajar en este proyecto me ha ayudado a seguir confirmando que me gusta el desarrollo de aplicaciones móviles. Disfruto con el proceso de crear algo útil yo mismo y aprendiendo tecnologías para conseguir todo tipo de funcionalidades. También ha servido para acostumbrarme al proceso laboral y de desarrollo en el cual me encuentro bastante cómodo. Finalmente, me ha supuesto una mejora en cuanto a solucionar problemas y encontrar formas diferentes de resolver una misma cuestión. No he tenido bloqueos ni estrés y el trabajo ha sido constante y estable.

Concluyendo, este proyecto me ha sido de utilidad para aprender más sobre el desarrollo de aplicaciones móviles, despertar mi curiosidad en nuevas formas de programar y afianzar más mi decisión de dedicarme a ello.

7 Bibliografía

1. **SDi** - https://www.sdi.es/

2. **Ager Technology** - http://www.agertechnology.com/app.php

3. **Syngenta** - https://www.syngenta.es/tankcalc

4. **VisionAgro** - https://www.visionagro.info/app-gestion-de-inventario-fitosanitario/

5. **AgrosLab** - https://www.agroslab.com/como-registrar-tratamientos-fitosanitarios-desde-la-app/

6. **PlantCare** - https://www.plantcare.es/#1510762173204-e445e298-d810

7. **FitoGest** - https://locatec.es/fitosgest/

8. **Documentación oficial de Ionic** - https://ionicframework.com/docs/

9. **Documentación oficial de Angular** - https://angular.io/

10. **Documentación de HTML** - https://devdocs.io/html/

11. **Documentación de CSS** - https://devdocs.io/css/

12. **Documentación oficial de TypeScript** - https://www.typescriptlang.org/

13. **Documentación oficial de Bootstrap** - https://getbootstrap.com/

14. **Documentación oficial de capacitor** - https://capacitorjs.com/

15. **Página oficial de VisualStudioCode** - https://code.visualstudio.com/

16. **Página oficial de AdobeXD** - https://www.adobe.com/es/products/xd.html

17. **Página oficial de Odoo** - https://www.odoo.com/es_ES

18. **Página oficial de Microsoft 365 Word** - https://www.microsoft.com/es-es/microsoft-365/word

19. **Página oficial de Outlook** - https://outlook.live.com/owa/

20. **Página oficial de GitLab** - https://about.gitlab.com/

21. **Página oficial de Laravel** - https://laravel.com/

22. **Página oficial de PostgreSQL** - https://www.postgresql.org/

23. **Información sobre Scrum** - https://www.atlassian.com/es/agile/scrum

24. **Módulo de Ionic Storage** - https://www.npmjs.com/package/@ionic/storage

25. **Documentación sobre el Ionic Grid** - https://ionicframework.com/docs/api/grid

26. **Módulo de html2canvas** - https://www.npmjs.com/package/html2canvas

27. **Módulo de jsPDF** - https://www.npmjs.com/package/jspdf

28. **Módulo de file-opener de ionic-native** - https://www.npmjs.com/package/@ionic-native/file-opener

29. **Módulo de file de ionic-native** - https://www.npmjs.com/package/@ionic-native/file