Grado en Ingeniería Informática

Trabajo Final de Grado

Desarrollo de una aplicación móvil de pedido de comida a domicilio para sistemas Android e iOS mediante React-Native

Autor:
Adrián Picazo Marín

Supervisor:
Sergio Aguado González
Tutor académico:
Pedro Latorre Carmona

Fecha de lectura: 22 de junio de 2018
Curso académico 2017/2018

Resumen

Proyecto *front-end* para proveer de un cliente a un sistema de pedido de comida a domicilio. Consiste en desarrollar una aplicación móvil para los dos sistemas operativos que actualmente predominan en el mercado: *Android* e *iOS*. El objetivo es desarrollar simultáneamente dos aplicaciones nativas (una para cada sistema) mediante la tecnología *React-Native*.

El proyecto se enmarca dentro de la asignatura *EI1054 - Prácticas en Empresa y Proyecto Fin de Grado* del Grado en Ingeniería Informática de la Universidad Jaume I (UJI) de Castellón. Ha sido propuesto por la empresa Soluciones Cuatroochenta S.L. y se realiza durante la estancia en prácticas. Se trata de un proyecto formativo sin intención de lanzar el producto al mercado. Para su desarrollo se emplea una metodología *Scrum* adaptada a las condiciones de la empresa y del proyecto.

En el Capítulo 1 de esta memoria explicamos el contexto, la motivación y los objetivos que se persiguen. En el Capítulo 2 describimos el proyecto con las tecnologías, herramientas, utilidades y lenguajes empleados. En el Capítulo 3 detallamos el desarrollo técnico del proyecto con la metodología, la planificación previa, la arquitectura, la estimación de recursos y los costes. En el mismo capítulo, documentamos la pila del producto y los sprints. En el Capítulo 4 abordamos la verificación y validación. Finalmente, en el Capítulo 5 presentamos las conclusiones.

Palabras clave

Pedidos, Comida, Front-end, Aplicación, Móvil, iOS, Android, React-Native, Scrum.

Keywords

Orders, Food, Front-end, Application, Mobile, iOS, Android, React-Native, Scrum.

Agradecimientos

La realización de este trabajo supone la culminación a cuatro años de grado en Ingeniería Informática en la Universidad Jaume I. Cuatro años durante los que he adquirido multitud de conocimientos en diversos ámbitos de la vida. Cuatro años de mucha formación impartida por grandísimos profesionales. Cuatro años durante los que he conocido a muchas personas que me han ayudado y apoyado para hacer esta travesía más corta y amena. Cuatro años que me han servido para sentirme más válido y demostrar de lo que soy capaz en lo personal y en lo profesional. Es por ello que me gustaría agradecer a todas las personas que me han ayudado y apoyado durante este proceso, y en este trabajo en particular.

En primer lugar, me gustaría agradecer a mis compañeros de grado y de trabajo Ángel, Rubén, Óscar y Oussama, con quienes he recorrido estos dos últimos años. A mis compañeros y amigos Pau y Benito, con los que empecé y espero un día ver terminar. A mis amigos de siempre. A los de Fardatxo. A Alejandro y a Pablo. A mi familia. En general, a todos quienes me han apoyado y ayudado en estos años después de unos muy malos momentos y un duro fracaso. Y, en especial, a mi gran amigo Carlos y a mi especial compañera Celia, quienes siempre han confiado en mí hasta en los momentos más bajos.

En segundo lugar, agradecer a todos los profesores que me han ayudado a llegar hasta aquí, a unos más, a otros menos, pero en especial a Reyes Granguel, Óscar Belmonte, Ramón Alberto Mollineda, Lledó Museros, Vicente Ramón Tomás, David Llorens, Juan Carlos Amengual, Asunción Castaño y Mercedes Segarra.

Finalmente, agradecer quienes han hecho posible que salga adelante este trabajo. A Sergio Aguado, CTO de Cuatroochenta, por brindarme esta oportunidad. A toda la gente de Cuatroochenta, y muy especialmente a Valeriu, Francisco y Cata, quienes me han ayudado mucho por pesado que fuera. Y, a mi tutor Pedro Latorre Carmona, por su profesionalidad, por su gran ayuda y colaboración, y a quien no dudaría en volver a escoger para otro proyecto.

Gracias a todos, de verdad.

Adrián Picazo Marín.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto que se presenta en este documento se lleva a cabo como parte de la asignatura *EI1054 - Prácticas en Empresa y Proyecto Fin de Grado* del Grado en Ingeniería Informática de la Universidad Jaume I (UJI) de Castellón.

Consiste en desarrollar **un cliente para un sistema de pedido de comida a domicilio para plataformas móviles**. Ha sido propuesto por Soluciones Cuatroochenta S.L como una tarea formativa en una determinada tecnología y está enmarcado dentro de un proyecto más grande para realizarse conjuntamente con otro alumno.

Soluciones Cuatroochenta S.L. (en adelante Cuatroochenta) se dedica principalmente al diseño web y de aplicaciones móviles para los sistemas operativos *Android* e *iOS*. Desarrolla tanto software a medida para clientes como productos y/o servicios para usuarios en general (Sefici, Vennova o 480Interactive). Tiene sus oficinas principales en el mismo campus universitario, concretamente en el edificio Espaitec 2.

Cuatroochenta ha propuesto el proyecto como una tarea de formación con una determinada tecnología. Su objetivo es que el alumno adquiera unos conocimientos con expectativas de contratarlo de cara al futuro. No hay intención de que el sistema desarrollado sea lanzado al mercado. Ni de emplearlo internamente con cualquier otro fin.

Sin embargo, nuestro objetivo se centra en el desarrollo de dicho sistema, y sí consideraremos que vaya a salir al mercado, ya que nuestra idea es que el proyecto se asimile lo máximo a uno real. En consecuencia, los objetivos y resultados esperados girarán en torno al producto y no a la formación que vayamos a recibir ni a las aptitudes y competencias que vayamos a adquirir.

1.2. Objetivos del proyecto

El objetivo principal es **desarrollar una aplicación móvil para sistemas *Android* e *iOS* mediante la tecnología *React-Native* que sirva como cliente a un sistema de pedido de comida a domicilio**.

El objetivo principal podemos desglosarlo en los siguientes subobjetivos:

1. Proporcionar un cliente al sistema de pedido de comida a domicilio como aplicación móvil.
   a) Permitir al usuario listar los restaurantes con servicio de reparto a domicilio.
   b) Permitir al usuario consultar los diferentes productos categorizados por restaurante.
   c) Permitir al usuario gestionar un pedido para un restaurante en concreto.
   d) Permitir al usuario registrarse y facilitarle la gestión de sus pedidos.
2. Desarrollar la aplicación nativamente para los sistemas *Android* e *iOS*.
   a) Conseguir que se muestre correctamente para la mayoría de dispositivos *Android*.
   b) Conseguir que se muestre correctamente para la mayoría de dispositivos *iOS*.
3. Realizar el desarrollo para ambos sistemas a la vez empleando la tecnología *React-Native*.
   a) Alcanzar un rendimiento mayor que con el desarrollo por separado para cada sistema.
4. Emplear una metodología ágil para llevar a cabo la gestión del proyecto.
5. Obtener una versión final en 300 horas de prácticas.
   a) Conseguir una versión estable y totalmente funcional.
   b) Conseguir un acabado consistente.
   c) Conseguir un alto grado de usabilidad.[1]

Los resultados esperados de cara al lanzamiento de la aplicación son:

- Estar presente en los dos sistemas operativos predominantes del mercado.
- Proporcionar una comodidad al usuario a la hora de realizar pedidos de comida.
- Resultar un sistema intuitivo que agilice el proceso de pedido para el usuario.
- Uniformizar los diferentes métodos de pedido de los restaurantes con servicio a domicilio.
- Ofrecer un sistema de reparto a aquellos restaurantes sin los suficientes recursos.
- Dar visibilidad a los nuevos restaurantes añadidos al sistema.

[1] Según la definición ISO-25010, la usabilidad es la 'capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones.'.

1.3. Estructura de la memoria

Capítulo 1. Contexto y motivación. Objetivos del proyecto. Estructura de la memoria.

Capítulo 2. Descripción del proyecto. Tecnologías. Herramientas, lenguajes y utilidades.

Capítulo 3. Metodología. Planificación. Arquitectura. Estimación y costes. Sprints.

Capítulo 4. Verificación y validación.

Capítulo 5. Conclusiones.

1.4. Código del proyecto

Todo el código desarrollado durante el proyecto se encuentra en el repositorio *GitHub* y está organizado en torno a dos versiones de la aplicación[2]: una primera versión que emplea servicios web[3], y una segunda versión donde se sustituyen esos servicios por los de *Firebase*[4]. La primera versión es posterior a la segunda y comparte un 80-90 % del código (prácticamente son iguales salvo en la implementación de los servicios). La segunda versión está terminada y es totalmente funcional, mientras que la primera está incompleta.

[2] En capítulos posteriores explicaremos el porqué de realizar dos versiones.
[3] https://github.com/adpimar/mow-webservices
[4] https://github.com/adpimar/mow-firebase

Capítulo 2

Descripción del proyecto

2.1. Meals On Wheels

El presente proyecto está planteado por Cuatroochenta como parte de una propuesta más amplia. Dicha propuesta, consistente en **desarrollar un sistema de pedido de comida a domicilio** completo, está subdividida en dos proyectos: (1) una parte servidora con su base de datos y sus servicios web junto con otra parte cliente para plataformas web, y (2) una parte cliente para plataformas móviles *Android* e *iOS*. El primero de ellos comprende tanto *back-end* como *front-end*, mientras que el segundo es exclusivamente *front-end*. El primero debe proveer de servicios web al segundo.

Cada proyecto lo desarrolla un alumno diferente. La idea es que ambos trabajen en equipo y se coordinen para sacar adelante el sistema completo, aunque pudiera existir la imposibilidad de llevarlo a cabo conjuntamente por diversas razones. Por ejemplo: que ambos alumnos no coincidan en horario de prácticas[1], que sus rendimientos sean muy dispares o que se produzcan grandes contratiempos en uno de los proyectos que retrasen al otro. En ese caso, Cuatroochenta cubriría el desarrollo de las dependencias necesarias (como los servicios web). Del primer proyecto se encarga Roberto Bernad y del segundo, Adrián Picazo, el autor de esta memoria.

El proyecto que aquí tratamos consiste en desarrollar la parte cliente del sistema como una **aplicación** para plataformas móviles, tanto de *Android* como de *iOS*.

Dicha aplicación debe permitir al usuario realizar pedidos de comida a domicilio de los distintos restaurantes que ofrezcan un servicio de reparto en Castellón de la Plana. Ha de mostrar un listado de todos ellos y la posibilidad de filtrarlos por tipo: mejicano, italiano, chino, etc. Para cada restaurante debe mostrar detalles propios tales como el nombre, el tipo y una imagen además de una lista de productos por categoría: ensaladas, entrantes, pastas, etc. Ha de gestionar los pedidos con cada uno de los restaurantes: debe permitir añadir y eliminar los productos a un pedido y efectuarlo indicando la dirección de entrega. Debe mostrar un resumen de un pedido con el coste en euros una vez haya sido efectuado. Debe ofrecer una opción para

[1]En Cuatroocenta hay total libertad para que los estudiantes realicen su horario de prácticas, por lo que pueden ser muy diferentes.

almacenar la dirección de entrega para que el usuario no tenga que introducirla cada vez que haga un pedido. Para almacenar una dirección, será necesario que los usuarios estén registrados. En consecuencia, ha de haber un registro y un acceso (*login*). El nombre de la aplicación: **Meals On Wheels**.

Una aplicación como ésta está pensada para comodidad de los usuarios, tanto de los clientes que piden productos como de los restaurantes que los ofertan. A ambos les facilita las labores de gestión. Sirve de oportunidad a aquellos restaurantes con pocos recursos para disponer de un servicio de pedido a domicilio. Les da a conocer a la comunidad de usuarios (les publicita). Además, supone un sistema de pedido unificado: no existen multitud de formas ni de ofrecer productos ni de realizar pedidos, sino una única forma sencilla e intuitiva.

El alcance del proyecto comprende todo el desarrollo del *front-end* para una aplicación móvil: pintado de componentes, navegación entre pantallas, comprobación de formularios, etc. En cambio, no comprende ni el diseño de la base de datos ni el diseño de los servicios web: obtener el listado de restaurantes, registrar un usuario, comprobar si el usuario ya está registrado, etc. Es tarea de Roberto Bernad implementar el *back-end* (o de Cuatroochenta en su defecto).

El alcance de la aplicación comprende exclusivamente funcionalidad *front-office*[2]. Tanto los clientes como los encargados de los restaurantes son quienes gestionan y mantienen sus datos y los datos de los pedidos. Sin embargo, para no desarrollar un proyecto de tanta envergadura, no se tiene en cuenta ni el mantenimiento de los restaurantes ni la baja de los clientes. Sí que se tiene en cuenta el acceso a los restaurantes y a sus productos, la gestión de los pedidos y el alta de los clientes.

2.2. Tecnologías

Entre las tecnologías empleadas destacan dos: *React-Native* y *Redux*.

React-Native es la principal y está basada a su vez en *React*. *React* es una tecnología que permite que las vistas o interfaces de usuario se diseñen por componentes y que además éstos sean reactivos, es decir, que reaccionen ante cualquier cambio. Por otro lado, *React-Native* permite desarrollar simultáneamente una aplicación móvil en los dos sistemas operativos *Android* e *iOS*. Emplear *React-Native* implica usar también *React*.

Redux es una tecnología secundaria pero que ayuda enormemente en el diseño de una aplicación. Permite desacoplar en cierta manera su estado global de su parte visual. El estado de una aplicación puede comprender tanto los datos provenientes de los servicios como los estados de las distintas vistas. Sigue el patrón de diseño *Flux* y su funcionalidad puede ser extendida mediante *middlewares*, entre los que destaca (y en este proyecto se emplea) *Redux-Thunk*. La tecnología *Redux* encaja a la perfección con *React*.

[2]En Ingeniería del Software, el término *front-office* hace referencia a la parte del sistema que tiene contacto con el cliente. En contraposición, el *back-office* es la parte del sistema dedicada a su gestión, operada por los administradores. No confundir con los términos *back-end* y *front-end*, que se refieren al área lógica del sistema (base de datos y servicios) y a su diseño gráfico (pantallas o ventanas gráficas) respectivamente. El *front-office* y el *back-office* son las dos partes que conforman el *front-end*.

Para entender perfectamente cómo funcionan explicaremos con más detalle en qué consiste cada una de ellas y de las tecnologías que se hacen servir.

2.2.1. React

Hasta hace poco, las interfaces de usuario se han diseñado mediante plantillas (*templates*) y directivas *HTML*. De alguna forma estas plantillas y directivas nos obligan a seguir un conjunto de reglas y a ceñirnos a un patrón MVC[3], limitando la capacidad de abstracción del diseño. Para evitarlo, *React* parte de un punto de vista diferente: dividir las interfaces en componentes.

***React* es una biblioteca *JavaScript* para diseñar interfaces de usuario en base a componentes**. Es declarativa, eficiente y flexible. Los componentes, que encapsulan la lógica de las vistas, son reutilizables y extensibles. En consecuencia, permite conseguir un diseño general muy consistente, escalable y mantenible. Cada componente administra su propio estado, de manera que reaccionan[4] a los cambios de sus datos. Esto es, que ante cualquier cambio automáticamente el componente vuelve a dibujarse en pantalla. Aunque se trate de una biblioteca, existe todo un ecosistema de herramientas y aplicaciones que lo equiparan a un *framework*[5]. [21] [11] [8]

Además de *JavaScript*, *React* emplea también *JSX*, una extensión de sintaxis cercana al *HTML* que ha sido creada expresamente para dotar de legibilidad al código cuando se requiere implementar directamente partes de la interfaz de usuario. [21] [11]

React es de código abierto. Actualmente está siendo mantenido por *Facebook* e *Instagram* junto con una comunidad de compañías y desarrolladores independientes. [25]

2.2.2. React-Native

Hoy en día, una aplicación móvil puede ser de tres maneras: web, híbrida y/o nativa. Una aplicación web realmente no es una aplicación móvil. Está escrita en código web (*HTML*, *CSS* y *JavaScript*) siguiendo un diseño *responsive*[6] que permite percibirla como una aplicación móvil si accedemos a ella a través de un navegador. No se ejecuta en el dispositivo, sino en un servidor, y de ahí se accede. Una aplicación híbrida sí es una aplicación móvil. También está escrita en código web. Sin embargo, sí se ejecuta en el dispositivo, dentro de un contenedor que transforma dicho código a nativo. Aparenta ser una aplicación nativa, pero no lo es. Una aplicación nativa es una aplicación móvil, escrita en código nativo y ejecutada por el dispositivo sin ningún tipo de navegador o contenedor. En la Figura 2.1 mostramos los tres tipos de aplicaciones.

A nivel de desarrollo, las aplicaciones web e híbridas las desarrollamos una sola vez, mientras

[3]Patrón de arquitectura Modelo-Vista-Controlador (MVC) que separa los datos y la lógica de negocio de su representación en dos capas (modelo y vista) y que emplea una intermedia para gestionar los eventos y comunicaciones (controlador).

[4]De ahí el nombre React.

[5]Entorno o marco de trabajo. Conjunto estandarizado de conceptos, prácticas y criterios.

[6]El *responsive web design* o diseño web adaptable es una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visitarlas. [24]

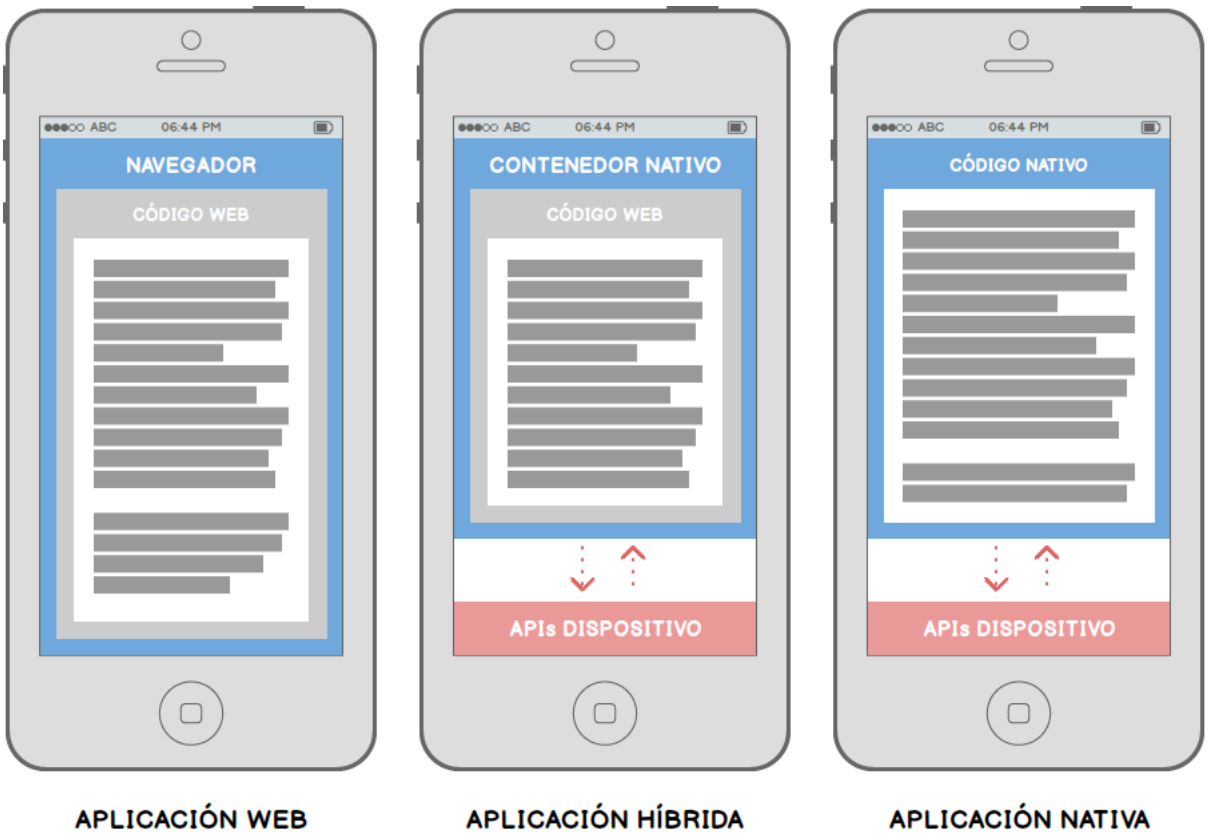


Figura 2.1: Los tres tipos de aplicaciones móviles de hoy en día: (1) la aplicación web accedida a través de un navegador, (2) la aplicación híbrida ejecutada en un contenedor y traducida a nativa y (3) la aplicación nativa directamente ejecutada en el dispositivo. Fuente: adaptación propia de [12]

que las nativas tantas veces como en sistemas queramos que estén presentes. Actualmente, *Android* e *iOS* son los dos sistemas operativos predominantes en el mercado de móviles. Por tanto, si pretendemos tener una misma aplicación para ambos sistemas es necesario desarrollarla dos veces.

Todo ello trae consigo una serie de ventajas y desventajas. Un desarrollo web e híbrido tiene unos costes más bajos frente a uno nativo. La reutilización de código es mayor. En contraposición, el resultado de sus aplicaciones es peor. Por ejemplo, las aplicaciones web no pueden emplear casi ninguna de las APIs[7] del dispositivo frente a las nativas que pueden usarlas todas. La interfaz de usuario tampoco es tan consistente en las aplicaciones web e híbridas. Pero lo principal es el rendimiento, que es mucho más bajo en estas dos: peor consumo de la memoria, una menor velocidad y un peor aprovechamiento del *hardware*. En la Figura 2.2 mostramos una tabla con una comparativa resumen. [13]

En este sentido, si queremos desarrollar una aplicación para ambos sistemas, hay que decidir entre hacer dos por separado o una común. Tardar más o menos tiempo. Obtener un mejor o peor resultado. De tal disyuntiva surge *React-Native*.

[7]Sistema de herramientas y recursos que ayuda a los desarrolladores a crear aplicaciones de software. Por ejemplo, la API de geolocalización de los dispositivos móviles.

	WEB	HÍBRIDO	NATIVO
Coste de desarrollo	Bajo	Bajo	Alto
Reusabilidad del código / Portabilidad	Alta	Alta	Baja
Acceso a las APIs del dispositivo	Pocas	Bastantes	Todas
Consistencia de la interfaz de usuario alcanzable	Media	Media	Alta
Distribución en la store: beneficios vs. restricciones	Balanceado	Balanceado	Balanceado
Rendimiento	Bajo	Bajo	Alto
Oportunidades de monetización	Altas	Altas	Bajas

Figura 2.2: Comparativa entre el desarrollo y resultado de una aplicación web, híbrida y nativa. Fuente: adaptación propia de [13]

***React-Native* es un *framework* para diseñar aplicaciones nativas usando *React*.** Con *React-Native* podemos crear simultáneamente aplicaciones nativas para *Android* e *iOS*. De manera oficial, para *Android* se programa en *Java*, mientras que para *iOS* se puede optar por *Swift* u *Objective-C*. El lenguaje que empleamos con *React-Native* es *JavaScript*. [22]

Cuando creamos un proyecto en *React-Native*, automáticamente se crean dos proyectos nativos en *Android* y en *iOS*. A partir de aquí, conforme desarrollamos nuestro código en *JavaScript* en el proyecto base, *React-Native* nos permite transpilar[8] el código a los otros dos lenguajes en sus correspondientes proyectos. Salvo ciertas configuraciones nativas, no necesitamos sumergirnos en el código nativo de cada proyecto. De este modo, logramos tardar el mismo tiempo que con el desarrollo híbrido y web pero obteniendo el mismo resultado que con el nativo. Además, aprovechamos todas las bondades que provee *React* para el diseño. [22]

Actualmente *React-Native* está siendo mantenido por *Facebook*.

2.2.3. Flux

***Flux* es un patrón de arquitectura concebido por *Facebook* para reemplazar el patrón MVC.** La idea es sustituir el flujo de datos bidireccional que tiene el patrón MVC (y sus derivados) por uno unidireccional. De ese modo, el control y manejo de los datos se simplifica, y tareas como la depuración se tornan más fáciles. [7] [5]

Flux define tres capas en su arquitectura. La **capa de presentación**, que presenta el estado de la aplicación. Engloba las vistas y componentes. La **capa de almacenamiento**, que contiene el estado y lo muta. Comprende los almacenes (*stores*). Viene a ser el modelo del MVC. La **capa creadora de acciones**, que crea y despacha acciones. Abarca el creador de acciones (*action creator*), el despachador de acciones (*dispatcher*) y las propias acciones. Viene a ser el controlador del MVC. [7] [19]

El flujo de los datos ocurre en un sentido. Cada ciclo podemos describirlo como sigue: (1)

[8]Generar a partir de código en un lenguaje código en otro lenguaje. No confundir con compilar, que se refiere a generar a partir de código en un lenguaje código ejecutable.

en la vista se activa un evento que realiza una llamada al *action creator*; (2) el *action creator* crea la acción correspondiente con su tipo y (opcionalmente) datos asociados; (3) el *dispatcher* propaga la acción a todas las *stores*; (4) las *stores* interesadas (por el tipo de acción) reaccionan mutando el estado; (5) el nuevo estado es notificado a la vista. En la Figura 2.3 mostramos el flujo de datos de forma gráfica. [5] [19]

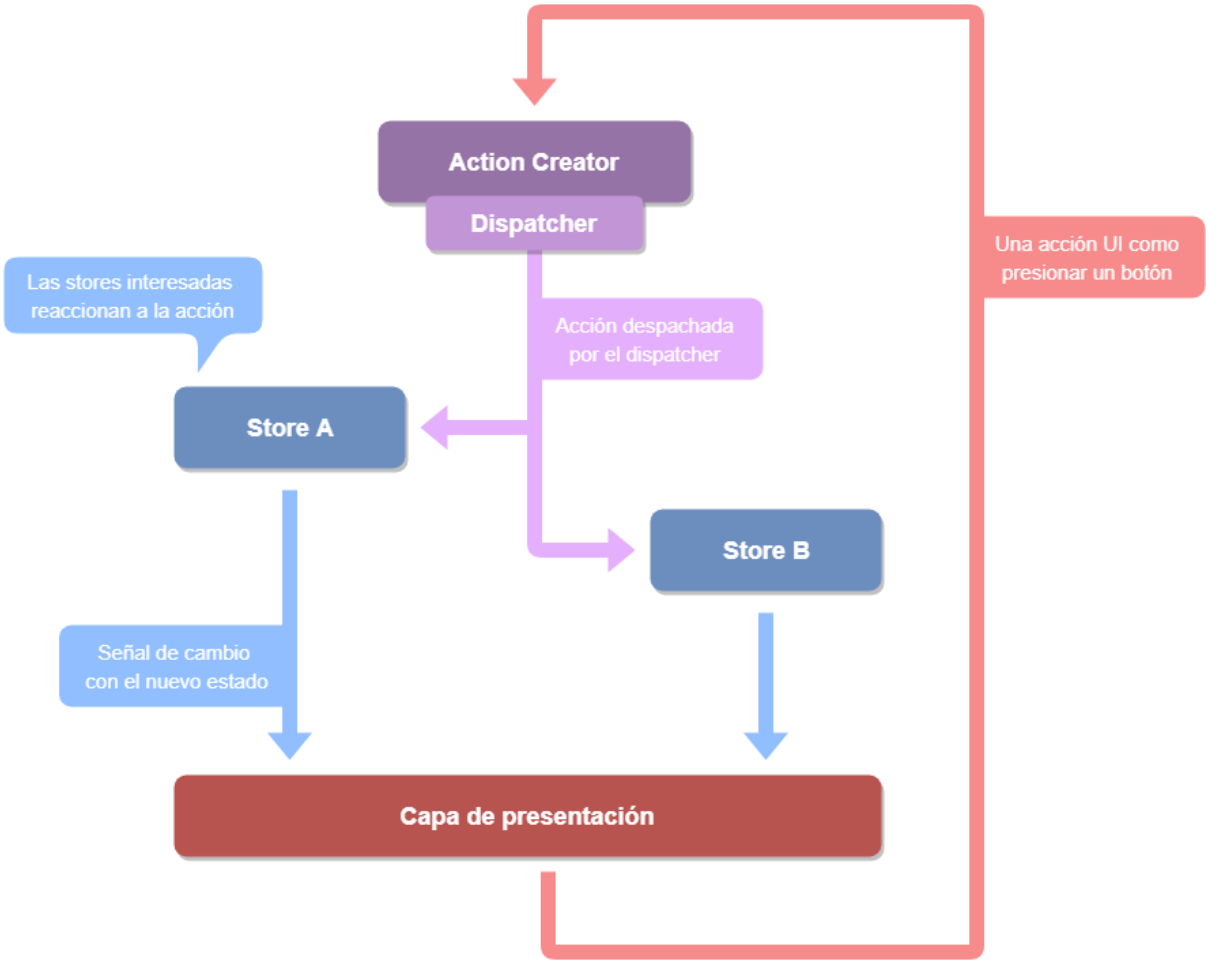


Figura 2.3: Estructura y flujo de datos de la arquitectura Flux. Fuente de la imagen: adaptación propia de [19]

El patrón *Flux* tiene muchas implementaciones. Hoy por hoy *Redux* es la más usada y se ha convertido prácticamente en un estándar. [5]

2.2.4. Redux

***Redux* es una biblioteca *JavaScript* que implementa el patrón *Flux* incorporando ciertos cambios**. Estos cambios se basan en tres principios. (1) Única fuente de la verdad: *Redux* no dispone de varios sino de un único *store*. (2) El estado es de sólo lectura: la única forma de modificar el estado es emitiendo una acción. (3) Los cambios se realizan con funciones puras:

los reductores (*reducers*) son funciones puras[9] que toman el estado anterior y una acción, y devuelven un nuevo estado. [18]

Es preciso remarcar que un reductor no devuelve el estado anterior modificado, sino uno nuevo. Podemos empezar con un único reductor que abarque todo el estado y, a medida que la aplicación vaya creciendo, dividirlo en reductores más pequeños que administren partes específicas. Entonces, el *store* aparentará estar dividido en subestados: uno por cada reductor. Pero seguirá siendo uno sólo, lo que facilita las labores de desarrollo y depuración. [18]

El objetivo final de *Redux* es desacoplar el estado de la aplicación de la parte visual. En lugar de que las vistas se pasen sus cambios las unas a las otras el *store* se encarga de gestionarlo. En consecuencia, la complejidad para cambiar las vistas se reduce enormemente, como podemos apreciar en la Figura 2.4. [18]

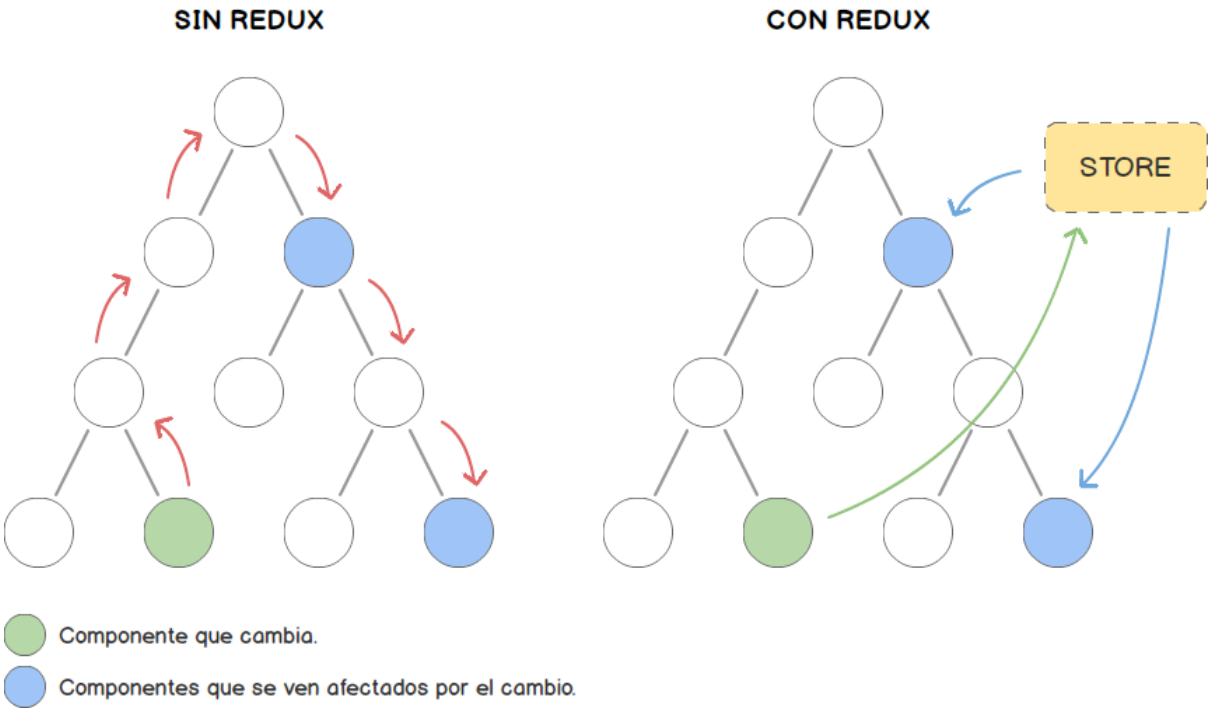


Figura 2.4: A la izquierda, sin *Redux*, un cambio en un componente que afecta a otros implica comunicarlo a través de otros; a la derecha, con *Redux*, todo se gestiona a través del *store*. Fuente de la imagen: adaptación propia de [15]

Aunque *Redux* resulta tedioso para proyectos pequeños, su éxito reside en lo bien que escala en aplicaciones grandes y complejas. Además, se integra perfectamente con *React* y puede ser extendido con funcionalidades personalizadas a través de *middlewares*. [18]

[9]Función que devuelve el mismo resultado ante los mismos datos de entrada.

2.2.5. Redux-Thunk

Cuando desarrollamos un cliente que está conectado con unos servicios web es imprescindible disponer de mecanismos asíncronos. La respuesta a una petición de un servicio web no es inmediata. Por ejemplo, si una vista realiza una llamada a un servicio que le debe proveer de unos datos, desde que se hace la petición hasta que llegan los datos a la vista ya le ha dado tiempo a dibujarse en pantalla. Para entonces, no hay datos que mostrar. Este es uno de los problemas de *Redux*, que no puede despachar acciones asíncronas, sino normales. [18] [15]

***Redux-Thunk* es un *middleware* para *Redux* que evita estos problemas de sincronía**. Permite escribir *actions creators* que devuelven una función en lugar de una acción. A esta función se le conoce como *thunk* y puede ser utilizada para retrasar el envío de una acción o despacharla sólo si se cumple una determinada condición. [1]

Por ejemplo, para obtener un listado de restaurantes, mostrarlo en una pantalla móvil y solucionar la asincronía crearíamos una función que hiciera lo siguiente: (1) enviase una acción de inicio que activase un icono de carga (*loading*) en la vista; (2) realizara la petición al servicio web; (3) en caso de éxito, enviara una acción de éxito con los datos para que la vista volviera a pintarse con ellos; (4) en caso de fracaso, enviara una acción de fracaso con el error para que la vista volviera a pintarse con un mensaje de fallo.

2.3. Herramientas, lenguajes y utilidades

El proyecto lo desarrollaremos principalmente con el IDE[10] *WebStorm*. De manera auxiliar, emplearemos los IDEs *Android Studio* y *XCode* para la configuración de bibliotecas nativas y los emuladores de *Android* e *iOS*. Como lenguaje utilizaremos *JavaScript* con el estándar *ECMAScript 6*. Para la gestión y seguimiento emplearemos la herramienta *Jira*. Opcionalmente, para escribir documentación emplearemos *Confluence*.

Asimismo, nos serviremos de varias utilidades para facilitarnos el desarrollo: *ESLint* para supervisar el código *JavaScript* en *WebStorm*; *Webpack* para compilar y ejecutar el proyecto en los emuladores y/o dispositivos; *Yarn* para añadir las bibliotecas a nuestro proyecto.

A continuación se detallan las herramientas, lenguajes y utilidades con los que partiremos inicialmente:

WebStorm	IDE multiplataforma basado en *IntelliJ*. Pertenece a la empresa *JetBrains*. Admite los lenguajes *JavaScript* y *TypeScript*. Permite el desarrollo de proyectos *React-Native* (*Android* e *iOS*).
Android Studio	IDE oficial para la plataforma *Android*. Está basado en *IntelliJ* y pertenece a la empresa *JetBrains*. Admite el lenguaje *Java*. Proporciona emuladores *Android*.

[10] Entorno de Desarrollo Integrado.

XCode IDE oficial para la plataforma *iOS*. Pertenece a la empresa *Apple*. Admite los lenguajes *Objective-C* y *Swift*. Proporciona emuladores *iOS*.

JavaScript Lenguaje de programación interpretado. Orientado a objetos, basado en prototipos, imperativo, sin tipos y dinámico. Actualmente sigue el estándar *ECMAScript 6*.

Yarn Gestor de paquetes rápido, seguro y fiable. Bastante más rápido que otros gestores como *NPM*. [27]

ESLint Supervisor de escritura *JavaScript*. Analiza estáticamente el código e indica patrones erróneos y/o código que no se adhiere a ciertas pautas de estilo. Las reglas de escritura que emplea son totalmente modificables. Es de código abierto. [6]

Webpack Empaquetador (*bundler*) que separa el código en módulos (paquetes) que sirven como dependencias para otros módulos y que se encarga de gestionar (carga bajo demanda). Reduce los tiempos de compilación: sólo recompila los módulos modificados. Útil para el desarrollo de aplicaciones *front-end* con filosofía modular. [23] [2]

Jira Herramienta en línea para la gestión de proyectos. Desarrollada y mantenida por *Atlassian*. Se ajusta perfectamente a proyectos pensados con una metodología ágil. Entre otras labores administrativas, permite: crear historias de usuario, estimarlas en puntos de historia, dividirlas en tareas, agruparlas en sprints y hacer un seguimiento con gráficas de tipo *burn-down*. [4]

Confluence Herramienta en línea para la colaboración en equipo. Desarrollada y mantenida por *Atlassian*. Se emplea para la documentación de proyectos. Complementa a *Jira*: permite asociar documentación a un proyecto. [3]

Balsamiq Mockup Herramienta en línea que facilita y agiliza la realización de bocetos (*mockups*). Estos bocetos son los prototipos gráficos que se emplean para la implementación de pantallas en proyectos *front-end*. La herramienta está desarrollada y mantenida por *Balsamiq*.

Para los IDEs *WebStorm* y *Android Studio* contaremos con una licencia *JetBrains* gratuita de estudiante de la UJI; el IDE *XCode* no necesita licencia. Las utilidades *Yarn*, *ESLint* y *Webpack* son totalmente gratuitas. Las herramientas *Jira* y *Confluence* nos las proporciona Cuatroochenta; sin embargo, para una total independencia sobre la gestión con Jira obtendremos una licencia *Atlassian*[11]. Finalmente, para la herramienta *Balsamiq Mockup* emplearemos una licencia *Balsamiq* gratuita de estudiante de la UJI.

Durante el desarrollo se incorporarán nuevas herramientas, lenguajes y utilidades que en un principio no estaban planteadas. Entre ellas destacamos:

Realtime Database *Firebase* es una plataforma de desarrollo de aplicaciones web y móviles perteneciente a *Google*. Cuenta con varios servicios de análisis y desarro-

[11] La versión Jira de Cuatroochenta está muy limitada a su forma de trabajar. Para seguir lo máximo posible un proyecto *Scrum* es necesario obtener una licencia aparte.

llo, entre ellos *Realtime Database*. *Realtime Database* proporciona una base de datos NoSQL alojada en la nube la cual puede ser accedida y actualizada por las aplicaciones en tiempo real. Puede estar integrada dentro de la aplicación o accedida a través de una API REST. Los datos se almacena en formato JSON. Para funcionar con *React-Native* requiere la siguiente biblioteca: *react-native-firebase*[12]. [10]

OneSignal Plataforma que permite crear y enviar notificaciones *push*[13] a todos aquellos dispositivos que estén debidamente registrados. Los dispositivos tienen que ser reales, no sirve que sean virtuales como los emuladores. Para funcionar con *React-Native* requiere la siguiente biblioteca: *react-native-onesignal*. [16]

Google Analytics Herramienta de analítica web de *Google*. Ofrece información agrupada del tráfico que llega a la aplicación o sitio web según la audiencia, la adquisición, el comportamiento y las conversiones que se llevan a cabo. Para funcionar con *React-Native* requiere la siguiente biblioteca: *react-native-google-analytics-bridge*. [9]

Postman Herramienta que permite realizar pruebas contra una API que cumpla con los protocolos REST mediante el envío de peticiones[14]. Las peticiones se pueden almacenar en colecciones. *Postman* tiene tanto una versión web como una para escritorio. [17]

Reactotron Herramienta de depuración visual para proyectos *React-Native*. Permite hacer un seguimiento de los errores y desglosar su contenido. Facilita enormemente la detección de errores y su posterior corrección. [20]

TypeScript Versión de *JavaScript* con tipos. Ayuda con la depuración de errores. Recomendable para el desarrollo de proyectos grandes.

[12]Existe como mínimo otra biblioteca con la que *Firebase* puede funcionar con *React-Native*, pero el rendimiento no es tan bueno. De hecho, a principio del desarrollo empleábamos dicha biblioteca y más tarde se optó por cambiar a *react-native-firebase* por sus mejores resultados.

[13]Las notificaciones *push* son aquellas que avisan a un dispositivo de que algo ha ocurrido en la aplicación a la vez que le pasan información. Por contra, las notificaciones *pull* avisan pero no pasan información, sino que una vez llega el aviso es el dispositivo el que debe acceder a la aplicación y pedir la información correspondiente. *OneSignal* sólo utiliza notificaciones *push*.

[14]Estas peticiones se envían a las direcciones web de los servicios que conforman dicha API. Por ejemplo, enviar a la dirección del servicio de *login* una petición que contenga el correo electrónico y la contraseña del usuario para comprobar si se recibe un credencial de que se ha autenticado o se recibe un error de que no existe dicho usuario (o de que directamente el servicio no funciona).

Capítulo 3

Desarrollo técnico del proyecto

3.1. *Scrum* como metodología

Cuatroochenta esta dividida en varios departamentos. Cuenta con un departamento de gestión y testeo, un departamento de diseño, un departamento de desarrollo web y un departamento de desarrollo móvil, entre otros.

Cuando un cliente les solicita una aplicación, el departamento de gestión se reúne con él para acordar un precio cerrado y los plazos de entrega. A partir de aquí, este departamento crea una serie de incidencias a través de *Jira* que el resto de ellos va completando en orden: (1) el departamento de diseño realiza los prototipos; (2) los departamentos de desarrollo, la implementación; (3) el departamento de testeo, las pruebas. Como la sala donde se encuentran los departamentos es totalmente diáfana, la comunicación entre ellos es directa. Además, el departamento de gestión planifica reuniones periódicas para saber cómo van los proyectos, qué problemas hay, etc. Al final de cada plazo lanzan un incremento del producto, que internamente denominan "Beta".

Cuatroochenta da total libertad a la hora de escoger la metodología que el alumno de prácticas prefiera y/o mejor se le ajuste. No tiene por qué concordar con la que emplean. En ese sentido, la metodología que hemos escogido para este proyecto es *Scrum*. En el Apéndice A hacemos un breve resumen de la metodología.

Scrum difiere con la forma de trabajar de Cuatroochenta. Por ejemplo, no llevan a cabo reuniones diarias. Tampoco se reúnen constantemente con los clientes. Han adaptado la metodología a sus intereses. De igual modo, adaptaremos *Scrum* a los nuestros.

Sergio Aguado, CTO de Cuatroochenta, tomará el papel de cliente. En una primera reunión especificará los requisitos del sistema. El alumno encarnará los roles de propietario del producto y desarrollador. No será necesario el *Scrum Master* puesto que no habrá reuniones diarias. Tampoco habrá retrospectivas del sprint. Como existe un tiempo definido para la estancia en prácticas (300 horas), en principio habrá un total de cuatro sprints de dos semanas cada uno. Las revisiones del sprint se harán con Sergio.

3.2. Planificación previa

En una planificación inicial, el proyecto lo hemos dividimos en tres fases: (1) el desarrollo de la propuesta técnica, (2) el desarrollo técnico del proyecto y (3) la documentación y presentación del TFG. En la Figura 3.1 mostramos la descomposición del trabajo de cada una de ellas.

Nº	Tareas	Tiempo (h.)	Dependencias
	Meals On Wheels	**443**	
1	**Desarrollo de la propuesta técnica**	**8**	
1.1	**Primera visita**	**1**	
1.1.1	Definir proyecto con el tutor y el supervisor.	0,5	-
1.1.2	Definir método de trabajo.	0,5	1.1.1
1.2	**Redacción de la propuesta técnica.**	**7**	
1.2.1	Definir objetivos.	1	1.1.1
1.2.2	Definir metodología.	2	1.1.1
1.2.3	Definir planificación + EDT.	2	1.1.1
1.2.4	Definir tecnologías y herramientas.	2	1.1.1
1.2.5	Entregar la Propuesta técnica.	0	1.2.1, 1.2.2, 1.2.3, 1.2.4
2	**Desarrollo técnico del proyecto**	**300**	
2.1	**Formación**	**50**	
2.1.1	Visionar videotutorial.	50	1.2.5
2.2	**Desarrollo Scrum**	**250**	
2.2.1	Reunión con el cliente.	1	2.1.1
2.2.2	Definir pila del producto.	9	2.2.1
2.2.3	Sprint 1	60	2.2.2
2.2.4	Sprint 2	60	2.2.3
2.2.5	Sprint 3	60	2.2.4
2.2.6	Sprint 4	60	2.2.5
2.2.7	Presentación producto al cliente.	0	2.2.6
3	**Documentación y presentación del TFG**	**135**	
3.1	Redactar los informes quincenales.	4	1.2.5
3.2	Redactar la memoria técnica.	100	1.2.5
3.3	Entrega de la memoria técnica.	0	3.2
3.4	Preparar la presentación oral.	30	3.3
3.5	Presentación oral.	1	3.4

Figura 3.1: Estructura de descomposición de trabajo (EDT) inicial.

Dentro de esta planificación, la fase de desarrollo técnico del proyecto cobra especial importancia. Es la fase que llevaremos a cabo durante la estancia en prácticas. Su duración es de un máximo de 300 horas. En ella aplicaremos una metodología *Scrum* y la gestionaremos de manera independiente a las otras dos fases. Esto significa que la planificaremos de distinta

manera a cómo la hemos planificado inicialmente en la Figura 3.1 a través de *Jira*.

En Jira partiremos de una plantilla de proyecto *Scrum* que nos permitirá crear cinco tipos de incidencias: tarea, historia, épica, error y subtarea[1]. La formación la consideraremos como una tarea y los sprints vendrán definidos por las historias de usuario de la pila del producto. La pila del producto no tiene por qué ser la misma al igual que los sprints no tienen que ser cuatro ni tener la misma duración, como bien veremos más adelante. En el caso de que surjan nuevas funcionalidades las añadiremos como historias en la pila; y si surgen otro tipo de labores las añadiremos como tareas.

Cada historia constará en su descripción de la definición de hecho (*definition of done*) y de los criterios de aceptación (*acceptance criteria*). La definición de hecho es un conjunto de labores comunes a todas las historias que deberán estar completadas para considerar que una historia lo está a su vez. Estas labores se resumen en tres: implementada, revisada (*Android* e *iOS*) y documentada. Para que una historia esté implementada y revisada deberán cumplirse los criterios de aceptación. Los criterios de aceptación sí son diferentes para cada historia.

Las historias de usuario las estimaremos en puntos de historia. Los puntos de historia seguirán una escala basada en la sucesión de Fibonacci. Las estimaremos *a priori* según sus criterios de aceptación y las subtareas que pudieran tener.

Las historias las desglosaremos en tres tipos de subtarea: prototipos, servicios y componentes. El tipo prototipos representará a las subtareas de diseño gráfico de pantallas (*mockups*). El tipo servicios representará a las subtareas de implementación para conectar los servicios web con la aplicación. El tipo componentes representará a las subtareas de implementación de las pantallas y su lógica, si la hubiera. Indicaremos los tipos mediante etiquetas (*tags*). Las estimaremos en tiempo.

Yo seré tanto el responsable como el informador de todas y cada una de las incidencias. En la Figura 3.4 se muestra un ejemplo de la pila del producto inicial con la tarea de formación y las historias de usuario refinadas a partir de los requisitos del cliente (Sergio Aguado).

3.3. Arquitectura del sistema

Definiremos la arquitectura del sistema en función a dos tipos: la **arquitectura física** y la **arquitectura lógica**.

Como física el sistema responde a una **arquitectura cliente-servidor**. Los dispositivos móviles o computadores acceden a través de una API REST a los servicios web del servidor. Por ser REST, el protocolo empleado en las comunicaciones es HTTP sin estado con sus cuatro operaciones básicas: *POST* (crear), *GET* (leer y consultar), *PUT* (editar) y *DELETE* (eliminar). Además, se utiliza JSON para el formato de los datos de las operaciones.

Como lógica el sistema responde a una **arquitectura de tres capas**: capa de presentación,

[1]Las subtareas sólo pueden crearse a partir de una historia de usuario.

lógica de negocio y capa de datos. La capa de datos comprende tanto la base de datos como el acceso. La capa de presentación está alojada en la parte cliente, mientras que la lógica de negocio y la capa de datos, en la parte servidor.

En la Figura 3.2 mostramos un mapeado físico-lógico de ambas arquitecturas. Además, ampliamos la información mostrando la arquitectura Flux que implementaremos en el cliente móvil, la cual está compuesta por acciones, reductores y componentes. Como se muestra, a través de las acciones realizamos las llamadas a la API, mientras que el usuario interactúa directamente con los componentes reactivos.

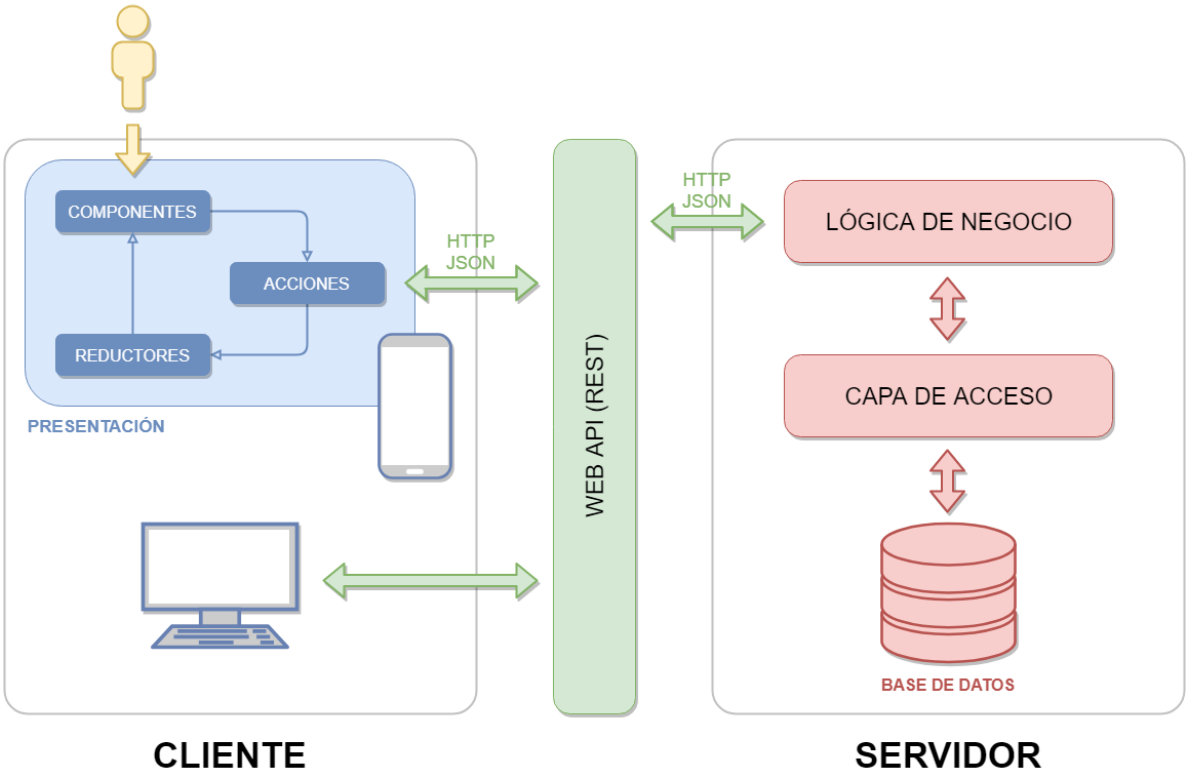


Figura 3.2: Arquitectura lógico-física del sistema completo. Consiste en una arquitectura de tres capas con una arquitectura física cliente-servidor: presentación (cliente) y negocio y acceso (servidor).

Como ya hemos comentado, Roberto Bernad se encarga de la parte servidor y de la aplicación web, y yo de la aplicación móvil. En consecuencia, inicialmente la arquitectura de todo el sistema es como la Figura 3.2. Sin embargo, al comienzo del desarrollo tomaremos la decisión de desarrollar la parte servidor por nuestra cuenta.

Para ello utilizaremos el servicio *Realtime Database* de *Firebase*. En la Figura 3.3 mostramos la nueva arquitectura. La base de datos será sustituida por otra NoSQL almacenada en la nube, y parte de la lógica de los servicios se incluirá en las acciones, reductores y componentes de la aplicación. De este modo, pasamos de tener un **arquitectura de dos capas** con la lógica de negocio y la capa de presentación juntas.

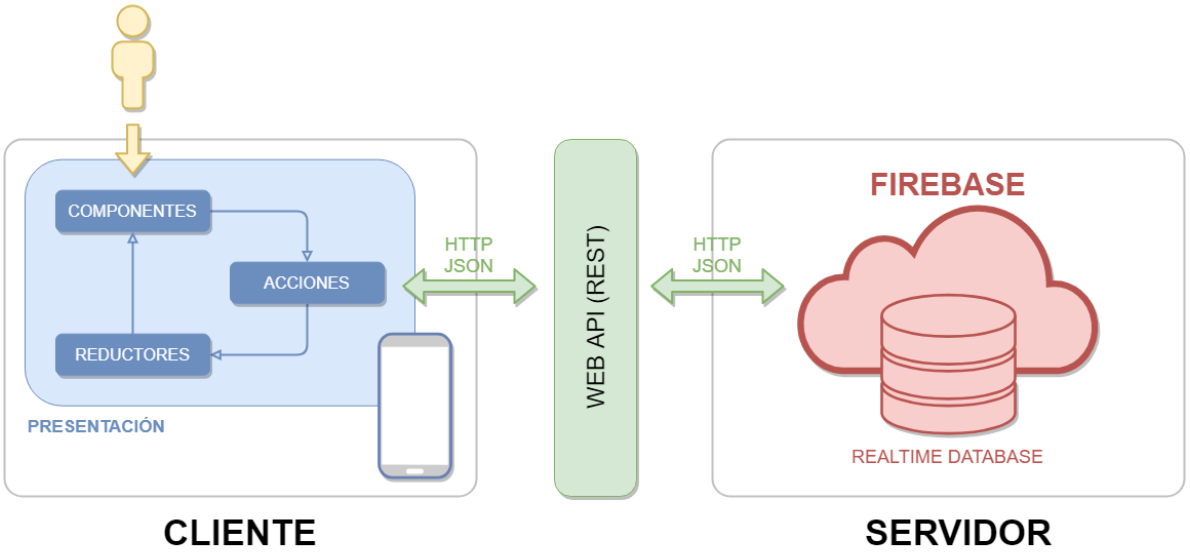


Figura 3.3: Arquitectura lógico-física provisional del proyecto. Consiste en una arquitectura de dos capas con una arquitectura física cliente-servidor: presentación junto con lógica de negocio (cliente) y acceso (servidor) .

3.4. Estimación de recursos y costes del proyecto

Para la estimación de recursos y el coste del proyecto emplearemos COCOMO[2]. Dentro de COCOMO existen tres modelos: básico, intermedio y detallado. Nosotros emplearemos el básico, que es el modelo más rápido pero menos preciso.

Para el modelo básico de COCOMO hay que diferenciar entre tres tipos de proyectos:

Proyecto Orgánico	Entre unas pocas miles y decenas de miles de líneas. Pocas restricciones. Equipos con mucha experiencia.
Proyecto Medio	Entre unas decenas y cientos de miles de líneas. Equipos con personas de mucha y poca experiencia.
Proyecto Empotrado	Entre unas decenas y cientos de miles de líneas. Fuertes restricciones (sobretodo con el *hardware*). Problema a resolver único y, en consecuencia, no se pueden basar en la experiencia.

Las fórmulas a aplicar son:

$$E = a \times KLOC^b$$

$$D = c \times E^d$$

$$P = E \ / \ D$$

[2]Modelo Constructivo de Costes (*COnstructive COst MOdel*) basado en líneas de código.

siendo:

KLOC	El número de líneas estimadas para el proyecto (en miles).
E	El esfuerzo aplicado en persona-mes.
D	El tiempo de desarrollo en meses.
P	El número de personas necesarias.
a, b, c, d	Coeficientes en función del tipo de proyecto.

Los valores de los coeficientes en función del tipo de proyecto están especificados en la Tabla 3.1.

	a	b	c	d
Proyectos Orgánicos	2,40	1,05	2,50	0,38
Proyectos Medios	3,00	1,12	2,50	0,35
Proyectos Empotrados	3,60	1,20	2,50	0,32

Cuadro 3.1: Tipos de proyecto según COCOMO básico.

Para nuestro caso en concreto, se trata de un proyecto de tipo orgánico, con pocas restricciones y de un tamaño pequeño (de unas séis mil líneas aproximadamente)[3]. Por tanto:

$$E = 2.40 \times 6^{1},05 = 15,750 \quad \text{persona-mes}$$

$$D = 2.50 \times 15,750^{d} = 7,127 \quad \text{meses}$$

$$P = 15.750 \ / \ 7,127 = 2,210 \quad \text{personas}$$

La duración será de unos siete meses con un mínimo de tres personas. Sin embargo, la estancia en prácticas es de 300 horas. Suponiendo que estamos contratados por la empresa a jornada completa, significa que trabajaremos 40 horas semanales, es decir, 160 horas al mes. La estancia supondría 1,875 meses. Por tanto:

$$P = 15,750 \ / \ 1,875 = 8,400 \quad personas$$

Eso son ocho personas a jornada completa y una a media (8,5 a efectos prácticos).

[3]Para el conteo se ha empleado un *plugin* del IDE *WebStorm*. No se han contado ni líneas en blanco ni líneas de comentario. El resultado ha sido de unas 6.015 líneas de código.

	Año	Mes	Hora
Project Manager	35.036,00 €	2.920,00 €	20,00 €
Analista	26.261,00 €	2.188,00 €	15,00 €
Programador Senior	24.116,00 €	2.010,00 €	14,00 €
Programador Junior	12.000,00 €	1.000,00 €	7,00 €

Cuadro 3.2: Salarios brutos medios.

Teniendo en cuenta los salarios de la Tabla 3.2, el salario medio al mes entre los cuatro roles es de unos 2.029,50 €. Tomando este salario medio como referencia, el coste de los recursos humanos es de:

$$Coste\ RRHH = 2.029,50\ /\ 7,127 \quad = 17.250,75\ €/\text{mes}$$

$$Coste\ RRHH = 17.250,75 \times 1,875 = 32.345,16\ €$$

Contando con un 30 % de impuestos por los trabajadores:

$$Coste\ RRHH = 32.345,16 \times 1,30 = 42.048,70\ €$$

Herramienta / Material	Coste	
Licencia *WebStorm*	0,00 €	(Licencia *JetBrains* gratuita de estudiante UJI)
Licencia *Android Studio*	0,00 €	(Licencia *JetBrains* gratuita de estudiante UJI)
Licencia *XCode*	0,00 €	(Gratuita)
Licencia *Balsamiq Mockup*	0,00 €	(Licencia *Balsamiq* gratuita estudiante UJI)
Licencia *Jira*	10,00 €	(Licencia *Atlassian* anual con servidor propio)
Dispositivo móvil (*Android*)	200,00 €	(Móvil personal)
Ordenador personal	0,00 €	(Proporcionado por Cuatroochenta)
Material de oficina (silla, mesa, etc)	0,00 €	(Proporcionado por Cuatroochenta)

Cuadro 3.3: Coste de materiales y herramientas.

Si el coste total de las herramientas y materiales es de 210,00 €, entonces:

$$Coste\ Total = 42.048,70 + 210,00 = 42.258,70\ €$$

Finalmente, con un coste extra de luz, agua, etc, entorno al 20 %, el coste total es de:

$$Coste\ Total = 42.258,70 \times 1,20 = 50.710,44\ €$$

Matización respecto al coste final con COCOMO

El modelo COCOMO, especialmente el básico, no es el mejor modelo para la estimación. De hecho, es un modelo que se ha quedado un tanto obsoleto. Ello se debe a que la medida que emplea, número de líneas de código, no es la más acertada. Aunque en este proyecto hemos evitado contar las líneas en blanco y las líneas de comentario, no hay forma de contar código duplicado (estructuras que se repiten a lo largo del proyecto) o de tener en cuenta el estilo de programación (línea de nueve palabras que supera los 80 caracteres y ha sido dividida en nueve líneas de una palabra cada una). En consecuencia, el coste total para un proyecto tan pequeño es altísimo.

Hay otros modelos más precisos, como Albretch o MARK II, los cuales se centran en puntos de función, pero que son más complejos de utilizar. Además, no van bien para proyectos de tan poca envergadura.

Con el fin de comparar resultados, hemos vuelto a realizar los cálculos para otros dos casos distintos: (1) suponiendo que el código real es un 50 % del obtenido, es decir, unas tres mil líneas, y (2) obviando el número de líneas y teniendo en cuenta que el proyecto lo hace una única persona. En la Tabla 3.4 mostramos directamente los resultados.

	6k líneas	3k líneas	1 persona
Personas	8,5	4	1
Meses	1,875	1,875	1,875
Coste RRHH	32.345,16 €	15.221,25 €	3.805,31 €
con 30 % de impuestos	42.048,70 €	19.787,63 €	4.946,91 €
Costes materiales	210,00 €	210,00 €	210,00 €
Coste total	42.258,70 €	19.997,63 €	5.156,91 €
con 20 % coste extra	50.710,44 €	23.997,15 €	6.188,29 €

Cuadro 3.4: Tabla comparativa de los costes en los distintos casos.

A la vista de los resultados y siendo realistas, si tuviéramos que vender nuestro producto a un cliente, el precio final de una persona sería el más razonable, o como mucho el de tres mil líneas; pero no el de seis mil líneas.

3.5. Pila del producto

Durante la primera reunión con el cliente hemos recopilado una serie de requisitos. En éstos especificamos que la aplicación permita: listar los restaurantes de reparto a domicilio de Castellón de la Plana; filtrarlos por tipo (mexicano, italiano, etc); seleccionarlos y ver sus listas de productos categorizados (entrantes, bebidas, pizzas, etc); añadir y quitar sus productos a un pedido; realizar el pedido indicando una dirección de entrega; y guardar una dirección como

predeterminada siendo un usuario registrado.

Dichos requisitos los hemos refinado en historias de usuario. Estas historias forman la pila del producto inicial y son las que siguen:[4]

MOW-2 Como usuario quiero obtener un listado de todos los restaurantes de reparto a domicilio para saber qué opciones hay a la hora de hacer un pedido.

MOW-3 Como usuario quiero filtrar los restaurantes del listado por tipo (mexicano, chino, italiano, etc) para encontrar rápidamente aquellos correspondientes a un tipo en concreto.

MOW-4 Como usuario quiero acceder a la información de un restaurante al seleccionarlo del listado para conocer el nombre, tipo (o tipos), logotipo y descripción.

MOW-5 Como usuario quiero obtener un listado de productos categorizados al seleccionar un restaurante para saber qué productos tiene.

MOW-6 Como usuario quiero disponer de una cesta de productos (pedido) para poder añadir los distintos productos de un restaurante.

MOW-7 Como usuario quiero poder añadir y eliminar los productos de un determinado restaurante en mi pedido para tener un control de lo que voy a pedir.

MOW-8 Como usuario quiero efectuar mi pedido indicando mi dirección para que me hagan la entrega de la comida en mi casa.

MOW-9 Como usuario quiero que me muestren un resumen de mi pedido al efectuarlo para tener constancia de lo que he pedido.

MOW-10 Como usuario quiero acceder (*login*) en el sistema para acceder a una cuenta de usuario y facilitarme ciertas operaciones como indicar mi dirección de manera automática.

MOW-11 Como usuario quiero darme de alta en el sistema (crear una cuenta de usuario) para almacenar datos personales e información sobre mis pedidos, direcciones, etc. que me faciliten ciertas operaciones.

MOW-12 Como usuario quiero almacenar y modificar mi dirección de entrega para no tener que introducirla cada vez que realizo un pedido.

En la Figura 3.4 las mostramos priorizadas, estimadas y asignadas. Priorizadas según las preferencias del cliente, estimadas según la dificultad que a priori van a tener, y asignadas al único integrante del equipo de desarrollo. Son once historias que suman un total de 99 puntos de historia.

A mitad de desarrollo, con la mayor parte de las historias de usuario completadas, hemos planteado nuevas historias de usuario. Estas historias surgen de las últimas revisiones de sprint con el cliente y de mejoras propuestas por mis compañeros de departamento. Son las que siguen:

[4]El código mostrado a la izquierda es el mismo que el asignado automáticamente en el Jira, de ahí que no haya una correlación numérica.

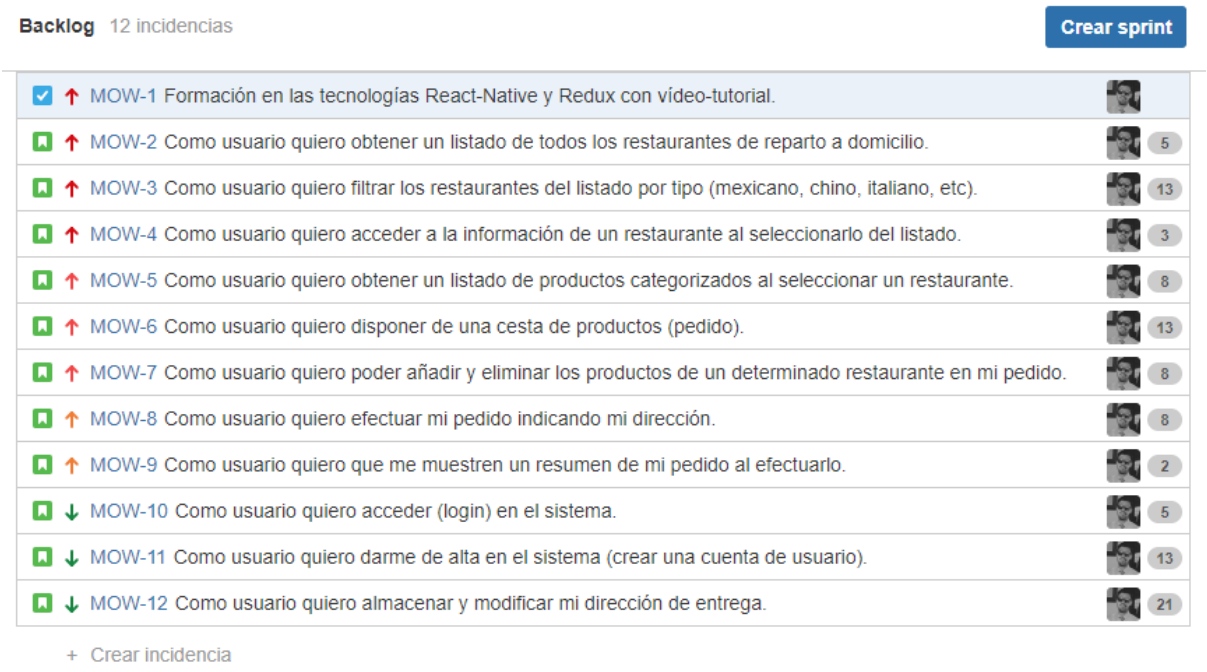


Figura 3.4: Pila del producto inicial con las primeras historias de usuario y la tarea de formación (Jira).

MOW-69 Como usuario quiero mantener más de una dirección de entrega para poder escoger entre ellas a la hora de realizar un pedido.

MOW-70 Como usuario quiero almacenar todos mis pedidos realizados para poder escoger entre ellas a la hora de realizar un pedido.

MOW-71 Como usuario quiero tener la opción de visualizar mi contraseña a la hora de acceder y registrarme en el sistema para saber con certeza qué contraseña estoy escribiendo.

MOW-72 Como usuario quiero recibir una notificación cuando un pedido está a punto de llegar a casa para preparar el pago.

MOW-73 Como usuario quiero recibir una notificación cuando un nuevo restaurante es incluido en el sistema para enterarme al momento de las novedades.

MOW-74 Como administrador quiero obtener diversos datos de los usuarios como el acceso a ciertos componentes para realizar informes que me ayuden a mejorar el sistema.

MOW-75 Como usuario quiero poder seleccionar el idioma de la aplicación para mi comodidad y entendimiento.

MOW-76 Como usuario quiero visualizar los restaurantes en un mapa con sus ubicaciones exactas para saber qué restaurantes tengo más cerca y cómo llegar a ellos.

En la Figura 3.5 las mostramos priorizadas, estimadas y asignadas. Son nueve historias nuevas que suman un total de 71 puntos de historia.

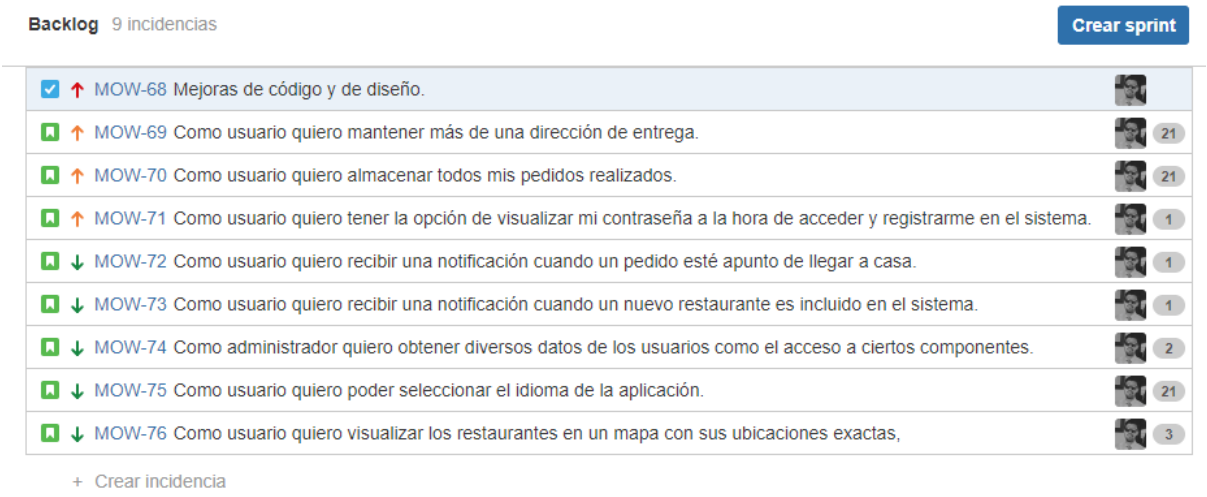


Figura 3.5: Pila del producto con las nuevas historias y la tarea de mejora de código y diseño (Jira).

3.6. Desarrollo de los sprints

En esta sección desarrollamos gran parte del proyecto. La explicación del análisis, diseño, implementación, verificación y validación, y seguimiento del proyecto está estructurada por sprints. El hecho de seguir esta estructura es que es más natural a la filosofía *Scrum* y, al ser cronológica, facilita la comprensión.

Contamos con una primera etapa de formación y planificación previa a los sprints donde se crea la pila inicial del producto. A partir de aquí, desarrollamos los sprint en orden cronológico con varios apartados: planificación, diseño, implementación, criterios y seguimiento.

El apartado de planificación consta de las historias de usuario escogidas y la descomposición en subtareas y estimaciones de tiempo en minutos. También se muestra el estado inicial de la pila del producto en *Jira*. Esta fase se correspondería con la fase de análisis.

El apartado de diseño incluye los bocetos (*mockups*) de las pantallas en diferentes estados: normal, de éxito, de fallo, etc. Para indicar qué pantallas corresponden a qué historias de usuario empleamos *post-its*. Un *post-it* indica que esa pantalla y las siguientes hasta el próximo *post-it* pertenecen a una misma historia de usuario. Todo el apartado de diseño lo hemos realizado con la herramienta *Balsamiq Mockup*.

El apartado de detalles de implementación contiene detalles y explicaciones además del esquema de navegación entre pantallas. Partimos de la base de que todo son componentes, y que se van anidando entre ellos para formar otros componentes: botones, paneles, secciones, modales, etc. Por ello, este apartado está enfocado a las pantallas, que son los componentes que sobresalen del resto.

El apartado de criterios de aceptación se corresponde con la validación. Incluye todos los criterios de las historias de usuario escogidas que se deben cumplir para considerarlas completadas. En el Capítulo 4 hablaremos tanto de la validación como de la verificación.

Por último, el apartado de seguimiento abarca las actividades llevadas a cabo en cada sprint y la gráfica de puntos de historia completados. Además, se presentan los problemas acaecidos y las medidas correctoras tomadas.

3.6.1. Formación y planificación

Esta etapa, que no es un sprint, tiene como meta **la formación en las tecnologías *React-Native* y *Redux* y la definición de la pila del producto**. Detallamos sus principales características a continuación:

Intervalo de tiempo	12 de febrero - 25 de febrero (1ª Quincena)
Horas de prácticas	56,5 horas (9 jornadas)
Incidencias incluidas	MOW-1 (Tarea)
Incidencias completadas	MOW-1 (Tarea)
Estimación total	21 puntos de historia (completados: 8)
Velocidad	-

1. Visionado del vídeo-tutorial *The Complete React Native and Redux Course*.
2. Reunión con el cliente (Sergio Aguado) para concretar los requisitos del sistema.
3. Creación del proyecto en Jira.
4. Definición de la pila del producto.

3.6.2. Sprint 1: Listado, filtrado y acceso a restaurantes.

Este sprint tiene como meta **el listado, filtrado por tipo y acceso a los restaurantes de reparto de comida a domicilio**. Detallamos sus principales características a continuación:

Intervalo de tiempo	26 de febrero - 11 de marzo (2ª Quincena)
Horas de prácticas	35 horas (6 jornadas)
Incidencias incluidas	MOW-2, MOW-3, MOW-4
Incidencias completadas	MOW-2, MOW-4
Estimación total	21 puntos de historia (completados: 8)
Velocidad	9,143 puntos/semana

PLANIFICACIÓN DEL SPRINT

Las historias de usuario escogidas para este sprint son las que siguen:

MOW-2 Como usuario quiero obtener un listado de todos los restaurantes de reparto a domicilio para saber qué opciones hay a la hora de hacer un pedido.

MOW-3 Como usuario quiero filtrar los restaurantes del listado por tipo (mexicano, chino, italiano, etc) para encontrar rápidamente aquellos correspondientes a un tipo en concreto.

MOW-4 Como usuario quiero acceder a la información de un restaurante al seleccionarlo del listado para conocer el nombre, tipo (o tipos), logotipo y descripción.

La Figura 3.6 muestra la planificación del primer sprint y el estado de la pila del producto previo al inicio del sprint. Una vez seleccionadas las historias de usuario, se descomponen en subtareas y se estiman en minutos. La Figura 3.7 muestra la descomposición en subtareas (las historias están estimadas en puntos de historia).

DISEÑO DE PROTOTIPOS

El diseño de la historia MOW-2 consta de un boceto (Figura 3.8) con el listado de restaurantes. El diseño de la historia MOW-3 consta de cuatro bocetos (Figuras 3.8 y 3.9) con una pequeña modificación en el anterior boceto, el modal de filtrado y los resultados en caso de que existan restaurantes y no existan (diferentes estados de una misma pantalla). Finalmente, el diseño de la historia MOW-4 consta de un boceto (Figura 3.9) con la información del restaurante.

DETALLES DE IMPLEMENTACIÓN

Listado de restaurantes

Al iniciar la aplicación accedemos directamente a la pantalla del listado de restaurantes. Esta pantalla consta de dos componentes principales: la cabecera y la lista.

La cabecera la importamos de la biblioteca *react-native-router-flux*. Esta biblioteca es externa, por lo que hay que instalarla con *Yarn*. La cabecera está conformada por un título 'Restaurantes' y un botón para acceder al filtrado. El botón es una propiedad de la cabecera, por lo que no hay necesidad de implementarlo. Sin embargo, es complicado retocar su diseño y podemos perder la consistencia del diseño en un futuro.

La lista es un componente propio de *React-Native*, de hecho pertenece a la biblioteca *react-native*. La lista, a su vez, emplea un componente implementado a propósito para mostrar la imagen, el nombre y el tipo de un restaurante. Además permite acceder a la pantalla de información del restaurante.

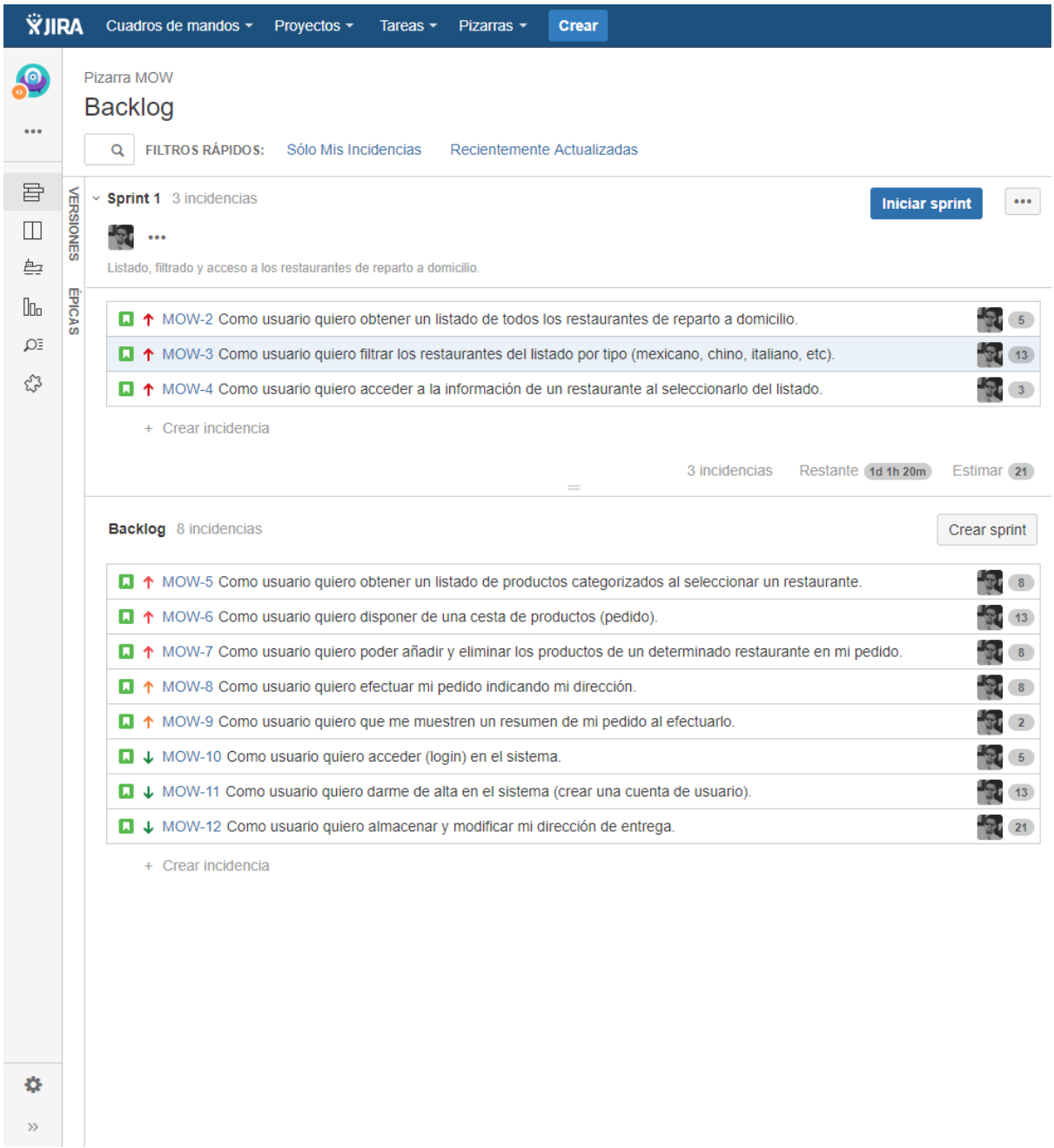


Figura 3.6: Planificación del primer sprint y estado de la pila del producto (Jira).

ID	EST	INCIDENCIAS	
MOW-2	**5**	**Como usuario quiero obtener un listado de todos los restaurantes de reparto a domicilio.**	
MOW-13	20	**Prototipos**	Realizar los mockups de las pantallas para el listado de restaurantes.
MOW-14	60	**Firebase**	Configurar un proyecto en Firebase y enlazarlo con el sistema.
MOW-15	20	**Firebase**	Diseñar la base de datos para almacenar los restaurantes (nombre, tipo e imagen).
MOW-16	120	**Servicios**	Implementar el servicio para obtener los restaurantes.
MOW-17	240	**Componentes**	Implementar la pantalla de restaurantes con la lógica de listado.
MOW-3	**13**	**Como usuario quiero filtrar los restaurantes del listado por tipo (mexicano, chino, italiano, etc).**	
MOW-18	80	**Prototipos**	Realizar los mockups de las pantallas para el filtrado de restaurantes.
MOW-19	20	**Firebase**	Modificar la base de datos para almacenar los tipos de restaurante.
MOW-20	120	**Servicios**	Implementar el servicio para obtener los tipos de restaurante.
MOW-21	120	**Servicios**	Implementar el servicio para obtener los restaurantes filtrados por tipo.
MOW-22	60	**Componentes**	Modificar la pantalla de restaurantes con el botón de filtrado.
MOW-23	240	**Componentes**	Implementar la pantalla modal de filtrado con la lógica de selección.
MOW-4	**3**	**Como usuario quiero acceder a la información de un restaurante al seleccionarlo del listado.**	
MOW-24	40	**Prototipos**	Realizar los mockups de las pantallas para el acceso a la información de un restaurante.
MOW-25	20	**Firebase**	Modificar la base de datos para almacenar las descripciones de los restaurantes.
MOW-26	120	**Servicios**	Implementar el servicio para obtener la información de un restaurante.
MOW-27	240	**Componentes**	Implementar la pantalla de información de un restaurante.

Figura 3.7: Planificación del primer sprint en subtareas.

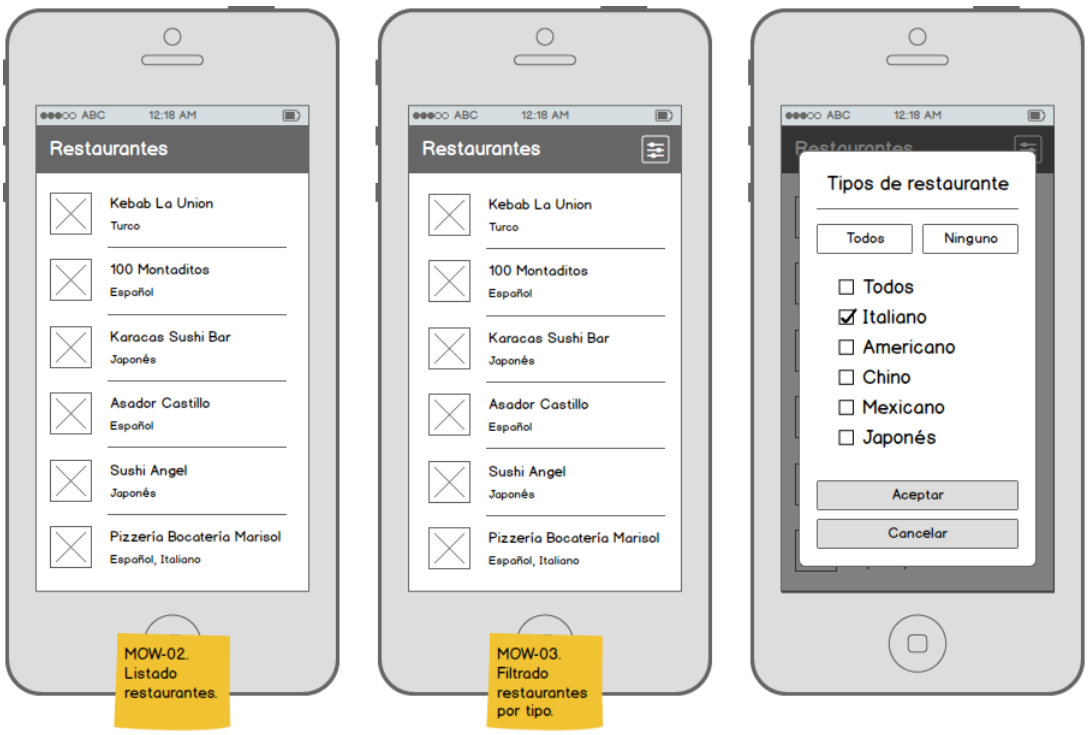


Figura 3.8: Bocetos del primer sprint (1/2).

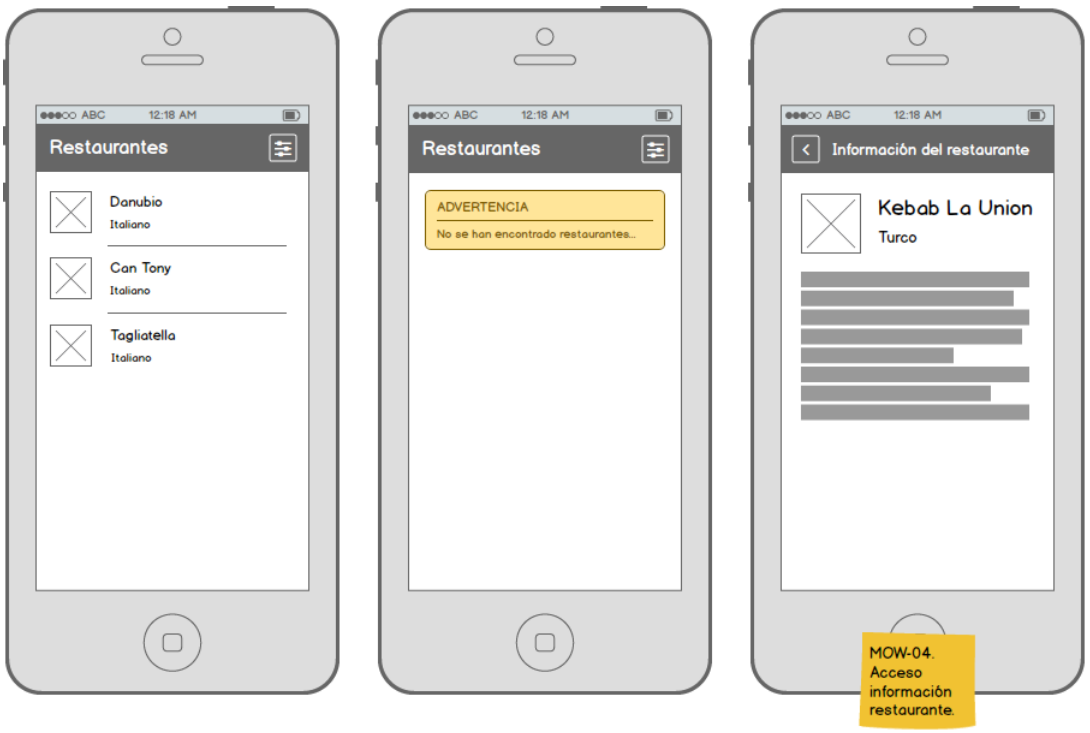


Figura 3.9: Bocetos del primer sprint (2/2).

La lista de componentes está conectado a una acción de *Redux* que realiza una petición a *Firebase* y obtiene el listado de los restaurantes con todos sus datos. Este listado se almacena en un reductor[5] de manera que la petición se hace una única vez y las próximas veces que se acceda a la pantalla los datos se cargan mucho más rápido. Mientras se realiza la petición se muestra un *spinner*[6] que indica la carga de datos.

Filtrado de restaurantes por tipo

La pantalla de filtrado es una pantalla modal. A efectos prácticos es igual que una pantalla, sólo que sus márgenes internos han sido reducidos y tiene un fondo negro transparente. Esta pantalla se pinta encima de la otra pantalla desde donde se accede, de modo que da un efecto de modal.

La pantalla modal consta de un título 'Tipos de restaurante', una lista de los tipos con casillas seleccionables, dos botones para seleccionar todos o ninguno (por comodidad) y los botones de aceptar y cancelar. En un principio las casillas seleccionables se importaban de una biblioteca externa que había que instalar, pero como tenían un diseño muy rígido decidí implementarlas por mi cuenta. De esta forma, disponemos de un componente casillero.

El modal funciona del siguiente modo: (1) se realiza una petición para traer la lista de tipos y almacenarla en el reductor; (2) se muestran todos sin seleccionar; (3) cada vez que se selecciona uno (o todos, o ninguno) se modifica la información del reductor; (4) al aceptar se ejecuta una

[5]Recordemos que un reductor es una porción de estado que normalmente se asocia a una pantalla y al que le llegan acciones para modificar dicho estado.
[6]Rueda giratoria animada.

acción que toma la lista de restaurantes y la lista de tipos y realiza el filtrado; (5) se vuelve a la pantalla de restaurantes y se muestra la lista filtrada.

Cabe decir que por un lado se almacena la lista con todos los restaurantes y por otro la lista filtrada. Los tipos seleccionados quedan almacenados en el reductor, de modo que si volvemos a acceder al modal estarán marcados.

Si al filtrar los restaurantes no se producen resultados, la pantalla de restaurantes muestra un mensaje de advertencia. Este tipo de mensajes son componentes reutilizables que hemos implementado también. Hay de cuatro tipos: información, advertencia, éxito y fallo. Ambos contienen un título y un cuerpo totalmente modificables. Y cada uno mantiene unos colores acordes al mensaje. El de información tiene unos colores azules; el de advertencia, amarillos; el de éxito, verdes; y el de fallo, rojos.

Información del restaurante

La pantalla de información es un componente reutilizable. En el listado anterior, al pulsar sobre un restaurante, se envía su información al componente que se encarga de mostrarlo. Este tipo de pantallas no tienen un reductor que almacene datos sobre ellas. No lo necesitan.

Como la pantalla de restaurantes, tiene una cabecera con el título 'Información del restaurante' y con un botón para volver atrás, que de nuevo lo proporciona la cabecera. En el cuerpo muestra la imagen, el nombre el tipo y una descripción del restaurante. Todo ello bien maquetado.

Esquema de navegación

En la Figura 3.10 mostramos el esquema de navegación entre pantallas.

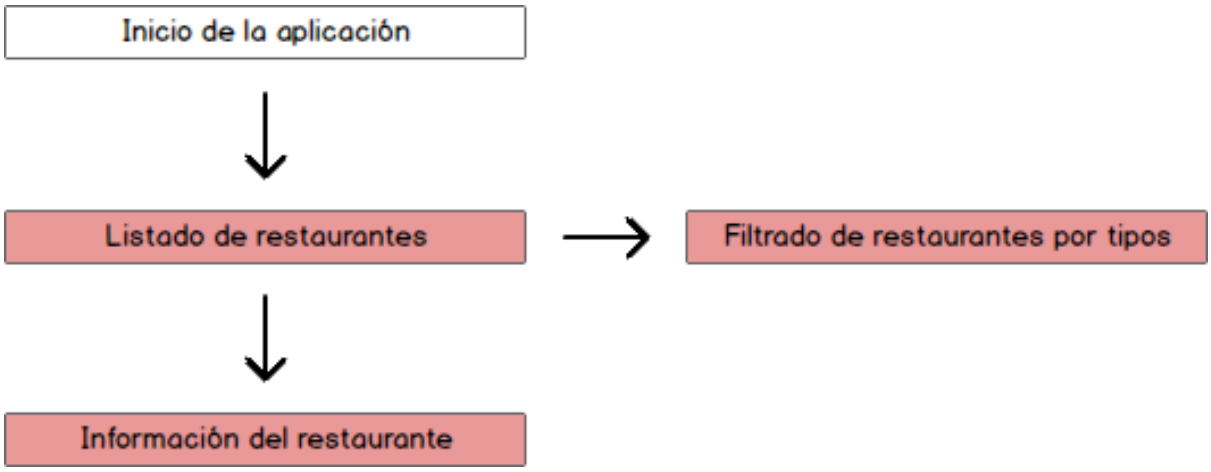


Figura 3.10: Esquema de navegación entre pantallas del primer sprint.

CRITERIOS DE ACEPTACIÓN

Los criterios para validar estas tres historias de usuario son los que se describen a continuación:

MOW-2 Como usuario quiero obtener un listado de todos los restaurantes de reparto a domicilio.

Pantalla de restaurantes

- Accesible automáticamente al inicio de la aplicación.
- Muestra una cabecera con el título 'Restaurantes'.
- Muestra un listado de todos los restaurantes.
- Muestra para cada restaurante su nombre y su tipo (mexicano, chino, italiano, etc).
- Muestra para cada restaurante una pequeña imagen con su logo.
- Muestra un aviso para el caso de que no existan restaurantes.

MOW-3 Como usuario quiero filtrar los restaurantes del listado por tipo (mexicano, chino, italiano, etc).

Pantalla de restaurantes

- Muestra un botón de filtrado en la cabecera que permite acceder a la pantalla de filtrado de restaurantes.

Pantalla de filtrado de restaurantes

- Muestra una pantalla modal con el título 'Tipos de restaurante'.
- Muestra un listado de los tipos de restaurante como casillas seleccionables.
- Permite seleccionar uno, ninguno o todos los tipos.
- Muestra por defecto todos los tipos seleccionados.
- Muestra un botón 'Todos' que permite seleccionar todos los tipos a la vez.
- Muestra un botón 'Ninguno' que permite deseleccionar todos los tipos a la vez.
- Muestra un botón 'Aceptar' que permite filtrar los restaurantes por tipo.
- Muestra un botón 'Cancelar' que permite volver atrás sin hacer nada más.
- Recuerda las opciones seleccionadas después de aceptar o cancelar.

MOW-4 Como usuario quiero acceder a la información de un restaurante al seleccionarlo del listado.

Pantalla de restaurantes

- Permite acceder desde cada restaurante a su pantalla de información.

Pantalla de información de restaurante

- Muestra una cabecera con el título 'Información del restaurante'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de restaurantes.
- Muestra el nombre, el tipo y una pequeña imagen con el logo del restaurante.
- Muestra una descripción detallada del restaurante.

SEGUIMIENTO DEL SPRINT

1. Planificación del primer sprint:
   a) Elección de las historias de usuario.
   b) Subdivisión de las historias en subtareas.
   c) Estimación en minutos de las subtareas.
2. Inicio del primer sprint.
   a) Creación y configuración del proyecto *React-Native*.
   b) Diseño de los prototipos.
   c) Implementación de los servicios y componentes.
   d) Validación de los criterios de aceptación.
3. Revisión del incremento del primer sprint con el cliente.
4. Cierre del primer sprint.
5. Revisión de la pila del producto.

Al comienzo de este sprint se ha planteado el dilema de qué hacer respecto a la dependencia de este proyecto con el del compañero Roberto. Si seguir como estaba planificado en un primer momento o ampliar el alcance del proyecto. Anteriormente hemos explicado que los servicios web de los que hacemos uso no pertenecen al alcance de este proyecto; no es nuestra labor implementarlos. Sin embargo, esto plantea un problema si el otro proyecto tiene un peor rendimiento o sufre algún tipo de contratiempo. Se produciría un retraso.

Durante el visionado del vídeo-tutorial aprendimos a utilizar la herramienta *Realtime Database* de *Firebase*, *Google*. Esta herramienta permite diseñar una base de datos *NoSQL* y emplear unos servicios de acceso y desconexión que facilitan enormemente crear una parte *back-end* para una aplicación móvil. A raíz de ello hemos tomado la decisión de implementar los servicios junto con la base de datos. Esto llevará más tiempo.

La idea es continuar con este desarrollo hasta que estén implementados todos los servicios web del otro proyecto que necesitamos. En ese momento, sopesaremos la viabilidad de adaptar la aplicación a los servicios de Roberto. La Figura 3.7 muestra subtareas de tipo 'Firebase' y 'Servicios' surgidas por esta decisión. La Figura 3.3 muestra la nueva arquitectura provisional del sistema.

Otro problema que se nos ha presentado tiene que ver con los conocimientos adquiridos durante la etapa de formación. No están lo suficientemente interiorizados. En parte por no

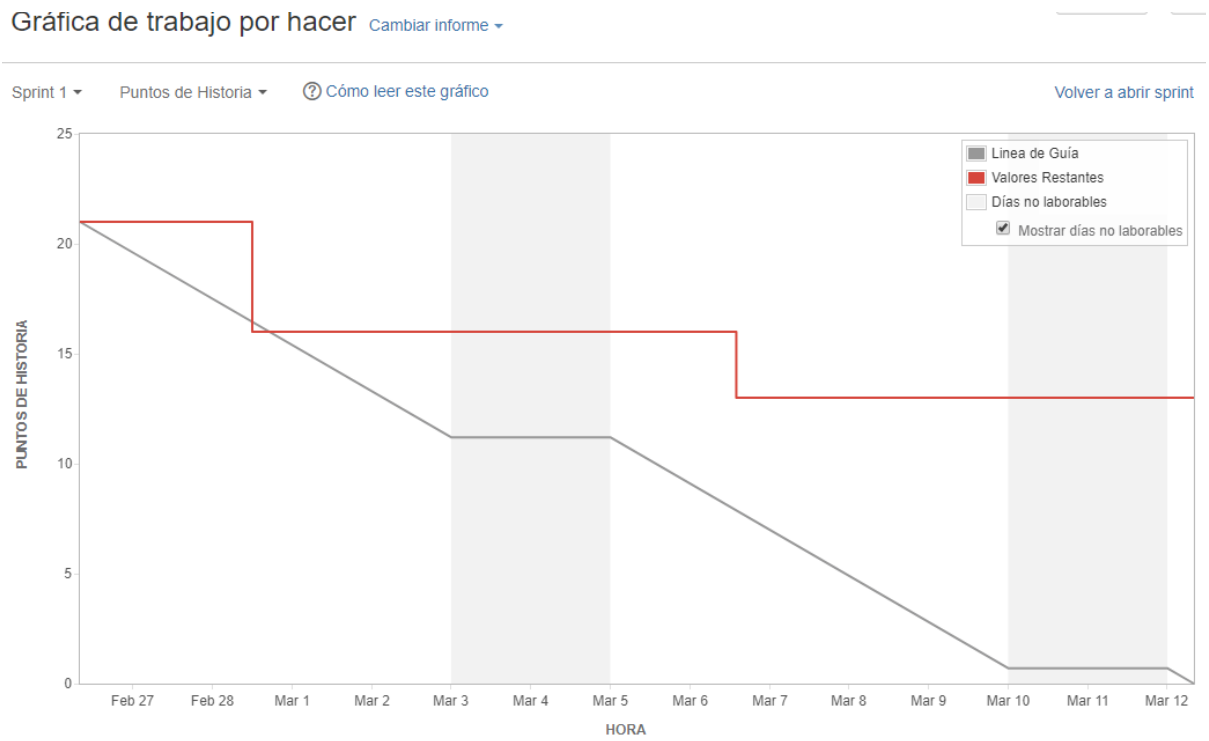


Figura 3.11: Gráfica de trabajo por hacer (en puntos de historia) del primer sprint.

tener conocimientos previos ni en *React-Native* ni en *JavaScript* y empezar casi de cero. Por ello, la adaptación está siendo lenta. Además hemos empleado poco tiempo en el desarrollo de este sprint: un total de 35 horas.

La Figura 3.11 muestra una gráfica de trabajo de puntos de historia realizados. Por un lado muestra la línea guía de puntos de historia por día que debemos completar para cumplir con un sprint planificado para dos semanas. Por otro lado, muestra la línea real de puntos de historia que hemos finalizado.

El primer escalón representa la historia de usuario sobre el listado (MOW-2), que ha sido la primera en realizarse. El segundo representa el acceso a un restaurante (MOW-4), que junto con la anterior hacen un total de 8 puntos de historia completados. El filtrado por tipo (MOW-3), que supone 13 puntos, no ha sido finalizada todavía. La razón se debe a que la lógica del filtrado de una lista es bastante compleja para un iniciado. Si no hubiéramos optado por implementar los servicios por nuestra cuenta quizá no habríamos tenido este problema. La historia volverá a la pila del producto y será incluida en el próximo sprint.

La velocidad de sprint ha sido de unos 9 puntos/semana aproximados.

3.6.3. Sprint 2: Listado de productos y gestión del pedido.

Este sprint tiene como meta **el listado de productos de un restaurantes y la gestión del pedido (añadir y eliminar productos)**. Detallamos sus principales características a continuación:

Intervalo de tiempo	12 de marzo - 25 de marzo (3ª Quincena)
Horas de prácticas	58 horas (9 jornadas)
Incidencias incluidas	MOW-3, MOW-5, MOW-6, MOW-7
Incidencias completadas	MOW-3, MOW-5, MOW-6, MOW-7
Estimación total	42 puntos de historia (completados: 42)
Velocidad	28,966 puntos/semana

PLANIFICACIÓN DEL SPRINT

Las historias de usuario escogidas para este sprint son las que siguen:

MOW-3 Como usuario quiero filtrar los restaurantes del listado por tipo (mexicano, chino, italiano, etc) para encontrar rápidamente aquellos correspondientes a un tipo en concreto.

MOW-5 Como usuario quiero obtener un listado de productos categorizados al seleccionar un restaurante para saber qué productos tiene.

MOW-6 Como usuario quiero disponer de una cesta de productos (pedido) para poder añadir los distintos productos de un restaurante.

MOW-7 Como usuario quiero poder añadir y eliminar los productos de un determinado restaurante en mi pedido para tener un control de lo que voy a pedir.

La Figura 3.12 muestra la planificación del segundo sprint y el estado de la pila del producto previo al inicio del sprint. Una vez seleccionadas las historias de usuario, se descomponen en subtareas y se estiman en tiempo. La Figura 3.13 muestra la descomposición en subtareas y la estimación en minutos (las historias están estimadas en puntos de historia). La descomposición de la historia MOW-5 ya se muestra en la Figura 3.7.

DISEÑO DE PROTOTIPOS

El diseño de la historia MOW-5 consta de cuatro bocetos (Figuras 3.14 y 3.15) con el listado de productos categorizados. El diseño de la historia MOW-6 consta también de cuatro bocetos (Figuras 3.15 y 3.16) con la información y el panel del pedido. El diseño de la historia MOW-7 consta de dos bocetos (Figuras 3.16 y 3.17) con las opciones de añadir y eliminar productos del pedido.

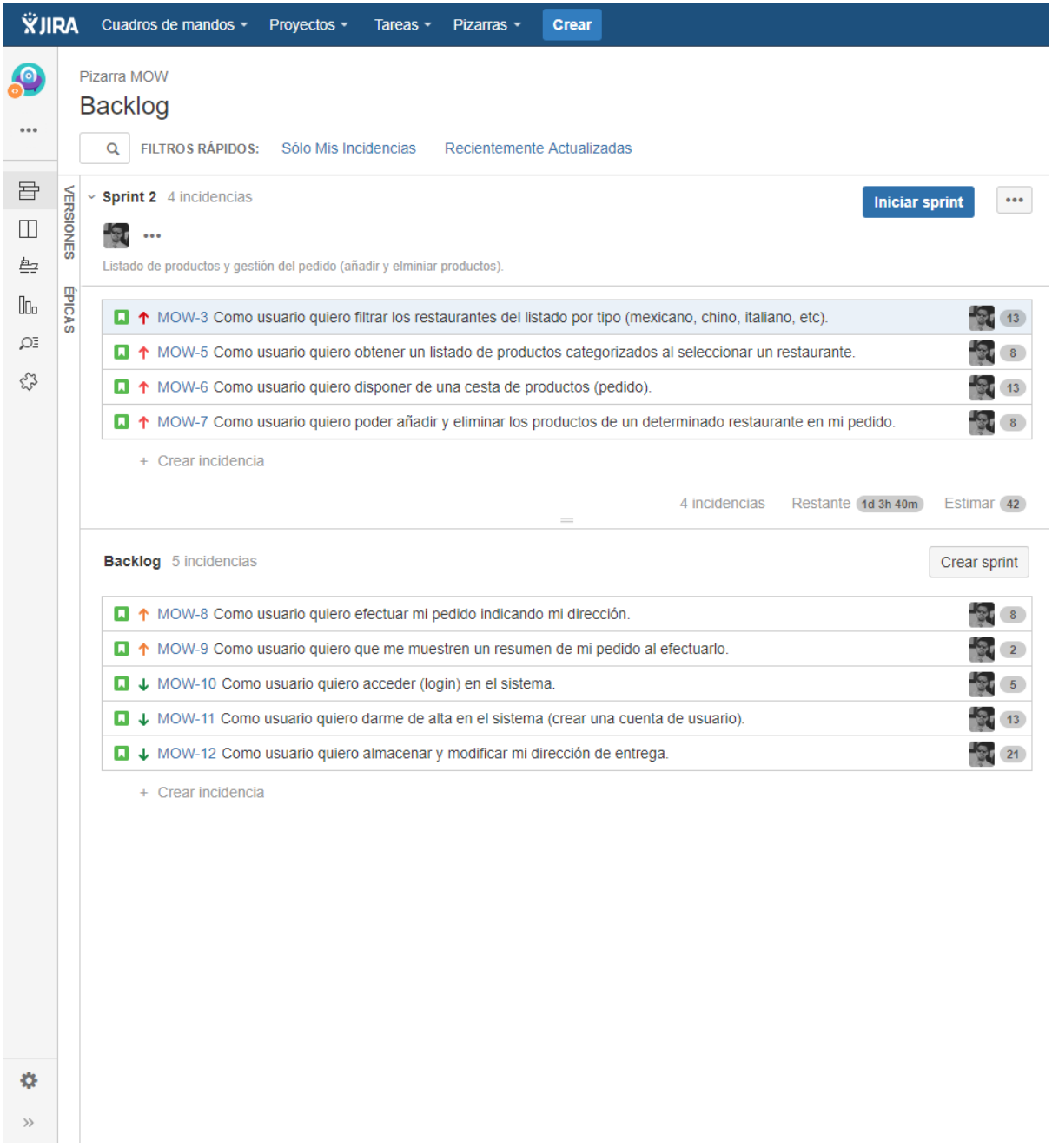


Figura 3.12: Planificación del segundo sprint y estado de la pila del producto (Jira).

ID	EST	INCIDENCIAS	
MOW-5	**8**	**Como usuario quiero obtener un listado de productos categorizados al seleccionar un restaurante.**	
MOW-28	80	**Prototipos**	Realizar los mockups de las pantallas para el listado de productos categorizados.
MOW-29	20	**Firebase**	Modificar la base de datos para almacenar las categorías y productos de cada restaurante.
MOW-30	120	**Servicios**	Modificar el servicio para obtener la información de un restaurante junto con sus categorías y productos.
MOW-31	60	**Componentes**	Modificar la pantalla de información con el listado de categorías.
MOW-32	240	**Componentes**	Implementar la pantalla de productos.
MOW-6	**13**	**Como usuario quiero disponer de una cesta de productos (pedido).**	
MOW-33	80	**Prototipos**	Realizar los mockups de las pantallas para el pedido.
MOW-34	20	**Firebase**	Modificar la base de datos para almacenar otros gastos.
MOW-35	120	**Componentes**	Implementar el panel informativo como componente.
MOW-36	60	**Componentes**	Modificar las pantallas de información y de productos con el panel informativo.
MOW-37	240	**Componentes**	Implementar la pantalla de pedido con las secciones total y resumen.
MOW-7	**8**	**Como usuario quiero poder añadir y eliminar los productos de un determinado restaurante en mi pedido.**	
MOW-38	20	**Prototipos**	Realizar los mockups de las pantallas para añadir y eliminar productos del pedido.
MOW-39	120	**Servicios**	Implementar el servicio para añadir productos al pedido.
MOW-40	120	**Servicios**	Implementar el servicio para eliminar productos del pedido.
MOW-41	120	**Servicios**	Implementar la lógica para reiniciar un pedido al cambiar de restaurante.
MOW-42	60	**Componentes**	Modificar la pantalla de productos para poder añadir los productos.
MOW-43	60	**Componentes**	Modificar la pantalla de pedido para poder eliminar los productos.

Figura 3.13: Planificación del segundo sprint en subtareas.

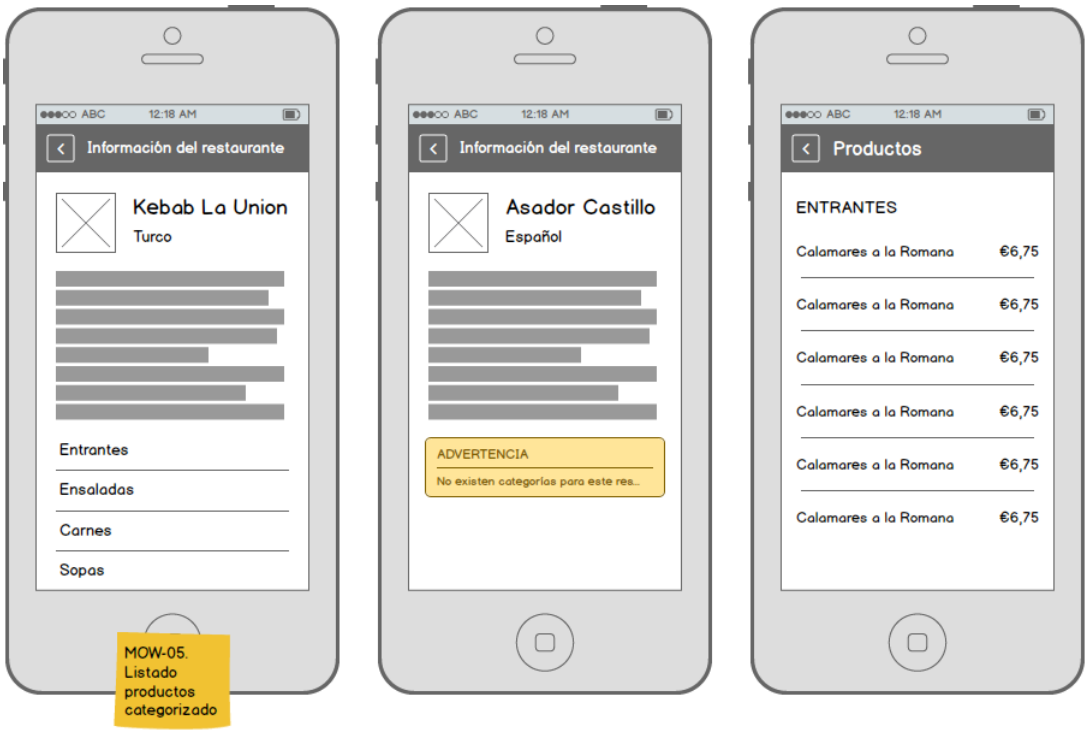


Figura 3.14: Bocetos del segundo sprint (1/4).

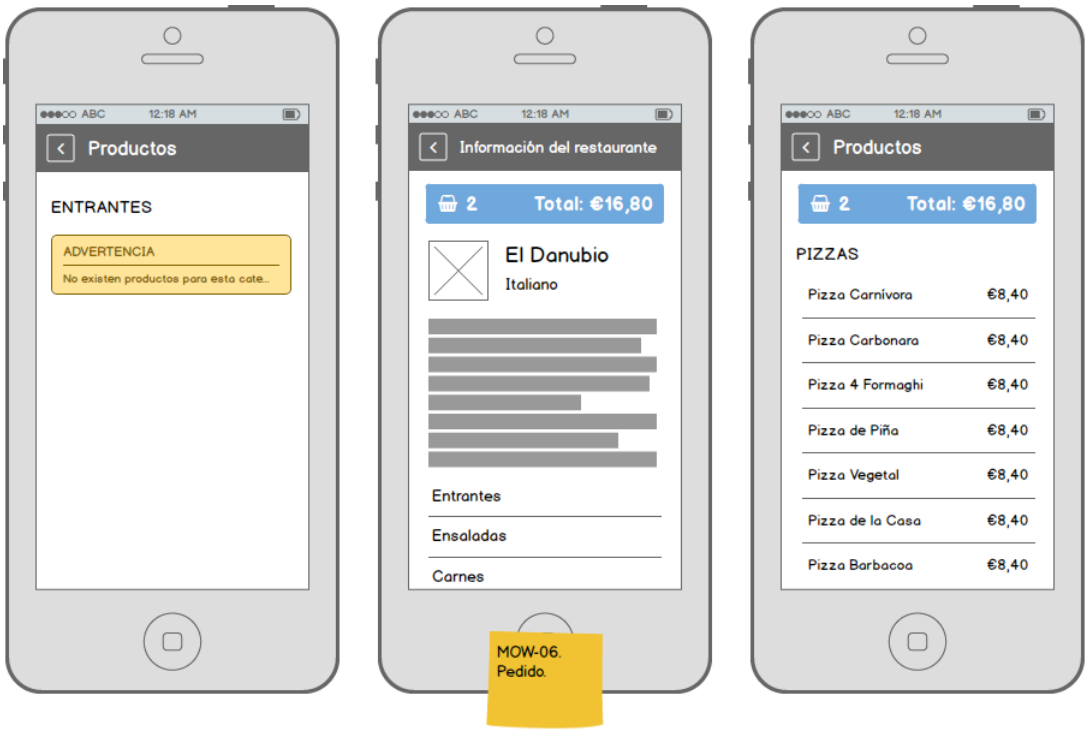


Figura 3.15: Bocetos del segundo sprint (2/4).

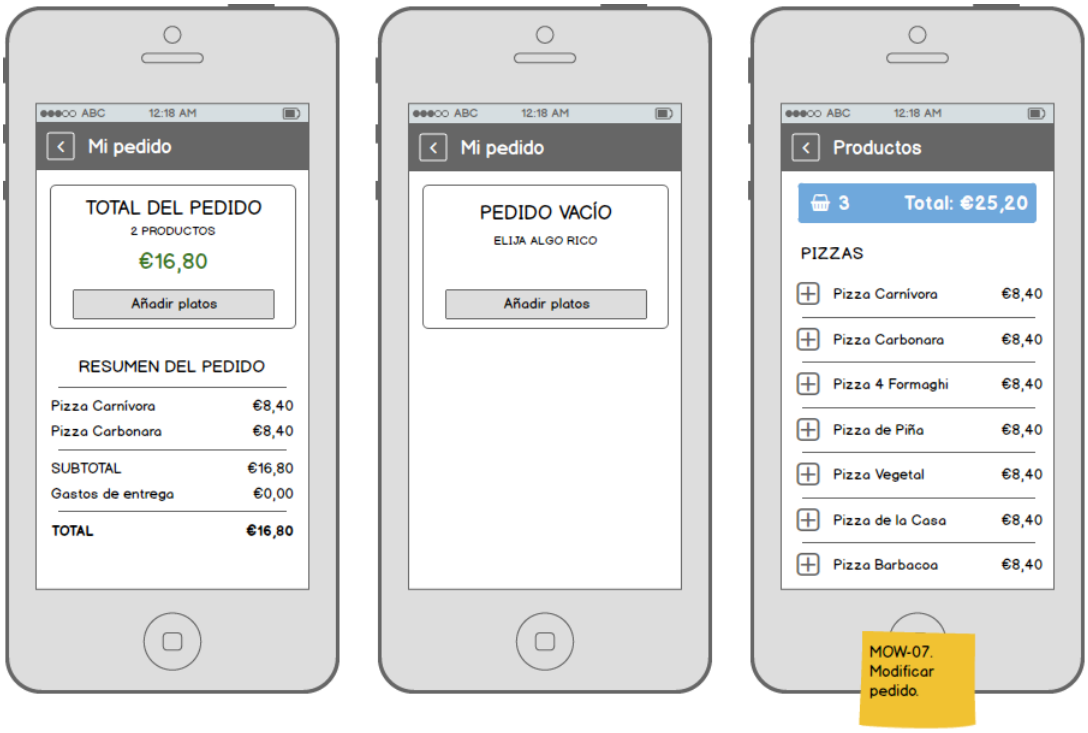


Figura 3.16: Bocetos del segundo sprint (3/4).

Figura 3.17: Bocetos del segundo sprint (4/4).

DETALLES DE IMPLEMENTACIÓN

Información del restaurante

A la pantalla de información le hemos añadido una lista que muestra ítems seleccionables con los nombres de las categorías. Las listas de *react-native* permiten hacer *scroll*, es decir, desplazarse verticalmente con el dedo. En el caso de la pantalla de restaurantes va bien, pero en este caso que la lista tiene muy poco espacio por el resto de componentes (descripción, imagen, título y tipo) hemos implementado el *scroll* en toda la pantalla. De aquí en adelante será algo que hagamos en casi todas las pantallas.

Desde cada categoría podemos acceder a la pantalla de productos. Si no existen categorías se muestra un mensaje de advertencia.

Listado de productos

La pantalla de productos sigue el mismo esquema que pantallas anteriores. Tiene una cabecera con el título 'Productos' y un botón para volver atrás, y una lista con los productos de la categoría. Estos datos están ya incluidos al obtener la información de los restaurantes. Podríamos emplear acciones que hicieran peticiones para obtener la información de un restaurante o sus productos, pero la estructura JSON de la base de datos en *Firebase* facilita traer toda la información de golpe.

Por cada producto hay un botón de adición para añadir el producto al pedido. Si no existen productos se muestra un mensaje de advertencia.

Información del pedido

La pantalla del pedido está compuesta por una cabecera con el título de 'Mi pedido' y un botón de regreso, y dos secciones: total y resumen. La sección total muestra el número de productos elegidos y el total en euros. Además tiene un botón 'Añadir platos' que lleva al restaurante o a los productos, según desde donde se haya accedido. La sección resumen está formada por un lista de los productos con sus precios y de otros gastos, como el de entrega. Si hay más de un mismo producto, se indica con un número entre paréntesis a la derecha. Al final se indica el total también. En caso de no haber ningún producto se muestra la sección total únicamente con un mensaje para animar a elegir algo.

El pedido está complementado por un panel informativo de un color azul. Se trata de un nuevo componente que hemos creado y que mostramos en las pantallas de información y de productos. Indica el número de productos y el total en euros. De ese modo el usuario tiene un control de lo que pide. El panel es seleccionable de modo que lleva a la pantalla de pedido.

Dentro de la sección de resumen, cada producto tiene un botón de eliminación para quitarlo del pedido.

El proceso funciona del siguiente modo. Cuando un usuario pulsa el botón de añadir, se envía una acción para almacenar en un reductor la información del producto. Esta información consta de un código, el nombre y el precio unitario en euros. Además, cada vez que se añade se hacen cálculos para almacenar el total, el subtotal, etc. De modo que este reductor simula el carro de la compra.

A todo ello, hemos incorporado una lógica de negocio para que cuando un usuario navegue a otro restaurante que no es de su pedido reinicie el pedido. Es decir, un usuario no puede hacer más de un pedido a la vez. Antes de reiniciarlo se muestra una pantalla modal advirtiendo del reinicio.

Esquema de navegación

En la Figura 3.18 mostramos el esquema de navegación entre pantallas. Desde información y/o productos podemos acceder al pedido y, desde esta, regresar.

CRITERIOS DE ACEPTACIÓN

Los criterios para validar tres de las cuatro historias de usuario son los que se describen a continuación:

MOW-5 Como usuario quiero obtener un listado de productos categorizados al seleccionar un restaurante.

Pantalla de información de restaurante

- Muestra un listado de las categorías del restaurante seleccionado.
- Muestra para cada categoría el nombre (entrantes, ensaladas, etc).

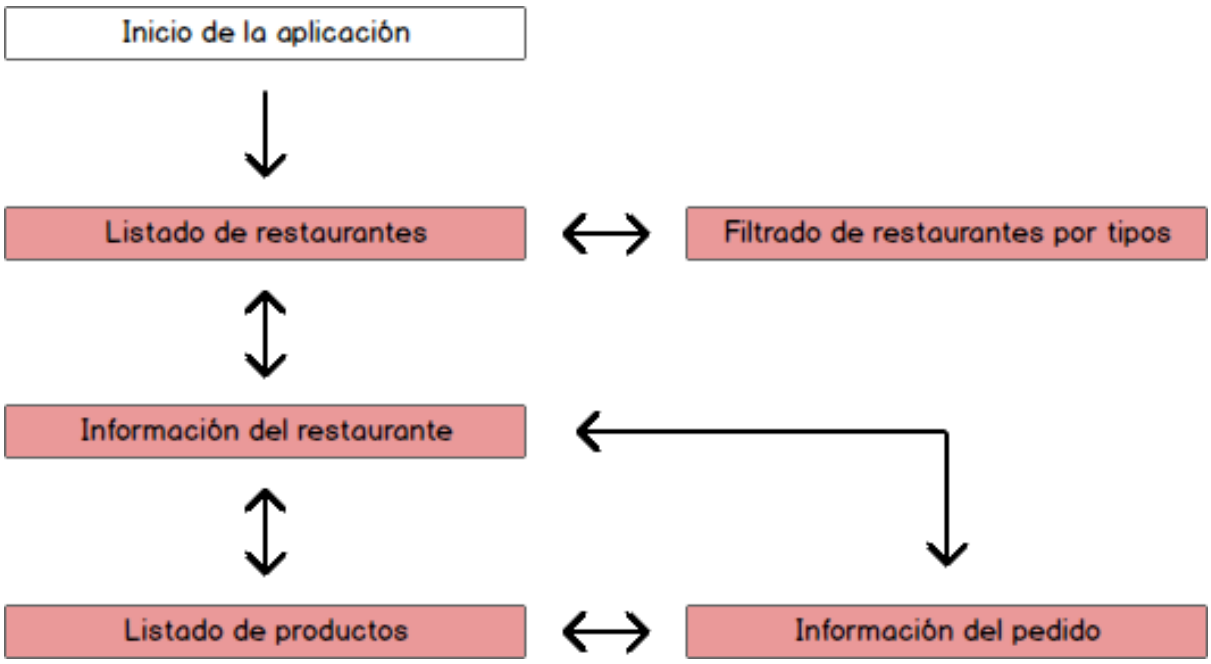


Figura 3.18: Esquema de navegación entre pantallas del segundo sprint.

- Muestra un aviso para el caso de que no existan categorías.
- Permite acceder desde cada categoría a la pantalla de sus productos.

Pantalla de productos

- Muestra una cabecera con el título 'Productos'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de info de restaurante.
- Muestra un listado de todos los productos para cada categoría.
- Muestra para cada producto el nombre y el precio en euros.
- Muestra un aviso para el caso de que no existan productos.

MOW-6 Como usuario quiero disponer de una cesta de productos (pedido).

Pantalla de información + Pantalla de productos

- Muestra un panel informativo con el número total de productos y el precio final en euros.
- Muestra el panel informativo en la parte superior de la pantalla.
- Permite acceder desde el panel informativo a la pantalla de pedido.

Pantalla de pedido

- Muestra una cabecera con el título 'Mi pedido'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de información o de productos (según desde donde se haya accedido).

- Muestra una primera sección con el total del pedido: número de productos y precio final.
- Muestra en la primera sección un mensaje de pedido vacío si no existen productos añadidos.
- Muestra en la primera sección un botón que permite seguir añadiendo productos.
- Muestra una segunda sección con el resumen del pedido con todos los productos y otros gastos.
- Muestra en la segunda sección para cada producto el nombre, la cantidad y el precio acumulado.
- Muestra en la segunda sección para cada gasto el nombre y el precio.
- Muestra en la segunda sección el subtotal como la suma de los precios de los productos.
- Muestra en la segunda sección el total como la suma del subtotal y los otros gastos.
- No permite tener más de un pedido para distintos restaurantes a la vez.

MOW-7 Como usuario quiero poder añadir y eliminar los productos de un determinado restaurante en mi pedido.

Pantalla de productos

- Muestra para cada producto un botón que permite añadir el producto al pedido.
- Permite añadir un producto tantas veces como se quiera.
- Actualiza el panel informativo cada vez que se añade un producto.

Pantalla de pedido

- Muestra para cada producto un botón que permite eliminar una unidad del producto del pedido.
- Permite añadir productos de distintas categorías para un mismo pedido de un mismo restaurante.
- No permite añadir productos de distintos restaurantes para un mismo pedido.
- Reinicia el pedido al cambiar de restaurante.

SEGUIMIENTO DEL SPRINT

1. Planificación del segundo sprint:
   a) Elección de las historias de usuario.
   b) Subdivisión de las historias en subtareas.
   c) Estimación en minutos de las subtareas.
2. Inicio del segundo sprint.

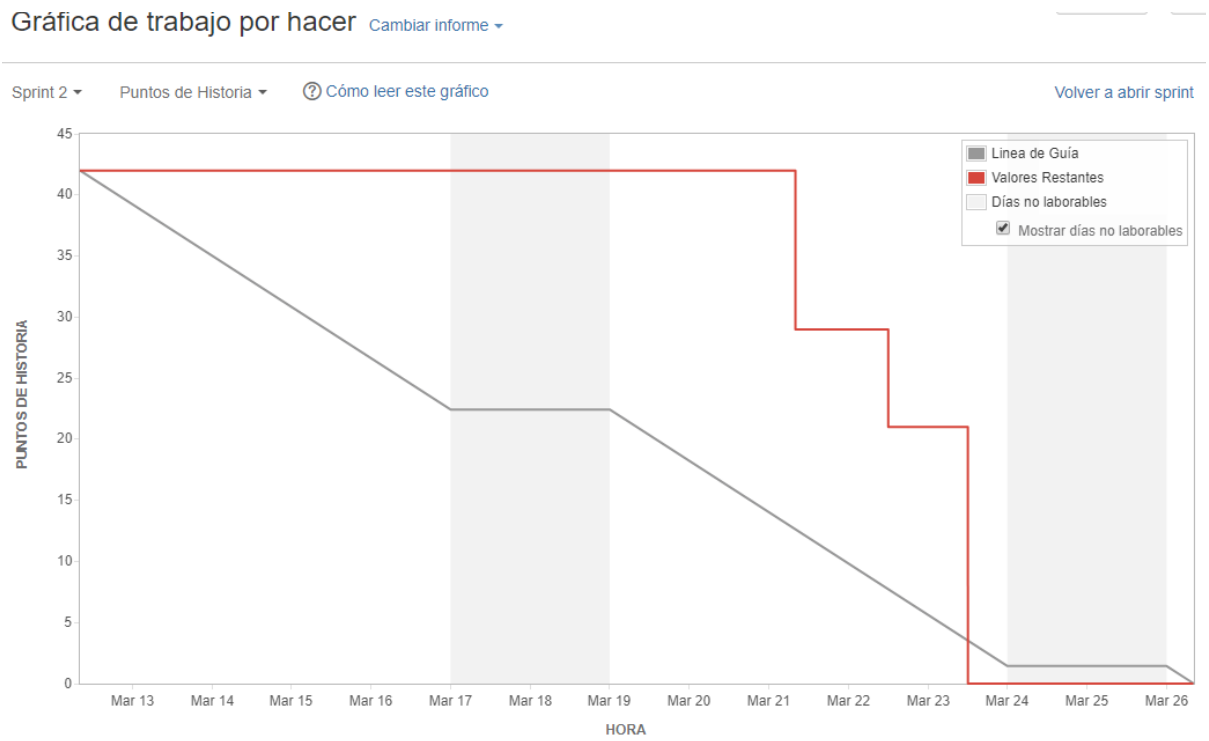


Figura 3.19: Gráfica de trabajo por hacer (en puntos de historia) del segundo sprint.

   a) Diseño de los prototipos.
   b) Implementación de los servicios y componentes.
   c) Validación de los criterios de aceptación.

3. Revisión del incremento del segundo sprint con el cliente.
4. Cierre del segundo sprint.
5. Revisión de la pila del producto.

Este sprint ha constado de 42 puntos de historia: 13 puntos de antiguas historias (MOW-3) y 29 puntos de nuevas (MOW-5, MOW-6 y MOW-7). Con 21 puntos el sprint anterior no lo terminamos. Este han sido el doble de puntos. Sin embargo, hay dos razones para haber planificado este sprint con tantos puntos: (1) la antigua historia no estaba totalmente sin hacer; de sus 13 puntos parte estaban hechos; y (2) al principio, debido a la adaptación a las nuevas tecnologías, es más costoso el desarrollo, pero conforme avanzamos el rendimiento aumenta. De hecho, el sprint ha sido completado.

La Figura 3.19 muestra el desarrollo de este sprint. El primer escalón se corresponde con el filtrado (MOW-3). Como se aprecia, ha sido la historia de usuario más costosa y que ha durado casi toda la quincena. A partir de aquí, el resto de historias las hemos finalizado en pocos días, terminando el sprint incluso antes de tiempo. La velocidad ha sido de unos 29 puntos/semana aproximados. En comparación al sprint anterior, son unos 20 puntos de diferencia.

A consecuencia de esta mejora de rendimiento, durante la reunión con el cliente nos han propuesto tres nuevos aspectos para un futuro.

- Analíticas para recabar información del tráfico que llega a la aplicación y del comportamiento de los usuarios.
- Notificaciones.
- Internacionalización (i18n).[7]

Valoraremos qué nuevos aspectos son posibles cumplir antes de finalizar la estancia en prácticas y los incluiremos como historias de usuario en la pila del producto.

3.6.4. Sprint 3: Realización del pedido.

Este sprint tiene como meta **la realización del pedido**. Detallamos sus principales características a continuación:

Intervalo de tiempo	26 de marzo - 2 de abril (4ª Quincena: Semana 1)
Horas de prácticas	23,5 horas (4 jornadas)
Incidencias incluidas	MOW-8, MOW-9
Incidencias completadas	MOW-8, MOW-9
Estimación total	10 puntos de historia (completados: 10)
Velocidad	17,021 puntos/semana

PLANIFICACIÓN DEL SPRINT

Las historias de usuario escogidas para este sprint son las que siguen:

MOW-8 Como usuario quiero efectuar mi pedido indicando mi dirección para que me hagan la entrega de la comida en mi casa.

MOW-9 Como usuario quiero que me muestren un resumen de mi pedido al efectuarlo para tener constancia de lo que he pedido.

La Figura 3.20 muestra la planificación del tercer sprint y el estado de la pila del producto previo al inicio del sprint. Una vez seleccionadas las historias de usuario, se descomponen en subtareas y se estiman en tiempo. La Figura 3.21 muestra la descomposición en subtareas y la estimación en minutos (las historias están estimadas en puntos de historia).

[7] La internacionalización es el proceso de diseño de software para adaptar el producto a diferentes idiomas sin la necesidad de hacer grandes modificaciones en el código. De manera abreviada se les conoce i18n. El número la abreviatura indica la cantidad de letras internas del vocablo inglés (entre la *i* y la *n* de *internationalization* hay 18 letras).

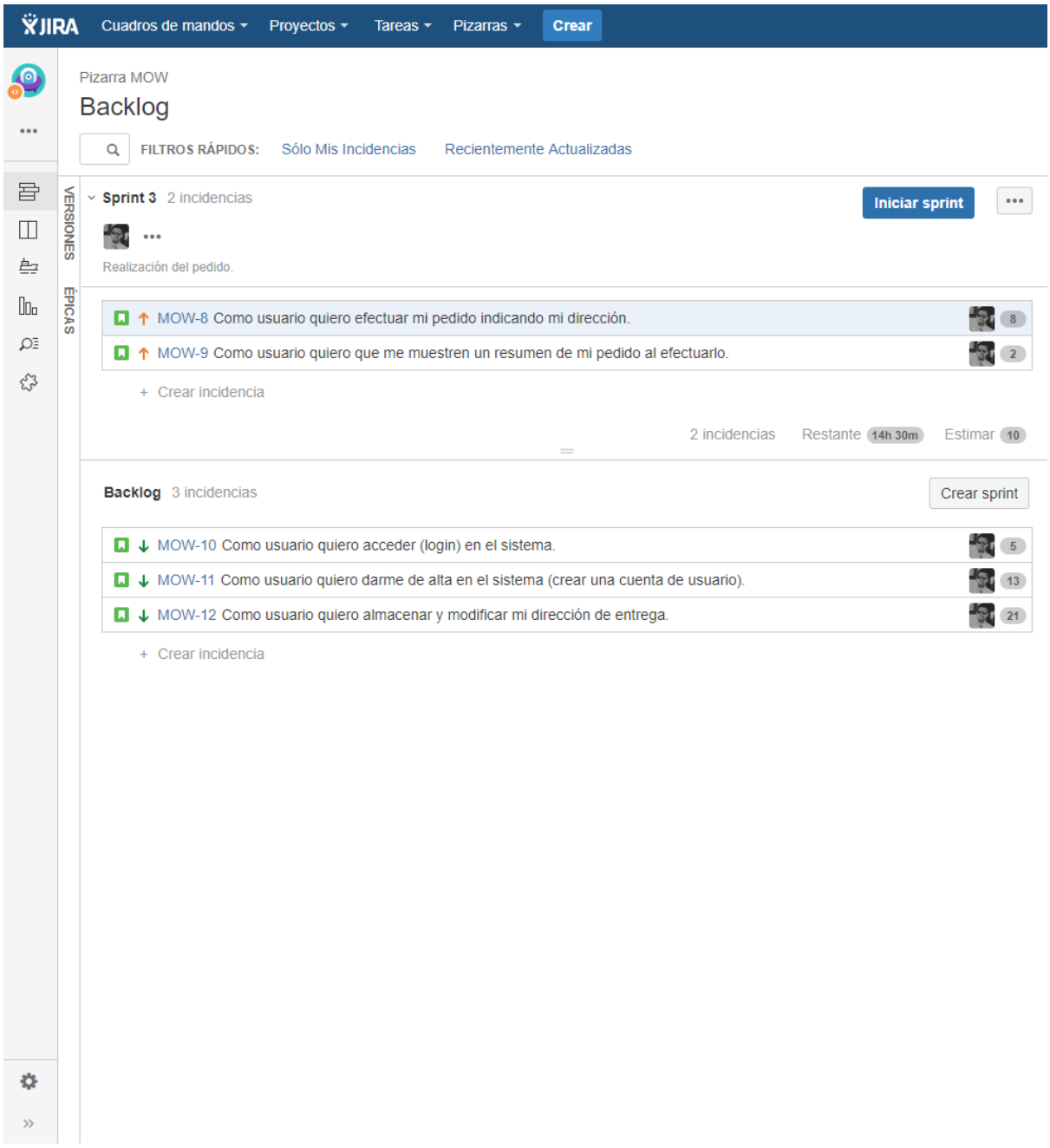


Figura 3.20: Planificación del tercer sprint y estado de la pila del producto (Jira).

ID	EST	INCIDENCIAS	
MOW-8	**8**	**Como usuario quiero efectuar mi pedido indicando mi dirección.**	
MOW-44	60	**Prototipos**	Realizar los mockups de las pantallas para indicar la dirección de entrega.
MOW-45	120	**Servicios**	Conectar pantalla de dirección con servicio web de autocompletado de Google.
MOW-46	120	**Servicios**	Conectar pantalla de dirección con servicio web de mapas de Google.
MOW-47	30	**Componentes**	Modificar la pantalla de pedido con el botón.
MOW-48	240	**Componentes**	Implementar la pantalla de dirección de entrega con el autocompletado y el mapa.
MOW-9	**2**	**Como usuario quiero que me muestren un resumen de mi pedido al efectuarlo.**	
MOW-49	60	**Prototipos**	Realizar los mockups de las pantallas para la realización del pedido.
MOW-50	240	**Componentes**	Implementar la pantalla de pedido realizado con las tres secciones.

Figura 3.21: Planificación del tercer sprint en subtareas.

DISEÑO DE PROTOTIPOS

El diseño de la historia MOW-8 consta de tres bocetos (Figura 3.22) con la forma de indicar la dirección de envío. El diseño de la historia MOW-9 consta también de tres bocetos (Figura 3.23 con el resumen del pedido una vez realizado y los distintos estados que se pueden dar: éxito o fallo.

DETALLES DE IMPLEMENTACIÓN

Información del pedido

En la pantalla de información hemos añadido un botón que redirige a la pantalla de dirección de entrega del pedido.

Dirección de entrega del pedido

La pantalla de dirección del pedido consta de un formulario con un único campo que permite introducir una dirección y registrarla. No existe ni autocompletado ni un mapa de *Google* para indicar la posición. En el seguimiento explicamos el porqué. También consta de un botón para finalizar el pedido y de la cabecera con el título 'Dirección de envío' y el botón de regreso.

Resumen del pedido realizado

La pantalla de resumen aprovecha la sección de la pantalla de información del pedido. Hemos reutilizado el componente, sólo que en este caso no hay botones de eliminación. En caso de realizarse correctamente el pedido se muestra un mensaje de éxito, y en caso de no hacerse, de fracaso. La cabecera tiene de título 'Mi pedido' y no cuenta con un botón de regreso. Una vez hecho el pedido no se puede volver atrás. Pero si que hay un botón que permite regresar a los restaurantes. Cabe apuntar que el pedido no se almacena.

Mejoras

Figura 3.22: Bocetos del tercer sprint (1/2).

Figura 3.23: Bocetos del tercer sprint (2/2).

Después del sprint, y para no arrastrar errores de diseño, código y rendimiento en un futuro, con los problemas que eso conllevaría, hemos llevado a cabo una serie de mejoras. Estas mejoras se centran en varias cosas.

Una es en cambiar el componente lista por otro denominado *flatList*. Este componente no es tan problemático como el primero, y libra de implementar bastante código auxiliar para conseguir un mejor rendimiento. Un rendimiento que se debe a que de todos los elementos de una lista solamente carga aquellos que están mostrados en pantalla.

Otra es emplear una biblioteca distinta que conecte con *Firebase*. La nueva biblioteca tiene es más rápida, pero tiene problemas con la instalación nativa en *iOS*. De momento hemos hecho el cambio y más adelante nos enfrentaremos con los problemas en dicho sistema.

También hemos añadido un margen superior para no repintar sobre la barra de estado del dispositivo, donde aparece el icono de carga, fecha, etc.

Finalmente, le hemos dado consistencia a todo el apartado gráfico con un estilo y colores uniformes ya que estaba siendo una amalgama de diferentes diseños. En consecuencia, hemos refactorizado la cabecera. Ahora deja un margen superior para no repintar sobre la barra de estado del dispositivo, donde aparece el icono de carga, fecha, etc.; y podemos añadir botones personalizados (no son los de la biblioteca *react-native-router-flux* los cuales no se podía modificar su diseño).

Esquema de navegación

En la Figura 3.24 mostramos el esquema de navegación entre pantallas. Una vez visto el resumen del pedido realizado y vuelto a restaurantes no podemos volver al resumen.

CRITERIOS DE ACEPTACIÓN

Los criterios para validar las dos historias de usuario son los que se describen a continuación:

MOW-7 Como usuario quiero poder añadir y eliminar los productos de un determinado restaurante en mi pedido.

Pantalla de productos

- Muestra para cada producto un botón que permite añadir el producto al pedido.
- Permite añadir un producto tantas veces como se quiera.
- Actualiza el panel informativo cada vez que se añade un producto.

Pantalla de pedido

- Muestra para cada producto un botón que permite eliminar una unidad del producto del pedido.
- Permite añadir productos de distintas categorías para un mismo pedido de un mismo restaurante.

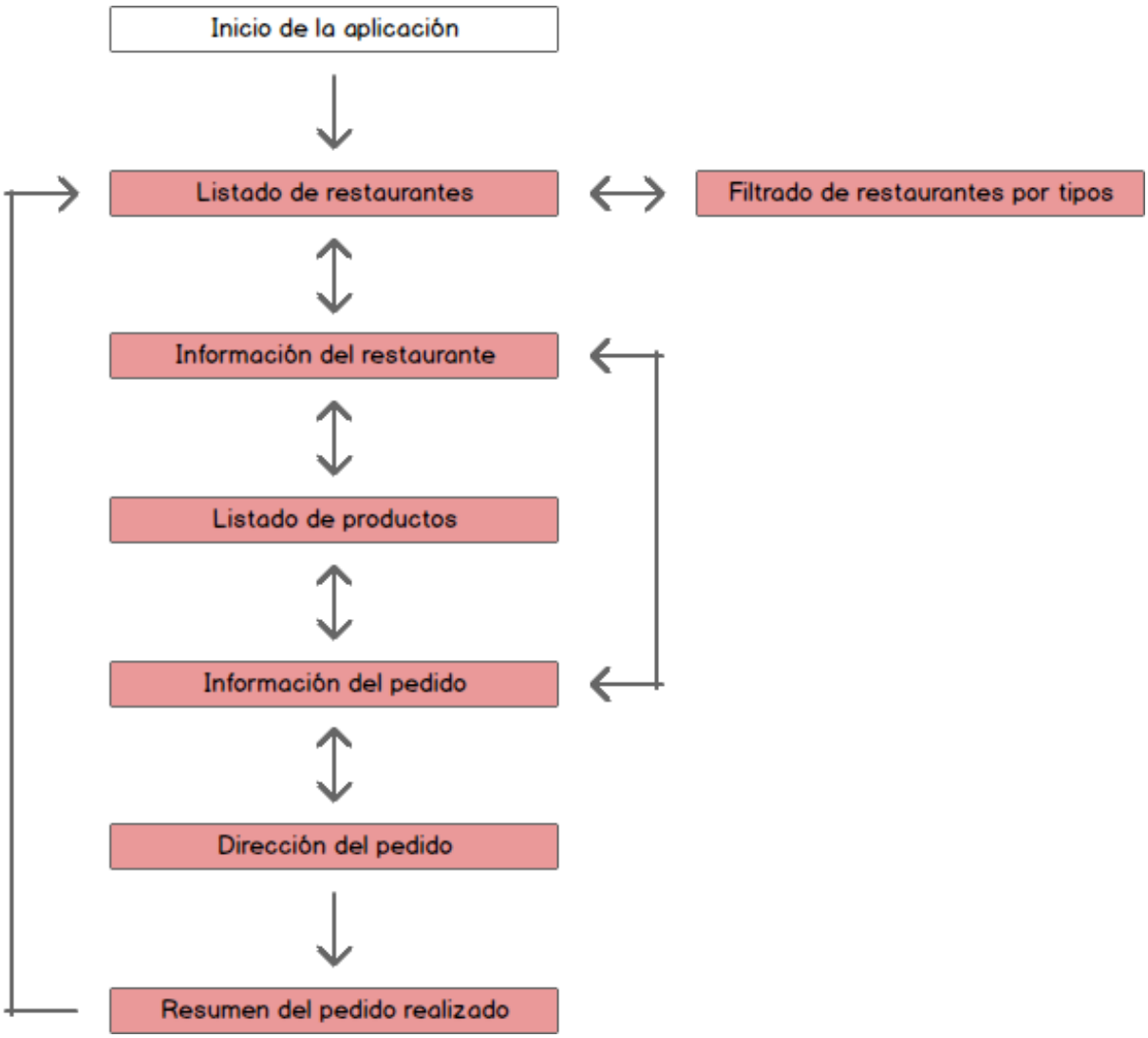


Figura 3.24: Esquema de navegación entre pantallas del tercer sprint.

- No permite añadir productos de distintos restaurantes para un mismo pedido.
- Reinicia el pedido al cambiar de restaurante.

MOW-8 Como usuario quiero efectuar mi pedido indicando mi dirección.

Pantalla de pedido

- Muestra un botón 'Realizar pedido' que redirige a la pantalla de dirección de entrega.

Pantalla de dirección de entrega

- Muestra una cabecera con el título 'Dirección de envío'.

- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de pedido.
- Muestra un formulario para introducir la dirección.
- Muestra un botón 'Finalizar el pedido' que redirige a la pantalla de pedido realizado.
- Permite el autocompletado de la dirección.
- Permite indicar la dirección mediante un mapa interactivo.

SEGUIMIENTO DEL SPRINT

Actividades principales

1. Planificación del tercer sprint:
   a) Elección de las historias de usuario.
   b) Subdivisión de las historias en subtareas.
   c) Estimación en minutos de las subtareas.
2. Inicio del tercer sprint.
   a) Diseño de los prototipos.
   b) Implementación de los servicios y componentes.
   c) Validación de los criterios de aceptación.
3. Cierre del tercer sprint.
4. Revisión de la pila del producto.
5. Mejoras de código, diseño y rendimiento.

Durante este sprint se ha presentado el problema de cómo indicar la dirección de entrega en la aplicación (MOW-8). Hemos propuesto dos soluciones: (1) un mapa de *Google* que permitiera al usuario indicar su dirección con sólo señalar su ubicación; (2) un campo de direcciones con un servicio de autocompletado de *Google*. La implementación de ambas no ha tenido éxito. Además, tampoco tenían sentido puesto que la dirección de entrega debe incluir portal, piso y número de puerta, cosa que no es posible representar con estas soluciones. Todo ello nos ha retrasado unos días como se muestra en el primer escalón de la Figura 3.25.

Aún así, el sprint comprendía pocos puntos de historia (10 puntos) y ha sido completado en muy poco tiempo (una semana). En consecuencia, no ha habido una reunión con el cliente. Para que los futuros sprints duren unas dos semanas deberemos planificarlos con más puntos de historia.

La Figura 3.25 muestra una línea plana indicando que desde que terminamos el sprint hasta que lo hemos cerrado han pasado varios días. Esto se debe a que hemos emprendido una serie de mejoras a nivel de código, diseño y rendimiento. De otra manera nos hubiera complicado el mantenimiento y las futuras implementaciones.

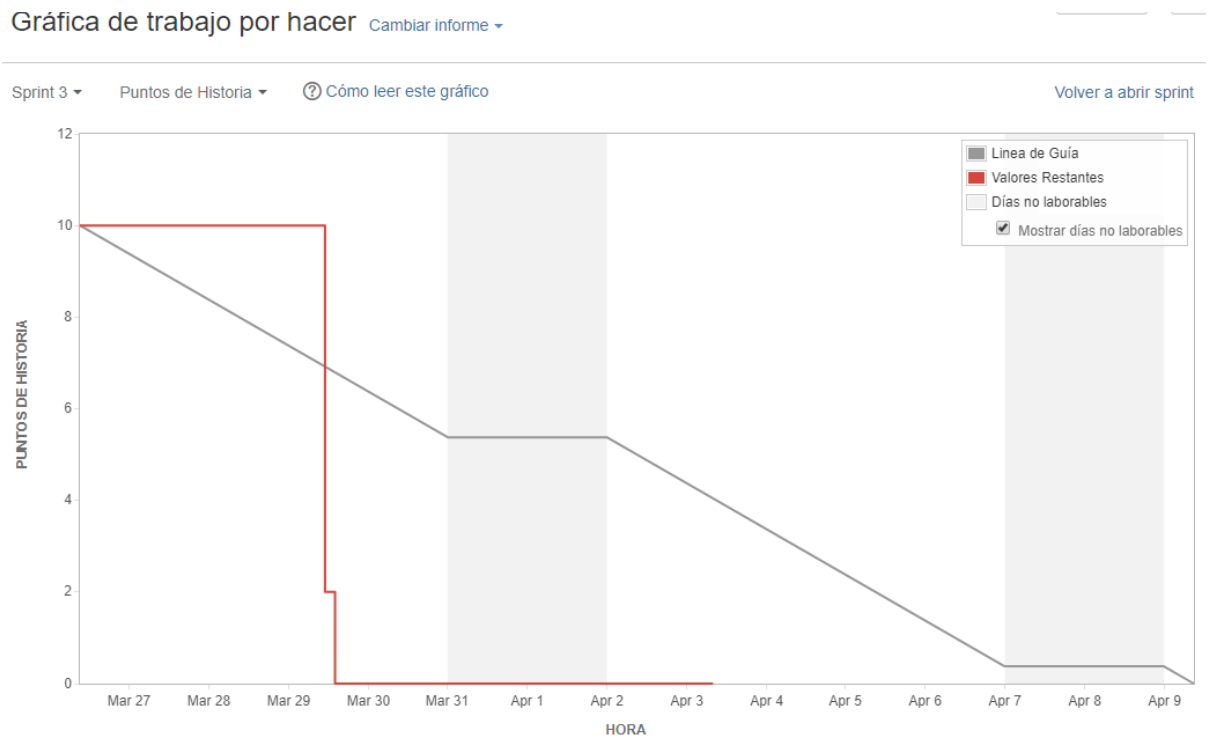


Figura 3.25: Gráfica de trabajo por hacer (en puntos de historia) del tercer sprint.

Antes del cierre, diversos compañeros de departamento han sugerido ciertas mejoras. Valoraremos qué mejoras son posibles cumplir antes de finalizar la estancia en prácticas y las incluiremos como historias de usuario en la pila del producto en un futuro.

3.6.5. Sprint 4: Registro de usuario y dirección de entrega.

Este sprint tiene como meta **el registro de usuarios y el mantenimiento de la dirección de entrega (almacenar y modificar sólo una)**. Detallamos sus principales características a continuación:

Intervalo de tiempo	3 de abril - 9 de abril (4ª Quincena: Semana 2)
Horas de prácticas	32 horas (4 jornadas)
Incidencias incluidas	MOW-10, MOW-11, MOW-12
Incidencias completadas	MOW-10, MOW-11, MOW-12
Estimación total	39 puntos de historia (completados: 39)
Velocidad	48,750 puntos/semana

PLANIFICACIÓN DEL SPRINT

Las historias de usuario escogidas para este sprint son las que siguen:

MOW-10 Como usuario quiero acceder (*login*) en el sistema para acceder a una cuenta de usuario y facilitarme ciertas operaciones como indicar mi dirección de manera automática.

MOW-11 Como usuario quiero darme de alta en el sistema (crear una cuenta de usuario) para almacenar datos personales e información sobre mis pedidos, direcciones, etc. que me faciliten ciertas operaciones.

MOW-12 Como usuario quiero almacenar y modificar mi dirección de entrega para no tener que introducirla cada vez que realizo un pedido.

La Figura 3.26 muestra la planificación del cuarto sprint y el estado de la pila del producto previo al inicio del sprint. Una vez seleccionadas las historias de usuario, se descomponen en subtareas y se estiman en tiempo. La Figura 3.27 muestra la descomposición en subtareas y la estimación en minutos (las historias están estimadas en puntos de historia).

DISEÑO DE PROTOTIPOS

El diseño de la historia MOW-10 consta de tres bocetos (Figuras 3.28) con las pantallas de presentación y de acceso. El diseño de la historia MOW-11 consta de cuatro bocetos (Figuras 3.29 y 3.30) con el registro y unas modificaciones en el acceso. Finalmente, el diseño de la historia MOW-12 consta también de cuatro bocetos (Figuras 3.30 y 3.31) con el menú de usuario, el botón para acceder a él y la pantalla de direcciones.

DETALLES DE IMPLEMENTACIÓN

Acceso

Previamente a la pantalla de acceso hay una pantalla de presentación que únicamente muestra el título de la aplicación y un botón para ir al acceso.

La pantalla de acceso consta de una cabecera con el título 'Acceso' y de un formulario. El formulario está formado por los campos de dirección de correo electrónico y contraseña. La contraseña está oculta. En la parte inferior tiene un botón para acceder y otro para registrarse.

Si se accede se envía una acción que realiza una petición de *login* a *Firebase*. El *login* es un servicio propio de *Firebase*, de modo que si no lo hacemos no permite obtener información de la base de datos. Es una manera de simular el acceso a una aplicación.

Si durante el acceso ocurre cualquier problema o existen campos vacíos se muestra el mensaje de error pertinente.

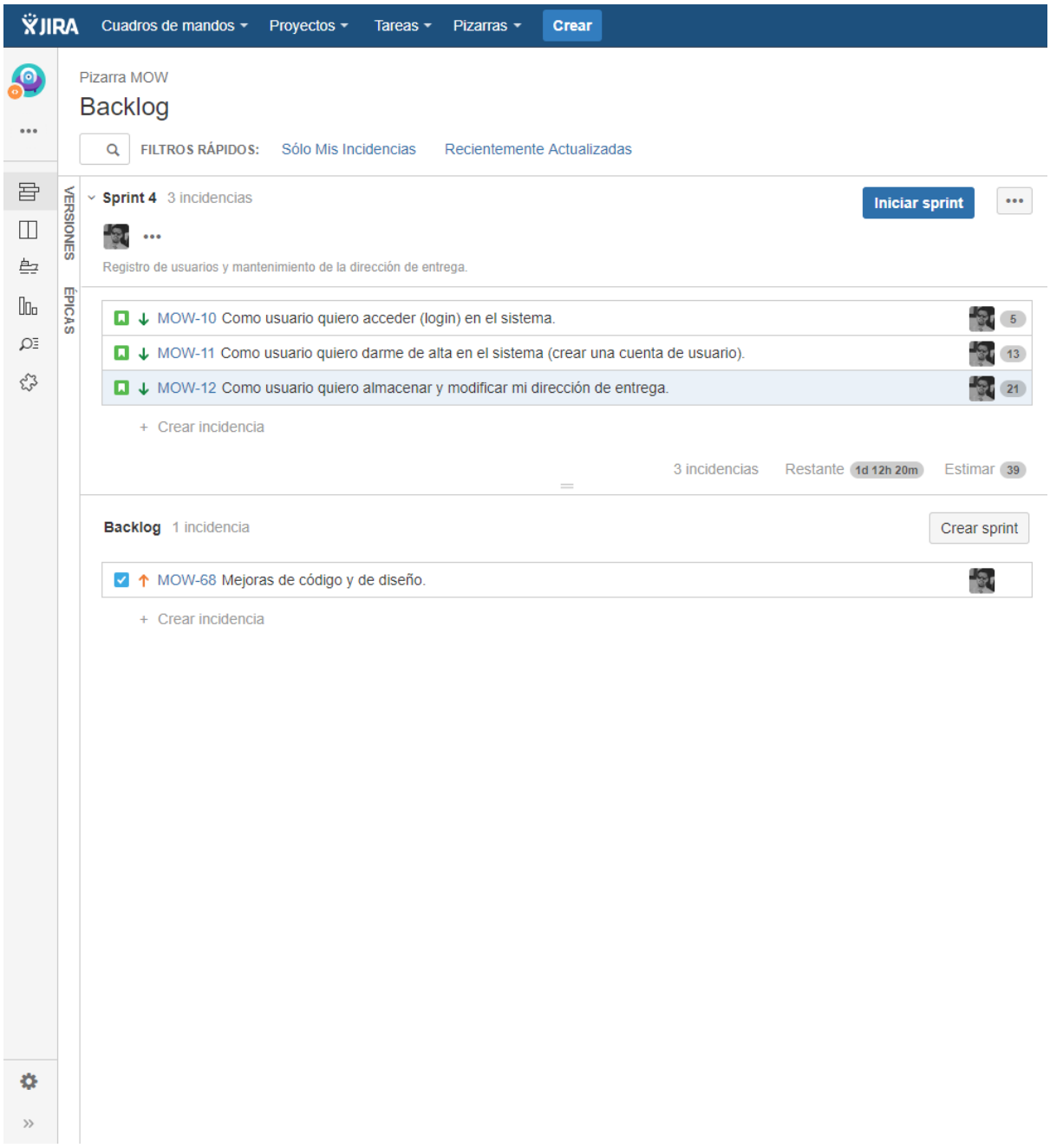


Figura 3.26: Planificación del cuarto sprint y estado de la pila del producto (Jira).

ID	EST	INCIDENCIAS	
MOW-10	**5**	**Como usuario quiero acceder (login) en el sistema.**	
MOW-51	60	**Prototipos**	Realizar los diseños gráficos para el acceso.
MOW-52	120	**Servicios**	Implementar el servicio para acceder al sistema (login).
MOW-53	120	**Componentes**	Implementar la pantalla de presentación con la lógica de comprobación de usuario conectado.
MOW-54	240	**Componentes**	Implementar la pantalla de acceso.
MOW-11	**13**	**Como usuario quiero darme de alta en el sistema (crear una cuenta de usuario).**	
MOW-55	80	**Prototipos**	Realizar los mockups de las pantallas para el registro.
MOW-56	20	**Firebase**	Modificar la base de datos para almacenar cuentas de usuario (nombre, apellidos y correo electrónico).
MOW-57	120	**Servicios**	Implementar el servicio para darse de alta en el sistema (sign in).
MOW-58	120	**Servicios**	Implementar el servicio para obtener los datos del usuario al conectarse.
MOW-59	240	**Componentes**	Implementar la pantalla de registro.
MOW-60	120	**Componentes**	Modificar la pantalla de acceso con la lógica de mostrar los valores después del registro.
MOW-12	**21**	**Como usuario quiero almacenar y modificar mi dirección de entrega.**	
MOW-61	80	**Prototipos**	Realizar los mockups de las pantallas para almacenar y modificar la dirección de entrega.
MOW-62	20	**Firebase**	Modificar la base de datos para almacenar una dirección de entrega.
MOW-63	120	**Servicios**	Implementar el servicio para desconectarse del sistema (logout).
MOW-64	120	**Servicios**	Implementar el servicio para registrar una dirección de usuario.
MOW-65	60	**Componentes**	Modificar la pantalla de restaurantes con el menú.
MOW-66	300	**Componentes**	Implementar la pantalla de menú de usuario.
MOW-67	240	**Componentes**	Implementar la pantalla de direcciones de usuario.

Figura 3.27: Planificación del cuarto sprint en subtareas.

Registro

La pantalla de registro consta de una cabecera con el título 'Registro' y un botón de regreso a la pantalla de acceso. Presenta un formulario con los campos de nombre, apellidos, correo electrónico y contraseña. La contraseña son dos campos para obligar a repetirla. En la parte de abajo hay un botón para realizar el registro.

Cuando un usuario realiza el registro se ejecuta una acción que realiza una llamada con los datos del formulario. Si el usuario no existe se crea una cuenta en *Firebase* con estos datos. El modo de crearse una cuenta es añadiendo un identificador de usuario a la base de datos del que cuelguen sus datos. Cada vez que el usuario acceda a la aplicación a través del *login* se tomarán estos datos.

Si no hay ningún error en el formulario, como un campo vacío o contraseñas distintas, se redirige al usuario a la pantalla de acceso y se le muestra un mensaje de éxito o de fallo según lo ocurrido. Si tiene éxito automáticamente se rellenan los campos de acceso para comodidad del usuario.

Menú de usuario

Para mantener la dirección de entrega es necesario un menú desde donde poder acceder. El

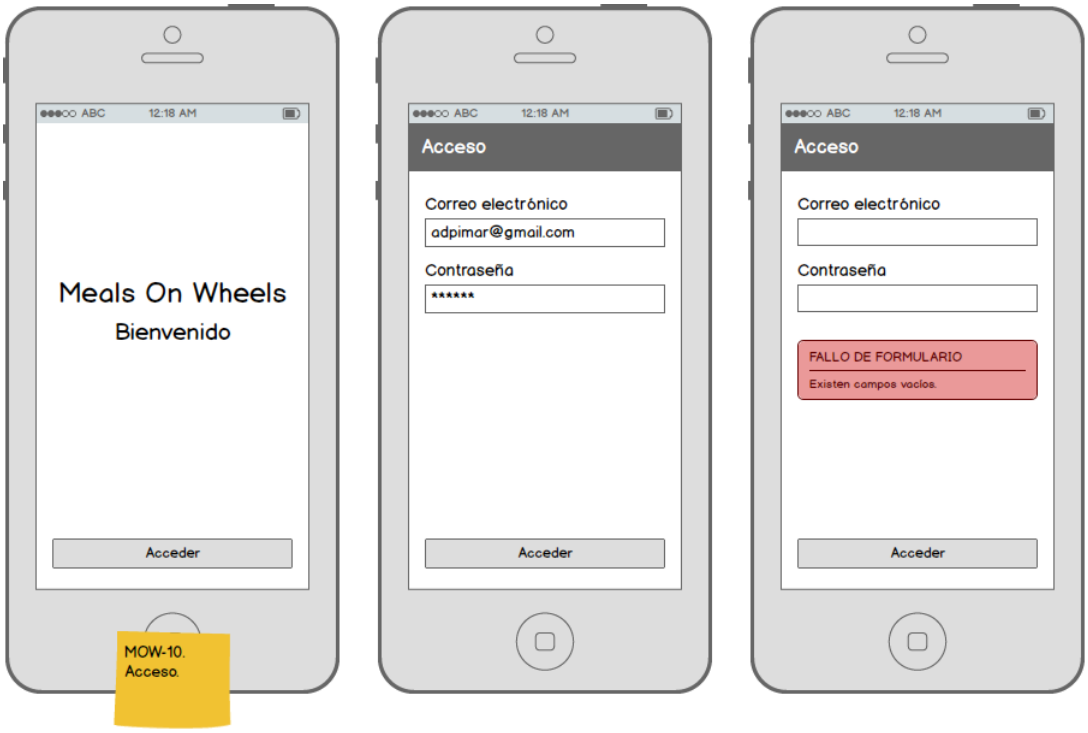


Figura 3.28: Bocetos del cuarto sprint (1/4).

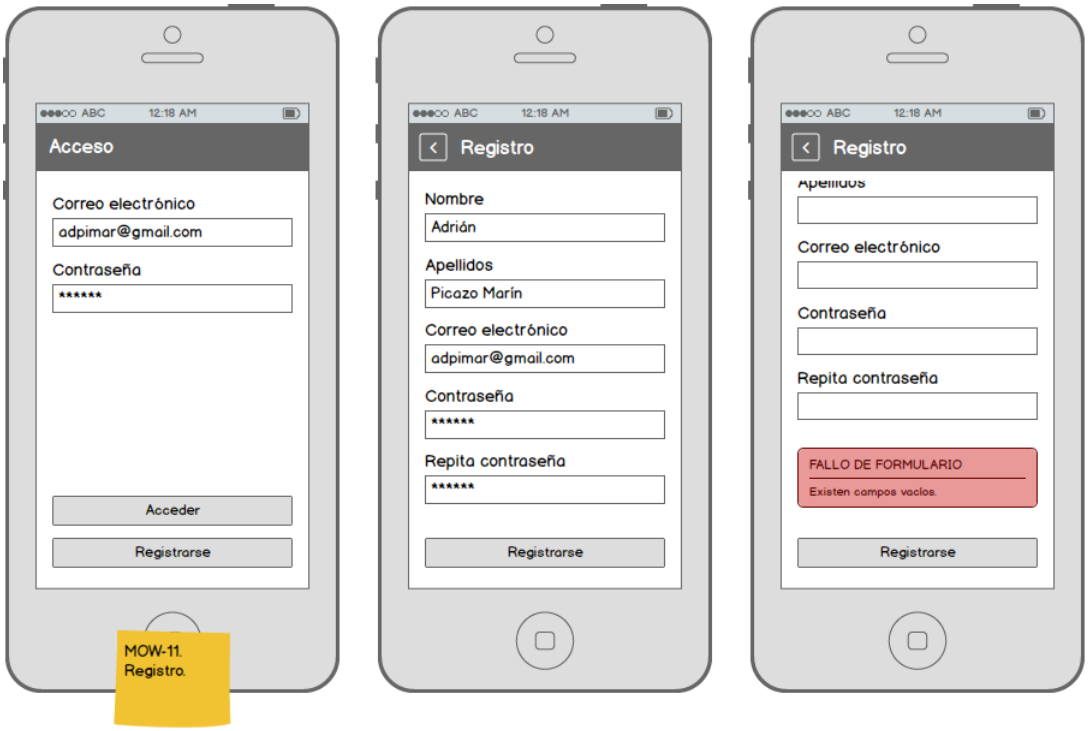


Figura 3.29: Bocetos del cuarto sprint (2/4).

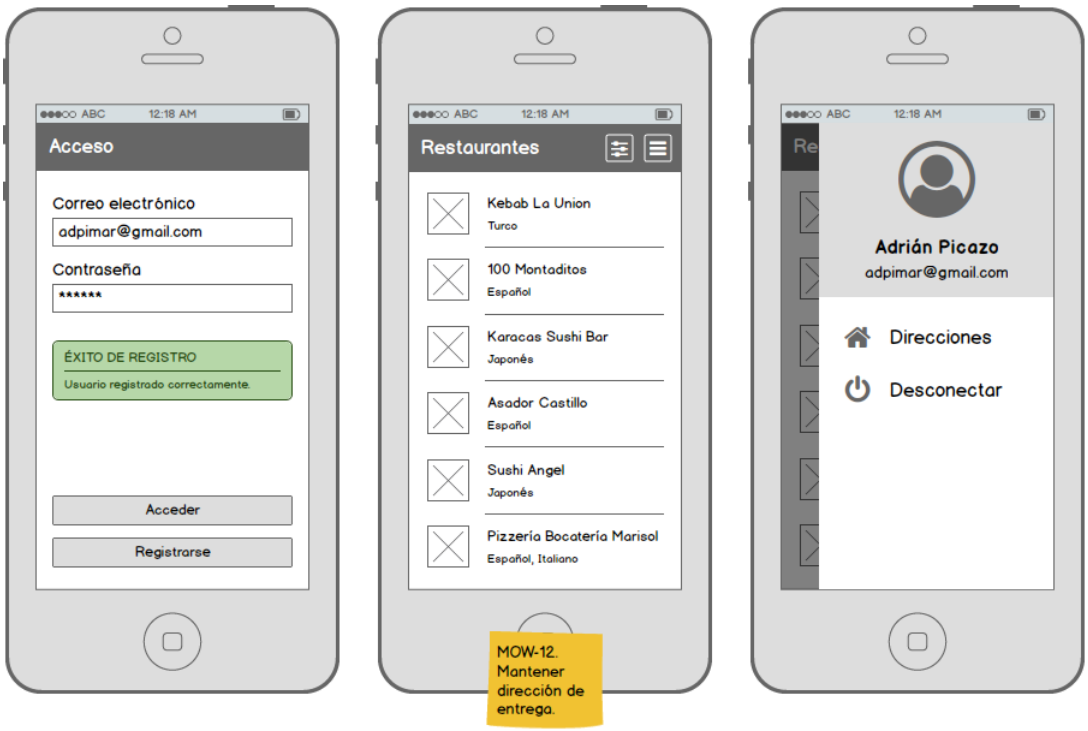


Figura 3.30: Bocetos del cuarto sprint (3/4).

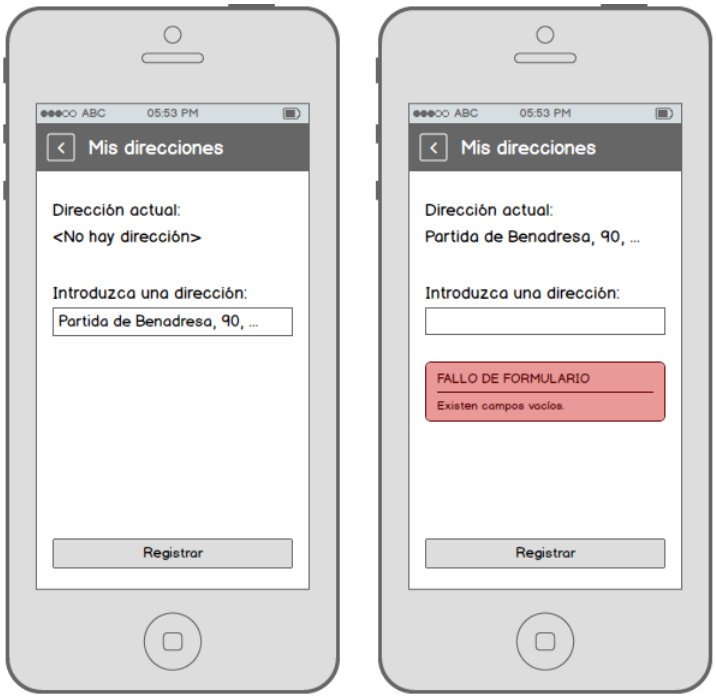


Figura 3.31: Bocetos del cuarto sprint (4/4).

menú de usuario es una pantalla que proporciona la biblioteca *react-native-router-flux*. Hasta ahora no hemos hablado del componente *router* que forma parte de esta biblioteca.

El componente *router* es una especie de gestor de pantallas. En él se indican todas las pantallas además de las modales y es lo que permite que podamos navegar de unas a otras. Además, permite incluir en las pantallas que lo requieran un menú lateral. El menú lateral podemos mostrarlo con el dedo simplemente arrastrándolo de izquierda a derecha o de derecha a izquierda, según donde lo hayamos situado. Para nuestro caso, también contamos con un botón en la cabecera de restaurantes desde el que se puede mostrar.

El menú de usuario está maquetado de tal modo que enseña la información del usuario en la parte superior y abajo una lista de submenús. De momento sólo está el submenú de direcciones de entrega y la opción de desconectar, que desconecta de *Firebase* y lleva a la pantalla de acceso.

Direcciones de entrega

La pantalla de direcciones tiene una cabecera con el título 'Mis direcciones' y un botón de regreso al menú. Tiene también un formulario con un único campo y un texto con la dirección actualmente guardada. Sólo es posible almacenar una dirección, y se almacena en la cuenta del usuario (cuelga del identificador del usuario en Firebase). Si no hay dirección no se muestra nada, y si se produce algún error en el formulario muestra el debido mensaje.

Esquema de navegación

En la Figura 3.32 mostramos el esquema de navegación entre pantallas donde se han añadido colores para diferenciar entre tres partes. Las pantallas en rojo se corresponden con toda la parte de los restaurantes y los pedidos comunes a todos los usuarios; las pantallas en verde, a la parte de acceso y registro; y las pantallas en azul, al menú de usuario y sus distintas opciones.

El inicio de la aplicación ahora es una pantalla más desde donde se accede a la pantalla de acceso. Si el usuario ya ha hecho un acceso previo (no ha habido un *logout*) directamente conduce a los restaurantes[8]. Desde el menú de usuario se vuelve al acceso si se hace un *logout*.

CRITERIOS DE ACEPTACIÓN

Los criterios para validar las tres historias de usuario son los que se describen a continuación:

MOW-10 Como usuario quiero acceder (*login*) en el sistema.

Pantalla de presentación

- Accesible automáticamente al inicio de la aplicación.
- Muestra un botón 'Acceder' que permite ir a la pantalla de acceso.
- Redirige automáticamente a la pantalla de restaurantes si el usuario no se ha desconectado (*logout*).

[8]Esta funcionalidad todavía no ha sido implementada. Se implementará más adelante, cuando se traten las notificaciones.

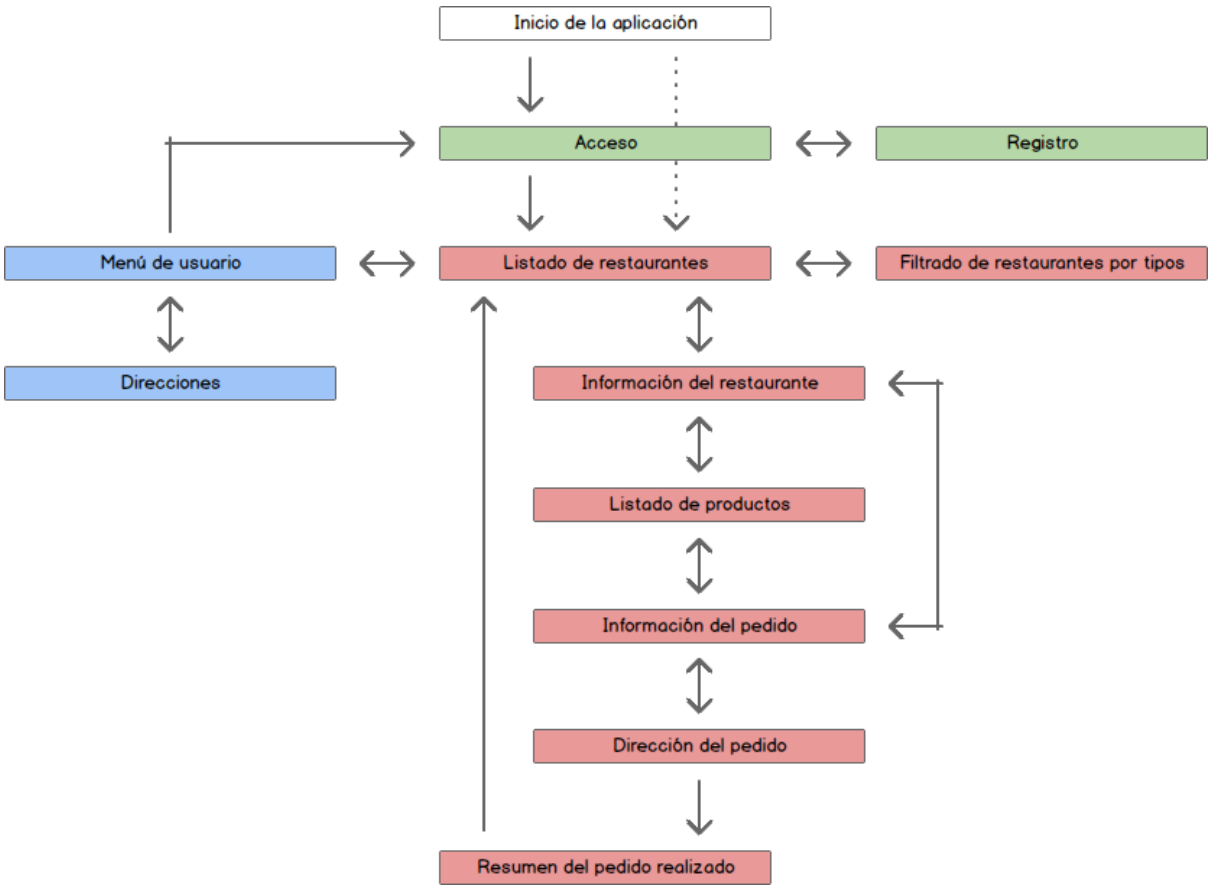


Figura 3.32: Esquema de navegación entre pantallas del cuarto sprint.

Pantalla de acceso

- Muestra una cabecera con el título 'Acceso'.
- Muestra un formulario que permite introducir el correo electrónico y la contraseña.
- Muestra textos de ejemplo (*placeholders*) en el formulario.
- Muestra un botón 'Acceder' que permite realizar el acceso.
- Notifica con un mensaje de fallo si al acceder existen campos del formulario vacíos.
- Notifica con un mensaje de fallo si al acceder no existe el usuario (o cualquier otro problema).
- Redirige a la pantalla de restaurantes si se ha efectuado el acceso con éxito.

MOW-11 Como usuario quiero darme de alta en el sistema (crear una cuenta de usuario).

Pantalla de registro

- Muestra una cabecera con el título 'Registro'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de acceso.

- Muestra un formulario que permite introducir el nombre, apellidos, correo electrónico y la contraseña.
- Muestra textos de ejemplo (*placeholders*) en el formulario.
- Permite indicar dos veces la contraseña para mayor seguridad.
- Muestra un botón 'Registrarse' que permite realizar el registro.
- Notifica con un mensaje de fallo si ambas contraseñas no coinciden al registrarse.
- Notifica con un mensaje de fallo si existen campos del formulario vacíos al registrarse.
- Notifica con un mensaje de fallo si ya existe el usuario (correo electrónico repetido) al registrarse.
- Redirige a la pantalla de acceso si se ha efectuado el registro con éxito.

Pantalla de acceso

- Muestra un botón 'Registrarse' que permite ir a la pantalla de registro.
- Notifica con un mensaje de éxito si se ha efectuado el registro con éxito.
- Muestra automáticamente el correo electrónico y contraseña registrados si se ha efectuado el registro con éxito.

MOW-12 Como usuario quiero almacenar y modificar mi dirección de entrega.

Pantalla de restaurantes

- Muestra un botón de menú en la cabecera que permite acceder a la pantalla de menú de usuario.
- Permite acceder también a la pantalla de menú de usuario arrastrando el dedo de derecha a izquierda en el lateral izquierdo.

Pantalla de menú de usuario

- Muestra por la derecha una pantalla lateral con dos secciones: perfil y menú.
- Muestra para el perfil una imagen del usuario, el nombre con apellidos y el correo electrónico.
- Muestra para el menú una opción Direcciones que redirige a la pantalla de direcciones de usuario.
- Muestra para el menú una opción Desconectar que permite desconectarse y volver a la pantalla de acceso.
- Permite volver a la pantalla de restaurantes pulsando fuera de la pantalla o arrastrándola hacia la derecha.

Pantalla de direcciones de usuario

- Muestra una cabecera con el título 'Mis direcciones'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de menú de usuario.
- Muestra la dirección actualmente almacenada en la cuenta del usuario.
- Muestra un formulario que permite introducir la dirección de entrega.

- Muestra textos de ejemplo (*placeholders*) en el formulario.
- Muestra un botón 'Registrar' que permite almacenar la nueva dirección.
- Notifica con un mensaje de fallo si existen campos del formulario vacíos al registrar.
- No permite registrar más de una dirección de entrega para un mismo usuario.

SEGUIMIENTO DEL SPRINT

1. Planificación del cuarto sprint:
   a) Elección de las historias de usuario.
   b) Subdivisión de las historias en subtareas.
   c) Estimación en minutos de las subtareas.
2. Inicio del cuarto sprint.
   a) Diseño de los prototipos.
   b) Implementación de los servicios y componentes.
   c) Validación de los criterios de aceptación.
3. Revisión del incremento del tercer y cuarto sprint con el cliente.
4. Cierre del cuarto sprint.
5. Revisión de la pila del producto.
6. Adición de nuevas historias de usuario (aspectos y sugerencias).

En este sprint hemos completado un total de 39 puntos de historia en menos de una semana. Esto supone una velocidad aproximada de 49 puntos/semana. El buen rendimiento se debe, entre otras cosas, a que no ha surgido ningún problema. En la revisión del incremento hemos presentado lo del anterior sprint también.

La gráfica de la Figura 3.33 muestra el progreso de este sprint. No hay más historias de usuario, por lo que los aspectos sobre notificaciones, analíticas e i18n comentados en el seguimiento del segundo sprint y las recomendaciones comentadas en el seguimiento del tercero los hemos incluido como nuevas historias. Además, hemos añadido otra nueva historia relacionada con un mapa de *Google* a petición del cliente.

3.6.6. Sprint 5: Direcciones, pedidos y visualización de contraseñas.

Este sprint tiene como meta **el registro de usuarios y el mantenimiento de la dirección de entrega (almacenar y modificar sólo una)**. Detallamos sus principales características a continuación:

Figura 3.33: Gráfica de trabajo por hacer (en puntos de historia) del cuarto sprint.

Mantenimiento de direcciones de entrega, almacenamiento de mis pedidos y visualización de contraseñas.

Intervalo de tiempo	10 de abril - 19 de abril (5ª Quincena)
Horas de prácticas	49 horas (7 jornadas)
Incidencias incluidas	MOW-69, MOW-70, MOW-71
Incidencias completadas	MOW-69, MOW-70, MOW-71
Estimación total	39 puntos de historia (completados: 39)
Velocidad	35,102 puntos/semana

PLANIFICACIÓN DEL SPRINT

Las historias de usuario escogidas para este sprint son las que siguen:

MOW-69 Como usuario quiero mantener más de una dirección de entrega para poder escoger entre ellas a la hora de realizar un pedido.

MOW-70 Como usuario quiero almacenar todos mis pedidos realizados para poder escoger entre ellas a la hora de realizar un pedido.

MOW-71 Como usuario quiero tener la opción de visualizar mi contraseña a la hora de acceder y registrarme en el sistema para saber con certeza qué contraseña estoy escribiendo.

La Figura 3.34 muestra la planificación del quinto sprint y el estado de la pila del producto previo al inicio del sprint. Una vez seleccionadas las historias de usuario, se descomponen en subtareas y se estiman en tiempo. La Figura 3.35 muestra la descomposición en subtareas y la estimación en minutos (las historias están estimadas en puntos de historia).

DISEÑO DE PROTOTIPOS

El diseño de la historia MOW-69 consta de cinco bocetos (Figuras 3.36 y 3.37) con el listado de direcciones y la manera de añadir y eliminarlas. El diseño de la historia MOW-70 consta de cuatro bocetos (Figuras 3.37 y 3.38) con los pedidos realizados. Finalmente, el diseño de la historia MOW-71 consta de dos bocetos (Figura 3.39) con el campo contraseña modificado para ver lo que se escribe.

DETALLES DE IMPLEMENTACIÓN

Direcciones de entrega

La pantalla de direcciones de entrega de usuarios se ha modificado para mostrar una lista de elementos. Cada elemento es un componente que muestra una dirección y tiene asociado un botón para eliminar esa dirección. Estas direcciones se obtienen de *Firebase*, donde ahora el usuario puede almacenar más de una dirección. Debajo de la lista está el botón para añadir una dirección nueva, que automáticamente muestra una ventana modal.

El modal consta de un formulario para introducir una dirección. Si el campo esta vacío al añadirla muestra un error. Si la dirección está repetida, el error lo muestra en la pantalla de direcciones.

Para entender bien cómo funciona vamos a profundizar en la lógica de *Redux*. Cuando un usuario le da a aceptar genera una función *thunk*. Recordemos que una función *thunk* es asíncrona, de modo que podemos y, de hecho, despachamos varias acciones.

La primera acción sirve para indicar que estamos llevando a cabo la petición de registro. Esta acción llega a todos los reductores, pero por el tipo únicamente afectará al reductor de la pantalla de direcciones. El reductor cambiará una propiedad llamada *loading* a verdadero, con lo que se activará un *spinner*. Mientras todo esto ocurre, se está realizando la petición. Si tiene éxito, la función despachará otra acción con la nueva lista de direcciones (la anterior con la dirección añadida). El reductor reescribirá los datos, cambiará la propiedad *loading* para que desaparezca el *spinner* y los componentes afectados volverán a pintarse. Si no tiene éxito, pasa algo parecido pero mostrando el error.

Dirección de entrega del pedido.

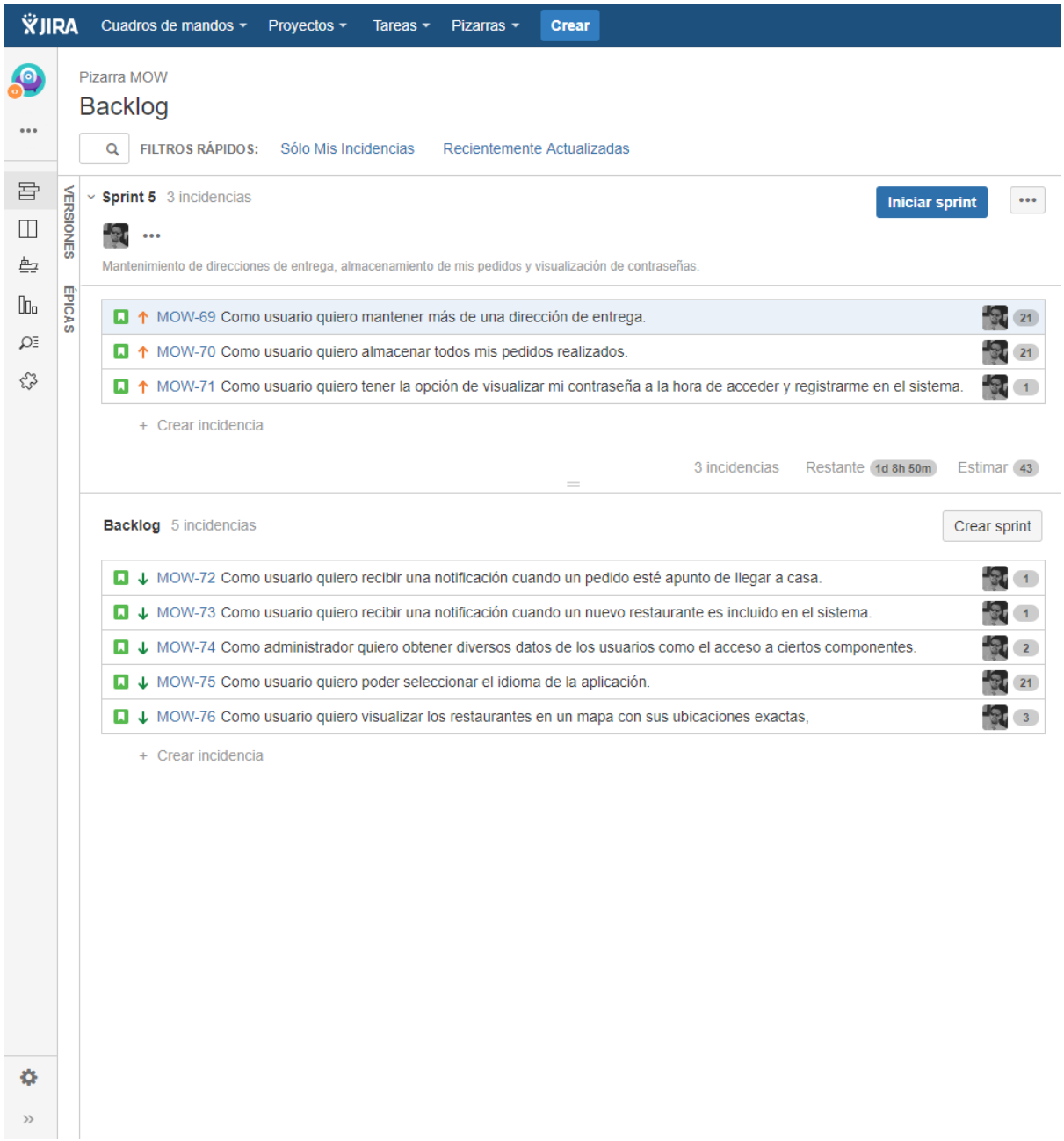


Figura 3.34: Planificación del quinto sprint y estado de la pila del producto (Jira).

ID	EST	INCIDENCIAS	
MOW-69	**21**	**Como usuario quiero mantener más de una dirección de entrega.**	
MOW-77	100	**Prototipos**	Realizar los mockups de las pantallas para mantener direcciones de entregas y escoger entre ellas.
MOW-78	20	**Firebase**	Modificar la base de datos para almacenar más de una dirección de entrega.
MOW-79	120	**Servicios**	Implementar el servicio para obtener las direcciones de entrega de un usuario.
MOW-80	60	**Servicios**	Modificar el servicio de registro de una dirección para poder registrar varias.
MOW-81	120	**Servicios**	Implementar el servicio para eliminar una dirección de entrega de un usuario.
MOW-82	240	**Componentes**	Modificar la pantalla de direcciones de usuario.
MOW-83	120	**Componentes**	Implementar la pantalla modal de añadir dirección.
MOW-84	240	**Componentes**	Modificar la pantalla de dirección del pedido con la lógica de elección.
MOW-70	**21**	**Como usuario quiero almacenar todos mis pedidos realizados.**	
MOW-85	80	**Prototipos**	Realizar los mockups de las pantallas para consultar los pedidos.
MOW-86	20	**Firebase**	Modificar la base de datos para almacenar los pedidos realizados de un usuario.
MOW-87	120	**Servicios**	Implementar el servicio para obtener los pedidos realizados por un usuario.
MOW-88	120	**Servicios**	Implementar el servicio para almacenar un pedido.
MOW-89	30	**Componentes**	Modificar la pantalla de menú de usuario con la opción Pedidos.
MOW-90	240	**Componentes**	Implementar la pantalla de pedidos del usuario.
MOW-91	240	**Componentes**	Implementar la pantalla de información del pedido.
MOW-71	**1**	**Como usuario quiero tener la opción de visualizar mi contraseña a la hora de acceder y registrarme en el sistema.**	
MOW-92	40	**Prototipos**	Realizar los mockups de las pantallas para visualizar las contraseñas.
MOW-93	60	**Componentes**	Modificar el input contraseña con la lógica de visualización.

Figura 3.35: Planificación del quinto sprint en subtareas.

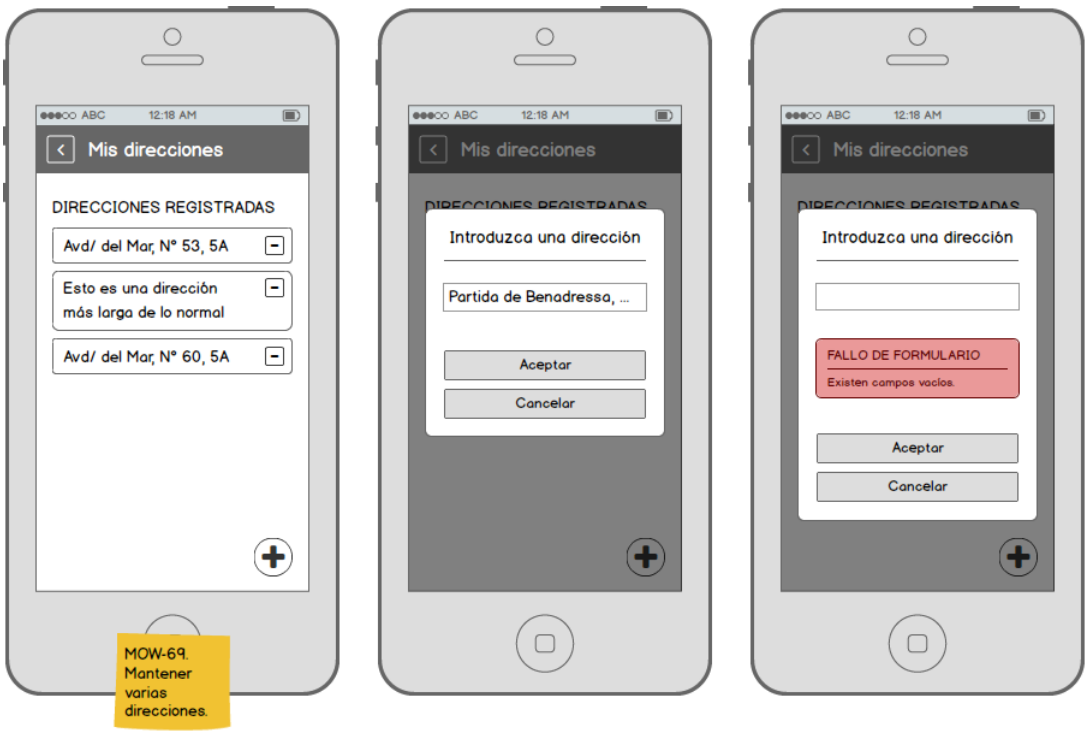


Figura 3.36: Bocetos del quinto sprint (1/4).

Figura 3.37: Bocetos del quinto sprint (2/4).

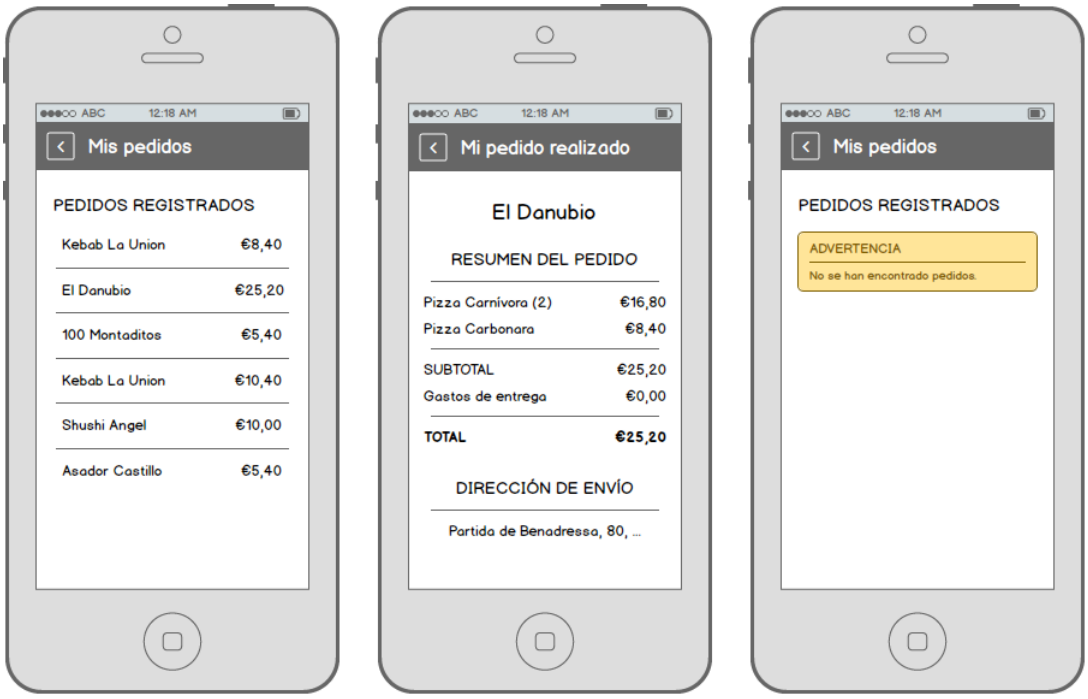


Figura 3.38: Bocetos del quinto sprint (3/4).

Figura 3.39: Bocetos del quinto sprint (4/4).

En esta pantalla hemos facilitado la usabilidad añadiendo una lista con la direcciones registradas que pueden ser seleccionadas, de tal forma que la dirección se copia en el campo del formulario.

Pedidos realizados

La pantalla de pedidos realizados comprende una cabecera con el título 'Mis pedidos' y un botón de regreso, además de una lista con los pedidos realizados y sus precios. Los elementos son seleccionables y conducen a la pantalla de información del pedido. Si no hay pedidos se muestra un mensaje de advertencia.

Asimismo, hemos añadido un submenú de pedidos en el menú de usuario.

Información del pedido realizado

La pantalla de información del pedido realizado reutiliza los componentes de la pantalla del pedido. Cambia el título de la cabecera, que es 'Mi pedido realizado'.

Visualizar contraseña

Para visualizar la contraseña hemos modificado el componente del campo contraseña. Todos los campos vienen de un componente reutilizable que creamos en su momento, en el que se le pueden indicar la etiqueta, el *placeholder*, etc. En este caso, para añadir la lógica de visualización hemos creado un componente distinto. Este componente incluye un botón a la derecha que cuando es pulsado enseña lo que hay en el campo sin asteriscos.

Esquema de navegación

En la Figura 3.40 mostramos el esquema de navegación entre pantallas donde hemos añadido nuevas pantallas al menú de usuario.

CRITERIOS DE ACEPTACIÓN

Los criterios para validar las tres historias de usuario son los que se describen a continuación:

MOW-69 Como usuario quiero mantener más de una dirección de entrega.

Pantalla de direcciones de usuario

- Muestra un listado de direcciones de entrega.
- Muestra un botón de eliminación para cada dirección que permite eliminarla del listado
- Muestra un botón de adición que redirige a la pantalla de añadir dirección.
- Muestra una nueva dirección si el registro de la dirección ha tenido éxito.
- Muestra un mensaje informativo si no existen direcciones registradas.

Pantalla de añadir dirección

- Muestra una pantalla modal con el título 'Introduzca una dirección'.
- Muestra un formulario que permite introducir una dirección de entrega.
- Muestra una dirección de ejemplo (*placeholder*) en el formulario.
- Muestra un botón 'Aceptar' que permite registrar la nueva dirección.
- Muestra un botón 'Cancelar' que permite volver atrás sin hacer nada más.
- Muestra un mensaje de fallo si existen campos vacíos en el formulario al aceptar.

Pantalla de dirección de pedido

- Muestra un formulario que permite introducir una dirección de entrega.
- Muestra una dirección de ejemplo (*placeholder*) en el formulario.
- Muestra un listado con las direcciones de entrega predefinidas por el usuario.
- Permite introducir una dirección predefinida automáticamente en el formulario al presionar sobre ella.
- Muestra un botón 'Finalizar el pedido' que permite finalizarlo.
- Muestra un mensaje de fallo si existen campos vacíos en el formulario al finalizar el pedido.

MOW-70 Como usuario quiero almacenar todos mis pedidos realizados.

Pantalla de menú de usuario

- Muestra para el menú una opción Pedidos que redirige a la pantalla de pedidos del usuario.

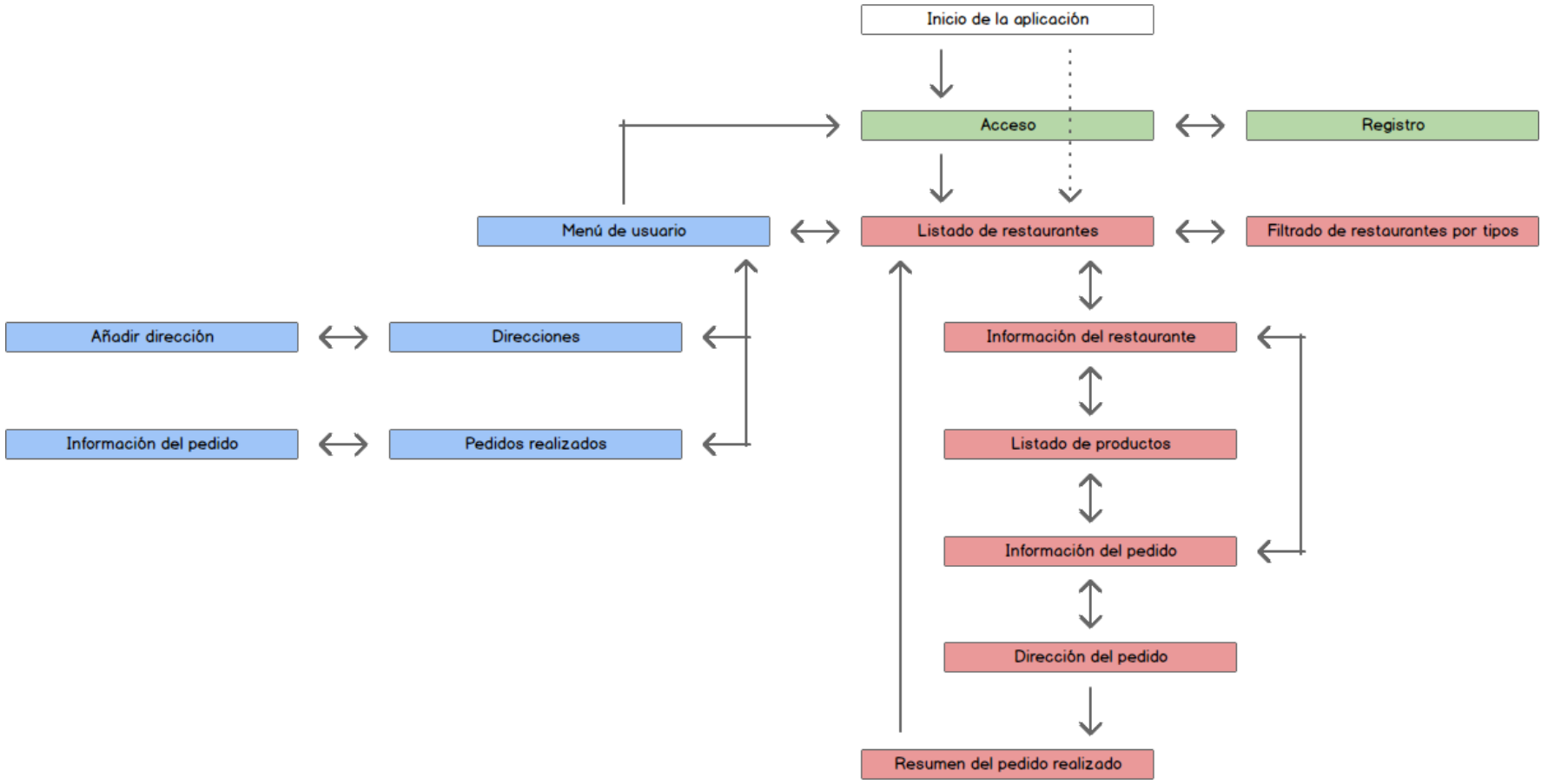


Figura 3.40: Esquema de navegación entre pantallas del quinto sprint.

Pantalla de pedidos del usuario

- Muestra una cabecera con el título 'Mis pedidos'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de menú de usuario.
- Muestra un listado de los pedidos realizados.
- Muestra para cada pedido el nombre del restaurante y el total del pedido en euros.
- Muestra un mensaje informativo si no existen pedidos realizados.
- Permite seleccionar cada uno de los pedidos (redirige a la pantalla de información del pedido).

Pantalla de información del pedido

- Muestra una cabecera con el título 'Mi pedido realizado'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de pedidos del usuario.
- Muestra una primera sección con el nombre del restaurante.
- Muestra una segunda sección con el resumen del pedido: productos, cantidad, otros gastos, precios, subtotal y total.
- Muestra una tercera sección con la dirección de envío.

Pantalla de dirección del pedido

- Almacena automáticamente el pedido una vez se finalice.

MOW-71 Como usuario quiero tener la opción de visualizar mi contraseña a la hora de acceder y registrarme en el sistema.

Pantalla de acceso

- Muestra para la contraseña un botón que permite visualizarla.

Pantalla de registro

- Muestra para cada contraseña un botón que permite visualizarla.

SEGUIMIENTO DEL SPRINT

1. Planificación del quinto sprint:
   a) Elección de las historias de usuario.
   b) Subdivisión de las historias en subtareas.
   c) Estimación en minutos de las subtareas.
2. Inicio del quinto sprint.
   a) Diseño de los prototipos.

Figura 3.41: Gráfica de trabajo por hacer (en puntos de historia) del quinto sprint.

   b) Implementación de los servicios y componentes.
   c) Validación de los criterios de aceptación.

3. Revisión del incremento del quinto sprint con el cliente.
4. Cierre del quinto sprint.
5. Revisión de la pila del producto.
6. Creación de la colección de servicios en *Postman* para Roberto.

Al comienzo de este sprint ha surgido un problema respecto a la versión de *iOS*. Debido a las mejoras que hicimos en el tercer sprint, donde cambiamos la biblioteca nativa que enlazaba con *Firebase* por otra, la configuración nativa en *iOS* ha presentado varios errores que no han permitido compilar su versión. Después de una horas discutiendo con los compañeros de departamento sobre como solucionarlo, finalmente se ha optado por la decisión de prescindir de dicha versión. Es decir, que a partir de este momento sólo comprobaremos la aplicación para la versión *Android*.

A pesar de ello, el desarrollo del sprint se ha llevado sin más contratiempos. En la gráfica de la Figura 3.41 lo podemos apreciar.

Con el buen rendimiento que estamos teniendo queremos volver a intentar conectar la aplicación con el *back-end* de Roberto. Para ello, los días 18 y 19 de abril, después del cierre del sprint y la revisión de la pila del producto, los hemos dedicado a crear una colección de pruebas con la herramienta *Postman*. Estas pruebas se realizan contra los servicios web especificados. Permiten ver los datos de salida en las llamadas según los datos de entrada indicados. Como

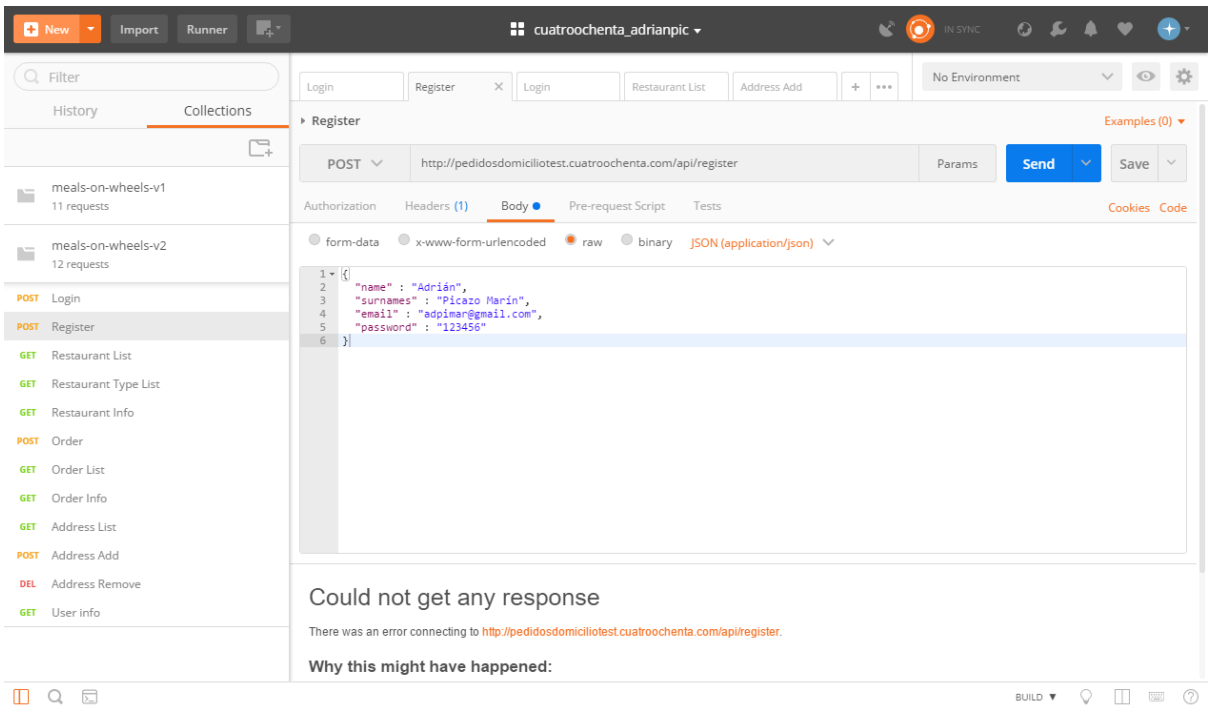


Figura 3.42: Herramienta Postman con la colección de peticiones para el proyecto de pedidos de comida a domicilio. En el panel izquierdo se listan todas las peticiones. En el panel derecho se muestra diversa información de la petición seleccionada: URL, método (POST, GET, PUT, etc), cuerpo, respuesta, etc.

a estas alturas pueden diferir los servicios de Roberto de los propios hemos decidido utilizar esta colección de pruebas como documentación para tener una interfaz común. De ese modo, los datos que yo le envío serán los indicados en la colección de *Postman*, y para saber los datos que me envía basta con ejecutar la prueba. En la figura 3.42 se muestra la colección que creé para tal efecto.

3.6.7. Sprint 6: Notificaciones, analíticas, i18n y mapa interactivo.

Este sprint tiene como meta **gestionar notificaciones recibidas, registrar analíticas, la internacionalización y emplear un mapa interactivo para localizar los restaurantes del listado**. Como características principales detallamos a continuación:

Intervalo de tiempo	20 de abril - 27 de abril (6ª Quincena)
Horas de prácticas	40 horas (6 jornadas)
Incidencias incluidas	MOW-72, MOW-73, MOW-74, MOW-75, MOW-76
Incidencias completadas	MOW-72, MOW-73, MOW-74, MOW-75, MOW-76
Estimación total	28 puntos de historia (completados: 28)
Velocidad	28,000 puntos/semana

PLANIFIFACIÓN DEL SPRINT

Las historias de usuario escogidas para este sprint son las que siguen:

MOW-72 Como usuario quiero recibir una notificación cuando un pedido está a punto de llegar a casa para preparar el pago.

MOW-73 Como usuario quiero recibir una notificación cuando un nuevo restaurante es incluido en el sistema para enterarme al momento de las novedades.

MOW-74 Como administrador quiero obtener diversos datos de los usuarios como el acceso a ciertos componentes para realizar informes que me ayuden a mejorar el sistema.

MOW-75 Como usuario quiero poder seleccionar el idioma de la aplicación para mi comodidad y entendimiento.

MOW-76 Como usuario quiero visualizar los restaurantes en un mapa con sus ubicaciones exactas para saber qué restaurantes tengo más cerca y cómo llegar a ellos.

La Figura 3.43 muestra la planificación del sexto sprint y el estado de la pila del producto previo al inicio del sprint. Una vez seleccionadas las historias de usuario, se descomponen en subtareas y se estiman en tiempo. La Figura 3.44 muestra la descomposición en subtareas y la estimación en minutos (las historias están estimadas en puntos de historia).

DISEÑO DE PROTOTIPOS

El diseño de la historia MOW-75 consta de tres bocetos (Figura 3.45) con las pantallas para la selección de idioma. El diseño de la historia MOW-76 consta de tres bocetos (Figura 3.46) con el listado de los restaurantes mediante un mapa interactivo. Para el resto de historias de usuario no hay bocetos.

DETALLES DE IMPLEMENTACIÓN

Notificaciones con *OneSignal*

Para las notificaciones hemos usado el servicio de *OneSignal*. Éste permite crear y enviar notificaciones *push* a todos los dispositivos que uno quiera. Para ello es necesario disponer de una cuenta *OneSignal* y registrar los dispositivos. En nuestro caso hemos registrado únicamente un dispositivo *Android*. Los dispositivos tienen que ser reales, no sirve que sean virtuales como los emuladores. Para la gestión de las notificaciones es necesario instalar una biblioteca llamada *react-native-onesignal*.

La figura 3.47 es una captura de la aplicación web con un listado de notificaciones que hemos realizado de prueba. Hemos creado dos notificaciones que la aplicación debe tratar: (1) el registro de un nuevo restaurante y (2) la llegada del pedido de un usuario a la dirección.

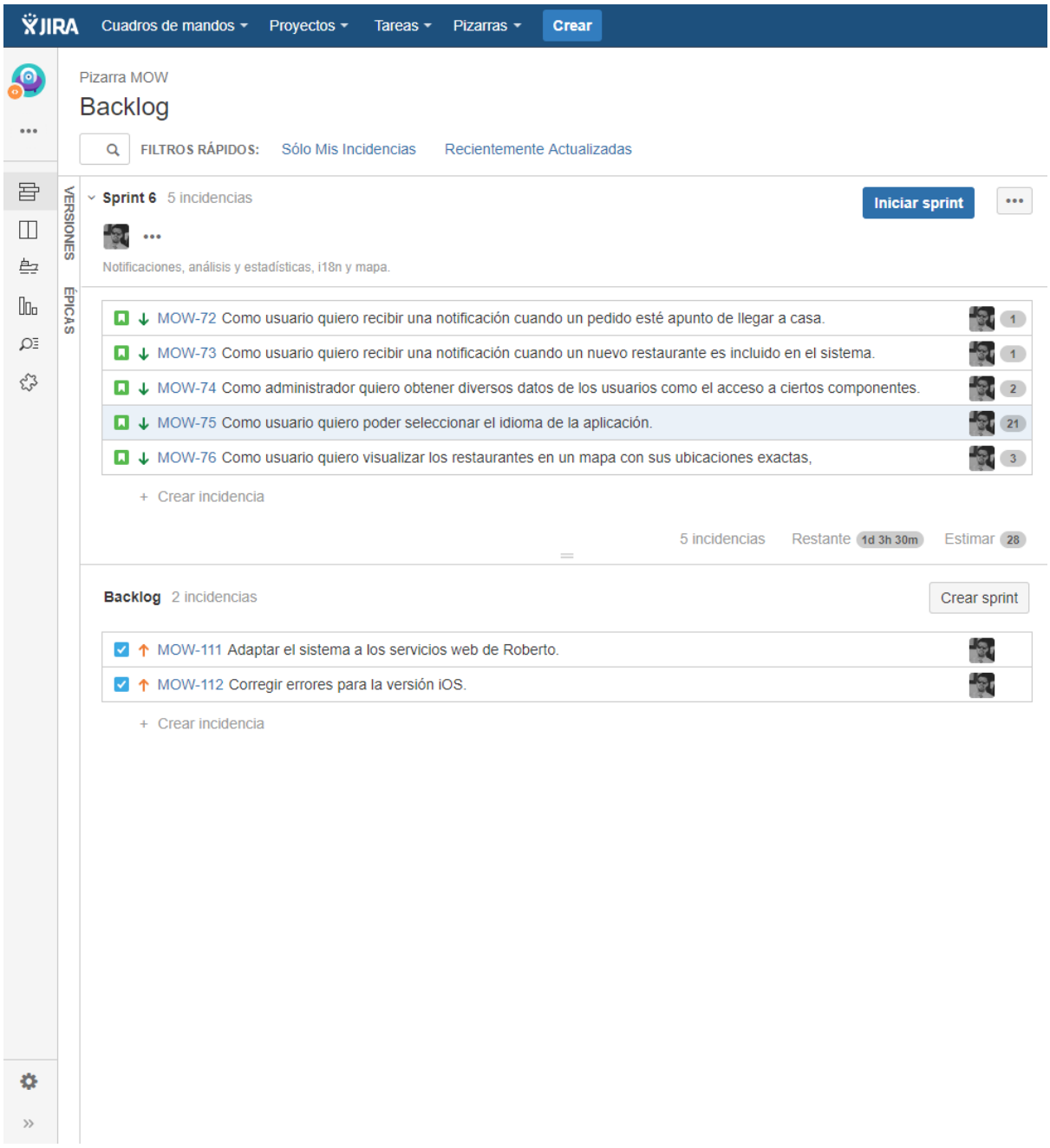


Figura 3.43: Planificación del sexto sprint y estado de la pila del producto (Jira).

ID	EST	INCIDENCIAS	
MOW-72	**1**	**Como usuario quiero recibir una notificación cuando un pedido esté apunto de llegar a casa.**	
MOW-94	120	**Servicios**	Configurar un proyecto en OneSignal y enlazarlo con el sistema.
MOW-95	10	**Servicios**	Diseñar una plantilla de notificación para un pedido.
MOW-73	**1**	**Como usuario quiero recibir una notificación cuando un nuevo restaurante es incluido en el sistema.**	
MOW-96	10	**Servicios**	Diseñar una plantilla de notificación para un nuevo restaurante.
MOW-74	**2**	**Como administrador quiero obtener diversos datos de los usuarios como el acceso a ciertos componentes.**	
MOW-97	120	**Servicios**	Configurar un proyecto en Google Analytics y enlazarlo con el sistema.
MOW-98	120	**Componentes**	Modificar todas las pantallas, botones y ciertos eventos para registrar estadísticas de uso.
MOW-75	**21**	**Como usuario quiero poder seleccionar el idioma de la aplicación.**	
MOW-99	60	**Prototipos**	Realizar los mockups de las pantallas para la selección de idioma.
MOW-100	20	**Firebase**	Modificar la base de datos para almacenar el idioma en la cuenta del usuario.
MOW-101	180	**Servicios**	Implementar el servicio para cambiar y almacenar el idioma del sistema.
MOW-102	30	**Componentes**	Modificar la pantalla de menú de usuario con la opción Configuración.
MOW-103	120	**Componentes**	Implementar la pantalla de configuración.
MOW-104	120	**Componentes**	Implementar la pantalla modal de selección de idioma con la lógica de selección.
MOW-105	300	**Componentes**	Abstraer el texto de todos y cada uno de los componentes mediante variables en una clase aparte (es-ES).
MOW-106	60	**Componentes**	Realizar una traducción de todos los textos en español al inglés (en-US).
MOW-76	**3**	**Como usuario quiero visualizar los restaurantes en un mapa con sus ubicaciones exactas.**	
MOW-107	60	**Prototipos**	Realizar los mockups de las pantallas para visualizar los restaurantes mediante un mapa.
MOW-108	20	**Firebase**	Modificar la base de datos para almacenar la posición.
MOW-109	60	**Componentes**	Modificar la pantalla de restaurantes con el botón del mapa.
MOW-110	240	**Componentes**	Implementar la pantalla de mapa de restaurantes con la lógica de mostrar el enlace.

Figura 3.44: Planificación del sexto sprint en subtareas.

Dado que el envío de notificaciones debe encargarse la parte *backend* y emplear *OneSignal* a través de *Firebase* es muy complicado, hemos tenido que enviar las notificaciones de manera manual. Para no tener que crearlas cada vez, *OneSignal* permite crear plantillas que pueden reutilizarse y que facilitan las pruebas.

Una notificación hay que tratarla diferente dependiendo de si un usuario está autenticado o no lo está. Si lo está la notificación puede direccionarle a la pantalla correspondiente; si no lo está, puede llevarle a la de acceso. En nuestro caso, la notificación sobre un restaurante nuevo lleva a su pantalla de información y la notificación de que ha llegado el pedido lleva a la pantalla del pedido realizado. Eso en caso de estar autenticado, sino lleva a la pantalla de acceso.

La verdadera dificultad estriba en obtener los datos del usuario autenticado en el momento que gestionamos una notificación y estos datos no están, como en el caso de salir de la aplicación sin desautenticarte. Para eso hemos repensado la lógica de acceso de tal modo que cuando un usuario accede, además de almacenar sus datos en el reductor y lo hacemos en otro almacén denominado *asyncStorage*. Este almacén perdura incluso si se apaga la aplicación, lo que soluciona el problema. Si el usuario se desautentica automáticamente se destruye su información.

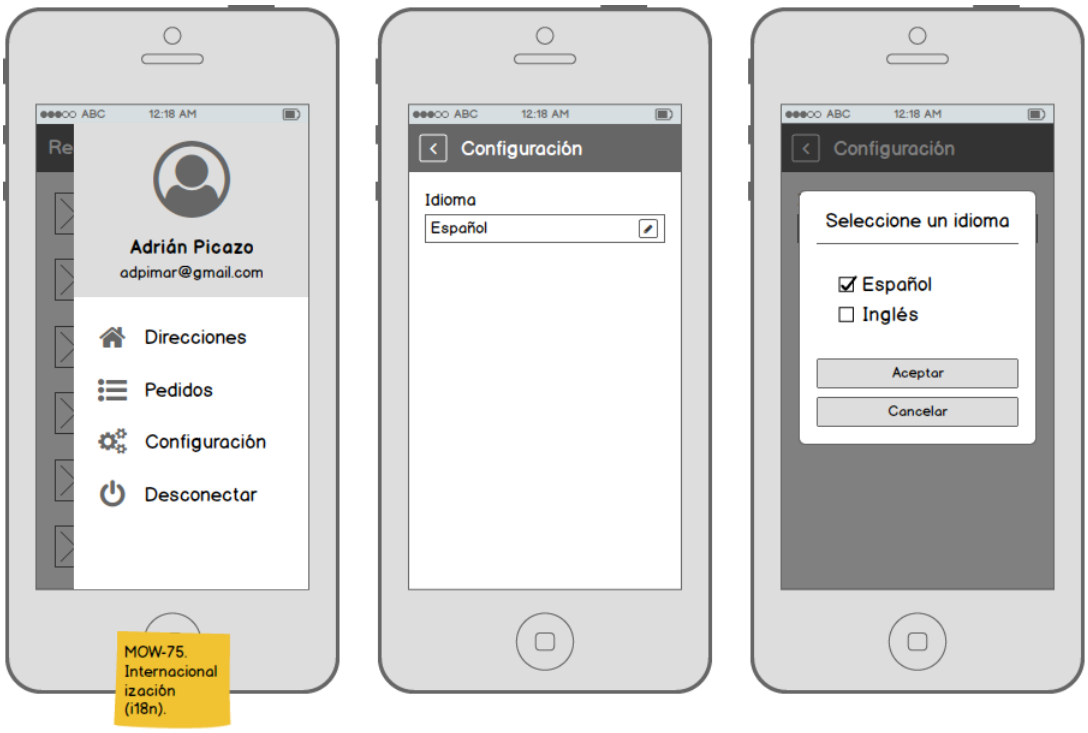


Figura 3.45: Bocetos del sexto sprint (1/2).

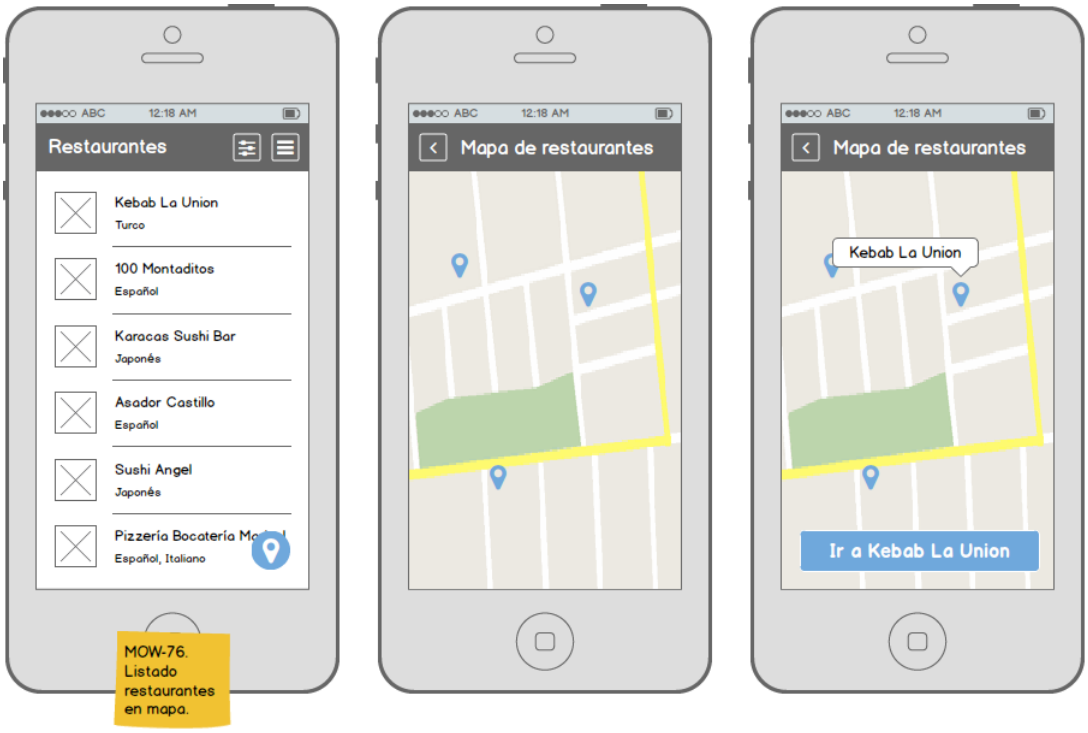


Figura 3.46: Bocetos del sexto sprint (1/2).

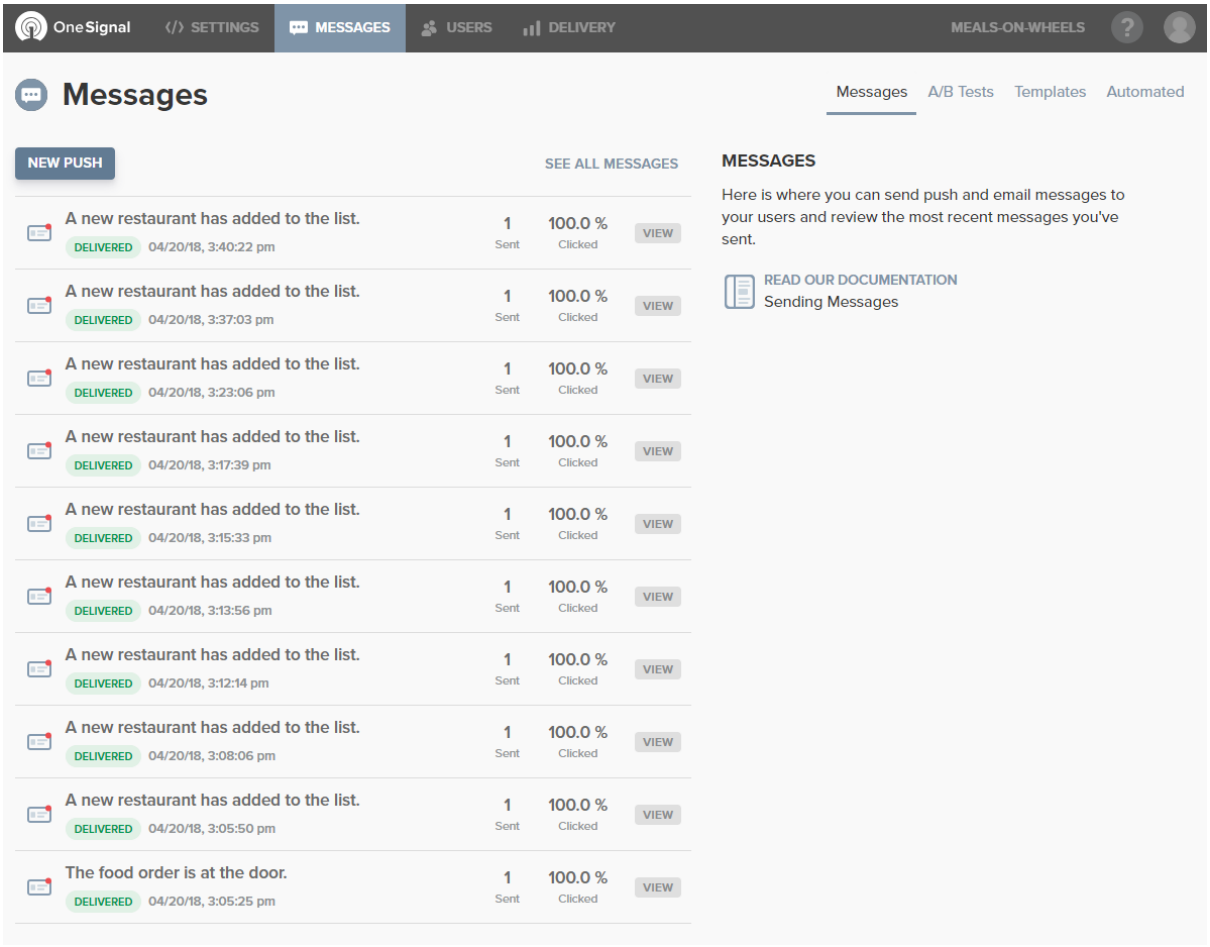


Figura 3.47: Listado de las notificaciones *push* enviadas a un dispositivo en *OneSignal*.

Analíticas con *Google Analytics*

Para analizar el tráfico y comportamiento de los usuarios hemos utilizado el servicio de *Google Analytics*, y para ello hemos necesitado la biblioteca *react-native-google-analytics-bridge*.

Los datos se toman de forma automática. Basta con enlazar cada componente con el servicio. En nuestro caso hemos enlazado todas las pantallas y algún que otro componente, como el panel informativo del pedido.

Para hacernos una mejor idea, en la figuras 3.48 y 3.49 mostramos dos pantallas del tablero web (*dashboard*) de *Google Analytics*.

Internacionalización

Respecto a la internacionalización de la aplicación hemos llevado a cabo dos tareas: (1) abstraer a la aplicación del texto propio e (2) implementar la lógica necesaria para recordar el idioma escogido por el usuario.

Para abstraer a la aplicación del texto propio hemos tenido que aislar cada mensaje en

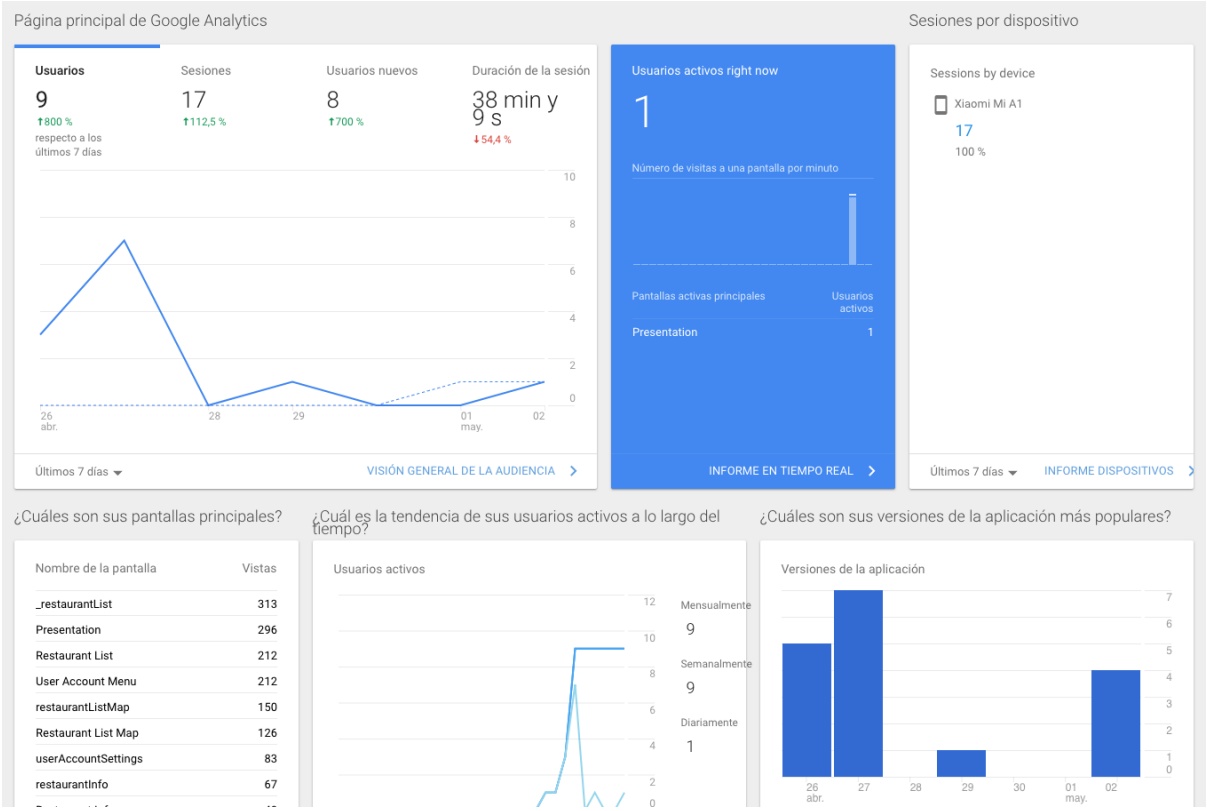


Figura 3.48: Página principal del *dashboard* de *Google Analytics*. En ella se muestran las veces que han sido vistas las pantallas registradas, los usuarios activos, los dispositivos que han accedido a la aplicación, etc. La página principal puede configurarse añadiendo y quitando módulos.

variables. Con texto propio nos referimos a todo aquel texto que no dependa de los servicios del *backend* sino de la propia aplicación: títulos de las cabeceras, etiquetas de campos de formularios, títulos de secciones, etc; y no la descripción de los restaurantes, que están almacenadas en la base de datos, o los mensajes de error que lanzan los servicios.

En principio, la aplicación estaba configurada en un único idioma, el español, pero la hemos preparado para que acepte también el inglés. Para ello hemos tenido que traducir cada mensaje aislado empleando la misma variable. De esta forma existen dos valores para cada una. Por ejemplo, la variable WELCOME tiene un valor en español, 'Bienvenido', y otro en inglés, 'Welcome'. En función de qué idioma se escoja automáticamente cada componente cambiará sus textos de un valor a otro, traduciéndose toda la aplicación al momento.

Para la tarea de recordar el idioma escogido por el usuario hemos partido de dos casos: qué ocurre cuando el usuario esté autenticado y qué ocurre cuando no lo esté.

En el caso de que no lo esté, la aplicación busca por orden en los idiomas de preferencia en el teléfono móvil hasta encontrar uno para la que está preparada. Por ejemplo, si nuestro dispositivo está configurado con los idiomas catalán y español, en ese orden, primero busca si tiene el catalán como opción de traducción. Si no lo tiene, continua con el español. En caso de tenerlo se iniciará con ese idioma. Si se da el caso de que ninguno de los idiomas del dispositivo existe en la aplicación, ésta se inicia con un idioma por defecto (el español).

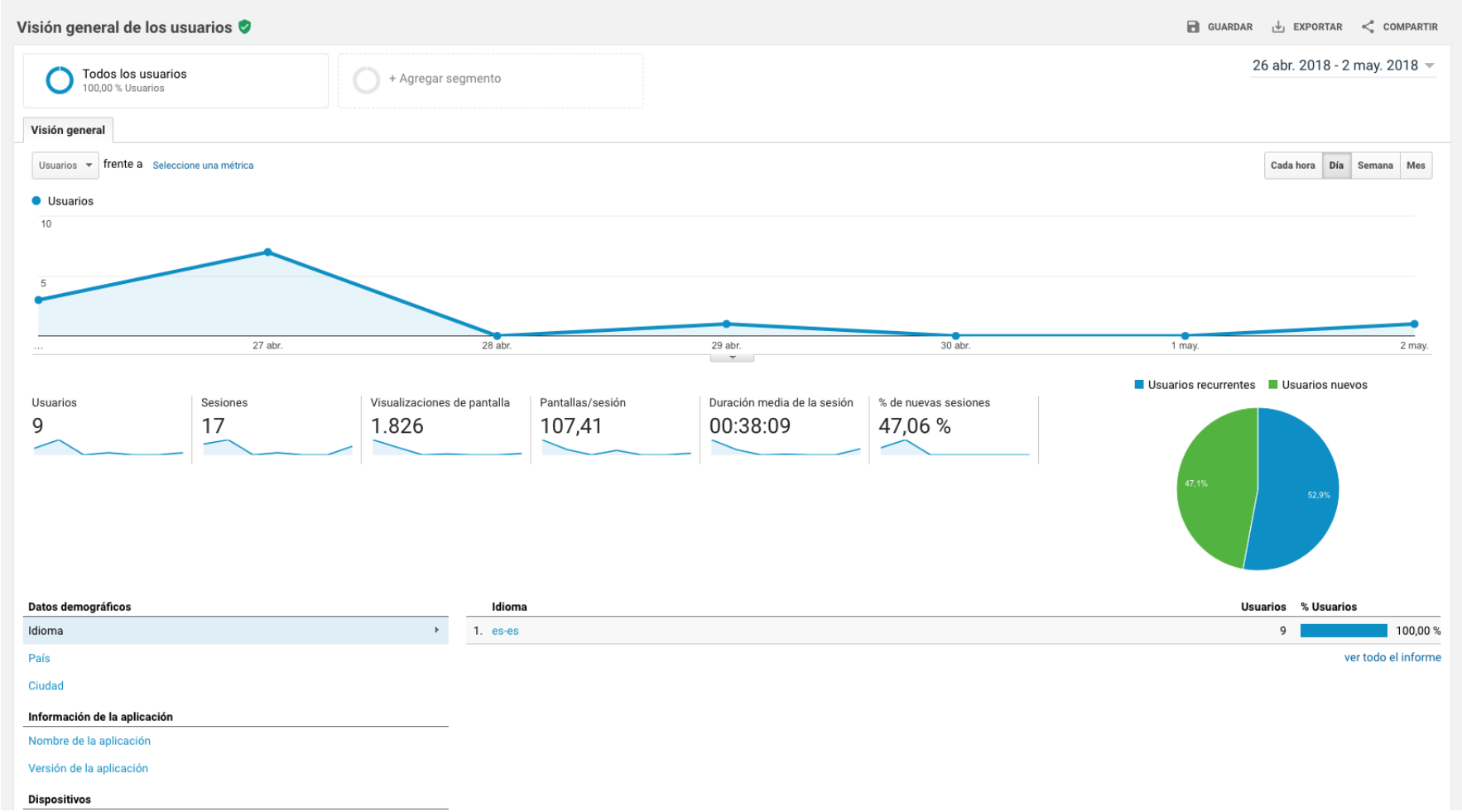


Figura 3.49: Pantalla de 'Visión general' del *dashboard* de *Google Analytics* con datos de los usuarios que acceden a la aplicación.

Mientras, en el caso de que sí lo esté (o se autentique en el momento), la aplicación busca el idioma que tiene registrado en su cuenta de usuario. En un principio al usuario se le asigna el español, pero una vez dentro de la aplicación puede cambiar el idioma en un menú de configuración para tal efecto. La aplicación recordará el cambio la siguiente vez que acceda.

Mapa de restaurantes

En la pantalla de restaurantes hemos implementado un botón que lleva a la pantalla del mapa de restaurantes. En dicho mapa se muestran todas las localizaciones exactas (y ficticias) de los restaurantes del listado mediante marcadores. Estos marcadores indican automáticamente el nombre del restaurante al ser pulsados. Además, hemos diseñado lo necesario para que también muestren un enlace con el que acceder al restaurante.

Toda la lógica para acceder a un restaurante y controlar que no se haga más de un pedido a la vez, que ya estaba implementada en la pantalla del listado, está también implementada para esta nueva pantalla. Por ejemplo, recordemos que si se accede a un restaurante, se escogen unos productos para un pedido, se vuelve atrás y se escoge otro restaurante, aparece un modal que avisa de que el pedido se reiniciará si se continúa accediendo a otro restaurante. Ocurre lo mismo para esta nueva pantalla.

Para implementar el mapa de *Google* ha sido necesario utilizar la biblioteca *react-native-maps*.

Esquema de navegación

En la Figura 3.50 mostramos el esquema de navegación entre pantallas donde hemos incluido una nueva opción al menú de usuario y el mapa de restaurantes.

CRITERIOS DE ACEPTACIÓN

Los criterios para validar las tres historias de usuario son los que se describen a continuación:

MOW-72 Como usuario quiero recibir una notificación cuando un pedido está a punto de llegar a casa.

General

- Recibe una notificación sobre el pedido tanto dentro como fuera de la aplicación (apagada).
- Redirige a la pantalla de información del pedido si el usuario está autenticado.
- Redirige a la pantalla de acceso si el usuario no está autenticado.

MOW-73 Como usuario quiero recibir una notificación cuando un nuevo restaurante es incluido en el sistema.

General

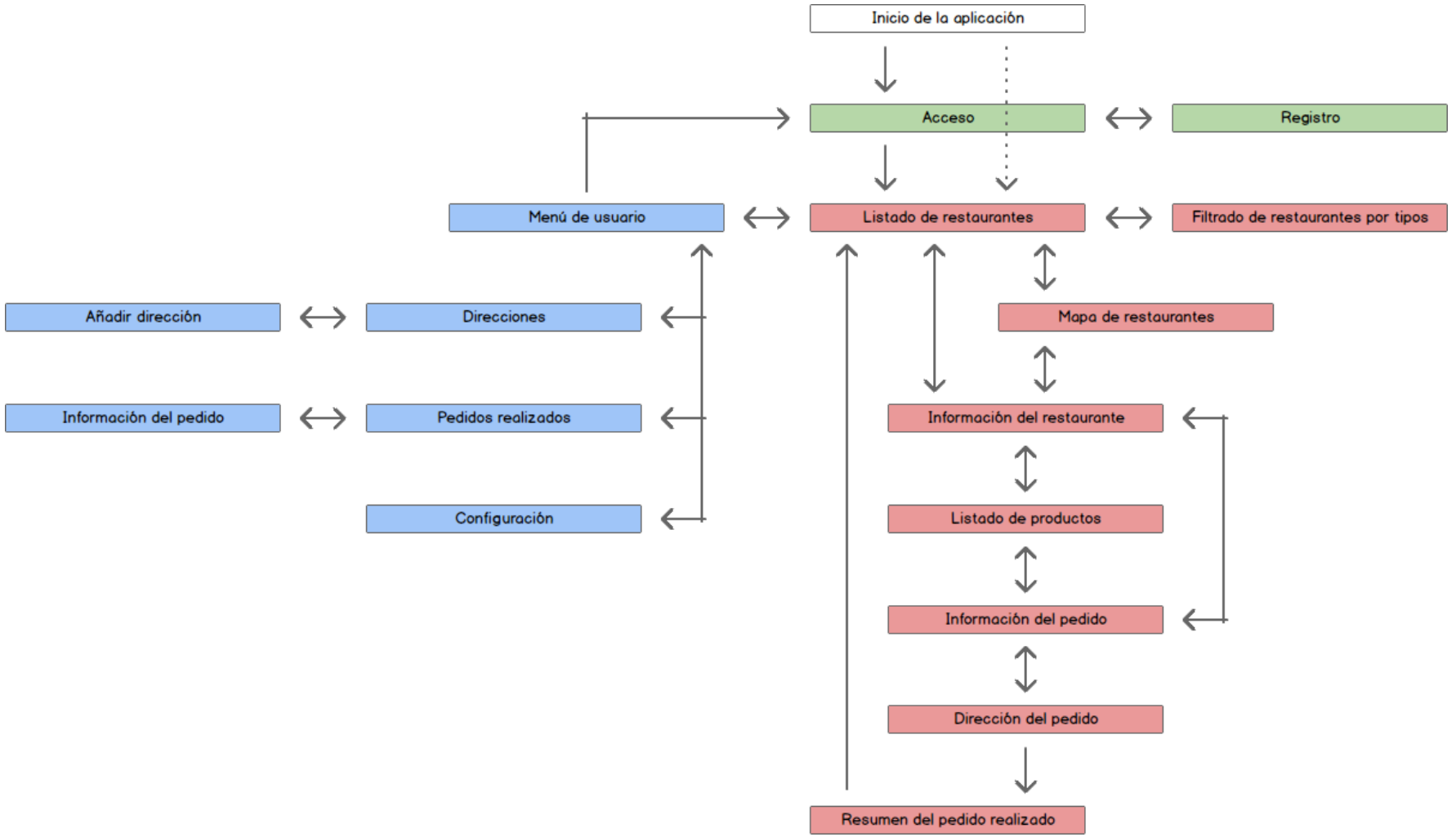


Figura 3.50: Esquema de navegación entre pantallas del sexto sprint.

- Recibe una notificación sobre el nuevo restaurante tanto dentro como fuera de la aplicación (apagada).
- Redirige a la pantalla de información del restaurante si el usuario está autenticado.
- Redirige a la pantalla de acceso si el usuario no está autenticado.

MOW-74 Como administrador quiero obtener diversos datos de los usuarios como el acceso a ciertos componentes.

General

- Registra información de acceso para todas y cada una de las pantallas.
- Registra información sobre si es pulsado (se lanza el evento) el panel informativo del pedido.
- Registra información sobre si es pulsado el botón para finalizar el pedido.

MOW-75 Como usuario quiero poder seleccionar el idioma de la aplicación.

General

- Muestra la aplicación en el idioma por defecto si el usuario no está autenticado y no tiene ninguno de los idiomas del sistema indicados en el dispositivo.
- Muestra la aplicación en uno de los idiomas del dispositivo si el usuario no está autenticado y coincide con uno de los idiomas del sistema.
- Muestra la aplicación en el idioma escogido en cuenta de usuario si éste está autenticado.
- Al crearse una cuenta de usuario el idioma toma el valor por defecto o del dispositivo en caso de existir en el sistema.
- Por defecto el idioma es el español (si no coincide el idioma del dispositivo con alguno del sistema será este el valor escogido).
- Permite cambiar el idioma por cualquiera de los que ofrece el sistema.

Pantalla de configuración

- Muestra una cabecera con el título 'Configuración'.
- Muestra un botón de regreso en la cabecera que permite volver a la pantalla de menú de usuario.
- Muestra una opción de idioma con el idioma escogido.
- Muestra un botón de edición que permite ir a la pantalla de selección de idioma.

Pantalla de selección de idioma

- Muestra una pantalla modal con el título 'Seleccione un idioma'.
- Muestra un listado de los idiomas disponibles (mínimo español e inglés).
- Muestra un botón 'Aceptar' que cambia todo el sistema al idioma seleccionado.

- Muestra un botón 'Cancelar' que vuelve a la pantalla de configuración sin hacer nada más.
- No permite seleccionar más de un idioma.
- Recuerda la opción configurada para un usuario (almacenado el idioma en la cuenta de usuario).

MOW-76 Como usuario quiero visualizar los restaurantes en un mapa con sus ubicaciones exactas.

Pantalla de restaurantes

- Muestra un botón en la esquina inferior derecha que permite ir a la pantalla de mapa de restaurantes.

Pantalla de mapa de restaurantes

- Muestra una cabecera con el título 'Mapa de restaurantes'.
- Muestra un botón de regreso que permite volver a la pantalla de restaurantes.
- Muestra un marcador por cada restaurantes del listado y que tenga una posición.
- Muestra un enlace al restaurante al seleccionar cualquiera de los marcadores que permite ir a la pantalla de información del restaurante.
- Muestra como posición de partida la ciudad de Castellón de la Plana.
- Permite hacer lo mismo a través del mapa que del listado (misma lógica) y se muestran los mismos avisos (pantalla modal de reinicio de pedido).

SEGUIMIENTO DEL SPRINT

1. Instalación de *Reactotron*.
2. Planificación del sexto sprint:
   a) Elección de las historias de usuario.
   b) Subdivisión de las historias en subtareas.
   c) Estimación en minutos de las subtareas.
3. Inicio del sexto sprint.
   a) Diseño de los prototipos.
   b) Implementación de los servicios y componentes.
   c) Validación de los criterios de aceptación.
4. Revisión del incremento del sexto sprint con el cliente.
5. Cierre del sexto sprint.

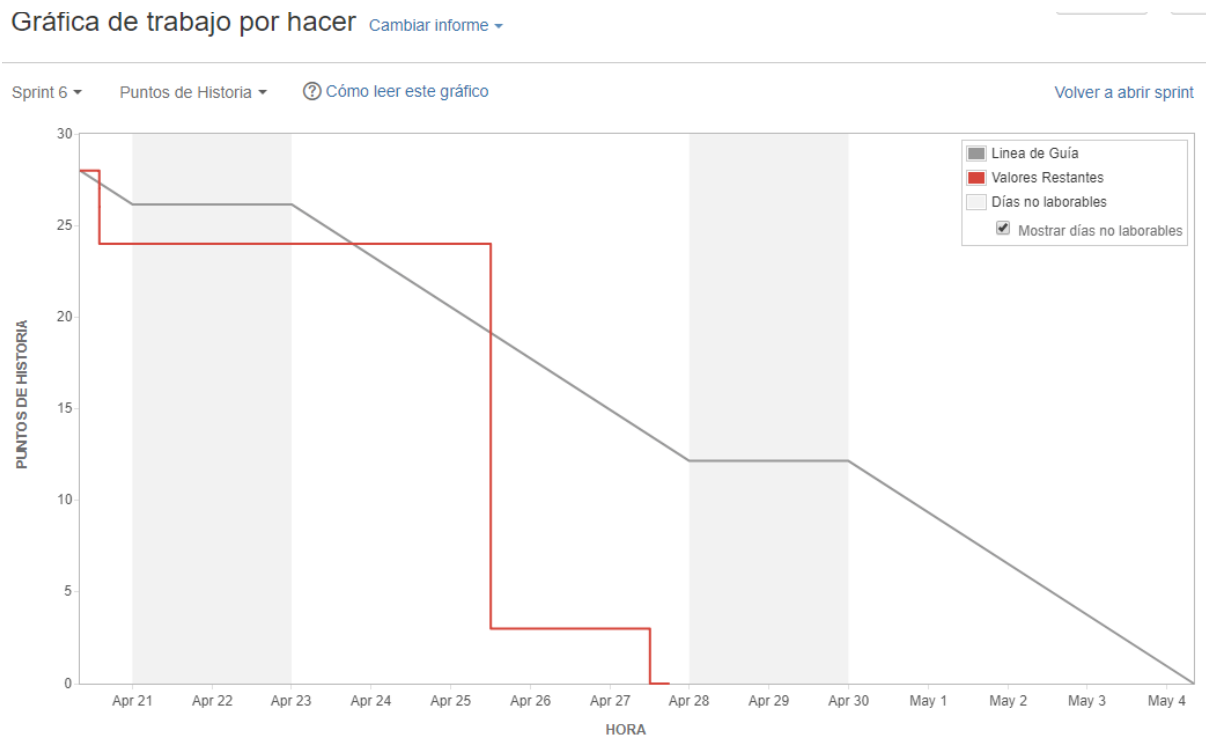


Figura 3.51: Gráfica de trabajo por hacer (en puntos de historia) del sexto sprint.

Durante este sprint hemos completado el resto de historias de usuario que faltaban. Las historias sobre notificaciones y analíticas se han completado en un breve periodo de tiempo. En la gráfica de la Figura 3.51 estas historias corresponden al primer escalón, completadas el 21 de abril.

Las siguientes días los hemos empleado para la internacionalización. Ha sido lo más costoso del sprint puesto que había que localizar, abstraer y traducir todo el texto que hasta entonces teníamos. Además, tuvimos problemas con la herramienta de depuración.

Hasta este momento hemos empleado la consola del navegador *Google Chrome* para ver las salidas de datos y los errores. Es una herramienta bastante útil y fácil de utilizar, pero que de manera frecuente falla: la conexión entre la aplicación y la consola no se establece y no se muestra nada por pantalla. Como solución, hemos instalado una herramienta denominada *Reactotron* que funciona y permite ver de manera muy vistosa los datos y errores (Figura 3.52).

A pesar de las dificultades el ritmo ha sido bueno. Finalmente, los días 26, 26 y 28 de abril hemos implementado la última historia de usuario. El resultado de la revisión del incremento ha sido muy satisfactorio. El cliente (Sergio Aguado) ha quedado sorprendido por el rendimiento en el desarrollo y el acabado final de la aplicación.

Como todavía faltan horas de estancia en prácticas me he propuesto adaptar la aplicación a los servicios de Roberto y solucionar los fallos en *iOS*. Para ello he creado otra aplicación a partir de esta para realizar todos los cambios. Así nos aseguramos de tener una versión 100 % estable mientras modificamos la otra.

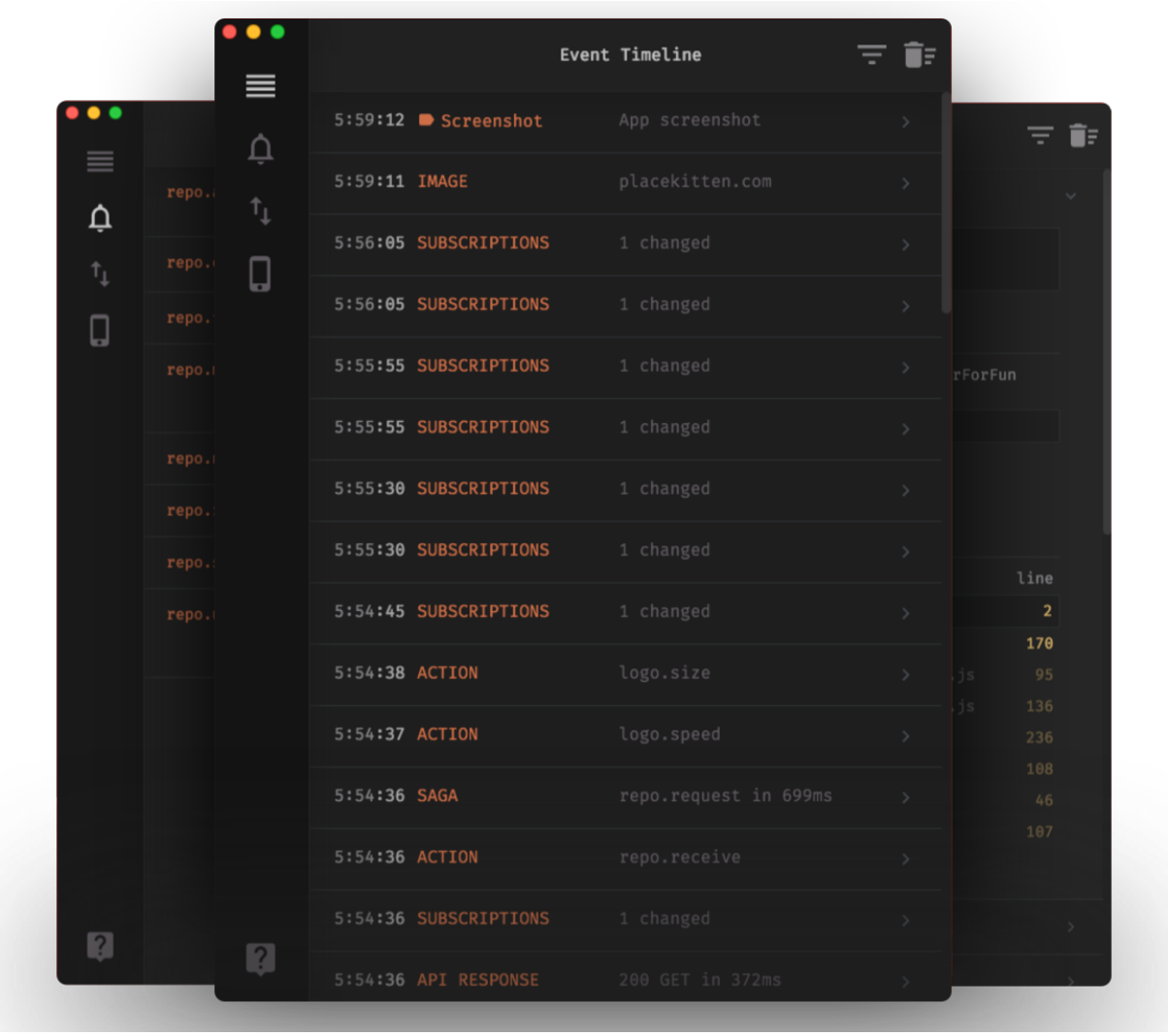


Figura 3.52: Capturas oficiales de *Reactotron*. Fuente de la imagen: documentación oficial [20]

3.6.8. Balance general

Haciendo balance de todos los sprints, más del 80 % han resultado exitosos. Como podemos ver en la gráfica de la Figura 3.53 hemos conseguido completar todos aquellos puntos de historia con los que nos comprometimos, a excepción del primer sprint (MOW-3). Es lógico tratándose del primer contacto.

Desde una perspectiva por quincenas, la gráfica de la Figura 3.54 muestra los puntos de historia completados frente a las horas de prácticas invertidas. La primera quincena no tiene puntos de historias completados porque se corresponde con la etapa de formación. La segunda quincena, además de ser el primer sprint, observamos que se empleó poco tiempo, lo que influyó a que dicho sprint no resultase tan exitoso y sólo completásemos parte de los puntos de historia. Por lo general, a más horas mayor puntos de historia completados.

Por último, volviendo a una perspectiva por sprints, en la gráfica de la Figura 3.55 mostramos

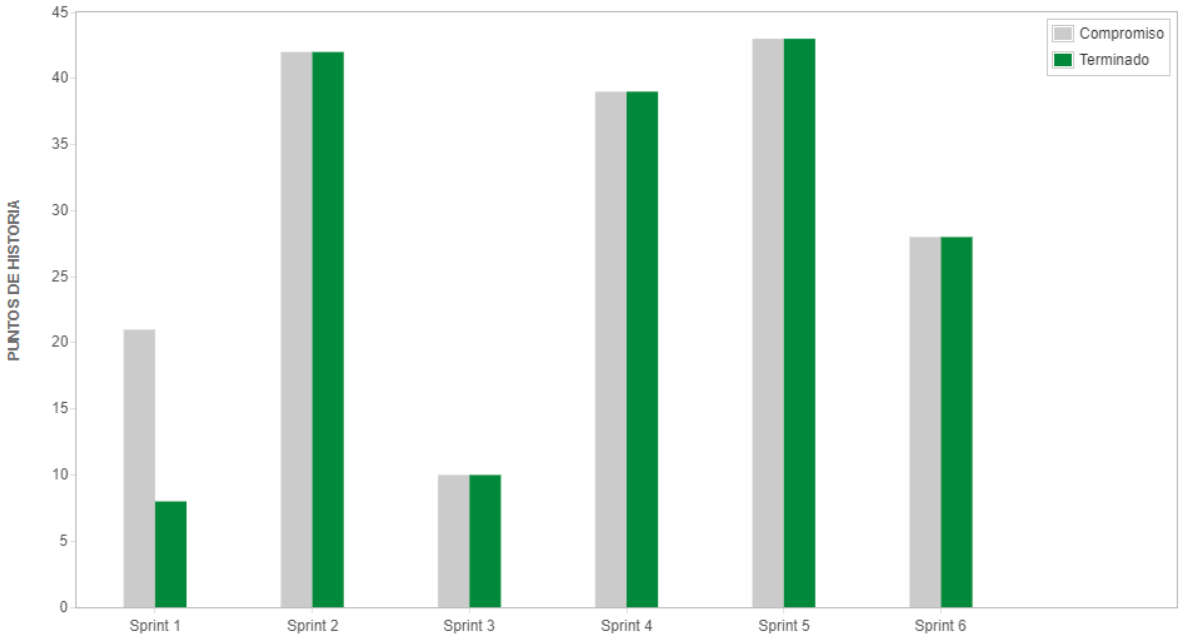


Figura 3.53: Seguimiento de 'compromiso y terminado' (Jira).

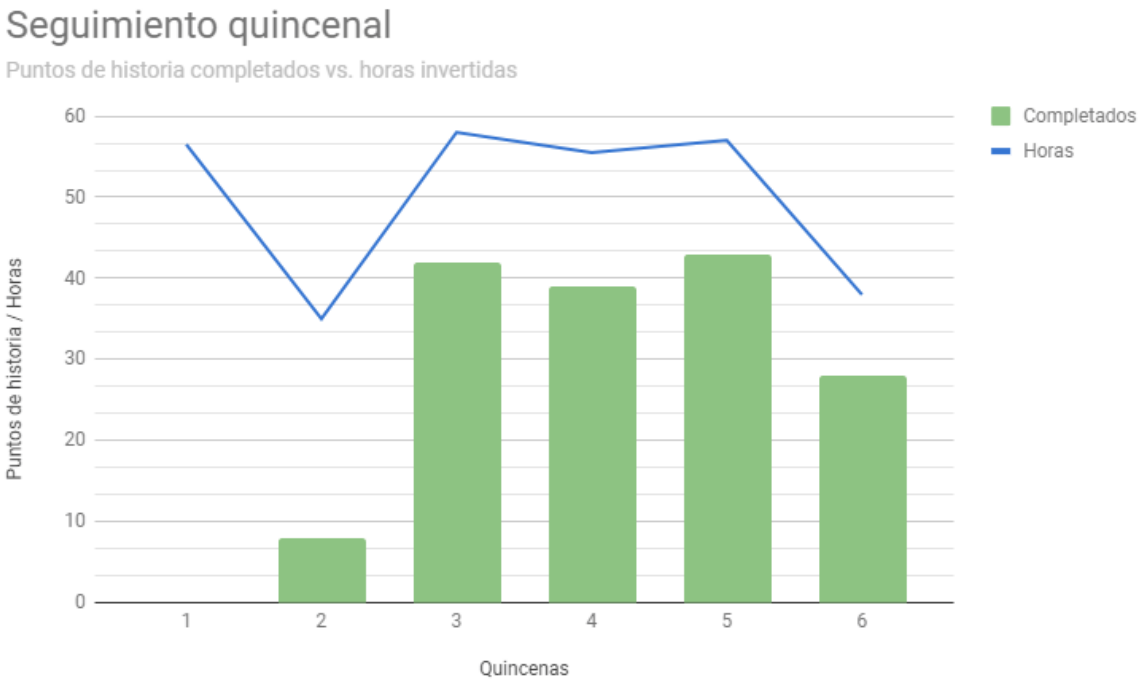


Figura 3.54: Seguimiento quincenal de puntos de historia completados vs. horas invertidas.

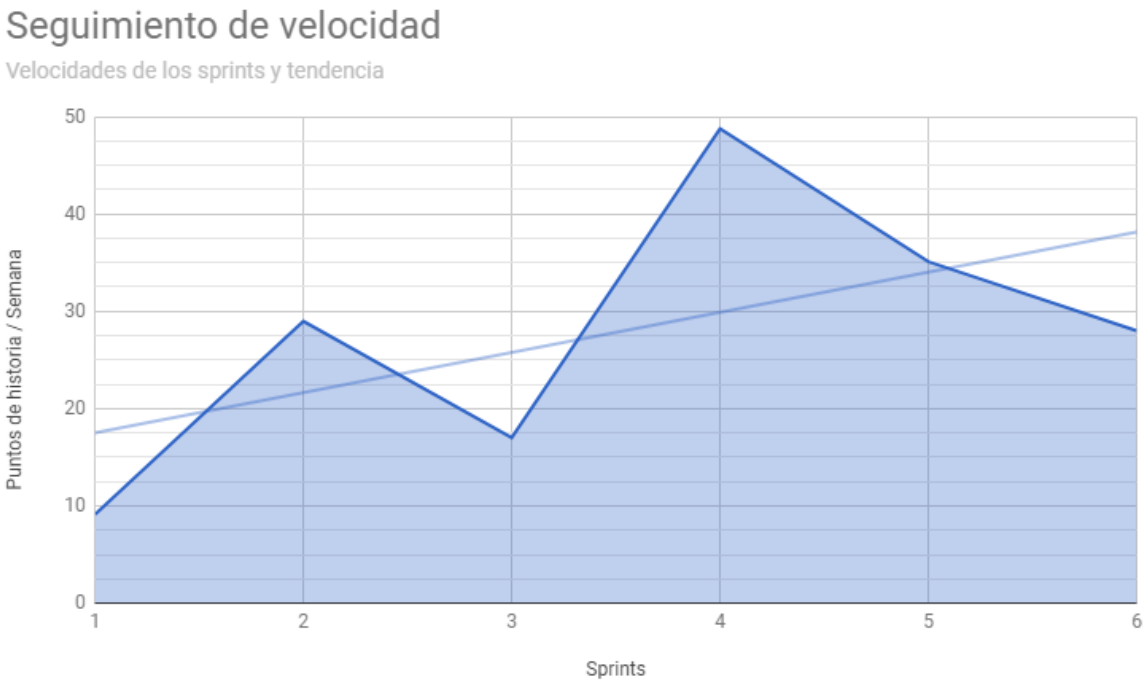


Figura 3.55: Seguimiento por sprint de las velocidades.

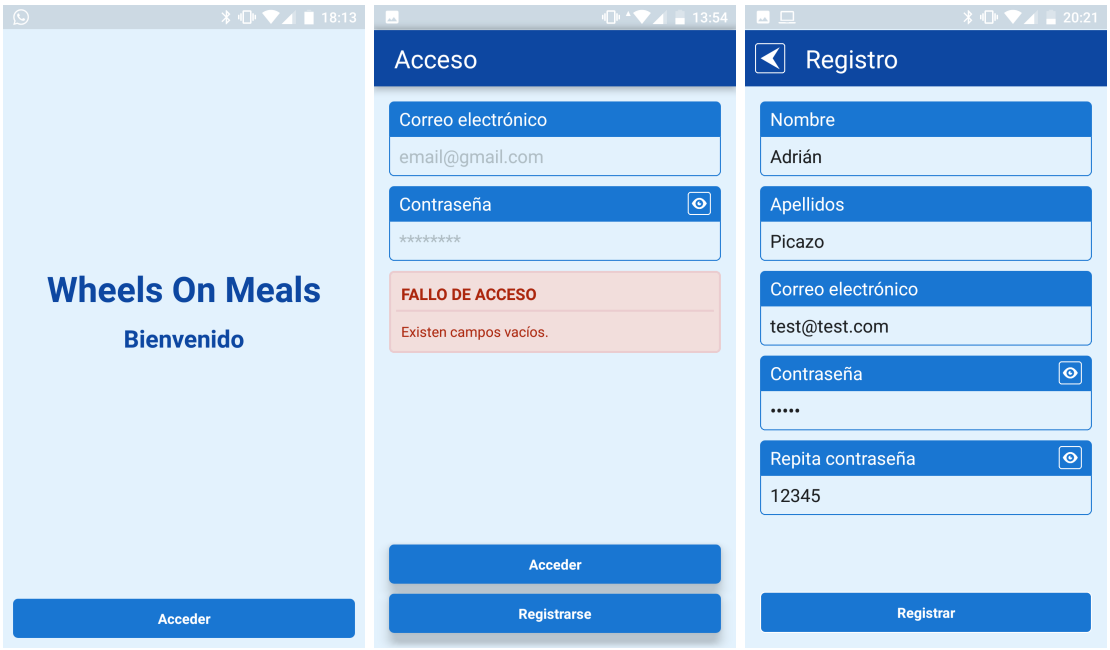
las velocidades. Las velocidades aquí mostradas se han obtenido a partir de dividir los puntos de historia completados entre las horas invertidas y multiplicar por 40 horas semanales. Es decir, son las velocidades supuestas en una semana de trabajo a jornada completa.

Lo más importante de la gráfica es la tendencia. La documentación oficial indican que a partir del cuarto sprint la velocidad se acerca al rendimiento real. Suponemos que en la documentación oficial los sprints son de cuatro semanas. Entonces, a partir de la semana 16 podemos medir nuestro rendimiento real.

En nuestro caso, sólo llevamos 10 semanas, pero se puede intuir por la tendencia que alcanzaríamos una velocidad de 30 puntos/semana. De hecho, en la gráfica de la Figura 3.55 podemos observar que la última quincena, con cerca de 40 horas casi alcanzamos los 30 puntos.

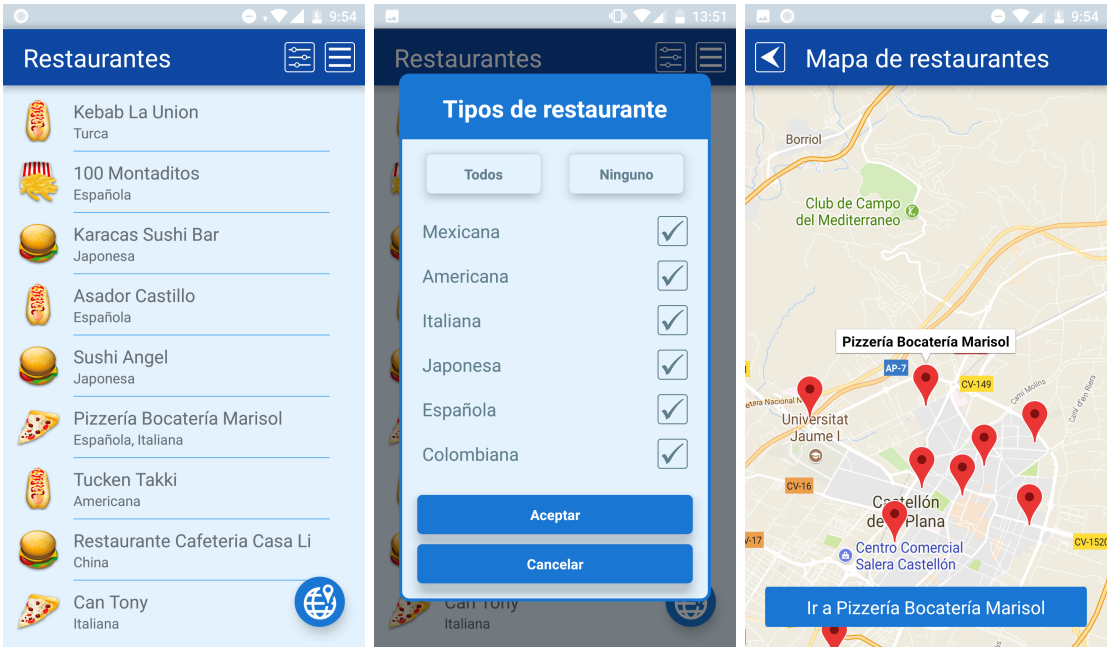
3.6.9. Resultado final

Como resultado final, hemos conseguido una versión de la aplicación con todos los requisitos iniciales del cliente además de otros funcionalidades posteriormente planteadas. Tenemos una versión nativa totalmente funcional para *Android*; sin embargo, no la tenemos para *iOS*. Tampoco hemos conectado la aplicación con la parte de Roberto, aunque la hemos suplido implementando nuestros servicios con ayuda de *Firebase*. Todo ello en un plazo menor a 300 horas, lo que dura la estancia en prácticas, y empleando una metodología ágil, concretamente *Scrum*. Por último, el acabado final es muy consistente y con un alto grado de usabilidad, como se muestra en las Figuras 3.56, 3.57, 3.58 y 3.59



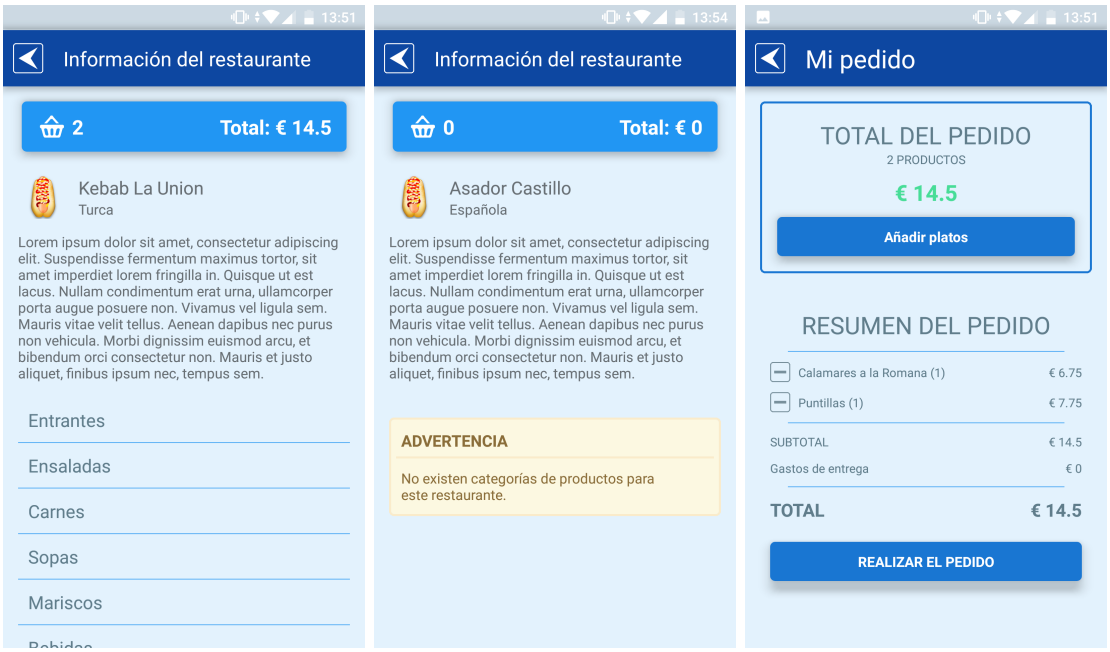
(a) Presentación. (b) Acceso con fallo. (c) Registro.

Figura 3.56: Capturas de la versión final (1/4).



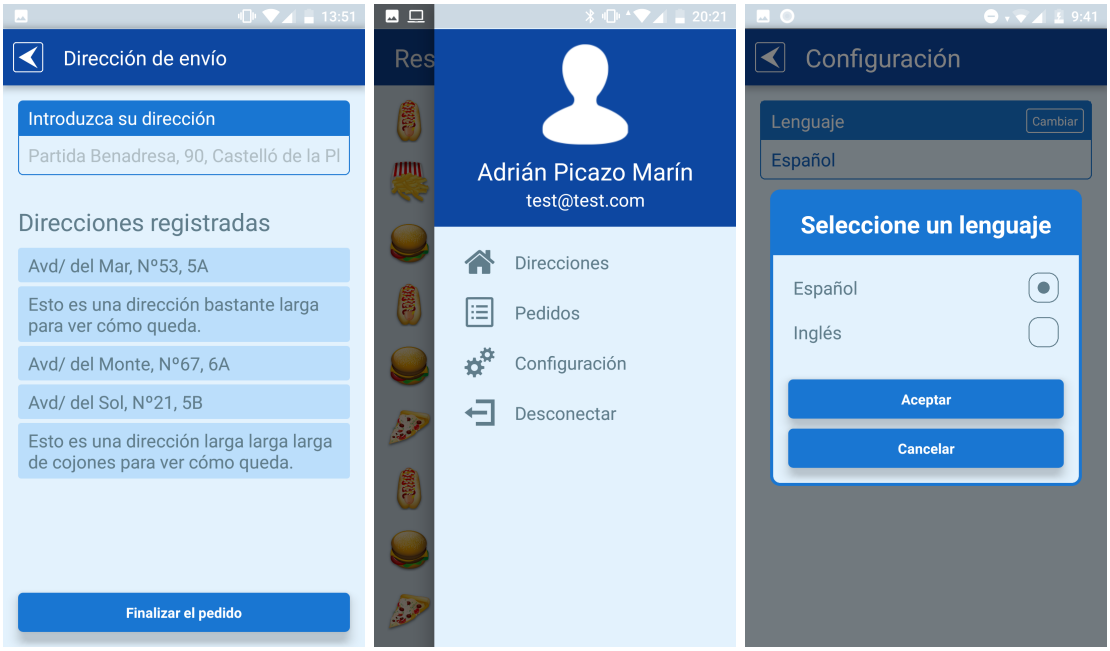
(a) Listado restaurantes. (b) Filtrado por tipos. (c) Mapa restaurantes.

Figura 3.57: Capturas de la versión final (2/4).



(a) Restaurante. (b) Restaurante sin productos. (c) Pedido.

Figura 3.58: Capturas de la versión final (3/4).



(a) Dirección del pedido. (b) Menú de usuario. (c) Configuración idioma.

Figura 3.59: Capturas de la versión final (4/4).

Capítulo 4

Verificación y validación

La verificación[1] consiste en llevar un control sobre los procesos para asegurarnos en todo momento que la pila del producto y los sprints están bien definidos. Durante el desarrollo la verificación la hemos aplicado al crear la pila del producto y en cada revisión de la pila del producto (después de cada sprint).

Para la pila del producto, cada historia de usuario debía estar registrada en *Jira*, detallada en su descripción con el objetivo, la definición de hecho (*Definition of Done*) y los criterios de aceptación (*Acceptance Criteria*), y priorizada y estimada en puntos de historia.

La definición de hecho es un conjunto de labores comunes a todas las historias que deben estar completadas para considerar que una historia lo esté a su vez. Labores que resumimos en tres: implementada, revisada (*Android* e *iOS*) y documentada. Para que una historia estuviera implementada y revisada debían cumplirse los criterios de aceptación, los cuales son diferentes para cada historia.

Para cada sprint, las historias escogidas debían estar divididas en subtareas. Cada subtarea debía estar definida con un nombre explicativo, etiquetada con uno de los tipos de subtarea (prototipos, firebase, servicios, componentes) y estimada en tiempo.

Si alguna de estas condiciones no se cumplía la verificación fallaba. La Figura 4.1 muestra parte de la información registrada en *Jira* necesaria para dar por válida una historia de usuario. La historia tiene una descripción con el objetivo, la definición de hecho y los criterios de aceptación. También aparecen las subtareas en las que está dividida, las cuales están categorizadas y estimadas en tiempo.

La validación[2] consiste en hacer cumplir para cada historia sus criterios de aceptación en los distintos sistemas.

En Cuatroochenta el departamento de testeo se encarga de realizar las pruebas a las aplicaciones con dispositivos reales y de manera manual. No emplean ninguna herramienta que

[1] ¿Estamos desarrollando el sistema correctamente?
[2] ¿Estamos desarrollando el sistema correcto?

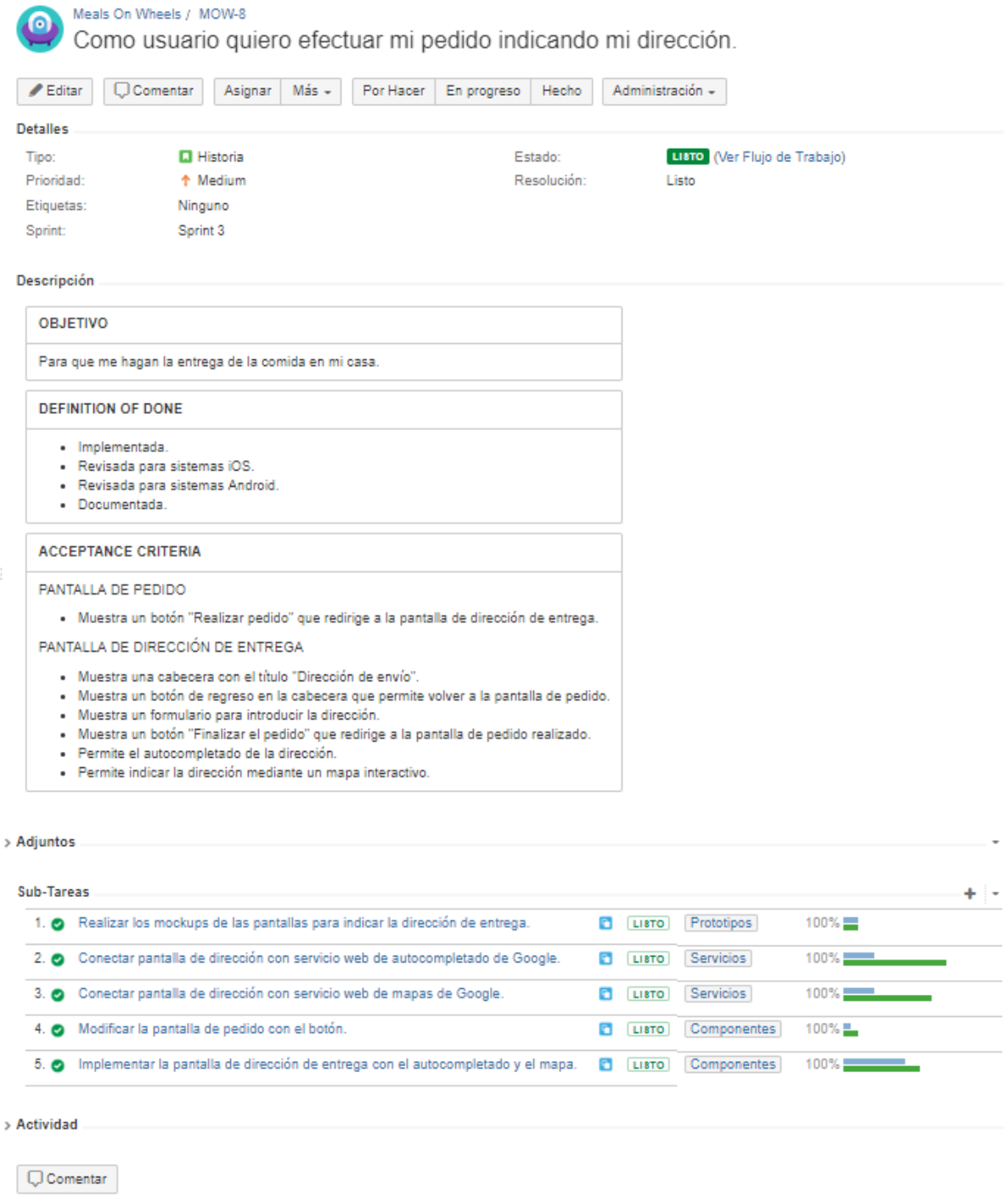


Figura 4.1: Pantalla de información de una historia de usuario ya completada con la descripción y las subtareas (Jira).

les automatice dichas labores. Si existe cualquier fallo que no cumpla con las expectativas del cliente lo indican a los desarrolladores.

De igual modo, durante el desarrollo hemos realizado las pruebas de manera manual. En un principio hemos empleado emuladores para las pruebas en sistemas *Android* e *iOS*. Más adelante, hemos incluido un dispositivo real con *Android*. Finalmente, y debido a problemas surgidos con el desarrollo *iOS*, hemos dejado de lado las pruebas en esta plataforma y nos hemos centrado sólo en *Android* con el dispositivo real.

Para tener un mejor control, todos los criterios de aceptación han estado organizados en una tabla (Figura 4.2). La tabla tiene dos columnas con casillas de colores: la columna de la izquierda representa al sistema *Android* y la columna de la derecha al sistema *iOS*. Un color verde significa que dicho criterio ha sido probado y cumple en ese sistema; uno rojo, que ha sido probado y no cumple; y uno amarillo, que no ha sido probado.

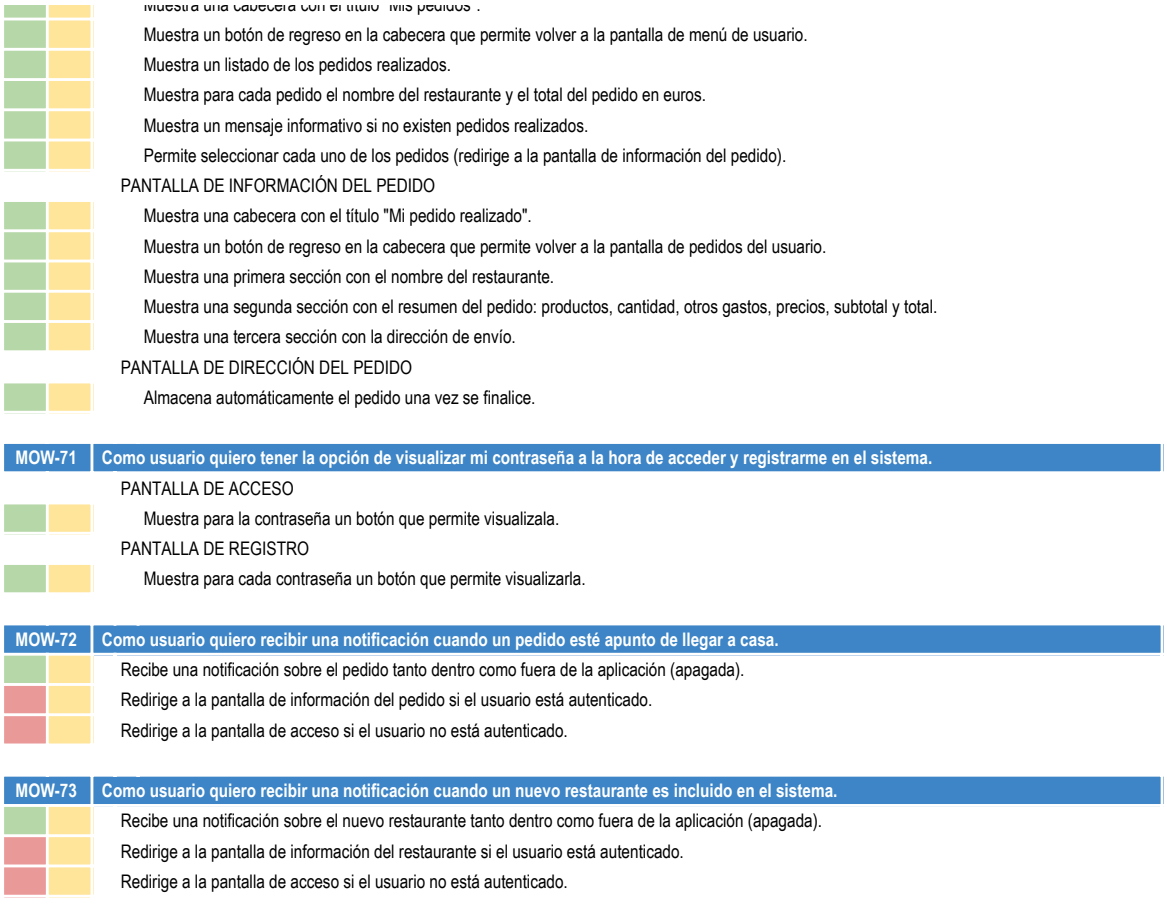


Figura 4.2: Parte de la tabla de control de los criterios de aceptación. En el momento de captura el sistema *iOS* ya no se tenía en cuenta, sólo *Android*.

Capítulo 5

Conclusiones

En el ámbito formativo he consigo completar mis conocimientos de desarrollador *full-stack*[1]. Durante el grado en Ingeniería Informática todo lo que nos han enseñado estaba más enfocado a la parte *back-end*. Sólo en unas pocas asignaturas hemos tratado con la parte del *front-end*, y en esos casos ha consistido en aplicaciones web o aplicaciones de escritorio, nunca en aplicaciones móviles.

Es por ello que este proyecto me ha ayudado ampliamente a saber diseñar y desarrollar una aplicación *front-end*. En particular, una aplicación móvil. Me ha permitido aprender *JavaScript*, un idioma con el que no estaba familiarizado y que es ampliamente utilizado a nivel de diseño de interfaces; *React-Native*, una tecnología novedosa para desarrollar aplicaciones en sistemas *Android* e *iOS* simultáneamente; *Flux*, un nuevo patrón de diseño que está empezando a sustituir al patrón MVC por su sencillez y flexibilidad. Además, me ha posibilitado gestionar mediante una metodología ágil todo un proyecto y reforzar mis conocimientos sobre la herramienta *Jira*.

En el ámbito profesional ha sido mi primer contacto con una empresa de corte tecnológico. He conocido cómo funciona un mediana empresa de este tipo, los departamentos que la forman, el proceso de desarrollo, la gestión a nivel de empresa y a nivel de proyecto, así como la cultura de la empresa.

Al tratarse de una empresa tecnológica me ha beneficiado enormemente en el aspecto formativo. Mis compañeros de departamento tenían un gran conocimiento de las tecnologías, herramientas y demás utilidades que he empleado en el desarrollo del proyecto. Seguramente no hubiese aprendido tanto en otro tipo de empresa. Tampoco por mi cuenta. Esto es algo que valoro muy positivamente sobre la experiencia en Cuatroochenta.

En el ámbito personal he crecido como persona. Me siento más válido, capaz de cumplir proyectos con unos buenos resultados y a un buen rendimiento. Señalar que Sergio Aguado, al presentarle el último incremento, dijo que había obtenido un proyecto con unos acabados muy buenos, que perfectamente se podría sacar al mercado como una primera versión. Además, recalcó que mi rendimiento durante el desarrollo había sido también muy bueno.

[1] Que sabe desarrollar tanto en *back-end* como *front-end*.

Bibliografía

[1] Aikoven. Documentación oficial de redux-thunk. https://github.com/reduxjs/redux-thunk. [Consulta: 15 de mayo de 2018].

[2] José Manuel Alarcón. Webpack: qué es, para qué sirve y sus ventajas e inconvenientes. https://www.campusmvp.es/recursos/post/webpack-que-es-para-que-sirve-y-sus-ventajas-e-inconvenientes.aspx. [Consulta: 16 de mayo de 2018].

[3] Atlassian. Documentación oficial de confluence. https://es.atlassian.com/software/confluence. [Consulta: 20 de mayo de 2018].

[4] Atlassian. Documentación oficial de jira. https://es.atlassian.com/software/jira. [Consulta: 20 de mayo de 2018].

[5] Carlos Azaustre. ¿qué es flux? entendiendo su arquitectura. https://carlosazaustre.es/como-funciona-flux. [Consulta: 15 de mayo de 2018].

[6] Comunidad ESLint. Documentación oficial de eslint. https://eslint.org/. [Consulta: 20 de mayo de 2018].

[7] Facebook. Documentación oficial de flux. https://facebook.github.io/flux/docs/in-depth-overview.html#content. [Consulta: 15 de mayo de 2018].

[8] Jose Mª Baquero García. ¿por qué tiene tanto éxito la librería react? https://www.arsys.es/blog/programacion/react-javascript/. [Consulta: 19 de mayo de 2018].

[9] Google. Documentación oficial de google analytics. https://www.google.es/intl/es/analytics. [Consulta: 3 de junio de 2018].

[10] Google. Documentación oficial de realtime database de firebase. https://firebase.google.com/docs/database/?hl=es-419. [Consulta: 3 de junio de 2018].

[11] Pete Hunt. Why did we build react? https://reactjs.org/blog/2013/06/05/why-react.html. [Consulta: 15 de mayo de 2018].

[12] IBM. Overview: Ibm worklight projects, applications, environments, and skins. https://www.ibm.com/support/knowledgecenter/en/SSZH4A_5.0.5/com.ibm.worklight.help.doc/devref/c_overview_projects_apps_envs_skins.html. [Consulta: 20 de mayo de 2018].

[13] Web2App Infotech. Mobile app comparison – native vs. hybrid vs. web. http://www.web2appinfotech.com/mobile-app-comparison-native-vs-hybrid-vs-web. [Consulta: 20 de mayo de 2018].

[14] Neon Rain Interactive. Agile scrum for web development. https://www.neonrain.com/agile-scrum-web-development. [Consulta: 6 de junio de 2018].

[15] Jaime Olmo. ¿qué es redux? http://www.jaimeolmo.com/2017/02/que-es-redux/. [Consulta: 20 de mayo de 2018].

[16] OneSignal. Documentación oficial de one signal. https://onesignal.com. [Consulta: 3 de junio de 2018].

[17] Postman. Documentación oficial de postman. https://www.getpostman.com. [Consulta: 3 de junio de 2018].

[18] Comunidad Redux. Documentación oficial de redux. https://es.redux.js.org/docs. [Consulta: 15 de mayo de 2018].

[19] Samuel Simões. Facebook flux and mvc. https://blog.samuelsimoes.com/javascript/2016/02/27/facebook-flux-and-mvc.html. [Consulta: 20 de mayo de 2018].

[20] Skellock. Documentación oficial lde reactotron. https://infinite.red/reactotron. [Consulta: 7 de junio de 2018].

[21] Facebook Open Source. Documentación oficial de react. https://reactjs.org. [Consulta: 15 de mayo de 2018].

[22] Facebook Open Source. Documentación oficial de react-native. https://facebook.github.io/react-native. [Consulta: 15 de mayo de 2018].

[23] Comunidad Webpack. Documentación oficial de webpack. https://webpack.js.org/concepts. [Consulta: 16 de mayo de 2018].

[24] Wikipedia. Diseño web adaptable. https://es.wikipedia.org/wiki/Dise%C3%B1o_web_adaptable. [Consulta: 20 de mayo de 2018].

[25] Wikipedia. React. https://es.wikipedia.org/wiki/React. [Consulta: 15 de mayo de 2018].

[26] Ken Schwaber y Jeff Sutherland. La guía de scrum. http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf. [Consulta: 6 de junio de 2018].

[27] Comunidad Yarn. Documentación oficial de yarn. https://yarnpkg.com/lang/en/. [Consulta: 21 de mayo de 2018].

Apéndice A

Metodología Scrum

Scrum se encuentra dentro del marco de las metodologías ágiles. Las metodologías ágiles se caracterizan por seguir los siguientes principios: (1) individuos e interacciones por delante de procesos y herramientas, (2) software terminado y funcional por delante de extensa documentación, (3) colaboración con el cliente por delante de contratos y (3) respuesta al cambio por delante de seguir un plan. Scrum sigue todas ellas.

Scrum adopta un desarrollo incremental frente a una planificación y ejecución completa del producto. Se caracteriza por confiar más en el conocimiento de las personas que en la calidad de los procesos empleados. [1] También se caracteriza por el solapamiento de las diferentes fases del desarrollo (análisis, diseño, codificación, prueba), en lugar de la realización de una tras otra como en una metodología en cascada.

Scrum esta compuesto por roles, eventos y artefactos:

- Los **roles** son cada una de las figuras que participan en la metodología. Dentro de los roles nos encontramos: a las **partes interesadas** (*Stakeholders*), al **propietario del producto** (*Product Owner*), al ***Scrum Master*** y al **equipo de desarrollo** (*Development Team*). Estos tres últimos forman a su vez lo que se conoce como el **equipo *Scrum*** (*Scrum Team*), el cual se caracteriza por ser autoorganizado y multidisciplinar.
- Los **eventos** son básicamente las reuniones que se dan durante el proceso. Dentro de los eventos nos encontramos: la **planificación del sprint** (*Sprint Planning Meeting*), la **reunión diaria** (*Daily Scrum Meeting*), la **revisión del sprint** (*Sprint Review*) y la **retrospectiva del sprint** (*Sprint Retrospective*).
- Los **artefactos** son los elementos que garantizan el registro de la información clave del proceso y que sientan las bases de la calidad y la productividad. Dentro de los artefactos nos encontramos: la **pila del producto** (*Product Backlog*), la **pila del sprint** (*Sprint Backlog*) y el **incremento** del producto.

[1] Según *La Guía de Scrum* de Ken Schwaber y Jeff Sutherland: «Scrum se basa en la teoría de control de procesos empírica o empirismo. El empirismo asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo.» [26]

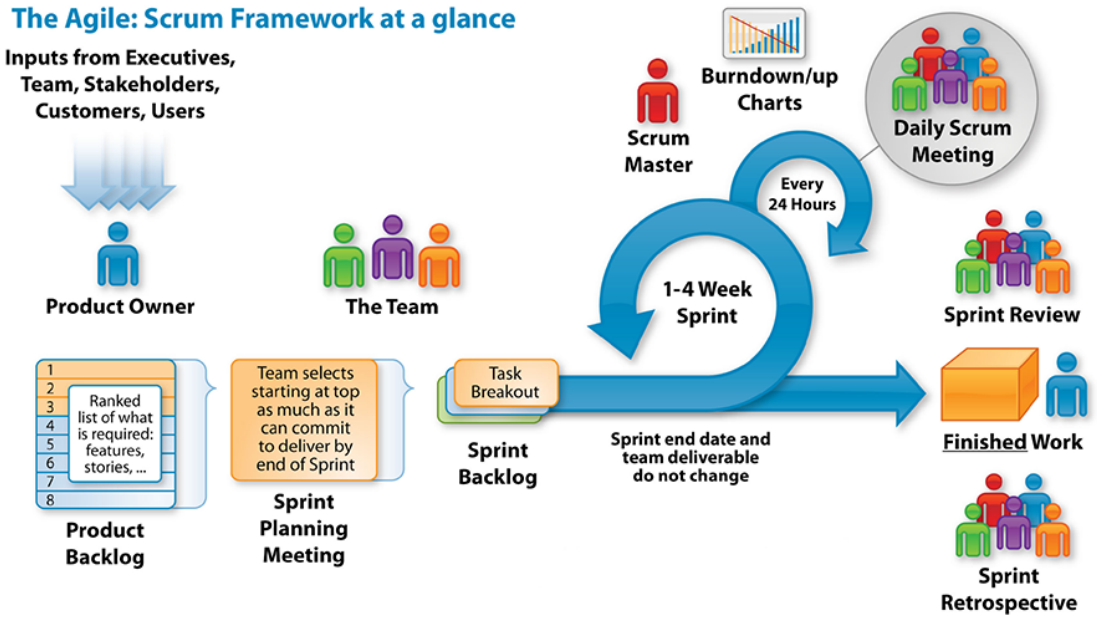


Figura A.1: Diagrama de la metodología ágil *Scrum* donde se pueden observar los distintos roles, artefactos, eventos y sus relaciones. El propietario del producto optimiza el valor del producto y gestiona la pila del producto. El equipo de desarrollo crea los incrementos del producto. El *Scrum Master* gestiona el proceso *Scrum* y elimina los impedimentos que puedan surgir.
Fuente de la imagen: [14]

En la Figura A.1 se muestra un diagrama de cómo es *Scrum* con sus roles, artefactos y eventos.

El propietario del producto recoge toda la información proporcionada por las partes interesadas (usuarios, clientes, etc). Esta información (requisitos) se transforma en historias de usuario (*User Stories*), que el propietario del producto ordena según prioridad y estima según puntos de historia (*story points*)[2]. En conjunto, todas las historias de usuario priorizadas y estimadas forman la pila del producto.

Al principio de cada sprint el equipo *Scrum* se reúne, decide qué historias de usuario realizarán y las descomponen en tareas. Las tareas, a diferencia de las historias, se estiman en tiempo: horas, días, semanas. Este conjunto de historias de usuario seleccionadas y descompuestas en tareas forman la pila del sprint. Es importante que los sprints tengan velocidades similares, es decir, que el número total de puntos de historia sea parecido entre cada sprint.

Un sprint suele durar unas cuatro semanas. Durante un sprint, cada día se reúne el equipo de desarrollo para hablar sobre qué se ha hecho, qué se va a hacer y qué problemas han surgido. Es lo que se conoce como reunión diaria del *Scrum*. Uno del equipo toma el rol de *Scrum Master*, que vela por que se cumplan las reglas del proceso. En este caso de la reunión diaria, el *Scrum Master* vigila que se respete el turno de palabra y que todo el mundo participe.

[2]Los puntos de historia (*story points*) es una forma de estimar historias de usuario sin entrar en medidas de tiempo (horas, días, semanas). Puesto que una de las ideas de las metodologías ágiles es no planificar tan estrictamente, los puntos de historia viene a ser un sustituto de mediar en el tiempo. Consiste en asignarle un valor mínimo a la historia más simple y asignar valores mayores al resto de historias en proporción.

Al final de cada sprint se obtiene un incremento del producto. El incremento debe estar terminado y ser totalmente funcional. Además, el equipo realiza dos reuniones: una para hablar sobre el incremento (revisión del sprint) y otra para hablar sobre el sprint (retrospectiva del sprint). En ambas se intercambian impresiones, problemas que se han tenido, etc.

Por último, para llevar un control del tiempo y tener una previsión real de la entrega final, el *Scrum Master* se encarga de actualizar varios gráficos. En ellos puede indicar los puntos de historia realmente completados en cada sprint para hacerse a la idea de si el proyecto se desvía demasiado de su fecha tope. En ese caso tomaría medidas correctivas. A este tipo de gráficos se les conocen como *burndown charts*.

Apéndice B

Base de datos en *Firebase*

En este apéndice mostramos la base de datos NoSQL de *Realtime Database* de *Firebase* en la cual empleamos una estructura JSON. Las Figuras B.1 y B.2 muestran la evolución final de la base de datos hasta la fecha.

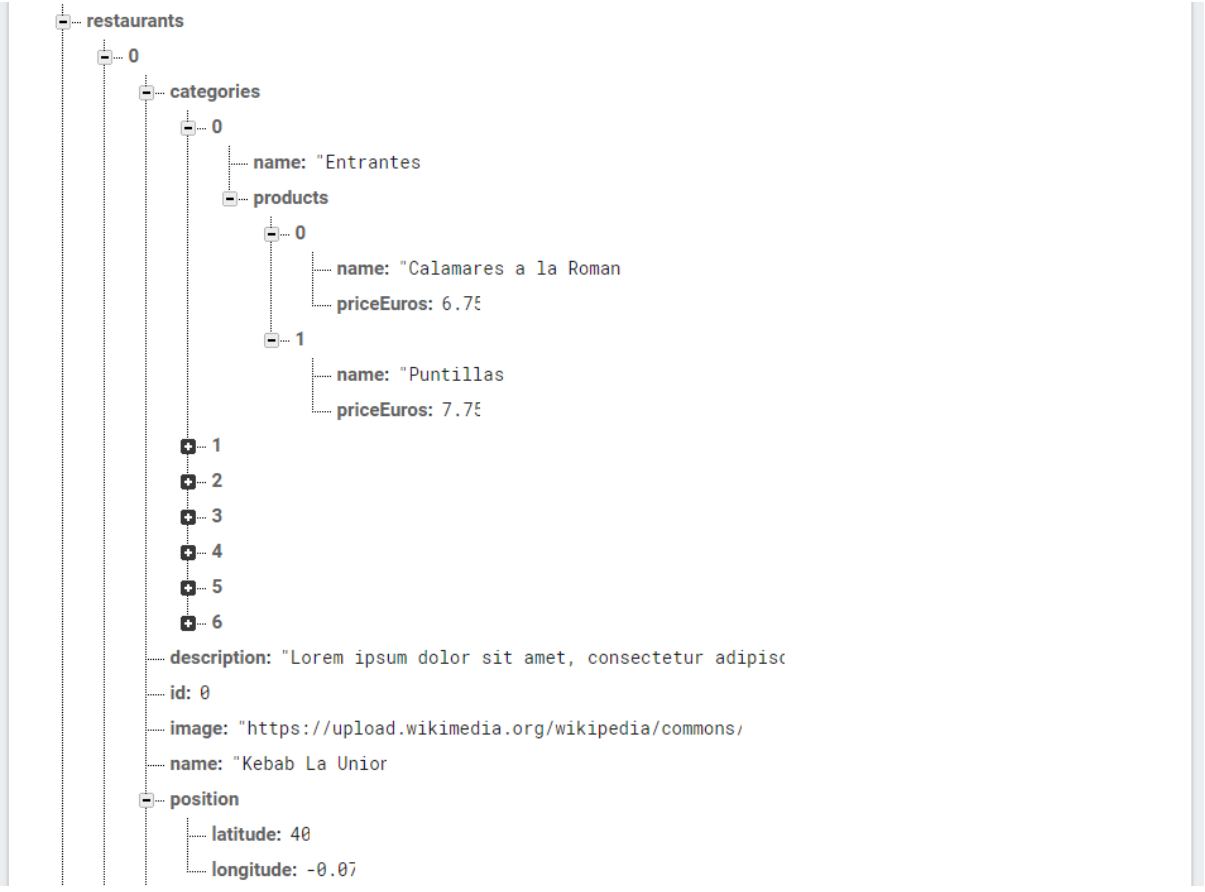


Figura B.1: Base de datos en *Realtime Database* de *Firebase* (1/2).

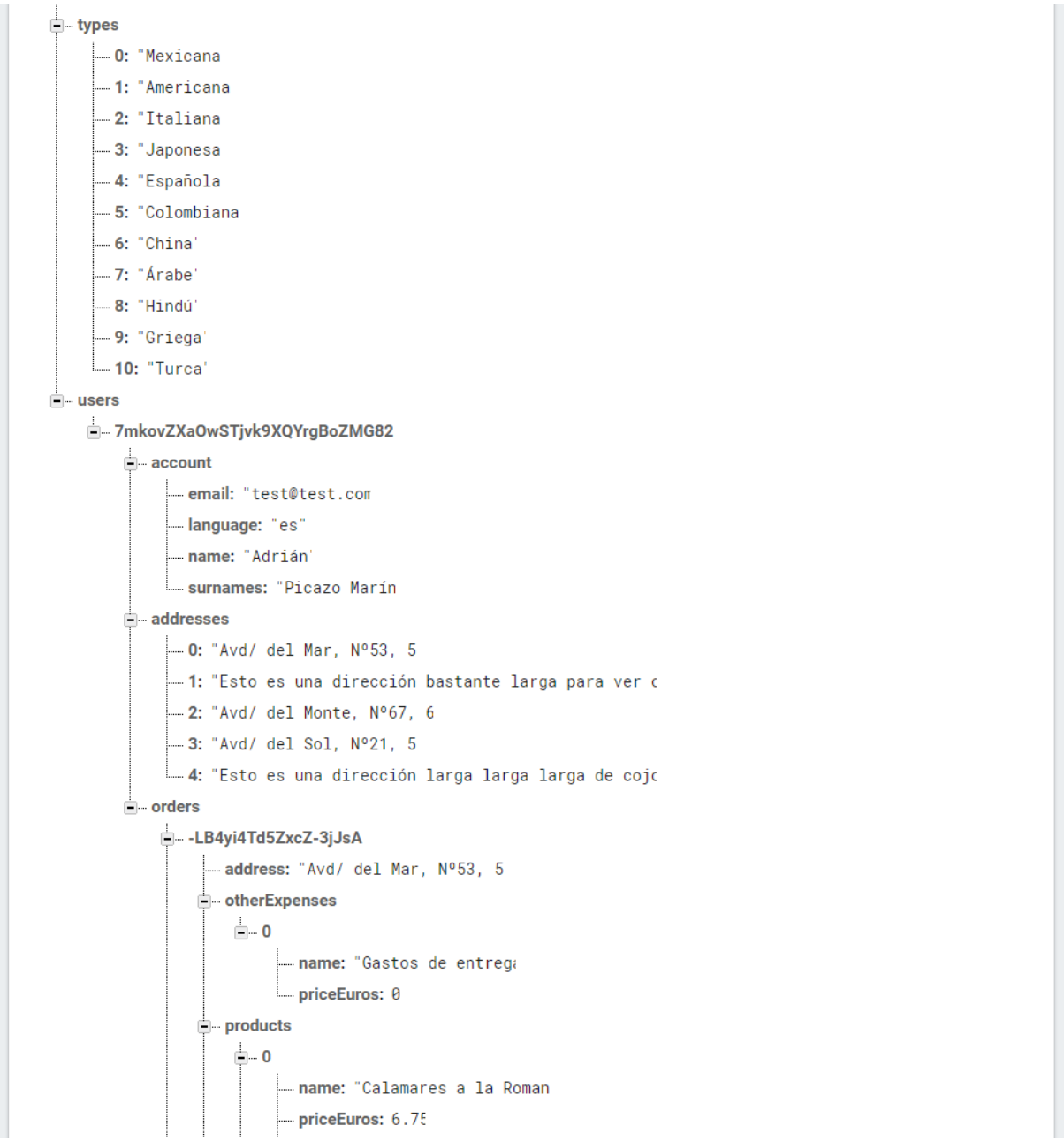


Figura B.2: Base de datos en *Realtime Database* de *Firebase* (1/2)