Gestión de Compras y Comunicación Interna para Pequeños Comercios

Nadia Arvez Villalba

Resumen—Los pequeños comercios siempre han necesitado aplicar técnicas de organización para el aprovisionamiento, y reparto de mercadería. Dado que en la actualidad el mercado de aplicaciones presenta un vacío en este sector, existen comercios que intentan llevar dicha organización por medio de aplicaciones de mensajería. En este proyecto se pretende cubrir este vacío utilizando tecnologías orientadas a la creación de aplicaciones multiplataforma e híbridas. Como resultado se ha obtenido una herramienta que ofrece un servicio de gestión de compra, pedidos y reparto de género, con lo que se busca que las pequeñas empresas también puedan adaptarse a las nuevas tendencias, incorporando una aplicación ligera e intuitiva.

Palabras claves—Angular.js, Cloud9, Gestión de comercio, Ionic Framework, IonicMaterial, JavaScript, MongoDB, Node.js, Sails.js, Waterline

Abstract—Small businesses have always needed to apply organizational techniques for provision and distribution of goods. Given that nowadays the applications market presents a gap in this sector, businesses try to lead this type of organizations through messaging applications. This project aims to fill this gap using technologies geared towards the creation of multiplatform and hybrid applications. As a result, it has been obtained a tool that offers a management service for purchase, orders and product delivery. This allows small businesses to adapt to new trends with a light and easy application.

Index Terms— Angular.js, Cloud9, Ionic Framework, IonicMaterial, JavaScript, MongoDB, Node.js, Sails.js, Shop management, Waterline

1 Introducción

DESDE siempre los pequeños comercios han tenido la necesidad de gestionar las tareas que se llevan a cabo, a nivel interno, para poder conseguir sus objetivos financieros y llevar una organización. Cuando se habla de gestión interna se hace referencia a la gestión de pedidos, compra y reparto de género, comunicación entre empleados, etc.

Partiendo de la base de que los pequeños comercios se componen de una o más tiendas y que necesitan comprar grandes cantidades de género a fabricantes. Se ha desarrollado esta herramienta para intentar controlar la compra y distribución de género.

Cada tienda necesita adquirir diversas mercaderías en diferentes cantidades y calidades. Muchas veces los encargados de las tiendas, por falta de comunicación con los compradores, se pierden información relevante respecto a ofertas puntuales o bien una descripción detallada de los productos.
Por otra parte el comprador y el transportista, personas encargadas de realizar compra, recogida y entrega del género, debido a su laboriosa tarea carecen de tiempo para ofrecer una descripción detallada de algunos productos.

Como consecuencia de esta falta de comunicación, para un pequeño comercio, muchas veces significa una falta de innovación lo que conlleva a descenso de venta.

Actualmente existen algunos comercios que intentan mantener la organización y la comunicación por medio de aplicaciones de mensajería, ya que no existen aplicaciones gratuitas y mucho menos para smartphones que les permita.

Está claro que existen softwars desarrollados para cubrir esta necesidad pero con un coste que muchos pequeños comercios no se lo pueden permitir.

Este proyecto nace de la necesidad de cubrir dicho vacío en el mercado de las aplicaciones para gestionar pequeños comercios tales como fruterías, panaderías etc., que necesitan de un sistema de comunicación interna.
No se trata de un simple gestor de stock como las que se encuentran actualmente en el mercado, va un poco más allá. Permite comunicar en tiempo real a todas las partes implicadas, lo que se reduce a una mejor comunicación.

Para este proyecto se ha tenido en cuenta los diversos dispositivos en los que se podría hacer uso de esta herramienta, por esta razón es un requisito primordial que sea multiplataforma. Para cumplir con este requisito se ha optado por implementarlo en JavaScript. A pesar de que existan diversas tecnologías para desarrollar el proyecto, uno de los motivos por lo que se ha optado por JavaScript, es porque permite desarrollar desde la interfaz de usuario hasta el servidor, pasando por la base de datos, con un mismo lenguaje.

En este documento se pretende detallar todas las fases por las que ha pasado el proyecto hasta conseguir los objetivos propuestos.

En primer lugar se presenta la descripción del proyecto, en el que se detallan los requisitos, objetivos y actores que interactuaran en la aplicación. Así también se expondrá la planificación planteada y los cambios que esta ha ido sufriendo durante la evolución del proyecto.
Por otra parte se muestran las diferentes tecnologías con las que se ha trabajado para conseguir los objetivos y funcionalidades.

Y por último se analizaran los objetivos conseguidos, problemas surgidos durante el desarrollo y cómo se han enfrentado.

1.1 Objetivos

A continuación, se muestra en la Tabla 1 los principales objetivos a cubrir en este proyecto con sus respectivas prioridades:

Tabla 1
Objetivos de Fill Your Store

OBJETIVOS	PRIORIDAD
Comunicación entre usuarios en tiempo real	Prioritario
Desarrollar una aplicación responsiva	Prioritario
Facilitar la gestión de compra de género	Prioritario
Facilitar la gestión de reparto de mercadería	Prioritario
Multiplataforma	Prioritario
Interfaz de usuario intuitiva	Prioritario
Alta escalabilidad	Prioritario
Testear ventajas e inconvenientes entre diferentes frameworks	Secundario
Implementar un sistema seguro	Secundario

1.2 Requisitos Funcionales

En este apartado se definen los requisitos:

R1. El sistema debe poder hacer login.
R2. El sistema debe permitir dar de alta usuarios.
R3. El sistema debe permitir agregar/eliminar productos de la base de datos.
R4. El sistema debe mostrar catálogo.
R5. El sistema debe permitir generar y enviar pedidos.
R6. El sistema debe permitir activar y desactivar productos del catálogo.
R7. El sistema debe ver lista de pedidos.
R8. El sistema debe permitir visualizar punto de recogida y de entrega de mercadería.
R9. El sistema debe permitir agregar productos al carrito.
R10. El sistema debe mostrar la lista de pedido antes de enviar.

Rol *Merchant*:
- Visualiza catálogo.
- Visualiza lista de pedidos antes de enviar.
- Agrega productos al carrito.
- Envía pedidos.

Rol *Shopper*:
- Agrega/elimina productos de la base de datos.
- Activar y desactivar productos del catálogo.

Rol *Admin*:
- Da de alta a nuevos usuarios
- Agrega/elimina productos de la base de datos.
- Agrega/elimina productos de la base de datos.

Rol *Transporter*:
- Visualiza detalle de pedidos
- Visualiza punto de recogida y de entrega de mercadería

2 ESTADO DEL ARTE

En la actualidad encontramos softwars que permiten gestionar pequeños y grandes comercios pero la gran mayoría son de pago.
En el mercado de aplicaciones encontraremos sistemas que hacen de terminal de punto de venta (TPV) [1], como por ejemplo *TPV Simple Bar & Tiendas Free*, que ayudan a controlar las ventas, cierre de caja y control de inventario, etc. [2].
En cambio la que se presenta en este proyecto ofrece una gestión a otro nivel y comunicación en tiempo real.

También existen aplicaciones móviles que permite gestionar empresas (PYME), aunque más escasas. Como por ejemplo *Sistema de gestión (stock) ERP*, este sistema ofrece administración de usuarios, mensajería interna, visualización en mapa, pedidos, clientes, productos, informes (PDF) y artículos [3].

Tal y como se puede ver, ofrece las mismas funcionalidades, y más, que la aplicación desarrollada en este proyecto. Cabe destacar que a diferencia de la explicada anteriormente esta se ha desarrollado para ser desplegada en cualquier plataforma ya sea para Android, iOS y Windows Phone.
Para conseguir sistema multiplataforma se ha decidido implementar con JavaScript. Una de las principales ventajas de utilizar este lenguaje es que permite desarrollar en un solo lenguaje desde la interfaz de usuario hasta la parte del servidor.

Para el back-end se ha utilizado Node.js. Es un entorno de programación que interpreta JavaScript. Node.js está enfocado a la programación orientada a eventos. Esta herramienta cambia el paradigma que se aplicaba hasta hace poco en el lado del servidor. En lugar de generar hilos en cada conexión con el servidor, Node genera eventos dentro del proceso del motor de Node [4]. Cabe destacar que cuenta con una gran cantidad de librerías con funcionalidades ya desarrolladas y abundante documentación.

Por otra parte el front-end ha sido desarrollado con Ionic framework. Con Ionic se ha conseguido crear una aplicación multiplataforma usando los estándares de HTML5. Este framework une la potencia de AngularJS con la capacidad de generar aplicaciones HTML5 con Cordova. Además dispone de un diseño CSS atractivo y dirigido especialmente a móviles [5]. Para implementar la interfaz de usuario se ha utilizado una extensión de Ionic llamada IonicMaterial, provee una serie de componentes con los que crear la aplicación [6].
En general toda la API de la aplicación se ha desarrollado en Sails.js [19]. Este es un framework diseñado para emular el patrón de Modelo Vista Controlador (MVC) que facilitará el desarrollo de la aplicación en Node.js. Ofrece la posibilidad de trabajar con cualquier tipo de base de datos, esto es posible gracias a que viene instalado con un potente ORM/ODM llamado Waterline [7]. Waterline es la herramienta de almacenamiento de datos que simplifica la interacción con la base de datos. Por otra parte permite crear modelo relacional, se pueden hacer asignaciones múltiples y asociaciones por modelo [8].
Otra característica a destacar es que permite iniciar el servidor de la aplicación por medio de blueprints. Blueprints permite crear rutas API y acciones basadas en el diseño de la aplicación [9]

3 METODOLOGÍA

Para desarrollar el proyecto se ha decidido aplicar una metodología de desarrollo ágil llamada Rapid Application Development (RAD) [10]. Con ello se pretende reducir los riesgos partiendo en segmentos más pequeños y así proporcionando más facilidad de cambio durante el desarrollo.
Por otra parte se ha aplicado nivel de prioridad para llevar un control del proyecto y establecer plazos de entrega. En caso de que el proyecto se vea con la necesidad de más tiempo para desarrollarlo se optará, en último caso, por la reducción de requisitos.

La metodología cuenta con cuatro etapas [10]:

4.a Definición Conceptual. En esta etapa se han planificado los requisitos y se han establecido unos objetivos básicos y necesarios para la aplicación. Así también se han definido las funcionalidades del proyecto y se han analizado los problemas que podrían surgir durante el desarrollo.

Además se ha realizado un estudio de la viabilidad de los objetivos propuestos. Como se mencionaba anteriormente para evitar riesgos se ha preferido añadir funcionalidades a la aplicación, si el tiempo lo permite, en lugar de reducir funcionalidades.

4.b Diseño Funcional. En el diseño del proyecto se ha planteado cómo satisfacer los requisitos, aspectos de la aplicación y funcionamiento general. Así también se han diseñado de los componentes que permitieron llevar a cabo la funcionalidad de la aplicación.

Para poder desarrollar la aplicación en JavaScript ha sido necesario consultar algunas fuentes de información sobre todo documentación propia de Sails.js y Node.js.
La implementación de la API de esta aplicación se ha desarrollado en Sails.js. En cuanto a formación ha sido necesario consultar la documentación y tutoriales para llevar a cabo la implementación de algunas funcionalidades empleando módulos de sails.

Para la base de base de datos no ha sido necesaria ninguna formación respecto al funcionamiento, más bien se han consultado la documentación de sails para poder configurar la conexión.

Por último como entorno de desarrollo para el proyecto se ha optado por utilizar servidor virtual llamado Cloud9. Esta valiosa herramienta ha permitido almacenar el código y probar la aplicación. Se modifican los archivos "en caliente" y a su vez se van guardado en la nube [11]. Ofrece la posibilidad de trabajar desde cualquier punto con solo loguearse. Permite trabajar con la base de datos y los lenguajes escogidos y realizar pruebas en un entorno más real.

4.c Desarrollo. Lo más importante de esta etapa ha sido la construcción de la aplicación. Gracias a la planificación y una buena formación sobre las tecnologías aplicadas se ha desarrollado un proyecto dentro de las fechas planificadas. Ante la aparición de algún problema se ha podido resolver con máxima flexibilidad y sin afectar al proyecto en ningún aspecto. Esta etapa se explica con mayor detalle en el apartado de Análisis y desarrollo del proyecto.

4.d Despliegue. En este apartado se han hecho pruebas en base a casos de uso. Se han realizado dichas pruebas con la finalidad de comprobar el correcto funcionamiento tanto a nivel de back-end como de front-end.

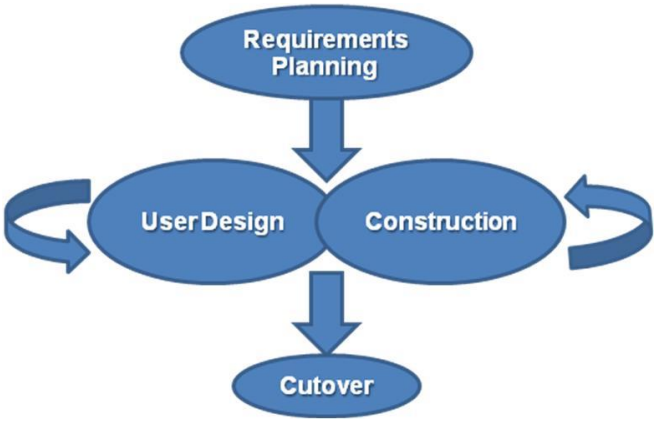

Figura 1. Etapas de RAD [10]

3.1 Planificación

Después de la captura de requisitos se dividió el proyecto en tareas y subtareas de manera que se pueda llevar un control y una organización. Inicialmente la planificación estaba prevista tal y como se muestra en la Figura 2.

Durante la fase de desarrollo de la gestión de usuario de la planificación se detectó un problema. En un principio se tenía planteado desarrollar esta sección con un módulo de sails pero debido a que la aplicación no la acababa de integrar, por una serie de fallos, se ha intentado encontrar solución pero no se halló ninguna. Por lo que se decidió reorganizar las fases del proyecto lo qué significo un cambio en la planificación. Los motivos de dicho cambio se expresan con más detalle en la conclusión.

Nombre	Fecha de inicio	Fecha de fin
Documentación del proyecto	15/02/16	22/06/16
Formación JavaScript	15/02/16	18/03/16
JavaScript	15/02/16	19/02/16
Sails.js	22/02/16	26/02/16
Ionic	29/02/16	4/03/16
MongoDB	7/03/16	11/03/16
Otras herramientas	14/03/16	18/03/16
Planificación de requisitos	21/03/16	24/03/16
Evaluación	21/03/16	24/03/16
Diseño	25/03/16	4/04/16
Diseño de interfaz	25/03/16	29/03/16
Maquetado final	30/03/16	4/04/16
Desarrollo gestión de usuarios	5/04/16	14/04/16
Evaluación	5/04/16	7/04/16
Desarrollo Back-end	6/04/16	14/04/16
Desarrollo Front-end	8/04/16	13/04/16
Desarrollo de autenticación	15/04/16	29/04/16
Evaluación	15/04/16	19/04/16
Desarrollo Back-end	20/04/16	29/04/16
Desarrollo Front-end	25/04/16	27/04/16
	2/05/16	23/05/16
Evaluación	2/05/16	4/05/16
Desarrollo Back-end	5/05/16	13/05/16
Desarrollo Front-end	10/05/16	23/05/16
Desarrollo gestión de pedidos	24/05/16	14/06/16
Evaluación	24/05/16	27/05/16
Desarrollo Back-end	30/05/16	6/06/16
Desarrollo Front-end	3/06/16	14/06/16
Menu de opciones	15/06/16	23/06/16
Pruebas finales	24/06/16	6/07/16

Figura 2. Planificación inicial

4 Análisis y desarrollo del proyecto

4.1 Base de datos

Para la base de datos se ha optado por usar mongoDB, se trata de una base de datos NoSQL. Esto quiere decir que a diferencia de las SQL esta guarda los datos en documentos tipo JSON y se caracteriza por [12]:
- Soporta búsquedas en la bases de datos tanto por campos como por rangos y expresiones regulares.
- Los campos pueden ser indexados.
- Se realizan las consultas con JavaScript, estas consultas son enviadas directamente a la base de datos.

A continuación se muestra la estructura de base de datos.

5.1.a Usuarios. La colección User contiene los campos username, email y password de cada usuario.

5.1.b Productos. En esta ocasión la colección Products contiene los datos del cada producto, nombre, descripción, precio, imagen.

5.1.c Pedidos. Por último la colección de pedidos cuenta con la id del usuario, primordial para relacionar pedido y usuario. Por otra parte contiene un campo llamado product tipo JSON que permite guardar la colección de productos.

4.2 Desarrollo del proyecto

En este apartado se explica con más detalle el desarrollo de las funcionalidades básicos del proyecto:

- Desarrollo de autenticación

Para la implementación de esta funcionalidad se ha decidido utilizar el módulo de Node.js llamado passport.js. Este módulo hace de middleware, dispone de diversas estrategias de autenticación aunque en este caso se ha optado por la básica que consiste en sesión con cookies [13].
En el back-end se han aprovechado los métodos básicos CRUD (Create, Read, Update, Delete) que proporciona Sails.js, estos métodos están integrados en el propio modelo y permite realizar operaciones de base de datos [14]. Para esta etapa del proyecto no ha sido necesario implementar métodos adicionales.

Desde el front-end el usuario se puede autenticar contra el servicio web restful, enviando el email y la contraseña.

- Desarrollo lista de productos

Del mismo modo que en el caso anterior, para la implementación de la lista de productos también se han aprovechado los métodos CRUD de sails. Por lo que sólo ha sido necesario crear el modelo del producto con los campos que lo identifican.
En el front-end se ha desarrollado el catálogo de productos y la funcionalidad que permite agregar y eliminar productos de la base de datos.

Figura 3. Catálogo de productos

- Desarrollo gestión de pedidos

En la parte del back-end se ha tenido que implementar la funcionalidad de agregar/eliminar todos los pedidos. En este caso se ha tenido que extender la funcionalidad básica que ofrece Sails.js.

En cuanto al front-end se ha desarrollado la funcionalidad de realizar y enviar pedidos. Por una parte, para que el usuario pueda ver el contenido del carrito antes de enviar

su pedido se ha utilizado el módulo ngStorage de Angular, concretamente el servicio de $localStogare. Con esto se consigue guardar en un array el pedido dentro del navegador antes de ser enviado al servidor [15].
Por otra parte se ha implementado la funcionalidad que permite enviar los pedidos desde el cliente al servidor.

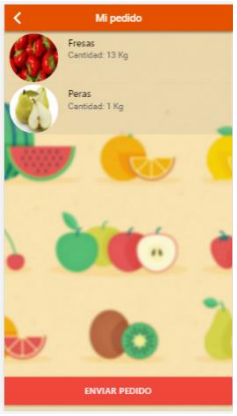

Figura 4. Carrito

- Desarrollo gestión de usuarios

En esta fase del proyecto se ha desarrollado el método que se encarga de asignar roles a los usuarios. Para llevar a cabo esta funcionalidad se ha utilizado el módulo sails-permissions. Este módulo soporta autenticación con passport.js y concede permiso basándose en los roles [16].
Para incorporar sails-permissions en el proyecto ha sido necesario desviar la planificación, se explica con más detalle en la conclusión.

Sails-permissions genera automáticamente un usuario administrador con las variables por defecto que son (se puede cambiar):

```
sails.config.permissions.adminUsername = 'admin'
sails.config.permissions.adminEmail = 'admin@example.com'
sails.config.permissions.adminPassword = 'admin1234'
```

Figura 5. Variables por defecto para entorno usuario admin

Dado que para esta aplicación los roles están bien definidos, se ha decidido generar automáticamente el role de transporter, merchant, shopper al iniciar el servidor.
Por último se ha implementado un método en el back-end que permite asociar un rol a un usuario.

En el front-end se ha generado la funcionalidad que permitirá al administrador asociar un rol ya definido a un usuario. Para ello deberá introducir username, email, password y seleccionar uno de los roles.

- Menú de opciones

En cuanto al menú de opciones tanto el administrador como el comprador cuentan con un menú, en el que se les muestran las tres opciones que pueden seleccionar. Alta usuario, editar catálogo o eliminar pedidos.

Por último, en el apéndice A1 se adjunta un diagrama de casos de uso para una mejor comprensión del proyecto.

4.3 Arquitectura

En la Figura 6 se muestran todas las tecnologías utilizadas para el desarrollo de la aplicación.

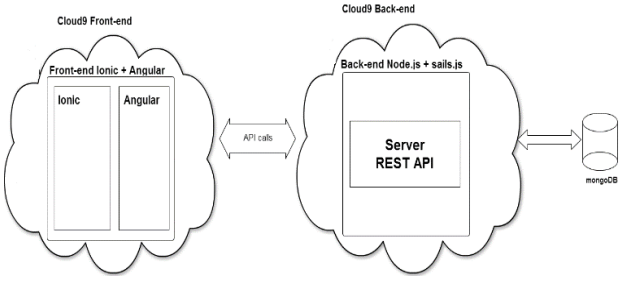

Figura 6. Arquitectura

5 Resultados

Como resultado final del proyecto hemos obtenido una aplicación que cumple con la mayoría de los objetivos propuestos inicialmente. Se ha conseguido presentar una herramienta que permite gestionar la parte de compra de género de pequeños comercios.
Los usuarios, dependiendo del rol, pueden visualizar un catálogo y efectuar sus pedidos. Así también se puede acceder a un listado detallado de pedidos. A nivel de administrador se puede crear/eliminar productos de la base de datos y dar de alta a nuevos usuarios. En la Tabla 2 se presenta un resumen de los requisitos cubiertos:

Tabla 2
Requisitos cubiertos

Requisitos Iniciales	Requisitos Cubiertos
El sistema debe poder hacer login	OK
El sistema debe permitir dar de alta usuarios	OK
El sistema debe permitir agregar/eliminar productos de la base de datos	OK
El sistema debe mostrar catálogo	OK
El sistema debe permitir generar y enviar pedidos	OK
El sistema debe permitir activar y desactivar productos del catálogo	OK
El sistema debe ver lista de pedidos	OK
El sistema debe permitir visualizar punto de recogida y de entrega de mercadería	OK
El sistema debe permitir agregar productos al carrito	OK
El sistema debe mostrar la lista de pedido antes de enviar	OK
El sistema debe poder hacer login	OK

6 Conclusión

6.1 Gestión de cambios

A continuación se explicará con más detalle los problemas que han ido surgiendo durante el desarrollo de la aplicación, y los cambios que estos han obligado a efectuar.

6.1.1 Planificación

En primer lugar en el momento de implementar la etapa de gestión de usuarios surgieron varios problemas. En un principio se tenía planteado desarrollar esta sección con el módulo sails-generate-role pero debido a que la aplicación no la acababa de integrar por una serie de fallos [17], se ha decidido probar con sails-permissions pero sólo se había conseguido trasladar el problema a otro módulo [16].

Dado que en ese momento no se encontraba la solución rápida se detectó que esta etapa requeriría de más tiempo por lo que se ha optado por desplazarla como última fase de desarrollo del proyecto. A pesar de ser un requisito prioritario, esto no ha significado que las demás funcionalidades no se pudieran ir desarrollando. Así que la solución fue aparcarla temporalmente. Por esta razón la planificación final ha quedado como se muestra en a Figura 7.

Nombre	Fecha de inicio	Fecha de fin
Documentación del proyecto	15/02/16	22/06/16
Formación JavaScript	15/02/16	18/03/16
JavaScript	15/02/16	19/02/16
Sails.js	22/02/16	26/02/16
Ionic	29/02/16	4/03/16
MongoDB	7/03/16	11/03/16
Otras herramientas	14/03/16	18/03/16
Planificación de requisitos	21/03/16	23/03/16
Evaluación	21/03/16	23/03/16
Diseño	24/03/16	25/03/16
Diseño de interfaz	24/03/16	24/03/16
Maquetado final	25/03/16	25/03/16
Desarrollo de autenticación	28/03/16	7/04/16
Evaluación	28/03/16	7/04/16
Desarrollo Back-end	29/03/16	6/04/16
Instalación modulos Node.js	29/03/16	6/04/16
Creación modelo usuario	29/03/16	6/04/16
Login y autenticación	29/03/16	6/04/16
Desarrollo Front-end	6/04/16	7/04/16
Vista login	6/04/16	7/04/16
Autenticar contra el servidor	6/04/16	7/04/16
Desarrollo de la lista de productos	8/04/16	15/04/16
Evaluación	8/04/16	8/04/16
Desarrollo Back-end		12/04/16
Creación modelo producto	8/04/16	12/04/16
Agregar/Eliminar producto	8/04/16	12/04/16
Desarrollo Front-end	13/04/16	15/04/16
Vista del catálogo de pedidos	13/04/16	15/04/16
Desarrollo gestión de pedidos	18/04/16	29/04/16
Evaluación	18/04/16	19/04/16
Desarrollo Back-end	20/04/16	29/04/16
Agregar pedidos a la base d...	20/04/16	29/04/16
Desarrollo Front-end	25/04/16	27/04/16
Vista de los pedidos	25/04/16	27/04/16
Desarrollo gestión de usuarios	2/05/16	23/06/16
Evaluación	2/05/16	23/05/16
Desarrollo Back-end	5/05/16	13/05/16
Asignación de rol y permiso...	5/05/16	13/05/16
Desarrollo Front-end	10/06/16	23/06/16
Admin (alta usuarios)	10/06/16	23/06/16
Menu de opciones	24/05/16	14/06/16
Pruebas finales	15/06/16	23/06/16

Figura 7. Planificación final

6.1.2 Fase gestión de pedidos

Por otra parte en la gestión de pedidos se tuvo realizar cambios en el planteamiento.
El principal problema en esta etapa era el hecho de asociar cantidad a los productos. En un principio la relación entre el modelo pedido y producto era 'many to many' [18], pero se tuvo que cambiar ya que el ORM de Sails, Waterline, no permite añadir atributos en la tabla intermedia.

Como solución se creó el campo productos en pedidos, como un JSON de id de producto y cantidad, ya que se necesitaba un atributo que guardara la cantidad de productos pedidos.

Aunque no se aprovechen las funcionalidades de Waterline y sus relaciones, no ha provocado ningún impacto de rendimiento gracias a que la base de datos utilizada no es relacional. En cambio ha supuesto un gran impacto a nivel de programación y de tiempo, ya que se tuvo que investigar mucho para dar con una solución correcta.

6.1.3 Fase gestión de usuario

Una vez que estaba en marcha la etapa de gestión de usuarios, concretamente la creación y asignación de roles mediante el módulo sails-permissions [16]. Se notó que dicho módulo no era compatible con el login que previamente se había desarrollado.

Después de varias búsquedas se encontró el problema y se aplicó la solución que menos impacto provocara. El problema se producía porque sails-permissions ya encapsula la autenticación de usuario con passport.js [16].

En este punto se ha tenido que desviar la planificación e integrar el módulo que gestiona los roles con el módulo de autenticación. La mejor solución fue utilizar el mismo sails-permissions tanto para la autenticación de usuario como para la creación y asignación de roles.

A nivel de impacto, ha supuesto un desvío en la planificación pero ningún impacto respecto a la funcionalidad final.

6.2 Líneas de futuro y ampliaciones

A medida que el proyecto avanzaba se han detectado posibles mejoras a nivel de funcionalidad.

Por una parte se podría mejorar en cuanto a la gestión de pedidos. Cuando un usuario realice un pedido que reciba automáticamente una copia en su email. Y así evitar perder datos cuando el administrador tenga la necesidad de eliminar todo los pedidos de la base de datos.

Por otra parte se podría ampliar la información que visualiza el transportista. Que en lugar de ver solo el nombre de quien realiza el pedido y los detalles del pedido, pueda ver la localización del punto de recogida de los productos (en el mercado) y de entrega (tiendas).
Es una ventaja importante que la aplicación sea escalable, ya que permitirá seguir desarrollando hasta conseguir una aplicación útil y única en el mercado.

A nivel personal me ha servido para aprender sobre diversos frameworks, me ha ayudado a entender el funcionamiento los pequeños comercios.

AGRADECIMIENTOS

A mi tutor de proyecto, Jordi Duran Cals, y a Javier B. Martínes. Gracias a los dos por confiar en mí, y por toda vuestra ayuda en el desarrollo de este proyecto.

BIBLIOGRAFÍA

[1] Es.wikipedia.org. (2016). Terminal punto de venta. [online] Disponible en: https://es.wikipedia.org/wiki/Terminal_punto_de_venta [Accedido 20 Junio 2016].
[2] Play.google.com. (2016). [online] Disponible en: https://play.google.com/store/apps/details?id=appinventor.ai_jaumetortosavalles.TPVSimpleBarFree&hl=es [Accedido 20 Junio 2016].
[3] Play.google.com. (2016). [online] Disponible en: https://play.google.com/store/apps/details?id=amalgame.emanager&hl=es [Accedido 20 Junio 2016].
[4] Álvarez, C. (2014). ¿Cómo funciona Node.js?. [online] Genbetadev.com. Disponible en: http://www.genbetadev.com/frameworks/como-funciona-node-js [Accedido 9 Marzo 2016].
[5] Ionicframework.com. (2016). Ionic Documentation - Ionic Framework. [online] Disponible en: http://ionicframework.com/docs/ [Accedido 10 Marzo 2016].
[6] zachfitzgerald.com, Z. (2016). Ionic Material / Material Design / Ionic Framework / AngularJS / Zach Fitzgerald. [online] Ionicmaterial.com. Disponible en: http://ionicmaterial.com/ [Accedido 9 Mayo 2016].
[7] GitHub. (2016). balderdashy/waterline. [online] Disponible en: https://github.com/balderdashy/waterline [Accedido 11 Marzo 2016].
[8] contributors, M. (2016). Models and ORM | Sails.js Documentation. [online] Sailsjs.org. Disponible en: http://sailsjs.org/documentation/concepts/models-and-orm [Accedido 18 Febrero 2016].
[9] contributors, M. (2016). Blueprints | Sails.js Documentation. [online] Sailsjs.org. Disponible en: http://sailsjs.org/documentation/concepts/blueprints [Accedido 9 Marzo 2016].
[10] Metodologia RAD. (2016). Metodologia RAD. [online] Disponible en: http://metodologiarad.weebly.com/ [Accedido 7 Marzo 2016].
[11] C9.io. (2016). Cloud9 - Your development environment, in the cloud. [online] Disponible en: https://c9.io/ [Accedido 17 Marzo 2016].
[12] Genbetadev.com. (2014). MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no). [online] Disponible en: http://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no [Accedido 9 Abril 2016].
[13] Passportjs.org. (2016). Passport. [online] Disponible en: http://passportjs.org/ [Accedido de 17 Abril 2016].
[14] contributors, M. (2016). Models | Sails.js Documentation. [online] Sailsjs.org. Disponible en: http://sailsjs.org/documentation/concepts/models-and-orm/models#?model-methods-aka-static-or-class-methods [Accedido 9 de Abril 2016].
[15] GitHub. (2016). gsklee/ngStorage. [online] Disponible en: https://github.com/gsklee/ngStorage [Accedido 17 Marzo 2016].
[16] GitHub. (2016). langateam/sails-permissions. [online] Disponible en: https://github.com/langateam/sails-permissions [Accedido 18 Abril 2016].
[17] GitHub. (2016). yasoonOfficial/sails-generate-role. [online] Disponible en: https://github.com/yasoonOfficial/sails-generate-role [Accedido 24 Febrero 2016].
[18] contributors, M. (2016). Many-to-Many | Sails.js Documentation. [online] Sailsjs.org. Disponible en: http://sailsjs.org/documentation/concepts/models-and-orm/associations/many-to-many [Accedido 24 Abril 2016].
[19] contributors, M. (2016). Sails.js | Realtime MVC Framework for Node.js. [online] Sailsjs.org. Disponible en: http://sailsjs.org/ [Accedido 24 Abril 2016].

APÉNDICE

A1. DIAGRAMA CASOS DE USO