Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una herramienta para la gestión de pedidos proveedores con tecnologías Spring

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Francisco Jesús Pérez Chavarría

Tutor: Joan Fons Cors

2019/2020

Agradecimientos

A mi familia y amigos por su apoyo y comprensión durante estos años ya que sin ellos no habría terminado.

A mi tutor Joan Fons Cors, por después de tantos años fuera, motivarme a adaptar el grado y cambiar de vida

.

Resumen

Este proyecto surge para plantear una forma diferente de trabajo en una multinacional dedicada al retail acerca de la gestión que realizan sus mandos en la revisión y seguimiento de sus pedidos proveedores. Actualmente la gestión de los mismos es revisada mediante hojas de cálculo las cuales muestran un listado de la mercancía en curso hacia la tienda en ese momento.

La automatización de ese análisis permite reducir errores y el tiempo dedicado a la gestión, unificando una forma de trabajo y una plataforma para la comunicación transversal entre departamentos.

Abstract

The propose of this project is to promote a different way of working in a multinational retail corporation regarding the management carried out by their managers in the review and tracking of their purchase orders. Currently, their management is reviewed through spreadsheets which show a list of the orders in progress.

The automation of this analysis allows to reduce errors and the time dedicated to management, promoting a unified working method and communication platform between departments.

Tabla de contenidos

Tabla de Imágenes

1. Introducción

Este proyecto surge para plantear una forma diferente de trabajo en una multinacional respecto a la gestión que realizan sus mandos en la revisión y seguimiento de sus pedidos proveedores. Actualmente la gestión de estos es revisada mediante hojas de cálculo las cuales muestran un listado de la mercancía en curso hacia la tienda en ese momento. Este fichero es conocido como LPRE.

Se plantearán las dificultades que genera el sistema actual y los beneficios que aporta y problemáticas que resuelve la aplicación que se va a desarrollar. Esta herramienta se centra en el seguimiento y revisión de estos, así como en el desarrollo del proyecto mediante las herramientas que nos ofrecen algunos de los proyectos de Spring.

1.1. Motivaciones

Una de las tareas que realiza un responsable de departamento en una empresa de retail es la realización y seguimiento de pedidos para reabastecer de mercancía la superficie comercial. En la empresa en la que se realizó el estudio existen diferentes tipos de pedidos y partes implicadas en la gestión y seguimiento de estos. Diariamente puede haber hasta 500 pedidos en camino hacia las tiendas.

Los pedidos a proveedores pueden ser realizados por un sistema de reaprovisionamiento, vendedores, responsables y central de compras. Al haber tantos actores implicados en esta tarea es difícil tener una visión global sobre que pedidos se han realizado y el propósito de estos.

Conocer el volumen de entregas diarias es importante para poder organizar adecuadamente el departamento y tener un control sobre pedidos críticos como son los realizados para operaciones comerciales, cambios de colección, repuestos...

Para realizar un pedido a proveedor hay que tener una serie de conocimientos previos sobre cómo funciona el sistema actual y el funcionamiento de estos como son el franco del proveedor, plazos de entrega, condicionantes, bonificaciones y previsiones de venta. Habitualmente las propuestas de compra son realizadas por la herramienta de la empresa, la cual sigue una serie de reglas que en ocasiones puede causar más problemas de los que soluciona. Estos problemas se refieren principalmente a sobre stock en referencias concretas o la realización de pedidos que no llegarán a franco, los cuales nunca serán entregados por los diferentes proveedores.

La revisión de estos pedidos corresponde a los mandos intermedios de departamento que se encargan de comprobar que estos pedidos hayan sido realizados correctamente para que lleguen con éxito a la tienda. No existe un

procedimiento estándar para el análisis de los mismos, ni todos los colaboradores tienen la habilidad ni el tiempo para realizar este análisis de una forma exhaustiva.

Uno de los principales problemas de los pedidos proveedores son los retrasos en entrega. En ocasiones los pedidos no están realizados correctamente debido al montante o la fecha de entrega del pedido. Detectar estos problemas es importante para que esos retrasos no se produzcan ya que, al haberse realizado incorrectamente habría que anular y volver a generar un pedido nuevo. El momento de detección de estos es clave para poder tener un tiempo de respuesta mayor en la resolución de este puesto que, normalmente, cuando alguna mercancía es pedida, suele ser porque hay un cliente detrás y manejamos fechas muy ajustadas.

También es muy importante la comunicación con la recepción de mercancía de la tienda puesto que, para determinadas épocas del año se recibe mucha más mercancía de lo habitual llevando al colapso de esta. Estas tareas de detección se realizan desde la recepción manualmente sobre el mismo fichero que utiliza la aplicación. Aquí, es imposible saber con un mínimo de acierto, el volumen de mercancía que se va a recibir ya que se disponen de más de 50,000 referencias en stock y 500.000 bajo pedido.

1.2. Objetivos

El objetivo de este trabajo consiste en diseñar e implementar una aplicación web que permita estructurar la información relacionada con los proveedores y sus pedidos. También pretende unificar una forma de trabajo de una manera más ágil, rápida y productiva que el sistema actual utilizado en la compañía. Se automatizarán las tareas de análisis siendo las mismas para todos los colaboradores basándose en un consenso para realizar las mismas ya que, actualmente no es igual gestionar una sección de picking a una sección proyecto.

Estos objetivos serán los siguientes:

1. El sistema será capaz de detectar una serie de alertas acerca de pedidos en curso.

2. Proporcionará un espacio con las alertas actuales en pedidos para poder ser gestionadas.

3. Facilitará un listado de pedidos en curso.

4. Mostrará el estado de cumplimiento de revisión de pedidos.

5. Permitir clasificar los pedidos en función de su tipo y flujo logístico.

6. Facilita el seguimiento de pedidos.

7. Mostrará de manera visual los detalles relacionados con la entrega de pedidos en las próximas 2 semanas.

8. El sistema utilizara los ficheros LPRE, que serán recibidos en una cuenta de correo por el departamento de reportes de levante. La información de este fichero será la que utilice la aplicación para realizar un primer análisis de estos y la creación de alertas ya que no es posible integrarlo dentro de la propia empresa.

9. Todo el proyecto será desarrollado mediante las herramientas que proporciona Spring para realizarlo de la forma más optima y de calidad.

1.3. Propuesta inicial

Inicialmente estudiaré la manera de conseguir obtener la información de una manera automatizada para que sea viable en el tiempo e informe de las intenciones y objetivos a los principales responsables de la empresa.

La primera intención es conseguir que la aplicación funcione mínimamente sobre los objetivos propuestos. Esto supone que deberá poder descargar los correos y realizar todo el proceso de actualización de la aplicación de manera autónoma.

Además, ofrecer una interfaz en la que poder iniciar sesión y visualizar un apartado de alertas en las que únicamente aparecieran las de un tipo. Estas fueron inicialmente el retraso en los pedidos.

Lo importante de este proyecto era hacerlo útil y que se utilizase, y por la experiencia en proyectos anteriores, esto no era posible si no se incorporaba los aportes de los usuarios finales desde etapas muy tempranas del desarrollo.

1.4. Metodología y planificación

Para la propuesta inicial, podría decirse que se utilizó una metodología en cascada puesto que se tenía un alcance definido como se ha comentado en la propuesta inicial. Esta parte del proyecto se realizó en las etapas típicas de este ciclo de vida.

Nº Actividad	Inicio	Final	04-mar	11-mar	18-mar	25-mar	01-abr	08-abr	15-abr	22-abr	29-abr	06-may	13-may	20-may	27-may	03-jun	10-jun	17-jun
Requisitos	04/03/2019	18/03/2019																
Diseño	19/03/2019	08/04/2019																
Implementación	09/04/2019	06/05/2019																
Pruebas	07/05/2019	20/05/2019																
Mantenimiento	21/05/2019	20/06/2019																

Imagen 1Diagrama de GANT

Una vez que ya existía una aplicación en la que era posible iniciar sesión y ver un tipo de alerta que se actualizaba todos los días, se dedicaba un breve tiempo en los 'comités de compras' para sugerir nueva funcionalidad por los usuarios y se definía cual de ella era alcanzable para próximas reuniones. Estos comités se realizaban 1 vez al mes y trataban temas relacionados con la calidad del stock de las secciones comerciales, planificación de operaciones comerciales y demás asuntos relacionados, por lo que era el foro adecuado para incorporar la utilización y colaboración con la herramienta.

Debido a la frecuencia de realización de estos comités y la presencia de todas las partes implicadas, era el momento de cambiar la metodología de trabajo y aplicar Scrum. Esta metodología se centra en entregar funcionalidad constantemente puesto que un proyecto puede cambiar constantemente durante su desarrollo. Este tipo de desarrollo es incremental y divide el proyecto en ciclos de desarrollo o Sprints, los cuales finalizan con la entrega de alguna funcionalidad útil.

Se aprovechaban estos comités para captar nuevas necesidades, explicar y formar a los usuarios en los desarrollos realizados en el sprint anterior y detectar dificultades.

En el momento que se pudo utilizar diariamente la aplicación surgieron muchas propuestas que se incorporarían en pila de producto para un posterior desarrollo.

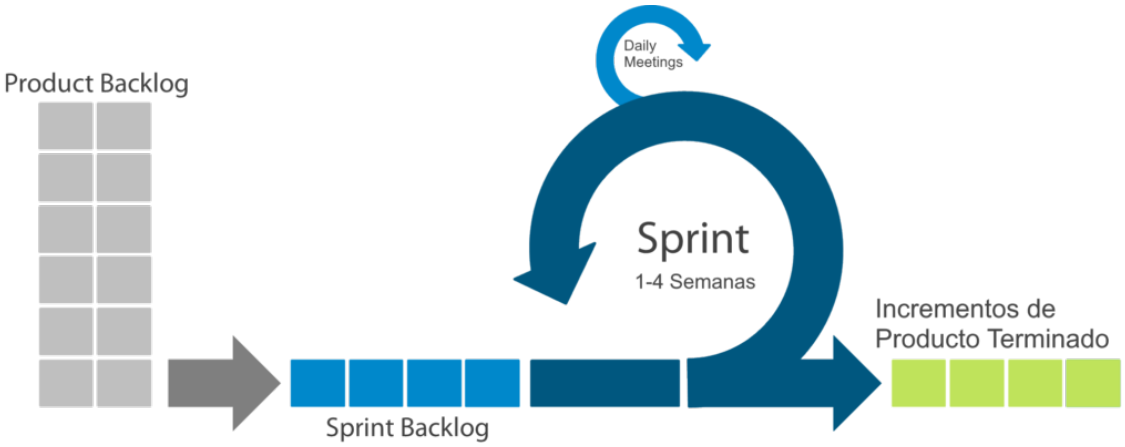

Imagen 2 Diagrama SCRUM

2. Contexto tecnológico

Esta aplicación será desarrollada sobre Java EE utilizando Apache Tomcat 8.5.43 como servidor de aplicaciones.

Para el almacenamiento de datos utilizaremos HSQLDB en desarrollo y en producción MySQL Server.

El entorno utilizado es Linux Ubuntu Server 18.04

El IDE de desarrollo será Spring Tool Suite 4, este es una adaptación del IDE eclipse con utilidades para facilitar la utilización del Framework Spring.

2.1. Tecnologías utilizadas

A continuación, enumerare los principales elementos que se utilizan para construir la aplicación con una breve descripción de su utilidad dentro del proyecto.

2.1.1. Frameworks y tecnologías transversales:

- **Spring Framework**

Spring es un framework caracterizado por la inversión de control (IoC). Para el desarrollo de la aplicación utilizaremos un módulo basado en MVC (Modelo-Vista-Controlador). Este módulo está basado en HTTP y servlets, que proveen de herramientas para la extensión y personalización de aplicaciones web y servicios REST.

Gran parte de la adquisición de conocimientos los he adquirido inicialmente a través del libro “Desarrollo de aplicaciones mediante el Framework de Spring” que está incluido en el apartado de documentación, así como por la documentación del sitio oficial de Spring.

- **Maven**

Maven es una herramienta para la gestión y construcción de proyectos Java. Utiliza un archivo XML en el que se nombran las dependencias que tendrá el proyecto y componentes necesarios para poder compilarse y arrancar.

Este fichero se denomina POM (Project Object Model). He optado por utilizar esta herramienta a diferencia de Gradle puesto que me resulta más familiar y me permite gestionar las dependencias, compilación y perfiles del proyecto.

Este proyecto no está 'Mavenizado', por lo que únicamente existe un único fichero POM.

Para futuros desarrollos el proyecto será dividido en subproyectos para que estos mismos puedan ser compilados en diferentes librerías que podrán ser importadas como dependencias entre los mismos. Estos serían un módulo de entidades, repositorio, implementación de los servicios, api de los servicios, common, front-end y recursos. Esto lo comentaré en futuras mejoras de la aplicación.

- **Spring Boot**

Boot proporciona un entorno amigable para el arranque del proyecto. Con una configuración mínima se consigue poner en marcha una aplicación con la tecnología Spring.

Es capaz de configurar automáticamente el contexto de Spring basándose en las librerías o archivos JAR incluidos en la aplicación. Por ejemplo, si Spring Boot encuentra en el classpath de la aplicación la librería de MySQL y se ha activado la autoconfiguración, Spring Boot automáticamente configurará la comunicación con la base de datos.

También incluye un soporte para incluir el servidor Tomcat embebido en la aplicación, por lo que no será necesario desplegar la misma en un servidor de aplicaciones externo.

Esta funcionalidad agiliza enormemente el desarrollo de aplicaciones, puesto que con una mínima configuración permite el arranque de aplicaciones desde el propio entorno de desarrollo sin realizar ningún ajuste adicional, además de facilitar las tareas de reinicio y depuración.

La mayor parte de este conocimiento lo he adquirido a partir del sitio oficial de Spring Boot, indicado en el apartado de bibliografía.

- **Spring Roo**

Roo es una herramienta para el desarrollo rápido de aplicaciones en Java. Este es un proyecto desarrollado por Pivotal y Disid(Valencia) y forma parte de las aplicaciones desarrolladas en el portafolio de los proyectos de Spring. Puesto que personalmente considero que este proyecto ha sido un gran descubrimiento para mí, dedicare especial atención a su utilización en la etapa de desarrollo.

Los principales frameworks que se utilizaran son Spring Framework en la versión 5.3.10, Hibernate en la versión 4.2.1 y Apache Tiles en la versión 3.0.7.

Como explicaré en la parte de implementación, esta herramienta será el punto de partida en la fase de desarrollo puesto que una vez que se conoce la relación entre los objetos reflejados en el diagrama de clases, es muy sencillo trasladarlo a Roo para tener un CRUD completamente funcional.

- **AspectJ**

Para la gestión de librerías se utilizará Maven. Esta herramienta permite, entre otras muchas funciones, la gestión de librerías y dependencias que se utilizaran en el proyecto., así como la separación por perfiles y el empaquetado de las mismas.

Es necesario para la utilización de Spring Roo ya que añade unos plugins de compilación para interpretaros los ficheros “*.aj” propios de esta tecnología. Por mi experiencia, los IDE como eclipse no suelen llevarse muy bien con estos por lo que, como explicaré posteriormente al arranque del proyecto por Spring Roo, realizaremos el proceso de push-in y eliminaremos estos plugins dejando todo el código en fichero *.java.

2.1.2. Tecnologías en BackEnd:

- **JPA**

Es la API de persistencia que utilizará Hibernate. Esta permite no perder las ventajas de la orientación a objetos al interactuar con la base de datos. Es

necesario para la utilización de Spring Data y nos recudirá enormemente las líneas de código necesarias para realizar operaciones sencillas.

- **Hibernate**

Hibernate es un framework de mapeo objeto-relacional (ORM) que facilita el mapeo de atributos entre una base de datos tradicional y el modelo de objetos de una aplicación mediante anotaciones (Hibernate Annotations) en los beans o archivos XML declarativos. Permite adaptarse a una base de datos ya existente o crearla a partir de la información contenida en los objetos.

La primera toma de contacto con esta tecnología la realice mediante el libro relacionado con Hibernate comentado en el apartado de bibliografía el cual, si bien es algo antiguo me sirvió para establecer los pilares para comprender el funcionamiento del mismo.

- **Criteria**

Son un conjunto de librerías que simplifican la forma de realizar búsquedas de datos en la base de datos. Spring Data no permite realizar todo tipo de consultas y con esta tecnología salvamos esa dificultad.

- **Spring Data JPA**

Este proyecto presenta una gran cantidad de utilidades referentes a la comunicación con la base de datos. Para este proyecto utilizaremos principalmente Spring Data JPA. Esta proporciona una interfaz para tener un DAO completamente funcional y la posibilidad de realizar métodos de búsqueda únicamente definiendo la firma de los métodos.

- **Apache POI**

Apache POI es una librería que permite la lectura de ficheros basados en el formato de Microsoft OLE 2 Compound Document Format. También permite la lectura de ficheros con formato Office Open XML (OOXML). Esta librería es la que nos permite extraer la información de las hojas de cálculo con extensión XLSX entre otras.

Las herramientas de reportes de la compañía utilizan este formato y esta librería se utiliza de una forma bastante intuitiva si se está familiarizado con estas hojas de cálculo.

2.1.3. Tecnologías en FrontEnd:

- **Apache Tiles**

Apache tiles es un framework de composición de plantillas. Permite a los autores definir fragmentos que se pueden ensamblar en las páginas en tiempos de ejecución. Estos fragmentos permiten reducir la duplicidad de código común a lo largo del proyecto centralizando los cambios en pocos documentos.

Aunque esta tecnología ya se encuentra obsoleta, es la que aprendí hace años y con la que realice las primeras versiones. Actualmente se utilizan otros motores de plantillas como Thymeleaf que son recomendados por Spring e incluyen la configuración con Spring Boot. La utilización de ambas es bastante similar.

- **JSTL**

JavaServer Pages Stantard Tag Library (JSTL) es un componente de Java EE. Extiende las JavaServer Pages (JSP) proporcionando cuatro bibliotecas de etiquetas con utilidades ampliamente utilizadas en el desarrollo de páginas web dinámicas.

Estas etiquetas son las que nos permiten incluir contenido dinámico dentro de las páginas html.

- **Swagger**

Swagger es un framework muy útil para poder documentar y probar las APIs REST que desarrollemos dentro de nuestro proyecto. Esta información es muy útil para los desarrolladores puesto que permite conocer y consumir los servicios que hagamos públicos. Swagger genera una interfaz que se actualiza en función de las anotaciones que vamos introduciendo dentro del proyecto.

Se ha incluido para poder ofrecer servicios al exterior y porque para futuros desarrollos está planteado desarrollar el interfaz con Angular.

- **Bootstrap**

Bootstrap es una framework para el diseño de sitios web. Este framework solo se ocupa del desarrollo del front-end. Incluye plantillas de diseño y otros elementos basados en HTML, CSS y javascript para adaptar la aplicación a diferentes dispositivos.

Inicialmente se optó por utilizar JSF e incluir Primefaces para toda la creación de formularios, tablas y componentes, pero finalmente se decidió utilizar Spring MVC y esto excluye a esa tecnología.

- **ChartJS**

ChartJS es una librería open source bajo licencia del MIT en JavaScript que permite la creación de distintos tipos de graficas totalmente personalizables y animadas.

Esta librería es utilizada para realizar gráficos de barras y gráficos de progreso.

2.1.4. Otras:

- **Git & GitHub**

Git es la herramienta que utilizare para el control de versiones y debido a que trabajo en diferentes máquinas, utilizaré GitHub como si de un proyecto colaborativo se tratase.

Existen otros sistemas como subversión o bitbucket pero github es gratuito, ofrece repositorios privados y dispone de multitud de clientes integrados con el IDE.

- **Otras librerías**

JavaMail es una API Java que facilita el envío y recepción de emails desde código java a través de protocolos SMTP, POP3 y IMAP. Está integrado dentro del framework Java EE.

Puesto que la información se recibe mediante correo electrónico, es necesario utilizar alguna librería para poder comprobar el estado de este, descargar los ficheros necesarios y notificar cambios en alertas a algunos usuarios.

2.2. Estado del Arte

La empresa en la que se ha desarrollado utiliza un ERP / CRM desarrollado hace más de 25 años el cual controla desde la impresión de etiquetas de precio, recepción de pedidos en logística, planificación de ofertas, históricos de ventas, pedidos a clientes, etc.

Debido a esto, se están realizando desarrollos paralelos para cubrir todas las necesidades que van surgiendo en aplicaciones paralelas y de tareas concretas, como la actualización de las interfaces de gestión e históricos de ventas.

Este proyecto podría clasificarse como un módulo Ad-hoc dentro de una aplicación ERP ya que hay partes que son realizadas por cualquier aplicación de este estilo como podría ser añadir comentarios sobre un pedido. Este desarrollo se deberá realizar a medida debido a la casuística concreta y necesidades de la empresa (número de proveedores, fechas de entregas y volúmenes, circuitos logísticos) lo cual no es tan habitual en las mismas tipologías de negocio.

Hay aplicaciones que complementan la parte relacionada con volúmenes como son los SGA, pero estos están más orientados hacia los espacios en los almacenes. La aplicación aporta valor para este tipo de sistemas en una etapa previa puesto que después de la revisión de un pedido se puede indicar el volumen y que debería de ocurrir con la mercancía de estos.

3. Requisitos del sistema

A continuación, se nombrarán las tareas que realizarán los diferentes actores del sistema.

- Sistema:
  - Revisión de email con nuevos informes para actualizar el sistema.
  - Creación y actualización de estados de los pedidos en curso del sistema.
  - Detección de alertas en nuevos pedidos como compra de artículos en vías de supresión, retrasos, fechas próximas de entregas para los que se requirieron avisos, pedidos que no llegan a franco.

- Colaboradores
  - Revisión de pedidos en el sistema.
  - Clasificación por tipología y etiquetas.
  - Añadir comentarios para el seguimiento de estos.
  - Modificación de fechas de entrega.
  - Envío de correos genéricos a proveedores y diferentes departamentos.
  - Búsqueda de pedidos en el sistema.
  - Editar el volumen de palets en pedidos.

A continuación, en lo referente al análisis del problema, se mostrarán los requisitos funcionales de la misma. Estos requisitos sirven para comprender de una mejor manera mostraré algunos de los bocetos mostrados a los usuarios finales para recibir su aprobación y continuar con el desarrollo. También incluiré los diagramas de clases de los subsistemas generados y los diagramas de secuencia. Estos fueron modificados hasta llegar a su versión final.

3.1. Requisitos

Entendemos los requisitos de la aplicación como una condición para resolver un problema. Esta capacidad deberá estar presente en el sistema para satisfacer esos requerimientos. Estos requerimientos deberán ser claros y deben estar definidos en cualquier documento de análisis.

3.1.1. Requisitos funcionales

Los requisitos funcionales definen los comportamientos del software que serán implementadas.

- El sistema se actualizará automáticamente con la información obtenida.
- Se crearán tiendas, secciones, proveedores y pedidos con esa información.
- Se mostrará información detallada acerca de los pedidos en curso de las secciones comerciales.
- Se podrá comunicar con los proveedores mediante emails específicos relacionados con los pedidos proveedores.
- Se podrán clasificar los pedidos en función de su circuito logístico.
- Se podrán clasificar los pedidos en grupos creados por el usuario.
- Se mostrarán alertas relacionadas con los pedidos.
- Se podrán añadir observaciones en los pedidos para su seguimiento.
- Que puedan incluirse datos de contacto de los proveedores.
- Que pueda calcularse el volumen de mercancía que va en curso hacia la sección.
- Se podrá buscar por números de pedido, pedidos clientes y referencias.
- Para la animación del seguimiento, se incluirá un resumen de la información de seguimiento de las secciones.
- Se mostrará únicamente la información de los pedidos vivos.

3.1.2. Requisitos no funcionales

Estos requisitos se encargan de la parte que es independiente al comportamiento.

- La información será almacenada en los servidores de la empresa.
- La comunicación por email será únicamente con los datos de los proveedores facilitados por la empresa.
- Se desarrollará en una interfaz moderna no siendo necesario la vista en móviles.
- Se almacenarán los ficheros LPREs descargados en una carpeta del sistema.

3.2. Casos de uso

A continuación, se incluyen los diferentes actores de la aplicación, así como sus casos de uso. En el apartado 5 incluiremos los diagramas de secuencia para dos de ellos. Actualizar información del proveedor realizada por el actor Usuario registrado y Actualizar Pedidos, realizado por el sistema al obtener la información de los ficheros LPRE.

Debido a que tanto la información de la existencia de tiendas, proveedores y pedidos viene por los informes que recibe la aplicación, será su responsabilidad la creación de los mismo dejando únicamente a los usuarios la posibilidad de actualizar información de estos.

El caso de uso de crear alertas será el encargado principal de la lógica de la aplicación puesto que se encargará de detectarlas, así como actualizar la información del sistema.

Imagen 3. Casos de uso

La acción de actualizar pedidos por parte del usuario registrado es el principal propósito de la aplicación. En el siguiente diagrama mostraré todas las acciones que puede realizar este actor del sistema respecto a la información que obtiene de los mismos. El hecho de detallar crear estos casos de uso supone una lógica diferente que se deberá seguir para dada una de esas acciones.

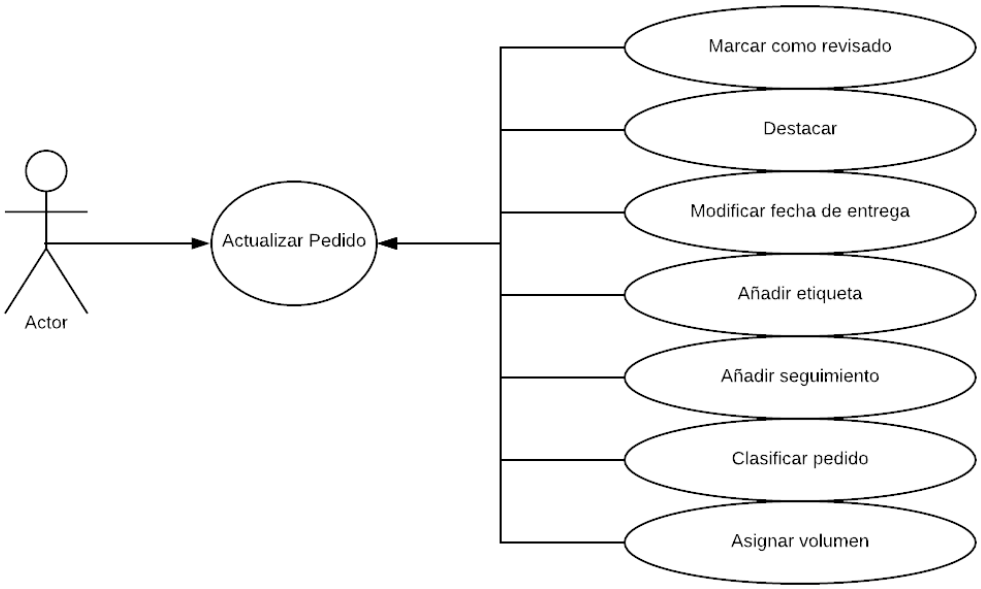

Imagen 4. Casos de uso actualizar pedido.

3.2.1. UC1 – Actualizar información del proveedor

- Breve descripción:
  - Este caso de uso permitirá al usuario actualizar la información referente al proveedor.
- Actores:
  - Usuario.
- Flujo de eventos:
  - El caso de uso se inicia cuando el usuario pulsa sobre el botón actualizar en la información de cualquier proveedor.
- Flujo básico:
  - El usuario pulsa el botón de actualizar.
  - Se redirigirá a la pantalla de actualización de proveedor con los datos de este.
  - El usuario aceptará los cambios.
- Flujo alternativo:
  - Si el usuario modifica el franco del proveedor, se revisarán los pedidos vivos para comprobar si llegan a franco creando las alertas correspondientes.
- Precondiciones:
  - El proveedor debe existir en el sistema.
- Postcondiciones:
  - Los datos serán actualizados en el sistema.

3.2.2. UC2 – Crear alertas

- Breve descripción:
  - En este caso, la aplicación comprobará si dispone de nueva información para actualizar su estado.
- Actores:
  - Sistema.
- Flujo básico:
  - Se comprobará el estado de la cuenta de correo para ver si se dispone de nueva información.
  - Descargará el informe y extraerá la información de este.
  - Actualizará el estado de la base de datos.
- Flujo alternativo:
  - Comprobará si la tienda existe, si no, la creará.
  - Comprobará si los proveedores existen, sino los creará.
  - Comprobará si los pedidos existen, si no, los creará.
  - Comprobará los pedidos en la base de datos respecto de los recibidos en el informe, si no existen los actualizará dejando de estar en curso.
- Precondiciones:
  - Se debe haber encontrado un correo como no leído en la cuenta de la aplicación con un informe adjunto, se revisarán únicamente los correos enviados desde la de reportes de la región.
  - La fecha de este informe debe ser posterior a la actual en la que se encuentra actualizada la aplicación.
- Postcondiciones:
  - Los datos serán creados o actualizados en el sistema.

3.2.3. UC3 – Buscar pedidos

- Breve descripción:
  - El usuario introducirá un número de pedido en el buscador.
- Actores:
  - Usuario.
- Flujo básico:
  - Se buscará el número de pedido dentro de la base de datos.
- Flujo alternativo:
  - Si el pedido no existe, se procederá a buscar en aquellos pedidos que tengan un número de pedido cliente.
- Precondiciones:
  - El pedido debe existir en la base de datos.
  - El número de pedido debe tener una longitud de 9 caracteres.
- Postcondiciones:
  - La aplicación redirigirá a una pantalla de datos de la aplicación.

3.2.4. UC3 – Visualizar próximas entregas

- Breve descripción:
  - El usuario seleccionará la información de su sección.
- Actores:
  - Usuario.

- Flujo básico:
  - Se buscará la información de la sección seleccionada, así como sus pedidos vivos.
- Flujo alternativo:
  - Ninguno.
- Precondiciones:
  - El usuario deberá pertenecer a una sección.
  - La sección deberá tener pedidos en curso.
- Postcondiciones:
  - La aplicación redirigirá a la pantalla de detalle de la sección. En ella cargará las gráficas con los datos de los pedidos en curso. Se incluirán el número de pedidos, el número de unidades en curso, el importe y el volumen de estos. Todos estos datos irán organizados por días para las próximas 2 semanas.

4. Análisis del problema

4.1. Diagrama de clases

En el siguiente diagrama de clases aparecen representadas las relaciones entre las diferentes entidades de la aplicación. Este diagrama es el punto de arranque al a hora de codificar la aplicación. Por simplificar el tamaño del diagrama se han obviado sus métodos dejando únicamente visibles sus atributos.

Para la representación de la aplicación se han desarrollado dos subsistemas de datos. El primero es el principal de la aplicación en el que se muestran todas las entidades participantes en el funcionamiento de la aplicación. El segundo diagrama corresponde únicamente a una estructura en la que se almacenará la información de los informes, así como las líneas de este. Este no será almacenado en la base de datos ya que contendría información redundante del primero.

4.1.1. Diagrama del subsistema LaBella

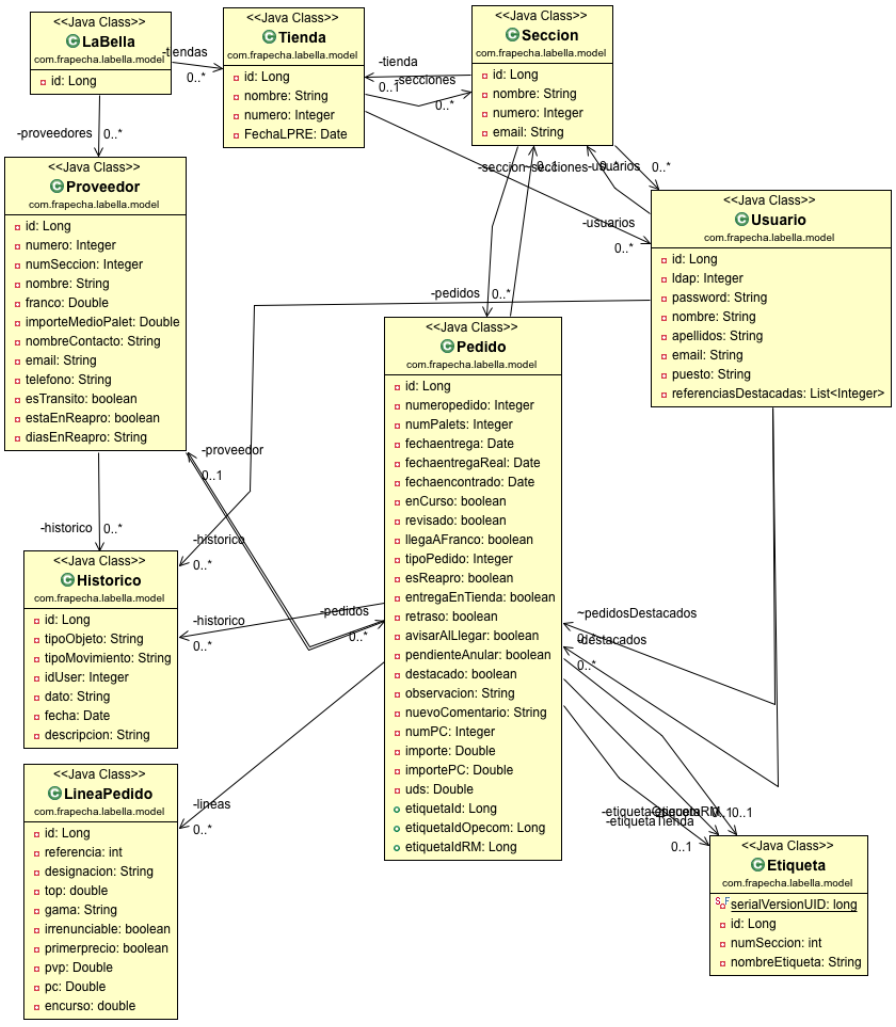

Imagen 5.Diagrama de clases

LaBella es la entidad principal de la aplicación puesto que es la que se encarga de que existan los proveedores y tiendas en el subsistema. Los procesos de actualización del sistema la utilizan para comprobar si la tienda que se pretende actualizar existe en el sistema o si debiera crear una nueva. Lo mismo ocurre con los proveedores.

La clase Tienda representa el concepto de tienda física, la cual se compone de secciones comerciales. Estas secciones son las entidades a las que se les asocian pedidos y usuarios. Es de esta manera como se determina que un usuario pueda únicamente ver los pedidos de las secciones a las que pertenece. Existen 14 secciones dentro de una tienda de las cuales 13 tendrán pedidos y la última será la sección/departamento de logística.

Un usuario puede pertenecer a más de una sección por lo que podrá gestionar los pedidos de aquellos departamentos en los que esté relacionado. Este dispondrá de una información mínima para poder iniciar sesión.

Los proveedores son las entidades que permiten tener la información de comunicación con los fabricantes de los pedidos. Entre esta información están los números de teléfono, direcciones de email y personas de contacto. La relación de los proveedores con los pedidos nos permite de un rápido vistazo ver toda la mercancía que viene en curso de un proveedor.

La entidad más importante de la aplicación es la de Pedido. Esta es la que contiene toda la información de la mercancía que viene hacia la tienda, fechas de entrega, volúmenes, unidades, importes... Sobre esta información es sobre la que es posible detectar alertas. Cada artículo se refleja con la entidad LineaPedido, la cual viene a representar toda la información que es posible obtener de un fichero LPRE con algunas diferencias como son las relaciones, importes etc.

Gracias a estructurar la información de esta manera, ya es posible indicar el volumen de mercancía que va a entrar por recepción, unidades para reponer y demás información detectada en la fase de análisis, así como las alertas.

Por último, se le pueden asignar etiquetas a los pedidos. Estas entidades permiten agrupar a los mismos para tener una visión en conjunto de pedidos de diferentes secciones, por ejemplo, en una operación comercial. O conocer aquellos pedidos que utilizan diferentes flujos logísticos, como RM o pedido cliente.

Todo movimiento que se realice en un pedido o en un proveedor quedará registrado en un histórico para poder monitorizar las acciones que se han realizado sobre los mismos, así como la actividad de los usuarios.

4.1.2. Diagrama del subsistema Carga de informes.

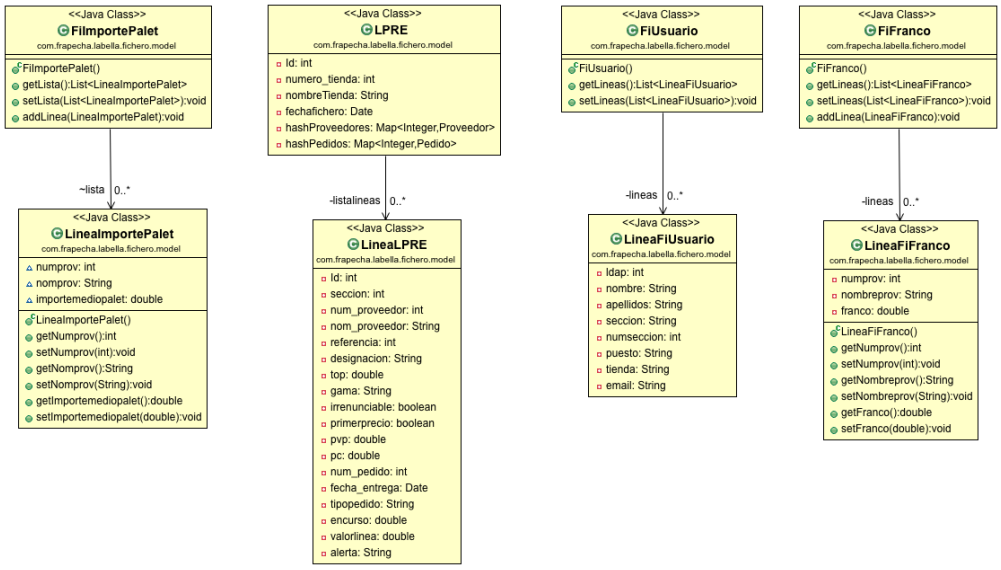

Imagen 6.Diagrama de clases subsistema carga

Estas estructuras son utilizadas únicamente para realizar cargas masivas de datos evitando así la introducción manual de los mismos. Esta información simplifica enormemente la detección de alertas e información al sistema puesto que existen más de 800 proveedores. Además, incluye la carga de usuarios por parte de ficheros Excel por parte de recursos humanos.

Las entidades FImportePalet, LPRE,FIUsuario y FIFranco representan el fichero Excel en como tal que va a ser utilizado y contempla información como la fecha de creación del mismo para poder compararla con la información que tienen la aplicación.

Las entidades LineaImportePlaet, LineaLPRE, LineaFIUsuario y LineaFIFranco representan la información de cada una de las líneas de esos ficheros Excel.

Realizar las cargas de datos iniciales de esta manera permite reducir enormemente el tiempo de utilizar los formularios, ya que dar de alta a todos los proveedores de la compañía o a todos los usuarios de una tienda sería una enorme tarea.

El fichero FIUsuario será el encargado de la creación de usuarios a una tienda, así como de asociarlos a una sección de la misma.

Los informes FImportePalet nos permiten asociar un importe promedio a un proveedor para determinar el número de palets que tiene un. Esto nos dará una idea aproximada del volumen de mercancía esperada, así como minimizar el impacto al no haber realizado la revisión de un pedido.

Los informes FIFranco nos permite asociar el franco que tiene un proveedor, de esta manera seremos capaces de detectar en la actualización del sistema si un pedido llega a ese importe y por lo tanto será enviado, o si por el contrario quedará en el sistema esperando que se realice alguna gestión con él.

Finalmente tenemos la entidad LPRE, la cual representa el fichero de compras a proveedor. Esta entidad es la que se encarga de volcar la información de los ficheros Excel y hacer la transformación a pedidos y proveedores dentro del sistema de LaBella. Esta entidad es utilizada cada vez que se encuentra un nuevo correo que contenga un fichero de compras a proveedor.

4.2. Diagramas de secuencia

A continuación, mostraremos algunos de los diagramas de secuencia para mostrar la ejecución de diferentes operaciones de los casos de usos anteriormente descritos. En ellos, se mostrarán los diferentes componentes que intervienen y como se realiza comunicación entre ellos a partir de métodos y sus respuestas.

4.2.1. Secuencia actualización pedidos de proveedor

El primer caso se trata de la actualización de los datos de un proveedor. El controlador captura la petición de actualización con sus datos y este delega en un componente de la capa de servicios. Este, delegará en la capa de persistencia la recuperación de los datos. Tras devolvérselos a la capa de negocio, comprobará los datos recibidos respecto a los que posee en la base de datos. En caso de haber diferencias, invocará a otro componente encargado de la recuperación de la información de los pedidos correspondientes a ese proveedor. Este nuevamente, invocará nuevamente a la capa de persistencia para recuperar a los mismos.

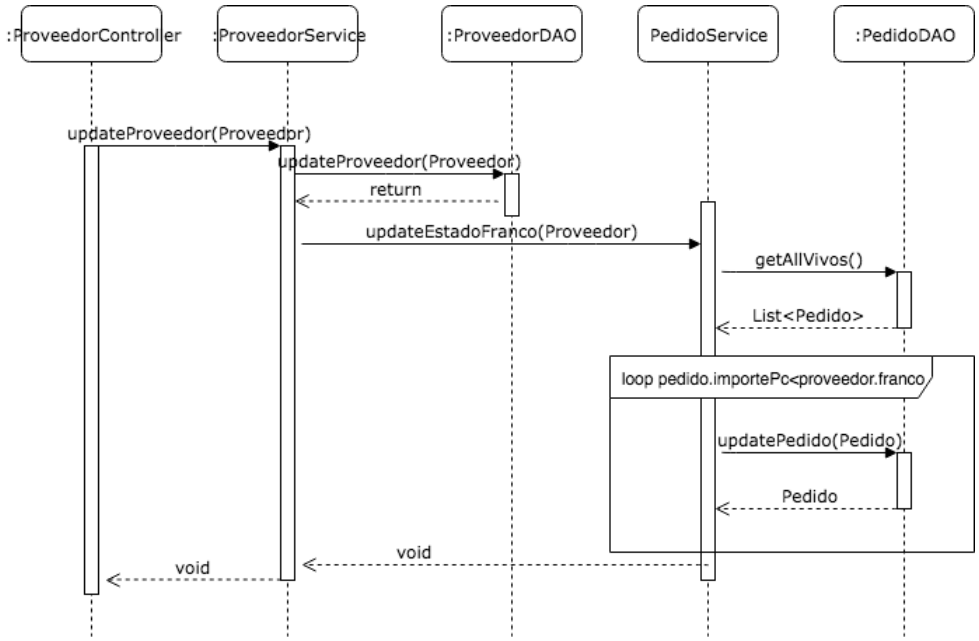

Imagen 7.Diagrama de secuencia actualización pedidos de proveedor

4.2.2. Secuencia actualización datos de usuario

El siguiente diagrama muestra el proceso de actualización de datos de un usuario en el que este es asignado a una nueva sección. Esta operación es transaccional puesto que se requiere que tanto el usuario y la sección sean conscientes de la asignación por parte de ambos.

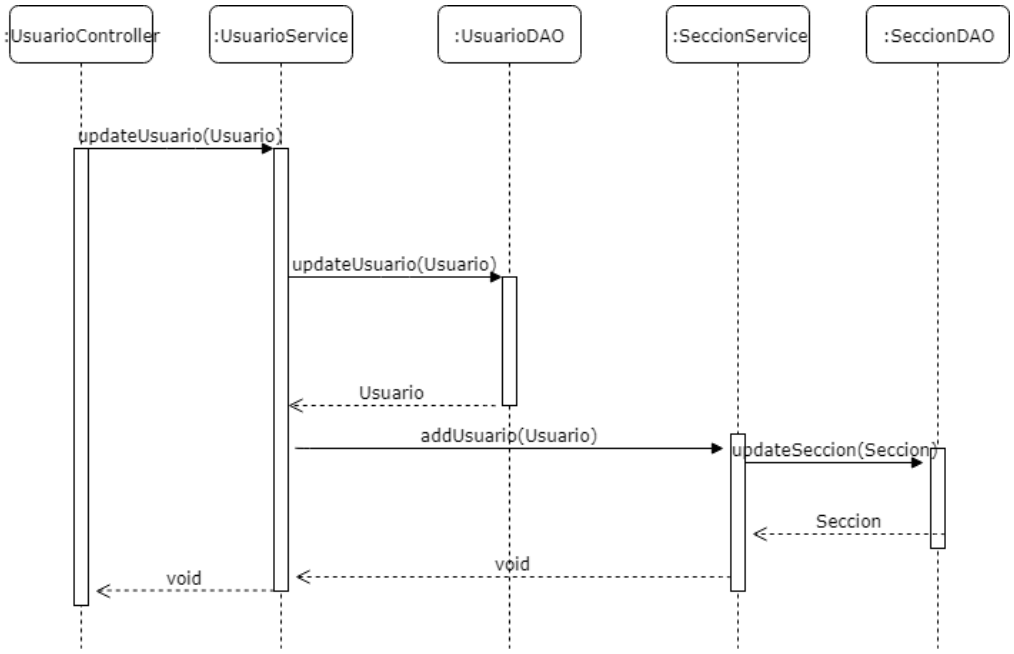

Imagen 8.Secuencia actualización usuario

4.2.3. Secuencia actualización pedido

Finalmente tenemos la actualización de un pedido proveedor por parte del usuario en la aplicación. Todos los datos introducidos en el formulario se enviarán por los diferentes componentes de las capas del sistema hasta devolver la respuesta al controlador.

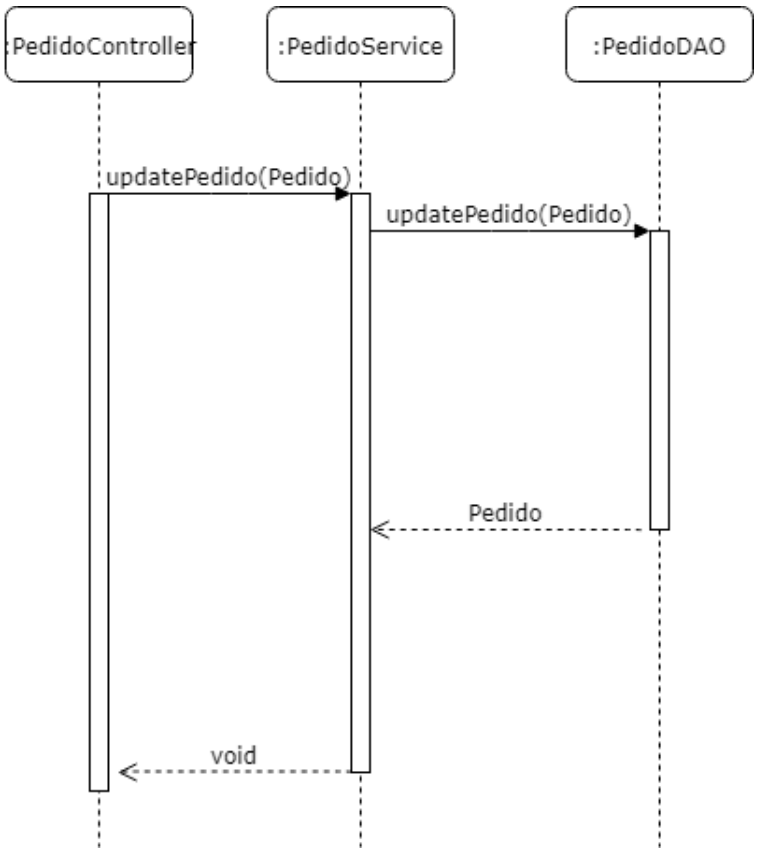

Imagen 9.Secuencia actualización pedido

4.3. Bocetos de interfaces de usuario

La pantalla principal muestra la situación global de la tienda por secciones en cuanto a pedidos revisados. Esta información se actualiza cada vez que la aplicación recibe un nuevo informe y actualiza el estado de estos, añadiendo nuevos, eliminando antiguos y mostrando número y porcentaje de revisión de estos.

En la parte superior se muestra una barra de navegación con los diferentes apartados que puede consultar el usuario. En ella se incluyen las alertas, secciones a las que pertenece el usuario, proveedores de esas secciones, pedidos vivos o en curso hacia la tienda y un apartado para crear y consultar las etiquetas que se pueden asociar a los pedidos. También muestra el nombre de la aplicación, el cual conducirá a la página principal y el usuario conectado, con sus opciones de logout e información del mismo y un buscador de pedidos y referencias.

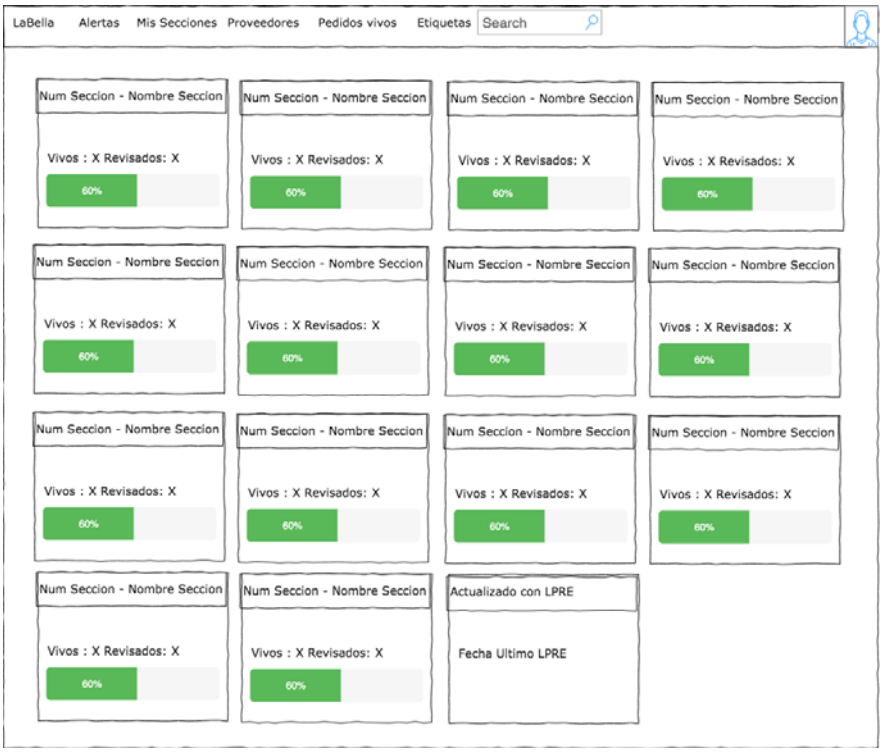

Imagen 10.Mockup pantalla principal

El siguiente wireframe es el referente al apartado de alteras, en él se aprecian el tipo de alertas que se pueden asociar a un pedido, esta información se detallará en el apartado de manual del usuario.

Imagen 11.Mockup pantalla alertas

El último wireframe es el referente a la revisión de un pedido. Fue diseñado y presentado veces hasta conseguir el diseño más aceptado por los usuarios. Este es el formulario que se utilizará más veces dentro de la aplicación.

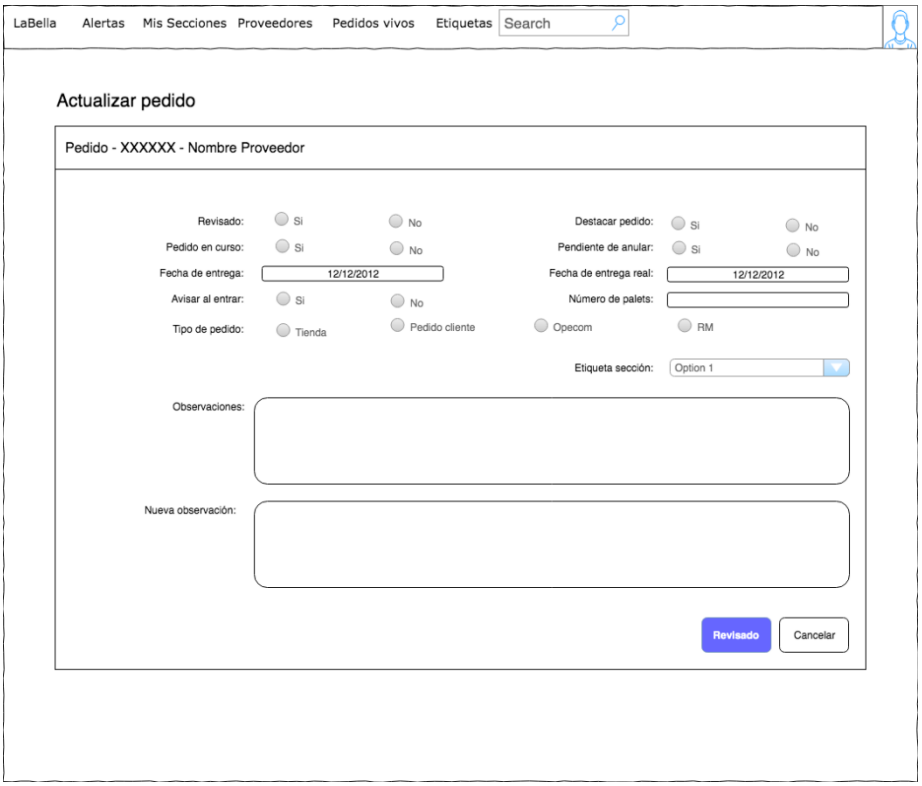

Imagen 12. Mockup pantalla actualizar pedidos

5. Diseño de la solución

En este capítulo abordaré la arquitectura de software y de equipos para diseñar la solución y ofrecer una visión sobré la distribución de esta.

5.1. Distribución de equipos

El diseño siguiente sería el equivalente para la aplicación en producción. En el centro del diseño se encontraría el servidor, el cual contendría la aplicación y el servidor web. También es quien ofrece el servicio Api REST.

Este servidor se conectaría a un servidor en varios momentos. Inicialmente para mantenerse actualizado, puesto que la información se deposita en un correo diario con un fichero Excel adjunto. Además, utilizaría este mismo servidor de correo para mandar informes, e informar acerca de alertas.

En otro equipo se encontraría una base de datos a la que se conectaría el servidor para gestionar la información.

Finalmente, los clientes interactúan con la aplicación mediante un navegador web o un cliente REST.

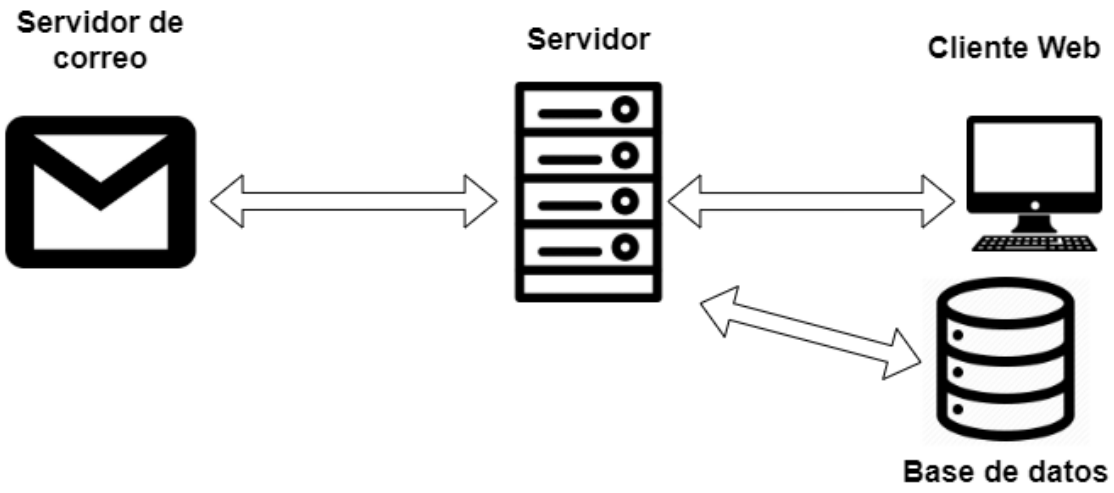

Imagen 13. Distribución de equipos

5.2. Arquitectura de software

Después de analizar los requisitos del software, comenzaremos a con la parte de diseño y posteriormente la de implementación. Utilizaremos el patrón de diseño MVC.

En esta aplicación se utiliza el patrón MVC, Modelo Vista Controlador. Con una arquitectura en 3 capas.

Arquitectura de una aplicación web

Normalmente se suelen dividir los aspectos funcionales de las aplicaciones en N-Capas, en este caso lo haremos en 3.

Estas capas son:

Presentación: Encargada de manejar y gestionar las peticiones HTTP por parte del cliente y generar la respuesta de la misma.

Negocio: Aquí se encuentra todo lo relacionado con la lógica de la aplicación y los flujos de trabajo, además contendrá las estructuras de datos del dominio.

Acceso a datos: Esta capa es la responsable de la persistencia de los datos en la base de datos. Se encarga de la construcción de las consultas y asila las estructuras de datos del dominio de los detalles de la base de datos.

Este patrón de diseño presenta una serie de beneficios como son la independencia entre módulos, lo cual permite substituir deferentes implementaciones de una misma capa sin afectar al resto.

Existe una dependencia entre las capas que indica el flujo de ejecución yendo desde la capa superior, pasando por las capas intermedias hasta la capa inferior.

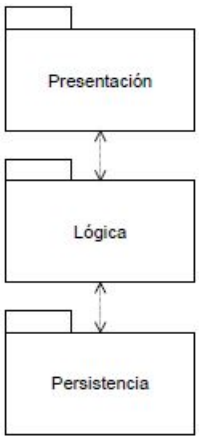

Imagen 14.Capas de la aplicación

El módulo del frontend está formado por los siguientes paquetes:

- Controladores

Contiene los diferentes controladores que se encargarán de capturar las peticiones y devolver las respuestas.

El módulo backend está formado por la siguiente estructura de archivos y carpetas, todas ellas cuelgan dentro del módulo en la jerarquía de carpetas src/main/java.

- DAO

Contiene los DAO de la capa de persistencia, en ellos se extiende la funcionalidad de Spring Data, principalmente JPA y Criteria. Estos serán accesibles únicamente por la capa de servicio.

- Model

Contiene las entidades de dominio de la aplicación. Estas se implementan siguiendo el modelo enriquecido que contendrá anotaciones de la interfaz de JPA para la creación y definición de la base de datos.

- Security

Clases relacionadas con la seguridad de la aplicación.

- Service

Contiene las interfaces e implementaciones de la capa de servicio que exponen, mediante operaciones, la lógica de negocio del backend a la capa de presentación. Todas las operaciones de los servicios del backend deben estar securizadas con anotaciones de Spring Security. Las anotaciones se encuentran en la interfaz del servicio.

6. Implementación

En esta fase del proyecto se realizará la implementación del mismo. Debido a los diferentes frameworks y herramientas utilizadas, nos extenderemos en el arranque del proyecto y en como estas herramientas simplifican enormemente este proceso.

6.1. Arranque del proyecto

Las dos partes fundamentales del arranque de un proyecto son Spring Boot y Spring ROO. Estas herramientas permiten, por un lado, la generación de la estructura básica del proyecto con sus dependencias y por otro, la creación de un CRUD completamente funcional.

6.1.1. Spring Boot

El proyecto de Spring Boot dispone de un asistente para el arranque de proyectos. Este se encuentra en la página del mismo en la parte de Initializer (http://start.spring.io). En menos de 5 minutos dispondremos de un proyecto Maven que importaremos a nuestro IDE y nos servirá como punto de partida. Desde este asistente indicaremos una serie de datos y dependencias que serán incluidas en el proyecto.

Las dependencias que seleccionaremos de arraque son:

- Spring Boot DevTools

Esta dependencia nos proporciona una serie de utilidades como son el reinicio del proyecto en el servidor, LiveReload y HotSwapping. Estas funciones nos permiten no tener que reiniciar por completo el servidor por pequeñas modificaciones en el código.

- Spring Web Starter

Incorpora las dependencias de Spring MVC e incorpora las mismas para tener Apache Tomcat como un contenedor embebido de la aplicación.

- Spring Session

Estas dependencias proporcionan una API para poder gestionar y acceder fácilmente a la información del usuario de la sesión.

- Templates Engines

Spring Boot utiliza a Thymeleaf como gestor de plantillas. En nuestro caso utilizaremos Apache Tiles. Este, al no ser soportado por Boot debe ser configurado independientemente, por lo cual, no aparecerá nada en el fichero application.properties que proporciona Boot, el cual es el que permite la rápida configuración del proyecto.

- Spring Security

Incluyendo estas dependencias dispondremos de unas utilidades para el control de acceso. De esta manera podremos controlar que acciones pueden ser o no ejecutadas por los diferentes usuarios.

- Spring Data JPA

Estas dependencias nos permiten utilizar el proyecto de Spring, Spring Data JPA. Con estas librerías podremos abstraernos casi completamente de la base de datos puesto que el acceso a los datos lo realizaremos definiendo firmas en las interfaces de los repositorios. Este se encargará de crear la implementación en tiempo de compilación para acceder a los datos.

- MySQL Driver

Incluye el driver de MySQL para poder conectarnos a nuestra base de datos.

- JDBC API

Este paquete permite conectarse a una base de datos, consultarla o actualizarla utilizando SQL. De esta manera se logra la independencia del proveedor de base de datos.

- Java Mail Sender

Es una API que facilita el envío y recepción de emails desde el código. A través de protocolos SMTP, POP3 e IMAP.

- Spring Boot Admin(Server)

Una vez seleccionadas todas las dependencias que serán necesarias para nuestro proyecto, lo generamos y obtendremos un proyecto Maven que podremos importar en nuestro IDE.

6.1.2. Spring Roo

Para el arranque del proyecto utilizaremos Spring Roo. Nuestro objetivo con esta herramienta es definir todo el modelo de datos, una versión inicial de la capa de servicios, datos y presentación. Roo nos permite generar un crud totalmente funcional que modificaremos posteriormente para adaptarlo a nuestro proyecto. De esta manera y junto a Spring Boot, habremos reducido el tiempo de configuración y desarrollo considerablemente y podremos centrarnos exclusivamente en la ampliación del proyecto para desarrollar la funcionalidad requerida. Esta parte del proyecto será explicada con especial detalle ya que esta herramienta de Spring no es especialmente conocida y para mí ha sido un gran descubrimiento debido su capacidad de generar esa funcionalidad básica pasando por todas las capas y a la inclusión de AspectJ.

Para inicializar Spring Roo accedemos dentro de la página de Spring al apartado de Projects y dentro de el al apartado de proyectos de la comunidad.

Una vez descargado crearemos una carpeta para poder alojar los ficheros que serán generados por Roo y que posteriormente incorporaremos en nuestro proyecto.

Para iniciar Roo únicamente necesitamos ejecutar el script roo.sh situado dentro de la carpeta bin. Con ello accederemos a la pantalla principal de Roo en la que podremos introducir comandos en función de la definición de nuestro proyecto.

```
MacBook-Pro-de-Francisco:roo thepuar$ sh ../../../software/spring-roo-1.3.2.RELEASE\
2/bin/roo.sh
    ____  ____  ____
   / __ \/ __ \/ __ \
  / /_/ / / / / / / /
 / _, _/ /_/ / /_/ /
/_/ |_|\____/\____/    1.3.2.RELEASE [rev 8387857]

Welcome to Spring Roo. For assistance press TAB or type "hint" then hit ENTER.
roo>
```

Ya podemos comenzar a definir nuestro proyecto. Según vamos introduciendo ordenes en Roo, este se va encargando de ir generando y ampliando los ficheros en función de la configuración introducida.

Inicialmente definimos la ruta del paquete padre así como la versión de Java que vamos a utilizar.

```
roo> project --topLevelPackage com.frapecha.labella --java 7
Created ROOT/pom.xml
Created SRC_MAIN_RESOURCES
Created SRC_MAIN_RESOURCES/log4j.properties
Created SPRING_CONFIG_ROOT
Created SPRING_CONFIG_ROOT/applicationContext.xml
```

Como podemos observar, se han generado los ficheros de la imagen en la carpeta del proyecto. Cabe destacar que, puesto que Roo utiliza AspectJ, nos habrá generado un pom.xml con las dependencias necesarias para poder utilizarlas y un plugin de compilación para poder compilar el proyecto. Todas estas dependencias las deberemos añadir al pom de nuestro proyecto.

Dependencias

```
  <!-- Spring Roo annotations -->
    <dependency>
        <groupId>org.springframework.roo</groupId>
        <artifactId>org.springframework.roo.annotations</artifactId>
        <type>pom</type>
    </dependency>

    <!-- AspectJ dependencies -->
    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjrt</artifactId>
    </dependency>
```

Plugin de compilación

```
<!-- AspectJ plugins -->
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>aspectj-maven-plugin</artifactId>
  <version>${aspectj.plugin.version}</version>
  <configuration>
    <source>${java.version}</source>
    <target>${java.version}</target>
    <Xlint>ignore</Xlint>
    <complianceLevel>${java.version}</complianceLevel>
    <encoding>UTF-8</encoding>
```

```
  </configuration>
  <executions>
    <execution>
      <phase>process-sources</phase>
      <goals>
<goal>compile</goal>
<goal>test-compile</goal>
      </goals>
    </execution>
  </executions>
  <dependencies>
    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjtools</artifactId>
      <version>${aspectj.version}</version>
    </dependency>
  </dependencies>
</plugin>
```

A continuación, como será la base de datos y que ORM queremos utilizar:

```
roo> jpa setup --provider HIBERNATE --database MYSQL
Created SPRING_CONFIG_ROOT/database.properties
Please update your database details in src/main/resources/META-
INF/spring/database.properties.
Updated SPRING_CONFIG_ROOT/applicationContext.xml
Created SRC_MAIN_RESOURCES/META-INF/persistence.xml
Updated ROOT/pom.xml [added dependencies
mysql:mysql-connector-java:5.1.18,
 org.hibernate:hibernate-core:4.3.6.Final,
 org.hibernate:hibernate-entitymanager:4.3.6.Final,
org.hibernate.javax.persistence:hibernate-jpa-2.1-api:1.0.0.Final,
 commons-collections:commons-collections:3.2.1,
 org.hibernate:hibernate-validator:4.3.2.Final,
 javax.validation:validation-api:1.0.0.GA, javax.transaction:jta:1.1,
org.springframework:spring-jdbc:${spring.version}, org.springframework:spring-
orm:${spring.version},
 commons-pool:commons-pool:1.5.6, commons-dbcp:commons-dbcp:1.4]
```

En nuestro caso utilizaremos Hibernate con JPA. Roo ya se ha encargado de añadir la configuración necesaria en el fichero persistence.xml y applicationContext.xml. Puesto que estamos en la versión 1.3.2 Release de Spring Roo, este aun no desarrollado para Spring Boot, por lo que más adelante configuraremos Boot desde el fichero application.properties para solucionar este problema.

Por otro parte, con este comando Roo se ha encargado también de añadir las dependencias necesarias en nuestro pom.xml relacionadas con Hibernate y Jpa. Podría ser que aparecieran algunas repetidas con nuestro pom actual, ya que este ha sido generado con el inicializador de Spring Boot, por lo que deberemos evitar repeticiones.

Con todo lo anterior ya tenemos el proyecto preparado para la creación de las entidades. Para no extenderme mucho en esta parte, realizare un ejemplo únicamente con las entidades Sección y Tienda obviando algunos tipos de relaciones.

Introducimos la siguiente orden:

```
roo> entity jpa --class ~.entity.Tienda --serializable true --equals --sequenceName
tienda_sequence --table tienda --activeRecord false

Created SRC_MAIN_JAVA/com/frapecha/labella/entity
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda.java
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_Serializable.aj
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_ToString.aj
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_Equals.aj
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_Jpa_Entity.aj
```

Con ella le estamos indicando a Roo que queremos que nos cree una entidad con soporte para jpa en el paquete com.frapecha.labella.entity llamada Tienda la cual será Serializable ya que será transmitida a la base de datos para ser persistida. También le indicamos como queremos que se llame el generador de secuencia de la base de datos que generara el Id que será la clave primaria de la entidad. Y por último le indicamos el nombre de la base de datos.

Como podemos observar, Roo no ha generado únicamente el archivo Seccion.java que cabría esperar sino que ha creado una serie de ficheros con la terminación .aj. Estos son los ficheros de AspectJ o Aspect Java. Aspect Java es la clave de la programación orientada a aspectos de Java, ya que intenta formalizar y representar de forma concisa los elementos que son transversales a todo el sistema.

Seccion_Roo_Serializable.aj

Este fichero contiene la información necesaria para poder serializar la entidad, en este caso implementa la interfaz serializable y crea el atributo serialVersionUID necesario para la misma.

Seccion_Roo_Jpa_Entity.aj

Aquí se almacena todas las declaraciones relacionadas con la base de datos, como es el nombre de la tabla, el Id, la declaración como @Entity y la Versión , este último necesario para el control de la concurrencia.

Seccion_Roo_JavaBean.aj

Este último fichero será el encargado de tener todos los getter y setters de los atributos que vayamos creando posteriormente y se añadirán aquí cuando los definamos. Tambien se encarga de redefinir los métodos equals, hashCode y toString.

Creamos el primer atributo de la entidad Tienda, en este caso es una String

```
~.entity.Tienda roo> field string --fieldName nombre --notNull --unique --column
seccion_nombre --
sizeMax 20
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda.java
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_Equals.aj
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_JavaBean.aj
```

Con esta definición hemos añadido un atributo llamado nombre en la entidad Sección que será de tipo String, el cual no podrá ser nulo y su representación en la base de datos vendrá dada por la columna seccion_nombre que tendrá una longitud máxima de 20 caracteres. Esto queda representado en el fichero Seccion.java de la siguiente manera.

```
@NotNull
@Column(name = "seccion_nombre")
@Size(max = 20)
private String nombre;
```

Con esta definición, al crear la tabla en la base de datos se añadirán las restricciones de tamaño máximo y nulidad. Si intentamos persistir alguna entidad que no cumpla con estas restricciones, JPA lanzará una excepción impidiéndonos realizar esa acción.

En el fichero Seccion_Roo_JavaBean.aj se han añadido los métodos Getter y Setter correspondientes a todos los atributos de la clase.

Continuaremos añadiendo los atributos básicos y posteriormente añadiremos las relaciones. En este caso hemos añadido el atributo número que nos servirá para identificar por número a las tiendas. Este campo es importante puesto que como veremos en la parte de lógica de la aplicación, es la base para la creación de las tiendas ya que es el campo utilizado para identificar a las mismas dentro de los ficheros LPRE.

```
~.entity.Tienda roo> field number --fieldName numero --type java.lang.Integer --notNull
--unique --column tienda_numero
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda.java
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_Equals.aj
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_JavaBean.aj
```

Por último, añadimos un campo de tipo Calendar para poder almacenar en cada una de las tiendas la fecha de su última actualización. Esta la necesitaremos para saber si estamos en la última versión de datos o debemos actualizarnos.

```
~.entity.Tienda roo> field date --fieldName fechaLPRE --type java.util.Calendar
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda.java
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_Equals.aj
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda_Roo_JavaBean.aj
```

Finalmente, creamos el constructor de la clase.

```
~.entity.Tienda roo> constructor --class ~.entity.Tienda
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Tienda.java
```

A continuación, realizamos lo mismo con la entidad Sección.

```
~.entity.Tienda roo> entity jpa --class ~.entity.Seccion --serializable true --equals --
sequenceName seccion_sequence --table seccion --activeRecord false
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion.java
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_Serializable.aj
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_ToString.aj
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_Equals.aj
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_Jpa_Entity.aj
~.entity.Seccion roo> field string --fieldName nombre --notNull --unique --column
seccion_nombre --sizeMax 20
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion.java
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_Equals.aj
```

```
Created SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_JavaBean.aj
~.entity.Seccion roo> field number --fieldName numero --type java.lang.Integer --notNull
--unique --column seccion_numero
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion.java
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_Equals.aj
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_JavaBean.aj
~.entity.Seccion roo> field string --fieldName email --notNull --unique --column
seccion_email --sizeMax 50
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion.java
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_Equals.aj
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_JavaBean.aj
```

En este momento vamos a crear la primera relación entre clases. Por simplicidad mantendremos siempre que sea posible en las relaciones ManyToOne la referencia en la clase débil. Esto nos hará ahorrar espacio en la base de datos y simplificará las tareas de mantenimiento.

Como regla general, configuraremos todas las relaciones con Fetch Lazy. Esto, al contrario que Eager, nos permitirá una mayor velocidad de respuesta por parte de la base de datos, ya que nos traeremos únicamente los datos que queramos, sin traer sus colecciones de la misma. Esto podría generar muchísima carga puesto que la aplicación está pensada para almacenar millares de registros.

```
~.entity.Seccion roo> field reference --fieldName tienda --type ~.entity.Tienda --
cardinality MANY_TO_ONE --fetch LAZY --notNull --joinColumnName seccion_tienda --
referencedColumnName id
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion.java
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_Equals.aj
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion_Roo_JavaBean.aj
```

Por último definimos su constructor.

```
~.entity.Seccion roo> constructor --class ~.entity.Seccion
Updated SRC_MAIN_JAVA/com/frapecha/labella/entity/Seccion.java
```

En este punto ya tenemos la capa del modelo de datos generada. Ahora ha llegado el momento de generar el resto de las capas y gracias a Roo es posible hacerlo con un par de comandos.

Comenzamos a generar la capa de Persistencia.

```
~.entity.Seccion roo> repository jpa --interface ~.dao.SeccionDAO --entity
~.entity.Seccion
Created SRC_MAIN_JAVA/com/frapecha/labella/dao
Created SRC_MAIN_JAVA/com/frapecha/labella/dao/SeccionDAO.java
Created SPRING_CONFIG_ROOT/applicationContext-jpa.xml
Updated ROOT/pom.xml [added dependency org.springframework.data:spring-data-
jpa:1.4.3.RELEASE]
```

Como se puede observar, Roo ha vuelto a generar 2 ficheros relacionados con la persistencia de esa entidad.

SeccionDAO.java

```
@RooJpaRepository(domainType = Seccion.class)
public interface SeccionDAO {
}
```

SeccionDAO_Roo_Jpa_Repository.aj

```
privileged aspect SeccionDAO_Roo_Jpa_Repository {

    declare parents: SeccionDAO extends JpaRepository<Seccion, Long>;

    declare parents: SeccionDAO extends JpaSpecificationExecutor<Seccion>;

    declare @type: SeccionDAO: @Repository;

}
```

En el fichero SeccionDAO vemos como únicamente se ha definido una interfaz sin ningún método, sin embargo, en el fichero de AspectJ se ha definido que esa interfaz debe extender JpaRepository. Esta pertenece a las librerías del proyecto Spring Data Jpa, la cual nos proporciona los métodos básicos para tener un CRUD funcional. También es la encargada de realizar las implementaciones a partir de la definición de las firmas en los métodos.

Buscar una sección por nombre o número sería tan sencillo como lo siguiente:

```
    public Seccion findByNumero(Integer numero);
    public Seccion findByNombre(String nombre);
```

Incluso podemos realizar operaciones típicas de SQL como seria la búsqueda por And u Or, y todo, sin realizar ninguna línea de implementación.

```
    public Seccion findByNombreAndNumero(String nombre, Integer numero);
    public Seccion findByNombreOrNumero(String nombre, Integer numero);
```

Para continuar, realizamos el mismo paso, pero con la entidad Tienda.

```
~.entity.Seccion roo> repository jpa --interface ~.dao.TiendaDAO --entity
~.entity.Tienda
Created SRC_MAIN_JAVA/com/frapecha/labella/dao/TiendaDAO.java
Created SRC_MAIN_JAVA/com/frapecha/labella/dao/TiendaDAO_Roo_Jpa_Repository.aj
```

Ya tenemos completada la capa de Acceso a datos, ahora procederíamos a generar la capa de Servicio. Esta también es muy sencilla y basta un único comando para indicar donde queremos que se genere la misma. Deberá disponer de una interfaz y una implementación para cada entidad creada.

Lanzando la orden:

```
~.entity.Seccion roo> service all --interfacePackage ~.service.basico --classPackage
~.service.basico
Created SRC_MAIN_JAVA/com/frapecha/labella/service/basico
Created SRC_MAIN_JAVA/com/frapecha/labella/service/basico/SeccionService.java
Created SRC_MAIN_JAVA/com/frapecha/labella/service/basico/SeccionServiceImpl.java
Created SRC_MAIN_JAVA/com/frapecha/labella/service/basico/TiendaService.java
Created SRC_MAIN_JAVA/com/frapecha/labella/service/basico/TiendaServiceImpl.java
Created SRC_MAIN_JAVA/com/frapecha/labella/service/basico/SeccionService_Roo_Service.aj
Created
SRC_MAIN_JAVA/com/frapecha/labella/service/basico/SeccionServiceImpl_Roo_Service.aj
```

```
Created SRC_MAIN_JAVA/com/frapecha/labella/service/basico/TiendaService_Roo_Service.aj
Created
SRC_MAIN_JAVA/com/frapecha/labella/service/basico/TiendaServiceImpl_Roo_Service.aj
```

Nos genera una capa de Servicios comunicada con nuestra capa de Persistencia creándonos una interfaz básica en el fichero TiendaService.java.

```
@RooService(domainTypes = { com.frapecha.labella.entity.Tienda.class })
public interface TiendaService {
}
```

TiendaServiceImpl.java

```
public class TiendaServiceImpl implements TiendaService {
}
```

Y la implementación que se inyectara en la compilación en el fichero anterior. Todos estos diferentes ficheros *.aj enlazados a un fichero*.java pueden ser incluidos mediando operaciones de 'Push in', los cuales eliminarán la parte de AspectJ.

```
privileged aspect TiendaServiceImpl_Roo_Service {

    declare @type: TiendaServiceImpl: @Service;

    declare @type: TiendaServiceImpl: @Transactional;

    @Autowired
    TiendaDAO TiendaServiceImpl.tiendaDAO;

    public long TiendaServiceImpl.countAllTiendas() {
        return tiendaDAO.count();
    }

    public void TiendaServiceImpl.deleteTienda(Tienda tienda) {
        tiendaDAO.delete(tienda);
    }

    public Tienda TiendaServiceImpl.findTienda(Long id) {
        return tiendaDAO.findOne(id);
    }

    public List<Tienda> TiendaServiceImpl.findAllTiendas() {
        return tiendaDAO.findAll();
    }

    public List<Tienda> TiendaServiceImpl.findTiendaEntries(int firstResult, int
maxResults) {
        return tiendaDAO.findAll(new
org.springframework.data.domain.PageRequest(firstResult / maxResults,
maxResults)).getContent();
    }

    public void TiendaServiceImpl.saveTienda(Tienda tienda) {
        tiendaDAO.save(tienda);
    }

    public Tienda TiendaServiceImpl.updateTienda(Tienda tienda) {
        return tiendaDAO.save(tienda);
    }

}
```

Únicamente nos falta la capa de Control y la de la vista, la cual, también podemos generar automáticamente mediante Roo, para ello únicamente debemos configurar el proyecto para MVC con la orden mvc setup.

```
~.entity.Seccion roo> web mvc setup
Created ROOT/src/main/webapp/WEB-INF/spring
Created ROOT/src/main/webapp/WEB-INF/spring/webmvc-config.xml
Created ROOT/src/main/webapp/WEB-INF/web.xml
Updated ROOT/src/main/webapp/WEB-INF/spring/webmvc-config.xml
Created ROOT/src/main/webapp/images
Created ROOT/src/main/webapp/images/add.png
Created ROOT/src/main/webapp/images/banner-graphic.png
Created ROOT/src/main/webapp/images/create.png
Created ROOT/src/main/webapp/images/delete.png
Created ROOT/src/main/webapp/images/favicon.ico
Created ROOT/src/main/webapp/images/list.png
Created ROOT/src/main/webapp/images/resultset_first.png
Created ROOT/src/main/webapp/images/resultset_last.png
Created ROOT/src/main/webapp/images/resultset_next.png
Created ROOT/src/main/webapp/images/resultset_previous.png
Created ROOT/src/main/webapp/images/show.png
Created ROOT/src/main/webapp/images/springsource-logo.png
Created ROOT/src/main/webapp/images/update.png
Created ROOT/src/main/webapp/styles
Created ROOT/src/main/webapp/styles/alt.css
Created ROOT/src/main/webapp/styles/standard.css
Created ROOT/src/main/webapp/WEB-INF/classes
Created ROOT/src/main/webapp/WEB-INF/classes/alt.properties
Created ROOT/src/main/webapp/WEB-INF/classes/standard.properties
Created ROOT/src/main/webapp/WEB-INF/layouts
Created ROOT/src/main/webapp/WEB-INF/layouts/default.jspx
Created ROOT/src/main/webapp/WEB-INF/layouts/layouts.xml
Created ROOT/src/main/webapp/WEB-INF/views
Created ROOT/src/main/webapp/WEB-INF/views/header.jspx
Created ROOT/src/main/webapp/WEB-INF/views/menu.jspx
Created ROOT/src/main/webapp/WEB-INF/views/footer.jspx
Created ROOT/src/main/webapp/WEB-INF/views/views.xml
Created ROOT/src/main/webapp/WEB-INF/views/dataAccessFailure.jspx
Created ROOT/src/main/webapp/WEB-INF/views/index-template.jspx
Created ROOT/src/main/webapp/WEB-INF/views/index.jspx
Created ROOT/src/main/webapp/WEB-INF/views/resourceNotFound.jspx
Created ROOT/src/main/webapp/WEB-INF/views/uncaughtException.jspx
Created ROOT/src/main/webapp/WEB-INF/tags/form
Created ROOT/src/main/webapp/WEB-INF/tags/form/create.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/dependency.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/find.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/list.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/show.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/update.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/checkbox.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/column.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/datetime.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/display.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/editor.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/input.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/reference.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/select.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/simple.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/table.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/form/fields/textarea.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/menu
Created ROOT/src/main/webapp/WEB-INF/tags/menu/category.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/menu/item.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/menu/menu.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/util
Created ROOT/src/main/webapp/WEB-INF/tags/util/language.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/util/load-scripts.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/util/pagination.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/util/panel.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/util/placeholder.tagx
Created ROOT/src/main/webapp/WEB-INF/tags/util/theme.tagx
Created ROOT/src/main/webapp/WEB-INF/i18n
Created ROOT/src/main/webapp/WEB-INF/i18n/messages.properties
Created ROOT/src/main/webapp/images/en.png
Updated ROOT/src/main/webapp/WEB-INF/i18n/application.properties
Updated ROOT/src/main/webapp/WEB-INF/web.xml
Updated ROOT/pom.xml [added dependencies org.springframework:spring-
webmvc:${spring.version}, org.springframework.webflow:spring-js-resources:2.2.1.RELEASE,
commons-digester:commons-digester:2.1, commons-fileupload:commons-fileupload:1.2.2,
javax.servlet.jsp.jstl:jstl-api:1.2, org.glassfish.web:jstl-impl:1.2, javax.el:el-
```

```
api:2.2, joda-time:joda-time:1.6, javax.servlet.jsp:jsp-api:2.1, commons-codec:commons-
codec:1.5; updated project type to war; added property 'tomcat.version' = '7.0.63';
added plugin org.apache.tomcat.maven:tomcat7-maven-plugin:2.2; added plugin
org.mortbay.jetty:jetty-maven-plugin:8.1.4.v20120524; added dependency
org.apache.tiles:tiles-jsp:2.2.2]
Updated SRC_MAIN_WEBAPP/WEB-INF/views/footer.jspx
```

Esta orden nos ha generado muchísimos ficheros que, aunque muchos no utilizaremos, nos deja el proyecto preparado para realizar la generación de los controladores y las vistas con la orden "web mvc all –package ~.controller"

```
~.entity.Seccion roo> web mvc all --package ~.controller
Created SRC_MAIN_JAVA/com/frapecha/labella/controller
Created SRC_MAIN_JAVA/com/frapecha/labella/controller/TiendaController.java
Created SRC_MAIN_JAVA/com/frapecha/labella/controller/SeccionController.java
Updated SRC_MAIN_WEBAPP/WEB-INF/spring/webmvc-config.xml
Created
SRC_MAIN_JAVA/com/frapecha/labella/controller/ApplicationConversionServiceFactoryBean.ja
va
Created SRC_MAIN_WEBAPP/WEB-INF/views/seccions
Created SRC_MAIN_WEBAPP/WEB-INF/views/seccions/views.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/views/seccions/views.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/i18n/application.properties
Created SRC_MAIN_WEBAPP/WEB-INF/views/tiendas
Created SRC_MAIN_WEBAPP/WEB-INF/views/tiendas/views.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/views/tiendas/views.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/i18n/application.properties
Created
SRC_MAIN_JAVA/com/frapecha/labella/controller/SeccionController_Roo_Controller.aj
Created SRC_MAIN_WEBAPP/WEB-INF/views/seccions/list.jspx
Created SRC_MAIN_WEBAPP/WEB-INF/views/seccions/show.jspx
Created SRC_MAIN_WEBAPP/WEB-INF/views/seccions/create.jspx
Updated SRC_MAIN_WEBAPP/WEB-INF/views/menu.jspx
Created SRC_MAIN_WEBAPP/WEB-INF/views/seccions/update.jspx
Created SRC_MAIN_JAVA/com/frapecha/labella/controller/TiendaController_Roo_Controller.aj
Created SRC_MAIN_WEBAPP/WEB-INF/views/tiendas/list.jspx
Created SRC_MAIN_WEBAPP/WEB-INF/views/tiendas/show.jspx
Created SRC_MAIN_WEBAPP/WEB-INF/views/tiendas/create.jspx
Created SRC_MAIN_WEBAPP/WEB-INF/views/tiendas/update.jspx
Created
SRC_MAIN_JAVA/com/frapecha/labella/controller/ApplicationConversionServiceFactoryBean_Ro
o_ConversionService.aj
```

Como se puede apreciar, se han creado las Clases TiendaController.java y TiendaController_Roo_Controller.aj.

En este momento ya tenemos una aplicación con un CRUD completamente funcional desde la que podemos crear, actualizar, borrar y asociar los datos relacionados con las Tiendas y las Secciones.

```
privileged aspect TiendaController_Roo_Controller {

    @Autowired
    TiendaService TiendaController.tiendaService;

    @RequestMapping(method = RequestMethod.POST, produces = "text/html")
    public String TiendaController.create(@Valid Tienda tienda, BindingResult
bindingResult, Model uiModel, HttpServletRequest httpServletRequest) {
        if (bindingResult.hasErrors()) {
            populateEditForm(uiModel, tienda);
            return "tiendas/create";
        }
        uiModel.asMap().clear();
        tiendaService.saveTienda(tienda);
        return "redirect:/tiendas/" + encodeUrlPathSegment(tienda.getId().toString(),
httpServletRequest);
    }

    @RequestMapping(value = "/{id}", produces = "text/html")
    public String TiendaController.show(@PathVariable("id") Long id, Model uiModel) {
```

```
        addDateTimeFormatPatterns(uiModel);
        uiModel.addAttribute("tienda", tiendaService.findTienda(id));
        uiModel.addAttribute("itemId", id);
        return "tiendas/show";
    }
.
}
```

6.1.3. Vista

Aunque Spring Roo nos ha dejado una aplicación completamente funcional, lo normal es adaptar el frontend a la petición del cliente. En este caso, descartaremos toda la vista generada por Roo para crear nuestro propio diseño.

Para ello, necesitaremos construir una estructura para las páginas web como vimos en los wireframes y utilizaremos un gestor de plantillas que, en este caso será Apache Tiles.

Este es fácil de integrar con Spring MVC debido a que únicamente hay que configurar un componente que se encargue de redirigir las peticiones a esas vistas.

Las aplicaciones web suelen definir sus páginas con un layout o estructura de bloques de información homogénea. Habitualmente, una cabecera, un menú, un bloque principal y un pie. Excepto el bloque principal que suele incluir el contenido específico de la página, por lo general el resto de las zonas son iguales o cambian muy poco en todas las páginas.

Las plantillas de página deben definir el contenido completo de la página: bloques de cabecera, menú, cuerpo, pie, etc. Duplicar los contenidos de los bloques comunes conlleva un coste considerable de desarrollo y mantenimiento.

Como solución a esta problemática, se emplean mecanismos de gestión de layouts. Un layout permite separar los distintos bloques que componen una página. De esta forma, las secciones comunes (cabecera, menú, pie, etc) se definen una única vez y cada página solo es responsable de generar el bloque principal con el contenido específico de la página.

Al construir las páginas de una aplicación, se diferencias tres tipos de elementos:

- Definición de layouts: Dependiendo de las características de la aplicación, puede existir un layout o varios. Un grupo de páginas pueden tener una estructura de bloques diferente a otro grupo de páginas, cada grupo con su layout relacionado. En la aplicación lo utilizamos para diferenciar las cabeceras que pertenecen a los usuarios y las que pertenecen al administrador.

- Plantillas para bloques comunes: Cabecera, menú, pie, etc.

- El cuerpo para cada una de las páginas de la aplicación.

Apache Tiles contiene un motor de plantillas de página para poder ser utilizado en entornos web. Los sistemas de plantillas permiten no duplicar código dentro de

las diferentes páginas y reemplazar únicamente las partes del documento en función de la petición por parte del cliente.

Spring Boot utiliza otros sistemas de plantillas como Thymeleaf y no permite la configuración del mismo.

Para incluir Apache Tiles dentro del proyecto, deberemos incluir las dependencias dentro del pom.xml del proyecto.

```
    <!-- Apache tiles -->
    <dependency>
            <groupId>org.apache.tiles</groupId>
            <artifactId>tiles-core</artifactId>
            <version>${tiles.version}</version>
    </dependency>

    <dependency>
            <groupId>org.apache.tiles</groupId>
            <artifactId>tiles-jsp</artifactId>
            <version>${tiles.version}</version>
    </dependency>

    <dependency>
            <groupId>org.apache.tiles</groupId>
            <artifactId>tiles-api</artifactId>
            <version>${tiles.version}</version>
    </dependency>
 <dependency>
     <groupId>org.apache.tiles</groupId>
     <artifactId>tiles-servlet</artifactId>
     <version>${tiles.version}</version>
 </dependency>
```

Las plantillas se encuentran dentro de la carpeta /src/main/webapp /WEB-INF/tiles. Estas contienen la estructura sobre la que se cargaran los diferentes componentes que se insertaran dentro de los documentos devueltos en la respuesta.

Los diferentes contenidos se encuentran en la ruta /srt/main/webapp /WEB-INF/pages/bodys.

Los controladores de Spring MVC se encargan de capturas las peticiones y redirigir a estos documentos. Toda la configuración de que debe ser devuelto en cada petición se encuentra en un fichero llamado tiles.xml que contiene el siguiente formato. En el se define el esqueleto de las diferentes partes de la vista y que será devuelto en la respuesta de la petición.

```
<tiles-definition>
  <!-- Base Definition -->
   <definition name="base-definition"
       template="/WEB-INF/tiles/layouts/defaultLayout.jsp">
       <put-attribute name="title" value="" />
       <put-attribute name="header" value="" />
       <put-attribute name="body" value="" />
       <put-attribute name="footer" value="" />
   </definition>

   <!-- Secciones -->
   <definition name="secciones" extends="base-definition">
       <put-attribute name="title" value="Secciones" />
       <put-attribute name="header" value="/WEB-INF/tiles/template/ adminNavBar.jsp"/>
       <put-attribute name="body" value="/WEB-INF/pages/bodys/secciones.jsp" />
   </definition>
   <!-- Seccion -->
   <definition name="seccion" extends="base-definition">
```

```
        <put-attribute name="title" value="Seccion" />
        <put-attribute name="header" value="/WEB-INF/tiles/template/adminNavBar.jsp"/>
        <put-attribute name="body" value="/WEB-INF/pages/bodys/detail/ seccion.jsp" />
    </definition>
<tiles-definition>
```

Este documento permite configurar las respuestas como páginas según las peticiones recibidas. La respuesta es fraccionada en diferentes partes que serán incluidas en la respuesta hacia el cliente. De esta manera, no necesitamos incluir la barra de navegación del usuario en todos los documentos, sino que únicamente la crearemos una vez y la incluiremos en las respuestas de las diferentes peticiones.

Por último, queda configurar un nuevo componente que se encargara de configurarse en el arranque del proyecto para integrar Apache Tiles con las peticiones recibidas.

```
@EnableWebMvc
@Configuration
public class TilesConfiguration  {

        @Bean
        public UrlBasedViewResolver tilesViewResolver() {

                UrlBasedViewResolver tilesViewResolver = new UrlBasedViewResolver();
                tilesViewResolver.setViewClass(TilesView.class);
                return tilesViewResolver;
        }

        @Bean
        public TilesConfigurer tilesConfigurer() {

                TilesConfigurer tconf = new TilesConfigurer();
                tconf.setDefinitions(new String[] { "/WEB-INF/tiles/tiles.xml" });
                return tconf;
        }
}
```

Para la creación del contenido dinámico de las pantallas utilizaremos un componente nativo de Java EE.

JavaServer Pages Tag Library son un conjunto de etiquetas que se utilizan para realizar operaciones en la vista como bucles, condiciones, redirecciones... dentro de la vista. Este componente extiende las JavaServer Pages(Jsp)

Para poder utilizarlos debemos incluir las dependencias en el pom.xml del proyecto.

```
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
</dependency>
```

Una vez hecho, añadiremos el tag en el namespace de las plantillas de Apache Tiles para que lo tengamos en todos los documentos que crearemos.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

Ahora podemos obtener el valor de las variables que mandemos a la vista utilizando la nomenclatura ${seccion.numero}. Esto llama al método toString() para imprimir el atributo numero de la sección.

Estas variables deberán ser enviadas desde el controlador que captura la petición hacia la vista en la que será procesada para generar la respuesta.

jQuery

Es una librería multiplataforma de JavaScript que permite simplificar la manera de interactuar con los elementos del documento HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con el lado servidor haciendo uso de llamadas AJAX.

Añadimos sus dependencias en el pom.xml del proyecto

```
<dependency>
	<groupId>org.webjars.bower</groupId>
	<artifactId>jquery</artifactId>
</dependency>
```

Una vez disponemos de las librerías en el proyecto, es necesario usarlas en cada una de las páginas, por lo que incluiremos la siguiente etiqueta dentro de las plantillas de Apache Tiles. De esta manera, dispondremos las librerías de jQuery en todas las pantallas de la aplicación. Esto es necesario puesto que alguna parte de la funcionalidad de Bootstrap requiere de las mismas para poder funcionar correctamente.

```
<script type="text/javascript" charset="utf-8"
src="https://code.jquery.com/jquery-1.12.4.js"
data-th-src="@{/webjars/jquery/1.12.4/dist/jquery.js}"></script>
```

Bootstrap

Es un conjunto de herramientas de código abierto para el diseño de páginas web. Ofrece una serie de estilos y clases CSS predeterminadas para cada uno de los componentes HTML que permite que el diseño de aplicaciones web sea sencillo, intuitivo y reutilizable.

Incluiremos la dependencia de Bootstrap en el pom del proyecto.

```
<dependency>
	<groupId>org.webjars.bower</groupId>
	<artifactId>bootstrap</artifactId>
</dependency>
```

Y añadiremos la plantilla tanto las ubicaciones de los webjas de los estilos CSS como del fichero Javascript.

```
<link rel="stylesheet" type="text/css"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.css"
data-th-href="@{/webjars/bootstrap/3.3.7/dist/css/bootstrap.css}" />
<script type="text/javascript"
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.js"
data-th-src="@{/webjars/bootstrap/3.3.7/dist/js/bootstrap.js}"></script>
```

La utilización de Bootstrap es muy sencilla puesto que contiene muchos estilos CSS definidos para las diferentes etiquetas HTML que utilicemos. Se puede modificar su apariencia generando una estructura de etiquetas con clases de estilos según el contenedor que queramos diseñar.

6.1.4. Uso de la capa de servicios desde un controlador

```
@Controller
@SessionAttributes("usuario")
public class SeccionController {

	@Autowired
	SeccionValidator seccionValidator;

	@Autowired
	SeccionService seccionService;

	@RequestMapping("/useccion/info/{id}")
	public ModelAndView gotoInfoUSeccion(@PathVariable(value = "id") Long id) {
	ModelAndView mav = new ModelAndView("useccion/info");
	Seccion laseccion = seccionService.findById(id);

	mav.addObject("seccion", laseccion);
	return mav;
	}

	@RequestMapping(value = "/useccion/update/{id}", method = RequestMethod.POST)
	public ModelAndView UpdateSeccion(@Valid @ModelAttribute("seccion") Seccion
laseccion, @PathVariable(value = "id") Long id, BindingResult result, Model model) {
	ModelAndView mav = new ModelAndView();
	Seccion secciondb = seccionService.findById(id);
	mav.addObject("seccionInfo", secciondb);
	seccionValidator.validate(laseccion, result);
	if (result.hasErrors()) {
	    mav.setViewName("useccion/update");
	    mav.addObject(result.getModel());
	    return mav;
	    } else {
	    secciondb.setEmail(laseccion.getEmail());
	    seccionService.saveSeccion(secciondb);
	    mav.setViewName("redirect:/useccion/info/" + secciondb.getId());
	    return mav;
	    }
    }
```

Estas acciones en la mayoría de los casos generan información o datos que se envían a la capa de presentación que a su vez se presentará al usuario.

6.1.5. Lógica de negocio

Se utilizará Sping Framework ya que permite reutilizar objetos de negocio y acceso a datos. La lógica de la aplicación se implementa en servicios de la aplicación y el conjunto de estos constituyen la capa de servicios dentro de la capa de negocio.

Crearemos una interfaz y una implementación de servicio por cada entidad persistente (todo esto ha sido generado por Spring Roo). Este servicio contiene las operaciones básicas para esa entidad, también llamado CRUD(Create-Read-Update-Delete). Todas estas clases se encuentran dentro del paquete com.frapecha.labella.service.

Para la entidad Seccion crearíamos la interfaz SeccionService y la implementación del servicio SeccionServiceImpl que gestionará todas las operaciones que tengan que ver con las mismas.

```
public interface SeccionService {

	public Seccion findById(Long id);

	public void saveSeccion(Seccion Seccion);

	public void deleteSeccion(Seccion Seccion);

	public List<Seccion> findAll();

	public long countAllSeccions();

}

@Service
@Transactional
public class SeccionServiceImpl implements SeccionService{

	@Autowired
	SeccionDAO SeccionDAO;

	@Override
	public Seccion findById(Long id) {
		Optional<Seccion> opSeccion =  SeccionDAO.findById(id);
		if(opSeccion.isPresent())
			return opSeccion.get();
		else return null;
	}
	public List<Seccion> findAll(){
		return SeccionDAO.findAll();
	}
	public void saveSeccion(Seccion Seccion) {
		SeccionDAO.save(Seccion);
	}
	public void deleteSeccion(Seccion Seccion) {
		SeccionDAO.delete(Seccion);
	}
	@Override
	public long countAllSeccions() {
		return SeccionDAO.count();
	}

}
```

Cada servicio debe especificar su interfaz, que utilizara la capa de control para invocar a los servicios y todas las operaciones relacionadas con sus entidades deberán pasar por aquí. De esta manera obtenemos una separación entre capas no permitiendo que la Vista pueda acceder a datos sin pasar por el servicio.

Como se puede apreciar en la implementación, se utiliza la anotación @Autowired. Está indica al contenedor de Spring que deberá encargarse de la gestión de la misma.

El interfaz definido por la capa de servicio permite que la cada de presentación pueda ejecutar acciones del sistema aislándola completamente de su ámbito,

La capa de servicios delega en la capa de acceso a datos las tareas relacionadas con la persistencia de datos y lo hacen a través de las interfaces de la capa de repositorios. De esta forma se garantiza que se podrán utilizar diferentes implementaciones de los repositorios sin afectar a la capa de servicios.

6.1.6. Algoritmos desarrollados

A continuación, se detallarán los principales algoritmos utilizados para la actualización de los datos de la aplicación. Estos serán representados en pseudocódigo para agilizar su comprensión.

- **Actualización del sistema**

  - Obtención de datos por la estructura FILPRE
    - Se crean los proveedores relacionándolos con los pedidos.
  - Se comprueba si la tienda existe, sino es así se crea con sus secciones.
  - Se comprueba la información de la base de datos respecto a la estructura.
    - Se recorren los proveedores generados por la estructura FILPRE:
      - Si el proveedor existe se recorren sus pedidos.
        - Si el pedido no está en la estructura, pero si en la base de datos, se marca el pedido como no vivo.
        - Se crea un histórico con la información de la desaparición del pedido.
  - Se comprueba la información de la estructura respecto a la de la base de datos.
    - Se recorren los proveedores generados por la estructura FILPRE:
      - Si el proveedor no existe se crea.
      - Se crea un histórico con el momento de su creación
      - Se recorren sus pedidos:
        - Si los pedidos no existen se crean comprobando el franco del proveedor si lo hubiese.
        - Se crea un histórico con el momento de su creación.
  - Se actualiza la fecha del informe en la información de la tienda.

- **Actualización de un proveedor(franco)**
  - Se cargan los datos del proveedor.
  - Se comprueban los datos recibidos con los datos del proveedor.
  - Si el franco se ha modificado:
    - Se obtienen todos los pedidos en curso del proveedor.
    - Se compara si su importe es inferior al nuevo franco recibido.
      - Si el franco es inferior se marca como que no llega a franco.
    - Se crea un histórico con la actualización de los datos.

6.1.7. Base de datos.

Existen diferentes modalidades para la creación de la base de datos en función del código de las entidades o viceversa, estas son CodeFirst o DataBaseFirst. Con Roo, al incluir las anotaciones @Entity en los modelos de datos, estos le indican que se deberá representar en la base de datos. De esta manera nos olvidamos completamente de la creación por nuestra parte de esta.

Únicamente deberemos indicar en el fichero application.properties la url para la conexión de la base de datos, así como el usuario y contraseña.

Spring Data JPA incorpora unas interfaces que permiten la creación de un repositorio abstracto. Además, soporta todas las implementaciones de las operaciones de CRUD generando las operaciones SQL. Estas operaciones son fácilmente ampliables creando la firma de métodos con una sintaxis concreta en función de los atributos de las entidades a las que se refiere y las operaciones que se quieren realizar.

```
@Repository
public interface SeccionDAO extends JpaRepository<Seccion,
Long>,JpaSpecificationExecutor<Seccion>{
}
```

6.1.8. Controladores API REST y Swagger

La creación de un servicio REST consiste en generar un controlador que acepte peticiones y devuelva la información en Json o XML. En este proyecto únicamente se han realizado los servicios para poder consumir información y no para poder actualizarla. Únicamente atiende a peticiones GET.

El controlador se configurará para que la respuesta del mensaje lo genere como tipo Json ya que es el formato más habitual en servicios REST.

```
@RestController
@CrossOrigin(origins = "*",methods= {RequestMethod.GET})
@RequestMapping(value="/labellaboot/api/v1",produces=
{"application/json","application/xml"})
@Api(value="Etiqueta",description="Metodos para etiquetas",tags= {"Etiquetas"})
public class RestEtiquetaController {

	@Autowired
	EtiquetaService etiquetaService;

	@RequestMapping(value="/etiquetas", method= RequestMethod.GET)
	@ApiOperation(value="Obtiene todas las etiquetas",notes = "Obtiene todas las
etiquetas")
	public List<Etiqueta> retrieveAlletiquetas(){
		return etiquetaService.findAll();
```

```
        }
}
```

Gracias a Swagger, podemos fácilmente consumir los servicios que tenemos publicados observando la información y el formato. Esta información es muy útil para los desarrolladores puesto que tienen de una forma fácil y accesible acceder a todo lo que ofrece la aplicación.

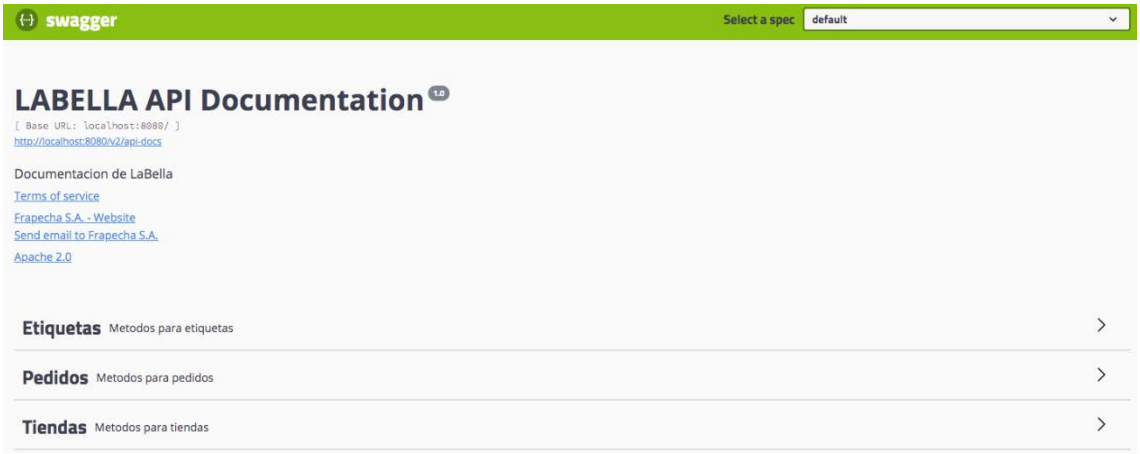

Imagen 15.Pantalla principal Swagger

Dentro de cada uno de los apartados anteriores, aparecerán los recursos publicados para cada uno de ellos. Todos ellos se corresponden con las operaciones Http (GET, POST, PUT, DELETE) disponibles por cada uno de esos recursos.

En el siguiente ejemplo se muestra la una petición GET para obtener un XML con toda la información disponible acerca de las etiquetas de los pedidos.

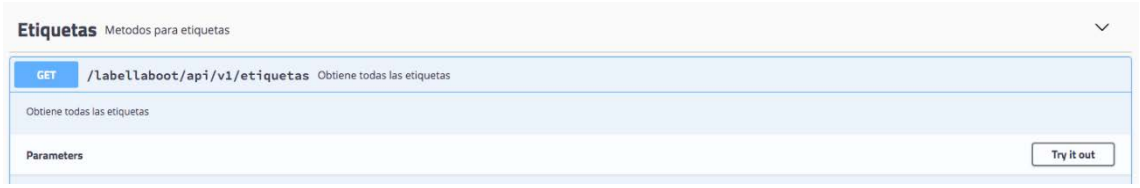

Imagen 16.Petición REST de Etiquetas

Imagen 17.Respesta petición REST Etiquetas

7. Pruebas

A continuación, realizaremos un breve vistazo a las pruebas que se han realizado sobre el proyecto para comprobar y validar su funcionamiento.

7.1. Pruebas unitarias

Para la realización de las pruebas unitarias se ha utilizado la tecnología Junit y Mockito.

Junit permite definir métodos que serán ejecutados para comprobar el resultado obtenido respecto del resultado esperado. De esta manera es más sencillo encontrar posibles errores en el desarrollo.

Debido a que las pruebas unitarias están pensadas para validar métodos y estos pueden depender de otros objetos del sistema, se utilizaran las librerías de Mockito para simular el comportamiento de estas sin hacer un uso real de ellas.

Todos los métodos que utilicen la anotación @Test y el resultado devuelto sea de tipo void serán ejecutados en la etapa de test por Junit.

Gracias a esta librería podemos introducir muy fácilmente multitud de datos para poder testear y el funcionamiento de la aplicación de forma independiente a la estructura de la aplicación final.

7.2. Pruebas realizadas

En nuestro caso, todas las pruebas las hemos realizado únicamente para el proceso de carga de datos diaria y sus clases relacionadas excluyendo el resto de los procesos del servicio y de la presentación.

Se ha seguido el mismo guion que el descrito en el apartado 6.1.6.1 acerca de la actualización del sistema.

Una vez especificado en qué consistirá el plan de pruebas y definido el resultado que debe obtenerse para cada una de las pruebas, comenzamos a ejecutarlos y a modificar la lógica del algoritmo desarrollado en caso de no obtener los resultados esperados.

7.2.1. Test lectura LPRES

```
package com.frapecha.labellaprov.test;

import static org.assertj.core.api.Assertions.assertThat;

import java.io.File;
import java.util.List;
import java.util.stream.Collectors;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.frapecha.labellaprov.model.Company;
import com.frapecha.labellaprov.model.CompanyType;
import com.frapecha.labellaprov.model.Provider;
import com.frapecha.labellaprov.model.lpre.PedidoLPRE;
import com.frapecha.labellaprov.model.lpre.ProveedorLPRE;
import com.frapecha.labellaprov.service.api.CompanyService;
import com.frapecha.labellaprov.service.api.FileReaderLpre;
import com.frapecha.labellaprov.service.api.ProviderService;

import lombok.extern.log4j.Log4j2;

@RunWith(SpringJUnit4ClassRunner.class)
@DataJpaTest
@Log4j2
public class ReadLPRETest {

	@Autowired
	FileReaderLpre fileReaderLpre;

	@Autowired
	ProviderService providerService;

	@Autowired
	CompanyService companyService;

	File file;

	@Before
	public void prepareTest() {
		file = new File("H:\\ArchivosLaBella\\2018418135214_COMPRAS_LPRE.xlsx");
	}

	//@Test
	public void readProvidersLPRE() {
Company company = companyService.initializeCompany(CompanyType.LEROYMERLIN);
List<ProveedorLPRE> proveedores = this.fileReaderLpre.getProveedoresLPRE(file);
		List<Provider> providers = proveedores.stream().map(pro -> {
Provider provider = new Provider(company, pro.getNombre(), pro.getNumero());
			return provider;
		}).collect(Collectors.toList());
		assertThat(proveedores.size() > 0).isTrue();

		providers.forEach(prov -> providerService.save(prov));
		assertThat(providerService.count()).isEqualTo(providers.size());
```

```
log.info("Proveedores en BBDD {} / Lista {}", providerService.count(),
providers.size());
	}

	//@Test
	public void readOrdersLPRE() {
		List<PedidoLPRE> orders = this.fileReaderLpre.getPedidosLPRE(file);
		assertThat(orders.size()).isNotZero();
		orders.forEach(order -> {
			assertThat(order.getLineas().size()).isNotZero();
		});
	}

	//@Test
	public void readProvidersAndOrdersLPRE() {
List<ProveedorLPRE> proveedores =
this.fileReaderLpre.getProveedoresAndPedidosLPRE(file);
		assertThat(proveedores.size()).isNotZero();
		boolean existeMasdeUno = false;
long numpedidos = proveedores.stream().filter(p-> p.getPedidos().size()>1).count();
		assertThat(numpedidos).isGreaterThan(1);
	}

}
```

Imagen 18 Resultado test lectura LPRE

7.2.2. Test actualización en base de datos

```
package com.frapecha.labellaprov.test;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.Assert.*;

import java.io.File;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.junit4.SpringRunner;

import com.frapecha.labellaprov.model.CompanyType;
import com.frapecha.labellaprov.model.Store;
import com.frapecha.labellaprov.model.lpre.LPRE;
import com.frapecha.labellaprov.service.api.CompanyService;
import com.frapecha.labellaprov.service.api.FileReaderLpre;
import com.frapecha.labellaprov.service.api.FillService;
import com.frapecha.labellaprov.service.api.OrderService;
import com.frapecha.labellaprov.service.api.ProductService;
import com.frapecha.labellaprov.service.api.ProviderService;
import com.frapecha.labellaprov.service.api.StoreService;

import lombok.extern.log4j.Log4j2;

@RunWith(SpringRunner.class)
@SpringBootTest
@Log4j2
```

```
public class FillServiceTest {

	@Autowired
	FillService fillService;

	@Autowired
	CompanyService companyService;

	@Autowired
	StoreService storeService;

	@Autowired
	FileReaderLpre fileReaderLpre;

	@Autowired
	ProviderService providerService;

	@Autowired
	OrderService orderService;

	@Autowired
	ProductService productService;

	Store store;

	File file;

	@Before
	public void initialize() {
		if (companyService.findAll().size() == 0) {
			companyService.initializeCompany(CompanyType.LEROYMERLIN);
			store = companyService.initializeStore(CompanyType.LEROYMERLIN,
"La Zenia", 63);
		}
		file = new File("H:\\ArchivosLaBella\\2018418135214_COMPRAS_LPRE.xlsx");

	}

	 @Test
	public void StoreAlltest() {
fillService.storeAll(store, fileReaderLpre.getProveedoresAndPedidosLPRE(file));
		assertThat(providerService.count()).isNotZero();
		assertThat(orderService.count()).isNotZero();
		assertThat(productService.count()).isNotZero();
		System.out.println("Providers :" + providerService.count());
		System.out.println("Orders :" + orderService.count());
		System.out.println("Products :" + productService.count());
	}

	 @Test
	public void getNumberLpre() {
		int numero = fileReaderLpre.getNumeroTienda(file);
		assertThat(numero).isNotZero();
	}

	@Test
	public void getNameLpre() {
		String nombre = fileReaderLpre.getNombreTienda(file);
		assertThat(nombre).isNotNull();
	}

	@Test
	public void getLPRE() {

		LPRE lpre = fileReaderLpre.getLPRE(file);
		assertThat(lpre.getNombreTienda()).isNotNull();
		assertThat(lpre.getNumeroTienda()).isNotZero();
		assertThat(lpre.getProveedores()).isNotNull();
	}

}
```

Imagen 19 Resultado test carga en BBDD

8. Manual de uso

En este capítulo se detallan las diferentes utilidades que proporciona la aplicación, así como se realiza un recorrido por las diferentes pantallas indicando sus elementos funcionales.

8.1. Acceso a la aplicación

Accedemos a la aplicación desde la pantalla de login con nuestro usuario y contraseña.

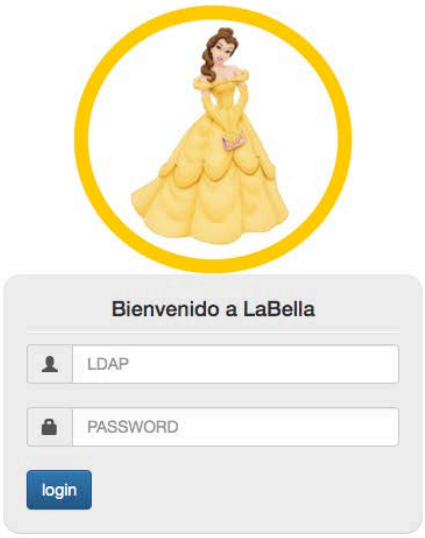

Imagen 20.Login aplicación

Una vez dentro de la misma, nos aparecerá una pantalla principal que contiene un resumen del estado del seguimiento de los pedidos de la tienda indicando el número de pedidos en curso hacia la tienda, así como cuántos de ellos han sido revisados. Las barras de porcentaje cambiarán de color en función del porcentaje seguimiento que se ha realizado.

En la parte superior disponemos de una barra de navegación en la que se muestran los enlaces a las Alertas, las secciones a las que pertenece el usuario, los proveedores de estas, los pedidos en curso actuales y las etiquetas para los pedidos.

También aparece un buscador por el que se puede buscar por número de pedido, referencia o número de pedido cliente.

Como información adicional, aparece la fecha a la que esta actualizada la aplicación con el último informe que encontró en el correo. Este fue descargado y utilizado para actualizar a la misma. Por último, aparece un apartado con la información de la última segmentación de stock que dispone la aplicación. Esta no es objeto de esta aplicación actualmente y será comentada como futuras ampliaciones.

8.2. Pantalla de resumen

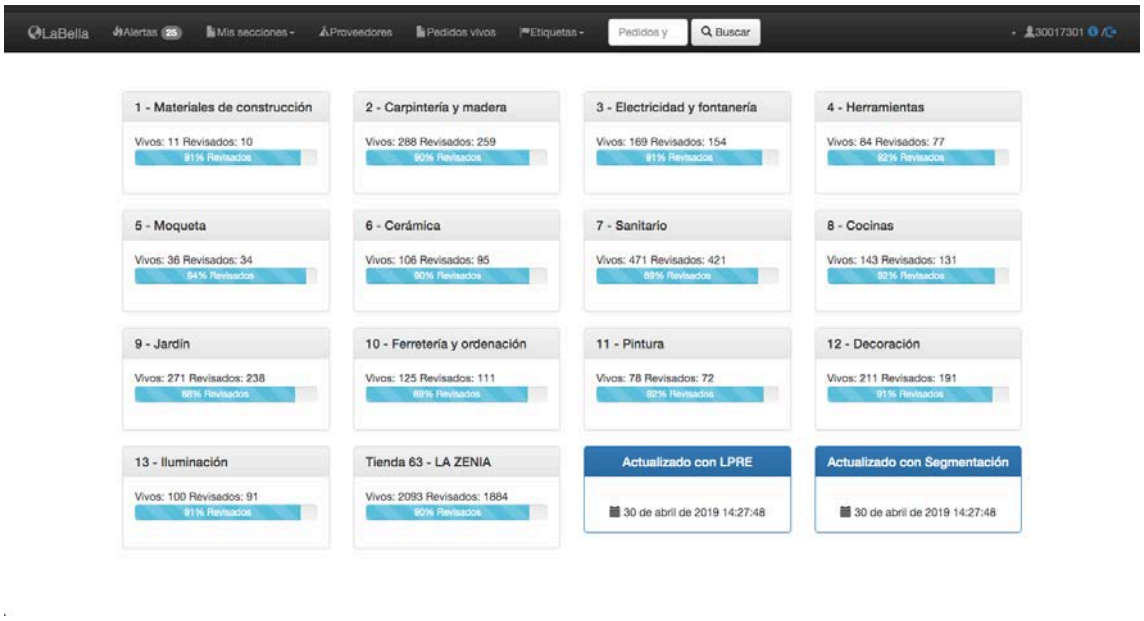

Imagen 21.Dashboard

Accederemos a la sección de alertas en la que se mostrará un listado de avisos sobre acciones que necesitan ser revisadas por el usuario. Las alertas se clasifican de la siguiente manera y tienen un tratamiento concreto.

Pedidos destacados: Estos pedidos son marcados por el usuario porque suponen una revisión posterior concreta. Esto puede ser debido a que el pedido sea realizado por un vendedor y falte información para poder tomar una decisión como podría ser cambiar el flujo logístico.

Pedidos por revisar: En este apartado aparecerán los pedidos que nunca han sido revisados. Normalmente al día suelen crearse por sección entre 15-30 pedidos, suponiendo esto más de 200 pedidos al día por tienda.

Pedidos que no llegan a franco: Los proveedores tienen tanto condicionantes en sus productos como francos en sus pedidos. Si el proveedor recibe un pedido con un montante inferior al franco negociado con la tienda, este no enviará el pedido. En muchas ocasiones los proveedores no informan de esta situación debido a la cantidad de pedidos que reciben y en la tienda no están informados de que no van a recibir ese pedido. Ese pedido puede tener mercancía para reaprovisionar la tienda que contará como mercancía en curso y no será enviada por el proveedor. Esto supone que las aplicaciones de reaprovisionamiento estén trabajando con datos erróneos y se produzcan rupturas de stock en la tienda. El tratamiento de este tipo de alertas es fundamental para tener una baja tasa de rupturas.

Pedidos para avisar al entrar: En este apartado se marcarán los pedidos a los cuales se requiere una especial atención a la hora de su entrada en la tienda. La aplicación se encargará de informar tanto a la sección como a recepción el día que la aplicación tiene marcado como real con un mensaje informativo e incluyendo los seguimientos realizados en el pedido.

Pedidos pendientes de anular: Esta función está pensada que cuando detectamos un pedido que va a ser anulado, podamos hacerlo en el sistema de la tienda y marcarlo aquí para que informe al proveedor del mismo y tenga en cuenta su desaparición de los LPRES ya que la información de los mismos siempre tiene 1 día de retraso.

8.3. Alertas en pedidos

Las alertas sobre el proveedor muestran un listado acerca de aquellos proveedores que no tienen su información acerca del franco y contacto completada.

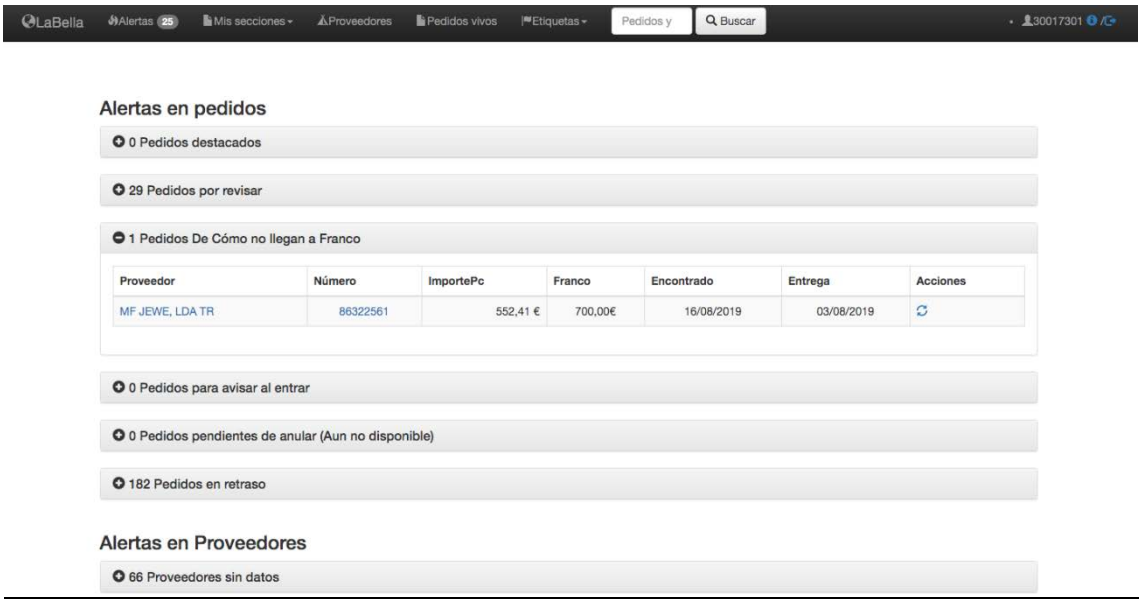

Imagen 22.Pantalla de alertas

El siguiente enlace en la barra de navegación nos mostrará un desplegable con las secciones a las que pertenece el empleado en las que si pulsamos en una aparecerá un breve resumen de la situación de los pedidos de la sección. Entre ellos aparecerá el número de pedidos revisados respecto del total, así como las etiquetas que se han asignado a los pedidos. Esto es útil para conocer el flujo que va a seguir la mercancía puesto que de los pedidos de Pedido cliente y de RM se encargan por completo otros departamentos. Los pedidos clasificados como de operaciones comerciales son útiles para tenerlos localizados fácilmente puesto que es importantísimo que la mercancía de estos esté presente en la tienda antes del comienzo de estas.

8.4. Detalles de sección

Imagen 23.Pantalla detalle de sección1

En la parte al resumen del estado de los pedidos y sus etiquetas, aparecen unas graficas de barras con la información sobre los pedidos en los próximos 14 días. Esta información se refiere al número de pedidos que pasan por tienda por día respecto del total de los pedidos para esos mismos días. También hay una pestaña con una misma grafica que indica el número de palets que se recibían en esos días. Aportar esta información es muy importante puesto que ahora dispondremos de 2 semanas para poder planificar la reposición y reorganizar los pedidos.

Un usuario asignado a una sección comercial solo podrá ver la información referente a la misma, pero un empleado asociado al departamento de logística dispondrá de la información de toda la tienda. Esto es fundamental puesto que de esta manera puede reorganizar la recepción moviendo pedidos de unas fechas a otras en función de la carga prevista para esos días. Esto ocurre constantemente en las vísperas de las operaciones comerciales puesto que se intenta que los pedidos lleguen los días más próximos al inicio de estas, lo cual puede ocasionar el colapso de la recepción.

También hay una pestaña con las unidades que vienen a tienda sabiendo las que pasan y las que no por tienda. Esta información da un detalle más concreto que la información sobre los palets, puesto que no es lo mismo recibir un palet con 7 unidades de mercancía voluminosa que otro con 1000 unidades de picking. La organización del equipo es completamente diferente y de esta manera se puede organizar mejor.

Por último, hay una pestaña con el importe de la mercancía que entra en tienda. Esto es útil únicamente para controlar el valor del Stock que va a subir a la cuenta de explotación.

En cualquiera de las gráficas, al pulsar sobre cada barra de cada día aparece un resumen de los pedidos en la parte inferior.

Imagen 24.Pantalla detalle de sección2

8.5. Gestión de proveedores

El siguiente enlace es Proveedores, este nos envía a una pantalla con el listado de proveedores asociados a las secciones a las que pertenece el empleado. Muestra la información general del mismo, así como el número de pedidos que tiene en curso. También están disponibles las acciones de información del proveedor, en que veremos la información más detallada del proveedor con todos sus pedidos en curso y la información de estos.

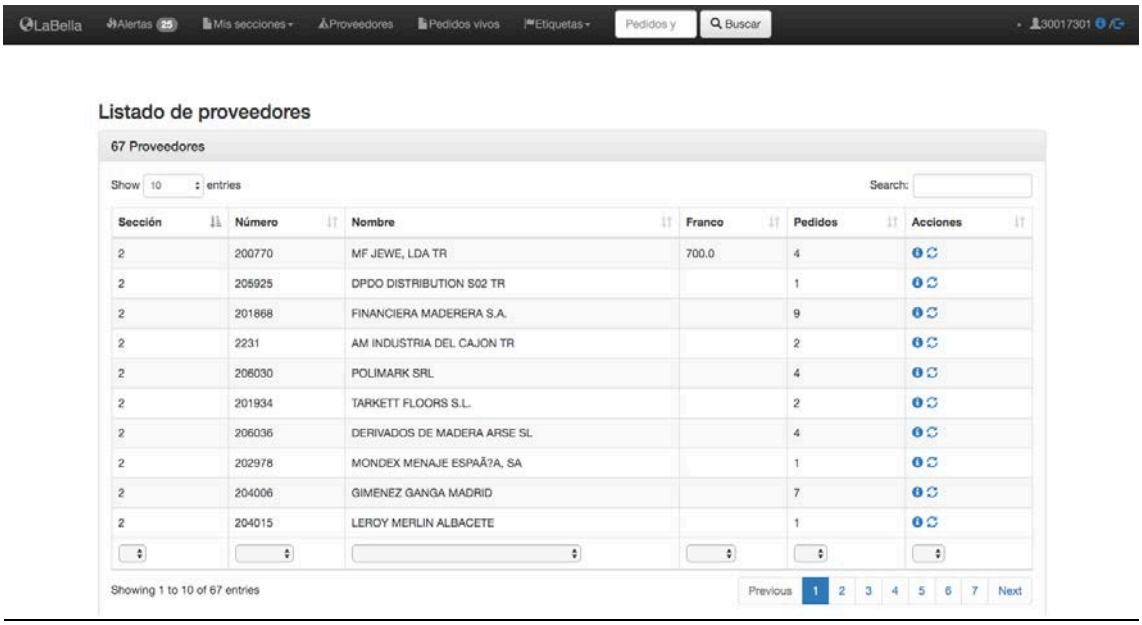

Imagen 25.Pantalla listado proveedores

Al pulsar sobre el icono de información situado en la columna acciones, accederemos a la información general detallada de ese proveedor con un listado del estado de sus pedidos en curso.

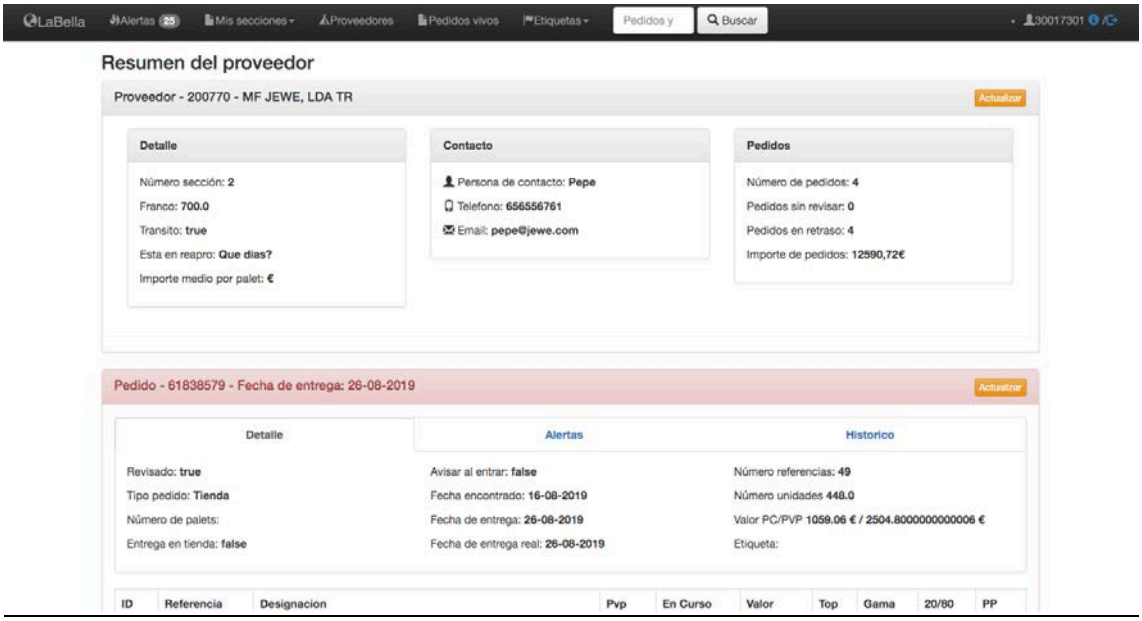

Imagen 26.Pantalla detalle proveedor

En cuanto a la actualización de los datos del proveedor, podemos actualizar la información relacionada con el franco, el importe medio de un palet, los datos de contacto, si es un proveedor de reapro o si es de tránsito.

Al modificar el franco, la aplicación buscará en los pedidos de ese proveedor, cuales están vivos y no llegan a franco. Esto lo tendrá también en cuanta para siguientes actualizaciones de datos. Si encuentra un pedido que no llega a franco, lo añadirá como alerta en la pantalla de estas.

Modificando el importe medio por palet, buscará aquellos pedidos vivos en los que no se haya especificado el número de palets, en función del precio de venta, añadirá más o menos. Esta información sirve para calcular el volumen de mercancía que entrara en la tienda. También se tendrá en cuenta en los siguientes pedidos que detecte para los proveedores que tengan este campo completado. Aun así, este dato podrá ser modificado en la pantalla de actualización de pedidos ya que, calculándolo de esta manera se hace un promedio.

El hecho de indicar si es un proveedor de reapro o de transito es debido a que, para aquellos proveedores que sean indicados como proveedores de Transito, la fecha de entrega que aparece en los informes no es real, será siempre 2 días laborales después y esto es tenido en cuenta por la aplicación. Normalmente aparece indicado en la descripción del proveedor puesto que al final de su nombre aparecen los caracteres ‘TR’, pero no es así en todos los casos. Esta diferencia de fecha será tenía en cuenta la hora de mostrar la información en las gráficas de entrega.

En el caso de actualizar la información del proveedor, se obtendrán para los pedidos de ese mismo, la posibilidad de enviar correos de confirmación, anulación, reclamación con la información del pedido y del proveedor. Esto es muy útil puesto que constantemente se tienen que enviar el mismo tipo de correo a diferentes proveedores por diferentes pedidos.

En este caso la aplicación abrirá una pestaña del navegador con un correo redactado para ese propósito. Esto es debido a que, al ser cuentas de correo corporativas, alojadas en Gmail, no es viable realizar esa configuración para cada cuenta y de esta manera, el usuario recibirá la contestación en la su cuenta personal.

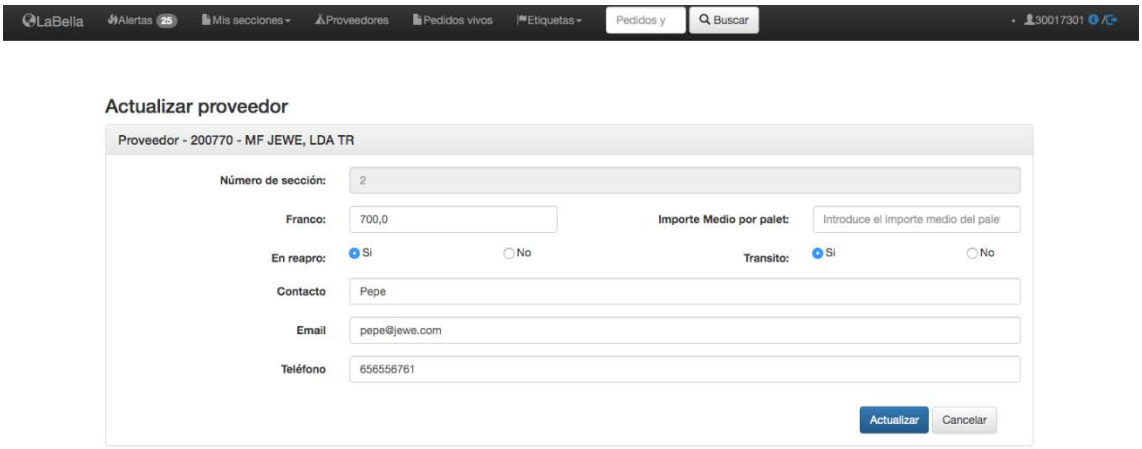

Imagen 27.Pantalla actualización de proveedor

8.6. Gestión de pedidos

El siguiente elemento de la barra de navegación es acerca de los pedidos en curso. En esta pantalla se obtendrá una lista de los pedidos proveedor en curso hacia la tienda con información acerca del tipo de pedido (Tienda, Operación comercial, RM, PC), nombre, número, etiquetas, fecha de entrega y días hasta su recepción. Sobre cada pedido, podremos consultarlo, actualizarlo, o consultar la información el proveedor con sus pedidos en curso.

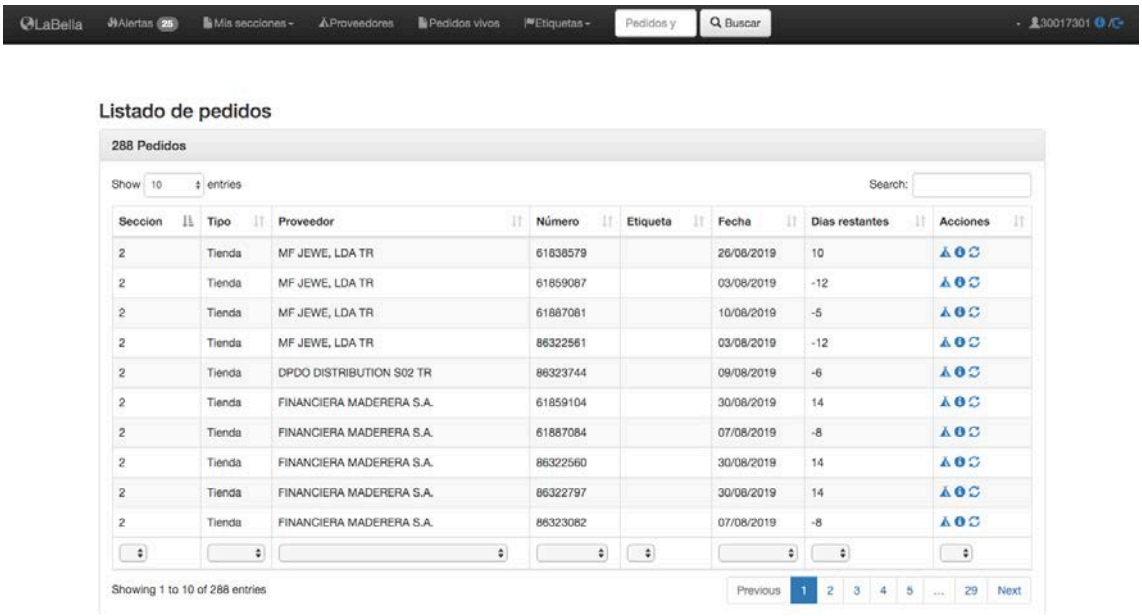

Imagen 28.Pantalla listado de pedidos

Cuando consultamos la información del pedido, obtendremos la siguiente información:

Detalle

Fecha encontrada: Es la fecha del LPRE en el que se encontró el pedido por primera vez.

Fecha entrega real: Esta fecha es calculada en función de la fecha indicada en el fichero, teniendo en cuenta que, si es un proveedor de tránsito, este no entregará directamente en la tienda, sino que lo entregará en el almacén central y llegará a la tienda 2 días laborales después.

Días restantes: Es la diferencia del día actual respecto a la fecha prevista, pudiendo ser negativo si el proveedor entra en retraso.

Unidades: El número de unidades que hay en el pedido.

Importe PC/PVP: Importe a precio de compra del pedido e importe a precio de venta.

Proveedor

Número: Número del proveedor

Nombre: Nombre del proveedor

Contacto: Nombre de la persona de contacto, esta información sirve a la hora de personalizar los correos que serán mandados desde la aplicación.

Teléfono: Teléfono de contacto del proveedor

Email: Email de contacto del proveedor. Esta es la dirección de correo a la que serán mandados los correos.

Estado

Revisado: Información sobre si el pedido ha sido revisado.

Tipo: Tipo de pedido, Tienda, RM, Pedido Cliente, Operación Comercial.

Destacado: Indica si es un pedido que aparecerá en página de alertas como destacado.

Avisar al llegar: Si es necesario enviar un correo el día previsto de entrega en tienda.

Esperando anular: Esperando a tener la confirmación de la anulación por parte del proveedor. Al estar en pendiente de anular, no será tenido en cuanta en las gráficas de entrega.

En el apartado Email a proveedor, aparecen una serie de botones para generar los correos personalizados a los proveedores. Estos correos son referidos sobre temas como la confirmación, reclamación de retrasos de entrega, adelantar fechas de entrega de pedidos, confirmar nuevas fechas de entrega, anulación de pedidos y confirmación de anulación de estos.

Estas opciones son las opciones por defecto, en el caso de que fuera un pedido cliente sin entrega en tienda, aparecerían 2 opciones más reclamando el conforme de entrega poniendo en copia al departamento de pedido cliente, que es el encargado de las posteriores gestiones a realizar.

El siguiente apartado son las observaciones, las cuales quedan registradas por el usuario que la realizo con la fecha.

8.7. Detalles de pedido

El último apartado es un detalle de los artículos incluidos en el pedido.

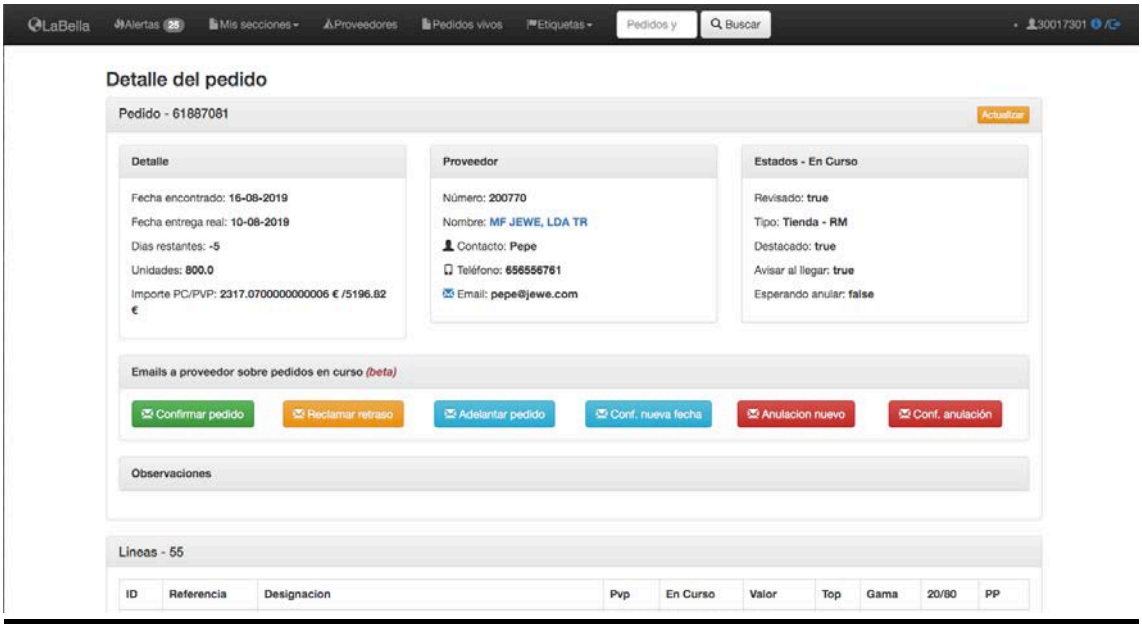

Imagen 29.Pantalla detalle de pedido

Esta pantalla también está pensada para agilizar las comunicaciones comunes con los proveedores y otros departamentos. Las acciones habituales son la confirmación de un pedido, reclamar un retraso, solicitar una anulación fuera de fecha, confirmar una anulación fuera de fecha, anular un pedido nuevo y confirmar la anulación de un pedido nuevo.

A continuación, se mostrará un correo generado al pulsar el botón de reclamar un retraso. Debido a limitaciones, la opción más adecuada fue generar un hipervínculo con abrir la propia cuenta de correo del usuario y volcar toda la información en un correo para Gmail, servicio utilizado por la empresa. Toda la información útil acerca del pedido y del motivo del mail será volcado dentro de ese correo incluyendo a los departamentos necesarios y proveedores.

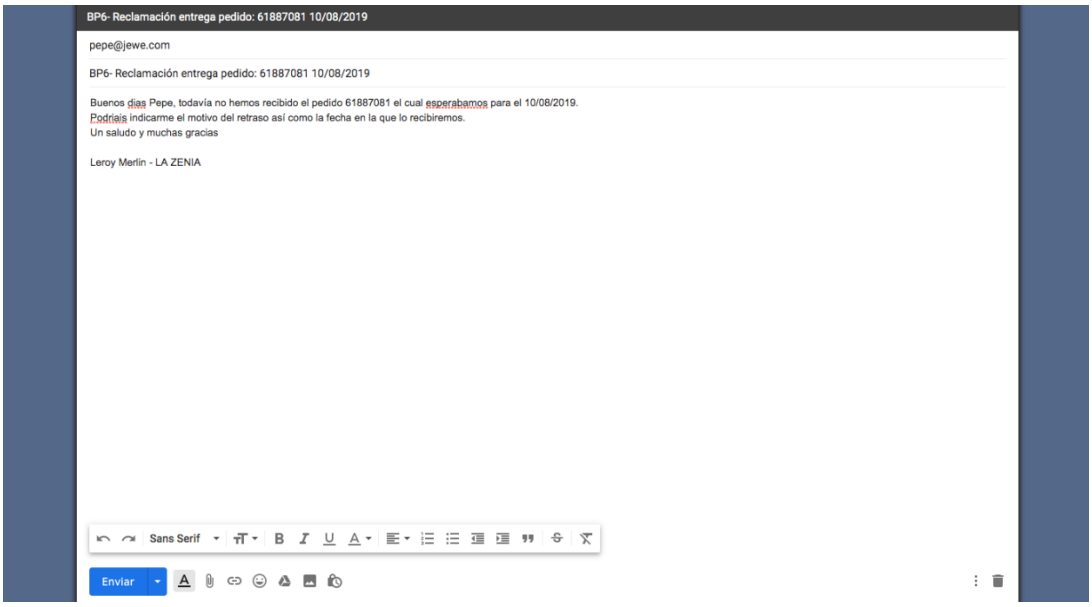

Imagen 30Pantalla email generico retraso

La última pantalla por mostrar es la referente a la actualización de los datos de un pedido. No es posible modificar la mercancía que incluye el pedido puesto que esto no tendría ningún sentido. Los datos por modificar están relacionados con la información extra a aportar a un pedido, las alertas y seguimiento acerca de ese pedido. La información extra tiene muchas veces que ver debido a llamadas de un proveedor indicando un retraso, el número de pedido cliente si fuera necesario y etiquetar los pedidos.

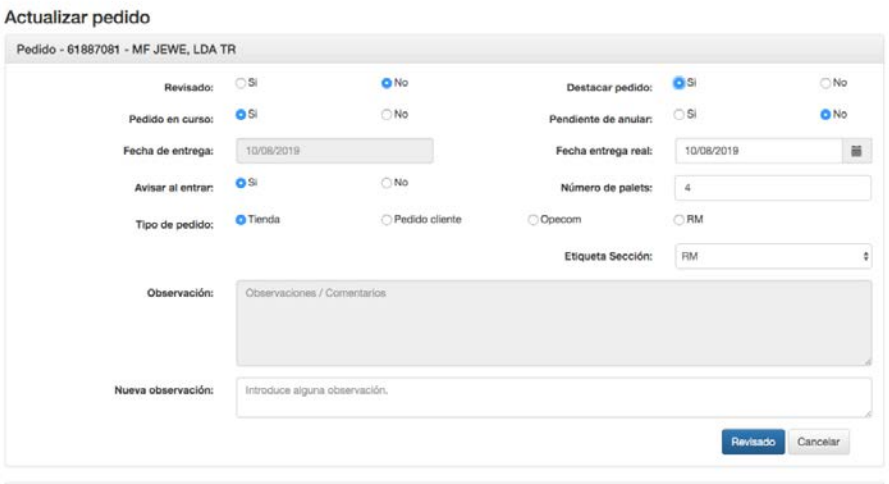

Imagen 31.Pantalla actualización de pedidos.

9. Conclusiones

Puesto que este proyecto forma parte de una serie de aplicaciones desarrolladas personalmente para utilizar por mí, junto a mis compañeros de trabajo, comentaré como ha sido la experiencia de análisis, desarrollo, implantación y los resultados después de unos cuantos meses en funcionamiento.

9.1. Aquello conseguido

La experiencia de realizar el análisis, desarrollo e implantación de la aplicación en una tienda ha sido muy enriquecedora. La utilidad de la misma se vio desde las primeras reuniones con los usuarios finales y las diferentes entregas del mismo. Los más enriquecedor y complicado ha sido conseguir cambiar la forma de trabajo de muchos de esos colaboradores y conseguir su implicación y aportación en el mismo.

Se ha conseguido implantar durante un periodo de 6 meses en una tienda con los 10 mandos intermedios de la misma. Su utilización era diaria registrando varios accesos al día por cada uno de ellos.

9.2. Experiencia personal

Personalmente ha supuesto una gran cantidad de esfuerzo en análisis, gestión, reuniones y encontrar el tiempo para poder desarrollarla. La formación universitaria ha servido para ser capaz de afrontar un proyecto de esta magnitud siguiendo un itinerario en las diferentes etapas del proyecto. Una vez realizada una aproximación inicial a estas etapas, el trabajo de desarrollo ha sido mediante ejemplos de uso de los diferentes Frameworks y horas de autoformación.

Adquirir el conocimiento tanto en estos Frameworks de Spring como en Roo ha supuesto una gran inversión de tiempo que se ha visto recompensada en el momento del desarrollo puesto que, gracias a estos, se ha conseguido reducir mucho el tiempo en desarrollo. Esta forma de realizar los arranques de proyectos junto con los arquetipos es, sin duda, la forma más ágil para poder comenzar un proyecto y centrarse en el desarrollo.

La parte más complicada ha sido saber delimitar el alcance del proyecto puesto que en función de las actualizaciones, de estas surgían nuevas ideas que podrían ser incorporadas fácilmente.

9.3. Trabajo futuro

En futuras versiones podría adaptar la usabilidad para hacerla compatible con dispositivos móviles puesto que, aunque se ha utilizado Bootstrap, no se ha configurado la parte visual para todos los tamaños. También se contempla incorporar la información que proporciona otro tipo de fichero de la compañía en el que muestra todo el stock de las tiendas, el cual, al recibirlo diariamente se sería capaz de analizar informaciones de ventas, rotación, seguimientos de venta... Estas funcionalidades ya las hace otra aplicación que desarrolle hace años con muchos menos conocimientos y supondría adaptarlas en la nueva versión.

Actualmente están publicados únicamente los servicios REST para consumir información. Podría terminar de generar el resto de las acciones para publicar todos los servicios necesarios para que la aplicación funcionará desde cualquier cliente.

Entre las opciones manejadas esta Angular, la cual permitiría incrementar enormemente la experiencia del usuario por la cantidad de controladores de terceros que permite y al ser una tecnología SPA daría la sensación de reducir los tiempos de respuesta.

Finalmente, lo ideal sería separar el proyecto en varios para poder generar las librerías de manera independiente gestionando sus dependencias. Esto permite generar versiones estables y de desarrollo por cada uno de los módulos generados.

10. Bibliografía

Libros:

- Desarrollo de aplicaciones mediante el Framework de Spring: Una panorámica del Framework para J2EE más utilizado del momento
  Eugenia Pérez Martínez

  ISBN: 9788499645568

- Hibernate Persistencia de objetos en JEE
  Eugenia Pérez Martinez

  ISBN: 9788499645582

Sitios web:

- Bootstrap: https://getbootstrap.com/
- Graficas (Chart.js): https://www.chartjs.org/
- Spring Framework: https://spring.io/projects/spring-framework
- Spring Boot: https://spring.io/projects/spring-boot
- Spring Roo: https://projects.spring.io/spring-roo/
- Spring Data: https://spring.io/projects/spring-data
- Apache Tiles: https://tiles.apache.org/
- Apache POI: https://poi.apache.org/
- Stackoverflow: https://stackoverflow.com/