Grado en Ingeniería Informática

Trabajo de Final de Grado

Sistema de trazabilidad para seguimiento de producción y control de stocks de materias primas

Autor:
Jorge Sánchez Jiménez

Supervisor:
Xavier García Gandía
Tutor académico:
Rafael Berlanga Llavori

Fecha de lectura: 13 de julio de 2022
Curso académico 2021/2022

Resumen

En este documento se recoge la memoria técnica correspondiente al proyecto desarrollado por Jorge Sánchez Jiménez junto a los integrantes de la empresa *Robottions*, en la cual se realizó la estancia en prácticas. El proyecto se define como una herramienta informática para conseguir la digitalización de los datos de una empresa industrial. Concretamente, se pretende mantener un sistema de trazabilidad para el seguimiento de la producción y control de estocaje de materias primas. Para ello se ha implementado una *API REST* con distintas funcionalidades bajo la tecnología de *Django REST framework*, y una aplicación *web* adaptada a los requerimientos del proyecto que ha sido desarrollada con el *framework Vuejs*. En esta memoria se documenta tanto el proceso de planificación, como las etapas de análisis, implementación y validación del proyecto.

Palabras clave

API REST, Aplicación web, Django, Vuejs, Gestión de estocaje.

Keywords

API REST, Web App, Django, Vuejs, Stock management.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Definimos la cuarta revolución industrial, o *Industria 4.0*, como la instauración de las nuevas tecnologías dentro del ecosistema de la industria fusionando así el proceso productivo con la digitalización, de tal modo que se enfoca a lograr la automatización de la producción, la conectividad y la globalización. Esto implica principalmente un gran ahorro de costes temporales respaldados por la reducción de fallas en el proceso productivo y por la implantación de tecnología predictiva. [1] Además podemos decir que la *Industria 4.0* va de la mano de la inteligencia artificial, puesto que el sector industrial es uno de los sectores donde mayor inversión de estas tecnologías se está realizando. [2]

El proyecto que en este documento se recoge se desarrolló en la empresa *Human-Robot Solutions S.L. Robottions*, acrónimo de *Human-Robot Solutions*, es una empresa que apuesta fuertemente por la *Industria 4.0* de la mano de la robótica industrial y la automatización tanto en los procesos de producción como en las cuestiones logísticas. Se especializan en la implementación de robótica colaborativa, vehículos autónomos y en celdas robóticas capaces de llevar a cabo distintas tareas de forma autónoma. Otra de las áreas de actuación de *Robottions* es el desarrollo e integración de *software* a medida para distintas empresas de cualquier sector, se centran tanto en el desarrollo de páginas web como en la creación de *ERPs* (*Enterprise Resource Planning*) con el fin de conseguir una gestión interna de la empresa entre otros productos informáticos. [3]

En este caso, el proyecto que se ha llevado a cabo consiste en un *software* para la gestión interna de una empresa dedicada a la recuperación de textil, es decir, al triturado de productos textiles sobrantes para la producción de borra que se distribuye entre distintos clientes que harán distintos usos de esta. Dicha empresa cuenta con una gran cartera tanto de clientes como de proveedores y por ello consiguen mantener un nivel de producción considerable, el cual debe de ser gestionado de manera óptima.

Concretamente el *software* que se ha desarrollado abarca todo el proceso de producción de la empresa, desde que se inicia con el proceso de recepción de materias primas hasta que finaliza

con el proceso de expedición de los productos finales, pasando por el control del proceso de transformación. Su correcto funcionamiento es gracias a la monitorización de la trazabilidad de las balas (tipo de embalaje de las materias utilizado en este sector industrial). Esta monitorización es llevada a cabo mediante un sistema de códigos de barras que permite la digitalización de los productos y con ello mantener el seguimiento del proceso productivo de la empresa acercándola hacia la integración de la *Industria 4.0*.

Entrando más en detalle, dicha digitalización se verá reflejada en una base de datos donde se almacenará la trazabilidad de las materias. Para poder manejar estos datos se ha decidido crear una *API REST* (*Application Programming Interfaces*) la cual permitirá la comunicación de la base de datos con los usuarios finales mediante una aplicación *web*. Dicha aplicación *web* consta de una serie de pantallas las cuales permitirán a los operarios realizar las tareas de manera cómoda e intuitiva. [4]

1.2. Objetivos del proyecto

El objetivo principal de este proyecto es **conseguir cubrir todas las necesidades que dicha empresa pueda tener a nivel de control de producción y estocaje de manera digital**. Podemos enunciar este objetivo usando una terminología técnica de la siguiente forma: **conseguir crear una aplicación *web* la cual sea capaz de ofrecer una herramienta informática a los empleados de la empresa de tal manera que abarque cada una de las tareas del proceso productivo**.

Con el fin de lograr este objetivo, primeramente lo dividiremos en diversos subobjetivos para ayudarnos a entender y desarrollar correctamente el proyecto.

- El primero de ellos será hacer un **estudio previo del proceso productivo** y del control de estocaje de la empresa de tal forma que se pueda definir claramente los requerimientos del proyecto.
- Una vez se conozcan estos detalles, deberemos **concretar los casos de usos** que la aplicación debe cumplir.
- Cumplido el subobjetivo anterior, nos centraremos en el **diseño y desarrollo de la base de datos** de tal forma que garantice la consistencia en los datos.
- El siguiente subobjetivo será **definir e implementar una *API REST*** de tal forma que abarque las funcionalidades que habremos definido anteriormente. Esta *API* deberá ser capaz de conectar la información o funcionalidades con los requerimientos de la aplicación cliente.
- De la mano del anterior subobjetivo, es momento de **diseñar e implementar una aplicación *web***. Como comentábamos, esta aplicación cliente será la encargada de comunicar al usuario con la *API*. Por ello, esta aplicación debe de estar adaptada tanto al usuario como a las funcionalidades definidas.
- Finalmente, podemos definir también como subobjetivo **verificar y validar** si se cumplen satisfactoriamente los requerimientos acordados.

Conceptualmente el proyecto consiste en crear una *API REST* que ofrezca las operaciones necesarias para llevar a cabo la trazabilidad de los productos mediante el tratamiento de la base de datos que previamente se ha diseñado. A su vez se ha de crear una serie de interfaces interactivas para el usuario para que este sea capaz de interactuar con dicha *API*.

Alcance funcional

Como hemos comentado, este proyecto consta de tres componentes: base de datos, *API* y aplicación web para el cliente.

En lo que a la base de datos respecta, se precisa que esta sea capaz de mantener en todo momento estados válidos de los datos garantizando su consistencia. En ella se almacenarán tanto datos de clientes y proveedores, como la trazabilidad de los productos.

En cuanto a la *API* esta debe de estar diseñada y adaptada a cada una de las etapas del proceso productivo. Para ello, es necesario que garantice que las operaciones que el usuario desee hacer sobre los datos puedan realizarse de forma válida. Además la API debe servir en todo momento al cliente una respuesta, ya bien sea de datos o de *feedback*.

Con lo que a la aplicación *web* del cliente se refiere, esta debe de ofrecer al usuario final una herramienta intuitiva para poder digitalizar el proceso de producción. Esta aplicación *web* será la encargada de hacer posible la comunicación entre el usuario y la *API*. Entre las pantallas podemos encontrar distintos listados, formularios o interfaces específicas para el proceso de producción.

Alcance organizativo

El proyecto al completo se ha diseñado e implementado íntegramente por el equipo de *Robottions*, concretamente por el departamento que se dedica al desarrollo de *software* a medida en el cual me considero incluido.

Cabe destacar que a la hora de analizar y definir los requerimientos de este, se han realizado entrevistas con los usuarios finales de la aplicación para conocer en exactitud los datos a almacenar, los requerimiento a cumplir, además de cómo deben de estar diseñadas las interfaces con las que los empleados interactuarán.

Por consecuente, los usuarios finales de la aplicación son los propios trabajadores de la empresa contratante. Se definen dentro de la empresa cinco grupos de trabajadores según su rol en el proceso de producción que se ven reflejados a su vez en la aplicación: Cargadores, Carretilleros, Jefes de turno, Oficinistas y los Administradores del sistema. Estos grupos de usuario son los que podrán utilizar dicho software de forma regulada por un sistema de autorización y permisos que posteriormente veremos.

Alcance informático

Veamos el proyecto esta vez desde una visión informática.

Dividiremos este proyecto en dos subproyectos, ***backend*** y ***frontend***, correspondientes respectivamente al **servidor** y al **cliente** del modelo de diseño homónimo.

Dentro del *backend* comprenderemos la base de datos y la *API*. Estas deben de suplir el tratamiento de datos y ser capaces de atender las peticiones del ***protocolo HTTP*** que el cliente realizará. Este proyecto, durante su desarrollo, va a ser albergado en un servidor ***Redis*** local. En un futuro, durante la fase de explotación, se hará uso de un servicio de *hosting* externo.

Por otro lado, en el *frontend* comprenderemos todas las pantallas interactivas del usuario que serán capaces de enviar las peticiones a la *API*. Este proyecto se ejecutará en su desarrollo mediante la herramienta de creación de entornos de ejecución ***Node.js***. De igual modo que con el *backend*, este proyecto se ejecutará sobre un servidor local. Una vez este esté en funcionamiento, se hará uso de un servicio de *hosting* externo.

Internamente, sendos subproyectos han sido desarrollados siguiendo el patrón de diseño ***Modelo-Vista-Controlador (MVC)*** que veremos en detalle posteriormente.

1.3. Descripción del proyecto

1.3.1. Descripción del proceso productivo

Con el fin de ayudar a entender el proyecto que se ha realizado, se explicará el funcionamiento interno de la empresa a nivel productivo.

Recuperados Llacer, que así se llama dicha empresa, se dedica al recuperado textil, es decir, se dedican al reciclado de deshechos de tejidos industriales o recortes de confección procedentes de distintos proveedores con el fin de producir la borra de distintas características para ser distribuida a distintos clientes.

Durante todo el proceso de producción, los productos van a estar empaquetados en ***balas***. Las *balas* son el formato tanto en el que se reciben las materias primas como en el que se preparan los productos finales. El funcionamiento interno de esta empresa es controlado por un sistema de códigos de barras, cada uno de estos códigos alberga la referencia de cada bala llevando así su trazabilidad.

El proceso de producción se divide principalmente en tres etapas:

- **Etapa de recepción de materias primas**. Es en esta etapa donde se reciben las balas de materia prima (***balas MP***) procedentes de los proveedores. Estas balas cuentan con distintas especificaciones como son los kilogramos, color o composición. Una vez la ***bala MP*** ha sido recepcionada, esta es almacenada en una posición de almacén concreta.

- **Etapa de transformación**. Una vez contamos con la materia prima, es turno de empezar a producir la borra. Esta etapa comienza creando la ***plantilla*** que se va a seguir para producir un tipo de borra concreto, en la *plantilla* se especifica que tipo de materias primas y en que porcentaje se van a utilizar para crear un tipo concreto de borra. Una vez la *plantilla* esté definida, se procede a crear la ***Mezclada***. Una *Mezclada* es la ***orden de fabricación*** donde se especifica lo qué se va a producir, dónde se va a producir y que características va a tener.

  Las *Mezcladas* son llevadas a cabo en los ***cuartos***, que no son más que las máquinas trituradoras donde se procesa la *Mezclada* (son llamadas así por su gran tamaño ya que ocupan una habitación entera). En esta etapa los operarios son los encargados de seleccionar las balas de la plantilla, pesarlas y cargar las máquinas para poder empezar a producir la *Mezclada*. Conforme se va produciendo la borra, esta se va empaquetando en balas de producto final (***balas PF*** o ***balas de borra***) que a su vez son almacenadas en un lugar específico. Al conjunto de todas estas balas producidas por una misma *Mezclada* se le denomina ***Partida***.

- **Etapa de expedición del producto final**. En esta etapa el cliente final es el protagonista. Al tener una gran cartera de clientes, cuentan con acuerdos preestablecido introduciendo así el concepto de ***Reserva***. En la *Reserva* se indica que una *Partida* está reservada para ser enviada a un cliente determinado. Cuando se realiza una venta con algún cliente, se genera una *Orden de salida* donde se detallan datos como el cliente o las balas de borra que se enviarán entre otros, dando como resultado el correspondiente albarán.

Comentaremos diversos aspectos que hay que tener en cuenta para comprender ciertos requisitos del proyecto.

Por motivos de optimización de costes, cuando se recepciona la materia prima, esta viene en conjuntos de *balas de MP* las cuales denominaremos ***Entradas***.

A la hora de la creación de una *Mezclada* puede ser que las *balas de MP* no sean completamente consumidas, por ello, se mantiene un seguimiento de los distintos estados de las balas. De una forma similar ocurre lo mismo con las *balas de borra*, estas tienen distintos estados según el estado del proceso de producción.

Del mismo modo, cuando se realiza una *Mezclada*, esta mantiene distintos estados según la linea temporal del proceso de producción en función de si está ejecutándose, si está acabada, etc. Lo mismo ocurre con las *Órdenes de salida* como con las *Reservas*.

Sistema de códigos de barras

Para la trazabilidad de las balas y el control de estas se utiliza un sistema de códigos de barras en formato *EAN*, de tal forma que el código corresponda con la referencia de la bala. Este código será sobre el cual el proyecto de digitalización será desarrollado.

La enumeración de las balas está definida según una codificación concreta preestablecida dentro de la empresa para favorecer la identificación tal y como se muestra a continuación.

Figura 1.1: *Codificación códigos de barras.*

En los códigos de barras se detalla el número de almacén donde dicha bala está almacenada. A continuación podemos ver en la siguiente figura, que la posición de almacenamiento de dicha bala corresponde al *ALMACÉN 1*.

Figura 1.2: *Código de barras de almacén.*

Para las *balas de MP* se presenta la siguiente etiqueta donde se indican los kilogramos totales de la bala, el código numérico, la materia prima que contiene y el proveedor que la ha suministrado.

Figura 1.3: *Código de barras de bala de materia prima.*

Por último las balas de borra vienen definidas con el siguiente esquema. Estas etiquetas indican los kilogramos totales de la bala, el código numérico, el tipo de borra que contiene, la *Partida* en la que se ha producido e información de la empresa.

Figura 1.4: *Código de barras de bala de materia final.*

Sirva la siguiente imagen como un breve resumen gráfico del proceso de producción donde se muestra el funcionamiento de la planta.

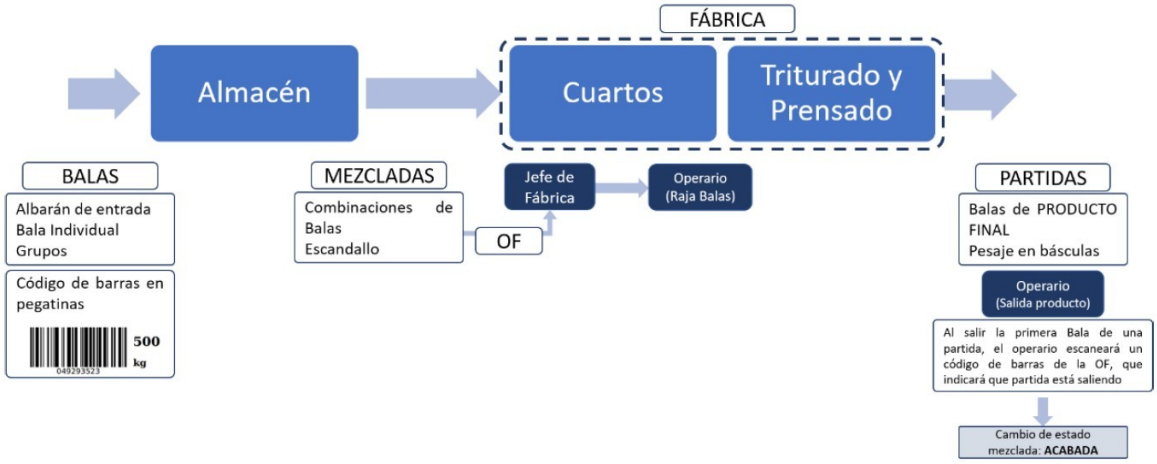

Figura 1.5: *Esquema de gestión de la línea productiva.*

1.3.2. Definición del proyecto

Para cubrir el proceso de digitalización se ha propuesto diseñar una herramienta que permita a los trabajadores de la empresa cubrir las distintas acciones del proceso productivo. Esta herramienta aprovechará el sistema de códigos de barras para llevar a cabo la trazabilidad de las balas.

Mediante esta herramienta, siguiendo el proceso productivo, los operarios podrán digitalizar las entradas de *balas de MP*, manejar el sistema de producción mediante la gestión de las *Mezcladas*, cubriendo así distintos casos de uso que detallaremos en adelante, y controlar el sistema de salidas de las *balas de borra*.

El proyecto estaba previsto ser implementado durante mi estancia en la empresa. La tota-

lidad de este no ha sido implementada por mi sino que nos dividimos las tareas entre el resto del departamento de desarrollo de *software* a medida. Concretamente yo me centré en lo que concierne a la etapa de recepción de materias primas y a la etapa de transformación. En este documento se recogerá únicamente todo aquello que abarca a estas etapas. Para ello, tanto la *API* como la aplicación web que se han propuesto diseñar deben cumplir dichos requerimientos de casos de uso.

1.3.3. Tecnologías utilizadas

Tras un consenso de los miembros del departamento se decidió que para el *backend* se hariá uso del *framework* de ***Python***, ***Django REST Framework*** y para el *frontend*, se hariá uso del *framework* ***Vue.js*** para el desarrollo. Finalmente, para la gestión de la aplicación *web*, se usaría la herramienta ***Node.js***.

Python

Este lenguaje de programación a día de hoy es considerado como uno de los lenguajes de programación más importantes puesto que se consolida con los años, como uno de los más usados en todo el mundo ya que este no tiene ningún propósito específico sino que son muchos los campos donde es usado.

Este simple y legible lenguaje de programación cuenta cada vez más con más adeptos por su sencillez a la hora de realizar programas simples, además, cuenta con una gran cantidad de funcionalidades extra ya que su catálogo de *frameworks* y librerías es bastante extenso abarcando una gran cantidad de ámbitos de desarrollo. [5]

Tal y como he mencionado, en este caso, para la creación del *API* se ha hecho uso de uno de estos *frameworks*, ***Django REST framework*** el cual ha sido instalado con el instalador de paquetes propios de *Python PyPI* (***pip***). [6]

Django REST framework

Django es un *framework* orientado exclusivamente para el desarrollo de *APIs REST*. Por ello, este cuenta con distintas herramientas realmente útiles para desarrollarlas cómodamente. Este *framework* automatiza muchas acciones del desarrollo como la creación de la base de datos o la implementación de los *endpoints* de las funciones *CRUD* entre otros.

Su funcionamiento está basado en el patrón de diseño *MVC* que se ve reflejado en el árbol de directorios que este genera sobre nuestro proyecto. *Django* cataloga como micro-aplicaciones internas a los distintos módulos de la aplicación, de los cuales se deben implementar los modelos, controladores y vistas de cada una mediante los ficheros que se generan automáticamente. [7]

JavaScript

JavaScript es el lenguaje más popular para la programación *web*. Su uso más común es la implementación de páginas *web* interactivas y dinámicas ya que con un código sencillo se pueden conseguir cosas sorprendentes. El código *JavaScript* es interpretado directamente en los navegadores web sin necesidad de ser implementado. Gracias a las tecnologías de peticiones ***HTTP* asíncronas**, *JavaScript* permite realizar intercambio de información con el servidor sin necesidad de recargar la página. [8]

Existen una gran cantidad de *frameworks* para facilitar la programación *web*, son ejemplos de estos los famosos *frameworks* ***React, AngularJS*** o ***Vue.js***. Este último es el elegido para nuestro proyecto. [9]

Node.js

Node.js es una herramienta para la generación de entornos de ejecución de *JavaScript* orientado a eventos asíncronos. *Node.js* permite realizar el tratamiento de aplicaciones *web* aportando herramientas para la creación de aplicaciones y a la gestión de estas de la mano de su gestor de paquetes ***npm***, también permite la ejecución de aplicaciones en cualquier dispositivo. [10] [11]

Vue.js

Uno de los *frameworks* más utilizados a día de hoy para la implementación de aplicaciones *web* es ***Vue.js***. Este está orientado a la creación de aplicaciones de una sola página diseñando una experiencia más fluida y cómoda para el usuario. Su arquitectura está centrada en componentes, normalmente procedentes de librerías como son ***BootstrapVue*** o ***Vuetify***, los cuales son cargados directamente en la interfaz gráfica pudiendo ser tratados y personalizados. Para este proyecto se ha hecho uso de *Vue 3* conjuntamente con la biblioteca *Vuetify*.

Internamente este *framework* utiliza el patrón *MVC* aportando el almacenamiento, tratamiento y visualizado de variables de una forma realmente sencilla.

Un aspecto a tener en cuenta es que para cuestiones complejas de desarrollo *web*, como el enrutamiento, gestión de estados o tratamiento de *SSE* se hacen a través de librerías externas, en nuestro caso se ha hecho uso de *VueRouter*, *Vuex* y *EventBus*. [12]

1.4. Estructura de la memoria

Para analizar el desarrollo del proyecto lo dividiremos en tres partes. Tomaremos el siguiente capítulo para detallar la planificación del proyecto indicando temas como la metodología, la planificación temporal que se han seguido y el seguimiento que se ha llevado sobre este entre otros aspectos.

En el capítulo número tres de este documento abordaremos temas sobre el análisis y diseño del sistema donde se presentará en detalle los requerimientos del proyecto y se describirá que aspectos se han seguido para el desarrollo del proyecto.

Finalmente, dedicaremos un capítulo a la implementación del proyecto donde abarcaremos temas que responden a la pregunta "Cómo se ha implementado?". Trataremos también cuestiones sobre la verificación y validación donde veremos si nuestro proyecto es apto para ser puesto en producción.

Capítulo 2

Planificación del proyecto

2.1. Metodología

En este punto tuvimos controversias con el equipo donde previamente, antes de realizar la planificación del proyecto, se comentó de realizar una metodología ágil de la mano de *SCRUM*, pero, por cuestiones de preferencias del equipo, al final se decidió realizar una metodología de **desarrollo en cascada**. El equipo no había trabajado previamente con una metodología ágil y por ello les incomodaba la idea de cambiar su forma de trabajo habitual. [13]

Se optó por el desarrollo en cascada puesto que además de la habitualidad de este dentro de la empresa, el proyecto propiciaba a realizarlo de esta forma, siguiendo una planificación lineal de las tareas dotándole de un gran seguimiento del proyecto. El proyecto está desarrollado siguiendo el patrón de diseño *MVC* por ello nos obligamos a seguir el orden de programación de *Modelo-Controlador-Vista* debido a sus dependencias, es decir, primero debemos tener implementado el *Modelo* para poder implementar el *Controlador* y así sucesivamente. Esto implica que el proceso de desarrollo seguirá la estructura de creación de la base de datos, desarrollo de la *API* y finalmente la implementación de la aplicación *web*, previamente encontraremos una etapa de estudio y diseño de estas. (Figura 3.11)

Tal y como comenté, el proyecto está dividido en distintas partes según el proceso productivo. Existe una dependencia entre las partes desarrolladas, pero esta es mínima puesto que se puede realizar gran parte de las funcionalidades sin que las demás estén implementadas, únicamente se requiere en la última parte del proyecto. Así los modelos quedan claramente diferenciados y altamente independiente entre ellos según las fases productivas.

Para poder entender el proceso de producción y los requerimientos, el equipo realizó una serie de consultas a los usuarios finales y a los clientes, de tal modo que los requerimientos estén bien definidos y poder delimitar los casos de uso y estimar costes y recursos para que con ello se pudiese crear la planificación adecuada.

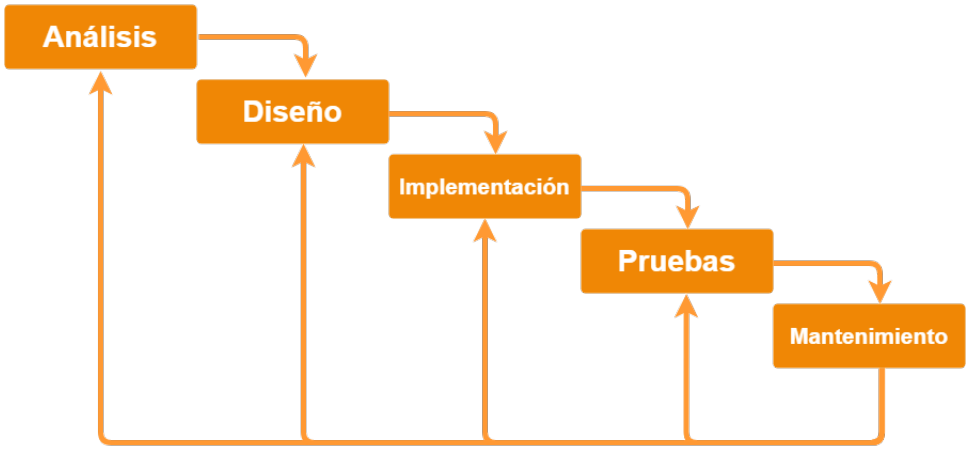

Figura 2.1: *Metodología en cascada seguida en el proyecto.*[14]

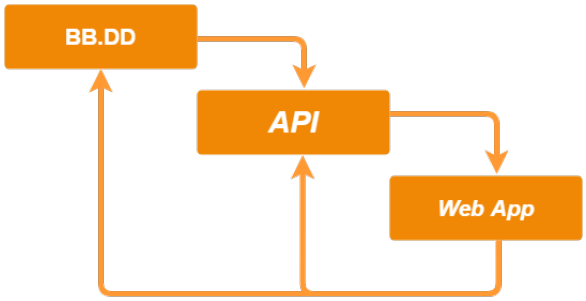

Figura 2.2: *Detalle de la fase de implementación.*

2.2. Planificación

A continuación detallaremos las tareas a realizar dentro del desarrollo del proyecto, esto nos servirá para planificar este correctamente.

- Como tarea previa al proyecto, a modo de formación, se va a realizar un **estudio de las tecnologías a utilizar**. De esta forma conoceremos los *frameworks* a utilizar mediante el visualizado y seguimiento de cursos *online* y la lectura de documentación de estos.
- Durante la **etapa de análisis** distinguiremos las tareas pertinentes a entender los requerimientos del proyecto. Entre estas encontramos el estudio del proyecto donde se analizará el proceso de la empresa y de que modo se puede digitalizar, la definición de requisitos donde se anotarán los requerimientos de casos de uso de la aplicación y que matices debe abordar, y por último, se realizará un estudio de los usuarios de tal forma que se pueda contrastar los requerimientos y detallar ciertos aspectos como el estudio de la accesibilidad de los usuarios finales.
- Seguidamente, en la **etapa de diseño**, se presentarán los prototipos de la solución informática donde primeramente se diseñará la base de datos en cuestión de tal forma que esta cumpla los requerimientos acordados. Se realizarán los prototipos iniciales del proyecto entre los cuales encontraremos los *mockups* de las interfaces y seguidamente se

presentará un esquema de la *API* donde se intentará presentar la lista de *endpoints* a implentar.

- En la **etapa de implementación** se abordarán las tareas pertinentes a la programación del código del programa. En esta se crearán los modelos, se programará la funcionalidad de los endpoints y se configurarán los tests pertinentes. Por último, se implementará la *WebApp* con sus respectivas interfaces gráficas.
- Una vez funcione todo correctamente, en la **etapa de pruebas y mantenimiento**, se realizará el despliegue en los servidores y se consultará y corregirá el funcionamiento de este de acuerdo con las expectativas del proyecto.

A continuación se recoge el listado de tareas distribuidas en las distintas etapas del proceso de desarrollo del proyecto, en este se muestra la estimación temporal de estas conjuntamente con las tareas antecesoras que deben cumplirse antes de poder realizar dicha tarea.

Nº	Tarea	Duración	Predecesoras
1	Estudio de tecnologías	14 días	-
	Análisis	**11 días**	
2	Estudio del proyecto	4 días	1
3	Definición de requisitos	4 días	2
4	Estudio de usuarios	3 días	3
	Diseño	**13 días**	
5	Definición de la BB.DD	4 días	4
6	Diseño de prototipos	3 días	5
7	Planificación del *API*	6 días	6
	Implementación	**23 días**	
8	Creación de modelos del *API*	5 días	7
9	Creación de *endpoints* y tests	10 días	8
10	Creación de *WebApp*	8 días	9
	Pruebas y mantenimiento	**3 días**	
11	Puesta en marcha	3 días	10

Cuadro 2.1: *Listado de tareas a realizar durante el desarrollo del proyecto.*

Para poder establecer dichas tareas en la línea temporal del proyecto se presenta el siguiente diagrama de *Gantt* donde se recoge de manera gráfica las tareas con sus respectivas indicaciones temporales y dependencias entre estas. Cabe destacar que estas tareas están presentadas únicamente para un único trabajador, para mi. Es por ello por lo que la programación de estas es totalmente lineal.

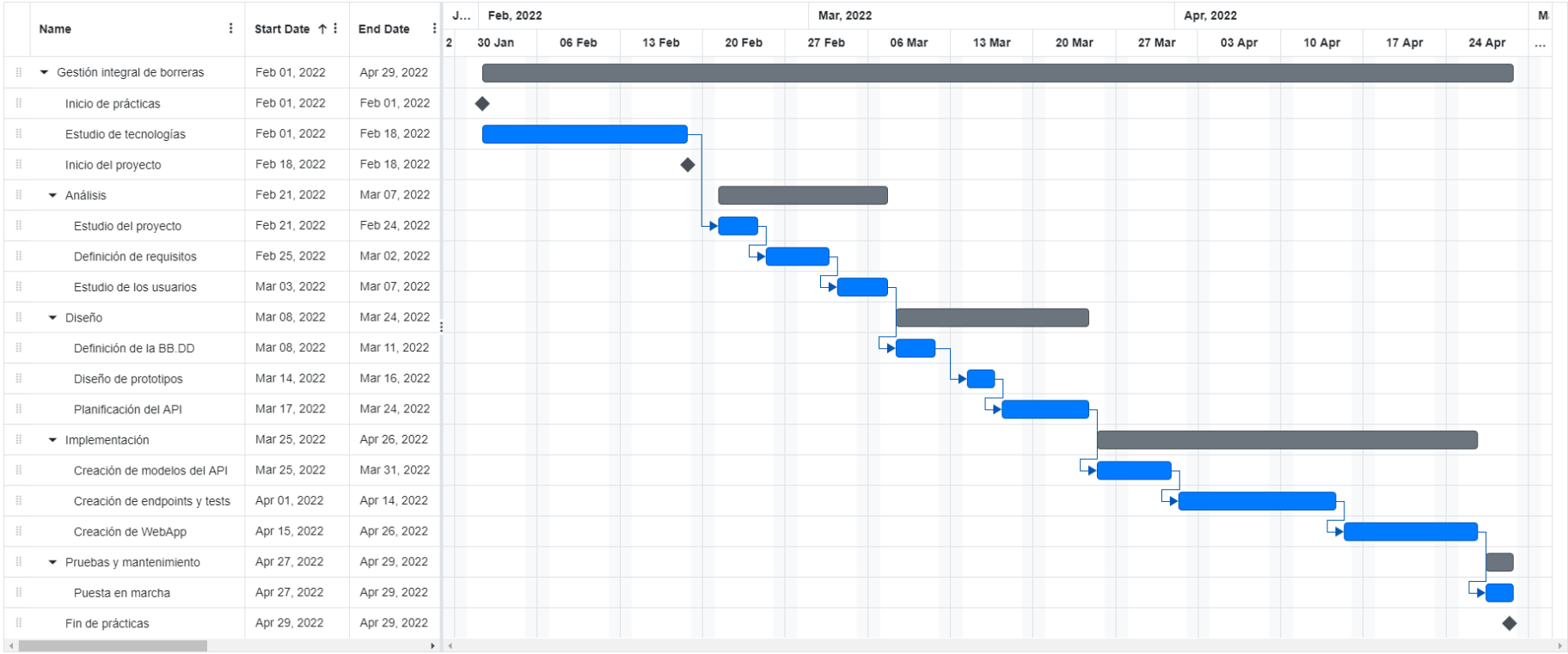

Figura 2.3: Diagrama de *Gantt* de la planificación del proyecto.

2.3. Análisis y gestión de riesgos

2.3.1. Análisis de riesgos

A la hora de planificar el proyecto se han tenido en cuenta una serie de riesgos que pueden generar una serie de incidencias en su correcto desarrollo. Estos riesgos pueden suponer demoras en el seguimiento de la planificación. Entre los riesgos valoraremos los que afecten directamente al producto informático y riesgos relacionados con el proyecto y su desarrollo.

Se han identificado los siguientes riesgos:

- **Inexperiencia**. Se ha planteado este riesgo de producto como un posible foco de retrasos durante la fase de implementación, y como foco de errores en el código de programación, suponiendo así demoras en la entrega del proyecto y fallas en la calidad de este.
- **Distribución temporal errónea**. Debido a una mala estimación temporal de las tareas, esto puede suponer tanto retrasos como avances temporales suponiendo los retrasos demoras en la entrega o trabajar con prisas que pueden conllevar errores de programación y suponer un producto de baja calidad.
- **Baja por enfermedad**. Debido al contexto socio-sanitario en el cual se ha desarrollado el proyecto, las probabilidades de baja por enfermedad son bastante altas valorando así la posibilidad de no poder trabajar durante el periodo de baja. Esto obviamente puede acarrear demoras en el proyecto.
- **Cambios en los requerimientos del proyecto**. Puesto que es un proyecto que no está acordado en su totalidad con los clientes, cabe la posibilidad que estos decidan cambiar algunos aspectos a ser implementados en el proyecto durante el desarrollo de este, y se tenga que hacer saltos atrás entre etapas del proyecto. Este riesgo de proyecto puede suponer importantes retrasos en la entrega.
- **Cambios en el diseño inicial**. Del mismo modo que con el anterior riesgo, pueden aparecer errores en alguna etapa del desarrollo que reflejen alguna falla en el diseño inicial y que deba ser solucionado para continuar con el funcionamiento correcto. Dependiendo de la gravedad de los fallos acometidos por este riesgo del producto, pueden suponer importantes dilaciones en el desarrollo.
- **Falta de control**. Debido a que no se lleve un seguimiento adecuado sobre las partes implicadas en el proyecto, es decir, que los desarrolladores no estén trabajando correctamente sobre la planificación, pueden ocasionar un mal trabajo en equipo y acarrear, normalmente, pequeños retrasos en el proyecto.

2.3.2. Planes de contingencia y prevención

Con la finalidad de que estos riesgos previamente identificados sean prevenidos de ocurrir o mitigados en caso de ser sucedidos, se presenta a continuación una serie de medidas a seguir para cumplir dicho objetivos.

- **Inexperiencia**. Para evitar esto, se ha detallado dentro de la planificación del proyecto una serie de horas donde se dedicarán a la formación sobre estas tecnologías. Si durante el proceso aparece una cuestión fuera de los cabales de la formación, se preguntará a superiores y a expertos en estas herramientas.

- **Distribución temporal errónea**. Con el fin de evitar que se creen fallos en la planificación temporal, es recomendable que previamente se consulte con expertos en materia que hayan participado previamente en distintos proyectos y tengan mayor experiencia para estimar costes temporales. Si tras esta prevención aparecen fallos relativos a la distribución, se realizarán reuniones con el equipo para acordar márgenes de mejora.

- **Baja por enfermedad**. Se pretende que se mantengan las medidas de seguridad en todo momento a modo preventivo. Puesto que las posibilidades de enfermar son altas, se propone el teletrabajo como forma de participación en el proyecto de forma que no aparezcan nuevas bajas que supongan aún más demoras.

- **Cambios en los requerimientos del proyecto**. Se pretende en todo momento zanjar los requerimientos y dejar claro en las reuniones con los clientes los requisitos a cumplir de tal forma que no sea un problema en el desarrollo del proyecto. Si por cualquier motivo aparecen modificaciones, estas no deben suponer mucho problema puesto que el proyecto deberá ser implementado de manera escalable y modular para favorecer estas posibles desviaciones.

- **Cambios en el diseño inicial**. De la mano del anterior riesgo, mantener definidos y concretados los requerimientos favorecerá a que el diseño no quede sujeto a modificaciones *a posteriori*. Por ello, las reuniones entre los implicados en el proyecto deben de ser claves. Puesto que el código se pretende que sea modular y escalable, las modificaciones en el diseño no deben de ser verse reflejadas de forma compleja.

- **Falta de control**. Las reuniones entre desarrolladores deben de servir sobre todo como una herramienta de seguimiento del proyecto donde se presente y se valore el trabajo realizado para así controlar que todo está siguiendo el rumbo preestablecido.

2.4. Estimación de recursos y costes del proyecto

Trataremos en esta sección otro aspecto a ser negociado a la hora de planificar el proyecto, la gestión de recursos utilizados durante el proyecto y un estudio de los costes que este conlleva. Durante el análisis se mantiene en cuenta tanto los recursos humanos presentes durante el progreso del proyecto como los recursos tanto *hardware* como *software* que han sido utilizados a la hora de desarrollar e implementar el proyecto.

Empecemos tratando los costes de los recursos humanos que esto supone para la empresa. Contamos con que en el proyecto han participado tres programadores, divididos en dos veteranos y a mi como junior. Desconociendo el salario de de los empleados de la empresa, se toma como salario base estándar de unos *20,00€* cada hora trabajada. Por otro lado, yo, como programador junior, se estima alrededor de *7,00€* cada hora trabajada. La duración del proyecto está fijada a trescientas horas. Se muestra a continuación el desglose de gastos en recursos humanos del proyecto.

Concepto	Precio (€/hora)	Horas	Coste Total (€)
Programador senior 1	20,00	300	6000,00
Programador senior 2	20,00	300	6000,00
Programador junior	7,00	300	2100,00

Cuadro 2.2: *Desglose de gastos de recursos humanos.*

En cuanto a gastos relacionados con el *software* vemos a continuación el desglose relacionado. Debido a que la cantidad de integrantes en el equipo es inferior cinco personas y a los requisitos del proyecto, el plan gratuito del software *BitBucket* es más que suficiente. En caso de que fuera necesaria la integración de nuevos desarrolladores, habría que solicitar un plan de pago. [15]

Proyecto	**Concepto**	**Coste Total (€)**
Ambos	*Visual Studio Code*	0,00
	GitHub	0,00
	BitBucket (Plan gratuito)	0,00
	Navegador *web*	0,00
Backend	*Framework Django*	0,00
	PostgeSQL	0,00
	Python PIP	0,00
	Python pyenv	0,00
	Redis Server	0,00
Frontend	*Framework Vuejs*	0,00

Cuadro 2.3: *Desglose de gastos de recursos de software.*

Relacionado con los gastos de *hardware* realizaremos una estimación en cuanto al coste proporcional al uso de estos durante el desarrollo del proyecto. Definiremos una vida útil de aproximadamente tres años de los equipos. El tiempo sobre el cual se van a utilizar dichos equipos es de tres meses. Veamos a continuación el desglose de gastos de *hardware*.

Concepto	**Coste Total** (€)	**Coste proporcional** (€)
Ordenador portátil	1200,00	100,00
Teclado y *mouse*	20,00	1,66
Pantalla secundaria	85,00	7,08
Servidor	800,00	66,66

Cuadro 2.4: *Desglose de gastos de recursos de hardware.*

Finalmente se computan todos los gastos para obtener el coste total del proyecto. Se calcula además el coste proporcional del proyecto en cuanto a las especificaciones de uso previamente citadas.

Concepto	Total (€)	Total proporcional (€)
Gastos de recursos humanos	14100,00	14100,00
Gastos de recursos *software*	0,00	0,00
Gastos de recursos *hardware*	2105,00	175,40
TOTAL	**16205,00**	**14275,40**

Cuadro 2.5: *Gastos totales.*

2.5. Seguimiento del proyecto

A continuación se abordarán cuestiones referentes al seguimiento del proyecto donde se presentan las distintas desviaciones acometidas en la planificación, sus causas, cómo se han tratado y sus consecuencias.

Tal y como se presentó en el análisis de riesgos, la empresa contratante del *software* realizó una serie de modificaciones en la propuesta del proyecto. Los requerimientos no quedaban concisos por parte de estos, retrasando la fecha de inicio en aproximadamente un mes.

Puesto que el proyecto estaba definido para ser desarrollado en tres meses, al empezarlo un mes más tarde implicaba una gran demora en el tiempo de entrega. Puesto que la empresa contratante fue la que ocasionó las molestias, estos no han presentado impedimentos a la hora de ajustar la fecha de entrega.

Cabe destacar que la parte que estaba encargado de implementar quedó prácticamente implementada puesto que tuve más tiempo para experimentar con las tecnologías ganando experiencia y fluidez con estas, y por ello, a mi no me afectaron dichas modificaciones en la planificación.

Puesto que se diseñó un plan de contingencia sobre este riesgo, a la hora de planificar el proyecto, se estimaron unos margenes temporales más amplios para reducir su impacto. A continuación se muestra una tabla con la reconfiguración temporal aplicada.

Nº	Tarea	Duración	Fecha inicio	Fecha fin
1	Estudio de tecnologías	26 días	01/02	08/03
	Análisis	**9 días**	09/03	21/03
2	Estudio del proyecto	4 días	09/03	14/03
3	Definición de requisitos	3 días	15/03	17/03
4	Estudio de usuarios	2 días	18/03	21/03
	Diseño	**9 días**	22/03	01/04
5	Definición de la BB.DD	3 días	22/03	24/03
6	Diseño de prototipos	3 días	25/03	29/03
7	Planificación del *API*	3 días	30/03	01/04
	Implementación	**18 días**	04/04	27/04
8	Creación de modelos del *API*	2 días	04/04	05/04
9	Creación de *endpoints* y tests	9 días	06/04	18/04
10	Creación de *WebApp*	7 días	19/04	27/04
	Pruebas y mantenimiento	**2 días**	28/04	29/04
11	Puesta en marcha	2 días	28/04	29/04

Cuadro 2.6: *Listado de tareas a realizar durante el desarrollo del proyecto aplicadas las modificaciones temporales.*

De igual modo a la planificación inicial, se muestra de manera gráfica un diagrama de *Gantt*.

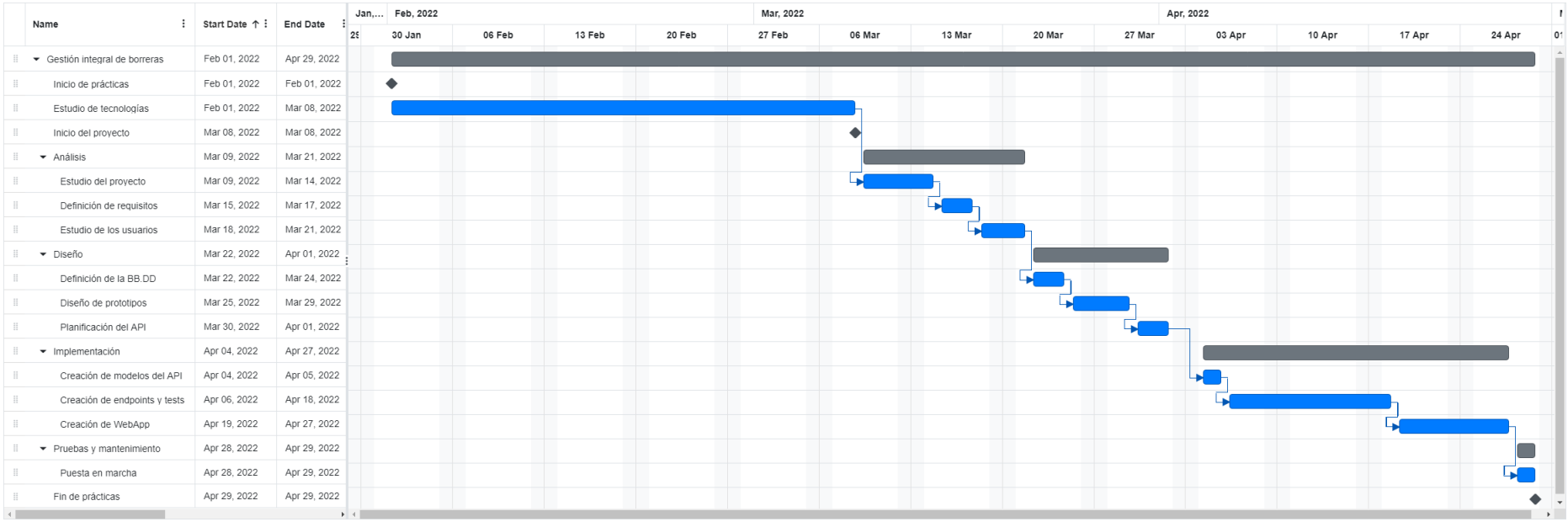

Figura 2.4: Diagrama de *Gantt* de la planificación del proyecto modificada.

Cabe destacar que las funcionalidades básicas de mi parte del proyecto han sido implementadas correctamente quedando una pequeña serie de tareas menos relevantes en el proyecto que no han podido ser implementadas durante la estancia de prácticas, como ha sido la distinción por roles. El resto del equipo se encargará de realizar dichas tareas y se encargarán de realizar el despliegue final del proyecto.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

Durante esta sección realizaremos un análisis exhaustivo sobre el proyecto desde una visión informática. Comenzaremos definiendo los distintos casos de uso que se han presentado y que deben de ser implementados en nuestro proyecto. Únicamente se prestará documentación acerca de las tareas que me han sido encomendadas.

A continuación se muestra el diagrama de casos de uso correspondiente al proyecto donde se muestran tanto los actores involucrados en la aplicación como los distintos casos de uso que deben de satisfacerse.

Figura 3.1: *Diagrama de casos de uso del proyecto.*

Definiremos ahora cada uno de ellos entrando en detalles, e identificando ciertos aspectos.

Código	CU01
Nombre	Gestionar usuarios del sistema
Complejidad	Media
Descripción	Los administradores del sistema mediante una herramienta de administración propia de *Django* serán encargados de crear, modificar y eliminar los usuarios del sistema otorgándole el rol y los permisos correspondientes.
Objetivo	Controlar y gestionar los usuarios según el rol atribuido.
Actor principal	Administradores
Actores secundarios	-
Requisitos previos	-
Secuencia normal	1. Accederá a la *URL* correspondiente de la herramienta. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de "Usuarios y grupos". 4. Elegirá la acción a realizar con el usuario. 5. Completará los datos pertinentes en caso de registro. 6. Confirmará las acciones perpetradas.

Cuadro 3.1: *Caso de uso CU01.*

Código	CU02
Nombre	Administrar "Proveedores"
Complejidad	Fácil
Descripción	Los oficinistas se encargarán de registrar, modificar, eliminar y consultar los proveedores de la empresa mediante una serie de pantallas. (Acciones *CRUD*).
Objetivo	Gestionar la lista de proveedores de la empresa.
Actor principal	Oficinistas
Actores secundarios	-
Requisitos previos	-
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de "Listados". 4. Elegirá la categoría de "Proveedores". 5. Escogerá la acción que desee realizar con los proveedores, siguiendo el proceso lógico del sistema. 6. Confirmará las acciones perpetradas.

Cuadro 3.2: *Caso de uso CU02.*

Código	CU03
Nombre	Administrar “Almacenes”
Complejidad	Fácil
Descripción	Los oficinistas se encargarán de registrar, modificar, eliminar y consultar los almacenes de la empresa mediante una serie de pantallas. (Acciones *CRUD*).
Objetivo	Controlar los almacenes que dispone la empresa.
Actor principal	Oficinistas
Actores secundarios	-
Requisitos previos	-
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Listados”. 4. Elegirá la categoría de “Almacenes”. 5. Escogerá la acción que desee realizar con los almacenes, siguiendo el proceso lógico del sistema. 6. Confirmará las acciones perpetradas.

Cuadro 3.3: *Caso de uso CU03.*

Código	CU04
Nombre	Administrar “Materias primas”
Complejidad	Fácil
Descripción	Los oficinistas se encargarán de registrar, modificar, eliminar y consultar las materias primas de la empresa mediante una serie de pantallas. (Acciones *CRUD*).
Objetivo	Gestionar los tipos de materias primas con las que trabaja la empresa.
Actor principal	Oficinistas
Actores secundarios	-
Requisitos previos	-
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Listados”. 4. Elegirá la categoría de “Materias primas”. 5. Escogerá la acción que desee realizar con las materias primas, siguiendo el proceso lógico del sistema. 6. Confirmará las acciones perpetradas.

Cuadro 3.4: *Caso de uso CU04.*

Código	CU05
Nombre	Administrar “Productos finales”
Complejidad	Fácil
Descripción	Los oficinistas se encargarán de registrar, modificar, eliminar y consultar los productos finales de la empresa mediante una serie de pantallas. (Acciones *CRUD*).
Objetivo	Gestionar los tipos de borra que produce la empresa.
Actor principal	Oficinistas
Actores secundarios	-
Requisitos previos	-
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Listados”. 4. Elegirá la categoría de “Productos finales”. 5. Escogerá la acción que desee realizar con los productos finales, siguiendo el proceso lógico del sistema. 6. Confirmará las acciones perpetradas.

Cuadro 3.5: *Caso de uso CU05.*

Código	CU06
Nombre	Gestionar “Máquinas”
Complejidad	Fácil
Descripción	Los oficinistas se encargarán de registrar, modificar, eliminar y consultar las máquinas de la empresa mediante una serie de pantallas. (Acciones *CRUD*).
Objetivo	Agregar y consultar las máquinas que tiene la empresa.
Actor principal	Oficinistas
Actores secundarios	-
Requisitos previos	-
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Listados”. 4. Elegirá la categoría de “Máquinas”. 5. Escogerá la acción que desee realizar con las máquinas, siguiendo el proceso lógico del sistema. 6. Confirmará las acciones perpetradas.

Cuadro 3.6: *Caso de uso CU06.*

Código	CU07
Nombre	Administrar “Balas MP”
Complejidad	Medio
Descripción	Los jefes de turno se encargarán de registrar, modificar, eliminar y consultar las balas de materia prima que se recepcionan en la empresa mediante una serie de pantallas.
Objetivo	Registrar y manejar las balas de materia prima nuevas de la empresa.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber registrado el proveedor y las materias primas correspondientes de la bala.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Balas”. 4. Elegirá la categoría de “Balas de materia prima”. 5. Escogerá la acción que desee realizar con las balas de materia prima, siguiendo el proceso lógico del sistema. 6. Confirmará las acciones perpetradas.

Cuadro 3.7: *Caso de uso CU07.*

Código	CU08
Nombre	Controlar “Entradas”
Complejidad	Medio
Descripción	Los jefes de turno se encargarán de registrar, modificar, eliminar y consultar las entradas de balas de materia prima que se recepcionan mediante una serie de pantallas.
Objetivo	Registrar y manejar las entradas de balas de materia prima nuevas de la empresa.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber registrado el proveedor y las materias primas correspondientes de las balas que se reciben.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Entradas”. 4. Escogerá la acción que desee realizar con las entradas de balas de materia prima, siguiendo el proceso lógico del sistema. 5. Confirmará las acciones perpetradas.

Cuadro 3.8: *Caso de uso CU08.*

Código	CU09
Nombre	Administrar “Plantillas”
Complejidad	Medio
Descripción	Los jefes de turno se encargarán de registrar, modificar, eliminar y consultar las plantillas de las “Mezcladas”que se seguirán en el proceso productivo, mediante una serie de pantallas.
Objetivo	Crear y administrar las “Plantillas”de las “Mezcladas”que se seguirán en la empresa.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber registrado las materias primas que se usarán en la “Plantilla”.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Mezcladas”. 4. Elegirá la opción de “Plantillas”. 5. Escogerá la acción que desee realizar con las plantillas, siguiendo el proceso lógico del sistema. 6. Confirmará las acciones perpetradas.

Cuadro 3.9: *Caso de uso CU09.*

Código	CU10
Nombre	Gestionar “Mezcladas”
Complejidad	Medio/Difícil
Descripción	Los jefes de turno se encargarán de registrar, modificar, eliminar y consultar las órdenes de fabricación de las “Mezcladas”que se producirán en el proceso productivo, mediante una serie de pantallas.
Objetivo	Generar y administrar las órdenes de fabricación de las “Mezcladas”que se van a producir.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber creado la “Plantilla” correspondiente y haber registrado las máquinas que la llevarán a cabo.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Mezcladas”. 4. Elegirá la opción de “Órdenes de fabricación”. 5. Escogerá la acción que desee realizar con las plantillas, siguiendo el proceso lógico del sistema. 6. Asignará una “Plantilla” y las máquinas donde se procesará, a la orden de fabricación. 7. Confirmará las acciones perpetradas.

Cuadro 3.10: *Caso de uso CU10.*

Código	CU11
Nombre	Listar “Balas MP” y consultar detalles
Complejidad	Fácil
Descripción	Los jefes de turno podrán realizar consultas de datos mediante listados filtrados de datos y acceder a la información de la “bala MP”.
Objetivo	Consultar detalles de las “balas MP”de forma sencilla.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber creado alguna “bala MP”.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Balas”. 4. Elegirá la opción de “Balas de materia prima”. 5. Escogerá los filtros de búsqueda. 6. Se le mostrará un listado filtrado de datos y seleccionará la “bala MP”que desee consultar. 7. Se le mostrarán los detalles de la bala.

Cuadro 3.11: *Caso de uso CU11.*

Código	CU12
Nombre	Listar “Balas MF” y consultar detalles
Complejidad	Fácil
Descripción	Los jefes de turno podrán realizar consultas de datos mediante listados filtrados de datos y acceder a la información de la “bala MF”.
Objetivo	Consultar detalles de las “balas MF”de forma sencilla.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber generado alguna “bala MF”.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Balas”. 4. Elegirá la opción de “Balas de materia final”. 5. Escogerá los filtros de búsqueda. 6. Se le mostrará un listado filtrado de datos y seleccionará la “bala MF”que desee consultar. 7. Se le mostrarán los detalles de la bala.

Cuadro 3.12: *Caso de uso CU12.*

Código	CU13
Nombre	Listar “Plantillas” y consultar detalles
Complejidad	Fácil
Descripción	Los jefes de turno podrán realizar consultas de datos mediante listados filtrados de datos y acceder a la información de la “Plantilla”.
Objetivo	Consultar detalles de las “Plantillas”de forma sencilla.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber creado alguna “Plantilla”.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Mezclada”. 4. Elegirá la opción de “Plantillas”. 5. Escogerá los filtros de búsqueda. 6. Se le mostrará un listado filtrado de datos y seleccionará la “Plantilla”que desee consultar. 7. Se le mostrarán los detalles de la “Plantilla”.

Cuadro 3.13: *Caso de uso CU13.*

Código	CU14
Nombre	Listar “Mezcladas” y consultar detalles
Complejidad	Fácil
Descripción	Los jefes de turno podrán realizar consultas de datos mediante listados filtrados de datos y acceder a la información de la “Mezclada”.
Objetivo	Consultar detalles de las “Mezcladas”de forma sencilla obteniendo información de las órdenes de fabricación y de las partidas que genera.
Actor principal	Jefes de turno
Actores secundarios	Oficinistas
Requisitos previos	Haber creado alguna “Mezclada”.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Mezclada”. 4. Elegirá la opción de “Órdenes de fabricación”. 5. Escogerá los filtros de búsqueda. 6. Se le mostrará un listado filtrado de datos y seleccionará la “Mezclada”que desee consultar. 7. Se le mostrarán los detalles de la “Mezclada”.

Cuadro 3.14: *Caso de uso CU14.*

Código	CU15
Nombre	Consultar detalles de las balas
Complejidad	Fácil
Descripción	Los carretilleros podrán realizar consultas de datos de las balas introduciendo el código de la barra que aparece en su etiqueta.
Objetivo	Obtener información de las balas de forma rápida y eficaz.
Actor principal	Carretilleros
Actores secundarios	Jefes de turno y Oficinistas
Requisitos previos	La bala de MP o de MF debe de existir.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de "Vista trabajadores". 4. Elegirá la opción de "Información de bala". 5. Introducirá el código de barras. 6. Se le mostrarán los detalles de la bala.

Cuadro 3.15: *Caso de uso CU15.*

Código	CU16
Nombre	Almacenar balas
Complejidad	Medio
Descripción	Los carretilleros a la hora de transportar las balas, asignarán el almacén donde se almacene.
Objetivo	Asignar una posición de almacén a la bala.
Actor principal	Carretilleros
Actores secundarios	Jefes de turno y Oficinistas
Requisitos previos	La bala de MP o de MF debe de existir y el almacén debe estar registrado.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de "Vista trabajadores". 4. Elegirá la opción de "Almacenar bala". 5. Introducirá el código de barras. 6. Seleccionará el almacén donde se almacene. 7. Confirmará los cambios perpetrados.

Cuadro 3.16: *Caso de uso CU16.*

Código	CU17
Nombre	Procesar “Mezclada”
Complejidad	Difícil
Descripción	Los cargadores a la hora de producir las balas, indicarán la orden de fabricación a seguir y seleccionará los pesos de las “balas de MF”que se han producido dando lugar a la “Partida”.
Objetivo	Crear las “balas de MF”que se han producido.
Actor principal	Cargadores
Actores secundarios	Jefes de turno y Oficinistas
Requisitos previos	Haber creado la “Mezclada” y haberle asignado las “balas MP” que se consumirán.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Vista trabajadores”. 4. Elegirá la opción de “Salida de máquinas”. 5. Seleccionará el código de máquina. 6. Seleccionará la “Mezclada” a procesar. 7. Seleccionará el almacén donde se almacene. 8. Indicará los pesos de las “balas de MF” que se han generado. 9. Terminará el proceso de producción.

Cuadro 3.17: *Caso de uso CU17.*

Código	CU18
Nombre	Asignar balas a “Mezclada”
Complejidad	Medio
Descripción	Los cargadores a la hora de producir las “balas de MF”, seleccionará las balas que se usarán en la “Mezclada”siguiendo los ingredientes de la “Plantilla”.
Objetivo	Crear las “balas de MF”que se han producido.
Actor principal	Cargadores
Actores secundarios	Jefes de turno y Oficinistas
Requisitos previos	Haber creado la “Mezclada” y las balas de “balas MP”que se pretenden usar.
Secuencia normal	1. Accederá a la *URL* correspondiente de la aplicación. 2. Iniciará sesión dentro de la aplicación. 3. Accederá al apartado de “Mezcladas”. 4. Elegirá la opción de “Órdenes de producción”. 5. Seleccionará la “Mezclada” a procesar. 6. Editará los datos de la “Mezclada”. 7. Seleccionará el ingrediente de la “Plantilla”. 8. Indicarán los códigos de las balas a consumir 9. Confirmará los cambios perpetrados.

Cuadro 3.18: *Caso de uso CU18.*

3.2. Diseño de la arquitectura del sistema

La arquitectura que se ha utilizado es una arquitectura basada en el patrón de diseño ***MVC*** **(Modelo-Vista-Controlador**). Este patrón de diseño representa la comunicación entre los datos del modelo, las acciones que realiza el controlador y la vista que es proporcionada al cliente.

La comunicación sobre el sistema está desarrollada mediante la **tecnología Cliente-Servidor** donde diferenciamos el servidor que será el encargado de gestionar los datos y las peticiones, y un cliente que es el encargado de generar los datos a través del usuario y entregárselos al servidor. Se muestra a continuación una definición gráfica del sistema donde se combina el **patrón de diseño *MVC*** y la **arquitectura *Cliente-Servidor***. [16] [17]

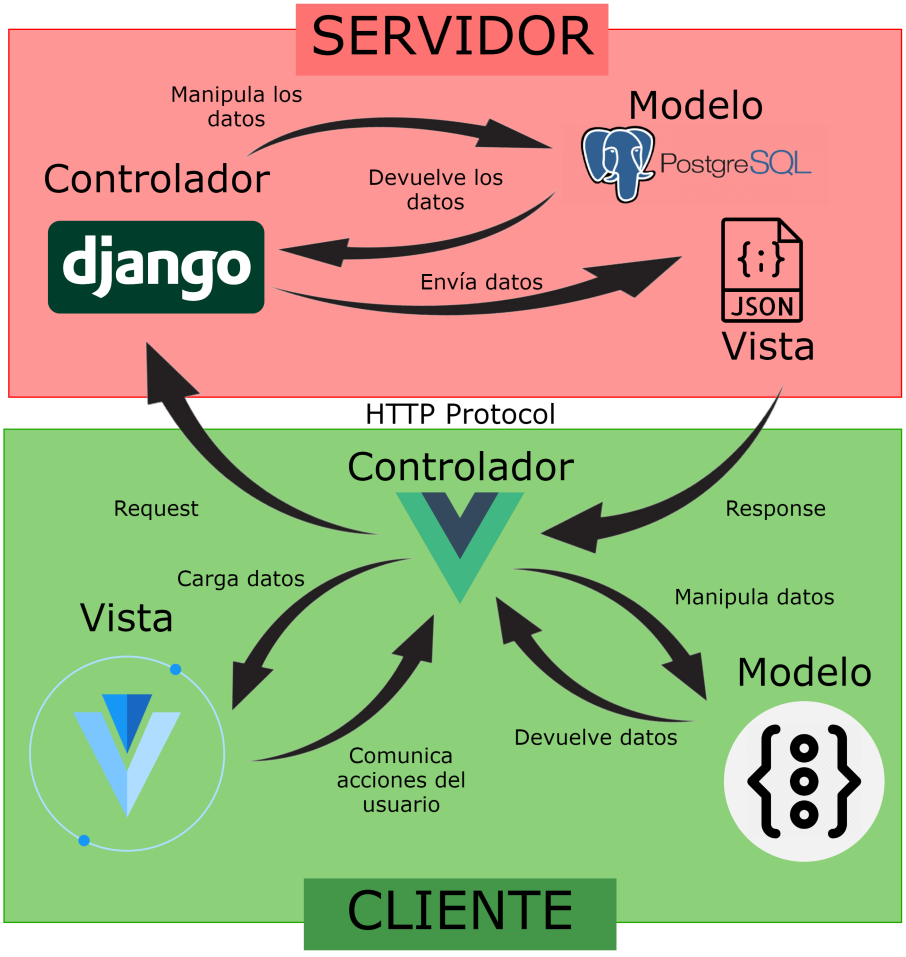

Figura 3.2: *Esquema de funcionamiento del sistema.*

Tal y como se puede apreciar en la Figura 3.2, tanto el cliente, que sería la aplicación web, como el servidor, que sería la API con su debida base de datos, están implementadas internamente con el patrón *MCV*.

La comunicación entre cliente y servidor es a través del protocolo *HTTP*, donde el cliente a través de del método *HTTP* correspondiente accede a la *URL* donde el servidor está proporcionando un servicio. En este caso dicha comunicación es iniciada a través del cliente que accede al *endpoint* deseado y el servidor le proporciona una respuesta correspondiente.

Dichas *URL*s son proporcionadas por los distintos *endpoints* del sistema y cada una realiza una serie de acciones sobre los datos de la aplicación a razón de la funcionalidad que tenga implementada. *Django REST framework* como comentamos anteriormente, es una herramienta muy potente para crear los *endpoints*, este *freamework* realiza automaticamente los endpoints correspondientes a las acciones ***CRUD****(Create, Retrieve, Update y Delete)* de los modelos, teniendo únicamente que implementar las variaciones que queramos hacer y ciertos métodos propios para nuestro sistema. Por ello, a continuación se presenta un ejemplo de estos. En la Figura 3.3 se muestra la definición un método propio del funcionamiento del sistema mediante la herramienta *online* de documentación de *APIs* ***SwaggerUI***.

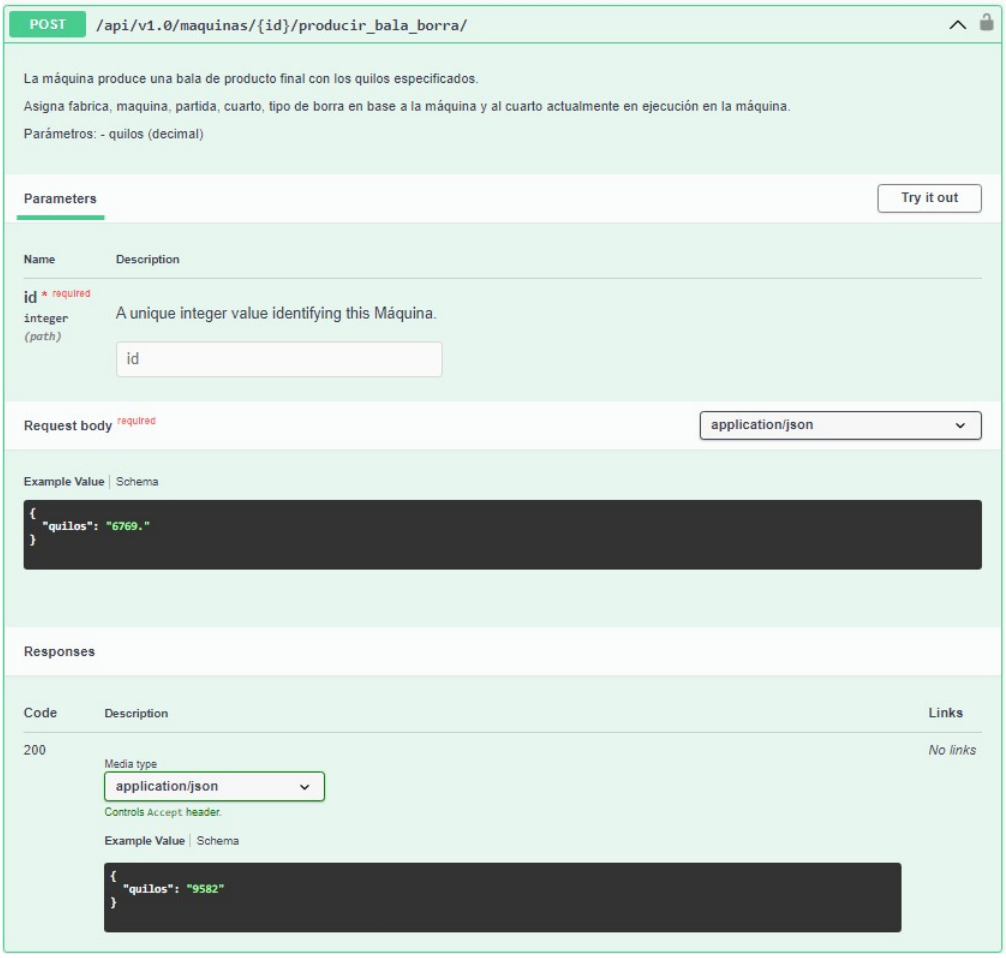

Figura 3.3: *Definición de método del API propio.*

Diseño de la base de datos

Se presenta a continuación el diseño conceptual de la base de datos donde se detallan los distintos atributos de las tablas, así como las relaciones entre estas cumpliendo las pautas del modelo relacional. Cabe destacar que solo se exponen las tablas correspondientes a la parte del proyecto que me ha sido asignada.

Separado por distintos colores se definen las distintas áreas de la aplicación, siendo la parte de Mezcladas la más compleja tanto a nivel informático como a nivel conceptual dentro del proceso productivo de la empresa. Sin entrar en detalles, podemos ver que las *Mezcladas* son el centro de la aplicación ya que se comunican con las demás áreas funcionales de la aplicación siendo esta compuesta por las tablas correspondientes a las *Plantillas* y a las *Órdenes de fabricación*.

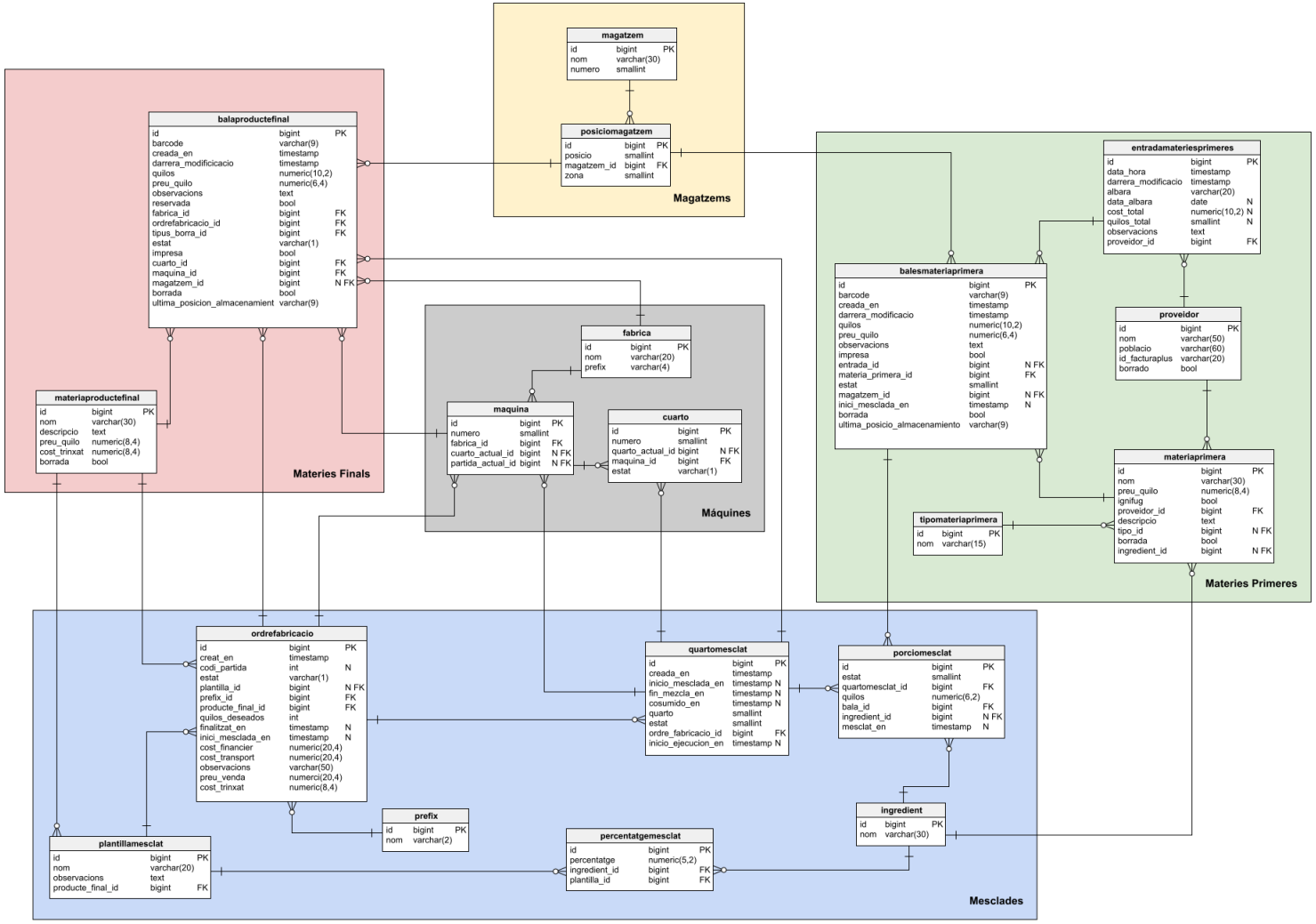

Figura 3.4: *Esquema conceptual de la base de datos.*

3.3. Diseño de la interfaz

Una vez tengamos diseñada nuestra *API* con sus respectivos *endpoints*, pasamos al diseño de la aplicación *web* que va a hacer uso de esta. Tras las consultas realizadas a los operarios de la fábrica, todos coincidían en un aspecto, no son expertos en las nuevas tecnologías.

El perfil promedio de usuario es el de un empleado de edad media comprendida entre los treinta y los sesenta años y con gran rodaje en la empresa, son personas que conocen en todo momento las materias primas que hay, los distintos tipos de producto que se producen, etc. Con esto vemos que están muy familiarizados con el proceso productivo y con los datos que aparecerán en las distintas interfaces de usuario.

Debido a que estas no deben de ser complejas, se ha optado por pantallas interconectadas, visuales e intuitivas, con distinción de tipos de datos por colores, un menú simple y mensajes de *feedback*.

A continuación se presentan los prototipos prediseñados que fueron presentados a la empresa para obtener una valoración de estos. Se muestran los *mockups* correspondientes a la ventana de listado de balas y al formulario de creación de balas.

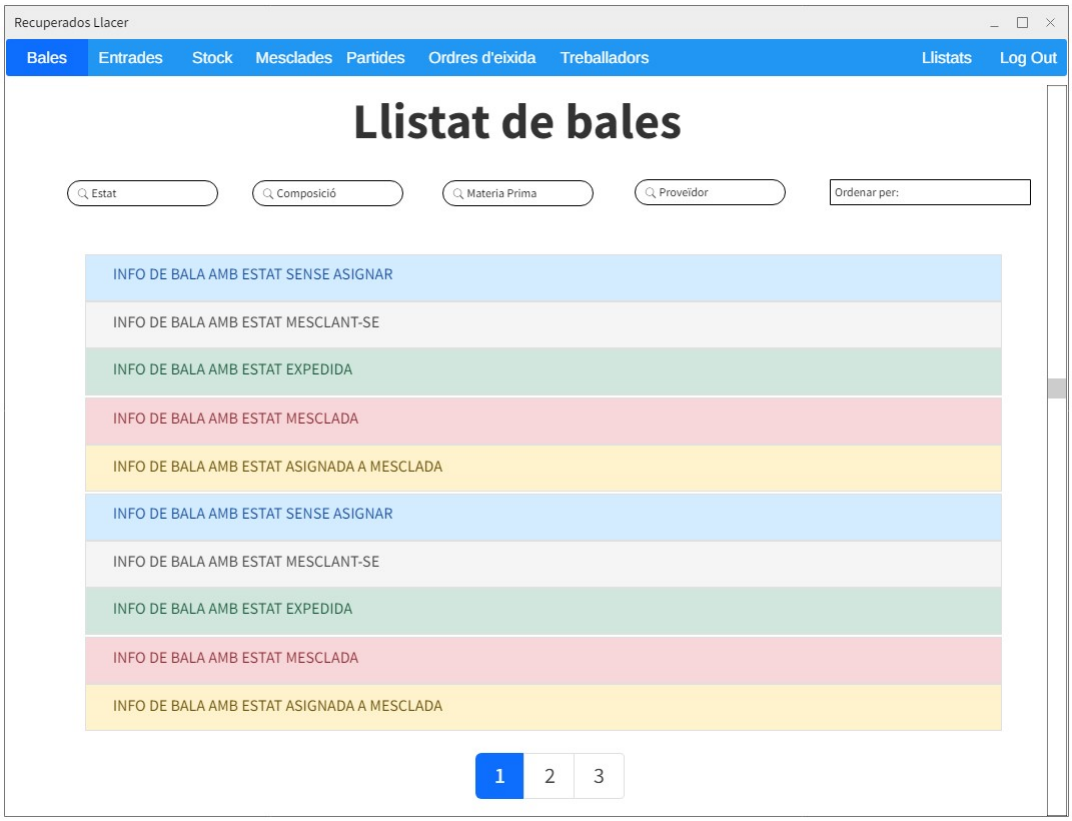

Figura 3.5: *Mockup de la pantalla de listado de balas.*

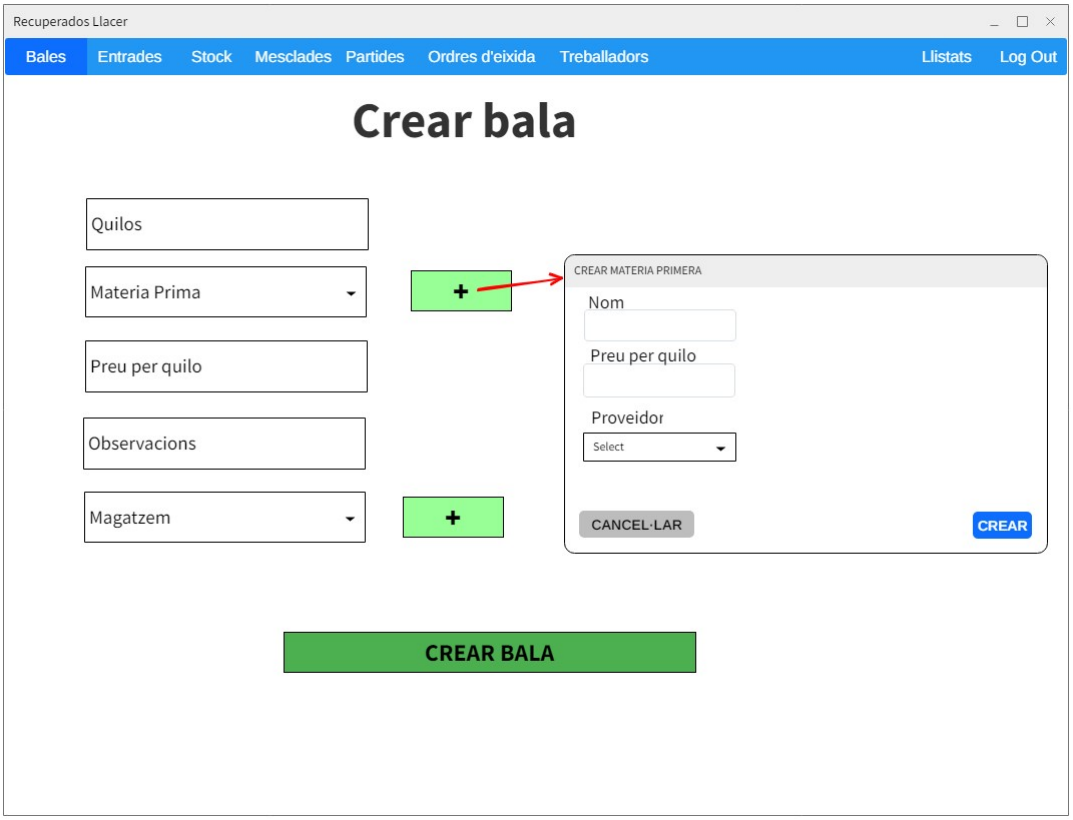

Figura 3.6: *Mockup de la pantalla de creación de balas.*

Para las pantallas referidas a las vistas de operarios de planta se presentan prototipos con una interfaz similar a las implementadas en las máquinas. Esta interfaz muestra distintos datos del proceso de transformación de forma muy visual para los operarios. Véase en la siguiente figura.

Figura 3.7: *Mockup de la pantalla de producción de Mezclada.*

Guía de estilos

Los colores que se han optado son los correspondientes a la biblioteca de *Bootstrap5* con sus respectivas modificaciones de tonalidades. [18] Son colores que aportan sencillez y limpieza al diseño, esto hace que las pantallas sean lo más simples posibles. Se ha optado por añadir funcionalidades de filtrado y ordenación que dotan de agilización a los trabajadores a la hora de consultar datos. Se consideran grandes espacios en blanco entre los datos para que sea mucho más visible la información. Los idiomas que se han optado por petición de la empresa son el catalán y el castellano debido a la localización geográfica de la empresa y por aspectos socio-culturales internos de la empresa.

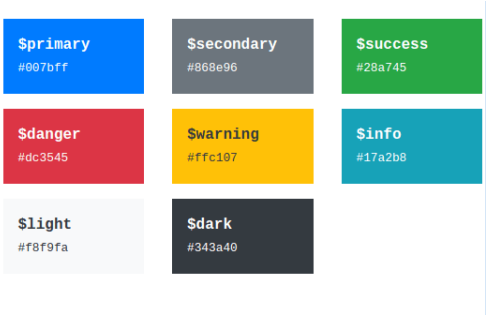

Figura 3.8: *Paleta de colores utilizada.*

En cuanto a la funcionalidad de las interfaces, estas son totalmente dinámicas donde se apreciarán los cambios que se realicen en ellas, además se muestra *feedback* a los usuarios a través de mensajes tanto de error como de notificaciones positivas.

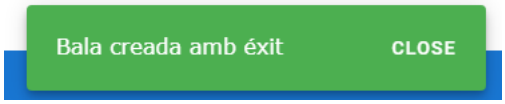

Figura 3.9: *Notificación de éxito utilizada.*

Como se muestra en la Figura 3.6, se han diseñado pantallas con paneles flotantes para ayudar al usuario a crear datos relacionados sin tener que cambiar de proceso. Esto reduce el riesgo de fallos ya que está controlado de tal forma que impida poder realizar ciertas acciones inválidas. Además, se pide doble factor de autorización a la hora de eliminar datos, finalizar procesos o perpetrar acciones que comporten importantes modificaciones, es decir, se muestran ventanas emergentes donde el usuario deberá de autorizar la acción.

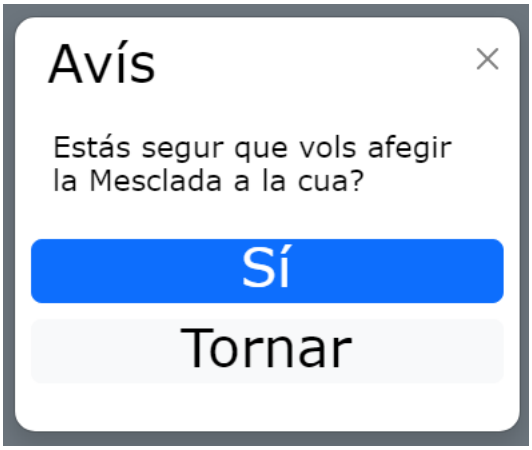

Figura 3.10: *Sistema de verificación.*

Sitemap

Se presenta a continuación el diseño conceptual de la arquitectura de la aplicación *web* donde se puede apreciar el listado de pantallas de la aplicación conjuntamente con la intercomunicación entre estas.

Mediante este diseño se pretende en todo momento guiar al usuario final para que este no se pierda durante la navegación sobre la aplicación. Se ha intenta conseguir a través de este diseño distribuir la aplicación mediante las funcionalidades de la aplicación, es por ello por lo que las acciones se ven agrupadas según las distintas partes de la aplicación ordenándolas cronológicamente según el proceso de producción.

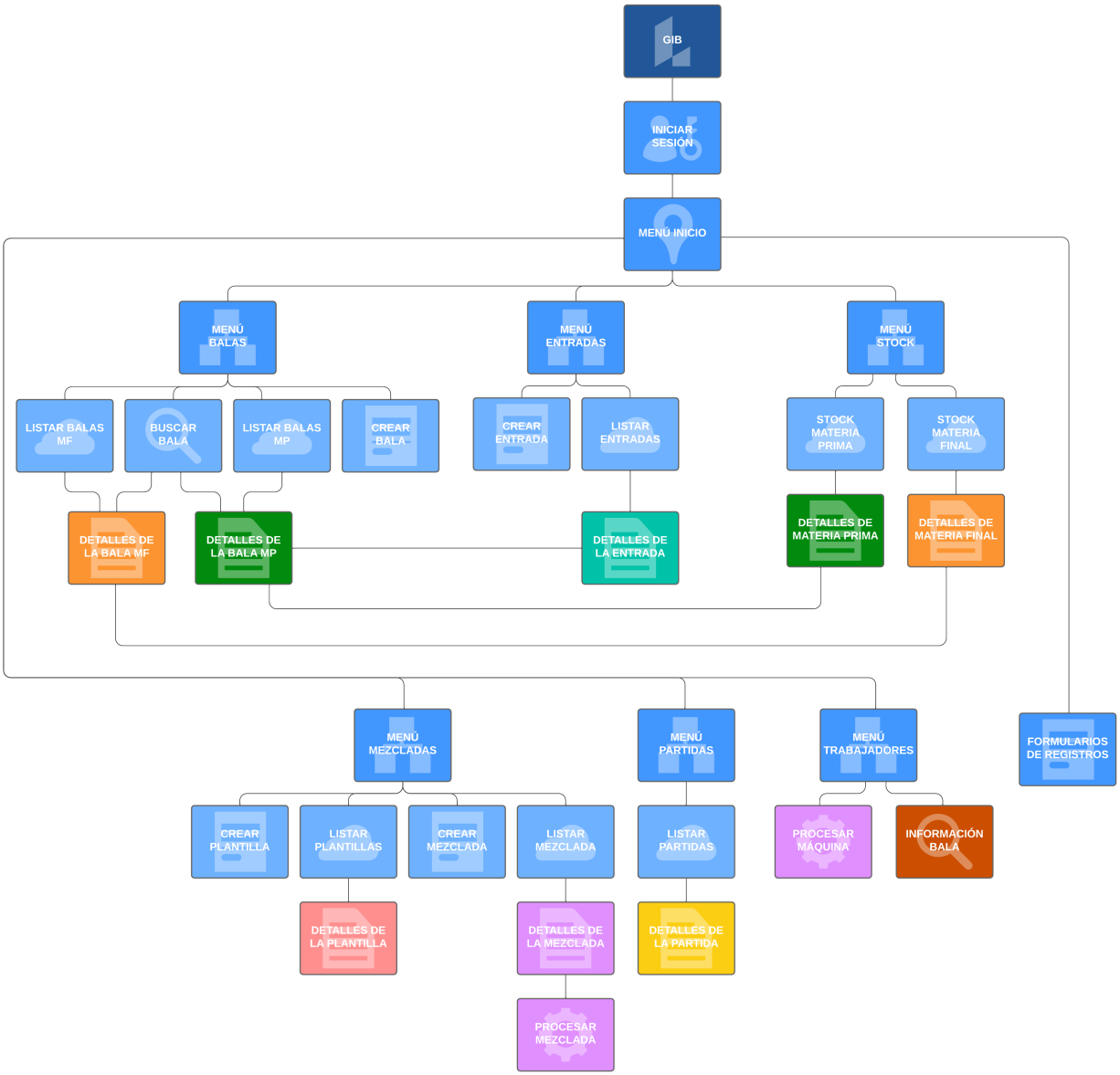

Figura 3.11: *Sitemap de la aplicación.*

Capítulo 4

Implementación y pruebas

4.1. Detalles de implementación

Durante esta sección abordaremos los temas relacionados con la implementación del proyecto donde se verán cuestiones referentes tanto al *backend* como al *frontend*. Trataremos aspectos tales como los patrones de implementación que se han seguido a la hora de crear código o la adaptación de la aplicación a los *frameworks* de desarrollo correspondientes.

4.1.1. Backend

Django REST Framework

Como ya hemos comentado en distintos puntos de este documento, el proyecto de *backend* ha sido desarrollado mediante la tecnología del *framework Django REST framework*. Este, al igual que muchos *frameworks* orientados a la creación de *API*s, cuentan con una estructura de diseño *MCV* el cual usa internamente el patón de diseño de inyección dependencias a la hora de comunicar las distintas clases de objetos de la aplicación.

Django trabaja a través de las micro aplicaciones internas registradas en la aplicación correspondientes, en nuestro caso, a las distintas áreas funcionales de la empresa. Es decir, las funcionalidades están separadas en pequeñas aplicaciones con patrones de diseño *MVC*. Para cada una de estas se genera un directorio con los distintos ficheros de configuración que deben de ser implementados.

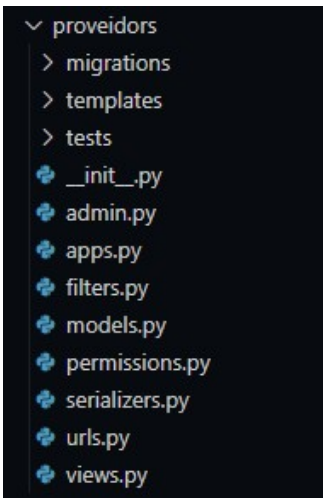

Figura 4.1: *Ficheros de configuración de Django.*

Tal y como se muestra en la "Figura 4.1", vemos un listado de los ficheros de configuración que deben de ser completados para el correcto funcionamiento del sistema. Cabe destacar que no en todos los casos, es necesario que existan algunos de estos, como por ejemplo "*filters.py*". Cada uno de estos alberga configuraciones distintas que a continuación veremos en detalle.

- El **fichero "*__init__.py*"** convierte un directorio en un módulo de la aplicación simplemente estando contenido en dicho directorio, no hace falta que tenga implementado código. [19]
- El fichero ***admin.py*** es el encargado de configurar el área de administración de la aplicación propia de *Django*, este área cuenta con un gestor de los distintos modelos del sistema, donde podemos encontrar tanto listados de datos, como formularios para tratar los modelos de forma manual desde el *backend*. [20]

  Mediante este fichero podemos configurar aspectos como qué datos se quieren mostrar, la ordenación de los datos o la forma de visualización. A continuación, en la Figura 4.2 se muestra un ejemplo de código implementado en la aplicación correspondientes al administrador de las "*Entradas*"de balas.

```python
@admin.register(EntradaMateriesPrimeres)
class EntradaAdmin(admin.ModelAdmin["EntradaMateriesPrimeres"]):
    list_display = (
        "id",
        "darrera_modificacio",
        "data_hora",
        "albara",
        "data_albara",
        "proveidor",
        "quilos_total",
        "cost_total",
        "num_bales",
    )
    list_filter = ("proveidor", "data_albara")
    ordering = ("-darrera_modificacio",)
    date_hierarchy = "data_hora"
    search_fields = ["proveidor", "data_albara"]
    preserve_filters = True
    inlines = [
        BalaMateriaPrimeraInline,
    ]

    def get_queryset(self, request: "HttpRequest") -> QuerySet["EntradaWithNumBales"]:
        queryset = super().get_queryset(request).select_related("proveidor")
        annotated_queryset: QuerySet["EntradaWithNumBales"] = queryset.annotate(
            num_bales=Count("bales", distinct=True)
        )

        return annotated_queryset

    def num_bales(self, entrada: "EntradaWithNumBales") -> int:
        return entrada.num_bales

    num_bales.short_description = "# Bales"
    num_bales.admin_order_field = "num_bales"
```

Figura 4.2: *Ejemplo de implementación del administrador de "Entradas".*

Como se puede apreciar, mediante esta implementación se registrará la aplicación en el administrador donde se mostrarán los campos de la variable `list_display` según ciertos aspectos indicados posteriormente. Algunos de estos campos son computados como `num_bales` o son referencias a otros modelos como `proveidor`.

- Mediante el fichero ***apps.py*** se permite configurar aspectos de las aplicaciones tales como el nombre o la ruta mediante la subclase *AppConfig*. Este fichero es leído durante la ejecución y carga la configuración detallada. [21]
- ***filters.py*** es un fichero que, como su nombre indica, alberga los métodos de filtraje de datos según ciertos aspectos. Este fichero trabaja sobre la consulta de datos del modelo y realiza las comparaciones pertinentes. En la Figura 4.3 se presenta el filtrado de "Almacenes"según dos campos del modelo.

```python
class MagatzemFilter(filters.FilterSet):  # type: ignore
    zona = filters.NumberFilter(field_name="posicions__zona", label="Zona")
    posicio = filters.NumberFilter(field_name="posicions__posicio", label="Posició")

    class Meta:
        model = Magatzem
        fields = ("nom", "numero", "zona", "posicio")
```

Figura 4.3: *Ejemplo de implementación del filtro de "Almacenes".*

- Uno de los fichero más importantes es el fichero ***models.py***. Este concretamente se encarga de gestionar el modelo de la aplicación. En él, se recoge la definición del modelo indicando los atributos que tendrá, los tipos de datos y la implementación de los métodos que actúan sobre estos. Es este el fichero encargado de generar la base datos, con sus respectivas tablas y relaciones. Cabe destacar que dentro de una misma aplicación puede que hayan distintos modelos de datos definidos, es el caso de la Figura 4.5, donde se generan los modelos de datos correspondientes a las "Balas MP" y a las "Balas MF" a través de una herencia procedente de "*BalaBase*".

```python
class BalaBase(models.Model):
    barcode = models.CharField(unique=True, max_length=9)
    creada_en = models.DateTimeField(auto_now_add=True)
    darrera_modificacio = models.DateTimeField(auto_now=True)
    quilos = models.DecimalField(max_digits=10, decimal_places=2)
    preu_quilo = models.DecimalField(
        max_digits=6,
        decimal_places=4,
        verbose_name="Preu per quilo",
        blank=True,
        default=0,
        validators=(MinValueValidator(0),),
    )
    observacions = models.TextField(
        blank=True,
    )
    impresa = models.BooleanField(default=False)
    # Para detectar borrados con reversion
    borrada = models.BooleanField(default=False)
    ultima_posicion_almacenamiento = models.CharField(
        max_length=9, blank=True, verbose_name="Última posición de almacenamiento"
    )

    class Meta:
        abstract = True

    def __str__(self) -> str:
        return self.barcode
```

Figura 4.4: *Implementación del modelo de las "BalaBase".*

```python
class BalaMateriaPrimera(BalaBase):
    """
    Bala de materia prima
    """

    class Estat(models.IntegerChoices):
        """
        SIN_ASIGNAR: Estado inicial de la bala de materia primera.
        ASIGNADA_A_MEZCLADA: La bala ha sido asignada a una mezclada.
        MEZCLANDO: La bala ha sido rajada en un cuarto pero aún no se han
            consumido todas sus porciones de la mezclada.
        MEZCLADA: Todas las porciones de mezclada de la bala han sido
            consumidas.
        """

        SIN_ASIGNAR = 1, "SIN ASIGNAR"
        ASIGNADA_A_MEZCLADA = 2, "ASIGNADA A MEZCLADA"
        MEZCLANDO = 3, "MEZCLANDO (parcialmente mezclada)"
        MEZCLADA = 4, "MEZCLADA"
        ASIGNADA_A_EIXIDA = 5, "ASIGNADA A ORDRE d'EIXIDA"
        EXPEDIDA = 6, "Expedida"

    estat = models.PositiveSmallIntegerField(
        choices=Estat.choices,
        default=Estat.SIN_ASIGNAR,
    )
    inici_mesclada_en = models.DateTimeField(null=True, blank=True)
    materia_primera = models.ForeignKey(
        MateriaPrimera,
        related_name="bales",
        on_delete=models.PROTECT,
        verbose_name="Matèria primera",
    )
    entrada = models.ForeignKey(
        EntradaMateriesPrimeres,
        related_name="bales",
        on_delete=models.CASCADE,
        blank=True,
        null=True,
    )
```

```python
class BalaProducteFinal(BalaBase):
    """Bala de borra"""

    class Estat(models.TextChoices):
        """Estado de la BalaProducteFinal"""

        CREADA = "c", "Creada"
        ALMACENADA = "m", "Almacenada"
        ASIGNADA = "a", "Asignada a ordre d'eixida"
        EXPEDIDA = "e", "Expedida"

    estat = models.CharField(
        max_length=1,
        choices=Estat.choices,
        default=Estat.CREADA,
    )
    tipus_borra = models.ForeignKey(
        MateriaProducteFinal,
        on_delete=models.PROTECT,
        related_name="bales",
        verbose_name="Tipus de Borra",
    )
    fabrica = models.ForeignKey(
        "fabriques.Fabrica",
        related_name="bales_productes_finals",
        on_delete=models.PROTECT,
    )
    maquina = models.ForeignKey(
        "fabriques.Maquina",
        related_name="bales_productes_finals",
        on_delete=models.PROTECT,
    )
    magatzem = models.ForeignKey(
        "magatzems.PosicioMagatzem",
        related_name="bales_productes_finals",
        on_delete=models.PROTECT,
        blank=True,
        null=True,
    )
```

Figura 4.5: *Parte de la implementación del modelo de las balas.*

Tal y como se ha citado, los métodos que en este fichero se implementan, son los que se encargan de realizar acciones en los modelos, es decir, los que modifican la base de datos. Como ejemplo se muestra a continuación en la Figura 4.6 el método encargado de asignar una posición de almacén a una bala, para ello, debe de modificar los datos de la bala.

```python
def almacenar(self, magatzem: "PosicioMagatzem", save: bool = True) -> None:
    """Almacena la bala en el magatzem"""

    if self.magatzem is not None:
        raise AssertionError(
            f"{self.log_prefix} Almacenar bala: La bala ya está almacenada"
        )

    if magatzem is None:
        raise AssertionError(f"{self.log_prefix} Almacenar bala: Almacén no válido")
    self.magatzem = magatzem
    self.ultima_posicion_almacenamiento = magatzem.barcode
    update_fields = ["magatzem", "ultima_posicion_almacenamiento"]

    if isinstance(self, BalaProducteFinal):
        setattr(self, "estat", BalaProducteFinal.Estat.ALMACENADA)
        update_fields.append("estat")

    if save:
        self.save(update_fields=update_fields)
    logger.info("%s Bala almacenada en %s", self.log_prefix, magatzem)
```

Figura 4.6: *Implementación del método "almacenar".*

- El fichero "***permissions.py***" se encarga de gestionar los permisos de acceso a las vistas por parte de los usuarios mediante las peticiones *HTTP*.
- Otro fichero relevante es el fichero ***"serializers.py"***. Este fichero se encarga de codificar y de decodificar los datos entre el formato nativo de *Python* y el formato de comunicación *JSON*. Mediante la implementación de este, se consiguen realizar conjuntos de datos que son sobre los cuales se trabaja, es decir, son el formato de las microconsultas que se realizan para poder tratarlas. A estas microconsultas se les llama ***queryset*** y son el corazón del tratamiento de los datos. A continuación se muestra un ejemplo (Figura 4.7)de este implementado en el proyecto para gestionar las "Materias Primas" donde se muestran los campos a ser codificados para ser tratados.

```python
class MateriaPrimeraSerializer(MateriaPrimeraModelSerializer):
    nom_ingredient = serializers.SerializerMethodField()
    codigo_grs = serializers.SerializerMethodField()
    quilos = serializers.SerializerMethodField()
    quilos_asignados = serializers.SerializerMethodField()
    quilos_sin_asignar = serializers.SerializerMethodField()

    class Meta:
        model = MateriaPrimera
        fields = (
            "id",
            "nom",
            "proveidor",
            "ingredient",
            "nom_ingredient",
            "preu_quilo",
            "ignifug",
            "tipo",
            "codigo_grs",
            "num_bales",
            "quilos",
            "quilos_asignados",
            "quilos_sin_asignar",
        )
        read_only_fields = ("id", "num_bales", "nom_ingredient")

    def get_nom_ingredient(self, materia_primera: "MateriaPrimera") -> str:
        return materia_primera.ingredient.nom if materia_primera.ingredient else ""

    def get_codigo_grs(self, materia_primera: "MateriaPrimera") -> str:
        return materia_primera.tipo.nom if materia_primera.tipo else ""

    def get_quilos(self, materia_primera: "MateriaPrimera") -> float:
        return float(materia_primera.quilos)

    def get_quilos_asignados(self, materia_primera: "MateriaPrimera") -> float:
        return float(materia_primera.quilos_asignados)

    def get_quilos_sin_asignar(self, materia_primera: "MateriaPrimera") -> float:
        return float(materia_primera.quilos_sin_asignar)
```

Figura 4.7: *Implementación del serializador de "Materias Primas".*

- Siguiendo con la lista de ficheros, encontramos el fichero "***urls.py***". Este es realmente importante ya que es el encargado de definir y registrar las *URL*s de los *endpoints* de la ***Router*** de la aplicación. Un aspecto muy potente de *Django* es que no hace falta definir los *endponits* como tal, simplemente basta con registrar los servicios implementados en el *Router*. Véase Figura 4.8.

```python
router = SimpleRouter()
router.register("proveidors", ProveidorViewSet, basename="proveidor")
```

Figura 4.8: *Registro de los endpoints correspondientes a "Proveedores".*

- Por último, pero no por ello menos importante, encontramos el fichero "***views.py***" el cual podemos decir que es encargado de aportar funcionalidad a la aplicación. Este fichero contiene los métodos que se esconden tras las *URL*s del *API*, es decir, es la encargada de gestionar las peticiones *HTTP* y realizar las acciones internas pertinentes. Otro aspecto muy poderoso de *Django* es que emite las acciones *CRUD* básicas de los modelos, por ello, únicamente se implementan los métodos suplementarios o la sobreescritura de los CRUD por si se desea modificar algún matiz.

  La implementación de estos métodos se realiza mediante la clase *ModelViewSet* que es la encargada de conseguir los *querysets* y realizar las distintas acciones sobre estos. Una vez la funcionalidad ha sido ejecutada, estos devuelven la respuesta *HTTP* al cliente con los datos y cabeceras correspondientes. En la Figura 4.9 se muestra la implementación de un método de consulta de datos filtrada sobre las "Balas MP" donde se combina tanto las clases de filtrado previamente mencionadas, con la codificación de los serializadores y con la funcionalidad *HTTP*.

```python
@action(detail=False, name="Balas Asignadas y no consumidas", methods=["get"])
def get_bales_materies_primeres_asignadas_no_consumidas(
    self, request: "Request"
) -> "Response":
    queryset = BalaMateriaPrimera.objects.asignadas_no_consumidas()
    filtro = self.filterset_class(request.query_params, queryset)
    balas = filtro.qs
    serializer = BalaMateriaPrimeraSerializer(balas, many=True)

    return Response(serializer.data, status=status.HTTP_200_OK)
```

Figura 4.9: *Implementación de método de filtrado de "Balas MP".*

Una vez tengamos implementadas las funcionalidades, debemos registrarlas en la configuración general de la aplicación. En nuestro caso, la *API* está formada por estas microaplicaciones. Véase Figura 4.10.

```python
LOCAL_APPS: list[str] = [
    "proveidors",
    "fabriques",
    "magatzems",
    "materies",
    "bales",
    "entrades",
    "mesclades",
]
```

Figura 4.10: *Listado de las aplicaciones internas del API.*

La implementación del *backend* se ha realizado siguiendo estos ficheros de configuración propios de *Django* de cada una de las microaplicaciones definidas. Por ello, podemos decir que el patrón de diseño seguido es el *MVC* con las modificaciones necesarias para poder acoplarlo al *framework* en cuestión. [22]

Migraciones

Otro aspecto a tener en cuenta en la implementación del proyecto son las **migraciones**. Las migraciones son la forma en la que *Django* hace constancia de las modificaciones en los modelos. Cada vez que se modifica o se crea un modelo, se debe de generar y ejecutar la migración de los datos de tal forma que estos se vean reflejados en la base de datos.

En el caso de este proyecto, el software usado para gestionar la base de datos es *PostgreSQL* el cual es el *SGBD* que mayor compatibilidad presenta con Django, manteniendo un sistema de migraciones excelente. [23]

Seguridad

En cuanto a cuestiones relacionadas con la seguridad del sistema, esta se ha implementado mediante el sistema de autenticación de usuarios basado en el sistema ***JSON Web Token (JWT)***. Este ha sido instalado en el proyecto mediante el gestor de paquetes *pip* y va a servir a la *API* para gestionar las peticiones *HTTP* de los clientes. En ellas, tanto usuario y contraseña, viajan cifradas, según el protocolo ***HS256***, por la red encapsuladas en un objeto *JSON*. [24] [25]

En cuanto a la seguridad utilizada en los datos generados de formularios, se usa la tecnología ***CSRF*** para controlar la integridad y la confidencialidad de estos mediante librerías propias de *Django*.

Resultados

Una vez hayan sido implementados todos los métodos, se documentan de la siguiente forma. Python proporciona la librería `SpectacularSwaggerView` la cual cuenta con distintas herramientas tales como *SwaggerUI* que nos permite autogenerar de forma gráfica la documentación de nuestra *API* de tal forma que se muestren los métodos de esta. A su vez, lo hemos mapeado en la tabla de rutas de nuestra aplicación para que pueda ser accesible para la consulta de los usuarios. [26]

A continuación, en la Figura 4.11 se muestra un pequeño listado de algunos métodos implementados en el sistema y que aparecen documentados.

Figura 4.11: *Listado de métodos implementados vistos con SwaggerUI.*

4.1.2. Frontend

Vuejs

Vuejs es un *framework* de *JavaScript* idóneo para la realización e implementación de aplicaciones *web* para clientes de *API*s. La funcionalidad implementada está regida por diversos módulos externos que veremos en detalle como funcionan a continuación. Concretamente este proyecto se ha implementado usando *Vue 3*.

- ***Peticiones HTTP al API***. El cliente, a la hora de realizar las llamadas a la *API*, realiza peticiones *HTTP* seguras bajo el protocolo de seguridad *JWT*. Estas peticiones son realizadas asíncronamente mediante *Vue*. El cliente, al recibir la respuesta del servidor, accede a los datos de la respuesta y los interpreta según el código *HTTP* de la respuesta.

  Vemos a continuación en la Figura 4.12, el código que se encarga de solicitar a la API los datos referidos al listado de "balas MP".

```
getBales() {
  if (this.inspectToken(localStorage.getItem('jwt'))==false){return}
  const url_endpoint = `bales_materies_primeres/`

  return Vue.prototype.$http.get(url_endpoint)
    .then((response) => {
      return { result: response.data.results, count: response.data.count }
    })
    .catch(error => {
      if (error?.response?.status === 401)
        EventBus.$emit('dialogLogin',true);
      else
        console.debug(error)
    })
},
```

Figura 4.12: *Código correspondiente a la petición a la API.*

- ***Gestor de estados***. Una vez que se ha interpretado la respuesta, los datos pasan a almacenarse en el modelo. En esta etapa interviene el gestor de estados ***Vuex*** el cual se encarga mediante métodos *setters* y *getters* de almacenar y tratar el estado de las variables del programa de forma centralizada. Para ello, debemos declarar las variables que queramos, y mediante una serie de métodos, estas son almacenadas en el sistema. [27]

  A continuación podemos ver un ejemplo en el cual se muestra como se llama al método que comunica con el *API* y luego a *setter* de nombre `GET_BALES` y este lo almacena en la variable correspondiente del gestor de estados. Figura 4.13.

```
getBales({ commit }) {
  if(!localStorage.getItem('jwt')) {EventBus.$emit('dialogLogin',true); return}
  if(localStorage.getItem('jwt')) authAPI.getBales().then(bales =>
    {commit('GET_BALES', bales)} )
},

GET_BALES(state, data) {
  state.bales = data.result
  state.bales_count = data.count
  state.bales_pagines = Math.ceil(data.count / 20)
},
```

Figura 4.13: *Métodos para almacenar datos en el modelo.*

- ***Gestor de Eventos***. Otro aspecto importante en la implementación del *frontend* es el gestor de eventos. Se define este como una herramienta la cual es capaz de controlar y dar respuesta a los eventos dinámicos de la aplicación mediante la comunicación entre componentes mediante un patrón de suscripción. En la Figura 4.13, vemos un ejemplo donde se comprueba si el usuario está autenticado en la aplicación y, si no lo está, se emite el evento el cual está siendo escuchado en otra parte del código, en este caso, mostrará el *login* de nuevo en la pantalla en cuestión.

- ***Control de redireccionamientos***. Se ha implementado en la aplicación un sistema de redireccionamiento entre las pantallas de la aplicación el cual está regido por el paquete *VueRouter*. Esta herramienta se encarga de mapear los distintos ficheros de vistas a una determinada ruta de la aplicación. Esto nos permite el cambio entre pantallas de una forma ágil y estructurada para el programador. En la Figura 4.14 se muestran algunos ejemplos de estos y como se hace uso del redireccionamiento en la Figura 4.15.

```
Vue.use(VueRouter)

const routes = [
  // General
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/login/',
    name: 'Login',
    component: Login
  },

  // Bales
  {
    path: '/bala-crear/',
    name: 'BalaCreate',
    component: BalaCreate
  },
  {
    path: '/bala-detall/:id/',
    name: 'BalaDetail',
    component: BalaDetail
  },
```

Figura 4.14: *Implementación del fichero de rutas.*

```
balaDetail(balaID) {
  this.$router.push({ name: "BalaDetail", params: { id: balaID , pagina: this.filtre.pagina} });
},
redireccio(component) {
  this.$router.push({ name: component });
},
```

Figura 4.15: *Métodos de redireccionado.*

- ***Vistas***. Para involucrar al usuario final en el proceso, se han diseñado e implementado una serie de pantallas dinámicas e interactivas. Estas funcionalidades han sido posibles gracias a al biblioteca de interfaces de usuario *Vuetify*. *Vuetify* nos permite agregar componentes predefinidos y modificables a nuestras pantallas y *Vue*, a su vez, nos permite el tratamiento dinámico de datos y darle vida a la aplicación mediante distintos *scripts*. [28]

  En el apartado de *scripts* está definido el objeto *Vue* el cual contiene la definición de variables que aparecen en la vista, los métodos que se usan en dichas pantallas y las acciones que se ejecutan en las distintas etapas del renderizado. Se muestra en la Figura 4.16 algunas partes del código correspondientes a los componentes del renderizado y en la Figura 4.17 un ejemplo de la implementación de los métodos que se ejecutan durante la creación del renderizado.

```
<v-col sm="1">
  <!-- Dialog/Modal per crear magatzem -->
  <v-dialog v-model="dialogMagatzem" persistent max-width="600">
    <template v-slot:activator="{ on }">
      <v-btn v-if="grupo!=4" outlined block color="success" dark v-on="on" tabindex="-1">+</v-btn>
    </template>
    <v-card>
      <v-card-title class="headline">Crear magatzem</v-card-title>
      <v-container>
        <v-form id="formMagatzem" ref="formMagatzem" class="mt-12">
          <v-row>
            <v-col md="12">
              <v-text-field v-model="magatzem_nou.nom" label="Nom" required></v-text-field>
              <v-text-field v-model="magatzem_nou.numero" label="Numero" required></v-text-field>
            </v-col>
          </v-row>
        </v-form>
      </v-container>
      <v-card-actions>
        <v-btn
          color="green darken-1"
          text
          @click="dialogMagatzem = false"
          tabindex="-1"
        >Cancel·lar</v-btn>
        <v-spacer></v-spacer>
        <v-btn color="green darken-1" text @click="btnCrearMagatzem" tabindex="-1">Crear</v-btn>
      </v-card-actions>
    </v-card>
  </v-dialog>
  <!-- ./Dialog/Modal per crear magatzem -->
</v-col>
```

Figura 4.16: *Implementación de un modal de creación de una "Almacén".*

Como se ha podido deducir, este framework está implementado para seguir el patrón *MVC*, en el modelo encontramos los aspectos de almacenamiento los cuales trata *Vuex*, en la vista encontramos los componentes de *Vuetify* mientras que los demás aspectos corresponderían al controlador del sistema.

Aspectos relacionados

En la implementación del *frontend* se han implementado ciertos aspectos que me gustaría comentar ya que se han realizado involucrando al usuario final en la experiencia de uso final.

Empecemos comentando que la aplicación alberga gran cantidad de información y para ayudar al usuario a que no se pierda en sus consultas, hemos implementado un sistema de paginación el cual divide los resultados de las consultas del *API* en lotes de veinte elementos. Esta funcionalidad ha sido posible gracias a que en el *API* cuenta con un sistema de *offsets* que favorece a esto.

Para ayudar aún más al usuario final, se han implementado sistemas de autocompletación en los distintos filtros de las pantallas para que el usuario se ciña a un patrón de búsqueda estándar de la empresa, y no se equivoque.

```
created() {

  this.getBales();
  if(this.$route.params.pagina) {this.pagina = this.$route.params.pagina;
  this.filtre.pagina = this.$route.params.pagina;
  this.paginar();}

  this.getIngredients();
  this.getMagatzems().then(
    response => {response.forEach(el=> el.posicions.forEach(posicio=>
    {
      posicio.nom = el.nom +'- Nº ' + posicio.zona
      this.OpcionesSubalmacenes.push(posicio)}) )}
  );
  this.getMateriesPrimeres();
  this.getMateriesPrimeresNomsSenseDuplicar();
  this.getProveidors();
  this.getTipusMateria();

},
```

Figura 4.17: *Métodos de la creación del renderizado.*

Tal y como se comentó en el diseño, se ha implementado un sistema de notificaciones emergentes para proporcionarle *feedback* al usuario de las acciones que esté realizando. El código se muestra a continuación en la Figura 4.18.

```
<!-- Notificació -->
<v-snackbar v-model="snackbar" :color="color">
  {{ text }}
  <v-btn text @click="snackbar = false">Close</v-btn>
</v-snackbar>
<!-- ./Notificació -->
```

Figura 4.18: *Implementación de notificación emergente.*

Estilos

Ajustándonos a los estilos definidos en la etapa de diseño de la aplicación, se han implementado distintas reglas de estilos mediante directivas *CSS*. Gracias a la combinación de estas se ha mantenido la consistencia de las pantallas en cuanto a estilos y visualizado se refiere. Un ejemplo de este es que la sección de los botones está implementada en el mismo formato en todas las pantallas. En la Figura 4.19 se muestran las reglas en cuestión.

```css
#botonera {
  padding-bottom: 50px;
  width: 100%;
  display: flex;
  flex-direction: row;
  justify-content: space-between;

  button {
    width: 30%;
  }
}
```

Figura 4.19: *Implementación de estilos CSS correspondientes a la botonera.*

En cuanto a las vistas de los operarios de fábrica, se han amoldado a los diseños donde se trataban aspectos como la visibilidad. Para ello, se han aplicado también reglas de estilos correspondientes para cumplir estos requerimientos de diseño. Se muestra en la Figura 4.20 parte de la implementación de una de las pantallas en cuestión donde se puede ver que se apuesta por la visibilidad.

```css
#info-bala-pa {
  background: rgb(211, 211, 211);
  padding: 0 2em;
  border: 1px solid black;
  height: 40vh;
  min-height: 400px;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  .row {
    margin: 0 !important;
    display: flex;
    flex-direction: column;
    justify-content: center;
    div {

      width: 100%;
      height: auto;
      min-height: 8vh;
      display: flex;
      flex-direction: row;
      justify-content: space-between;
      background: white;
      border: 1px dashed black;
      margin: 0.3em 0;
      padding: 0 0.5em;

      .num-quarto{
        height: auto;
        min-height: 8vh;
      }

      * {
        margin: auto 0;
      }
    }
    button {

      max-height: 50px;
    }
  }
}
```

Figura 4.20: *Implementación de estilos CSS correspondientes a la pantalla de trabajadores.*

Resultados

Ciñéndonos a los diseños de las interfaces, el resultado de las interfaces implementadas se muestra a continuación. Veremos interfaces de listado de datos (Figura 4.21), formularios de creación de datos (Figura 4.22) y algunas pantallas para los trabajadores (Figura 4.23).

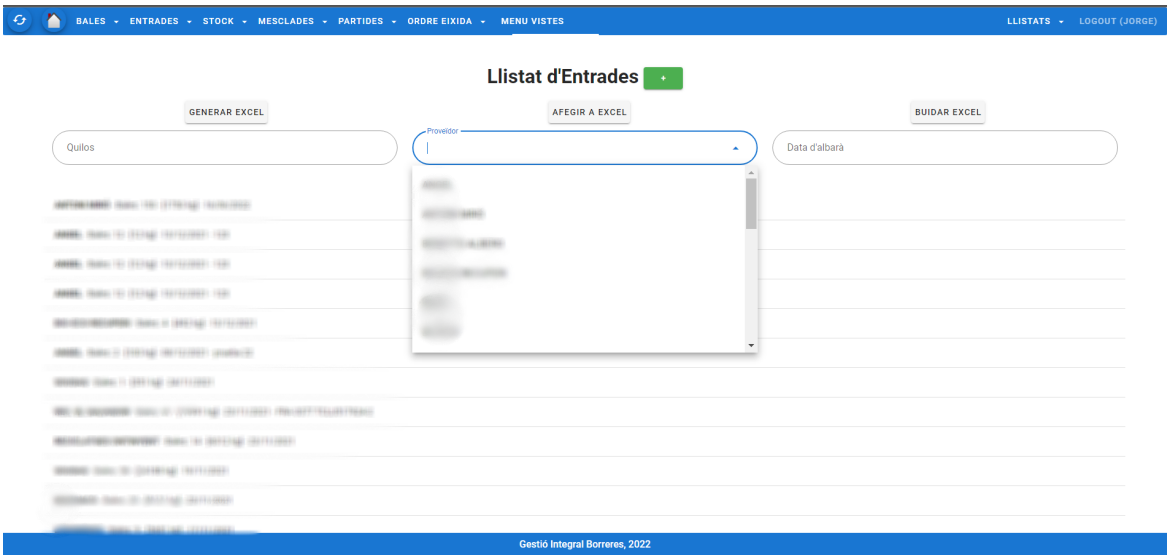

Figura 4.21: *Pantalla de listado de "Entradas".*

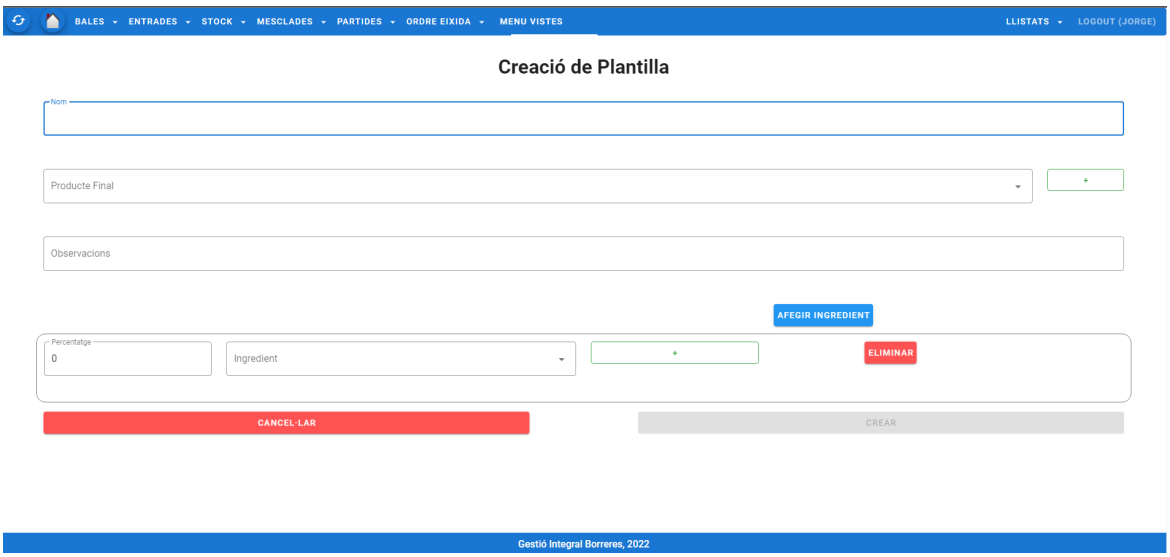

Figura 4.22: *Pantalla de formulario de creación de "Plantillas".*

Figura 4.23: *Pantalla de procesado de "Mezcladas".*

Por motivos de seguridad y confidencialidad, no está permitida la difusión de datos privados de la empresa, por ello, quedan ocultos al lector.

4.1.3. Problemas en la implementación

En esta sección se expondrán los problemas en la implementación que han producido pequeñas demoras en la entrega del proyecto.

El principal problema, y causante de todos los demás, ha sido la inexperiencia con el software de *Django*. Al no haber trabajado nunca con un framework similar, me ha producido muchas dudas y problemas que han tenido que ser resueltas por el resto del equipo de desarrolladores.

Esto ha generado problemas sobre todo alrededor de los serializadores ya que me costó asimilar el concepto y cuándo y cómo usarlos. A la hora de implementar una funcionalidad nueva del *API* implicaba muchas veces generar un nuevo serializador, y como he comentado no me estaba aclarando, por ello, los primeros días trabajando en la implementación han resultado ser bastante duros hasta que, a través de la práctica, se ha convertido en tareas más simples.

Otro problema que me ha surgido, es debido en general a la inexperiencia con el lenguaje *Python*. A la hora de comenzar la estancia en prácticas, se me ofrecieron por parte del equipo cursos y además proyectos similares los cuales estaban implementados por programadores senior, pero poco a poco, y practicando, se ha conseguido realizar las tareas acordadas.

4.2. Verificación y validación

Con la finalidad de analizar el correcto funcionamiento tanto del *API* como de la aplicación web, se han realizado distintas pruebas que seguidamente comentamos.

4.2.1. Backend

Siguiendo la metodología en cascada que se estaba usando durante el desarrollo del proyecto, una vez realizada la implementación de las funcionalidades del proyecto, se ha pasado la etapa de "Pruebas" y seguidamente a la etapa de "Mantenimiento". Con esto me refiero a que conforme se implementaba el método del *API*, a la par se iba testeando su correcto funcionamiento mediante la implementación de distintos tests. Para implementar estos tests se ha hecho uso de la herramienta ***pytest*** la cual proporciona funcionalidades para la creación de pruebas.

Veamos a continuación un ejemplo donde se muestra en la Figura 4.24 la implementación del test correspondiente al listado de "balas MP" ordenadas según sus pesos. El test realiza la petición autenticado con un usuario, la inserción de las balas y seguidamente, consulta los datos analizando si son los resultados esperados.

```python
def test_list_bala_materia_primera_orden_quilos_ascendiente(
    api_client: "APIClient",
    materia_primera_1: "MateriaPrimera",
    usuario: "User",
    expected_response_status: int,
) -> None:
    auth(api_client, usuario.username)
    bala1 = BalaMateriaPrimera.objects.create(
        quilos=3,
        materia_primera=materia_primera_1,
    )
    bala2 = BalaMateriaPrimera.objects.create(
        quilos=4,
        materia_primera=materia_primera_1,
    )
    bala3 = BalaMateriaPrimera.objects.create(
        quilos=2,
        materia_primera=materia_primera_1,
    )

    assert BalaMateriaPrimera.objects.count() == 3
    url = reverse("bala_materia_primera-list")
    url += f"?order=-quilos"
    response = api_client.get(url, format="json")
    assert response.status_code == expected_response_status

    if response.status_code == status.HTTP_200_OK:
        results = cast(dict[str, list[Any]], response.data).get("results", [])
        print(results)
        assert len(results) == 3
        print("pasa")
        expected_response = [BalaMateriaPrimeraSerializer(
            [bala2], many=True, read_only=True
        ),BalaMateriaPrimeraSerializer(
            [bala1], many=True, read_only=True
        ),BalaMateriaPrimeraSerializer(
            [bala3], many=True, read_only=True
        ),
        ]
        print(expected_response)
        assert results != cast(list[dict[str, Any]], expected_response)
```

Figura 4.24: *Implementación del test para comprobar la ordenación de 'balas MP".*

Otro ejemplo sería el test mostrado en la Figura 4.25. Este comprueba la creación de una "bala MP", para ello, de forma similar al anterior, realiza la inserción de una bala en una base de datos vacía y seguidamente comprueba que se ha añadido.

```python
def test_create_bala_materia_primera(
    api_client: "APIClient",
    materia_primera_1: "MateriaPrimera",
    entrada_1: "EntradaMateriesPrimeres",
    usuario: "User",
    expected_response_status: int,
) -> None:
    auth(api_client, usuario.username)
    url = reverse("bala_materia_primera-list")
    assert BalaMateriaPrimera.objects.count() == 0
    data = {
        "quilos": 3,
        "materia_primera": materia_primera_1.id,
        "entrada": entrada_1.id,
    }
    response = api_client.post(url, data, format="json")
    assert response.status_code == expected_response_status

    if response.status_code == status.HTTP_201_CREATED:
        assert BalaMateriaPrimera.objects.count() == 1
        bala_materia_primera = BalaMateriaPrimera.objects.first()
        expected_data = BalaMateriaPrimeraSerializer(bala_materia_primera).data
        assert response.data == expected_data
    else:
        assert BalaMateriaPrimera.objects.count() == 0
```

Figura 4.25: *Implementación del test para comprobar la creación de 'balas MP".*

4.2.2. Frontend

A la hora de comprobar el funcionamiento de la aplicación web, no se han realizado pruebas específicas en cuanto a la funcionalidad implementada, puesto que se ha cerciorado en todo momento que la *API* realizase correctamente sus funciones implementadas.

Otro aspecto a tener en cuenta es que las distintas pantallas cuentan con la capacidad de comprobar si es posible o no realizar la acción en cuestión, es decir, la consistencia en los datos y el control de esta han sido implementadas de tal forma que no supone un riesgo a la hora de que el usuario realice acciones. Esto se ha conseguido dotando de margenes a ciertas funcionalidades como el sistema de autocompletar el cual solo permite seleccionar valores validos pero sobre todo, las comprobaciones son a través del *API*.

Finalmente, a la hora de diseñar las pantallas se solicitó información a los operarios para comprender los requerimientos en las interfaces. Una vez fueron diseñadas, se solicitó *feedback* de estos mismos sobre los *mockups* para saber si eran correctos o en su defecto si había que modificar algo. A día de hoy el proyecto está en marcha y en labores de mantenimiento constantes realizando las modificaciones que la empresa solicite.

Capítulo 5

Conclusiones

Este proyecto ha supuesto en mi un crecimiento bastante importante en distintos aspectos de la vida tanto en lo formativo, como en lo profesional y en lo personal.

En cuanto a lo formativo me ha hecho ver el alcance que puede llegar a tener la informática en cualquier aspecto de la vida. Me ha parecido un proyecto increíble ya que, sin ser relativamente complejo, es capaz de digitalizar todo el proceso productivo de la empresa siendo este una herramienta de trabajo muy útil. Otro matiz a tener en cuenta, es que me ha enseñado que lo aprendido durante los cuatro años de grado, sí que sirve para algo y no solo para aprobar. También me ha hecho ver que este mundo es esfuerzo y constancia, y que a la hora de programar no sirve solo con lo básico aprendido tanto en al universidad, como cursos *online* o videotutoriales. El rodearte de gente que sabe del tema o afrontar retos reales ayuda mucho para aprender. No solo aprender, sino que también el éxtasis de felicidad al ver que algo en lo que te has estado esforzando funciona.

Como posible contra, he de decir que *Django* no me ha llegado a convencer demasiado. Supongo que al ser un *framework* tan potente, tiene muchas funcionalidades las cuales hay que saber manejar implicando así ser también un *software* muy complejo de controlar. Durante la estancia realicé una serie de cursos para familiarizarme con el *framework*, pero hasta que mis compañeros no me echaron una ayuda, no llegué a comprenderlo.

En el ámbito profesional estoy muy contento puesto que se me ha hecho llegar todo tipo de ayudas. El clima de trabajo es excepcional y esto ha hecho que el proyecto saliera adelante aún con todas las dificultades que se han acometido. El trabajar en grupo me ha ayudado mucho ya que al final, en los trabajos universitarios grupales, se resume en hacer un *collage* de las partes de cada uno, sin embargo, el trabajar con profesionales y pautar los requisitos en conjunto, definir un patrón y un rumbo que seguir es sinónimo de un buen trabajo.

Finalmente, en lo personal me siento muy contento de esta oportunidad, me ha dado a conocer que el mundo laboral no es fácil y que la inversión de tiempo y esfuerzo en el grado no han sido en vano. También estoy muy contento con mis compañeros que me han ayudado a que el día no se tan duro.

Como aspecto técnico a tener en cuanta, los objetivos planteados han sido cumplidos en su gran mayoría, el inicio del proyecto se retrasó considerablemente por razones ajenas a mi, siendo este el causante de tales incumplimientos. Estos no han supuesto nada relevante en el proyecto puesto que mis compañeros me hicieron llegar que implementaron lo que quedó sin realizar y que el proyecto estaba en marcha y los clientes satisfechos.

Bibliografía

[1] CIC Consulting Informático. Industria 4.0, la cuarta revolución industrial y la inteligencia operacional. https://www.cic.es/industria-40-revolucion-industrial/. [Consulta: 9 de Junio de 2022].

[2] Dynatec. Inteligencia artificial: el propulsor de la industria. https://dynatec.es/2020/06/07/inteligencia-artificial-el-propulsor-de-la-industria-4-0/. [Consulta: 9 de Junio de 2022].

[3] Equipo de Robottions S.L. Servicios ofrecidos por *Robottions* s.l. https://www.robottions.com/servicios. [Consulta: 9 de Junio de 2022].

[4] Finerio Connect. *API*s: ¿qué son y cómo están transformando el sector financiero? https://blog.finerioconnect.com/apis-que-son-y-como-estan-transformando-el-sector-financiero/. [Consulta: 9 de Junio de 2022].

[5] *Santander Universidades*. Python: qué es y por qué deberías aprender a utilizarlo. https://www.becas-santander.com/es/blog/python-que-es.html. [Consulta: 11 de Junio de 2022].

[6] *Python Software Foundation*. About *Python*. https://www.python.org/about/. [Consulta: 11 de Junio de 2022].

[7] *Django Project*. Documentación oficial de *Django REST framework*. https://www.django-rest-framework.org/. [Consulta: 11 de Junio de 2022].

[8] Rafael Ramos. ¿qué es *JavaScript* y para qué sirve? https://soyrafaramos.com/que-es-javascript-para-que-sirve/. [Consulta: 11 de Junio de 2022].

[9] Simran Kaur Arora. 10 best javascript frameworks to use in 2022. https://hackr.io/blog/best-javascript-frameworks. [Consulta: 11 de Junio de 2022].

[10] *OpenJS Fundation*. Acerca de *Node.js*. https://nodejs.org/es/about. [Consulta: 11 de Junio de 2022].

[11] Jesús Lucas. Qué es *NodeJS* y para qué sirve. https://openwebinars.net/blog/que-es-nodejs/. [Consulta: 11 de Junio de 2022].

[12] *Vue.js*. Documentación oficial de *Vuejs framework*. https://vuejs.org/guide/introduction.html. [Consulta: 11 de Junio de 2022].

[13] *Team Asana*. Las 12 metodologías más populares para la gestión de proyectos. https://asana.com/es/resources/project-management-methodologies. [Consulta: 13 de Junio de 2022].

[14] DirectorTIC. Software "ágil" frente al software "en cascada". https://directortic.es/estrategia-it/software-agil-frente-al-software-en-cascada-2015092214539.htm. [Consulta: 13 de Junio de 2022].

[15] *Atlassian Fundation*. Planes y precios *BitBucket*. https://www.atlassian.com/es/software/bitbucket/pricing. [Consulta: 15 de Junio de 2022].

[16] *Rafael Berlanga*. Ei1053 - tecnologias emergentes: Tema 4. arquitecturas orientadas a servicios. https://aulavirtual.uji.es/pluginfile.php/5946348/mod_resource/content/11/Tema%204.pdf. [Consulta: 20 de Junio de 2022].

[17] *Vue.js*. Concepts overview about vue.js. https://012.vuejs.org/guide/. [Consulta: 20 de Junio de 2022].

[18] *Bootstrap Team*. Documentación oficial de *Bootstrap5*. https://getbootstrap.com/docs/5.0/getting-started/introduction/. [Consulta: 21 de Junio de 2022].

[19] *PythonDiario*. __init__.py en python - que es y como funciona. https://pythondiario.com/2013/06/initpy-en-python-que-es-y-como-funciona.html#:~:text=El%20archivo%20__init__.py%20no%20tiene%20porque%20tener,lo%20hace%20para%20poder%20importarlos. [Consulta: 22 de Junio de 2022].

[20] *Django Software Foundation and contributors*. Escribiendo tu primera django app. https://djangotutorial.readthedocs.io/es/1.8/intro/tutorial02.html#enter-the-admin-site. [Consulta: 22 de Junio de 2022].

[21] *Django Software Foundation*. Documentación sobre las aplicaciones de *Django*. https://docs.djangoproject.com/en/4.0/ref/applications/#django.apps.AppConfig. [Consulta: 23 de Junio de 2022].

[22] *Uniwebsidad*. El patrón de diseño mtv. https://uniwebsidad.com/libros/django-1-0/capitulo-5/el-patron-de-diseno-mtv. [Consulta: 23 de Junio de 2022].

[23] *Runebook*. Migraciones en *Django*. https://runebook.dev/es/docs/django/topics/migrations. [Consulta: 23 de Junio de 2022].

[24] David Sanders. Documentación oficial de *Simple JWT*. https://django-rest-framework-simplejwt.readthedocs.io/en/latest/. [Consulta: 23 de Junio de 2022].

[25] Eduardo Zepeda. *Django Rest Framework* y *JWT* para autenticar usuarios. https://coffeebytes.dev/django-rest-framework-y-jwt-para-autenticar-usuarios/. [Consulta: 23 de Junio de 2022].

[26] T. Franzel. Documentación oficial de *drf-spectacular*. https://drf-spectacular.readthedocs.io/en/latest/readme.html. [Consulta: 23 de Junio de 2022].

[27] *Vue.js*. What is *Vuex*? https://vuex.vuejs.org/. [Consulta: 24 de Junio de 2022].

[28] *Vuetify Team*. Documentación oficial de *Vuetify*. https://vuetifyjs.com/en/introduction/why-vuetify/#why-vuetify3f. [Consulta: 24 de Junio de 2022].