Grado en Ingeniería Informática

Trabajo Final de Grado

Aplicación de Gestión de *Stock* mediante Códigos de Barras

Autor:
Juan José González Abril

Supervisor:
Vicente Javier Trilles Zumaquero
Tutor académico:
Mª Ángeles López Malo

Fecha de lectura: 17 de Septiembre de 2019
Curso académico 2018/2019

Resumen

Este proyecto trata de mejorar y agilizar el proceso de gestión de las existencias en el almacén, así como de las incidencias de los diferentes productos, identificados mediante códigos de barras, de los que dispone la empresa BabyEssentials S.L. Para esto se elabora una aplicación web que permite a cada uno de los diferentes departamentos de la empresa desarrollar sus tareas de una forma fácil y rápida, además de reducir la cantidad de errores en la información introducida.

Los departamentos en cuestión y sus tareas son las siguientes:

- El departamento de diseño se encarga de crear los diferentes modelos de productos, así como de la creación de los códigos de barras para cada producto individual que se fabrica. Estos productos se fabrican por lotes, la información de los cuales también se almacena en el sistema.
- El departamento de atención al cliente se encarga de registrar las incidencias que se produzcan en relación con los productos, tanto para poder tramitar su reparación si procediese, como para poder identificar problemas en modelos, lotes o con un cliente en concreto.
- El departamento del almacén se encarga de llevar un control de movimiento de los productos, registrando las salidas y entradas de lotes y productos individuales, si se trata de una incidencia, en el almacén.

Palabras clave

Aplicación web, Kotlin, Vue.js, API REST, Código de barras

Keywords

Web app, Kotlin, Vue.js, REST API, Barcode

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

La empresa en la que se han desarrollado las prácticas y el proyecto ha sido Baby Essentials S.L. Esta empresa se fundó en 2008 y actualmente se dedica al diseño y distribución de productos de puericultura, tanto al por mayor como al por menor. También proporciona soporte al usuario final mediante su departamento de atención al cliente, para cualquier problema con sus productos. Todas estas tareas las llevan a cabo tres departamentos diferentes: el departamento de diseño se encarga del diseño y la generación de los códigos de barras que identificarán cada uno de los productos; el departamento de atención al cliente se encarga de registrar las incidencias que se produzcan y de procesar la reparación si el producto está en garantía; y por último el departamento del almacén se encarga de registrar los movimientos de los productos que entran y salen del almacén. Adicionalmente existe un departamento de informática, algo muy necesario hoy en día para una empresa de venta de productos, ya que se encarga del mantenimiento de la página web de la empresa, que contiene toda la información sobre los productos que tienen a la venta.

Baby Essentials S.L. posee una amplia variedad de modelos de carritos de bebé, para cada uno de los cuales existen múltiples variantes. En la actualidad todo el control de la información relativa a dichos modelos y sus variantes, la producción de los diferentes lotes, así como las incidencias que se produzcan en relación a los productos distribuidos se lleva a cabo mediante el uso de hojas de cálculo de Excel. Estas hojas se encuentran almacenadas en un NAS[1] accesible únicamente desde la propia oficina de la empresa y no desde el exterior, de modo que el almacén de productos que se encuentra en una ubicación diferente no tiene acceso a estos datos, por lo que la información de los productos que entran o salen del almacén no se puede actualizar al momento.

Debido a esta gran variedad de productos y a que existen diferentes departamentos para cada una de las actividades que desarrolla la empresa, el sistema actual presenta una serie de problemas. Cabe destacar los siguientes:

[1] *Network Attached Storage*

- Es propenso a errores debido a que no existe ningún tipo de validación de los datos introducidos, además es posible que varios trabajadores modifiquen el mismo archivo lo que provocaría una inconsistencia en los datos.
- Es lento ya que cada vez que hay que añadir o modificar información hay que buscar en qué archivo se encuentra, luego en qué hoja y finalmente en qué fila.
- La información de los productos del almacén no está actualizada en tiempo real.

Mi trabajo como estudiante de prácticas consiste en desarrollar una aplicación web independiente que pueda ser utilizada desde cualquier dispositivo con acceso a Internet. La motivación que me llevó a elegir este proyecto fue la posibilidad de poder aplicar todos mis conocimientos sobre tecnologías web a un proyecto real, así como la posibilidad de poder adquirir nuevos conocimientos que pueda aplicar en un futuro.

1.2. Objetivos del proyecto

El objetivo principal del proyecto de acuerdo con mi supervisor es crear una aplicación web, que almacenará toda la información sobre productos, incidencias y movimientos del almacén en una base de datos MySQL, y que permita a los trabajadores de los diferentes departamentos llevar a cabo sus tareas de manipulación de información de una forma fácil y rápida. Cada departamento dispondrá de una vista personalizada y adaptada al trabajo que desempeñe.

Adicionalmente, puesto que se dispondrá de una base de datos con toda la información centralizada, se creará una vista adicional de análisis donde se generarán graficas con información sobre las incidencias que se producen en el tiempo, para facilitar la identificación de lotes y productos defectuosos.

1.3. Objetivos detallados

En esta sección se muestran en detalle los objetivos de este proyecto para cada uno de los departamentos:

- Permitir iniciar sesión a los trabajadores de los diferentes departamentos.
- Para el departamento de diseño:
  - Mostrar dos listas una con los modelos disponibles y otra con los lotes de cada modelo.
  - Crear un nuevo modelo o editar uno ya existente.
  - Crear un nuevo fabricante.
  - Para un modelo concreto, crear un nuevo lote o editar uno ya existente.
  - Para un lote concreto, introducir un número determinado de códigos de barras.

- Para el departamento de atención al cliente:
  - Mostrar una lista con las incidencias existentes.
  - Crear un nuevo cliente.
  - Crear una nueva incidencia o editar una ya existente.
- Para el departamento del almacén:
  - Mostrar dos listas, una con la información de entrada de materiales y otra con la de salida.
  - Registrar una nueva entrada o salida de material o editar una ya existente.
- Para la vista de análisis:
  - Mostrar una gráfica que representa el número de incidencias en el tiempo y donde se puede especificar un modelo, lote o producto concreto y ver la información correspondiente.

Capítulo 2

Descripción del proyecto

2.1. Situación inicial

Baby Essentials actualmente cuenta con tres departamentos, los cuales llevan a cabo una serie de tareas diferenciadas que forman parte del ciclo de vida de cada uno de sus productos. Sin embargo, estas tareas, que se explican a continuación, se realizan en su gran mayoría de forma manual; esto, además de ser un proceso lento y tedioso, es propenso a producir errores a la hora de introducir los datos en las hojas de Excel puesto que no existe validación alguna que asegure la consistencia de los datos. Y son estas también las principales razones por las cuales la empresa ha decidido crear un sistema que permita agilizar la introducción de los datos, así como asegurar que no existen errores en dichos datos.

2.1.1. Departamento de Diseño

El departamento de diseño sería la primera fase de un producto, aquí es donde diseñan y se definen los modelos que existen para los diferentes productos, por ejemplo: *Shom Elegance*. Una vez definida toda la información relativa a los modelos, esta se guarda en una hoja de Excel.

El siguiente paso sería, a partir de los modelos existentes, crear diferentes variantes donde principalmente cambia el color, por ejemplo: *Shom Elegance Arena*. Cada variante se suele fabricar en lotes de unos 100 productos. Una vez se ha elegido el fabricante que producirá el lote se generan los números de serie únicos para cada producto y se envían junto con toda la información necesaria al fabricante.

Finalmente, se almacena toda la información generada, que se puede ver en la figura 2.1, en una hoja de Excel.

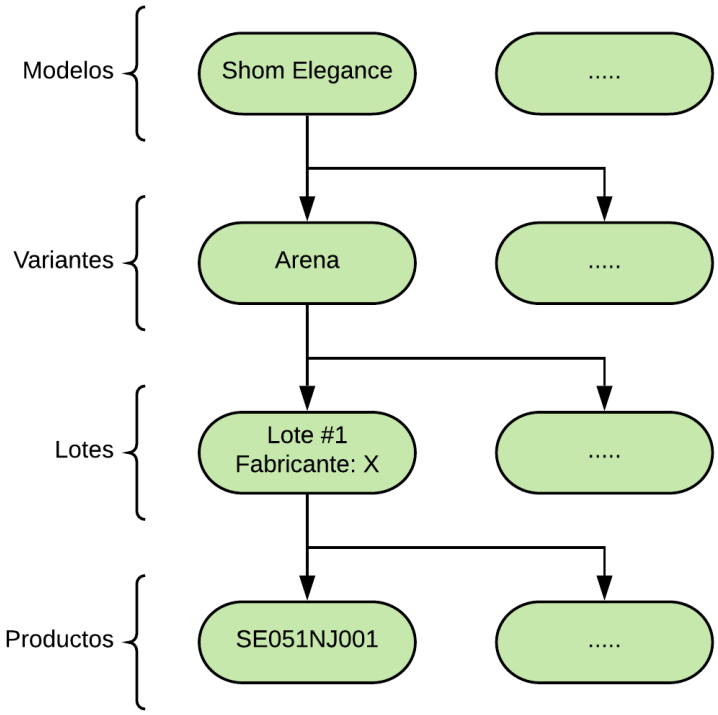

Figura 2.1: Información generada en el departamento de diseño

2.1.2. Almacén

El siguiente departamento por el que transitan los productos es el almacén. Aquí es donde llegarán formando parte de un lote una vez han sido fabricados. El control de entrada de los materiales actualmente se hace a mano apuntando el número de albarán del lote y los códigos de cada producto. Más adelante cada producto que sale del almacén para ser enviado las tiendas que los venden a los clientes finales, se registra de nuevo mediante su número de serie.

Adicionalmente cuando se registra alguna incidencia, tal y como se explica en la sección 2.1.3, también se registra su entra en el almacén y su posterior salida, en caso de que se produjese.

2.1.3. Departamento de Atención al Cliente

El último departamento por el que pueden pasar los productos es el de atención al cliente. Aquí se atienden los problemas que surjan a los clientes finales con respecto a los productos, como pueden ser roturas de piezas o desgaste, por nombrar algunas. Cuando se recibe una incidencia y se determina que se puede reparar, se registra la información del cliente afectado en una hoja de Excel, junto con la fecha en la que se ha producido la incidencia y el número de serie del producto.

En ese momento se procede a la recogida del producto para su reparación, momento en el que se registra la entrada del producto en el almacén. Una vez finalizada la reparación, se crea un albarán de salida del producto, se registra su salida del almacén y se envía al cliente de nuevo

mediante una empresa de transporte.

2.2. Descripción de la aplicación

Una vez implantada la aplicación en el día a día de la empresa, cada uno de los departamentos desarrollará sus tareas de una forma un poco diferente a la actual, pero el concepto será el mismo.

El departamento de diseño podrá crear diferentes modelos indicando un nombre único y una imagen a través de un formulario. Una vez creado un modelo, se podrán añadir lotes de dicho modelo indicando su información, como puede ser la variante, y una lista de códigos de barras que tendrán una relación 1:1 con cada uno de los productos que contenga el lote. Estos códigos de barras tendrán un formato definido por la empresa.

El departamento de atención al cliente tendrá la habilidad de registrar incidencias de un producto a partir de un código de barras ya registrado, junto con toda la información relevante incluyendo la información del cliente, una descripción de la incidencia, etc. Estas incidencias se podrán editar más adelante para indicar si se ha realizado una reparación o no.

El departamento del almacén podrá registrar las entradas y salidas de productos del almacén mediante el escaneo o introducción manual del código de barras de cada producto.

Beneficios

Los principales beneficios que aporta la aplicación al sistema actual son los siguientes.

- Centralización de toda la información en una única base de datos.
- Referencias cruzadas de la información entre los diferentes departamentos.
- Sistema de validación de la información introducida.
- Capacidad de acceder desde cualquier lugar sin necesidad de tener que estar en la oficina.

2.3. Tecnologías

Debido a que el proyecto se va a implementar desde cero, tenemos a libertad de elegir las tecnologías que consideremos más adecuadas basándonos en nuestros conocimientos previos, así como las tareas que se vayan a llevar a cabo, a excepción de alguna tecnología concreta por petición expresa del supervisor, como puede ser la base de datos. Puesto que el proyecto consiste en una aplicación web, podemos agrupar las tecnologías usadas en dos grupos dependiendo de si se usan en la parte del servidor o en la parte del cliente.

En la parte del servidor podemos encontrar:

- Kotlin [6], un lenguaje de programación para la JVM[1].
- Ktor [8], un *framework* para crear aplicaciones para servidores web.
- MySQL para la base de datos de la aplicación.
- Exposed [12], un *framework* SQL para trabajar con la base de datos.

En la parte del cliente podemos encontrar:

- React, una biblioteca para crear interfaces de usuario usando JavaScript.
- Bootstrap, una biblioteca con clases CSS predefinidas para crear elementos de la interfaz consistentes.

2.3.1. Kotlin

Durante mi estancia en Irlanda como estudiante del programa ERASMUS+ en 2017, conocí a un compañero de Ingeniería Informática que me introdujo al lenguaje, y en apenas 3 meses ya lo dominaba, gracias a mis conocimientos previos sobre Java. El lenguaje es relativamente nuevo ya que apareció por primera vez en 2011, aunque la versión 1.0 se publicó en febrero de 2016 [7], hace apenas 3 años.

Kotlin es un lenguaje de programación pragmático de código abierto y tipificado estáticamente para la JVM que combina funciones de programación orientada a objetos y funcional. Está enfocado en la interoperabilidad, seguridad, claridad y soporte de herramientas, con todo el ecosistema de Java actual.

Las principales razones por las que lo elegí fueron, para poder aplicar mis conocimientos sobre este nuevo lenguaje a un proyecto real, y para poder usar una tecnología del lenguaje recientemente anunciada, Kotlin Multiplatform.

Kotlin Multiplatform

En la versión 1.2 del lenguaje se introdujeron una serie de características y herramientas experimentales que permiten que código escrito en Kotlin pueda ser reutilizado en diferentes plataformas, evitando tener que re-implementar, por ejemplo, partes del modelo de datos en diferentes lenguajes.

Las plataformas actualmente soportadas son las siguientes:

- JVM
- Android

[1] *Java Virtual Machine*

- JavaScript
- iOS
- Linux
- Windows
- Mac

De estas plataformas las que nos interesan son la JVM que será la usada en la parte del servidor, y JavaScript que será la usada en la parte del cliente. La idea principal es escribir todo el código usando Kotlin y aprovechar esta nueva tecnología para no tener que implementar dos veces el modelo de datos, lo que debería permitirnos agilizar el proceso de desarrollo.

No obstante, debido a que se trata de una tecnología experimental y que podía presentar problemas de rendimiento y/o de compatibilidad, antes del comienzo de las prácticas realicé algún proyecto de prueba que sirvió como una primera toma de contacto con la tecnología, así como para familiarizarme con su funcionamiento. Durante estas pruebas, si bien no poseían la complejidad de un proyecto como el de las prácticas, no presentaron ninguna clase de problema que me diese a entender que no sería factible aplicarla a un proyecto real.

2.3.2. Ktor

A raíz de aprender Kotlin fui descubriendo toda una serie de bibliotecas y *framework* que aprovechaban a fondo las características y funcionalidades que ofrece el lenguaje, uno de estos *framework* fue Ktor.

Ktor es un *framework* para crear aplicaciones web asíncronas para servidores que está diseñado desde cero con Kotlin en mente, esto se puede apreciar en el hecho de que mediante el uso de un DSL[2] muy expresivo es posible crear de forma intuitiva *endpoints* para una API, como se puede apreciar en la figura 2.2. La razón por la que lo elegí fue por las facilidades que ofrece para crear servicios web, así como para poder aprender más en detalle cómo funciona y todo lo que ofrece.

2.3.3. Exposed

Exposed es otro de los *framework* escritos en Kotlin que descubrí. En este caso se trata de un *framework* SQL, desarrollado por un empleado de JetBrains como un proyecto personal.

Este *framework* permite, mediante el uso de DSL propios, trabajar con la información de una base de datos de una manera muy cómoda y sin tener que usar otro lenguaje. Más concretamente es posible definir tablas, columnas, así como sus tipos y restricciones usando clases y propiedades; la principal ventaja de esto es que las consultas que se realizan a la base de datos son *typesafe*[3],

[2] *Domain-Specific Language* (Lenguaje de Dominio Específico)
[3] Los tipos son seguros, previene errores de tipado debidos a discrepancia entre los tipos de las variables

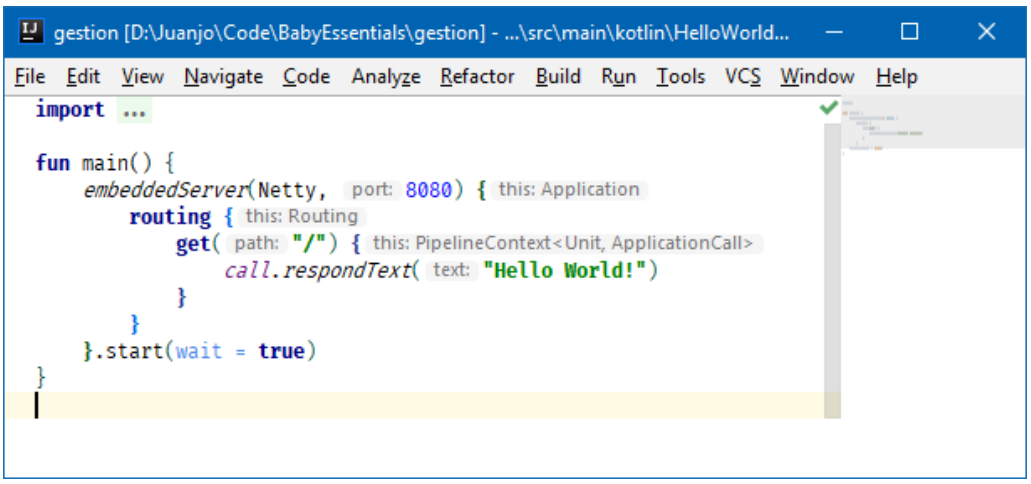

Figura 2.2: Ejemplo de la creación de un servidor en el puerto `8080` con un *enpoint* con el verbo `GET` en la ruta `/` y que responde con el texto `Hello World!`

lo que nos asegura en tiempo de compilación que no va a haber ningún problema de tipado al trabajar con la base de datos, por ejemplo que intentemos guardar un entero en una columna de tipo booleano.

2.3.4. MySQL

La base de datos elegida para usar en el proyecto fue MySQL. Esta elección estuvo condicionada por el hecho de que la empresa ya disponía de un servicio de alojamiento para sus páginas web actuales, el cual incluía acceso a una base de datos MySQL. A pesar de que tiene algunas diferencias con respecto a PostgreSQL que es con la que más experiencia tengo, como las *check constraints*, no ha supuesto ningún problema a la hora de trabajar con ella.

2.4. Software usado

Además de las tecnologías mencionadas también ha sido necesario el uso de un software especifico para el desarrollo del proyecto. Este se indica a continuación:

- IntelliJ IDEA, el IDE[4] usado para la programación tanto del cliente como del servidor.
- Postman para realizar pruebas contra la API REST.
- BitBucket como repositorio Git para el código del proyecto.
- Trello para llevar un control de las tareas.
- Heroku para el despliegue final de la parte del servidor.

[4] *Integrated Development Environment* (Entorno Integrado de Desarrollo)

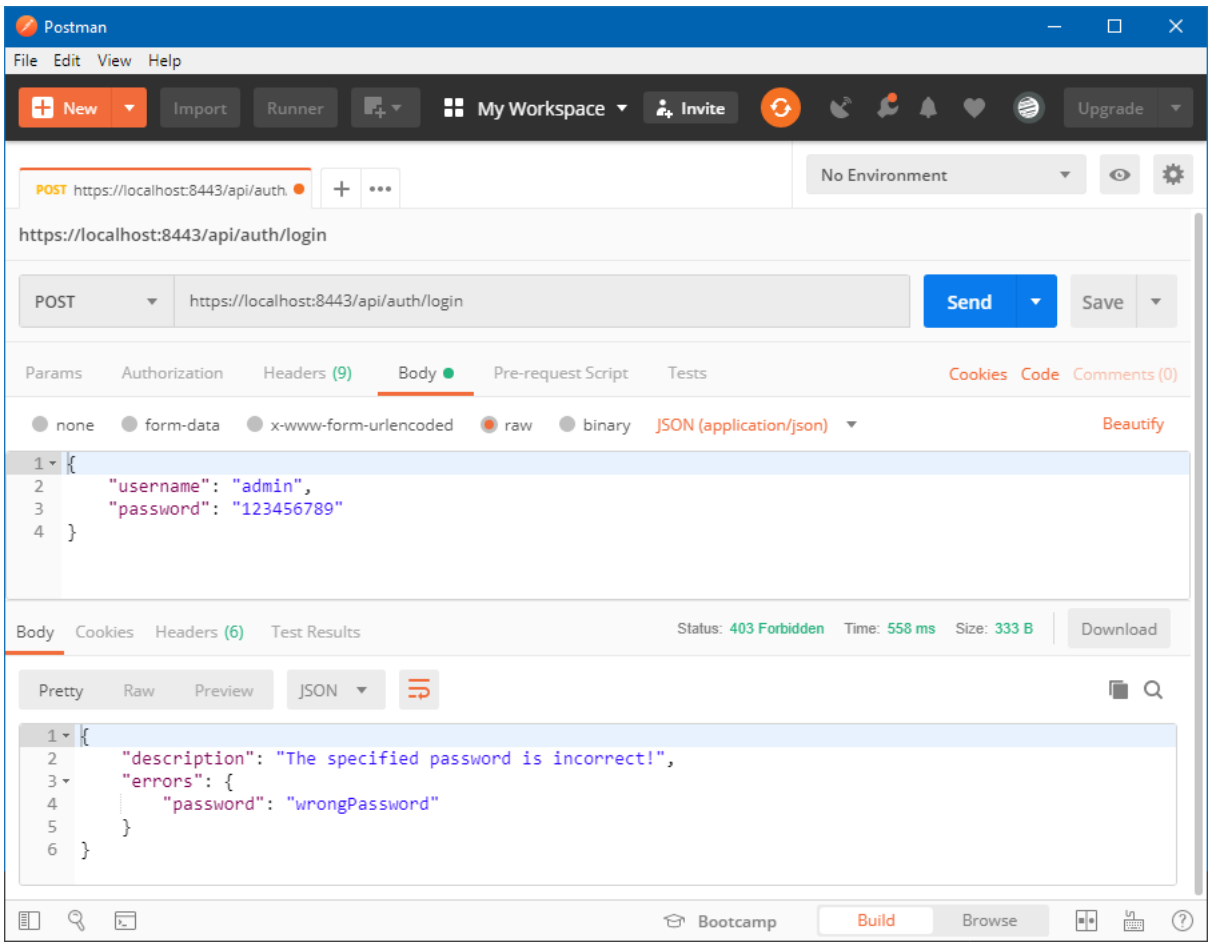

Figura 2.3: Petición `POST` a la ruta `/api/auth/login` y su resultado

2.4.1. Postman

Postman es una herramienta gratuita que permite realizar pruebas de funcionamiento contra servicios HTTP, como puede ser una API REST en nuestro caso. La principal utilidad que tiene es poder realizar peticiones POST o especificar las cabeceras que se enviarán en la petición, algo que no es fácil de hacer en los navegadores convencionales, donde lo único que se puede hacer son peticiones GET. En la figura 2.3 se presenta un ejemplo de una petición POST a un servidor de prueba. Se puede observar la información de la petición en la mitad superior y la información de la respuesta proporcionada por el servidor en la mitad inferior.

2.4.2. Heroku

Heroku es una plataforma como servicio (en inglés *platform as a service*, PaaS) que permite la ejecución de aplicaciones web desarrolladas en Java o Kotlin, entre otros lenguajes, sin apenas necesidad de configuración. Es debido a esta simplicidad y a mi experiencia previa con este servicio que decidí usarla para el despliegue final de la aplicación en producción.

Capítulo 3

Planificación del proyecto

3.1. Metodología

La metodología que elegí para este proyecto fue una metodología ágil basada en Scrum, pero no completamente fiel a la misma. Se estableció que los sprints tendrían una duración de una semana y que se realizarían reuniones al comienzo de cada sprint.

A continuación se presentan las principales diferencias con Scrum:

- No existe un *Scrum Manager*, únicamente el *Product Owner* que es el supervisor de la empresa y el equipo de desarrollo compuesto solo por mí.
- Debido a que únicamente existe un desarrollador no se realizan los Scrum diarios.

El sistema utilizado para la gestión del proceso de Scrum ha sido Trello, puesto que la empresa ya tenía integrado el uso de esta herramienta en su día a día para controlar diferentes tareas.

3.2. Planificación

En base a la metodología elegida, la planificación inicial para este proyecto se puede observar en la tabla 3.1.

3.2.1. Pila de producto

Como parte de la planificación inicial del proyecto se definió una pila de producto con las historias de usuario correspondientes con todos los objetivos expuestos en la sección 1.3. Estas historias de usuario se muestran a continuación en la tabla 3.2.

Sin embargo, a medida que avanzaba el proyecto fueron surgiendo historias de usuario que no había contemplado en la pila inicial, de modo que fui ampliándola con las nuevas historias. La pila de producto con todas las historias añadidas a lo largo del desarrollo se puede observar en la tabla 3.3.

3.3. Estimación de recursos y costes del proyecto

Al realizar una estimación de los recursos necesarios para este proyecto y teniendo en cuenta sus costes, se estima que el coste total del proyecto asciende a los $3\,169,97$€. A continuación, se realiza un desglose de cada una de las categorías de recursos: personal, software y hardware.

3.3.1. Personal

La estimación del personal necesario para el correcto desarrollo de este proyecto es de 3 personas-mes. Respecto a los conocimientos necesarios por parte de los desarrolladores, ha de ser el propio de un estudiante de Ingeniería Informática en el 4º curso de la carrera, con ciertos conocimientos extra sobre las tecnologías expuestas en las sección 2.3.

El desarrollador ha realizado 300 horas de trabajo. Basándonos en los sueldos medios obtenidos en Indeed [4], el sueldo anual medio de un desarrollador junior sería de $20\,602$€, dividido entre 52 semanas divididas entre jornadas de 40 horas, nos da un precio por hora de $9,90$€. Por lo tanto el coste del desarrollador ascendería a los $9,90 \times 300 = 2\,970$€.

3.3.2. Software

El software necesario sería el ya expuesto en la sección 2.4. Respecto al precio de dicho software, sería el siguiente:

- **IntelliJ IDEA:** El precio de una suscripción mensual es de $49,90$€ [5] por lo que el precio para los 3 meses de desarrollo ascendería a $49,90 \times 3 = 149,97$€.
- **Postman:** Es gratuito por lo que su coste es 0€.
- **BitBucket:** Es gratuito por lo que su coste es 0€.
- **Trello:** Es gratuito por lo que su coste es 0€.
- **Heroku:** Es gratuito por lo que su coste es 0€.

En resumen, el precio de todo el software usado es de $149,97$€.

3.3.3. Hardware

El hardware utilizado ha sido exclusivamente mi equipo personal, un ordenador portátil ASUS con un Intel Core i7 de 8 núcleos a 2,6 GHz, 16 GB de RAM, un SDD de 256 GB y un disco duro mecánico de 1 TB. El precio de compra de este equipo fue de 1 200 € teniendo en cuenta que la vida media de un portátil de estas características en de 6 años, podemos extraer que el precio amortizado para un periodo de 3 meses sería de $1\,200 \div 6 \div 12 \times 3 = 50$ €.

3.4. Seguimiento del proyecto

El tiempo dedicado a cada sprint no ha sido el previsto debido la naturaleza de las tecnologías usadas y a una serie de problemas que han surgido a lo largo del desarrollo del proyecto y que inevitablemente han obligado a realizar cambios en el proceso de desarrollo. Estos imprevistos han obligado a reducir la cantidad de funcionalidad a desarrollar, sin embargo esto no ha impedido el correcto desarrollo de la funcionalidad mínima necesaria.

Cabe destacar dos problemas que por su gravedad obligaron a realizar cambios en las tecnologías usadas para el desarrollo, lo que provocó un retraso en la implementación de cierta funcionalidad.

3.4.1. Sprint 1

Para el primer sprint elegí las historias de usuario `HU03`, `HU04`, `HU05` y `HU06`, las cuales corresponden al desarrollo de la funcionalidad del departamento de diseño.

Sin embargo, dado que se trataba del primer sprint y todavía no había nada desarrollado, tuve que dedicar la mayor parte del tiempo al diseño del modelo entidad-relación de la base de datos, el cual pasó por diferentes iteraciones siguiendo las indicaciones del supervisor. A continuación, diseñé el diagrama de clases del modelo de datos.

Una vez acabados los diseños, procedí a implementar primero el modelo de datos y dado que estaba usando Kotlin Multiplatform con implementarlo una vez era suficiente para poder usarlo en el servidor y en el cliente. Esto fue todo lo que me dio tiempo a realizar en el primer sprint, lo que significa que no conseguí finalizar ninguna historia de usuario.

3.4.2. Sprint 2

Al comienzo del segundo sprint tenía una mejor idea del tiempo que me iba a llevar completar cada una de las tareas de modo que reduje la cantidad de historias de usuario de las cuatro del primer sprint a una sola, la historia `HU04`.

Durante este sprint implementé los DAO y todo lo necesario para la comunicación entre el

servidor y la base de datos usando Exposed, todo esto basándome en los diseños creados en el sprint anterior.

Por último, me dio tiempo a empezar a preparar lo necesario para comenzar con el desarrollo de la interfaz de usuario. Lo que significa que en este sprint tampoco completé ninguna historia de usuario.

3.4.3. Sprint 3

En el tercer sprint volví a elegir la historia `HU04` y además añadí una nueva, la historia `HU19`. El objetivo era finalmente completar las historias elegidas, sin más retrasos, las cuales correspondían con la primera parte de la vista del departamento de diseño.

Sin embargo, empecé a tener problemas a la hora de usar alguna biblioteca concreta de JavaScript para la que no existían adaptadores para poder llamarla desde Kotlin, y con el tema de la serialización de información para la comunicación cliente-servidor. Después de buscar soluciones en diversas páginas web, me topé con algunas incidencias [11] que eran muy similares a las mías, pero que no tenían solución, de modo que opté por no perder más tiempo buscando soluciones y desistí en el uso de Kotlin Multiplatform.

Este fue el primer gran problema que tuve durante el desarrollo del proyecto, lo me obligó a buscar otra tecnología para usar en la parte del cliente. Opté por usar TypeScript como lenguaje debido a que posee tipado estático, algo que considero necesario para cualquier proyecto cuya *codebase* adquiera un tamaño considerable. En cuanto al *framework* para crear las interfaces de usuario, con Kotlin estaba usando React puesto que era el único *framework* con adaptadores para poder ser usado con Kotlin, pero tanto este como cualquier otro *framework* era nuevo para mí, ya que no tenía experiencia previa con aplicaciones complejas cliente-servidor.

Finalmente elegí Vue.js [13] como *framework* para las interfaz del cliente. La principal razón fue que en el momento de la decisión empezamos a desarrollar un proyecto muy similar, en cuanto a las tecnologías usadas, en la universidad y un compañero propuso este *framework* porque lo estaba aprendiendo y decía que era perfecto para el proyecto. Me decidí por este *framework* tras echar un vistazo a la documentación, además tenía la ventaja de tener alguien con quien poder comentar dudas e ideas de diseño.

Este cambio de *framework* me obligó a re-implementar todo el modelo de datos de nuevo en el cliente ya que ahora usaba otro lenguaje. Pero ahora ya podía empezar realmente con el desarrollo de la interfaz de usuario.

Al finalizar el sprint tenía implementado el código necesario para completar las dos historias elegidas, a falta de realizar las pruebas necesarias.

3.4.4. Sprint 4

Para el cuarto sprint tenía pendiente probar las dos historias anteriores y como nuevas historias elegí las historias `HU02`, `HU03`, `HU05` y `HU06`, que corresponden a la segunda parte de la vista del departamento de diseño.

Tras completar las pruebas de las dos historias pendientes del sprint anterior, procedí a la implementación y realización de las pruebas necesarias para completar las tres nuevas historias. El único percance que tuve es que dediqué más tiempo del que me habría gustado a la implementación de una funcionalidad que permitiese abrir una ventana modal delante de otra ventana modal ya abierta, esto fue debido a cómo están diseñadas las ventanas modales en Bootstrap.

Al finalizar el sprint se habían completado todas las historias elegidas.

3.4.5. Sprint 5

Antes de empezar el quinto sprint añadí tres nuevas historias, `HU20`, `HU21` y `HU22`.

Con las dos primeras historias acabadas di por completada la vista del departamento de diseño, a falta de un par de historias que añadían funcionalidad no esencial que decidí dejar para el final de la estancia si había tiempo. La cuarta historia era el comienzo del departamento de atención al cliente, la completé y probé sin mayores percances.

3.4.6. Sprint 6

En el sexto sprint elegí las historias restantes para completar la funcionalidad básica del departamento de atención al cliente (`HU11`, `HU23` y `HU24`).

Cuando empecé a desarrollar el editor de incidencias me topé con una serie de retos, por así decirlo. Necesitaba usar una serie de componentes complejos los cuales no estaban disponibles directamente en la biblioteca Bootstrap, de modo que empecé a buscar alguna biblioteca que me proporcionase estos componentes a la vez que mantenía el estilo visual de Bootstrap.

Tras varias horas de búsqueda, me encontré dos nuevos *framework* que parecían hechos a medida para mi proyecto: Bulma [2] y Buefy [1].

- Bulma es un *framework* CSS muy similar a Bootstrap con estilos predefinidos.
- Buefy es una biblioteca de componentes para interfaces de usuario construido encima de Vue.js y Bulma, que proporciona componentes preconstruidos y preparados para funcionar directamente con Vue, a la vez que utilizan el estilo visual de Bulma.

La principal ventaja que presentan estas dos bibliotecas frente a Bootstrap es que vienen con muchos componentes predefinidos y que son muy fácilmente personalizables, lo que reduce

el tiempo necesario para desarrollar las interfaces.

Este fue el segundo gran problema que tuve a lo largo del proyecto y lo que provocó fue que tuviese que adaptar todas las interfaces, que ya había creado con Bootstrap, para que usasen Bulma y Buefy.

Puesto que tenía que modificar todos los componentes que tenía hasta el momento, aproveché para refactorizarlos y desacoplarlos los más posible de modo que fuese lo más fácil posible añadir nuevas vistas.

Al finalizar el sprint se habían completado las historias elegidas.

3.4.7. Sprint 7

En el séptimo sprint elegí las historias relacionadas con el sistema de autenticación de usuarios (`HU01`, `HU25` y `HU26`).

El hecho de añadir autenticación suponía que todas las rutas que hasta ahora estaban disponibles, ahora requerían un usuario perteneciente a un grupo concreto.

Este sprint lo completé sin mayores problemas.

3.4.8. Sprint 8

El octavo sprint lo dediqué al desarrollo de la vista del departamento del almacén (`HU13`, `HU14` y `HU15`).

Lo único que me costó un poco más de implementar fue el escáner de códigos de barras ya que la documentación de la biblioteca que utilicé era un poco pobre y me dio algún que otro problema. A parte de eso el sprint se desarrolló con normalidad y se completaron todas las historias.

3.4.9. Sprint 9

El noveno sprint lo dedique a añadir la funcionalidad de búsqueda y paginación (`HU07`, `HU08`, `HU09`, `HU10`, `HU12` y `HU27`), y a realizar optimizaciones en el cliente y el servidor.

Este sprint lo completé sin ningún percance.

3.4.10. Sprint 10

Durante el décimo y último sprint mi supervisor revisó y probó la aplicación y me indicó algunos cambios y mejoras menores para implementar, además de algún que otro fallo menor que se solucionó sin mucho problema.

Finalmente se preparó la aplicación para su despliegue añadiendo los diferentes archivos de configuración necesarios.

Historias sin finalizar

A causa de los retrasos producidos por los diferentes problemas que han surgido a lo largo del proyecto no ha sido posible implementar todas las historias de usuario, quedando sin finalizar las historias relacionadas con la vista de análisis (`HU16`, `HU17` y `HU18`).

#	Tarea	Tiempo
1	**Redacción propuesta Técnica**	**15**
1.1	**Captura de Requisitos**	**10**
1.1.1	Definición del Proyecto	4
1.1.2	Reunión con Supervisor	4
1.1.3	Definición de Alcance	2
1.2	**Planificación del Proyecto**	**5**
1.2.1	Planificación de los Sprint	5
2	**Desarrollo técnico del Proyecto**	**285**
2.1	**Definición de requisitos**	**8**
2.1.1	Pila del Producto	5
2.1.2	Requisitos Tecnológicos	3
2.2	**Análisis**	**7**
2.2.1	Diagrama de Clases	5
2.2.2	Validar Análisis	2
2.3	**Diseño**	**8**
2.3.1	Refinar Diagrama de Clase de Diseño	2
2.3.2	Proponer Plataforma de Despliegue	2
2.3.3	Validar Diseño	4
2.4	**Desarrollo del proyecto**	**250**
2.4.1	Sprint 1	25
2.4.2	Sprint 2	25
2.4.3	Sprint 3	25
2.4.4	Sprint 4	25
2.4.5	Sprint 5	25
2.4.6	Sprint 6	25
2.4.7	Sprint 7	25
2.4.8	Sprint 8	25
2.4.9	Sprint 9	25
2.4.10	Sprint 10	25
2.5	**Puesta en marcha**	**12**
2.5.1	Implantación	5
2.5.2	Formación	5
2.5.3	Entrega final	2

Cuadro 3.1: Planificación inicial

#	Descripción	Puntos de historia
HU01	Iniciar sesión en la aplicación	8
HU02	Ver lista de lotes de un modelo concreto	2
HU03	Ver información detallada de un lote	3
HU04	Crear un nuevo modelo de producto	5
HU05	Crear un nuevo lote de un modelo	5
HU06	Editar la información de un lote	5
HU07	Buscar un lote concreto	2
HU08	Ver incidencias de un modelo concreto	2
HU09	Ver incidencias de un lote concreto	2
HU10	Ver incidencias de un producto en concreto	2
HU11	Crear una incidencia para un producto	8
HU12	Buscar una incidencia concreta	2
HU13	Ver los movimientos de producto en el almacén	2
HU14	Registrar el movimiento de un producto	5
HU15	Registrar el movimiento de un producto mediante escaneo de código de barras	5
HU16	Ver gráfica de incidencias en el tiempo para un modelo concreto	13
HU17	Ver gráfica de incidencias en el tiempo para un lote concreto	13
HU18	Ver gráfica de incidencias en el tiempo para un producto concreto	13

Cuadro 3.2: Pila de producto inicial

#	Descripción	Puntos de historia	Sprint
HU01	Iniciar sesión en la aplicación	8	7
HU02	Ver lista de lotes de un modelo concreto	2	4
HU03	Ver información detallada de un lote	3	1, 4
HU04	Crear un nuevo modelo de producto	5	1, 2, 3, 4
HU05	Crear un nuevo lote de un modelo	5	4
HU06	Editar la información de un lote	5	4
HU07	Buscar un lote concreto	2	9
HU08	Ver incidencias de un modelo concreto	2	9
HU09	Ver incidencias de un lote concreto	2	9
HU10	Ver incidencias de un producto en concreto	2	9
HU11	Crear una incidencia para un producto	8	6
HU12	Buscar una incidencia concreta	2	9
HU13	Ver los movimientos de productos en el almacén	2	8
HU14	Registrar el movimiento de un producto	5	8
HU15	Registrar el movimiento de un producto mediante escaneo de código de barras	5	8
HU16	Ver gráfica de incidencias en el tiempo para un modelo concreto	13	
HU17	Ver gráfica de incidencias en el tiempo para un lote concreto	13	
HU18	Ver gráfica de incidencias en el tiempo para un producto concreto	13	
HU19	Ver lista de modelos disponibles	2	3,4
HU20	Ver información detallada de un modelo	2	5
HU21	Editar un modelo existente	3	5
HU22	Ver lista de todas las incidencias	2	5
HU23	Ver información detallada de una incidencia	2	6
HU24	Editar una incidencia existente	3	6
HU25	Añadir un nuevo usuario	2	7
HU26	Editar la información de un usuario	3	7
HU27	Buscar un modelo concreto	3	9

Cuadro 3.3: Pila de producto al final de proyecto y los sprints en los que se ha desarrollado cada historia

Capítulo 4

Análisis y diseño del sistema

4.1. Análisis del sistema

Puesto que se ha usado una metodología ágil, la herramienta utilizada para realizar el análisis el sistema ha sido las Historias de Usuario.

A continuación, se muestran en detalle las historias de usuario que se han desarrollado a lo largo del proyecto. Estas historias ya se mencionaron en el cuadro 3.3.

- HU01 - Iniciar sesión en la aplicación.

```
Requisito: Iniciar sesión
  Como trabajador
  Quiero iniciar sesión
  Para poder usar la aplicación
```

- HU02 - Ver lista de lotes de un modelo concreto.

```
Requisito: Ver lista de lotes
  Como trabajador del departamento de diseño
  Quiero ver una lista de los lotes existentes de un modelo
  Para saber que lotes existen actualmente
```

- HU03 - Ver información detallada de un lote.

```
Requisito: Ver información de un lote
  Como trabajador del departamento de diseño
  Quiero ver la información detallada de un lote
  Para conocer sus detalles
```

- HU04 - Crear un nuevo modelo de producto.

```
Requisito: Crear nuevo modelo
  Como trabajador del departamento de diseño
  Quiero crear un nuevo modelo
  Para registrar la información del nuevo modelo
```

- HU05 - Crear un nuevo lote de un modelo.

```
Requisito: Crear nuevo lote
  Como trabajador del departamento de diseño
  Quiero crear un nuevo lote
  Para mantener un control de los productos mediante códigos de barras
```

- HU06 - Editar un lote existente.

```
Requisito: Editar lote existente
  Como trabajador del departamento de diseño
  Quiero editar un lote existente
  Para actualizar su información
```

- HU07 - Buscar un lote concreto.

```
Requisito: Buscar lote concreto
  Como trabajador del departamento de diseño
  Quiero poder introducir una cadena en un cuadro de búsqueda y que se
      muestren las coincidencias con esa cadena
  Para encontrar un lote concreto
```

- HU08 - Ver incidencias de un modelo concreto.

```
Requisito: Ver incidencias de un modelo
  Como trabajador del departamento de atención al cliente
  Quiero poder introducir una cadena en un cuadro de búsqueda y que se
      muestren las incidencias cuyo modelo coincida con la cadena
  Para ver la lista de incidencias para ese modelo
```

- HU09 - Ver incidencias de un lote concreto.

```
Requisito: Ver incidencias de un lote
  Como trabajador del departamento de atención al cliente
  Quiero poder introducir una cadena en un cuadro de búsqueda y que se
      muestren las incidencias cuyo producto pertenezca al lote que
      coincida con la cadena
  Para ver la lista de incidencias para ese lote
```

- HU10 - Ver incidencias de un producto concreto.

```
Requisito: Ver incidencias de un producto
  Como trabajador del departamento de atención al cliente
  Quiero poder introducir una cadena en un cuadro de búsqueda y que se
      muestren las incidencias cuyo producto coincida con la cadena
  Para ver la lista de incidencias para ese producto
```

- HU11 - Crear una incidencia para un producto.

```
Requisito: Crear nueva incidencia
  Como trabajador del departamento de atención al cliente
  Quiero crear una nueva incidencia para un producto
  Para mantener un control de los problemas que surgen con los productos
```

- HU12 - Buscar una incidencia concreta.

```
Requisito: Buscar una incidencia
  Como trabajador del departamento de atención al cliente
  Quiero poder introducir una cadena en un cuadro de búsqueda y que se
      muestren las incidencias que tengan algún campo que coincida con la
      cadena
  Para encontrar una incidencia concreta
```

- HU13 - Ver los movimientos de productos en el almacén.

```
Requisito: Ver movimientos de productos
  Como trabajador del departamento de almacén
  Quiero ver una lista con todos los movimientos de productos
  Para saber cuándo entran y salen los productos
```

- HU14 - Registrar el movimiento de un producto.

```
Requisito: Registrar movimiento de productos
  Como trabajador del departamento de almacén
  Quiero registrar el movimiento de un producto
  Para dejar constancia de la entrada o salida de un producto
```

- HU15 - Registrar el movimiento de un producto mediante escaneo de código de barras.

```
Requisito: Registrar movimiento de productos con escaneo
  Como trabajador del departamento de almacén
  Quiero registrar el movimiento de un producto mediante el escaneo de los
      códigos de barras de los productos
  Para dejar constancia de la entrada o salida de un producto de una forma
      más rápida
```

- HU19 - Ver lista de modelos disponibles.

```
Requisito: Ver lista de modelos
  Como trabajador del departamento de diseño
  Quiero ver una lista de los modelos existentes
  Para saber que modelos existen actualmente
```

- HU20 - Ver información detallada de un modelo.

```
Requisito: Ver información de un modelo
  Como trabajador del departamento de diseño
  Quiero ver la información detallada de un modelo
  Para conocer sus detalles
```

- HU21 - Editar un modelo existente.

```
Requisito: Editar modelo existente
  Como trabajador del departamento de diseño
  Quiero editar un modelo existente
  Para actualizar su información
```

- HU22 - Ver lista de todas las incidencias.

```
Requisito: Ver lista de incidencias
  Como trabajador del departamento de atención al cliente
  Quiero ver una lista de todas las incidencias
  Para saber que incidencias existen actualmente
```

- HU23 - Ver información detallada de una incidencia.

```
Requisito: Ver información de una incidencia
  Como trabajador del departamento de atención al cliente
  Quiero ver la información detallada de una incidencia
  Para conocer sus detalles
```

- HU24 - Editar una incidencia existente.

```
Requisito: Editar incidencia existente
  Como trabajador del departamento de atención al cliente
  Quiero editar una incidencia existente
  Para actualizar su información
```

- HU25 - Añadir un nuevo usuario.

```
Requisito: Crear nuevo usuario
  Como administrador
  Quiero poder crear un nuevo usuario
  Para permitir a más trabajadores usar la aplicación
```

- HU26 - Editar la información de un usuario.

```
Requisito: Editar información de un usuario
  Como administrador
  Quiero poder editar la información de un usuario
  Para mantener actualizados los permisos de acceso o la contraseña
```

- HU27 - Buscar un modelo concreto.

```
Requisito: Buscar modelo
  Como trabajador del departamento de diseño
  Quiero poder introducir una cadena en un cuadro de búsqueda y que se
      muestren los modelos que coincidan con la cadena
  Para poder encontrar un modelo concreto
```

4.2. Diseño del modelo datos

4.2.1. Base de datos

El diseño del modelo entidad-relación de la base de datos ha pasado por varias iteraciones hasta conseguir un modelo adecuado a las necesidades del proyecto. El resultado final se muestra en la figura 4.1, donde se pueden observar las diferentes categorías de datos marcados en colores diferentes. A continuación, se explica un poco por encima en que consiste cada categoría.

- Modelos: representa los diferentes modelos de productos que están disponibles, cada modelo tiene una imagen que ayude a identificarlo.
- Productos: representa los lotes de productos, donde cada lote está fabricado por un fabricante concreto y contiene una lista de números de serie que identifica cada producto individual.
- Albaranes: representa los albaranes de entrada y salida que sirven para mantener un control de los movimientos de cada producto.
- Incidencias: representa las incidencias que se producen en cada producto individual y que se registran en el sistema.
- Clientes: representa a cada uno de los clientes asociados a cada incidencia para controlar quien es el propietario y la cantidad de incidencias de cada cliente, entre otras cosas.

- Autenticación: representa los diferentes usuarios que pueden usar la aplicación, así como a qué departamentos tienen acceso.

4.2.2. Diagrama de clases

A partir del modelo entidad-relación creé el diagrama de clases que se usaría en la parte del servidor para representar todos los datos enviados y recibidos, por la base de datos y los clientes. Este diagrama se muestra en la figura 4.2.

En la parte del cliente utilicé una representación muy similar a la del diagrama de clases, simplemente adaptando los tipos de algunas variables a los tipos disponibles en la plataforma.

4.3. Diseño de la arquitectura del sistema

En esta sección se presenta la arquitectura del sistema desarrollado, así como una explicación en detalle de las herramientas y tecnologías usadas. El sistema está construido utilizando una arquitectura de tres capas: cliente, servidor y base de datos. En la figura 4.3 se puede observar un esquema de esta arquitectura.

La comunicación con el cliente se realiza mediante el envío y la recepción de datos codificados en formato JSON[1] a través de conexiones seguras HTTPS[2] a una API[3], protegida mediante un sistema de autenticación, que vive en la parte del servidor.

La comunicación con la base de datos se realiza mediante una conexión directa entre el servidor y la propia base de datos usando una biblioteca JDBC[4].

4.3.1. Cliente

Para la parte del cliente he utilizado el *framework* de JavaScript Vue.js, por las razones expuestas en la sección 3.4. Este *framework* está escrito en JavaScript y pensado para ser usado con JavaScript, pero también tiene soporte para ser utilizado con TypeScript y OOP[5], algo que prefiero debido a que casi toda mi experiencia se basa en lenguajes de programación orientados a objetos.

El uso de TypeScript me ha permitido aprovechar los beneficios de la programación orientada a objetos, como por ejemplo la herencia, agilizando y simplificando el desarrollo de la interfaz, lo que me ha permitido crear simplemente cinco tipos de componentes base, los cuales he extendido

[1] *JavaScript Object Notation*
[2] *Hypertext Transfer Protocol Secure*
[3] *Application Programming Interface*
[4] Java Database Connectivity
[5] Object Oriented Programming

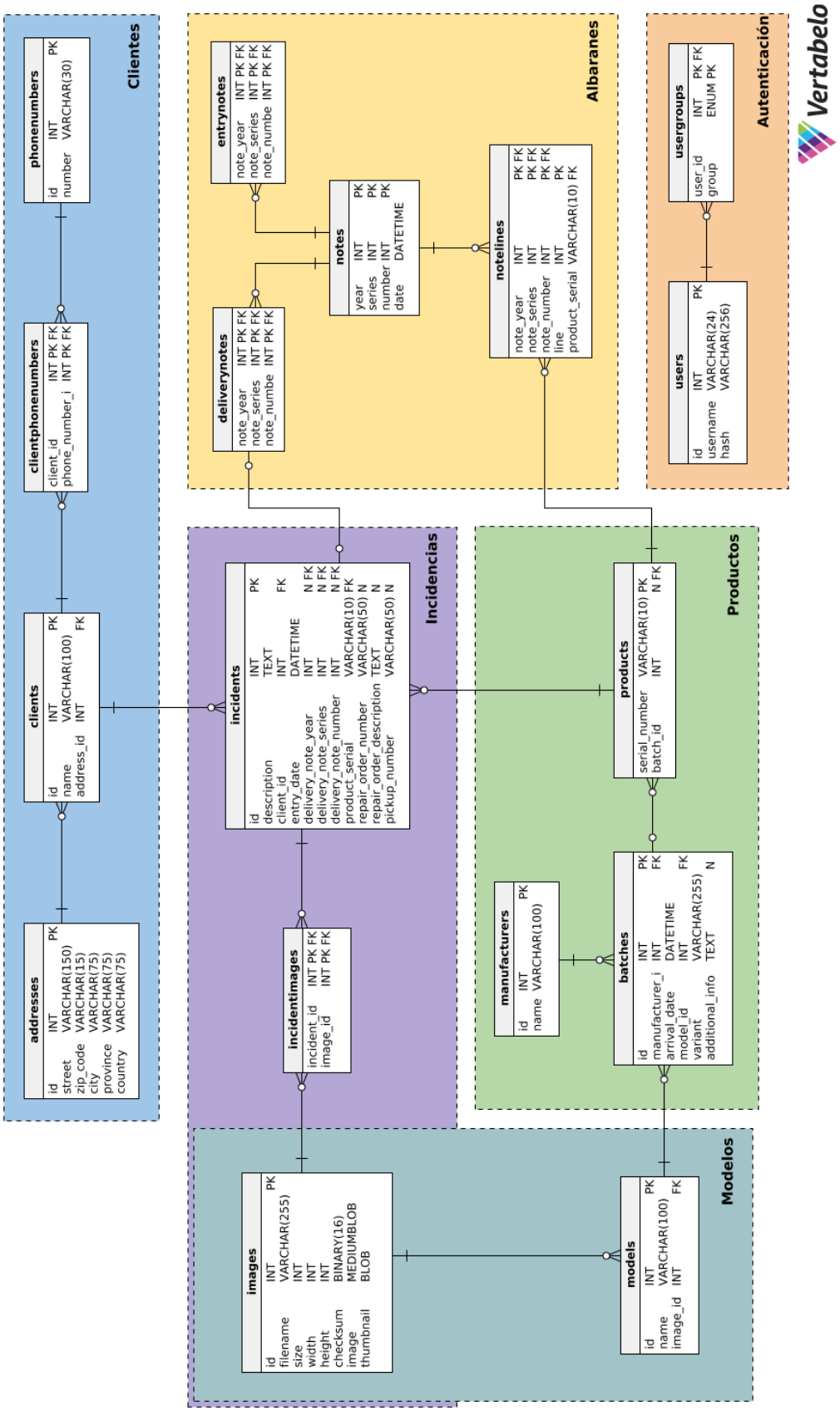

Figura 4.1: Modelo entidad-relación

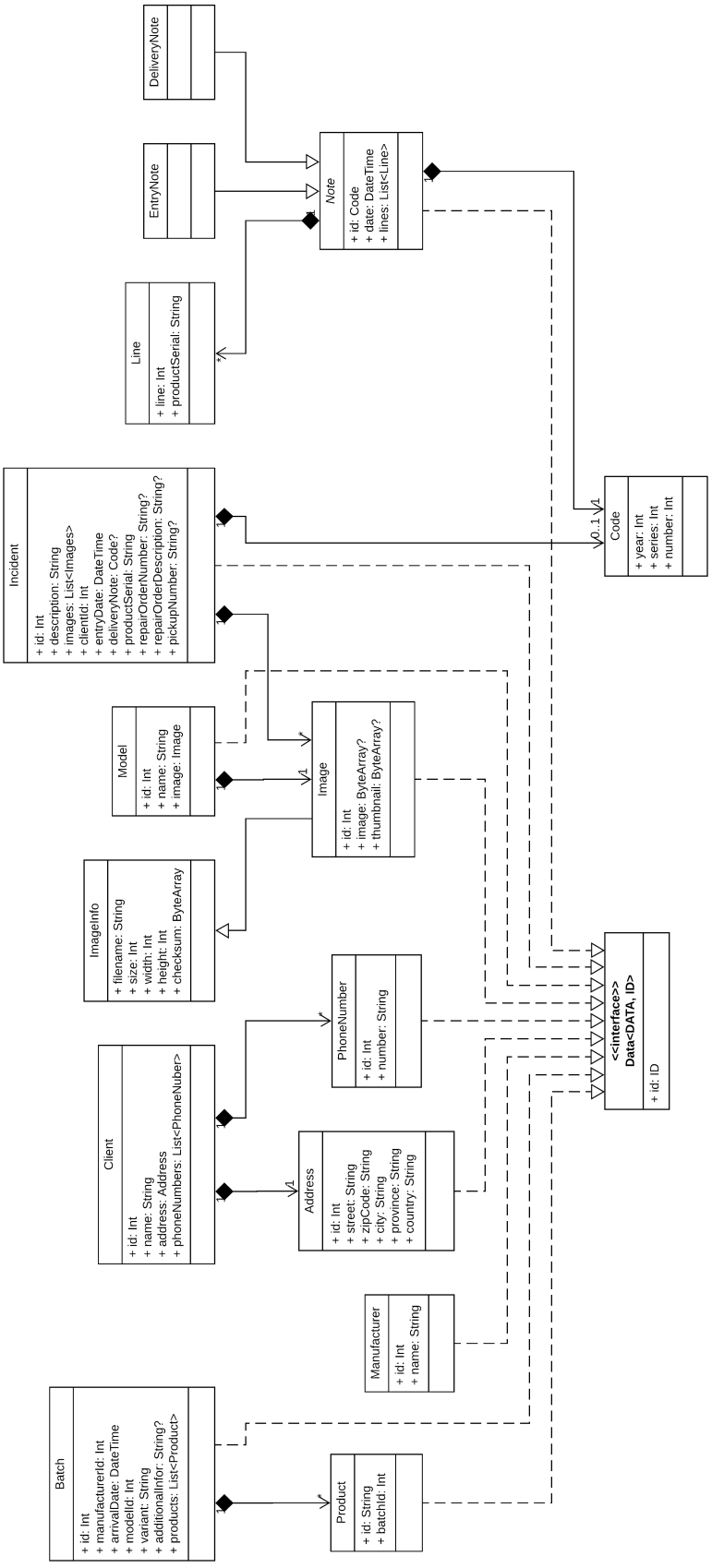

Figura 4.2: Diagrama de clases del modelo de datos en el servidor

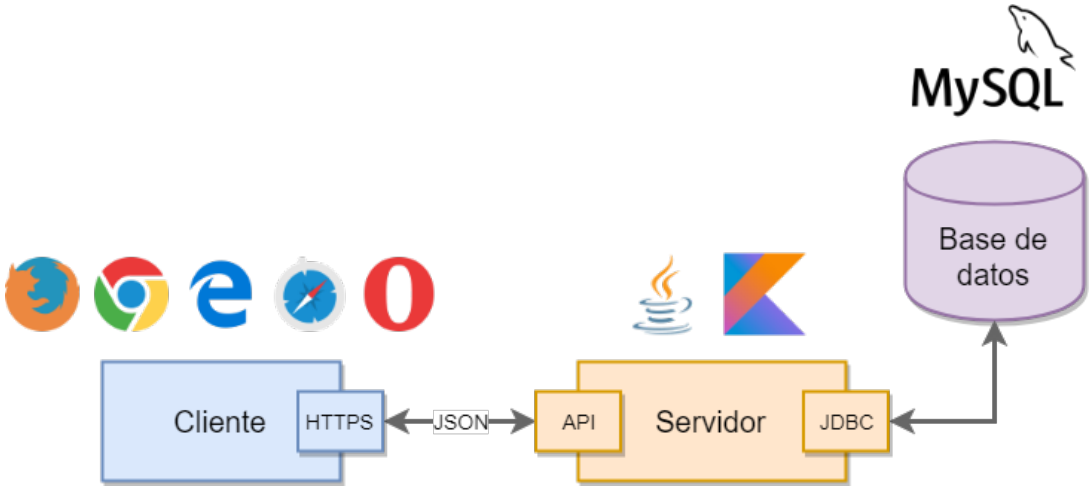

Figura 4.3: Esquema de la arquitectura

para que puedan trabajar con cada uno de los tipos de datos del modelo de datos. Los cinco tipos de componentes se explican a continuación.

- *Home*: Vista general de un departamento.
- *List*: Componente que muestra una serie de elementos.
- *Editor*: Ventana que sirve tanto para crear un elemento como para editar sus datos.
- *Viewer*: Ventana que muestra los datos de un elemento.
- *Input*: Componente que sirve para introductor un dato de un elemento.

Otra ventaja de usar TypeScript es la genericidad; esto junto con la herencia me ha permitido crear, entre otros componentes, una lista base genérica con una serie de propiedades, también genéricas como pueden ser el servicio proveedor de datos, un editor de datos o un visor de datos, cuyos tipos serán clases abstractas o interfaces genéricas. De este modo cuando implementemos, por ejemplo, la lista de modelos del departamento de diseño, simplemente tenemos que extender la lista base y especificar `Model` como parámetro genérico, por lo tanto a la hora de sobreescribir alguna de las propiedades mencionadas antes tenemos la seguridad en tiempo de compilación que el servicio o el editor que hemos especificado puede trabajar con modelos y no hemos pasado por error un editor de lotes.

Todo esto junto con el uso de un IDE, como es IntelliJ, ha favorecido y facilitado la organización de todo el código en paquetes dependiendo de la funcionalidad que realizan las diferentes clases.

Esta organización de paquetes para la parte del cliente, que se puede observar en la figura 4.4, se desglosa a continuación.

- `src` es el paquete raíz de donde cuelgan todo el resto de paquetes. Aquí se encuentran el punto de entrada al programa y algunos archivos de configuración de JavaScript y Vue.js

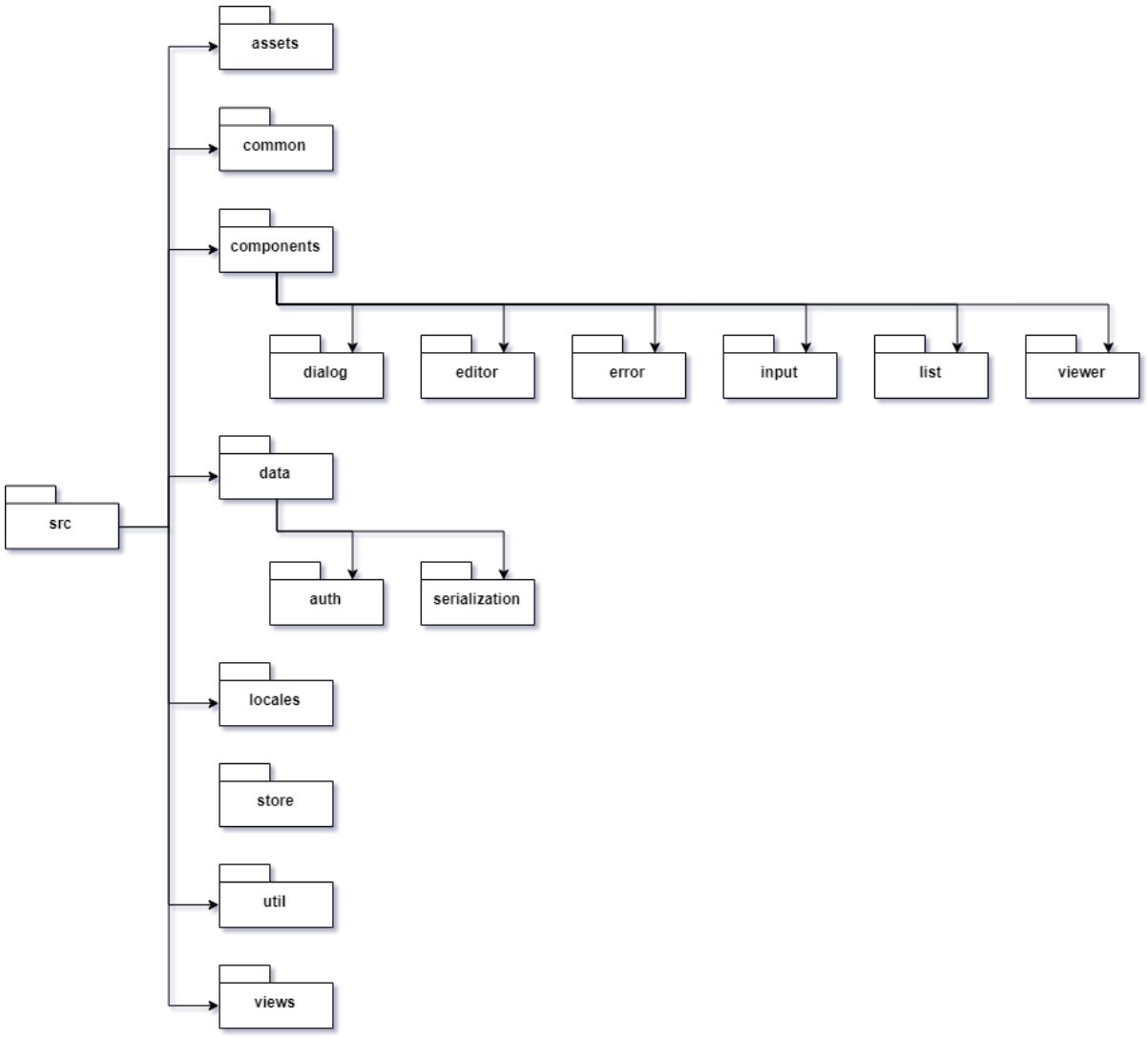

Figura 4.4: Organización de paquetes en la parte del cliente

- `assets` contiene recursos, por ejemplo, hojas de estilo CSS.
- `common` contiene código común utilizado en todo el proyecto, en este caso comunicación con la API del servidor y el servicio de autenticación.
- `components` contiene todas las clases e interfaces que se usan para crear los diferentes componentes, mencionados anteriormente, que forman las vistas de cada departamento.
  - `dialog` contiene algunas clases e interfaces para crear diálogos como el de confirmación de borrado.
  - `editor` contiene las implementaciones de los diferentes editores.
  - `error` contiene mensajes de error para cuando se recibe un código de estado de error de la API como puede ser un código 404.
  - `input` contiene las implementaciones de las diferentes entradas de datos.
  - `list` contiene las implementaciones de las diferentes listas.
  - `viewew` contiene las implementaciones de los diferentes visores.

- `data` contiene todas las clases que representan los objetos del modelo de datos.
  - `auth` contiene clases que representan datos relacionados con la autenticación, por ejemplo, la clase `User` que almacena el nombre de usuario y un *token* entre otras cosas.
  - `serialization` contiene las definiciones de como serializar los diferentes tipos de datos.
- `locales` contiene las traducciones de todos los textos que se muestran en la aplicación, actualmente solo está en castellano.
- `store` contiene código relacionado con la biblioteca Vuex.
- `util` contiene utilidades generales.
- `view` contiene las vistas base de cada departamento.

4.3.2. Servidor

En el servidor el *framework* que he usado ha sido Ktor, del cual hablé en la sección 2.3.2. Ktor es un *framework* modular que proporciona todas las herramientas necesarias para crear páginas web tan completas como puedas imaginarte. Para utilizar nuevos componentes y para añadir diferentes funcionalidades que no estén presentes por defecto, como serialización o autenticación, basta con añadir la dependencia de Maven correspondiente.

En mi caso, las dependencias adicionales que tuve que usar fueron:

- `server-netty`: Permite usar Netty como *engine* para el servidor.
- `gson`: Permite la serialización y de-serialización de datos a formato JSON de manera automática usando la biblioteca GSON.
- `auth-jwt`: Permite añadir autenticación de rutas mediante JWT[6].
- `network-tls`: Permite realizar conexiones seguras HTTPS.

Endpoints

Toda API REST proporciona una serie de rutas conocidas como *endpoint*, que corresponden a unas URL determinadas a las cuales se pueden realizar peticiones HTTP, utilizando una serie de verbos [9] dependiendo del tipo de operación que se quiera realizar, y normalmente se recibe un código de estado [10] como respuesta que indica si se produjo algún error o si todo funcionó correctamente.

En este caso las dependencias que he usado han sido `gson` y `network-tls`. La primera se encargaba de serializar y de-serializar las clases que representan los datos en el servidor a formato JSON que es el que entiende el cliente. La segunda se encargaba de habilitar el envío y la recepción de información usando de forma segura usando una conexión HTTPS.

[6] JSON Web Tokens

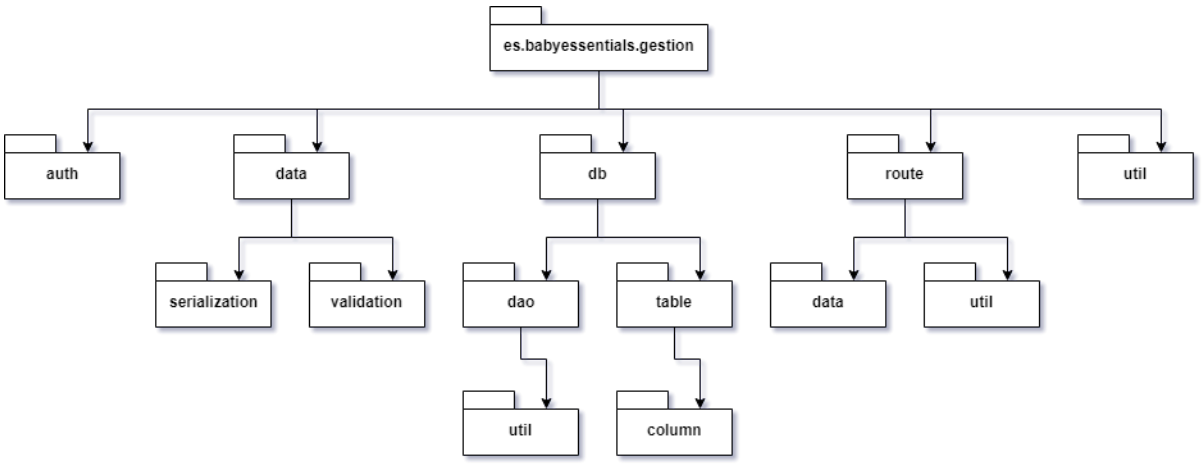

Figura 4.5: Organización de paquetes en la parte del servidor

Seguridad

Debido a que se trata de una aplicación de uso interno en la empresa la seguridad es un tema importante, es por esto que todos los *endpoints* están protegidos mediante un sistema de autenticación mediante usuario y contraseña, y el uso de identificadores JWT para mantener las sesiones activas.

Las contraseñas no se almacenan nunca directamente en el servidor, sino que estas se cifran utilizando la función PBKDF2[7] [15] junto con el algoritmo SHA-256. La razón por la que usé la función PBKDF2 en lugar de simplemente SHA-256 es porque esta proporciona un grado adicional de protección contra ataques de fuerza bruta, además de estar recomendado por la IETF[8] en el RFC 8018 [3].

En este caso la dependencia que he usado ha sido `auth-jwt`, que lo que hace es procesar el token cuando se envía o se recibe para serializarlo o de-serializarlo, para poder usarlo a través de una representación en forma de una clase de Kotlin, en lugar de tener que trabajar directamente con una cadena.

Estructura

El uso de un IDE como IntelliJ ha facilitado mucho el desarrollo y organización del proyecto, entre otras cosas porque proporciona una gran cantidad de herramientas para la refactorización del código. Esto junto con el uso de diversos patrones de diseño me ha permitido, a medida que ha ido avanzando el proyecto, organizar el código en paquetes con funcionalidades concretas y diferenciadas.

Esta organización de paquetes para la parte del servidor, que se puede observar en la figura 4.5, se desglosa a continuación.

[7] Password-Based Key Derivation Function 2
[8] Internet Engineering Task Force

- `es.babyessentials.gestion` es el paquete raíz de donde cuelgan todo el resto de paquetes, el único código presente en este paquete es la clase principal con el punto de entrada.
  - `auth` contiene todo lo necesario para realizar la autenticación de clientes: credenciales, la función *hash*, etc.
  - `data` contiene todas las clases que representan los objetos del modelo de datos.
    - `serialization` contiene algunas funciones de serialización personalizadas y optimizadas para ciertos tipos de datos.
    - `validation` contiene los mecanismos de validación de datos.
  - `db` contiene todo lo relativo a la comunicación con la base de datos.
    - `dao` contiene los DAO[9] necesarios para recuperar y almacenar datos.
      - `util` contiene algunas utilidades usadas por los DAO, como pueden ser clases para paginación de datos.
    - `table` contiene todas las definiciones de las tablas de la base de datos, necesarias para trabajar con la biblioteca Exposed.
      - `columns` contiene especificaciones de tipos de columnas que no proporciona la biblioteca.
  - `route` contiene las definiciones de las rutas a las que se puede conectar un cliente, en su mayoría *endpoints*.
    - `data` contiene únicamente los *endpoints* relativos a modelo de datos.
    - `util` contiene utilidades para recuperar información de las peticiones que realizan los clientes.
  - `util` contiene varias utilidades generales.

4.4. Diseño de la interfaz

Para diseñar las interfaces primero es necesario saber que quiere ver y poder hacer cada usuario. En este proyecto tenemos cuatro tipos de usuario con las siguientes necesidades.

- Administrador: necesita ver los usuarios existentes y poder editar los grupos a los que pertenecen y cambiar sus contraseñas.
- Departamento de diseño: necesita ver los modelos y lotes de productos existentes, así como poder crear otros nuevos o editar los ya existentes.
- Departamento de atención al cliente: necesita ver las incidencias que se han producido, así como poder crear nuevas o editar las existentes.
- Departamento del almacén: necesita ver los movimientos de productos que entran o salen del almacén así como poder registrar nuevos movimientos o editar los existentes.

En cuanto a los grupos a los que puede pertenecer un usuario, son los siguientes:

[9] Data access object

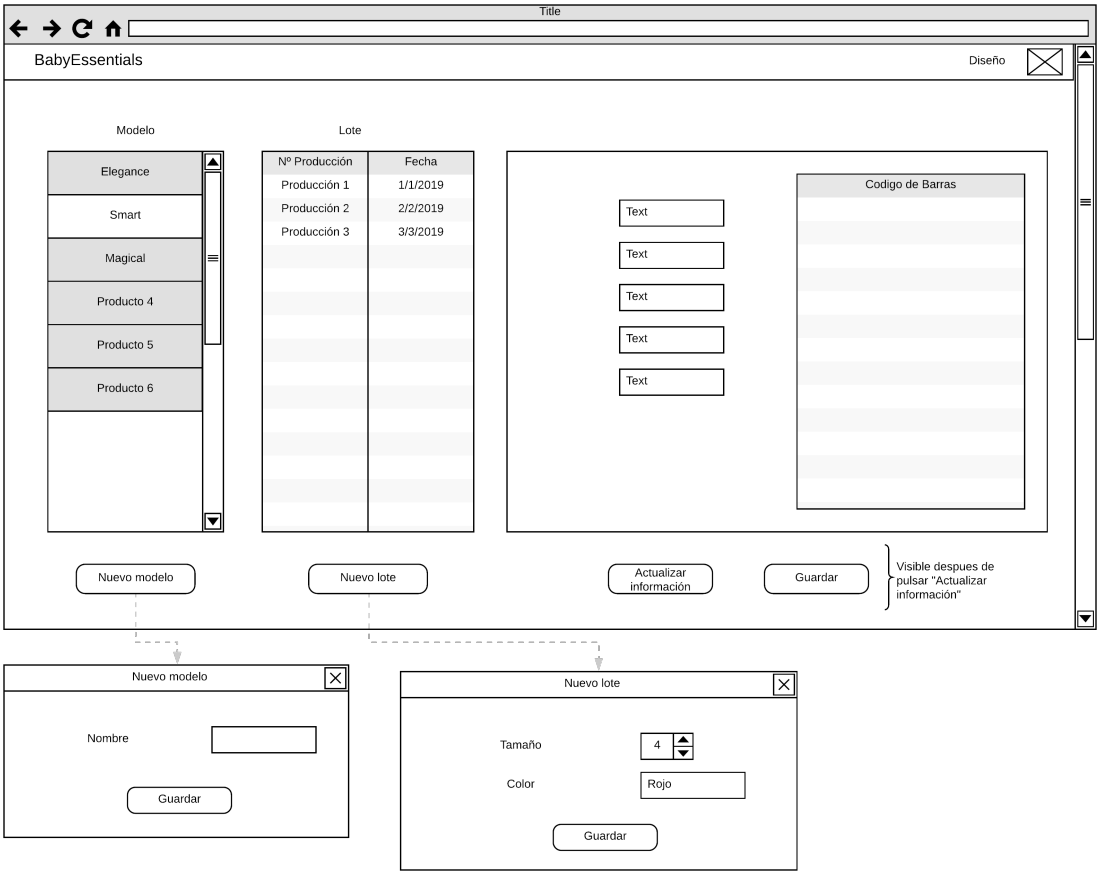

Figura 4.6: Boceto vista departamento de diseño

- `ADMIN`: Capacidad de crear, editar y borrar usuarios y cambiar a que grupos pertenecen.
- `DESIGN`: Acceso a la vista del departamento de diseño.
- `SUPPORT`: Acceso a la vista del departamento de atención al cliente.
- `WAREHOUSE`: Acceso a la vista del departamento del almacén.

Sabiendo esto, el diseño de la interfaz empezó con la realización de unos bocetos al principio del proyecto para tener una idea aproximada de la distribución de los componentes en la pantalla, así como de que información se mostraría y la secuencia de pantallas por las que se navegaría dependiendo de lo que se quisiese hacer. Los bocetos en cuestión se pueden observar en las figuras 4.6, 4.7, 4.8.

Mas adelante, después de estudiar mejor el *framework* que iba a usar para las interfaces de usuario, opté por un diseño un poco diferente al de los bocetos iniciales. Este nuevo diseño se puede definir como una serie de reglas a seguir. la

1. Cada departamento posee una vista individual donde se muestra una o varias listas de los elementos principales con los que se trabaja.

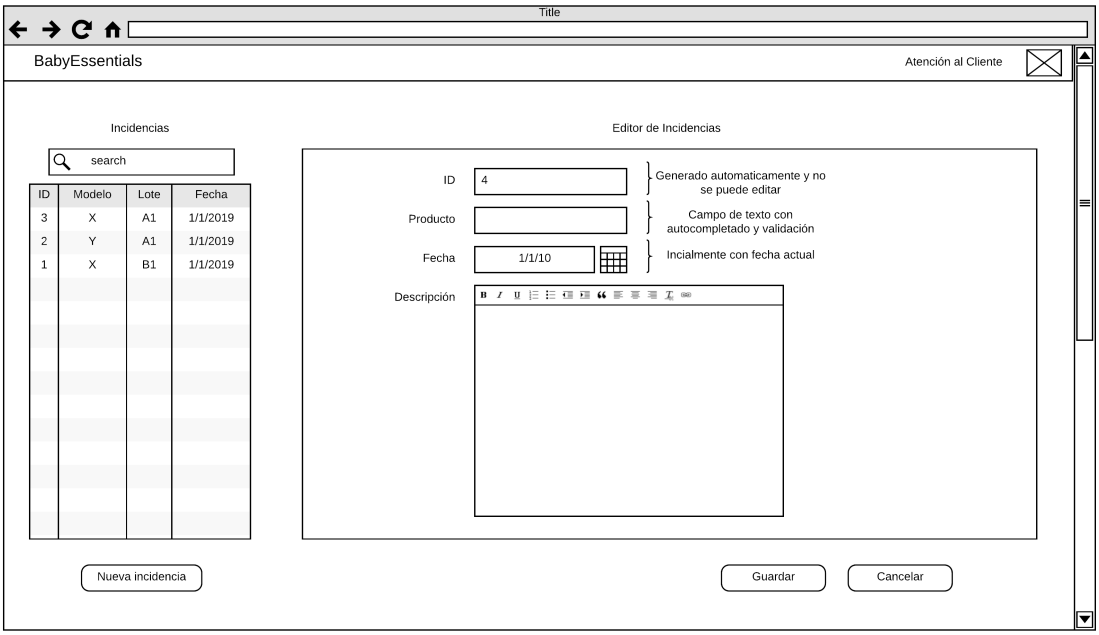

Figura 4.7: Boceto vista departamento de atención al cliente

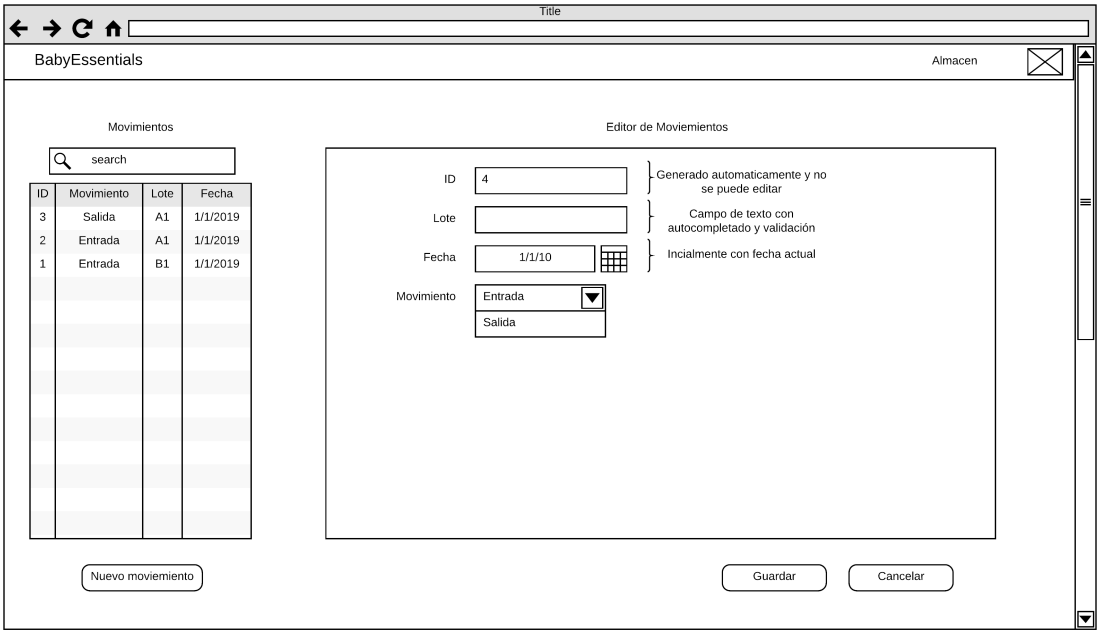

Figura 4.8: Boceto vista departamento del almacén

2. Cada lista posee un botón en la parte superior para añadir nuevos elementos.
3. Cada lista posee una o varias columnas con los campos más relevantes.
4. Cada elemento de la lista posee uno o varios botones para llevar a cabo acciones como: editar información, ver información completa o borrar el elemento.
5. Las acciones *crear* y *editar* mostrarán un editor en una ventana modal, superpuesta a la vista actual, que contendrá una serie de entradas de datos que variarán dependiendo del tipo de dato a introducir.
6. Las entradas de elementos complejos y reutilizables, como puede ser un fabricante, harán uso de un campo de texto con autocompletado y un botón para crear un nuevo elemento, el cual abrirá un nuevo editor.
7. La acción de ver información completa mostrará un visor en una ventana modal, al igual que el editor, que contendrá únicamente texto no editable.
8. La acción de borrar mostrará un mensaje de confirmación con dos botones, uno para confirmar el borrado y otro para cancelarlo.
9. Adicionalmente cada lista poseerá un cuadro de búsqueda en la parte superior para filtrar los elementos mostrados, así como un selector para elegir la página actual y cuantos elementos se muestran por página.

Siguiendo estas reglas y consultando la documentación del *framework* que estaba usando, primero Bootstrap y más tarde Bulma, fui desarrollando las interfaces eligiendo los componentes más adecuados dependiendo de la información a mostrar, la funcionalidad y la distribución de los elementos.

Además de todo lo mencionado, todos los componentes que sirven para la introducción de datos tienen un sistema de validación que impide la introducción de datos no válidos.

El resultado final fue una interfaz de la que estoy muy contento y que a algunos de los trabajadores de la empresa llamó la atención por el estilo elegido.

A continuación, en las figuras 4.9, 4.10, 4.11, 4.12 y 4.13, se muestran las pantallas que vería un trabajador del departamento de diseño si fuese a añadir, desde un ordenador, un nuevo lote para un modelo existente.

No obstante, la interfaz es completamente *responsive* por lo que se puede adaptar a cualquier tipo de pantalla. En las figuras 4.14 y 4.15 se muestra como ser vería la interfaz en un smartphone.

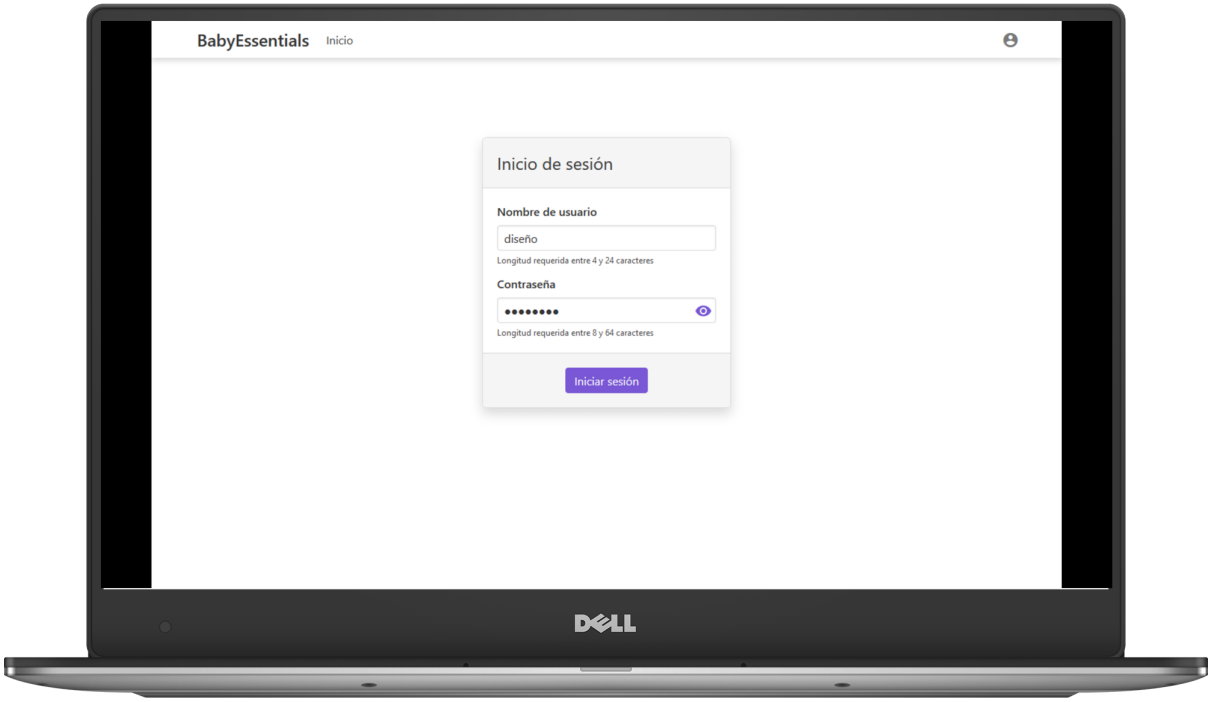

Figura 4.9: Pantalla de inicio de sesión en un ordenador

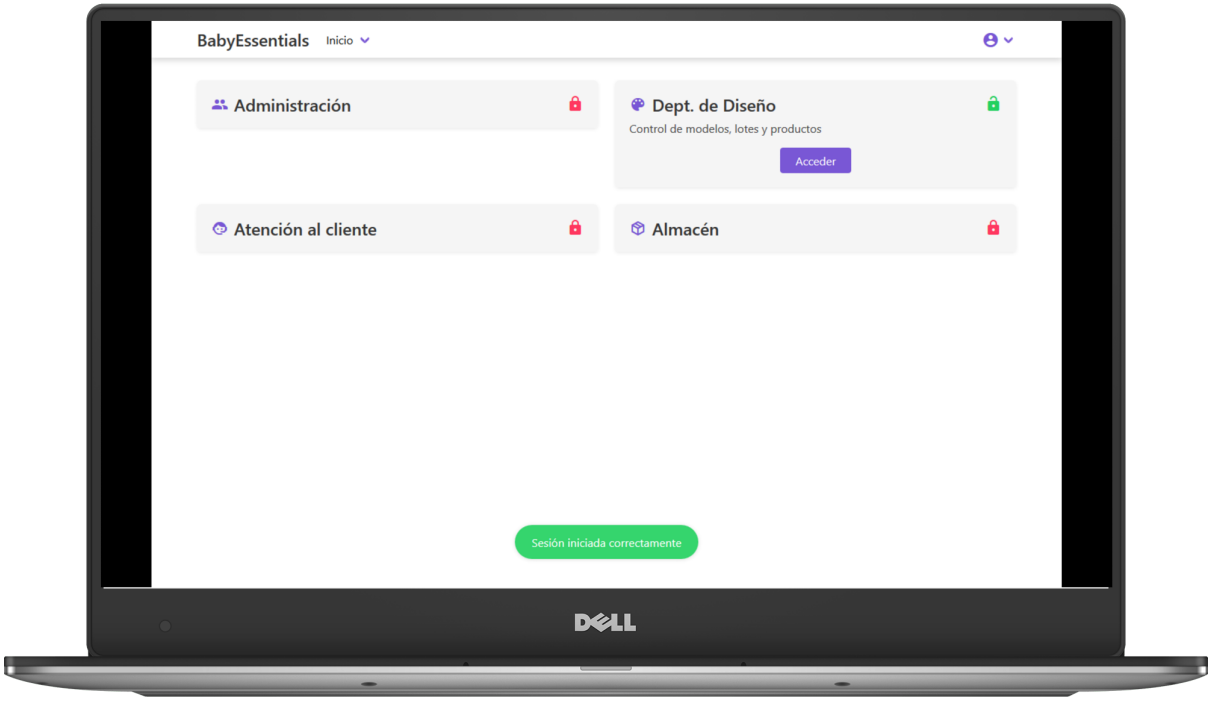

Figura 4.10: Vista principal en un ordenador

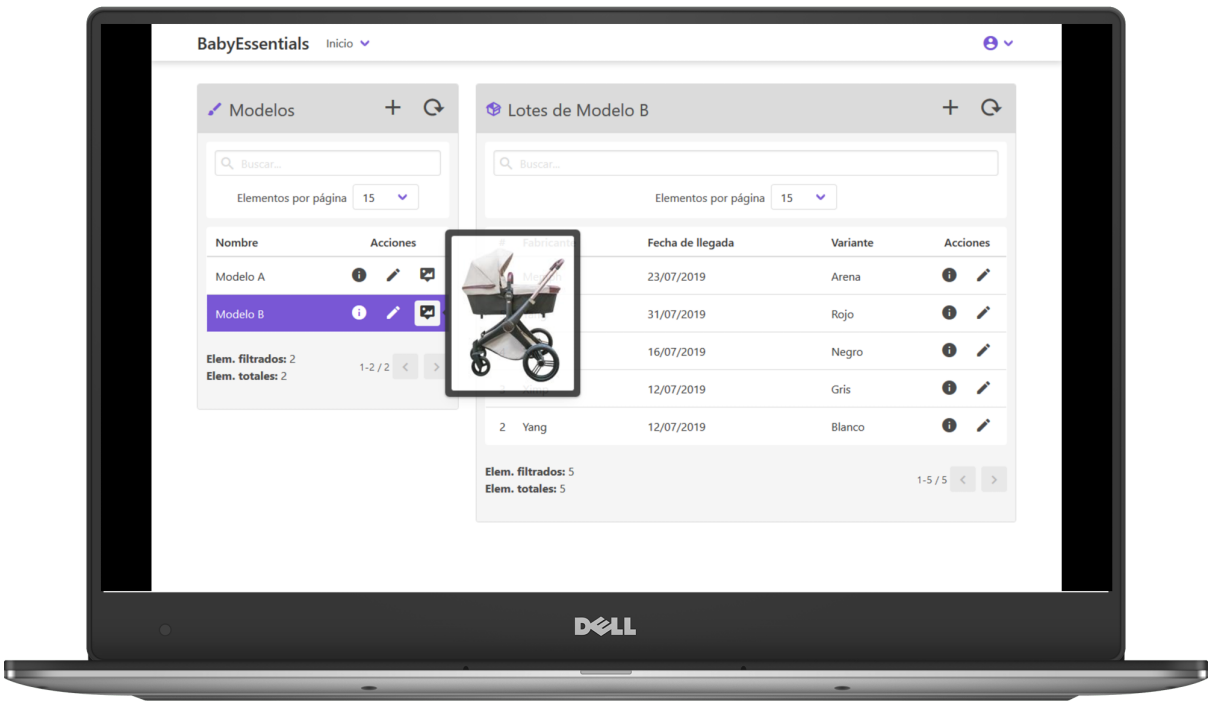

Figura 4.11: Vista del departamento de diseño en un ordenador

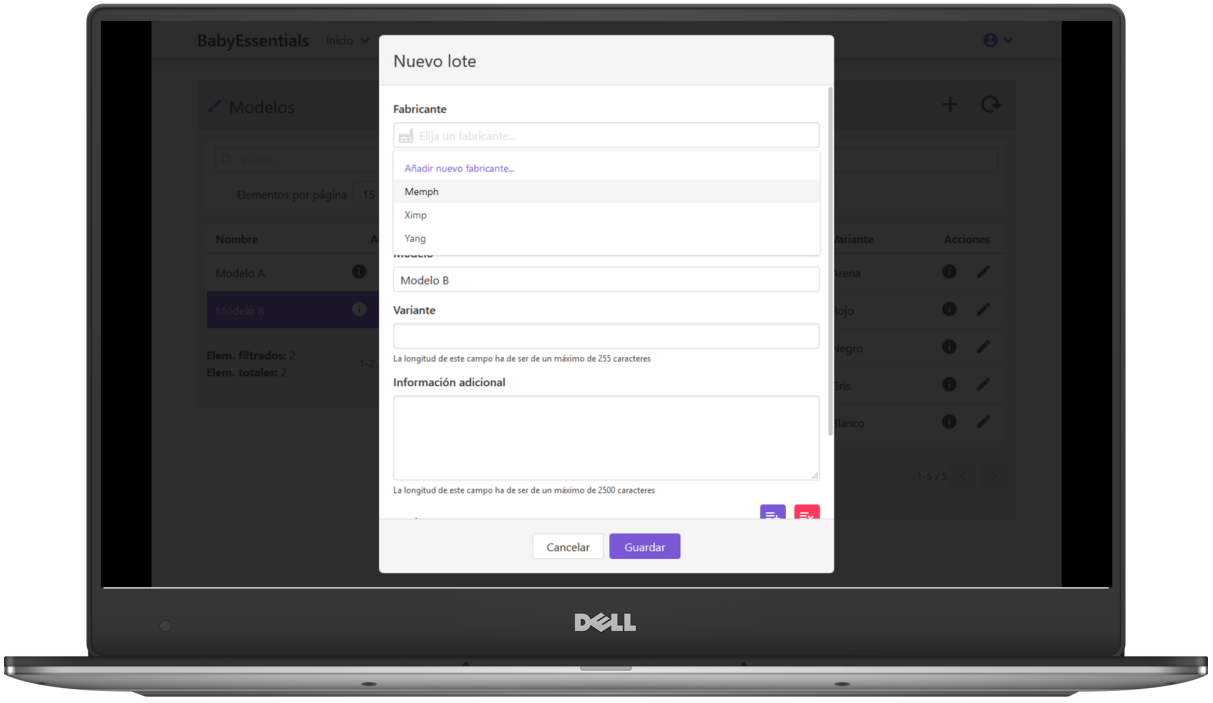

Figura 4.12: Vista del editor de lotes, creando un nuevo albarán, en un ordenador

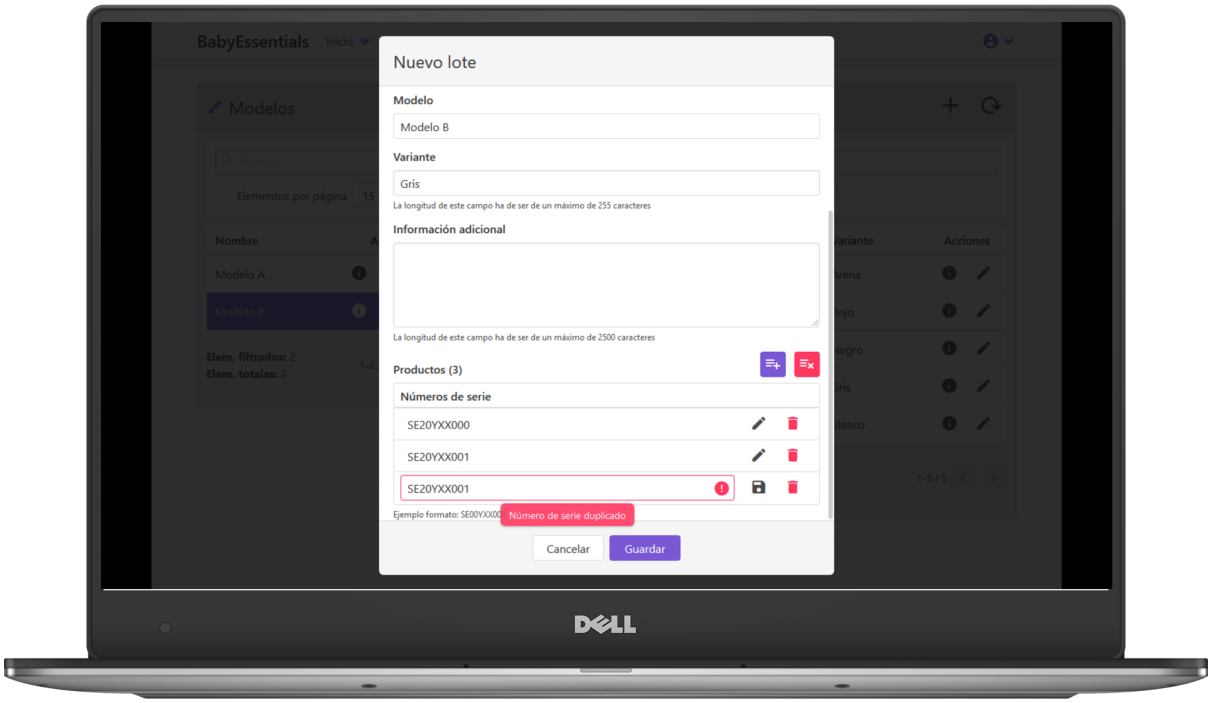

Figura 4.13: Vista del editor de lotes, con un mensaje de validación, en un ordenador

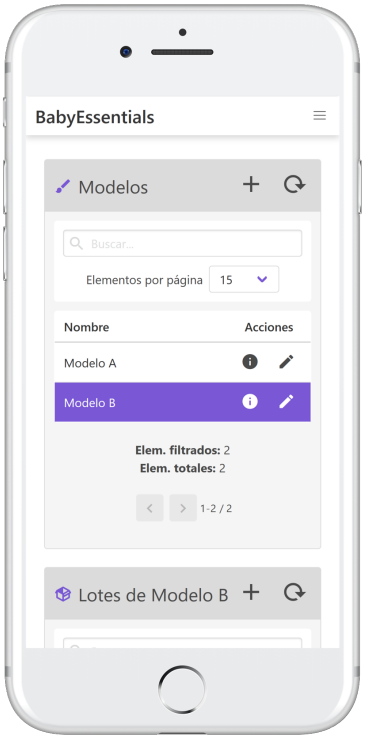

Figura 4.14: Vista del departamento de diseño en un smartphone

Figura 4.15: Vista del editor lotes, creando un lote nuevo, en un smartphone

Capítulo 5

Implementación y pruebas

5.1. Detalles de implementación

En esta sección se discuten los detalles de implementación y las particularidades del proyecto.

5.1.1. Base de datos

Tal y como he comentado con anterioridad, he hecho uso de la biblioteca Exposed para gestionar todo el tema de conexiones y trabajo con la base de datos desde el servidor.

Tablas

Una de las cosas más destacables de esta biblioteca es la posibilidad de crear definiciones de tablas directamente en Kotlin en forma de clases con propiedades gracias al uso de un DSL, eliminando la necesidad de escribir código SQL, a excepción de situaciones que requieran de construcciones complejas que no es nuestro caso.

Un ejemplo de este DSL se puede observar en la figura 5.1, donde se muestra la definición de la tabla `Batches` que representa los lotes de productos.

DAO

Otra de las partes en las que Exposed hace uso de un DSL es a la hora de realizar las consultas a la base de datos; esto lo hace gracias al uso de una mezcla entre programación funcional e interfaces fuidas [14].

Un ejemplo de una consulta a la base de datos que hace uso de este DSL se muestra en la figura 5.2.

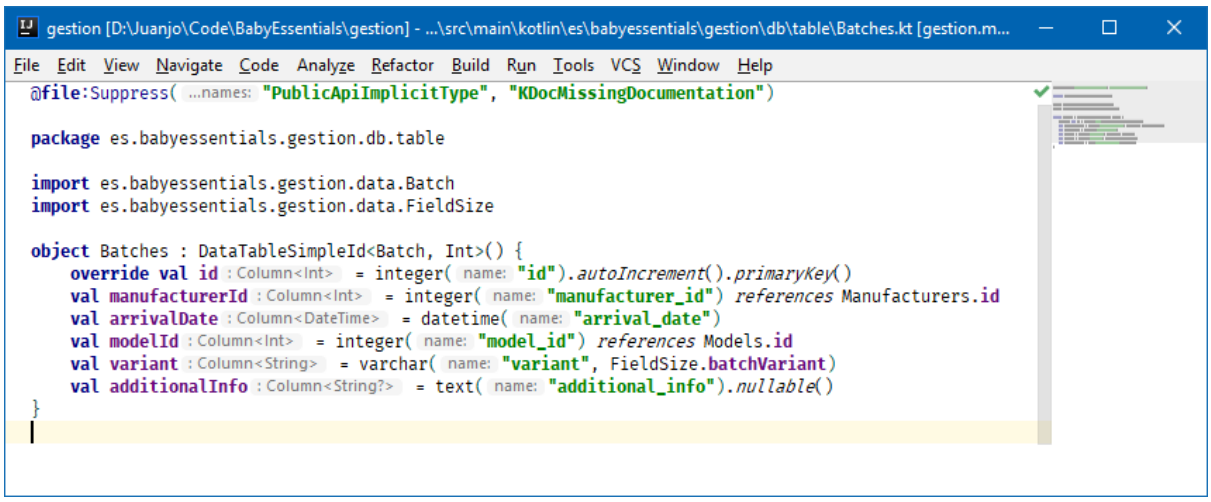

Figura 5.1: Extracto de definición de tabla mediante DSL

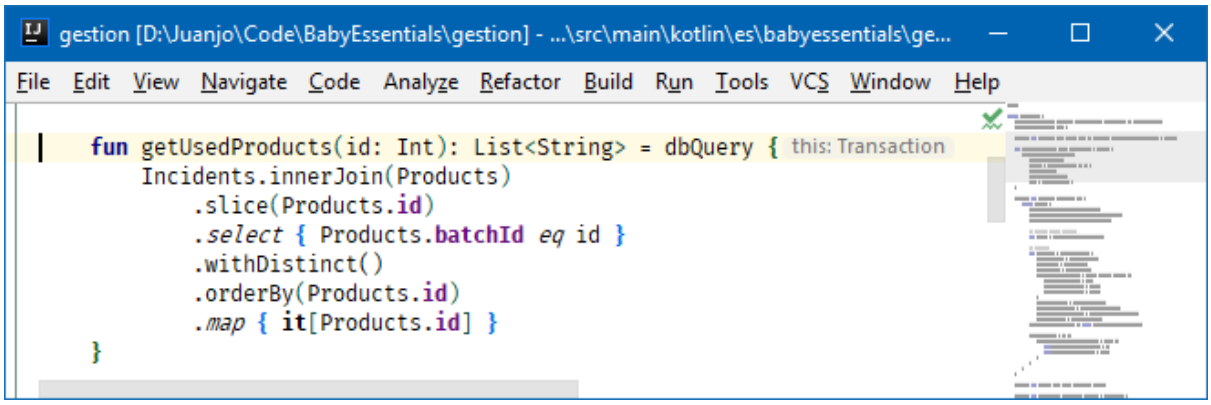

Figura 5.2: Extracto de consulta a la base de datos mediante DSL

5.1.2. Seguridad

Completando la información de la sección 4.3.2 sobre la seguridad presente en la aplicación, a continuación voy a explicar un poco más en detalle cómo funciona el proceso de autenticación de usuarios y qué mecanismos de seguridad existen.

El primer paso es introducir el usuario y contraseña; estos datos se envían mediante una petición POST al *endpoint* de inicio de sesión (`/api/auth/login`) a través de una conexión segura HTTPS al servidor, donde se encripta la contraseña y se compara, junto con el usuario, con las credenciales ya almacenadas en la base de datos. Si la combinación de usuario y contraseña son válidos, se generará un *token* que contendrá el id del usuario, un UUID[1] de la sesión que servirá para comprobar si el token es válido, y dos fechas, una de creación y una de espiración, que indican el rango de tiempo en el que el *token* es válido, normalmente 15 minutos.

Este token se envía al cliente como respuesta a la petición POST inicial, el cliente entonces lo almacena en el *localStorage* del navegador. Este localStorage es una memoria que persiste

[1] Universally unique identifier

a través de toda la sesión y solo se borra si se vacía la memoria del navegador o se cierra de la sesión con el botón correspondiente. A partir de ahora todas las peticiones que se realizan incluyen una cabecera con el token en cuestión para poder acceder a todas las rutas protegidas. Además, en el momento que se recibe el *token*, se inicia un temporizador de 10 minutos que al finalizar realizará una petición al servidor para que le proporcione un nuevo token para los próximos 15 minutos.

En la parte del servidor, cuando se recibe una petición a un *endpoint* protegido, se realizan una serie de comprobaciones antes de responder.

1. Se comprueba si se ha recibido un token.
2. Se comprueba que el token no haya expirado.
3. Se comprueba que si el usuario pertenece a algún grupo con acceso.
4. Si todo el correcto se responde normalmente, si no se devuelve el código de error correspondiente.

5.2. Verificación y validación

En esta sección se discute todo lo referente al proceso de verificación y validación seguido durante todo el proyecto.

5.2.1. Análisis estático de código

Entre las innumerables herramientas que proporciona IntelliJ se encuentra el soporte para análisis estático de código de diversos lenguajes entre los que se encuentran Kotlin y TypeScript, los dos principales de este proyecto. Esta herramienta me permite encontrar y solucionar errores de programación sin necesidad de ejecutar el código. Algunos ejemplos de errores que detecta pueden ser:

- Código inalcanzable.
- Inconsistencia entre tipo devuelto y tipo especificado en la firma de un método.
- Construcciones redundantes, como `if (true)` o `1 == 1`

Y todo esto ocurre en tiempo real, de modo que a medida que vas escribiendo se van resaltando los errores y puedes corregirlos al momento.

Además, como IntelliJ tiene una integración de Git, puedes hacer *commit* y *push* desde el propio IDE, de modo que si se diese el caso de intentar hacer un *commit* y existiese algún error, aparecería un mensaje indicándolo y recomendando que lo solucionase.

Adicionalmente también use `tslint`, una herramienta exclusiva de TypeScript con reglas específicas y más completas que las presentes en IntelliJ.

5.2.2. Pruebas

Debido a la serie de problemas que tuve a lo largo del proyecto, tal y como explico en la sección 3.4, no me ha sido posible dedicar el tiempo necesario para crear pruebas unitarias o de integración programática, por lo que todas las pruebas han sido manuales.

Para cada historia de usuario se han realizado pruebas unitarias para comprobar que se pueden realizar las tareas indicadas. Adicionalmente, cuando se finalizaban las tareas correspondientes a cada una de las vistas de los departamentos, se realizaban unas cuantas pruebas de integración para asegurar el correcto funcionamiento de todo el conjunto del sistema.

API REST

Para la realización de pruebas para comprobar que el funcionamiento de la API y sus *endpoints* es el correcto he hecho uso de la aplicación Postman, para poder realizar peticiones usando diferentes verbos y especificando cabeceras personalizadas y tokens para la autenticación. La aplicación es muy completa puesto que también te permite especificar el cuerpo de peticiones POST, muestra los resultados de las peticiones, sus cabeceras, el código de estado de la respuesta, etc.

Interfaz de usuario

En cuanto a las pruebas de la interfaz de usuario, para comprobar su correcto funcionamiento hice uso de las herramientas de desarrollador disponibles en el navegador Google Chrome para detectar cualquier tipo de error que se produjese durante su uso normal.

Además de las pruebas que realicé yo personalmente, el supervisor también realizó pruebas de la interfaz para comprobar que se ajustaba a la especificación de la aplicación.

Capítulo 6

Conclusiones

Este trabajo me ha servido mucho para ampliar mis conocimientos de programación de servidores y para empezar a tocar algo de la parte del cliente, cosa que no había hecho en el pasado. La parte del servidor por lo general ha sido la más fácil de desarrollar debido a mi experiencia previa con las tecnologías y lenguajes, a excepción de alguna parte concreta como la de la autenticación, puesto que nunca antes había implementado realmente un sistema de autenticación. En cuanto a la parte del cliente, a pesar de que era algo prácticamente nuevo para mí, no ha supuesto un reto ya que una vez aprendido un lenguaje has aprendido todos, lo único que necesitas es adaptarte a la sintaxis, y la de TypeScript es muy similar a la de Kotlin, así que muy difícil no fue. En lo relativo a los *framework* de CSS, dado que era algo visual y la documentación era muy detallada y el código auto explicativo, fue bastante intuitivo todo.

También cabe mencionar que el tener que adaptarme para solucionar las incidencias que han ido surgiendo a lo largo del proyecto, junto con el hecho de que se trata un proyecto relativamente complejo en cuanto a la profundidad de las tareas que tiene que poder realizar, han supuesto un reto en cuanto al tiempo del que disponía.

Finalmente, he de decir que estoy muy satisfecho con el producto final que ha resultado de estos 3 meses de trabajo.

Bibliografía

[1] Buefy.org. Buefy: lightweight UI components for Vue.js based on Bulma | Buefy. https://buefy.org/, último acceso 05/06/2019.

[2] Bulma.io. Bulma: Free, open source, & modern CSS framework based on Flexbox. https://bulma.io/, último acceso 03/06/2019.

[3] IETF. RFC 8018 - PKCS #5: Password-Based Cryptography Specification Version 2.1. https://tools.ietf.org/html/rfc8018, último acceso 11/07/2019.

[4] Indeed. Salarios para desarrollador junior en España. https://www.indeed.es/salaries/Desarrollador/a-junior-Salaries?period=yearly, último acceso 08/06/2019.

[5] JetBrains. IntelliJ IDEA Ultimate: JetBrains Toolbox subscription. https://www.jetbrains.com/idea/buy/, último acceso 08/06/2019.

[6] Kotlin. Kotlin Programming Language. https://kotlinlang.org/, último acceso 10/07/2019.

[7] Kotlin Blog. Kotlin 1.0 Released: Pragmatic Language for JVM and Android. https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/, último acceso 07/05/2019.

[8] Ktor.io. Ktor - asynchronous Web framework for Kotlin. https://ktor.io/, último acceso 02/06/2019.

[9] Mozilla. HTTP request methods. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods, último acceso 10/07/2019.

[10] Mozilla. HTTP response status codes. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status, último acceso 10/07/2019.

[11] ralfstuckert. Ktor Issue #961 - MPP JS client: Can't resolve 'text-encoding'. https://github.com/ktorio/ktor/issues/961, último acceso 02/07/2019.

[12] Tapac (Andrey Tarashevskiy). Exposed - Kotlin SQL Library. https://github.com/JetBrains/Exposed, último acceso 03/06/2019.

[13] Vue.js. Vue.js. https://vuejs.org/, último acceso 31/05/2019.

[14] Wikipedia. Interfaz fluida. https://es.wikipedia.org/wiki/Interfaz_fluida, último acceso 22/07/2019.

[15] Wikipedia. PBKDF2. https://en.wikipedia.org/wiki/PBKDF2, último acceso 02/05/2019.