Universidad de Cantabria

Facultad
de
Ciencias

APLICACIÓN WEB PARA LA DISTRIBUCIÓN DE INFRAESTRUCTURAS DESTINADAS AL APROVECHAMIENTO DE ENERGÍAS RENOVABLES

(Web application for the distribution of infrastructures for the use of renewable energies)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMATÍCA

Autor: Laureano Ateca González

Director: Patricia López Martínez

Septiembre – 2023

Resumen

La finalidad de este Trabajo de Fin de Grado es desarrollar una aplicación web para la venta de la infraestructura necesaria para el aprovechamiento de las energías renovables, y concienciar a la población de la necesidad de cuidar el medio ambiente.

La aplicación web se divide en front-end y back-end, que no deja de ser el patrón de tres capas separando la capa de presentación, de la de negocio y persistencia.

El front-end (capa de presentación) es un cliente web, esto permite que la aplicación sea accesible desde cualquier equipo a través del uso de un navegador. Independientemente de la potencia computacional del equipo del usuario, al no tener que instalar la lógica de negocio en el dispositivo, como ocurre con las aplicaciones de escritorio, el rendimiento de la aplicación depende solo del servidor que arranca esta lógica y de la calidad del trabajo del desarrollador. Por lo que es más cómodo y accesible para el usuario el uso de este tipo de aplicaciones.

El back-end (capa de negocio y capa de persistencia) es una API REST, la lógica de negocio por lo tanto es reutilizable y accesible desde cualquier lugar. Dado que se maneja información personal de los clientes se ha tenido como uno de los aspectos más relevantes la seguridad, implantando un sistema de autenticación y autorización haciendo uso de tokens.

La capa de persistencia hace uso de una base de datos relacional lo que facilita la organización y estructuración de la información. También permiten el uso de restricciones que aseguran la coherencia y consistencia de los datos, haciendo que no haya duplicaciones y que sean confiables. Permiten hacer transacciones, importantes para los pagos hechos por los clientes en el sistema, de forma que, si ocurre un inconveniente, la integridad de los datos se mantenga.

Se ha utilizado MySQL para la implementación de la base de datos relacional, el lenguaje de programación Java dentro del marco de Spring para la implementación de la API REST (back-end), y se ha utilizado la librería de React como marco para el desarrollo del front-end, haciendo uso del lenguaje de programación JavaScript, del lenguaje de marcado HTML, y el lenguaje de estilo CSS.

Palabras clave: Aplicación web, Spring Boot, ReactJS, MySQL, API REST

Abstract

The purpose of this Final Degree Project is to develop a web application for the sale of the necessary infrastructure for the use of renewable energies, and to raise public awareness of the need to care for the environment.

The web application is divided into front-end and back-end, which is the three-layer pattern separating the presentation layer, the business layer and persistence.

The front-end (presentation layer) is a web client, which allows the application to be accessible from any computer through the use of a browser. Regardless of the computational power of the user's computer, since the business logic does not have to be installed on the device, as is the case with desktop applications, the performance of the application depends only on the server that starts this logic and the quality of the developer's work. So, it is more comfortable and accessible for the user to use this type of applications.

The back-end (business layer and persistence layer) is a REST API, the business logic is therefore reusable and accessible from anywhere. Since personal customer information is handled, security has been one of the most relevant aspects, implementing an authentication and authorization system using tokens.

The persistence layer makes use of a relational database, which facilitates the organization and structuring of the information. They also allow the use of restrictions that ensure the coherence and consistency of the data, so that there are no duplications, and they are reliable. They allow transactions, important for payments made by customers in the system, so that, if an inconvenience occurs, the integrity of the data is maintained.

MySQL has been used for the implementation of the relational database, the Java programming language within the Spring framework for the implementation of the REST API (back-end), and the React library has been used as a framework for the front-end development, making use of the JavaScript programming language, the HTML markup language, and the CSS styling language.

Keywords: Web Application, Spring Boot, ReactJS, MySQL, API REST.

Índice

1. Introducción

En este capítulo se va a exponer el contexto que da a lugar al proyecto, los objetivos de este, y como estará estructurado.

1.1 Contexto

El presente proyecto surge buscando atenuar por un lado el cambio climático producido por el efecto invernadero resultado de la quema de combustibles fósiles, así como, mejorar la situación económica de la población teniendo en mente el contexto socioeconómico en el que se encuentra nuestro país, resultado de la pandemia provocada por el Covid-19 y la guerra de Ucrania.

Actualmente nos encontramos en uno de los ciclos de cambio climático que ha vivido la Tierra desde su existencia. Este proceso tiene lugar debido al efecto invernadero, cuyo catalizador es la quema de combustibles fósiles como son el carbón, el petróleo y el gas natural.

El efecto invernadero provoca que aumente la temperatura de la Tierra (debido a las emisiones de CO2) produciendo un gran impacto en la vida que alberga. Se está produciendo un efecto dominó que nos está afectando como especie progresivamente.

Como consecuencia del cambio climático podemos observar ya cambios en el suelo. De acuerdo con la (AEMA, 2021), la humedad ha ido cambiando en varias regiones, disminuyendo notablemente en la región mediterránea y aumentando en el norte de Europa desde la década de 1950. Se prevén efectos similares en los años venideros debido a los cambios en los patrones de lluvias. Estos descensos en la humedad pueden tener repercusiones drásticas en la producción de alimentos.

Por otra parte, la AEMA nos hace conocedores de que hay 13 Estados miembros de la UE que han declarado estar afectados por la desertización. La erosión que sufre el suelo puede acelerarse por acontecimientos climáticos extremos debido a los aumentos de temperatura; igualmente se pierden zonas de tierra debido al aumento del nivel del mar, zonas que no se pueden cultivar ya que el mar trae contaminantes, la sal entre ellos. Este es uno de los muchos impactos que genera el cambio climático, y tiene lugar con algo tan importante como es la producción de alimentos.

En España hay iniciativas para reducir la emisión de gases de efecto invernadero, pero aún hay dependencia de este tipo fuentes de energía para generar la electricidad necesaria para que la calidad de vida de las personas no se vea afectada. Producimos nuestra electricidad tanto de fuentes de energía no renovables como el carbón, como de energías renovables como la energía eólica e hidráulica (principales fuentes de energía de este tipo en el país).

Una forma de detener la aceleración del efecto invernadero es la de agilizar y promover la independencia energética mediante la explotación de las energías renovables en las viviendas de los ciudadanos. Utilizar fuentes inagotables de energía que no contaminen, como el sol, el viento, la energía cinética de los movimientos del agua…

Por otro lado, la pandemia provocada por el Covid-19 ha contribuido en gran medida a la inflación que percibimos actualmente en recursos de primera necesidad, bienes y servicios.
Como bien sabemos por el banco de España, en concreto por (Prades Illanes & Tello Casas, 2020), la expansión del Covid-19 y la consiguiente decisión de tomar medidas de confinamiento y distanciamiento social para contener la enfermedad, han supuesto una paralización en el conjunto de actividades desarrolladas en el país, y como resultado una caída del PIB sin precedentes en la historia reciente.

Tanto el sector primario como el energético mantuvieron su actividad de forma directa, pero el sector energético por el efecto de arrastre del cese de la actividad de varios sectores tuvo muchas pérdidas ya que estos sectores hacían un uso intensivo de la energía.

La pandemia ha provocado la pérdida de multitud de puestos de trabajo debido a las pérdidas generadas por las restricciones impuestas por las autoridades, así como, la subida en los precios en el sector servicios y en recursos de primera necesidad. Siendo el precio de la luz uno de ellos.

También debemos tener en cuenta que en los últimos meses ha habido una subida en los precios de la electricidad y del gas natural debido a las tensiones que ha provocado la invasión de Ucrania por parte de Rusia, esto ha hecho que tanto España como el resto de los países europeos estén sufriendo una crisis energética.

Rusia es el principal país que suministra energía a Europa ya que tiene la mayor reserva de gas natural de la Tierra, razón por la que Europa tiene una gran dependencia energética. Además, las tensiones con Argelia (principal proveedor de gas natural de España) han aumentado por el conflicto saharaui con Marruecos.

De acuerdo con el departamento de matemática aplicada y estadística de la universidad de San Pablo-CEU en concreto con (García Centeno, Rodríguez Sanchez, Aguirre Arrabal, & Inchausti Tabuenca, 2022), el incremento del precio de la energía ha tenido como consecuencia una subida generalizada de los precios, ya que el precio de los costes de transporte, producción y almacenamiento también sube proporcionalmente. Las empresas no pueden subir tanto el precio de sus productos finales porque no serían accesible para muchos hogares y tendrían muchos menos clientes, por lo que tienen que realizar despidos masivos con el consiguiente incremento del paro.

Con el aumento de la inflación y los despidos masivos resultado de la pandemia y de la guerra de Ucrania, el poder adquisitivo de los hogares españoles se ve profundamente afectado.

Teniendo en cuenta el contexto anterior se propone la realización de un proyecto que puede reducir en cierta medida los efectos de la inflación, en concreto, reducir los costes energéticos en los hogares mediante la venta e instalación de infraestructura que pueda aprovechar las fuentes de energía renovables. Los hogares pasarían a ser energéticamente autosuficientes, ya que con esta infraestructura podrían generar energía eléctrica y calorífica, todo ello siendo respetuosos con el medio ambiente, ya que el aprovechamiento de estas energías no es contaminante.

1.2 Objetivos

Con la realización de este proyecto se busca poner a disposición de los hogares un servicio de venta e instalación de la infraestructura necesaria para poder aprovechar fuentes de energía limpias para el medio ambiente e inagotables, a un coste a medio-largo plazo inferior al que ofrecen los servicios de compañías eléctricas a las cuales hay que pagar periódicamente, y cuyos precios siguen subiendo debido al contexto actual.

En el hogar no se pueden aprovechar todas las fuentes de energía renovables, solo es viable el aprovechamiento de algunas de ellas, por lo que la infraestructura que se oferta en la aplicación web será para el aprovechamiento de la energía solar, eólica, y la generación de biomasa y biogás. La energía hidráulica, mareomotriz, y geotérmica no son aprovechables en los hogares.

De la energía solar se puede sacar la energía solar térmica que transforma la energía del sol para producir calor, este calor se puede aprovechar directamente o se puede transformar en energía mecánica y a su vez en energía eléctrica. La infraestructura necesaria la componen colectores o captadores solares. Por otro lado, podemos aprovechar del sol la energía solar fotovoltaica, esta transforma directamente la radiación solar en electricidad. La infraestructura consta de paneles solares fotovoltaicos, estos tienen células de silicio que transforman la luz y calor del sol en electricidad.

La energía eólica se puede aprovechar mediante el uso de aerogeneradores, estos convierten el movimiento de las palas accionadas por el viento en energía eléctrica. Son generadores eléctricos movidos por turbinas accionadas por el viento.

En los hogares también se puede generar biomasa y biogás, la materia orgánica procedente de animales y plantas (biomasa) es descompuesta por bacterias (descomposición anaeróbica) mediante un biodigestor. Durante este proceso se generan gases como el metano y el dióxido de carbono (biogás). El biogás se puede utilizar para generar electricidad, energía térmica, o incluso como carburante para vehículos adaptados.

Se ofertará la infraestructura, la instalación de esta y la posibilidad de pedir piezas que puedan haberse dañado por su uso continuado incluyendo en su venta la posibilidad de que un técnico realice su instalación.

Desde la aplicación web también se buscará concienciar sobre el estado del medio ambiente, las ventajas del aprovechamiento de las energías renovables para mejorar el estado de este y las ventajas económicas que se obtienen teniendo en cuenta la inflación actual.

Se desarrollará una aplicación web intuitiva para que los clientes puedan usarla cómodamente y que personas con menos experiencia con estas tecnologías tengan una curva de aprendizaje mínima para su uso. Además, al tratarse de una aplicación web el usuario no tiene que realizar instalaciones, lo único que necesita es un navegador web e internet.

1.3 Estructura del proyecto

El presente trabajo presenta una estructura basada en el ciclo de vida del desarrollo software. Sus apartados corresponden a las diferentes fases de este:

- En el capítulo 2 se aborda la fase de requisitos. En esta se determinan las funcionalidades que se requiere que tenga el producto software para que el usuario final considere que está completo y se cubran todas sus necesidades. Puede ser preciso especificar ciertos requisitos no funcionales ya que el usuario puede querer alcanzar unos tiempos de respuesta máximos, usabilidad, seguridad…
- El capítulo 3 expone la fase de diseño, en la cual se realizan modelos que dan una visión más clara y global al desarrollador, permiten idear como resolver las necesidades recogidas en la fase de requisitos, de manera que, la implementación del sistema de información sea más sencilla y limpia.
- En el capítulo 4 se aportan detalles sobre la fase de implementación. Se recogen las herramientas, lenguajes y tecnologías, que se utilizan para crear el producto que requiere el usuario final. En esta fase se desarrolla el software de acuerdo con los modelos que han sido generados previamente, de manera que, el producto incluye las funcionalidades requeridas de manera eficiente.
- En el capítulo 5 se aborda la fase de pruebas en la cual se planifican una serie de pruebas que buscan comprobar el correcto funcionamiento del sistema, para lo cual se especifica un número finito de casos de prueba que sean susceptibles de generar errores inesperados para poder subsanarlos. Se incluyen pruebas unitarias, de integración, de sistema (en caso de haber requisitos no funcionales que atender), y de aceptación.

Los siguientes apartados son propios del trabajo:

- En el capítulo 6 se habla de las conclusiones resultantes de haber realizado el trabajo, si se han alcanzado los objetivos, si hay algún aspecto que mejorar, lo que ha supuesto u ofrecido el trabajo a nivel personal…
- En el capítulo 7 se trata el trabajo que se podría hacer a futuro teniendo como base el que se presenta ahora.
- En el capítulo 8 se encuentra la bibliografía, esta recoge todas las fuentes de información que se han tomado de referencia para realizar, quizá, la parte más técnica del trabajo.
- Los siguientes apartados son anexos con información de interés.

2. Requisitos

En este capítulo se recogerán los requisitos funcionales y no funcionales que ha de cubrir el producto software para satisfacer las necesidades del usuario final. Para ello se detallarán las funcionalidades necesarias haciendo uso de la técnica de casos de uso, y diagramas de clases para complementar y apoyar dicha información.

2.1 Visión general

La aplicación tendrá tres roles, el rol de administrador de la aplicación, el de usuario anónimo y el de usuario identificado (cliente). Cada rol podrá realizar una serie de acciones organizadas en grandes grupos de funcionalidad, cómo podemos observar en la Figura 1.

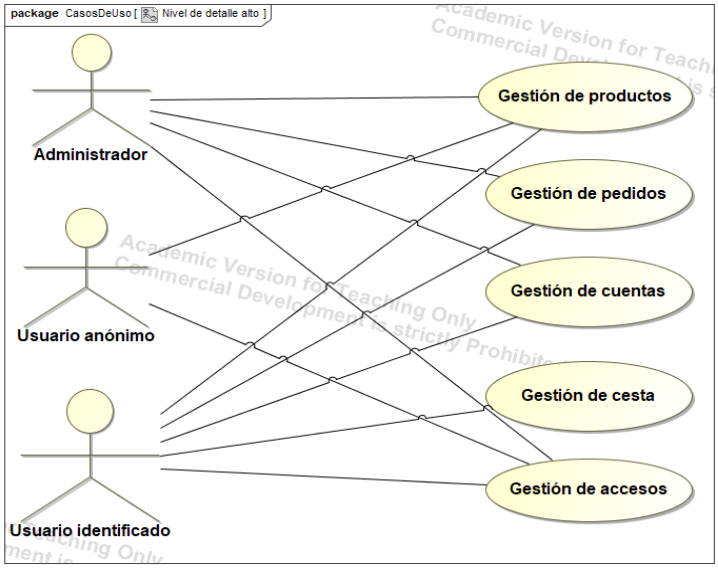

Figura 1: Diagrama de casos de uso de alto

El bloque de Gestión de accesos recoge las funcionalidades relacionadas con verificar que cada usuario es quien dice ser para poder disfrutar de la plataforma de forma segura. Las funcionalidades incluyen el registro del usuario, la posibilidad de restablecer la contraseña en caso de que esta sea olvidada, la identificación del usuario, y una vez que este ha iniciado sesión en el sistema, la posibilidad de cerrar la misma. En la Figura 2 se muestra el diagrama de casos de uso detallado que recoge las funcionalidades expuestas.

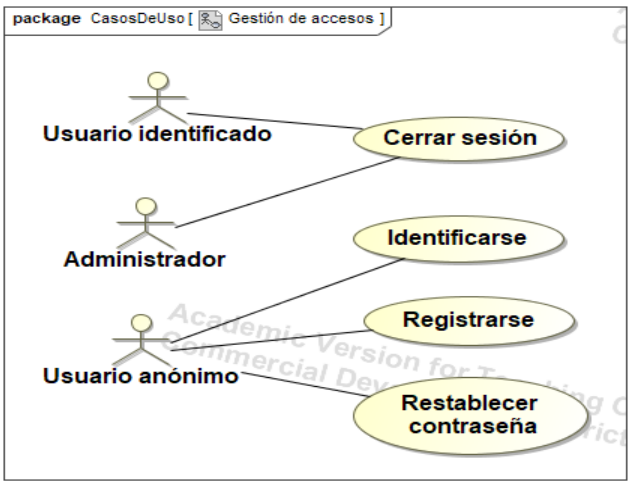

Figura 2: Diagrama de casos de uso de gestión de accesos

El bloque de Gestión de productos agrupa las funciones relacionadas con la adición, modificación, borrado, consulta y listado de productos, así como con mostrar la información de la página de inicio, que da una imagen general del objetivo del negocio. En la Figura 3 se muestra el diagrama de casos de uso detallado que recoge las funcionalidades definidas en este bloque.

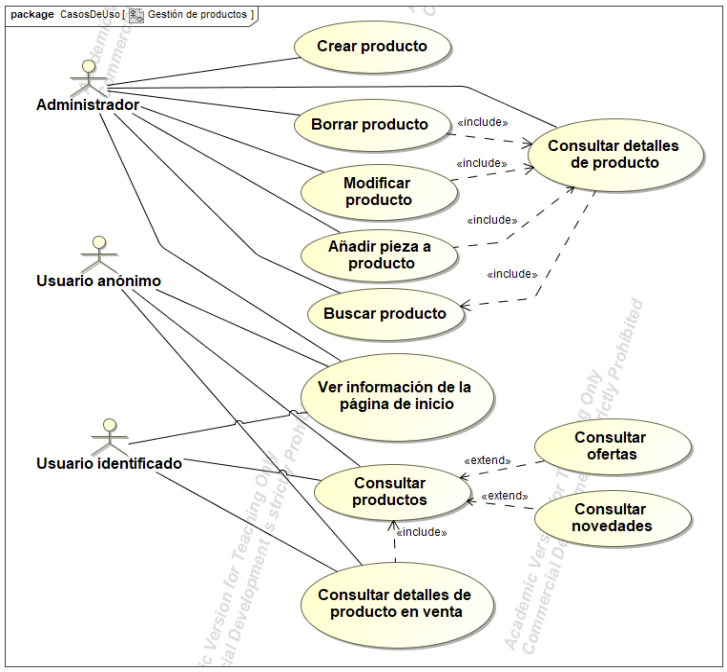

Figura 3: Diagrama de casos de uso de gestión de productos

El bloque de Gestión de cuentas engloba las funcionalidades relacionadas con la visualización y modificación de información relevante para poder dar servicio a los clientes del sistema. La información trata de identificar al cliente por sus datos personales, poder contactar con él, y poder realizar envíos al mismo, entre otros. En la Figura 4 se muestra el diagrama de casos de uso detallado que recoge las funcionalidades concretas de este bloque.

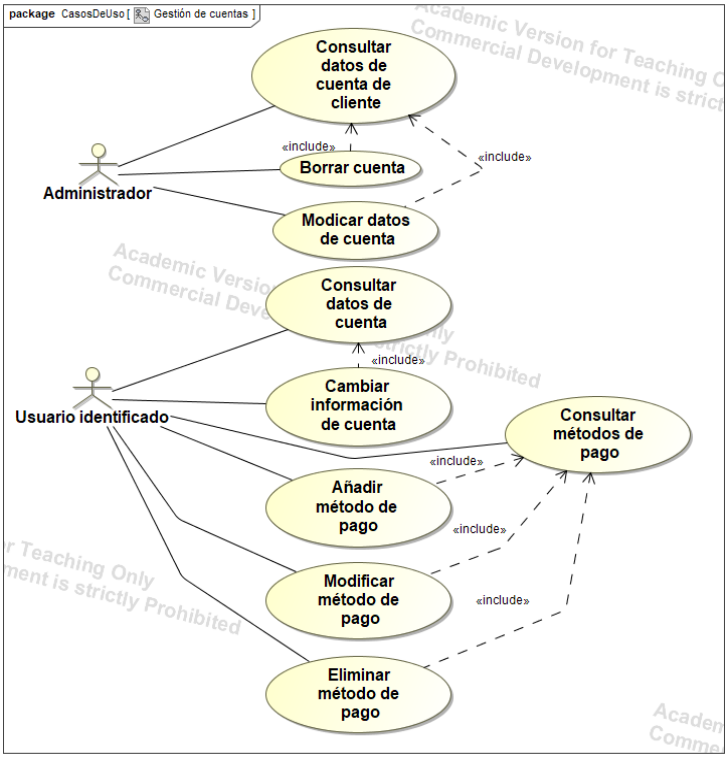

Figura 4: Diagrama de casos de uso de gestión de cuentas

El bloque de Gestión de cesta incluye funcionalidades que permiten al usuario decidir qué productos desea adquirir, para lo cual puede añadir artículos a la cesta, eliminar artículos de la cesta que finalmente no desea, modificar el número de ítems de un artículo, y por último tramitar el pedido para que se haga efectivo. En la Figura 5 se muestra el diagrama de casos de uso detallado que recoge las funcionalidades definidas.

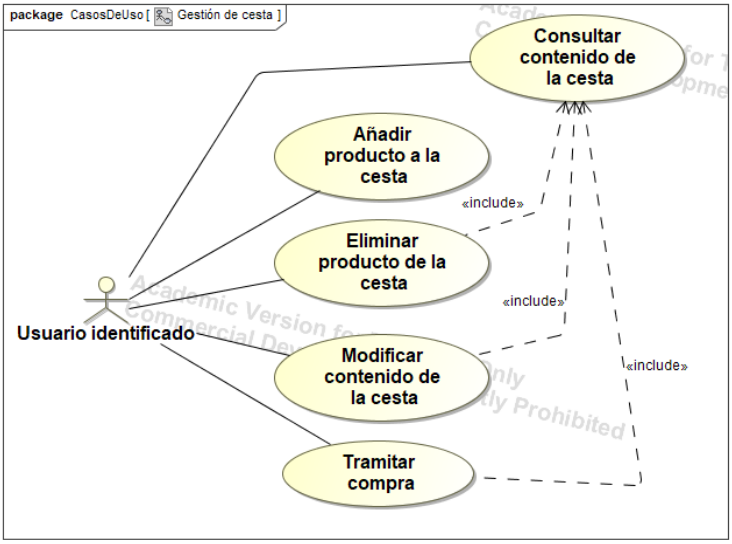

Figura 5: Diagrama de casos de uso de gestión de cesta

Finalmente, el bloque de Gestión de pedidos recoge las funcionalidades que permiten al usuario conocer los pedidos realizados con todos sus detalles, incluyendo el estado en el que se encuentra el pedido con respecto a su entrega. Además, hay funcionalidades que permiten realizar modificaciones en los pedidos en determinadas circunstancias. En la Figura 6 se muestra el diagrama de casos de uso detallado que recoge las funcionalidades definidas en este bloque.

Con la finalidad de comprender mejor el presente sistema de información se muestra en la Figura 7 el modelo de dominio de la aplicación como complemento a los casos de uso. El modelo de dominio proporciona información más detallada sobre cómo se estructura el sistema.

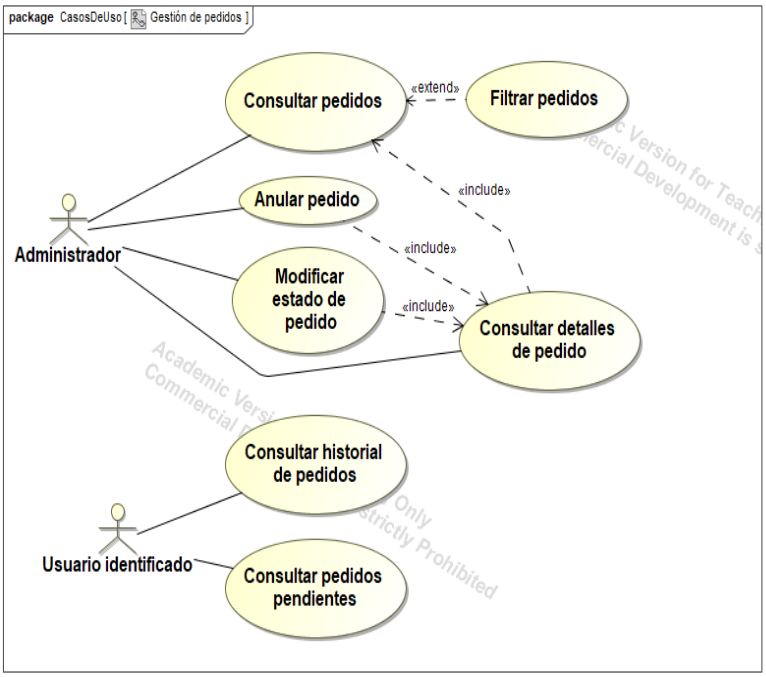

Figura 6: Diagrama de casos de uso de gestión de pedidos

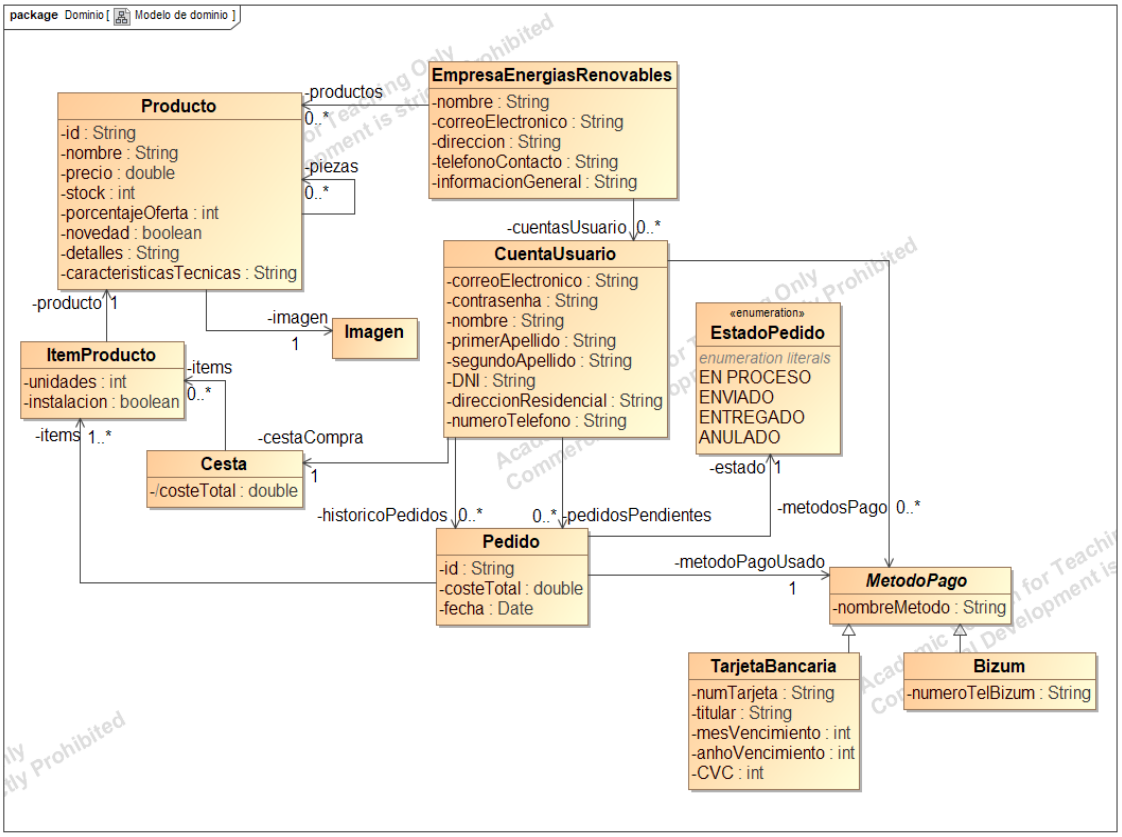

Figura 7: Modelo de dominio

2.2 Especificación detallada de casos de uso

En este apartado se especificarán algunos de los casos de uso más relevantes y/o complejos correspondientes a los cinco grandes grupos de funcionalidad. Las especificaciones de los casos de uso restantes se encuentran recogidos en el Anexo A completando así todos los casos de uso modelados.

2.2.1 Casos de uso – Gestión de accesos

Id + Nombre	Identificador: UseCase-004 Nombre: Restablecer contraseña.
Actor Principal	Usuario anónimo.
Descripción	El usuario registrado, pero no identificado, al haber olvidado la contraseña de su cuenta, decide sustituirla por una que sí pueda recordar siguiendo un proceso para demostrar que este es el caso, y que no intenta usurpar la identidad de otro usuario.
Precondición	El usuario debe encontrarse registrado en el sistema, no encontrarse identificado, recordar qué correo electrónico utilizó en el registro, y tener acceso a la bandeja de entrada del correo electrónico.
Postcondición	El usuario ha cambiado su contraseña satisfactoriamente.
Escenario Principal	1. El usuario selecciona el área donde pone “Cuenta”. 2. El sistema mostrará una nueva página que permite al usuario identificarse, además de dar como opciones restablecer contraseña o registrarse. 3. El usuario selecciona la opción correspondiente a reestablecer la contraseña. 4. El sistema muestra una página en la que se pide al usuario, mediante formulario, el email con el que se ha registrado en el sitio web. 5. El usuario introduce su email en el campo correspondiente y selecciona la

	opción para continuar. 6. El sistema envía un mensaje al email facilitado por el usuario con un código, y muestra una nueva página con un campo para introducir dicho código con el que se verifica que el usuario que restablece la contraseña es quien se espera. 7. El usuario consulta su email personal, y en la página de la aplicación introduce el código en el campo correspondiente seleccionando la opción continuar. 8. El sistema muestra una nueva página con un formulario, el formulario consta de dos campos donde podremos introducir el código recibido por email, y la nueva contraseña. 9. El usuario rellena los campos del formulario con el código y la nueva contraseña y selecciona la opción para guardar los valores. 10. El sistema muestra un mensaje confirmando el cambio de contraseña.
Escenarios alternativos	6.b El email introducido no se corresponde con el de ningún usuario que se encuentra en el sistema. 6.b.1 El sistema indica, desde la misma página que solicita el email, que el email introducido no se corresponde con el de ningún usuario. 7.c El código introducido para restablecer la contraseña no es correcto. 7.c.1 El sistema indica mediante un mensaje que el código no es correcto. 7.c.2 Se devuelve al usuario a la página donde se solicitaba introducir el email del que se quiere restablecer la contraseña.

2.2.2 Casos de uso – Gestión de productos

Id + Nombre	Identificador: UseCase-005 Nombre: Crear producto.
Actor Principal	Administrador.
Descripción	El administrador crea, introduciendo la información necesaria, un producto que será accesible por usuarios anónimos e identificados para su consulta y compra.
Postcondición	Se ha creado un producto del cual se ha generado un identificador y que puede ser consultado junto al resto de productos previamente creados.
Escenario Principal	1. El administrador desde la página inicial selecciona el área que engloba las operaciones de gestión de productos en venta. 2. El sistema muestra una nueva página con varias opciones, la de crear un nuevo producto, o buscar producto entre los existentes. 3. El administrador selecciona la opción correspondiente a crear un nuevo producto. 4. El sistema muestra una nueva página con un formulario para introducir la información propia del producto (el nombre del producto, su precio base, un porcentaje de descuento en caso de haber oferta, si se trata o no de una novedad, si el producto se encuentra activo, los detalles del producto, características técnicas, el stock disponible en este momento, el precio de la instalación, y la URL de una imagen). 5. El administrador rellena los datos del formulario y selecciona la opción que permite crear definitivamente el producto. 6. El sistema muestra un mensaje que confirma la creación del producto mostrando además su identificador único.
Escenarios Alternativos	6.b El nombre del producto introducido coincide con el de otro producto existente. 6.b.1 El sistema muestra un mensaje que indica que no puede haber dos productos que utilicen el mismo nombre. 6.b.2 Se impide la creación del producto y se devuelve al administrador a la página con el formulario para crear el producto.

Id + Nombre	Identificador: UseCase-015 Nombre: Consultar detalles de producto en venta.
Actor Principal	Usuario anónimo, usuario identificado.

Descripción	El usuario podrá visualizar la información detallada correspondiente a un producto previamente listado.
Precondición	El sistema debe de mostrar algún producto para que el usuario pueda seleccionarlo y así poder ver sus detalles.
Postcondición	El usuario puede visualizar la información del producto seleccionado con mayor detalle.
Escenario Principal	1. Incluye “Consultar productos”. 2. El usuario selecciona en cualquiera de los productos en venta que se encuentran listados en la página correspondiente. 3. El sistema devolverá al usuario una página con información detallada del producto (el nombre, la imagen, precio base, el porcentaje de descuento, el precio final aplicando la oferta, el nombre de las piezas que conforman el producto, detalles, características técnicas, stock, y el precio de instalación), así como la posibilidad de seleccionar el número de unidades que se desean, y la posibilidad de que los técnicos se encarguen de la instalación seleccionando la opción correspondiente. El sistema muestra una opción para añadir el producto a la cesta, si el usuario que selecciona esta opción en un usuario anónimo se le redirigirá a la página de identificación.

2.2.3 Casos de uso – Gestión de cuentas

Id + Nombre	Identificador: UseCase-019 Nombre: Consultar datos de cuenta.
Actor Principal	Usuario identificado.
Descripción	El usuario visualiza los datos personales asociados a su cuenta.
Postcondición	Se muestran los datos de perfil del usuario identificado.
Escenario Principal	1. El usuario selecciona el área de menú del cliente. 2. El sistema muestra en el desplegable varias opciones entre ellas la cuenta. 3. El usuario selecciona en el desplegable la opción que se corresponde con la cuenta de usuario. 4. El sistema muestra todos los datos personales del usuario, el nombre, los apellidos, DNI, correo electrónico, dirección residencial, y número de teléfono. Además, muestra una opción para editar algunos de estos datos.

2.2.4 Casos de uso – Gestión de cesta

Id + Nombre	Identificador: UseCase-025 Nombre: Añadir producto a la cesta.
Actor Principal	Usuario identificado.
Descripción	El usuario tras ver los detalles de un producto decide añadir este a la cesta de la compra con la posibilidad de adquirir finalmente el producto.
Postcondición	El producto que se estaba visualizando se ha añadido a la cesta de la compra, y en esta aparecen los valores de los parámetros que se habían especificado previamente, como el número de unidades deseadas del artículo, y si se desea que un técnico realice la instalación en la dirección de destino.
Escenario Principal	1. Incluye “Consultar detalles de producto en venta”. 2. El usuario selecciona si desea o no que además de la venta del producto se le haga la instalación mediante la selección de una opción e indica el número de unidades que requiere, encontrando como límite el stock disponible para el producto 3. El usuario selecciona la opción de añadir a la cesta. 4. El sistema añade el producto a la cesta.

	5. El sistema confirma que el producto se ha añadido a la cesta mediante un mensaje.

Id + Nombre	Identificador: UseCase-026 Nombre: Consultar contenido de la cesta.
Actor Principal	Usuario identificado.
Descripción	El usuario visualiza los artículos que tiene intención de comprar y que previamente han sido añadidos a la cesta para tal fin.
Postcondición	Se muestran todos los artículos añadidos a la cesta junto con parte de sus datos y el coste conjunto del contenido de la cesta. En caso de no haberse añadido artículos a la cesta se mostrará un mensaje informativo.
Escenario Principal	1. El usuario selecciona el área de la aplicación web que se corresponde con el menú del cliente. 2. El sistema muestra las opciones del desplegable. 3. El usuario selecciona la opción que se corresponde con la cesta. 4. El sistema muestra todos los productos que han sido añadidos a la cesta junto con el coste conjunto de los mismos, la información individual de cada producto (precio base, porcentaje de oferta, precio final aplicando la oferta, nombre, e imagen), las opciones para modificar si se requiere instalación para cada producto por separado, el número de productos requeridos, la opción para eliminar cada producto de la cesta, así como el botón para tramitar el pedido. En caso de no haber ningún producto en la cesta, se mostrará un mensaje informativo en lugar de la información previa.

Id + Nombre	Identificador: UseCase-029 Nombre: Tramitar compra.
Actor Principal	Usuario identificado.
Descripción	El usuario está conforme con el contenido de la cesta y sigue con el proceso para hacer efectiva la compra y recibir el producto.
Precondición	Ha de haber algún artículo en la cesta.
Postcondición	La compra se hace efectiva generando un pedido que tendrá el contenido de la cesta, el cual el usuario puede consultar accediendo al área de pedidos. El stock de los artículos que el usuario solicita queda actualizado.
Escenario Principal	1. Incluye “Consultar contenido de la cesta”. 2. El usuario selecciona la opción que permite tramitar la compra con el contenido de la cesta. 3. El sistema genera un pedido con un estado inicial de “pendiente de pago” y muestra la siguiente información para cada artículo que estaba disponible en la cesta (nombre, número de unidades, instalación, coste individual, coste conjunto), el coste de la cesta más el envío, la dirección a la que hay que enviar el pedido y como opciones de pago un desplegable con los métodos de pago almacenados previamente por el usuario. Se ofrece una opción que al seleccionarla redirige al usuario a la página que le permite añadir un nuevo método de pago a los ya almacenados con el fin de poder seleccionarlo como forma de pago, al finalizar se redirige al usuario a la página para continuar con el pedido. 4. El usuario selecciona uno de los métodos de pago. 5. El usuario selecciona la opción para confirmar pago. 6. El sistema realiza la petición correspondiente al sistema de pago elegido, cambiando el estado del pedido a “en preparación” (ya que el pago ha sido

	exitoso), y actualizándose el stock. Se muestra al usuario una página indicando que el pedido se ha realizado con éxito.
Escenarios alternativos	6.b El sistema realiza la petición correspondiente al sistema de pago elegido y resulta fallida, por lo cual no se ha podido realizar correctamente el pago. 6.b.1 El sistema muestra un mensaje al usuario indicando que las razones por las que no ha podido realizarse el pago pueden ser: que el servidor encargado de hacer efectiva la acción no funcione momentáneamente, o bien que los datos del pago son erróneos. En cualquier caso, se insta al usuario a verificar sus datos e intentar de nuevo el pago. 6.b.2 El sistema devuelve la página anterior, con la información del pedido, y la posibilidad de escoger un método de pago para efectuar el pago de este.

2.2.5 Casos de uso – Gestión de pedidos

Id + Nombre	Identificador: UseCase-030 Nombre: Consultar pedidos.
Actor Principal	Administrador.
Descripción	El administrador accede a la información de los pedidos de todos los clientes.
Postcondición	El sistema ha de mostrar todos los pedidos, en caso de no se haya realizado aún ningún pedido se mostrará un mensaje en la página informando de este hecho.
Escenario Principal	1. El administrador selecciona la opción que se corresponde con la consulta de pedidos. 2. El sistema muestra una nueva página en la que se listan todos los pedidos en orden cronológico, de más actual a más antiguo. Por cada pedido se muestra el identificador, los ítems pedidos para cada artículo, los nombres de los artículos, la fecha, el email y nombre del comprador. Además, es posible ver en más detalle cada pedido seleccionando la opción correspondiente. En caso de que no haya ningún pedido se indicará mediante un mensaje. Igualmente se podrán aplicar varios filtros según los cuales se mostrarán solo pedidos que cumplan con los criterios correspondientes a los mismos.

Id + Nombre	Identificador: UseCase-032 Nombre: Consultar detalles de pedido.
Actor Principal	Administrador.
Descripción	El administrador puede observar todos los pedidos y selecciona uno de ellos para ver información más detallada.
Postcondición	El sistema ha de mostrar toda la información propia de un pedido de forma detallada al seleccionar uno de los pedidos listados.

Escenario Principal	1. Incluye “Consultar pedidos”. 2. El administrador selecciona la opción correspondiente para consultar detalles de uno de los pedidos, esta se encuentra en cada pedido que está previamente listado. 3. El sistema muestra una nueva página con los detalles del pedido; los datos que se muestran son: el identificador, los nombres de los artículos, número de ítems de cada artículo, y si se ha requerido instalación para cada artículo del pedido, el coste de cada artículo y el total del pedido, la dirección a la que se ha de enviar el pedido, la fecha en la que se ha realizado el pedido, el método de pago usado, si se ha realizado el mismo (Bizum o tarjeta bancaria), los datos de ese método de pago (número de móvil para Bizum y número de tarjeta, nombre del titular, y fecha de vencimiento para la tarjeta bancaria), el estado del pedido (pendiente de pago, en preparación, enviado, entregado, o anulado), y la información del comprador, su id, nombre, apellidos, email, DNI, y número de teléfono. Además, se muestran dos opciones la de anular el pedido si aún se encuentra en estado de pendiente de pago o en preparación, y la de modificar el estado a cualquiera de las variantes antes mencionadas menos a pendiente de pago.

Id + Nombre	Identificador: UseCase-033 Nombre: Anular pedido.
Actor Principal	Administrador.
Descripción	El administrador accede a la información detallada de un pedido, desde ahí puede anular el pedido en caso de que el estado de este indique que está pendiente de pago o en preparación.
Precondición	El pedido debe tener un estado de pendiente de pago o de en preparación.
Postcondición	El pedido aún se mostrará en el listado de pedidos, pero se indicará que ha sido anulado.
Escenario Principal	1. Incluye “Consultar detalles de pedido”. 2. El administrador selecciona la opción para anular el pedido. 3. El sistema anula el pedido, y si se ha realizado el pago de este, se hace la devolución del dinero. 4. El sistema muestra la página donde se encuentran todos los pedidos listados. Entre los pedidos se encontrará el pedido recién anulado ya que se desea tener constancia de todos los pedidos, pero se reconocerá como anulado por su estado.
Escenarios alternativos	3.b El sistema anula el pedido, y si se ha realizado el pago de este, se hace la devolución del dinero. La devolución resulta fallida, por lo cual no se ha podido reembolsar el dinero a su propietario. 3.b.1 El sistema revierte las acciones previas dejando el estado como se encontraba antes de su cambio a anulado, y muestra un mensaje al administrador indicando que la razón por la que no ha podido realizarse el reembolso es que el servidor encargado de hacer efectiva la acción no funciona momentáneamente. También se insta al administrador a intentar de nuevo la devolución del coste del pedido. 3.b.2 El sistema permanece en la página con los detalles del pedido donde se puede intentar de nuevo su anulación.

3. Diseño

En este capítulo se explicará en detalle la solución propuesta, recogida en diferentes diseños, para cubrir las necesidades de los stakeholders del sistema, plasmadas en el capítulo de requisitos.

3.1 Visión general

En este apartado se dará una visión general de la arquitectura del sistema, que está estructurada en dos partes diferenciadas: el front-end y el back-end.

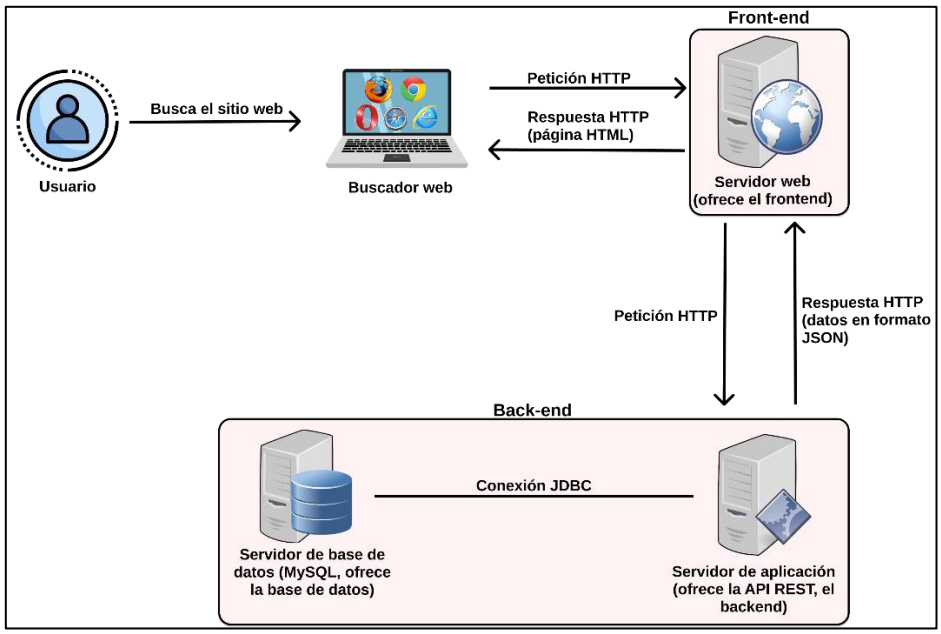

Figura 8: Diseño arquitectónico general

El front-end (capa de presentación) se encarga de mostrar la información de manera visual y facilitar la interacción del usuario con la aplicación. Esto incluye la presentación de textos, imágenes, botones, formularios y otros elementos visuales que permiten a los usuarios navegar por la aplicación, ingresar datos, realizar acciones y recibir retroalimentación visual sobre lo que están haciendo.

El back-end de una aplicación se refiere a la parte de la aplicación que se encarga de gestionar los datos, procesar solicitudes y realizar acciones en función de las interacciones del usuario en el front-end. En otras palabras, el back-end se ocupa de todas las operaciones y lógica que son esenciales para el funcionamiento de la aplicación.

3.2 Diseño arquitectónico

3.2.1 Diseño de la API REST

En la Tabla 1 se muestra el diseño de la API REST, definido en base a los recursos que ofrece, la URI que corresponde a cada recurso, los métodos HTTP necesarios para cubrir la funcionalidad requerida para el sistema de información, y los códigos de respuesta esperados para cada invocación; tanto para los escenarios de éxito como para los diferentes escenarios alternativos.

Tabla 1: Diseño de la API REST

Recurso	URI	Interfaz Uniforme	Códigos de respuesta	
			Éxito	Error
Productos	/productos	GET Query Params: -tipoFiltro: ofertas/novedades	200	
		POST	201	409
Producto	/productos/{id}	GET	200	404
		PUT	200	404
		DELETE	200	404
Piezas de producto	/productos/{id}/piezas	POST	201	404 409
Clientes	/clientes	POST	201	409
Cliente	/clientes/{email}	GET	200	404
		PUT	200	404
		DELETE	200	404
Métodos de pago del cliente	/clientes/{email}/metodosPago	GET	200	404
		POST	201	404 409
Método de pago del cliente	/clientes/{email}/metodosPago/{id}	GET	200	404
		PUT	200	404 409
		DELETE	200	404
Cesta	/clientes/{email}/cesta	GET	200	404
ItemProducto de la cesta	/clientes/{email}/cesta/{idProducto}	GET	200	404
		PUT	200	404
			201	404
		DELETE	200	404
Pedidos de un cliente	/clientes/{email}/pedidos	GET Query Params: -pendientes: true || false	200	404
		POST	201	404
Pedido de un cliente	/clientes/{email}/pedidos/{id}	GET	200	404
Método de pago del pedido	/clientes/{email}/pedidos/{idPedido}/metodoPago/{idMetodo}	PUT	200	404
Petición de cambio de contraseña	/clientes/{email}/codigoRecuperacionContrasenha	POST	201	404
Constraseña del usuario	/clientes/{email}/contrasenha	PUT	200	409
Identificación	/token	POST	201	

Pedidos	/pedidos	GET Query Params: -fechaIni -fechaFin -emailCliente -pendientes: true || false	200	
Pedido	/pedidos/{id}	GET	200	404
Estado del Pedido	/pedidos/{id}/estado	GET	200	404
		PUT	200	404 400
ItemsProducto de pedido	/pedidos/{id}/items	GET	200	404
ItemProducto del pedido	/pedidos/{idPedido}/items/{idProducto}	GET	200	404
Método de Pago del pedido (administrador)	/pedidos/{id}/metodoPago	GET	200	404
Cliente que ha adquirido el pedido	/pedidos/{id}/comprador	GET	200	404

3.2.2 Diseño arquitectónico - Back-end

Como se observa en la Figura 9, para el back-end, esto es, para la implementación de la API REST, se aplica una arquitectura de capas. Específicamente se trata de una arquitectura estructurada en dos capas:

- La capa de repositorio: se encarga del mapeado objeto-relacional de manera que las entidades en Java se correspondan con tablas en base de datos, y los atributos de estas entidades se correspondan con los campos de las tablas, usando los tipos correspondientes en bases de datos. Igualmente hace la conexión con la base de datos. Los repositorios permiten añadir, borrar, consultar, y modificar fácilmente la información en base de datos.
- La capa de controlador: es la que recibe las peticiones HTTP en la API REST. Cada petición consta de una ruta y un método HTTP, entre otros, y esto se corresponde con una operación en la capa de controlador. Se puede extraer información de la URL y/o del cuerpo de la petición HTTP, con esta información y con la necesaria procedente de la base de datos se realizan una serie de operaciones y se elaboran respuestas para los clientes de la API REST. La información que va destinada, o es procedente de la base de datos es accedida a través de la capa de repositorio.

Cómo podemos ver en la Figura 10 y previamente hemos comentado, las diferentes operaciones proporcionadas por la capa de controladores se corresponden con peticiones de los clientes de la API REST. Estas operaciones están agrupadas en grandes grupos funcionales que tratan la misma temática dentro de la aplicación web. Cada grupo funcional constituye un controlador. Un ejemplo de esta estructuración la encontramos en la interfaz IProductoController, en la que podemos observar operaciones que se focalizan principalmente en la información de los productos, en la obtención, eliminación, y modificación de estos.

Por otro lado, en la capa de repositorio nos encontramos con las diferentes interfaces de los repositorios que necesitamos para el correcto funcionamiento de la aplicación web.

En estas interfaces observamos operaciones CRUD, para obtener, añadir, borrar y modificar información en base de datos, aunque se incluyen también operaciones más elaboradas.

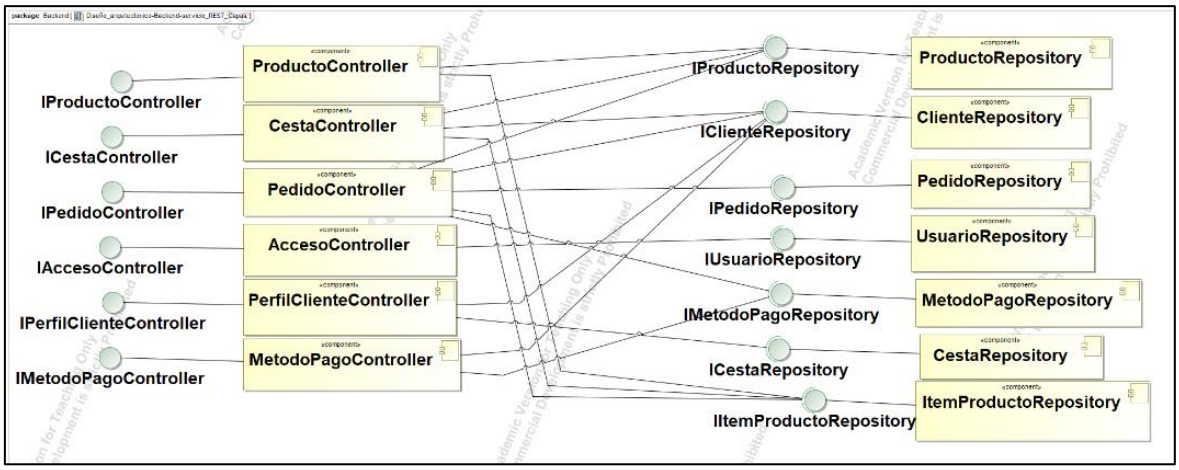

Figura 9: Diseño arquitectónico – Back-end – Capas

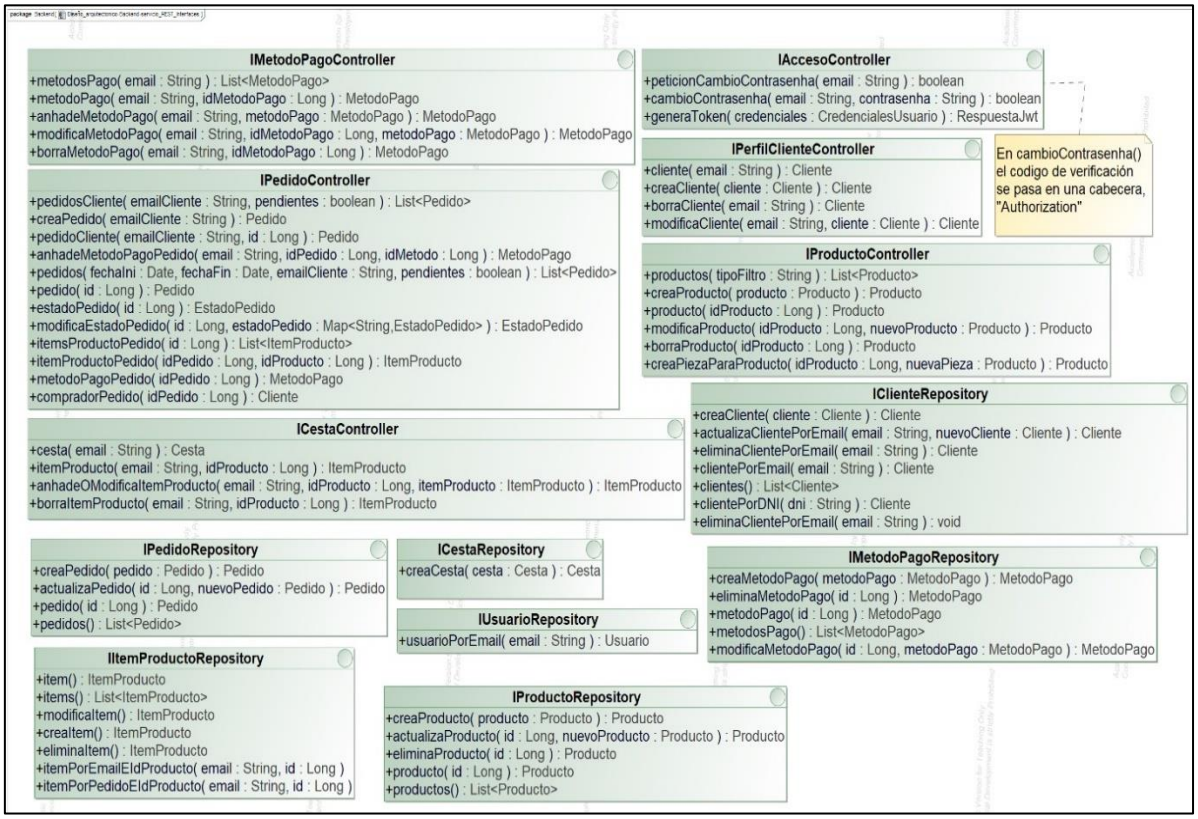

Figura 10: Diseño arquitectónico - Back-end - Interfaces

Muchas de las operaciones hacen uso de la clave primaria de la entidad para localizar la información requerida en base de datos, pero otras veces se puede acceder a esta información a través de otras operaciones usando valores únicos que no sean su clave.

Un ejemplo puede ser IClienteRepository, interfaz que contiene operaciones que utilizan valores únicos que no son la clave primaria (el email, o el dni) para acceder a la información requerida y operar sobre la misma.

Se utiliza el email del cliente para localizar su información ya que, para el usuario, desde la interfaz, es más cómodo y natural acceder a la información de perfil facilitando un email que un identificador numérico que no tiene ninguna lógica subyacente.

Los identificadores autogenerados son útiles a nivel de base de datos porque son muy eficientes, pero las claves naturales permiten al usuario un uso más intuitivo de la aplicación.

3.2.3 Diseño arquitectónico – Front-end

En este capítulo se especificará como está estructurado el front-end, sus diferentes capas y la razón de esta distribución.

De la Figura 11 a la 16, se muestra la estructura en capas del front-end. En este diseño arquitectónico, cada figura representa un grupo funcional. Se pueden diferenciar dos capas en todas las figuras:

- Los componentes React, en este caso son los componentes de más alto nivel, se corresponden con cada una de las páginas de la presente aplicación web. Algunos de estos componentes no requieren de la realización de peticiones HTTP a la API REST, pero en la mayoría de los casos sí es necesario. Por esta razón se ha creado una capa intermedia que modulariza las operaciones que realizan las peticiones a la API REST.
- La capa de servicio está constituida por diferentes módulos que agrupan operaciones con peticiones a la API REST relacionadas entre sí. Los componentes React utilizan los componentes Service para realizar las peticiones, de forma que cualquier cambio en ellas se encuentra localizado, no hay redundancia en el código, y es más legible. Con las peticiones se realizan las tareas oportunas y se muestra en la página correspondiente información actual y útil para el usuario.

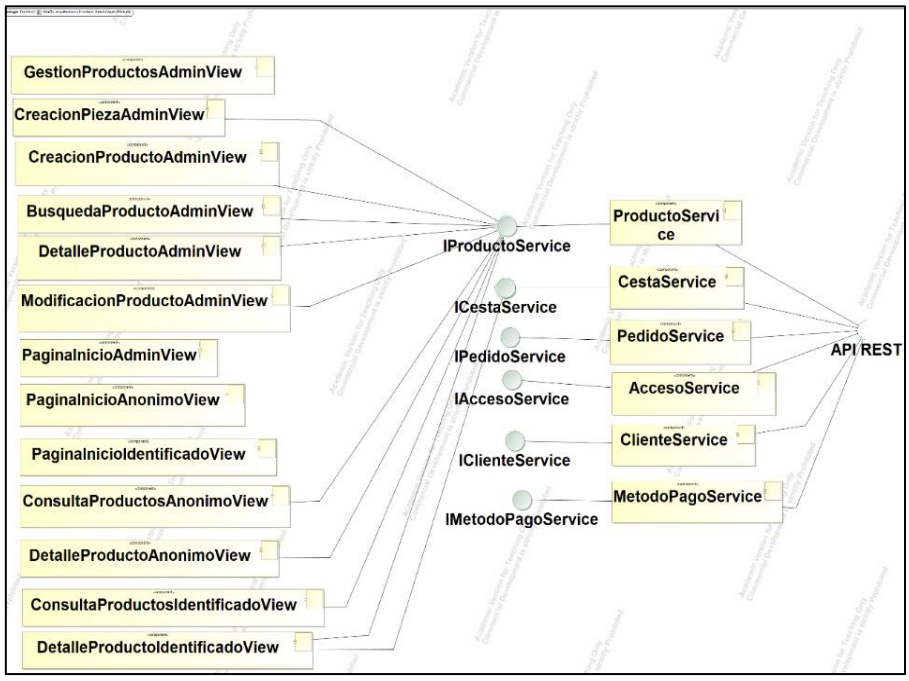

Figura 11: Diagrama arquitectónico - Front-end – Capas – Producto

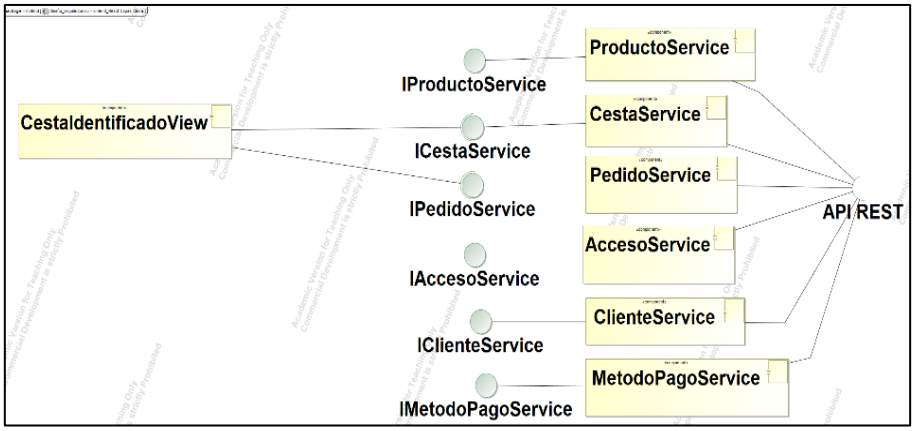

Figura 12: Diagrama arquitectónico - Front-end – Capas – Cesta

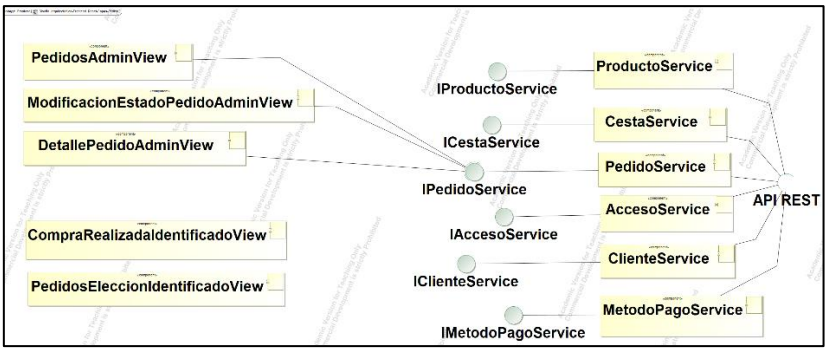

Figura 13: Diagrama arquitectónico - Front-end – Capas – Pedido

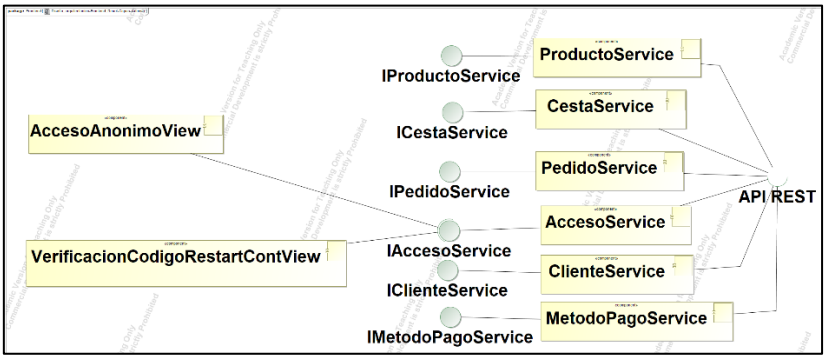

Figura 14: Diagrama arquitectónico - Front-end – Capas – Acceso

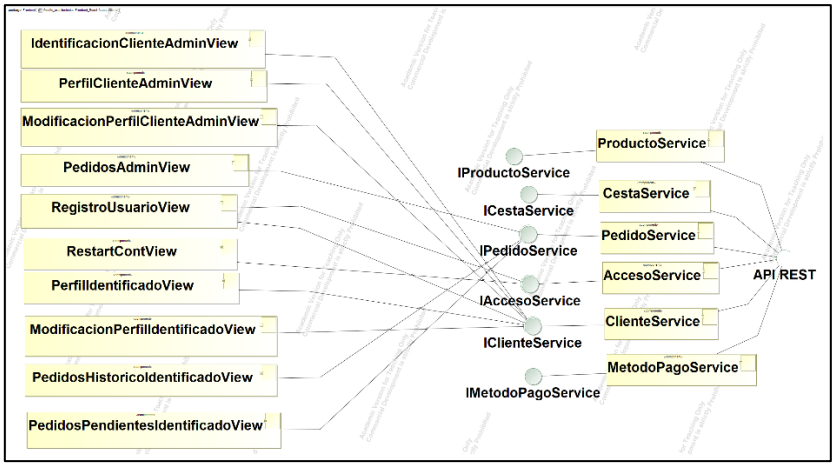

Figura 15: Diagrama arquitectónico - Front-end – Capas – Cliente

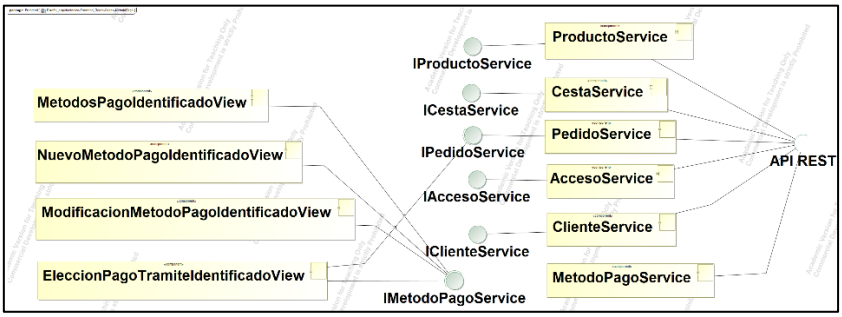

Figura 16: Diagrama arquitectónico - Front-end – Capas – Método de Pago

En la Figura 17 podemos observar las diferentes interfaces proporcionadas por la capa de servicio. En estas interfaces se encuentran las operaciones agrupadas por funcionalidad. Cada operación se corresponde con un método HTTP sobre una determinada ruta. A estas operaciones se les pasa como parámetro la información que se encontrará en la URL y en el

cuerpo de la petición HTTP, y que es necesaria para que se realicen los cambios necesarios en el back-end. El back-end proporcionará una respuesta que será procesada en el front-end y transformada en información útil para el usuario.

Figura 17: Diseño arquitectónico - Front-end – Interfaces

3.3 Diseño detallado

En este apartado se expondrá el diseño detallado de la aplicación web. Con el fin de evitar información redundante con respecto al modelo de dominio del capítulo de requisitos, se dará especial atención a los cambios más relevantes obviando lo ya expuesto.

La Figura 18 nos muestra los diferentes módulos de información y las relaciones entre los mismos que constituyen el dominio del sistema.

En la aplicación web se pone a disposición de los clientes una gran variedad de productos, los cuales, a su vez, están compuestos de piezas. Estas piezas son igualmente productos disponibles para su compra. Todo producto tiene una imagen que hace que el cliente tenga una idea más clara de lo que va a adquirir. Esta imagen se puede mostrar desde el front-end, independientemente de donde se encuentre alojada, a través de una URL. Por tanto, no es necesario convertir la imagen en una cadena de bytes para almacenarla y después devolverla a su estado original para mostrarla. La información que se añade a la base de datos sobre la imagen será una URL, y para recuperarla se obtiene la información junto con la del resto del producto.

Se puede observar que hay una clase Usuario y otra Cliente. Usuario contiene la información que se desea almacenar para el usuario de la aplicación web que tiene el rol de administrador, Cliente contiene la información propia de una persona que está identificada en la aplicación y cuyos objetivos están relacionados con la adquisición de productos, su rol es el de cliente. Como

los roles son excluyentes y no deseamos que un usuario pueda ser a la vez administrador y cliente desde la misma cuenta, asociamos cada uno de los roles a su respectiva clase.

Cada cliente posee una cesta, esta puede contener cero o un número indefinido de ítems, cada ItemProducto se corresponde con un único producto, y para ese producto se puede desear una o más unidades y la instalación de este. Cuando el cliente ha hecho los cambios que ha considerado oportunos sobre la cesta y se decide a confirmar la compra se crea un pedido. Este pedido pasa a añadirse a los pedidos pendientes y al histórico de pedidos. Igualmente, el pedido añade a la información de este quién ha sido su comprador por si el administrador quisiera conocer dicha información al observar todos los pedidos.

Inicialmente los pedidos tienen un estado de “pendiente de pago”, pasando al estado de “en preparación” cuando el pago se ha completado. El resto de los estados será el administrador quien los cambie cuando sea oportuno. Cuando el administrador cambia el estado de un pedido a cualquiera de los otros posibles, se elimina el pedido de los pedidos pendientes del cliente. Los ítems que se encontraban en la cesta de la compra ahora forman parte del pedido (estado de “pendiente de pago”), y al seguir con el proceso de compra se ha de seleccionar un método de pago de los ya almacenados por el cliente realizándose el pago (el estado del pedido pasa a “en preparación”. Una vez seleccionado el método de pago y verificada la compra se le da confirmación al cliente.

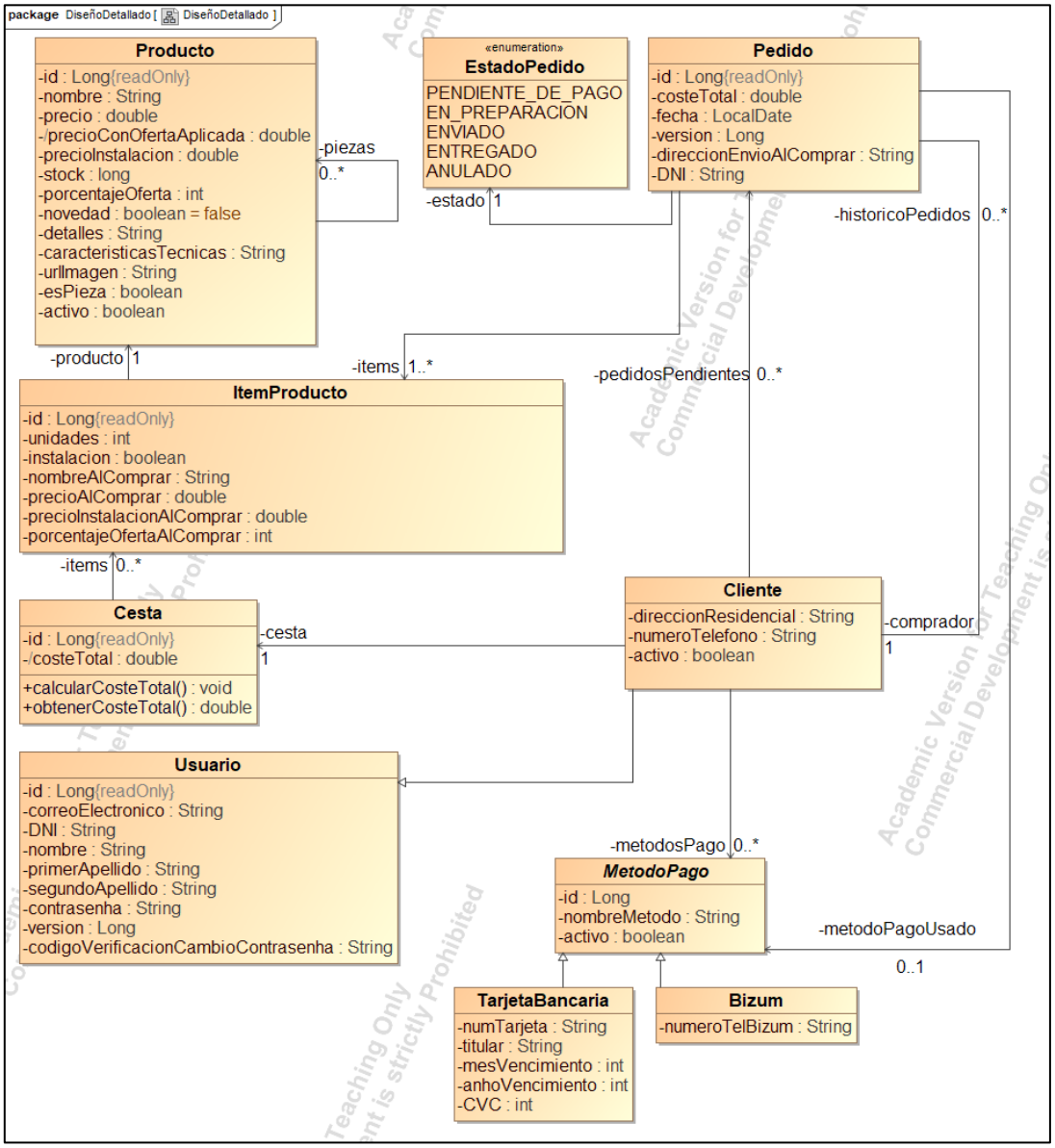

Figura 18: Diseño detallado

4. Implementación

En este capítulo se abordará el uso de las tecnologías específicas utilizadas para generar la aplicación web, así como, se detallarán los aspectos de mayor relevancia encontrados durante el desarrollo de esta.

4.1 Implementación - Back-end

En este apartado se explicarán los aspectos más relevantes encontrados durante el desarrollo de la API REST, incidiendo en el uso de las tecnologías escogidas y sus beneficios.

4.1.1 Tecnologías escogidas

Para implementar la API REST se ha escogido Spring como framework. Spring introduce un nivel de abstracción superior con respecto a las tecnologías utilizadas durante la carrera para desarrollar aplicaciones empresariales y servicios.

Spring Boot es un proyecto dentro del ecosistema de Spring Framework, que facilita la configuración y la creación de aplicaciones Java basadas en Spring.

Destaca que integra Tomcat, Jetty, o Undertow, lo que permite arrancar el servidor con la aplicación sin necesidad de exportar archivos WAR y desplegarlos en un servidor externo; permite crear el proyecto con dependencias fácilmente; y además no es necesario hacer configuraciones con ficheros XML.

Por otro lado, Spring Security es un módulo de Spring que se ha utilizado en este proyecto y que facilita conseguir una API segura, ya que contiene sistemas para identificar a los usuarios que la utilizan (autenticación), y que de acuerdo con una serie de restricciones introducidas en la configuración de estos se puede controlar el acceso a los recursos que ofrece la API (autorización).

Esto se complementa con el sistema de tokens JWT para que el usuario no tenga que introducir sus credenciales en cada petición.

Se ha utilizado el módulo de Spring Data para facilitar el acceso a los datos de base de datos de una forma más sencilla mediante el uso de interfaces predefinidas. No hay que hacer la implementación de las operaciones básicas de las DAO, y si se requiere introducir operaciones diferentes a las más básicas, Spring Data ofrece facilidades.

Se trata de un framework muy completo, que facilita el desarrollo, que es de código abierto, que tiene una gran comunidad, y que permite el uso del lenguaje de programación Java.

Por todas estas razones ofrece un marco de trabajo interesante.

4.1.2 Estructura del proyecto

El back-end de la aplicación web tiene un amplio conjunto de paquetes de forma que sea intuitivo encontrar los ficheros requeridos en cada momento por el nombre de estos.

Una correcta documentación y estructuración aumenta en gran medida la legibilidad del trabajo, y por lo tanto el rendimiento del individuo y del equipo de trabajo.

En la Figura 19 se puede observar la estructura del proyecto Maven que constituye la API REST con la lógica de negocio de la aplicación web, dicha estructura está compuesta por:

- El paquete terminado en “infraestructuraenergiasrenovables” que contiene la clase principal, esta al ejecutarse crea todos los componentes definidos en el código, arranca el servidor embebido y despliega la API REST.
- El paquete terminado en “controllers” que contiene los diferentes controladores que componen la API REST. Las peticiones realizadas por un cliente externo son redireccionadas al método del controlador correspondiente para que, realizando las tareas que se requieran en el mismo, se genere una respuesta con la información esperada por el cliente. Se han creado controladores que agrupan las posibles peticiones por funcionalidad.
- El paquete terminado en “deserializers” incluye un deserializer personalizado que transforma el contenido del cuerpo de determinadas peticiones que están en formato JSON en objetos de la clase MetodoPago. Esto se ha realizado de esta forma ya que el mapeo se hacía de forma deficiente debido a que se trata de una jerarquía de clases, en la clase deserializer se indica cómo ha de transformarse cada par clave-valor en atributos de objeto de la clase.
- El paquete terminado en “domain” incluye todas las clases que recogen la información que se maneja en el sistema, mediante anotaciones JPA se indica cómo se han de mapear en base de datos. Cada clase constituye una entidad, y cada entidad es una tabla en base de datos.
- El paquete terminado en “loader” contiene una clase que está anotada para indicar que contiene un Bean. El Bean se ejecuta al arrancar la aplicación y carga en la base de datos información que permite hacer pruebas, y proporcionar contenido inicial para comprobar cómo se ve la aplicación desde el front-end sin invertir tiempo en introducir toda la información de forma manual, lo cual sería muy ineficiente.
- El paquete terminado en “repositories” contiene las diferentes interfaces que facilitan el acceso a datos sin tener que implementarlo manualmente. En caso de desear métodos diferentes a las operaciones CRUD sencillas, Spring Data, a través de estas interfaces permite que, usando una sintaxis específica, se puedan introducir métodos con comportamientos personalizados sin tener que implementarlos. Igualmente se permite introducir métodos asociados a queries SQL.
- El paquete terminado en “representations” contiene clases que son DTO que extienden a otra clase que permite introducir hiperenlaces. Las API, para ser realmente API REST, deben de guiar mediante hiperenlaces al cliente que hace las peticiones. Estos DTO constituyen la respuesta a ciertas peticiones, y definen qué información de las clases de dominio debe de ser mostrada, pudiendo ser una parte de esta, e incluyen hiperenlaces.
- Hay una agrupación de paquetes que en su parte final comienzan por “security”, y son seguidos por otro nombre, es así ya que tratan sobre el tema de la seguridad de la API.
    - El paquete terminado en “config” contiene clases que se encargan de la configuración de la autenticación y autorización en la API, es decir, se encargan de indicar quien está accediendo a la API, y de qué peticiones se tiene permitido obtener respuesta. Se incluye dentro de la configuración un filtro de seguridad personalizado para que se reconozca la autenticación utilizando tokens JWT. Por otro lado, hay una clase que configura el CORS, indicando qué sistemas pueden hacer peticiones fuera de la aplicación, qué cabeceras son aceptadas, o si se aceptan credenciales, entre otros.
    - El paquete terminado en “jwt” contiene las clases que, por un lado, constituyen el filtro que hará que se identifiquen los usuarios a través de tokens en las peticiones sucesivas a la introducción de credenciales (se comprueba que existe un usuario en el sistema con la información introducida por el token incluido en la cabecera Authorization de las peticiones), y por otro, la clase restante ofrece métodos para el manejo, creación y validación de los tokens.
    - El paquete terminado en “logindomain” contiene DTO específicos para la recepción de las credenciales de los usuarios y para retornar como respuesta, en caso de que el usuario sea correctamente autenticado, el token para las sucesivas peticiones e información relacionada.
    - El paquete terminado en “mail” contiene las clases que tienen la lógica para poder enviar un correo electrónico, incluyendo destinatario/s, titulo, mensaje y

archivos adjuntos. Se han incluido entre los paquetes de seguridad ya que su finalidad es enviar un código secreto al usuario que solicite cambiar su contraseña al ser olvidada. Este accede a su correo, obtiene el código y al hacer la petición correspondiente con este, en el caso de ser correcto, podrá modificar su contraseña.

- El paquete terminado en “service” contiene clases que se utilizan para obtener información de los usuarios almacenada en base de datos dentro del contexto de seguridad, se utilizan para los procesos de autenticación y autorización.

- Por último, el paquete terminado en “tools” contiene clases cuyas funcionalidades son genéricas y auxiliares para el resto de las clases. Por ejemplo: incluyen métodos para validar DNI, o para eliminar espacios en URL y sustituirlos por caracteres válidos equivalentes.

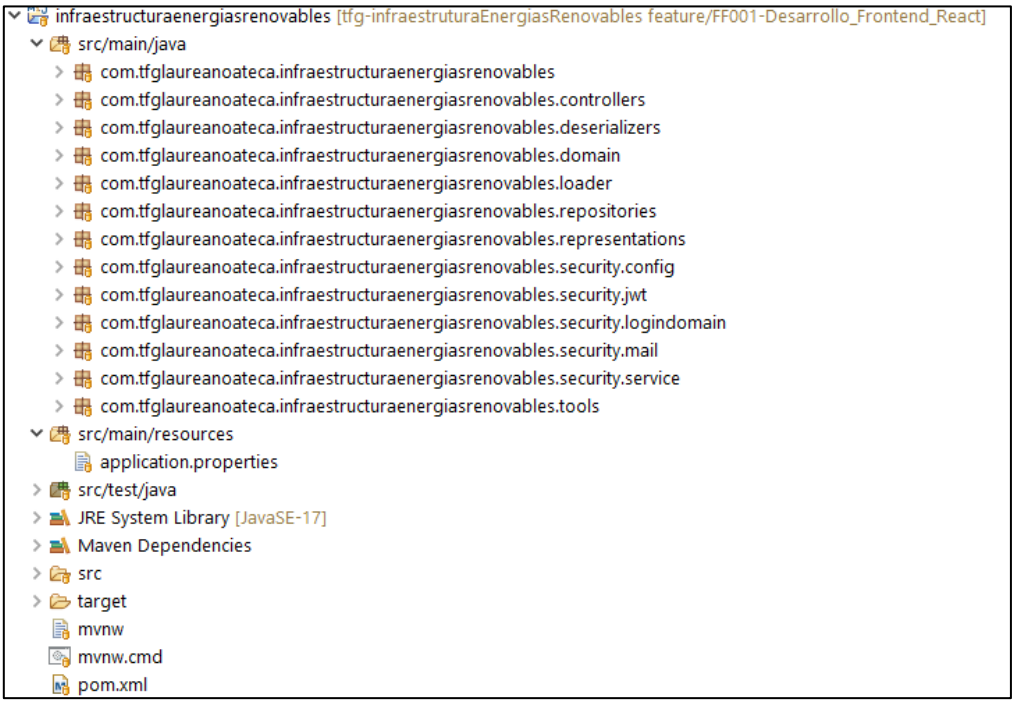

Figura 19: Estructura – Back-end

4.1.3Capa de acceso a datos

La capa de acceso a datos está constituida por las interfaces Repository, por las clases de dominio (entidades) y por la configuración para establecer conexión con la base de datos. Todos estos aspectos permiten que los datos de base de datos y los de la capa de negocio estén sincronizados bajo demanda.

4.1.3.1 Repositorios

Como se ha mencionado previamente, Spring Data proporciona interfaces que permiten el acceso a los datos de base de datos sin tener que implementar de forma explícita clases DAO. Se consigue una mayor abstracción del contexto de persistencia.

En la Figura 20 podemos observar que además de las operaciones CRUD que ofrece de forma automática JpaRepository para la clase Producto con clave primaria de tipo Long, tenemos definidos otros métodos en la interfaz.

Con el mecanismo del generador de consultas integrado en la infraestructura del repositorio de datos de Spring se pueden crear consultas restrictivas sobre entidades del repositorio. Por ejemplo, en el método findByOrderByNombreAsc se está indicando que se desea encontrar productos por la propiedad nombre y que se devuelvan ordenados por orden ascendente. Se permiten muchas combinaciones, algunas de ellas se pueden ver en el código fuente de los demás repositorios.

Otra forma de definir métodos personalizados, cuyo comportamiento deseamos que sea muy específico, es el uso de las anotaciones @Query y @Modifying. La primera anotación permite definir métodos que no modifican la base de datos y cuya salida está determinada por una consulta SQL. La segunda anotación nos permite definir métodos que modifican el contenido de la base de datos, operaciones como INSERT, UPDATE, o DELETE. En la interfaz de la Figura 20 se puede observar un ejemplo de cada uno de los casos anteriores.

En el primer método anotado con @Query indicamos mediante lenguaje SQL que queremos todos los productos cuyo campo activo tenga valor 1, ordenados por el campo nombre en orden ascendente. La salida resultante de la consulta es volcada en una lista de productos. En este caso no se tiene en cuenta la sintaxis del método para definir su comportamiento.

En el segundo método además de la anotación @Query, tenemos la anotación @Modifying, esta última permite realizar operaciones que modifiquen el estado de la base de datos. En específico con este método estamos eliminando los productos que son piezas.

```
40 @Transactional
41 @Repository
42 public interface ProductoRepository extends JpaRepository<Producto, Long> {
43     @Lock(LockModeType.PESSIMISTIC_READ)
44     public Optional<Producto> findWithPessimisticReadById(Long id);
45
46     @Lock(LockModeType.PESSIMISTIC_WRITE)
47     public Optional<Producto> findWithPessimisticWriteById(Long id);
48
49     @Lock(LockModeType.PESSIMISTIC_READ)
50     public Optional<Producto> findWithPessimisticReadByNombre(String nombre);
51
52     public List<Producto> findByOrderByNombreAsc();
53
54     @Query(value = "SELECT p.* FROM producto as p where p.activo=1 order by nombre asc", nativeQuery = true)
55     public List<Producto> productosSoloDisponiblesOrdenados();
56
57     @Modifying
58     @Query(value = "DELETE FROM producto where es_pieza=1", nativeQuery = true)
59     public void eliminarProductosQueSeanPiezas();
60 }
```

Figura 20: Ejemplo de Repositorio personalizado

4.1.3.2 Entidades

Las clases de dominio son clases POJO que se convierten en entidades cuando se utilizan sobre ellas anotaciones JPA. Cada entidad mapea a una tabla en base de datos, y los atributos se mapean a campos. Las relaciones entre tablas también se definen en las entidades.

En la Figura 21 podemos observar una serie de anotaciones que definen como se ha de mapear la entidad en la base de datos. Las anotaciones más frecuentes son:

- @Entity: indica que la clase se identifica como entidad.
- @Id: indica que el atributo sobre el cual se encuentra la anotación se trata de la primary key, o identificador único de cada tupla en base de datos.
- @GeneratedValue: Indica el modo en que se genera la clave primaria. Dándole valor al atributo strategy indicaremos de qué forma se le da valor al campo. En el caso anterior, y en todas las entidades de este proyecto, se ha elegido usar GenerationType.IDENTITY. La base de datos con cada inserción incrementa el valor de una columna. Se produce un incremento automático en el valor de esa columna, lo que en base de datos está muy optimizado.

- @Column: permite definir cómo se mapeará cada atributo a un campo de una tabla en base de datos. Se puede indicar el nombre que tendrá el campo, si admite valores nulos, el tipo que ha de tener el campo, el tamaño máximo reservado para el campo, si se trata de un campo cuyo valor ha de ser único, entre otros.
- @Temporal: permite definir los tipos relacionados con tiempos y fechas en base de datos.
- @Enumerated: configura el modo en el que se mapea un valor enumerado. En este caso se mapea a String y se encontrará en la misma tabla (pedido).
- Anotaciones para definir relaciones entre tablas:
    - @JoinColumn: indica que un atributo actúa como foreign key de otra entidad.
    - @OneToOne: indica que el atributo está asociado con una tupla de otra entidad. Se mapea a una columna foreign key en la tabla donde aparece la anotación.
    - @OneToMany: indica que el atributo está asociado a varias tuplas de otra entidad. Se mapea a una columna foreign key en la tabla de la clase referenciada si se trata de una relación unidireccional, si fuese bidireccional se mapearía en la tabla en la que no se encuentra el atributo mappedBy.
    - @ManyToOne: indica que el atributo (lado con varios elementos), está asociado con una tupla de otra entidad. Se mapea a una columna foreign key en la tabla de la propia clase donde aparece la anotación.

```java
@Entity
public class Pedido implements Comparable<Pedido> {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(unique = true, nullable = false)
    private Long id;// Identificador unico autogenerado
    @Column(nullable = false, length = 8)
    private double costeTotal;
    @Temporal(value = TemporalType.DATE)
    @Column(nullable = false)
    private LocalDate fecha;
    @Column(nullable = true, length = 8)
    @Version
    private Long version;// Se incrementa cada vez que se modifica para el mecanismo de lock optimista.
    /*
     * Relacion de uno a muchos
     */
    @OneToMany
    @JoinColumn(name = "pedido_fk")
    private List<ItemProducto> numItems = new ArrayList<ItemProducto>();
    /*
     * Relacion muchos a uno bidireccional
     */
    @ManyToOne
    @JoinColumn(name = "cliente_fk")
    private Cliente comprador;
    /*
     * Relacion uno a uno
     */
    @OneToOne
    @JoinColumn(name = "metodo_pago_usado_fk", nullable = true)
    private MetodoPago metodoPagoUsado;
    // El enumerado se mapea a String en base de datos
    @Enumerated(value = EnumType.STRING)
    private EstadoPedido estado;

    // Datos del cliente que pueden cambiar y que es importante para la informacion
    // del pedido que no cambien
    @Column(nullable = false, length = 80)
    private String direccionEnvioAlComprar;
    @Column(nullable = false, length = 20)
    private String DNI;
```

Figura 21: Entidad – Anotaciones

Por otro lado, se ha decido usar la estrategia Single Table en las herencias del sistema. La decisión se ha tomado teniendo en cuenta el rendimiento de las consultas. Las opciones son:

- MappedSupperClass, que hace que solo haya entidades por las clases hijas.
- Single Table, que hace que la herencia se guarde en una sola tabla.
- Joined Table, que hace que cada clase tenga su propia tabla. Al hacer consultas a las subclases se requiere unir tablas para acceder a los datos que se heredan de la clase padre, esto provoca un menor rendimiento.
- Table-Per-Class, en la que todas las propiedades de una clase están en su tabla, por lo tanto, en las consultas no es necesario unir tablas.

Como no tenemos jerarquías muy cambiantes no nos interesa la estrategia Joined Table, que es más flexible ya que si es habitual añadir nuevos tipos simplemente se añaden nuevas tablas, y si se desea añadir nuevos atributos solo hay que ir a la tabla concreta. El problema viene al hacer consultas ya que hay que hacer joins con las tablas padres para obtener toda la información.

La estrategia Table-Per-Class duplica información que por un lado está bien porque accediendo a una de las tablas tenemos toda su información, pero por otro lado tenemos información duplicada entre las tablas de la jerarquía, lo que desperdicia memoria. Además, introduce serios problemas de rendimiento si trabajamos con polimorfismo.

Se ha escogido la estrategia de Single Table por ser la que mejor rendimiento ofrece, ya que las consultas se hacen sobre una sola tabla. Como inconveniente tenemos que todos los campos de las clases hijas tienen que admitir nulos, ya que cuando guardemos un objeto que se corresponde a una clase hija, los atributos que no sean los propios de esta clase tendrán valor nulo en la tabla que contiene la jerarquía. La desventaja de esta estrategia en los casos presentes no es tan grande debido a que las jerarquías no son de muchas clases, y las clases no tienen muchos atributos.

Las anotaciones JPA para establecer la estrategia elegida son las siguientes:

```
16 @Entity
17 @DiscriminatorColumn(name = "tipoPago") // Columna que en base de datos tomara como valor BIZUM o Tarjeta, diferenciando
18                                         // a unos de otros
19 @Inheritance(strategy = InheritanceType.SINGLE_TABLE) // La jerarquia se mapeara en BD a una sola tabla
20 @JsonDeserialize(using = MetodoPagoDeserializer.class) // Se indica que la desserealizacion sera personalizada y definida
21                                                        // en esa clase
22 public abstract class MetodoPago {
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     @Column(unique = true, nullable = false)
26     private Long id; // Identificador autogenerado
27
28     @Column(nullable = false, length = 30)
29     private String nombreMetodo;
30
31     @Column(nullable = false, length = 1)
32     private boolean activo = true;
33
```

Figura 22: Clase padre de la jerarquía

```
14 @Entity
15 @DiscriminatorValue(value = "TARJETA") // Indica que la fila en la tabla representa una tarjeta y no un bizum
16 public class TarjetaBancaria extends MetodoPago {
17
18     @Column(nullable = true, length = 30)
19     private String numTarjeta;
20     @Column(nullable = true, length = 80)
21     private String titular;
22     @Column(nullable = true, length = 4)
23     private int mesVencimiento;
24     @Column(nullable = true, length = 4)
25     private int anhoVencimiento;
26     @Column(nullable = true, length = 4)
27     private String CVC;
```

Figura 23: Clase hija TarjetaBancaria

```
12 @Entity
13 @DiscriminatorValue(value = "BIZUM") // Los metodos de pago independientemente del subtipo se guardan en una sola
14                                      // tabla
15 public class Bizum extends MetodoPago {
16
17     @Column(nullable = true, length = 30)
18     private String numeroTelBizum;
19
```

Figura 24: Clase hija Bizum

Cómo podemos observar en la Figura 22 para establecer la estrategia Single Table a la hora de hacer el mapeo en la jerarquía disponemos de dos anotaciones:

- @DiscriminatorColumn: junto con el atributo name podemos definir, para la tabla que contendrá nuestra jerarquía, el nombre de una columna cuyo valor permite diferenciar si una tupla pertenece a una subclase u otra a la hora de mapear de nuevo a objetos.
- @Inheritance: junto con el atributo strategy, cuyo valor ha de ser InheritanceType.SINGLE_TABLE, para aplicar la estrategia que deseamos.

En las Figuras 23 y 24 podemos observar una anotación característica para tratar el mapeo en la jerarquía:

- @DiscriminatorValue: junto con el atributo value se define para cada clase hija el valor que tendrá el campo introducido por la clase padre con la anotación @DiscriminatorColumn. Para la clase Bizum el valor de este campo en la tabla que recoge la jerarquía será “BIZUM”, esto diferencia unas tuplas de otras y permite el mapeo a objetos a su subclase correspondiente.

Se puede consultar el modelo de la base de datos donde se pueden apreciar estos aspectos en el Anexo B.

4.1.4 Capa de negocio

En el presente apartado se indica cómo, mediante anotaciones de la tecnología escogida, la capa de negocio se convierte en una API REST, y qué infraestructura del proyecto es necesaria para dar respuesta a las peticiones correctamente.
Los controllers constituyen el pilar fundamental de la capa de negocio, apoyados por las representations, deserializers y algunas herramientas que se encuentran en el paquete “tools”.

4.1.4.1 Controllers

Para exponer de qué forma se han de anotar los controllers se utilizarán una serie de ejemplos, incluyendo en cada uno, un método HTTP diferente.

```
43 @RestController
44 @RequestMapping("")
45 public class AccesoController {
```

Figura 25: Anotaciones - Controller - Comienzo

En la Figura 25 se pueden observar dos anotaciones que son necesarias en todas las clases que son controladores:

- @RestController: indica a Spring que la clase se trata de un controlador, y que ha de dar respuesta a las peticiones que se correspondan con las rutas que este controlador maneja.
- @RequestMapping: indica una ruta raíz común a todos los métodos que manejan peticiones en el controlador.

El siguiente método maneja peticiones GET y tiene queries en la URL, esta petición se utiliza para hacer consultas:

```
@GetMapping("/clientes/{email}/pedidos")
public ResponseEntity<ListaPedidosClienteRepresentation> pedidosCliente(
        @PathVariable("email") String emailCliente,
        @RequestParam(name = "pendientes", required = false) boolean pendientes, HttpServletRequest request) {
```

Figura 26: Método GET

En la Figura 26 podemos observar las siguientes anotaciones:

- @GetMapping: define la ruta a la que ha de dar respuesta el método correspondiente (petición con método HTTP GET). Cuando un cliente realiza una petición sobre esta ruta, es el método sobre el cual se introduce esta anotación, el encargado de dar respuesta a esta.

- @PathVariable: mapea información que se encuentra en la URL de la petición por su nombre a una variable para poder manejar dicha información. Suele contener información para identificar elementos dentro de una colección de estos.
- @RequestParam: mapea la información contenida en una query incluida o no en la URL por un nombre a una variable para poder manejar dicha información. Suele contener información para realizar filtrados.

En la Figura 27 hay dos anotaciones relevantes:

- @PostMapping: indica que frente a una petición POST a la ruta asociada, el método sobre el que se encuentra la anotación, se encargará de dar respuesta a esta.
- @RequestBody: Spring mapea de forma automática (si puede) el cuerpo de la petición en formato JSON a un objeto del tipo indicado. Deserializa el contenido del cuerpo de la petición.

Este método maneja peticiones POST, esta petición se utiliza para almacenar nueva información o enviar información para obtener una respuesta con esta procesada:

```
@PostMapping("/clientes")
public ResponseEntity<ClienteRepresentation> creaCliente(@RequestBody Cliente cliente,
        HttpServletRequest request) {
```

Figura 27: Método POST

En la Figura 28 nos encontramos con una anotación nueva:

- @PutMapping: indica que frente a una petición PUT a la ruta asociada, el método sobre el que se encuentra la anotación se encarga da dar respuesta a esta.

Este método maneja peticiones PUT, sirve para modificar elementos o para crearlos incluyendo un identificador específico en la URL:

```
@Transactional
@PutMapping("/clientes/{email}")
public ResponseEntity<ClienteRepresentation> modificaCliente(@PathVariable("email") String email,
        @RequestBody Cliente cliente, HttpServletRequest request) {
```

Figura 28: Método PUT

En la Figura 29 nos encontramos con una anotación nueva:

- @DeleteMapping: indica que frente a una petición DELETE a la ruta asociada, el método sobre el que se encuentra la anotación se encarga da dar respuesta a la misma.

Este método maneja peticiones DELETE, sirve para eliminar elementos incluyendo un identificador específico de los mismos en la URL:

```
@DeleteMapping("/clientes/{email}")
public ResponseEntity<ClienteRepresentation> borraCliente(@PathVariable("email") String email,
        HttpServletRequest request) {
```

Figura 29: Método DELETE

Por otra parte, además de las anotaciones, se puede observar que los controladores responden con objetos contenedores del tipo “ResponseEntity”. Esto permite dar una respuesta adecuada a las peticiones, ya que se incluye información del estatus de la respuesta en forma de código HTTP, entre otros.

Los códigos de estatus más habituales son el 200 para indicar que la operación ha sido exitosa en consultas, procesamiento de datos, modificaciones, borrados; el 201 para creaciones, el 404

para indicar que el recurso no existe, y el 409 para indicar que ha tenido lugar un escenario alternativo.

4.1.4.2 Representations

Las representaciones son DTO que permiten añadir hiperenlaces a las respuestas en el cuerpo del mensaje HTTP, junto con el resto de información en formato JSON. Para esto Spring de forma automática utiliza serializers, estos transforman los objetos Java en información en formato JSON que recibe el cliente que envió la petición. Por otra parte, las representaciones sirven para decidir qué información va a ser devuelta.

Se puede elegir qué atributos tendrá el objeto ya que puede desearse omitir información inútil o delicada. Esto no sería posible tan fácilmente si se utilizasen las clases de dominio como respuesta.

En definitiva, según el contexto, se puede desear mostrar más o menos información. La información destinada a un cliente puede no ser la misma que la destinada a un administrador, entre otros. Igualmente, la representación de un objeto en una lista puede ser diferente a la representación de este si se solicita de forma aislada.

Se puede observar cómo no se muestra la misma información de pedido diferenciándose si el destinatario es un cliente, o un administrador:

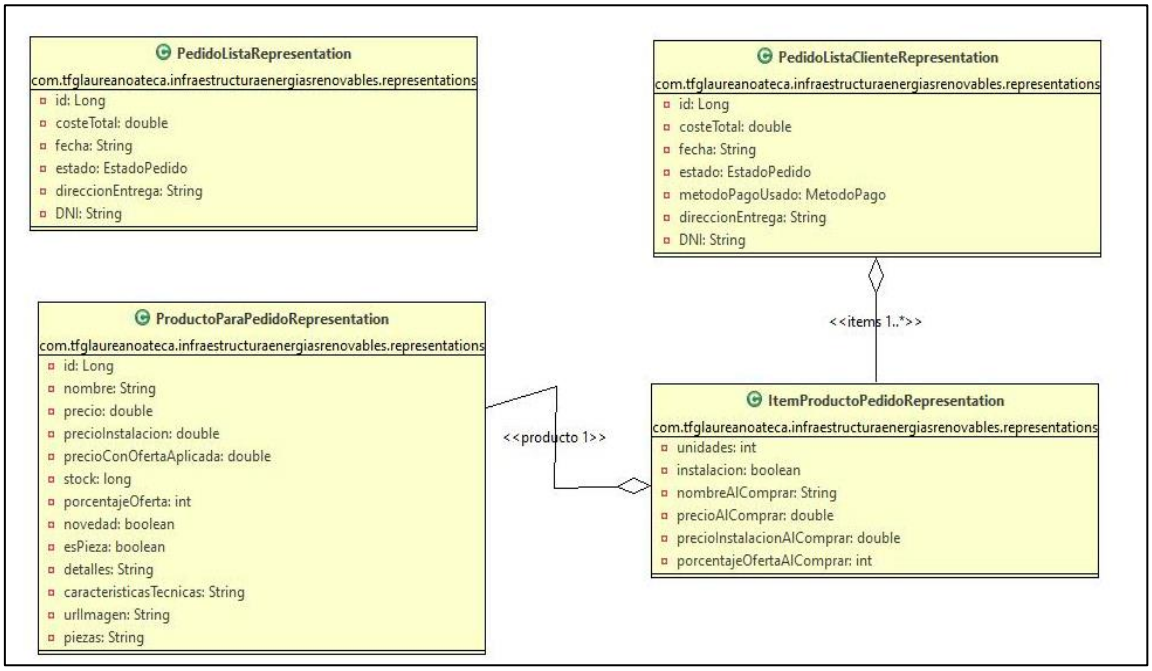

Figura 30: Diagrama de clases – diferencia entre representaciones de pedido

En la Figura 30 para un mismo recurso, como es un pedido, en función de cuál sea el destinatario de la respuesta, se muestra una información u otra, siendo “PedidoListaRepresentacion” la representación para el administrador, y “PedidoListaClienteRepresentation” la representación para el cliente.

Es relevante indicar que los valores que se van a mostrar en el cuerpo de la respuesta los definen los getters, es decir, solo los atributos que tienen getters serán serializados, igualmente solo se incluyen como atributos los datos que se desean mostrar.

Además, para que la representación pueda incluir hiperenlaces debe de extender de la clase “RepresentationModel”. Se han incluido métodos personalizados en las clases representation para incluir los hiperenlaces adecuados, pudiéndose incluir unos en las respuestas de ciertas peticiones, y otros en otras, en función del rol que tenga el usuario que hace la petición.

4.1.4.3 Deserializers

Durante del desarrollo de la API REST ha sido necesario desarrollar un deserializer personalizado, ya que Spring no era capaz de deserializar automáticamente la información que llegaba en el cuerpo de las peticiones para añadir un nuevo método de pago, o para modificar uno ya que forma parte de una jerarquía.

En la Figura 31 nos encontramos con el deserializer personalizado para las peticiones que envían en el cuerpo, o bien un Bizum, o bien una tarjeta bancaria. Para que la información se convierta en el objeto adecuado se comprueba si lo que se recibe en la petición es un Bizum, o una tarjeta, verificando mediante un parser si en el JSON nos encontramos la clave "numeroTelBizum" que es exclusiva de los Bizum o "numTarjeta" que es exclusiva de las tarjetas.

Una vez identificado el tipo de método de pago se obtienen los valores de cada una de las claves de método de pago y se construye un objeto, el cual será devuelto, para que cuando en el controlador se encuentre la anotación "@RequestBody", en el caso de métodos de pago, obtengamos el objeto correctamente.

```
public class MetodoPagoDeserializer extends StdDeserializer<MetodoPago> {

    private static final long serialVersionUID = 1L;

    protected MetodoPagoDeserializer(Class<?> vc) {
        super(vc);
    }

    protected MetodoPagoDeserializer() {
        this(null);
    }

    /**
     * Metodo que recibe como parametros una herramienta para parsear JSON, y el
     * contexto de deserializacion
     */
    @Override
    public MetodoPago deserialize(JsonParser p, DeserializationContext ctxt) throws IOException, JacksonException {
```

Figura 31: Deserializer personalizado

En la clase de dominio de MetodoPago se especifica que se usa este deserializer en lugar del que hay por defecto que no funciona correctamente.

```
@JsonDeserialize(using = MetodoPagoDeserializer.class) // Se indica que la deserealizacion sera personalizada y definida
                                                       // en esa clase
public abstract class MetodoPago {
```

Figura 32: Uso de un deserializer personalizado

En la Figura 32 se observa observa cómo se hace uso de una anotación para indicar el uso de un deserializer personalizado:

- @JsonDeserialize: junto con el atributo using y el nombre de la clase deserializer le dice a Spring que ha de usar esta clase en lugar del deserializer por defecto.

4.1.5 Seguridad en la API REST

La seguridad presente en el back-end es proporcionada por el módulo Spring Security y por el uso de una librería para generar y manipular tokens.

Los aspectos que se controlan en este ámbito son la autenticación y la autorización. Es necesario realizar ciertas configuraciones para indicar qué roles de los usuarios pueden acceder a qué recursos, como autenticar a los usuarios por medio de tokens sin necesidad de introducir credenciales (usuario y contraseña) en cada petición, qué clientes externos a nuestro sistema pueden realizar peticiones a la API, y en qué condiciones.

Todos estos aspectos los trataremos en los siguientes apartados.

4.1.5.1 Autenticación y Autorización

En el momento en el que la API REST arranca, el framework de Spring carga las configuraciones definidas en las clases de configuración, las anotaciones, entre otros.

Entre las configuraciones se encuentra la de Spring Security, que esta relacionada con la autenticación y la autorización. También se encuentra la configuración de CORS.

Posteriormente Spring crea el contexto de la aplicación, que contiene los beans y componentes definidos en la aplicación, entre ellos están los controladores, los servicios, repositorios…
Entre estos beans se encuentran los de Spring Security, que incluyen los filtros de seguridad, los proveedores de autenticación, entre otros.

En el momento en el que el cliente hace una petición a la API REST Spring Security la intercepta, si la petición requiere autenticación, Spring Security verifica las credenciales del usuario, ya sea mediante usuario y contraseña, tokens, u otros métodos.

Si el usuario se autentica con éxito Spring Security verifica si el usuario puede acceder al recurso de la petición. Para la autorización se comprueba si el usuario cumple una serie de criterios definidos en la configuración de Spring Security. Si el usuario cumple estos criterios se le da acceso al recurso.

La siguiente figura muestra el método principal de la clase WebSecurityConfig, este indica para cada ruta, qué métodos HTTP son permitidos, para qué roles de usuario.

```
@Bean
public SecurityFilterChain web(HttpSecurity http) throws Exception {
    http.csrf().disable().authorizeHttpRequests((authorize) -> authorize
            /*
             * Aqui se indica que rutas se pueden acceder, con que metodos y por que roles
             * de usuario.
             */
            .requestMatchers(HttpMethod.GET, "/clientes/{email}/cesta")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.GET, "/clientes/{email}/cesta/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.PUT, "/clientes/{email}/cesta/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.DELETE, "/clientes/{email}/cesta/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.GET, "/clientes/{email}/metodosPago")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.POST, "/clientes/{email}/metodosPago")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.GET, "/clientes/{email}/metodosPago/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.PUT, "/clientes/{email}/metodosPago/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.DELETE, "/clientes/{email}/metodosPago/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.GET, "/clientes/{email}/pedidos")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.POST, "/clientes/{email}/pedidos")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.PUT, "/clientes/{email}/pedidos/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.GET, "/clientes/{email}/pedidos/**")
            .access(new WebExpressionAuthorizationManager("hasRole('CLIENTE') and #email == authentication.name"))
            .requestMatchers(HttpMethod.GET, "/pedidos").hasRole("ADMIN")
            .requestMatchers(HttpMethod.GET, "/pedidos/**").hasRole("ADMIN")
            .requestMatchers(HttpMethod.PUT, "/pedidos/**").hasRole("ADMIN")
            .requestMatchers(HttpMethod.POST, "/clientes").anonymous().requestMatchers(HttpMethod.GET, "/clientes")
            .hasRole("ADMIN").requestMatchers(HttpMethod.GET, "/clientes/{email}")
            .access(new WebExpressionAuthorizationManager(
                    "hasRole('ADMIN') or ((hasRole('CLIENTE') and #email == authentication.name))"))
            .requestMatchers(HttpMethod.PUT, "/clientes/{email}")
            .access(new WebExpressionAuthorizationManager(
                    "hasRole('ADMIN') or ((hasRole('CLIENTE') and #email == authentication.name))"))
            .requestMatchers(HttpMethod.DELETE, "/clientes/{email}").hasRole("ADMIN")
            .requestMatchers(HttpMethod.GET, "/productos").permitAll()
            .requestMatchers(HttpMethod.POST, "/productos").hasRole("ADMIN")
            .requestMatchers(HttpMethod.GET, "/productos/{id}").permitAll()
            .requestMatchers(HttpMethod.PUT, "/productos/{id}").hasRole("ADMIN")
            .requestMatchers(HttpMethod.DELETE, "/productos/{id}").hasRole("ADMIN")
            .requestMatchers(HttpMethod.POST, "/productos/{id}/piezas").hasRole("ADMIN")
            .requestMatchers(HttpMethod.POST, "/clientes/{email}/codigoRecuperacionContrasenha").anonymous()
            .requestMatchers(HttpMethod.PUT, "/clientes/{email}/contrasenha").anonymous()
            .requestMatchers(HttpMethod.POST, "/token").anonymous()).cors(withDefaults())
            .addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class)// Se anhade el filtro al
                                                                                        // resto de los filtros de
                                                                                        // Spring
            .sessionManagement((session) -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

    return http.build();
}
```

Figura 33: Configuración de la autorización

En los siguientes apartados se dará mayor detalle de la implementación necesaria para gestionar la seguridad de la API REST, así como, el proceso que tiene lugar desde que el usuario realiza una petición para autenticarse.

Un usuario realiza una petición de autenticación y en la clase AccesoController, en el método generaToken que maneja la petición, se deserializan las credenciales pasadas en el cuerpo del mensaje encapsulándose estas en una clase para su uso. Contienen el email y la contraseña del usuario.

En la clase de configuración se ha creado un Bean para poder obtener, mediante inyección de dependencias, un objeto AuthenticationManager el cual gestiona la autenticación.

En el método generaToken hacemos uso del AutenticationManager con el método autenticate pasando como parámetro un objeto de la clase UsernamePasswordAuthenticationToken con las credenciales. Si el usuario está en el sistema se crea el SecurityContextHolder, que es un objeto que está en memoria durante todo el contexto de la aplicación y que tiene toda la información de seguridad del usuario. Después en función de la configuración de la clase WebSecurityConfig, que configura la autorización, se verifica si el usuario está autorizado para obtener el recurso que se solicita con la petición, siendo así sigue la ejecución del método.

Si queremos obtener usuarios que se encuentran en base de datos dentro del contexto de seguridad, debemos de presentar al AuthenticactionManager un proveedor, que es DaoAuthenticationProvider. A este hay que proveerle un componente de tipo UserDetailsService, que se apoya en la clase UserDetails, y otro de tipo PasswordEncoder. Al hacer esto, el proceso de autenticación pasa por el AuthenticationManager y llama al DaoAuthenticationProvider para traer la información del usuario de la base de datos. Con esta información el AuthenticationManager, si la información se corresponde, autentica al usuario y genera el SecurityContextHolder.

La clase UserDetailsService permite buscar a un usuario en base de datos, en este caso por el email, y devuelve la información necesaria del mismo dentro del contexto de seguridad.

La clase UserDetails encapsula la información necesaria de un usuario dentro del contexto de seguridad. Esta recibe un objeto con el usuario y recoge solo la información relacionada con la seguridad, se asignan los roles del usuario, que son tan importantes para la autorización, y se recogen email y contraseña.

A continuación, en el método generaToken, se crea un objeto de la clase que implementa UserDetails buscando con el método de la implementación de la interfaz UserDetailsService un usuario en base de datos con el email de las credenciales.

Utilizando métodos auxiliares de la clase JwtUtils, se crea el token a partir de la información del objeto de la clase UserDetails y de una clave secreta encriptada, este token permitirá identificar al usuario en posteriores peticiones.

Finalmente, el método generaToken devuelve el token, en cuyo interior se encuentra como parte de este, el email y los roles del usuario autenticado en forma de claims.

Para las siguientes peticiones el usuario debe de acompañar las mismas con una cabecera Authorization con el valor “Bearer “y el token recibido como respuesta en la petición de autenticación inicial.

Para que el usuario se pueda autenticar en estas peticiones mediante tokens, se ha de crear un filtro e incluirle en la cadena de filtros de Spring Security, esto con la finalidad de procesar el token y comprobar que la información contenida en él se corresponde con la de un usuario de base de datos.

Después de la autenticación del usuario, en la fase de autorización, a partir de los roles que este usuario posea, y de la configuración de WebSecurityConfig, se le dará respuesta o no a la petición realizada por el mismo.

La clase JwtTokenFilter es un filtro que utilizando UserDetailsService y JwtUtils (clase auxiliar para manejar tokens) verifica que el usuario que hace una petición, e incluye la cabecera Authorization con el token, es un usuario del sistema.

Para esto se procesa el valor de la cabecera obteniendo únicamente el token. A partir del token y el método extractUsername de la clase auxiliar JwtUtils se obtiene el email del usuario haciendo uso de la clave secreta. Si el email no es nulo y el usuario no está autenticado, se obtienen los datos de seguridad del usuario haciendo uso de las implementaciones de las clases UserDetails y UserDetailsService. Si el token es válido (se comprueba con el método validateToken de la clase JwtUtils) se autentica al usuario en el sistema, y se sigue con el siguiente filtro de la cadena. Después de estar autenticado, de nuevo se comprueba si está autorizado a tener acceso al recurso que se solicita en la petición.

Si un usuario hace una petición sin token, para Spring Security este usuario tiene rol de usuario anónimo.

En la Figura 33 podemos observar la configuración de la autorización para todas las rutas, en ella se indica que métodos son permitidos para los administradores, cuales, para los usuarios identificados como clientes, cuales para los usuarios anónimos (usuarios sin identificar, ni por credenciales, ni por token), y cuales, para cualquier usuario, para todas las rutas de nuestro sistema.

La línea verde subraya un ejemplo de método permitido a los usuarios anónimos. No hay que confundir usuario anónimo con cualquier usuario, es un usuario que está necesariamente sin identificar. Solo los usuarios anónimos pueden registrarse en el sistema.

La línea roja incluye nuestro filtro personalizado para autenticar a los usuarios por sus tokens en la cadena de filtros, de forma que, cuando se carguen los filtros de seguridad se cargue también y sea funcional.

4.1.5.2 Configuración de CORS

CORS significa intercambio de recursos de origen cruzado, es un mecanismo basado en cabeceras HTTP que permiten al servidor indicar qué clientes pueden acceder al contenido de la API REST como medida de seguridad. Esto es así porque JavaScript y CSS podrían cargar sin que el usuario lo supiese contenido de otros servidores, (quizá contenido malicioso).

```
@Configuration
public class CorsConfig {

    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Arrays.asList("http://localhost:4200", "http://localhost:3000"));// Se permite
                                                                                                          // React en
                                                                                                          // el puerto
                                                                                                          // 3000
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));// Se permite obtener
                                                                                       // informacion, anhadir,
                                                                                       // modificar y borrar.
        configuration.addAllowedHeader("*");
        configuration.addExposedHeader("*");
        configuration.setAllowCredentials(true);// Permite credenciales
        configuration.setMaxAge(Duration.ofSeconds(3600));
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

Figura 34: Configuración de CORS

Como se puede observar en la Figura 34 en la configuración se definen los dominios y puertos que son permitidos, nos interesa localhost y el puerto 3000 ya que son la dirección y el puerto del front-end realizado con React; también se definen los métodos permitidos, se permiten todas las cabeceras, y se permiten credenciales.

Es necesario indicar que se utiliza como dirección localhost ya que el servidor de React para este trabajo es local. La comunicación entre front-end y back-end se realiza dentro del mismo equipo.

De la misma forma, el servidor de Spring, que arranca la API REST, tiene como dirección localhost por la misma razón.

En caso de desear que tanto el front-end, como el back-end, fueran accesibles para el resto de los usuarios de internet (siendo esta decisión la más razonable), habría que asignar como dirección, una dirección publica a la que tenga acceso cada servidor de forma independiente.

4.1.5.3 Sistema de recuperación de contraseña

Se ha implementado un sistema de cambio de contraseña seguro para que los clientes que no recuerdan la misma puedan acceder al sistema de nuevo con una nueva contraseña.

El proceso consta de dos fases:

1. La primera fase del sistema consiste en realizar una petición POST a la ruta "/clientes/{email}/codigoRecuperacionContrasenha" donde {email} se sustituye por el email del cliente. En el método encargado de manejar la petición (método de la clase AccesoController), se verifica si el cliente existe, y en caso de ser así se crea una cadena de caracteres aleatoria de diez elementos que se almacena con el usuario en base de datos, y se envía al correo del usuario. El usuario para acceder al correo electrónico con el que está registrado y obtener el código ha de ser quien dice ser.

2. En la segunda fase se hace una petición PUT a la ruta "/clientes/{email}/contrasenha". El método encargado recibe en el cuerpo de la petición la nueva contraseña, y en la cabecera Authorization el código de seguridad. Si el código de seguridad coincide con el almacenado para el usuario identificado con el email pasado en la URL de la petición, se efectúa el cambio de contraseña.

   De esta forma se puede efectuar un cambio de contraseña utilizando la seguridad del servicio de correo electrónico del usuario, teniendo los servicios actuales una seguridad que implementa autenticación multifactorial.

4.1.6 Aspectos funcionales relevantes

4.1.6.1 Bloqueos optimista y pesimista

Dado que la aplicación web permite su uso simultaneo a una gran cantidad de usuarios, y cierta información del sistema puede ser alterada por varios de estos usuarios a la vez, pueden tener lugar inconsistencias en la información que posee cada usuario.

Esto puede tener consecuencias que provoquen un gran impacto en el correcto funcionamiento del sistema y por lo tanto en el negocio subyacente. Un ejemplo claro de estas inconsistencias puede tener lugar cuando un usuario introduce en su cesta de la compra un producto con una serie de unidades de este a adquirir, siendo el stock limitado, mientras que otro usuario ha realizado la misma operación sobre el mismo producto superando entre las unidades de los dos el stock disponible para este. Los dos clientes tramitan con muy poco tiempo de diferencia su compra de forma que para los dos aún hay stock disponible para adquirir sus respectivos productos de la cesta.

Como es lógico tenemos una inconsistencia en la información, y un problema que solventar con el cliente ya que no tenemos stock para satisfacer uno de los pedidos, pero hemos permitido realizar el mismo.

Para evitar este tipo de colisiones provocadas por cambios en los datos, sin poseer la versión más actual de los mismos, hay una serie de mecanismos que se han aplicado en el sistema. En concreto, se han aplicado dos estrategias:

- Bloqueo optimista
- Bloqueo pesimista

La primera estrategia (bloqueo optimista) es viable cuando la circunstancia que provoca la colisión no es muy frecuente, por ejemplo, que un usuario modifique sus datos de perfil y un administrador modifique estos mismos datos de forma simultánea no es muy probable. Esta estrategia obliga a cargar al usuario, que tiene la información desactualizada, la información más reciente con el fin de tener una versión correcta de los datos y poder realizar modificaciones sobre los mismos de forma consistente.

Si hay multitud de usuarios intentando realizar cambios en la información, para cualquiera de ellos esta estrategia no es viable, ya que no van a poder disponer de información actual la mayoría del tiempo.

En estos casos se utiliza la estrategia de bloqueo pesimista, que hace que la información no sea accesible al resto de usuarios hasta el usuario que está haciendo uso de esta desbloquee el recurso, permitiendo que otro usuario bloquee la información para uso personal. Esta estrategia provoca por otro lado un menor rendimiento, ya que tendremos multitud de usuarios esperando por el mismo recurso, pero en ciertas circunstancias es la solución más viable.

En este proyecto se ha utilizado el bloqueo optimista en estas circunstancias:

- Colisiones entre administradores que modifican la información de un mismo producto. No es una colisión muy frecuente y es resultado de una comunicación deficiente entre administradores.
- Colisiones entre administradores que cambian el estado de un pedido, no es muy habitual que dos administradores se encarguen de gestionar el mismo pedido simultáneamente.
- Colisiones entre cliente y administrador al modificar información de un pedido. El cliente realiza el pago de un pedido cuyo estado es pendiente de pago y el administrador desea cambiar el estado de este en el mismo instante. Muy poco probable.
- Colisiones entre administradores modificando los datos de perfil de un cliente. Altamente improbable ya que esta funcionalidad está pensada para que el administrador ayude al cliente bajo demanda de este, que haya dos administradores realizando la misma labor no es lo habitual.
- Colisiones entre cliente y administrador modificando datos personales del cliente. Esto es muy poco probable ya que la funcionalidad que permite que el administrador pueda modificar datos de perfil de un cliente tiene como finalidad ayudar a un cliente con dificultades en el uso de la tecnología a modificar sus datos al tratar con atención al cliente, pudiendo cambiar dicha información el administrador.

Por otro lado, se ha elegido el bloqueo pesimista en estos casos:

- Colisión entre un administrador realizando cambios en la información de un producto y clientes cambiando el stock de este mediante la tramitación de pedidos. Muy frecuente porque los clientes, al ser numerosos, realizarán pedidos en un flujo constante, mientras que el administrador, aunque no es tan frecuente que realice estos cambios, sí que encontrará en el momento de hacerlos a usuarios realizando pedidos.
- Colisión entre usuarios tramitando pedidos, provocarán fácilmente inconsistencias en los productos ya que, al tramitar el pedido, el stock del producto que se adquiere se actualiza en función del número de unidades adquiridas por el cliente. Es muy probable que varios usuarios tengan en su cesta de la compra el mismo producto en el momento de tramitar el pedido.

Como se implanta el bloqueo optimista en Spring:

Para utilizar el bloqueo optimista, debe agregarse a la clase de entidad un atributo con la anotación **@Version**. Este atributo puede ser de tipo int, Integer, short, Short, long, Long, o java.sql.Timestamp, escogiéndose el tipo Long en este sistema. El atributo version es administrado por el proveedor de persistencia, no es necesario cambiar su valor manualmente. Si se cambia la entidad, el número de versión se incrementa en 1.

Si la versión original no coincide con la versión de la base de datos al guardar la entidad, se produce una excepción, que se traducirá en una respuesta con código de estatus 409. Por esta razón no es recomendable utilizar este sistema en situaciones de colisión muy probables.

Cómo se implanta el bloqueo pesimista en Spring:

Con el bloqueo pesimista, las filas de la tabla se bloquean a nivel de base de datos. Como bien nos señala (Yaremenko, F, 2022), hay tres tipos de bloqueo pesimista en JPA:

- Lectura pesimista – nos permite adquirir un bloqueo compartido, y la entidad bloqueada no se puede cambiar antes de una confirmación de transacción. Varios usuarios pueden acceder al recurso, pero solo para su consulta, si se intentan realizar cambios, habiendo hecho un bloqueo de lectura, salta una excepción.
- Escritura pesimista – permite adquirir un bloqueo exclusivo, y la entidad bloqueada se puede cambiar.
- Incremento forzado pesimista – permite adquirir un bloqueo exclusivo y actualizar la columna de versión, la entidad bloqueada se puede cambiar. Este no le utilizaremos.

En la Figura 35 podemos ver, que, para utilizar el bloqueo pesimista, hemos de colocar sobre los métodos de consulta del repository de la clase en la que va a haber colisiones frecuentes, la anotación @Lock.

El tipo de bloqueo se indicará dentro de la anotación como LockModeType.PESSIMISTIC_READ ó LockModeType.PESSIMISTIC_WRITE según se desee si la información puede ser compartida solo para lectura, o bloqueada completamente con la idea de realizar cambios en la tupla correspondiente en base de datos.

```
@Transactional
@Repository
public interface ProductoRepository extends JpaRepository<Producto, Long> {
    @Lock(LockModeType.PESSIMISTIC_READ)
    public Optional<Producto> findWithPessimisticReadById(Long id);

    @Lock(LockModeType.PESSIMISTIC_WRITE)
    public Optional<Producto> findWithPessimisticWriteById(Long id);

    @Lock(LockModeType.PESSIMISTIC_READ)
    public Optional<Producto> findWithPessimisticReadByNombre(String nombre);
```

Figura 35: Bloqueo pesimista – Repository

En la Figura 36 se puede observar cómo se usa el bloqueo pesimista de escritura, simplemente se accede a la información con el método correspondiente del repositorio.

Es importante remarcar que es necesario el uso de la anotación **@Transactional**. Esto es así ya que, si tiene lugar un error, si se han realizado ya cambios, estos deben de poder ser reversibles para asegurar la consistencia de los datos. Se trata el método como una transacción, en la que los cambios se hacen efectivos si no ha habido ningún problema durante la ejecución completa del método.

```
@Transactional
@PutMapping("/productos/{id}")
public ResponseEntity<ProductoRepresentation> modificaProducto(@PathVariable("id") Long id,
        @RequestBody Producto nuevoProducto, HttpServletRequest request) {
    try {
        String baseUrl = ServletUriComponentsBuilder.fromRequestUri(request).replacePath(null).build()
                .toUriString();
        Optional<Producto> pAComprobarSiYaExiste = productoRepository
                .findWithPessimisticReadByNombre(nuevoProducto.getNombre());
        Optional<Producto> pACambiar = productoRepository.findWithPessimisticWriteById(id);
```

Figura 36: Bloqueo pesimista – Controller

4.1.6.2 Borrado de información

En esta aplicación web de venta de infraestructura para el aprovechamiento de energías renovables hay información que los usuarios pueden querer borrar, pero que por otra parte es necesaria para el correcto funcionamiento del sistema.

Un cliente puede querer borrar un método de pago de entre los que tiene almacenados en su cuenta, ya que no desea volver a pagar con el mismo. Este método de pago se ha utilizado para costear varios pedidos del cliente. Si se borra de forma permanente del sistema, la información del pedido, que incluye con qué método de pago se ha hecho frente a los gastos que incluía el propio pedido desaparece, y esta información es de gran relevancia.

De igual manera si un usuario es borrado del sistema, y en la información que incorporan los pedidos está la información del comprador, esta información desaparece, deseándose que la misma sea permanente para el sistema. Lo mismo sucede al eliminar un producto.

La solución aplicada para solventar este problema es mixta, por un lado, se ha incluido un atributo en las clases “Producto”, “MetodoPago”, y “Cliente, cuyos objetos son susceptibles de ser borrados. Este atributo recibe el nombre de “activo”, al borrar objetos de estas clases en lugar de hacerlos desaparecer de la base de datos se modifica el valor de este atributo a false, y en función de este valor estos objetos serán o no visibles para ciertos usuarios. Los administradores pueden tener acceso a estos elementos “inactivos”, mientras que a los usuarios anónimos y/o clientes esta información no existe.

Igualmente hay casos en los que se vuelven a habilitar estos objetos, un ejemplo de esto es que se ha borrado/inhabilitado un método de pago para un cliente, pero este cliente desea añadir un nuevo método de pago con el mismo nombre o número de tarjeta. El proceder en este caso es habilitar el método de pago realizando modificaciones en los campos cuyos valores son diferentes a los introducidos por el usuario a la hora de querer añadir este método de pago. Hay más situaciones similares para las demás clases.

La otra solución ha sido duplicar cierta información que es susceptible de ser modificada y que debe almacenarse con el valor que tenía en cierto momento.

Esta situación se da cuando un usuario tramita un pedido. Los ítems de este pedido tienen un precio determinado en el momento de la compra, y este precio debe de ser inamovible. Cuando un administrador modifica el precio de un producto y el precio del ítem en el pedido depende directamente del precio del producto, el precio al que el usuario adquirió el producto se pierde. Para solventar esta situación se han duplicado, en la clase “ItemProducto”, ciertos datos del producto al realizar los pedidos, ya que los ItemProducto no se eliminan y la información del precio del producto no se puede modificar. Se ha utilizado esta solución en otras clases.

4.2 Implementación - Front-end

En este apartado se explicarán los aspectos más relevantes encontrados durante el desarrollo del front-end, incidiendo en el uso de las tecnologías escogidas y sus beneficios. Ofrece el medio por el que el usuario interactúa con la capa de negocio de forma intuitiva y cómoda por lo que es de gran relevancia.

4.2.1 Tecnologías escogidas

El front-end del proyecto se lleva a cabo usando el lenguaje JavaScript, usando como librería principal React. Se ha escogido React por las siguientes razones:

- Es una tecnología basada en componentes reutilizables, esto permite que para cada una de las páginas que constituyen el front-end de este proyecto se puedan utilizar

componentes comunes, y no tener que implementar el mismo elemento visual en las diferentes páginas. También hace que la aplicación sea más escalable y que en caso de encontrarse errores, estos se encuentren localizados.

- El uso de JSX, extensión de la sintaxis de JavaScript, facilita en gran medida el desarrollo del front-end ya que lo que antes se hacía de forma independiente, el maquetado por un lado, y la lógica por otro, ahora se hace de forma conjunta, trabajando el maquetado usando JavaScript, y combinándolo en un mismo archivo junto con la lógica de la interfaz de usuario.
- React cuenta con una gran comunidad por lo que hay una gran cantidad de librerías externas, por ejemplo, se hace uso de Axios para realizar las peticiones a la API REST.
- Puede ser interpretado tanto por navegadores antiguos (haciendo uso de la librería Babel), como por los nuevos.
- Una de las características interesantes de React, de cara al rendimiento, es que puede generar el DOM “Modelo de datos del Documento”, permitiendo visualizar los cambios en los datos sin tener que renderizar toda la página de nuevo, es decir, cada vez que cambia un elemento en una página se actualiza ese elemento específico, siendo mucho más eficiente.

4.2.2 Estructura del proyecto

En este apartado se aborda cómo está estructurado un proyecto que utiliza la librería React y en específico como está estructurado este proyecto.

Como observaremos en la Figura 37, en la columna de la izquierda, vemos la estructura general del proyecto React, en el resto de las columnas vemos en mayor detalle los componentes reutilizables, las páginas, y los servicios que conforman la interfaz de usuario, y que constituyen el eje vertebral del front-end.

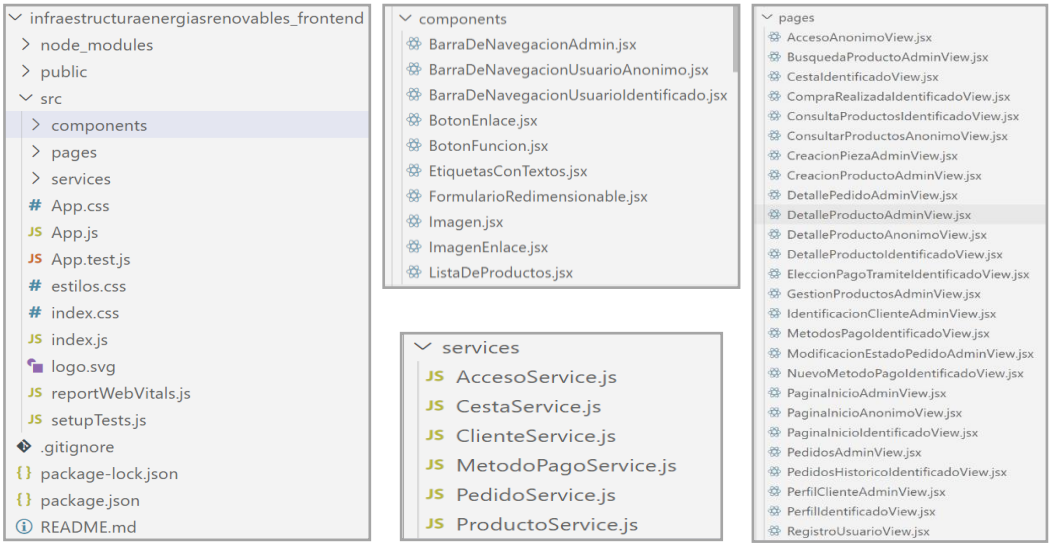

Figura 37: Estructura del front-end

La estructura consta de varios paquetes y archivos:

- La carpeta “node_modules” contiene las librerías y dependencias que se utilizan en el proyecto.
- La carpeta “public” contiene archivos que son leídos por el navegador, el más importante “index.html” que es sobre el que React construye toda la aplicación
- La carpeta “src” es la más importante del proyecto, ya que contiene toda la aplicación React. La estructura interna de esta carpeta es propia de este proyecto y responde a la arquitectura previamente diseñada.

- La carpeta “pages” contiene los componentes de más alto nivel del front-end, solo por debajo de App.js, que es el componente que orquesta la navegación entre las páginas, y es el de más alto nivel. Cada archivo de la carpeta “pages” representa una página de la interfaz de usuario. Las páginas hacen uso de componentes reutilizables, componentes de más bajo nivel que funcionan como las piezas del puzzle que constituye cada página. Las páginas hacen uso de estos componentes, pero incluyen partes que se implementan en las propias páginas, tienen su estado y proporcionan propiedades con la información necesaria para estos componentes reutilizables.
- La carpeta “components” contiene componentes reutilizables, componentes como pueda ser la barra de navegación de los clientes, que aparece en la parte superior de todas las páginas destinadas a los usuarios con el rol de clientes. También contiene componentes notablemente más complejos que permiten generar un formulario altamente personalizable mediante propiedades, que devuelve la información requerida a la página correspondiente.
- La carpeta “services” contiene, agrupadas por funcionalidad en ficheros, todas las funciones para obtener la información necesaria del back-end. Las peticiones se corresponden, como es de esperar, con los métodos de los controladores del back-end que gestionan las peticiones aquí enviadas.
- El archivo “App.css” es una página de estilo generada automáticamente al crear el proyecto.
- Por otro lado, se ha creado un fichero llamado “estilos.css” que contiene los estilos utilizados en todos los componentes del front-end.
- El archivo “App.js” contiene el componente principal. Mediante la librería “react-router-dom” este componente redirecciona, en función de las diferentes rutas que aparecerán en la barra superior del navegador, a las páginas de la carpeta “pages”, inicialmente el servidor arrancará con la página de inicio para los usuarios anónimos.
- El archivo “index.js” es el primer punto de entrada, está definido por node, y renderiza nuestro componente principal “App.js” en la página “index.html” que carga nuestro navegador.

- El archivo “package.json” es la carpeta raíz del proyecto, es como el pom.xml de nuestro back-end. Contiene el nombre del proyecto, la versión del paquete, una descripción, el punto de entrada del proyecto, una colección de scripts del proyecto, las dependencias del proyecto (muy importante), entre otros.
- El archivo “package-lock.json” sirve para que cuando alguien clona nuestro repositorio y ejecuta “npm install”, npm instale la versión exacta de los paquetes que nosotros habíamos instalado sin tener en cuenta los ^ y ~ de delante de las versiones de las librerías.

4.2.3 Capa de servicio

En este apartado se mostrará el funcionamiento de la capa de servicio del front-end, prestando especial atención a la librería escogida (Axios) para realizar las peticiones a la API REST que contiene la lógica de la aplicación web.

Se mostrará un ejemplo con dos peticiones.

```
async function cliente(email, token) {
  return axios
    .get("http://localhost:8080/clientes/" + email, {
      headers: {
        Authorization: token,
      },
    })
    .then(function (response) {
      return response;
    })
    .catch(function (error) {
      return error.response;
    });
}
```

Figura 38: Petición HTTP – GET

Las peticiones GET, como la de la Figura 38, solo tienen dos argumentos posibles, la URL, y el objeto con las cabeceras, si este último no fuera necesario se incluye un objeto vacío.

```
async function creaCliente(cliente) {
  return axios
    .post(
      "http://localhost:8080/clientes",
      {
        correoElectronico: cliente.correoElectronico,
        nombre: cliente.nombre,
        contrasenha: cliente.contrasenha,
        primerApellido: cliente.primerApellido,
        segundoApellido: cliente.segundoApellido,
        direccionResidencial: cliente.direccionResidencial,
        numeroTelefono: cliente.numeroTelefono,
        dni: cliente.dni,
      },
      {}
    )
    .then(function (response) {
      return response;
    })
    .catch(function (error) {
      return error.response;
    });
}
```

Figura 39: Petición HTTP – POST

En la petición POST de la Figura 39 solo es necesario pasar como argumentos la URL del recurso, el cuerpo del mensaje, y si fuera necesario, que en este caso no lo es, un tercer parámetro con las cabeceras. En este caso se pasa como tercer parámetro un objeto vacío ya que esta petición no necesita de token para identificar al usuario, solo los usuarios anónimos pueden registrarse, y la manera de que el back-end considere al usuario como anónimo es no recibir un token que procesar.

Todas las peticiones son asíncronas ya que una vez que se hace la petición no se conoce cuanto va a tardar el back-end en dar respuesta a las mismas, por esta razón se usa async/await para esperar la respuesta de la petición y poder operar con sus datos.

4.2.4 Componentes de bajo nivel – Componentes reutilizables

En este apartado se explica en qué consisten y como se utilizan los componentes reutilizables en las páginas utilizando un componente sencillo como ejemplo.

```
function BarraDeNavegacionUsuarioAnonimo() {
  return (
    <nav className="navbar">
      <Link className="nav-link" to="/home" style={{ textDecoration: "none" }}>
        The Home of the Renewable Enegies
      </Link>
      <a
        className="nav-link"
        href="/anonimo/productos"
        style={{ textDecoration: "none" }}
      >
        Productos
      </a>
      {/**
       * Estos enlaces incluyen un filtro cuyo criterio se pasa en forma de quer
       * dos siguientes casos se especifica que se quiere obtener los productos
       * oferta o son novedad.
       */}
      <a
        className="nav-link"
        href="/anonimo/productos?tipoFiltro=oferta"
        style={{ textDecoration: "none" }}
      >
        Ofertas
      </a>
      <a
        className="nav-link"
        href="/anonimo/productos?tipoFiltro=novedad"
        style={{ textDecoration: "none" }}
      >
        Novedades
      </a>
      <Link className="nav-link" to="/login" style={{ textDecoration: "none" }}>
        Cuenta
      </Link>
    </nav>
  );
}

export default BarraDeNavegacionUsuarioAnonimo;
```

Figura 40: Componente React reutilizable

La Figura 40 muestra una barra de navegación para los usuarios anónimos, en este caso se trata de un componente sin apenas lógica, y que será utilizado en múltiples páginas sin necesidad de repetir el mismo código en estas. Para que los componentes página puedan usar este componente deben importarle, y este componente debe ser exportado.

4.2.5 Componentes de alto nivel - Paginas

En este apartado se abordará como está estructurada una página, y como esta hace uso de la capa de servicios y los componentes reutilizables.

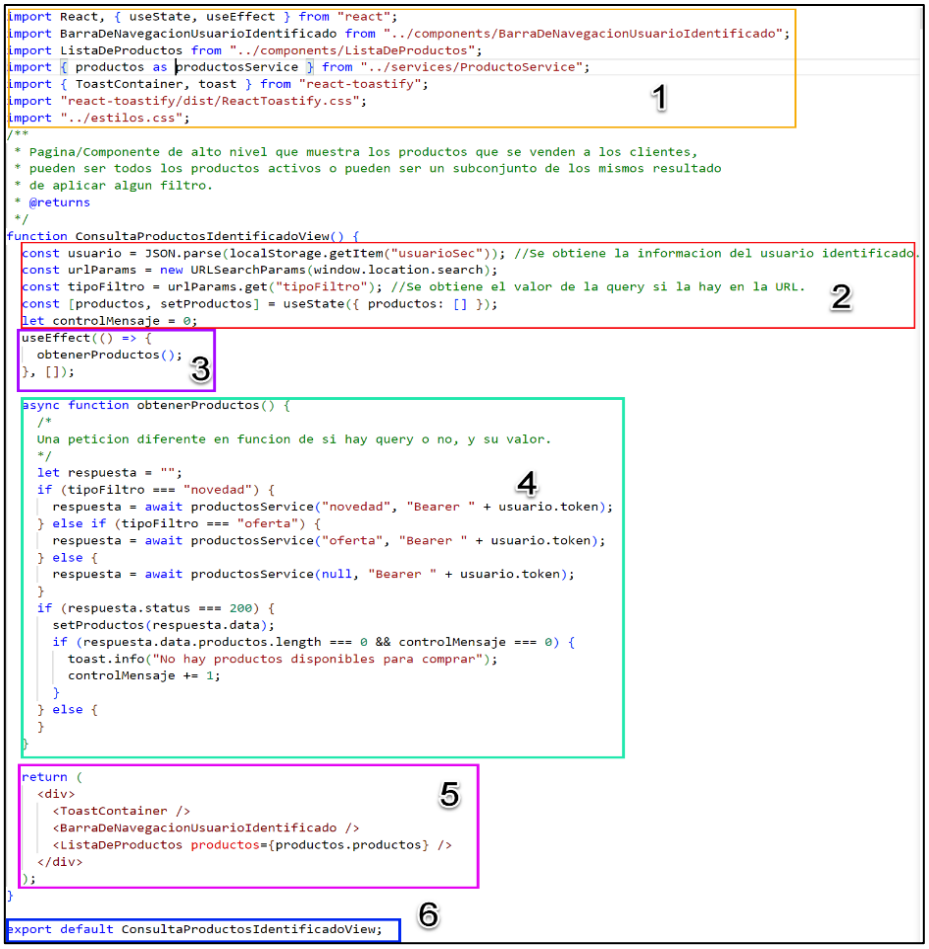

Figura 41: Página - Componente React

En la Figura 41 podemos observar diferentes zonas numeradas, estas agrupan las diferentes áreas de la página por finalidad:

- En el área que se corresponde con el número uno se incluyen los elementos que necesitamos en la página, se importan componentes reutilizables como “BarraDeNavegacionUsuarioIdentificado”, o “ListaDeProductos”, funciones de la capa de servicios como “obtenerListaProductos”, y hooks de React como “useState” o “useEffect”.
- En el área que se corresponde con el número dos se encuentran los valores iniciales que la página necesita, algunos de ellos definen el estado del componente, como es el caso las variables “productos”, y “setProductos”, recibiendo productos un valor inicial haciendo uso del hook useState(). La constante “usuario” alberga la información del usuario identificado, su token y rol, entre otros.

- En el área que se corresponde con el número tres se encuentra un hook muy utilizado, useEffect, este se ejecuta después de que se renderiza el componente por primera vez, se suele utilizar para dar valor al estado del componente con valores procedentes de APIs externas, siendo este el caso, ya que en su interior se ejecuta la función que se encarga de hacer la petición.
- En el área que se corresponde con el número cuatro se encuentran las funciones que usa la página, suelen ser funciones que manejan eventos producidos al interactuar con los elementos que se renderizan, o bien, como es el caso, para realizar peticiones a una API externa y obtener nueva información que, en este caso actualiza el estado del componente, para lo cual se utiliza setProductos. Igualmente se maneja el status de la petición realizada para dar una salida u otra en función de si la petición ha tenido éxito o ha tenido lugar un escenario alternativo.
- En el área que se corresponde con el número cinco se encuentran los elementos que se muestran por pantalla, son elementos visuales que representan botones, barras de navegación, entre otros, con los que el usuario puede interactuar en el navegador. Por eficiencia, y para no repetir código, se utilizan componentes reutilizables de la forma que se muestra en el área, estos pueden recibir información en forma de propiedades, como es el caso del componente “ListaDeProductos”, que recibe como propiedad los productos obtenidos resultado de la petición hecha al back-end.
- Por último, en el área con el número seis se muestra cómo se exporta el componente. Es necesario ya que el componente de más alto nivel, “App.js”, necesita importar los componentes página para poder asociarlos a las diferentes rutas y que estos se muestren al ingresar en las mismas.

4.3 Control de versiones y recursos resultado del desarrollo

En este apartado se ofrece la forma de acceder a los recursos resultado de la implementación del proyecto. Se ha utilizado Git como sistema de control de versiones y se ha seguido el sistema de ramificación GitFlow, ya que es una forma de trabajar en la implementación, limpia, ordenada, e intuitiva, muy conveniente para desarrollar en un equipo de trabajo. Se puede verificar de una forma más visual incluyendo el repositorio en Sourcetree.

Este es el repositorio en GitHub de la implementación del proyecto, permanecerá público durante su evaluación.

LaureanoAteca/tfg-infraestruturaEnergiasRenovables: Este repositorio contiene la implementación del sistema de información del trabajo de fin de grado en ingeniería informática de Laureano Ateca González sobre la distribución de infraestructuras para el aprovechamiento de energías renovables. (github.com)

Igualmente se proporcionará el código fuente como parte del material.

5. Pruebas

En este capítulo se abordarán las pruebas realizadas que verifican el correcto funcionamiento de la aplicación web, haciendo posible la detección de errores para su posterior corrección, evitando así que estos tengan lugar cuando el usuario final haga uso del producto.

5.1 Pruebas - Back-end

En este apartado se especifican las metodologías y herramientas utilizadas para verificar el correcto funcionamiento de la API REST.

5.1.1 Pruebas unitarias

No se han realizado pruebas unitarias ya que el back-end no tiene algoritmos cuya complejidad sea suficiente para justificar su implementación. El trade-off entre planificar e implementar las pruebas y sus beneficios van a ser mínimos.

5.1.2 Pruebas de integración

Las pruebas de integración se han llevado a cabo haciendo uso de Postman, se han realizado de forma manual.

En Postman se ha creado una colección con todas las peticiones posibles que admite la API REST, agrupadas por funcionalidad, por controladores. Como se observa en la Figura 42, en el workspace de Postman se encuentra la colección con la que se ha probado el correcto funcionamiento de la API REST.

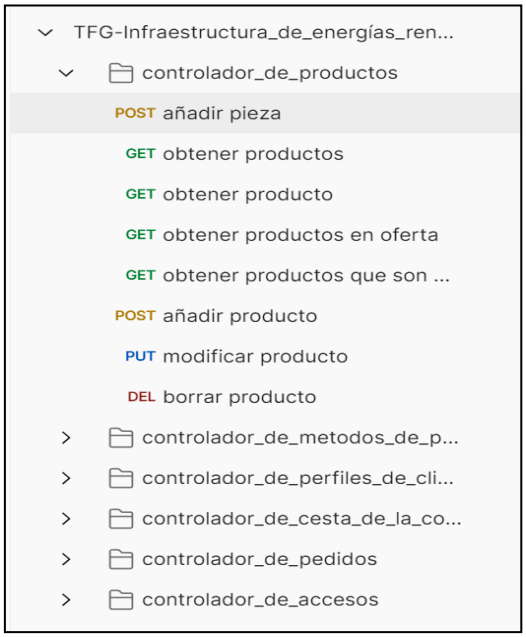

Figura 42: Colección de peticiones - Postman - Pruebas de integración

En la Figura 43 podemos ver una de las peticiones disponibles en la colección del workspace de Postman. En el área de color rojo se encuentra el método HTTP de la petición, en la verde la URL, en la amarilla el código de status de la respuesta, y en la morada el cuerpo de la respuesta.

Se ha verificado para cada una de ellas que las salidas eran las esperadas teniendo en cuenta el estado inicial de la base de datos en el momento de hacer la petición. Igualmente, no solo se han tenido en cuenta los escenarios de éxito, si no también todos los escenarios alternativos.

Para las peticiones que requieren de token, se ha hecho la petición correspondiente para identificar al usuario en el sistema, se ha copiado el token y se ha introducido, seleccionando la opción donde pone “Authorizacion”, debajo de la URL, en el área destinada a los tokens. Es necesario seleccionar en el desplegable “Bearer token” para indicar el tipo de token utilizado. Cuando están todos los parámetros de la petición introducidos, y se ha arrancado la API REST, se puede enviar la petición HTTP deseada.

El workspace de Postman con toda la colección de peticiones está disponible para su consulta a través del siguiente enlace:

https://red-flare-465449.postman.co/workspace/New-Team-Workspace~3c2a71aa-ac69-41df-816e-c43733a82e10/collection/24716695-89da9763-2f9d-43c9-afd0-a9f81295fddc?action=share&creator=24716695

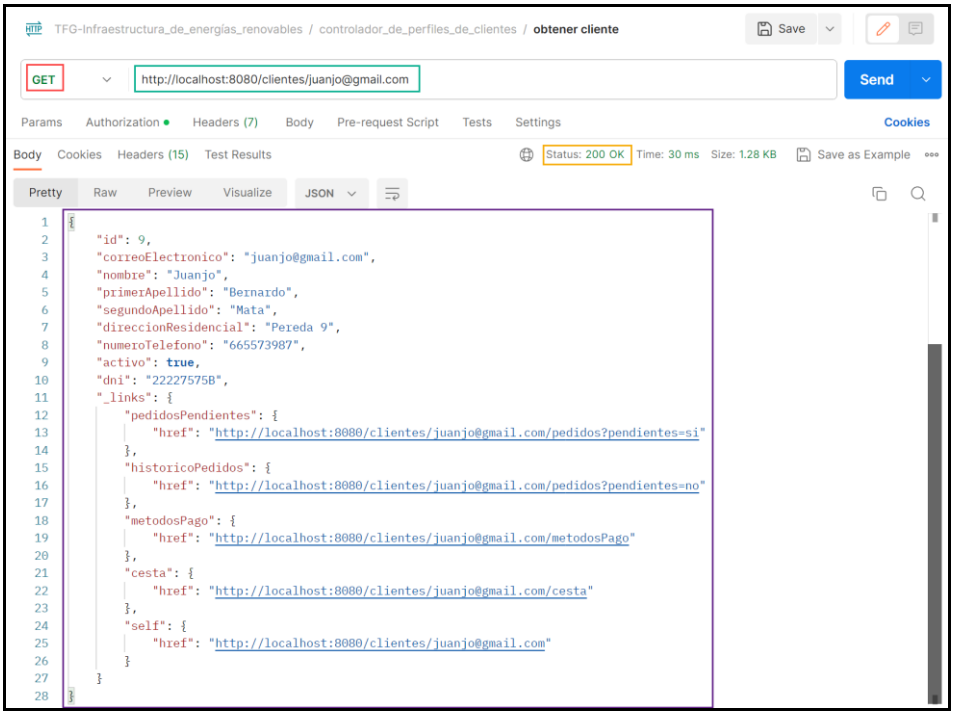

Figura 43: Petición – Postman

5.2 Pruebas - Front-end

En este apartado se comprueba el correcto funcionamiento de la capa de presentación y del sistema completo.

5.2.1 Pruebas end-to-end

Se han llevado a cabo pruebas end-to-end funcionales de forma manual, ya que, debido a la extensión del proyecto no ha habido tiempo suficiente para automatizarlas.

El proceso consiste en verificar que el comportamiento es el esperado para los diferentes flujos de trabajo, analizando tanto el front-end, como el back-end y la base de datos. Se tienen en cuenta tanto los escenarios de éxito, como los alternativos, que no son tan frecuentes.

Algunos de los errores más relevantes detectados durante estas pruebas son:

- En el momento en el que se seleccionaba uno de los productos de la lista de los destinados a los usuarios anónimos, en lugar de mostrar la página con los detalles del producto para los usuarios anónimos, se mostraban los detalles para usuarios identificados. Este error se debía a que cuando un usuario identificado cerraba su sesión, el sistema navegaba a la página de inicio del usuario anónimo sin realizar la operación que cambiaba la información del almacenamiento local que indicaba que el nuevo rol del usuario es el de usuario anónimo.
- Cuando un administrador visualiza los productos que se encuentran en el sistema debe de poder ver tanto los productos activos, como los inactivos (borrados/inaccesibles para los demás usuarios). En la práctica, durante la prueba se borró uno de los productos que contenía piezas, y esta aparecía como inaccesible, pero al realizar la operación con un producto que era una pieza, este pasaba a no ser visible para el administrador, se verificó que se encontraba en base de datos correctamente.
- En los casos de escenarios alternativos, se mostraba varias veces el mismo pop-up cuando solo era necesario que se mostrase una. Esto se debe a que React primero renderiza los componentes y luego permite acceder a datos externos como los de una API y vuelve a renderizar. Las condiciones para que se mostrase el pop-up se cumplían dos veces.
- Los enlaces de la barra de navegación dirigidos a mostrar los productos en oferta y que son novedad no actualizaban la página tras consultar los productos sin filtro alguno. Esto es así ya que React identificaba la página de destino como la misma que la actual, por esa razón no volvía a cargar la página con la URL con las nuevas queries.

5.2.2 Pruebas de aceptación

Se han realizado pruebas de aceptación, pero debido a la falta de tiempo no se han realizado de forma muy exhaustiva.

Se ha proporcionado la aplicación a una persona externa que ha estado practicando con ella. Resultado de esta prueba se ha obtenido un feedback que permite corregir ciertos errores menores, mejorar la experiencia de usuario, y planificar funcionalidades que pueden ser interesantes para versiones posteriores del producto.

Entre las observaciones de la persona que hizo uso de la aplicación encontramos que:

- Se debería de pulir el estilo de la interfaz de usuario de forma que sea más agradable para el usuario visualizar información relevante, y que desee permanecer más tiempo en la aplicación, lo que hará que sea más probable que el cliente vea algún artículo que le resulte de interés, y quiera adquirirlo.
- Sería buena idea incluir alguna funcionalidad que diferencie este servicio de los de la competencia. La propuesta fue la de crear un sistema de puntos para los clientes, en los que, llegada cierta puntuación se haría, en nombre del cliente, una labor positiva para el medio ambiente, como plantar un árbol, o invertir una cierta cantidad de dinero en una actividad cuya finalidad sea promover la concienciación social. Esta iniciativa puede ser un reclamo para los clientes que hayan optado por los servicios ofrecidos, no solo por el ahorro, si no por mejorar la situación medio ambiental actual.
- Se propuso también la creación de una barra de búsqueda para encontrar fácilmente el producto deseado entre las diferentes opciones ofrecidas.
- También se podría incluir la posibilidad de suscribirse a un servicio de mantenimiento para los productos vendidos, para prevenir su rotura, o solucionar la misma a cambio de una mensualidad. Habría varios planes que cubrirían más o menos aspectos en función del coste de estos.

6. Conclusiones

Considero que los objetivos del proyecto se cumplen. Por un lado, se ha conseguido, mediante la aplicación web, poner a disposición de los potenciales clientes un servicio de venta e instalación para poder aprovechar las energías renovables, con el consiguiente ahorro económico a medio-largo plazo.

También se ha intentado concienciar, desde la página de inicio de la aplicación web, del estado del medio ambiente, y de las ventajas del aprovechamiento de las energías renovables, no solo por sus ventajas económicas, sino también por su potencial para mejorar los ecosistemas que nos rodean y nuestro propio bienestar.

En cuanto a los requisitos de la aplicación web, considero que se han cumplido en su mayoría, a excepción de la implementación de ciertas páginas del front-end que ofrecían el acceso a funcionalidades menores, como formularios de modificación de ciertos datos, que se pueden suplir, de manera no tan eficiente con adiciones y borrados. Faltaría también la parte de recuperación de contraseñas y ciertos sistemas de filtrado en el front-end, estando por otro lado toda esta funcionalidad implementada en la API REST.

Desde el punto de vista de la lógica de negocio de la aplicación, recogida en la API REST (back-end), se han implementado satisfactoriamente todos los requisitos.

Habría sido provechoso automatizar las pruebas, ya que al introducir cambios sobre el sistema actual se podría verificar fácilmente si la parte ya desarrollada sigue teniendo el comportamiento esperado.

La parte no implementada del front-end se debe a la gran extensión del proyecto. Se llegó a la conclusión de que no era necesario implementar toda la interfaz de usuario, pero sí las partes fundamentales que permitiesen ofrecer el servicio completo a los usuarios.

Desde el punto de vista personal considero que ha sido muy provechoso para mí planear desde cero cual sería el tema de mi trabajo de fin de grado, recoger requisitos, realizar el diseño, implementar y probar el sistema, y, en definitiva, realizar un proyecto completo.

Esta experiencia ofrece un nuevo nivel de madurez, ya que, en comparación con otros trabajos que por limitaciones temporales son más pequeños y de menor complejidad, este ha resultado un reto mayor.

El uso de nuevas tecnologías, totalmente desconocidas, ya que no se ha usado previamente ninguna similar, hace que haya mejorado mi capacidad de investigación y aprendizaje autónomo, consiguiendo así una herramienta muy valiosa para mi profesión.

Durante mis prácticas pude comprobar la importancia de una buena documentación en los proyectos, ya que tuve que entender un proyecto de gran envergadura sin apenas documentación ni comentarios en el código fuente. Por esta razón he intentado aprender de la experiencia y ofrecer un trabajo con una documentación aceptable.

A raíz de estas experiencias considero que estoy bastante más preparado para entrar en el mundo laboral.

7. Trabajo futuro

En primer lugar, considero oportuno implementar la parte restante del front-end para poder acceder a la funcionalidad implementada en la capa de negocio.

Sería un aspecto muy importante automatizar las pruebas ya que cuando los proyectos escalan no es viable realizar las pruebas de forma manual. Para una empresa sería una pérdida de dinero sustancial ya que tendrían a una serie de trabajadores consumiendo tiempo de su jornada laboral en realizar comprobaciones que podría hacer un computador.

Incluiría la funcionalidad de que cada usuario tenga varias direcciones de entrega y no solo una. Considero que es básico ya que la competencia cuenta con la misma.

Incluiría funcionalidades que diferenciasen este negocio de los demás. Por ejemplo, el sistema de puntuación para los clientes antes mencionando, abogando por el interés de conservar el medio ambiente.

Mejoraría todo lo posible la experiencia de usuario, utilizando estilos limpios y no sobrecargando la interfaz de elementos, o bien, redistribuyéndolos de una forma más eficiente. La comodidad del usuario en la aplicación puede significar un mayor éxito para el negocio.

8. Bibliografía

AEMA. (05 de Diciembre de 2019). *El suelo, la tierra y el cambio climático.* Obtenido de Agencia Europea de Medio Ambiente: https://www.eea.europa.eu/es/senales/senales-2019/articulos/el-suelo-la-tierra-y

García Centeno, M. C., Rodríguez Sanchez, S., Aguirre Arrabal, C., & Inchausti Tabuenca, E. (2022). Impacto de la crisis del gas natural en la inflación española. (A. E. (ASEPUMA), Ed.) *Anales de ASEPUMA*(30), 1-21. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=8692712

Martínez del Olmo, W. (2022). Ideas para combatir el cambio climático. *Revista De La Sociedad Geológica De España, 35*(2), 20-27. doi:https://doi.org/10.55407/rsge.96426

Prades Illanes, E., & Tello Casas, P. (27 de Mayo de 2020). Heterogeneidad en el impacto económico del Covid-19 entre regiones y países del área del euro. *Boletín económico del Banco de España*, 1-18. Obtenido de https://repositorio.bde.es/handle/123456789/12701

VMware Tanzu, "Spring boot3.1.3", Spring Boot, 2023, fecha de consulta verano 2023, en https://spring.io/projects/spring-boot.

Accessing data with mysql (2023) Getting Started | Accessing data with MySQL. Obtenido de: https://spring.io/guides/gs/accessing-data-mysql/ (verano de 2023).

MEDINA, D., "Curso Gratuito spring data JPA", *danielme.com*, 2023, fecha de consulta verano 2023, en https://danielme.com/curso-gratuito-spring-data-jpa/.

Faci, S. (2023) *Acceso a datos*, *apuntes:spring [Acceso a Datos]*. Obtenido de: https://datos.codeandcoke.com/apuntes:spring (Verano de 2023).

Gierke, O.; Turnquist, G.; Bryant, J.; Paluch, M.; Strobl, C.; Darimont, T., Spring Data JPA - Reference Documentation, 2023, fecha de consulta verano 2023, en https://docs.spring.io/spring-data/data-jpa/docs/current/reference/html/#repositories.custom-implementations.

Pivotal, Inc. (2023) *Class representationmodel<T extends representationmodel<? extends T>>*, *RepresentationModel (Spring HATEOAS 2.1.2 API)*. Obtenido de: https://docs.spring.io/spring-hateoas/docs/current/api/org/springframework/hateoas/RepresentationModel.html (verano de 2023).

wayanFounder at Kode Java OrgA programmer, W. (2023) *How do I get servlet request URL information*, *Kode Java*. Obtenido de: https://kodejava.org/how-do-i-get-servlet-request-url-information/ (Verano de 2023).

PRATT, M., "Spring security: Check if a user has a role in Java", *Baeldung*, 2021, fecha de consulta verano 2023, en https://www.baeldung.com/spring-security-check-user-role.

IGNASI ; MATT C; ALEXANDER IVANCHENKO; PATRICK "Spring Security - authorize request for certain URL & HTTP-method using httpsecurity", *Stack Overflow*, 2023, fecha de consulta verano 2023, en https://stackoverflow.com/questions/28907030/spring-security-authorize-request-for-certain-url-http-method-using-httpsecu.

baeldung, "Pessimistic locking in JPA", Baeldung, 2023, fecha de consulta verano 2023, en https://www.baeldung.com/jpa-pessimistic-locking.

Yaremenko, F., “Bloqueo Optimista Y Pesimista en JPA”, HackerNoon, 2022, fecha de consulta verano 2023, en https://hackernoon.com/es/optimista-y-pesimista-bloqueo-en-jpa.

Rodríguez Calle, N., “Pessimistic locking con JPA en spring boot”, Refactorizando, 2023, fecha de consulta verano 2023, en https://refactorizando.com/pessimistic-locking-jpa-spring-boot/.

QUALITRAIN TV, “Hibernate 2018: 15. transacciones, ¿Qué son? bloqueos optimista Y pesimista”, *YouTube*, 2018, YouTube, fecha de consulta verano 2023, en https://www.youtube.com/watch?v=_dSSSPH9Mrk.

RODRÍGUEZ CALLE, N., “Herencia con Hibernate”, *Refactorizando*, 2023, fecha de consulta verano 2023, en https://refactorizando.com/herencia-hibernate/.

Pérez García, A., “Hibernate y el mapeo de la herencia”, Adictos al trabajo, 2018, fecha de consulta verano 2023, en https://www.adictosaltrabajo.com/2007/06/27/hib-inheritance/.

BAELDUNG, “Spring @requestparam annotation”, *Baeldung*, 2022, fecha de consulta verano 2023, en https://www.baeldung.com/spring-request-param.

Howe, R., “Jacksonpolymorphicdeserialization”, GitHub, 2020, fecha de consulta verano 2023, en https://github.com/FasterXML/jackson-docs/wiki/JacksonPolymorphicDeserialization.

Alex, B.; Taylor, L., “Expression-Based Access Control -Part IV. Authorization”, Expression-based access control, fecha de consulta verano 2023, en https://docs.spring.io/spring-security/site/docs/3.0.x/reference/el-access.html.

SPRING (ed.), “Authorize httpservletrequests”, *Authorize HttpServletRequests :: Spring Security*, 2023, fecha de consulta verano 2023, en https://docs.spring.io/spring-security/reference/servlet/authorization/authorize-http-requests.html#match-requests.

Cabello, J., “Spring Security 6 - Spring Boot 3 - JWT. Desde lo Básico”, YouTube, 2023, YouTube, fecha de consulta verano 2023, en https://www.youtube.com/watch?v=5MBYIYSczGg.

Vincent Abba, I., “Tutorial de react router Versión 6 – cómo navegar a otros componentes y configurar UN Enrutador”, freeCodeCamp.org, 2023, freeCodeCamp.org, fecha de consulta verano 2023, en https://www.freecodecamp.org/espanol/news/tutorial-de-react-router-version-6-como-navegar-a-otros-componentes-y-configurar-un-enrutador/.

Fazt Code, “React router DOM V6 - tutorial PRÁCTICO DESDE CERO”, YouTube, 2021, YouTube, fecha de consulta verano 2023, en https://www.youtube.com/watch?v=7xRVnmWcTE8.

Díaz López, J. F., “Entendiendo cors Y Aplicando Soluciones”, Enmilocalfunciona, 2021, Enmilocalfunciona, fecha de consulta verano 2023, en https://www.enmilocalfunciona.io/entendiendo-cors-y-aplicando-soluciones/.

Ceddia, D., “Run code in react before render”, Dave Ceddia, 2020, fecha de consulta verano 2023, en https://daveceddia.com/react-before-render/.

A. Especificación de casos de uso

A.1 Casos de uso – Gestión de accesos

Id + Nombre	Identificador: UseCase-001 Nombre: Registrarse.
Actor Principal	Usuario anónimo.
Descripción	El usuario anónimo se dispone a dar la información requerida por el sistema para ser identificado, de forma que pueda disfrutar de los servicios que ofrece el mismo.
Postcondición	El usuario pasa a estar registrado en el sistema y puede identificarse en el mismo. En el momento de finalizar el registro satisfactoriamente el usuario pasa a estar identificado.
Escenario Principal	1. El usuario selecciona el área donde pone "Cuenta". 2. El sistema mostrará una nueva página donde permite al usuario identificarse, además da como opciones restablecer contraseña o registrarse. 3. El usuario selecciona la opción para poder registrarse. 4. El sistema mostrará una nueva página con un formulario para que el usuario meta la información requerida (nombre, apellidos, DNI, correo electrónico, dirección residencial, número de teléfono y contraseña). 5. El usuario rellena la información requerida en el formulario y selecciona la opción de confirmar. 6. El sistema devolverá al usuario a la página de inicio y el área donde ponía Cuenta ahora indicará que estamos identificados.
Escenarios alternativos	6.b El correo electrónico del usuario ya existe. 6.b.1 El sistema muestra un mensaje en la página del formulario indicando que los datos introducidos se corresponden a un usuario existente. 6.b.2 Se permite al usuario introducir otros datos para registrarse.

Id + Nombre	Identificador: UseCase-002 Nombre: Identificarse.
Actor Principal	Usuario anónimo.
Descripción	El usuario anónimo ha de identificarse introduciendo sus credenciales en el sistema para poder realizar ciertas acciones no accesibles para un usuario anónimo.
Precondición	El usuario debe encontrarse registrado en el sistema. Por supuesto un usuario que se encuentra ya identificado no se puede identificar mientras no cierre sesión.
Postcondición	El usuario pasa a estar identificado, desbloqueando así ciertas acciones que requieren de este proceso
Escenario Principal	1. El usuario selecciona el área donde pone "Cuenta". 2. El sistema mostrará una nueva página donde permite al usuario identificarse, además da como opciones restablecer contraseña o registrarse. 3. El usuario rellena los campos del formulario que corresponden al correo electrónico y a la contraseña, selecciona la opción de continuar. 4. El sistema devolverá al usuario a la página de inicio y el área donde ponía "Cuenta" ahora indicará que estamos identificados.
Escenarios alternativos	4.b El correo del usuario y/o la contraseña no se corresponden con ninguno de los usuarios de sistema. 4.b.1 El sistema se mantiene en la página del formulario mostrando un mensaje que indica que el email y/o la contraseña son erróneos.

Id + Nombre	Identificador: UseCase-003 Nombre: Cerrar sesión.
Actor Principal	Usuario identificado, administrador.
Descripción	El usuario identificado y el administrador pueden querer desear cerrar su sesión para que cualquier otra persona que utilice el dispositivo desde el cual se ha iniciado sesión

	no pueda realizar acciones indebidas con una cuenta que no es la suya.
Postcondición	El usuario no tiene acceso a acciones propias de un usuario identificado o de un administrador, como ver pedidos, datos de cuenta…
Escenario Principal	1. El usuario selecciona la opción de cerrar sesión desde el menú propio de la cuenta de usuario. 2. El sistema vuelve a la página de inicio habiendo ya salido el usuario de la sesión.

A.2 Casos de uso – Gestión de productos

Id + Nombre	Identificador: UseCase-006 Nombre: Buscar producto.
Actor Principal	Administrador.
Descripción	El administrador puede ver todos los productos e incluso filtrar los mismos según algún criterio.
Postcondición	Se mostrarán todos los productos si no se aplica ningún filtro, o aquellos que cumplan el criterio del filtro. En cualquiera de los dos casos si no hay productos en el sistema se muestra un mensaje indicando que no se encuentra ningún producto.
Escenario Principal	1. El administrador desde la página inicial selecciona el área que engloba las operaciones de gestión de productos en venta. 2. El sistema muestra una nueva página con varias opciones, la de crear un nuevo producto, o buscar producto entre los existentes. 3. El administrador selecciona la opción de buscar producto. 4. El sistema muestra una nueva página con el listado de todos los productos (o un mensaje que indica que no hay productos en el sistema si así fuera). Por cada fila de la lista se muestra su nombre, el precio, si se encuentra activo, y algunos detalles; además para cada producto se tiene la posibilidad de ver toda su información seleccionando la opción correspondiente.

Id + Nombre	Identificador: UseCase-007 Nombre: Consultar detalles de producto.
Actor Principal	Administrador.
Descripción	El administrador puede visualizar los detalles de un determinado producto.
Postcondición	Se muestran los detalles del producto seleccionado entre los demás productos tras su búsqueda en el sistema.
Escenario Principal	1. Incluye el caso de uso “Buscar producto”. 2. El administrador selecciona la opción para consultar uno de los productos listados. 3. El sistema muestra en una nueva página toda la información del producto (la URL de la imagen, el nombre, el precio base, el precio de instalación, el porcentaje de oferta, el precio con la oferta aplicada, las piezas que lo componen, si es una novedad, si se encuentra activo, si es una pieza, el stock que tiene, los detalles, y las características técnicas). Además, se muestra tres opciones, una para modificar el producto, otra para eliminarlo y otra para añadir una pieza al producto.

Id + Nombre	Identificador: UseCase-008 Nombre: Borrar producto.
Actor Principal	Administrador.
Descripción	El administrador elimina el producto seleccionado.
Postcondición	Se elimina el producto del sistema mostrando un mensaje que confirma el éxito de la operación. El producto no aparecerá en futuras búsquedas en el sistema para el resto de los usuarios, para el administrador aparecerá como inactivo.

Escenario Principal	1. Incluye el caso de uso “Consultar detalles de producto”. 2. El administrador selecciona la opción para eliminar el producto. 3. El sistema borra/inhabilita el producto y devuelve la página de búsqueda de productos.

Id + Nombre	Identificador: UseCase-009 Nombre: Modificar producto.
Actor Principal	Administrador.
Descripción	El administrador modifica uno o varios de los datos que conforman un producto ya existente en el sistema.
Postcondición	Se modifican los datos del producto seleccionado mostrando un mensaje que confirma el éxito de la operación, así como los datos actualizados en la página de detalles del producto.
Escenario Principal	1. Incluye “Consultar detalles de producto”. 2. El administrador selecciona la opción correspondiente para modificar el producto. 3. El sistema muestra una nueva página con un formulario para introducir la nueva información del producto (el nombre del producto, su precio base, su precio de instalación, un porcentaje de descuento en caso de haber oferta, si se trata o no de una novedad, los detalles del producto, características técnicas, el stock disponible en este momento, si debe estar activo, y la URL de una imagen), alguno de sus campos o todos ellos. Además, se muestra la opción para confirmar la modificación de los datos del producto. 4. El administrador introduce la nueva información en los campos a modificar del producto y selecciona la opción para modificar los datos. 5. El sistema muestra un mensaje que confirma la modificación del producto, así como los detalles del producto con los datos ya modificados.
Escenarios Alternativos	5.b Se introduce nueva información incluyendo como nombre el de uno de los productos ya existentes en el sistema. 5.b.1 El sistema muestra un mensaje que indica que no se puede modificar el nombre de un producto introduciendo el de uno ya existente en el sistema. 5.b.2 El sistema muestra de nuevo la página con el formulario para realizar la modificación de los datos.

Id + Nombre	Identificador: UseCase-010 Nombre: Añadir pieza a producto.
Actor Principal	Administrador.
Descripción	El administrador crea, introduciendo la información necesaria, una nueva pieza de un producto, que es igualmente un producto, y que será accesible por usuarios anónimos e identificados para su consulta y compra.
Postcondición	Se ha creado una nueva pieza de un producto (que es un producto) del cual se ha generado un identificador y que puede ser consultado junto al resto de productos previamente creados. Se muestra un mensaje que confirma su creación.
Escenario Principal	1. Incluye “Consultar detalles de producto”. 2. El administrador selecciona la opción correspondiente para crear una nueva pieza para el producto. 3. El sistema muestra una nueva página con un formulario para introducir la información propia de la pieza (el nombre, su precio base, un porcentaje de descuento en caso de haber oferta, si se trata o no de una novedad, si está activo, los detalles de la pieza, características técnicas, el stock disponible en este momento, el precio de la instalación, la URL de la imagen). Además, se muestra una opción para confirmar los datos. 4. El administrador rellena los datos del formulario y selecciona la opción que permite crear definitivamente la pieza. 5. El sistema muestra un mensaje que confirma la creación de la pieza mostrando además su identificador único.

Escenarios Alternativos	6.b El nombre de la pieza introducida coincide con el de otra pieza existente. 6.b.1 El sistema muestra un mensaje que indica que no puede haber dos piezas que utilicen el mismo nombre (a fin de no introducir piezas duplicadas). 6.b.2 Se impide la creación de la pieza y se devuelve al administrador a la página con el formulario para crear la pieza.

Id + Nombre	Identificador: UseCase-011 Nombre: Ver información de la página de inicio.
Actor Principal	Usuario anónimo, usuario identificado, administrador.
Descripción	Cualquier usuario que acceda al sistema podrá observar la información que presenta los productos e ideales de la empresa, así como información de contacto.
Postcondición	El usuario puede visualizar la información base del sistema.
Escenario Principal	1. El usuario selecciona la opción correspondiente para poder visualizar la página de inicio, en este caso el nombre de la empresa. 2. El sistema devolverá al usuario a la página de inicio, esta contiene información sobre los principales atractivos de ser energéticamente autosuficiente, así como información de contacto con el servicio de atención al cliente.

Id + Nombre	Identificador: UseCase-012 Nombre: Consultar productos
Actor Principal	Usuario anónimo, usuario identificado.
Descripción	El usuario que acceda al sistema, tanto identificado como sin identificar, puede ver todos los productos que se ofrecen en el mismo.
Postcondición	El usuario puede visualizar los productos que se encuentran disponibles en el sistema en caso de haber productos en el mismo.
Escenario Principal	1. El usuario accede al área de la página que se corresponde con la consulta de productos. 2. El sistema devolverá al usuario una página con los productos que ofrece la aplicación, o ningún artículo y un mensaje explicativo en caso de no haber productos. Por cada artículo se mostrará una imagen, el nombre, el porcentaje de oferta, el precio base y el final aplicando la oferta, en caso de ser un artículo en oferta, y una imagen, el nombre, y el precio en caso de ser una novedad, o cualquier otro artículo que no está en oferta o es novedad.

Id + Nombre	Identificador: UseCase-013 Nombre: Consultar ofertas
Actor Principal	Usuario anónimo, usuario identificado.
Evento de activación	El caso de uso comienza cuando se cumple la condición de que el usuario desee ver dentro del conjunto de todos los productos del sistema solo los que están en oferta del caso de uso “Consultar productos”.
Descripción	El usuario que acceda al sistema, tanto identificado como sin identificar, puede ver los productos que se ofrecen en el mismo y que disponen de un determinado porcentaje de oferta.
Postcondición	El usuario puede visualizar los productos que se encuentran en oferta en caso de encontrarse alguno en esta condición.
Escenario Principal	1. El usuario selecciona el área que indica ofertas. 2. El sistema devolverá al usuario una página con los productos que se encuentren en oferta, o ningún artículo y un mensaje explicativo en caso de no haber productos. Por cada artículo se mostrará una imagen, el nombre, el porcentaje de oferta, el precio base y el final.

Id + Nombre	Identificador: UseCase-014 Nombre: Consultar novedades

Actor Principal	Usuario anónimo, usuario identificado.
Evento de activación	El caso de uso comienza cuando se cumple la condición de que el usuario desee ver dentro del conjunto de todos los productos del sistema solo los que son novedad del caso de uso “Consultar productos”.
Descripción	El usuario que acceda al sistema, tanto identificado como sin identificar, puede ver los productos que se ofrecen en el mismo y que se han añadido recientemente como novedosos.
Postcondición	El usuario puede visualizar los productos que novedad en caso de ser alguno marcado como tal.
Escenario Principal	1. El usuario selecciona el área que indica novedades. 2. El sistema devolverá al usuario una página con los productos que son novedad, o ningún artículo y un mensaje explicativo en caso de no haber productos. Por cada artículo se mostrará una imagen, el nombre, y el precio.

A.3 Casos de uso – Gestión de cuentas

Id + Nombre	Identificador: UseCase-016 Nombre: Consultar datos de cuenta de cliente.
Actor Principal	Administrador.
Descripción	El administrador visualiza los datos de cuenta/perfil del usuario del cual se ha introducido el correo electrónico.
Postcondición	Se muestran los datos de la cuenta de usuario buscada en el sistema.
Escenario Principal	1. El administrador desde la página inicial selecciona la opción que corresponde con la gestión de cuentas. 2. El sistema muestra una nueva página con un formulario para introducir el correo electrónico de la cuenta que se desea visualizar, así como, una opción para continuar con el proceso. 3. El administrador rellena los datos del formulario y selecciona la opción para continuar. 4. El sistema muestra una nueva página en la que se pueden ver los datos del usuario propietario de la cuenta (el nombre, los apellidos, DNI, correo electrónico, dirección residencial, número de teléfono y si es un usuario activo), además de varias opciones sobre la cuenta, como borrarla, y modificar sus datos.
Escenarios Alternativos	6.b El email introducido no se corresponde con el de ningún usuario registrado en el sistema. 6.b.1 El sistema muestra un mensaje indicando que la cuenta especificada no se encuentra en el sistema.

Id + Nombre	Identificador: UseCase-017 Nombre: Borrar cuenta.
Actor Principal	Administrador.
Descripción	El administrador borra la cuenta de un usuario, el cual no podrá acceder al sistema para realizar ninguna compra.
Postcondición	Se elimina/inhabilita la cuenta de forma que no es accesible para usuarios que no sean administradores.
Escenario Principal	1. Incluye “Consultar datos de cuenta de cliente”. 2. El administrador selecciona la opción que corresponde con el borrado de la cuenta. 3. El sistema borra/inhabilita la cuenta y devuelve al administrador a la página previa donde se ingresó el correo electrónico identificativo de la cuenta actualmente borrada.

Id + Nombre	Identificador: UseCase-018 Nombre: Modificar datos de cuenta.
Actor Principal	Administrador.
Descripción	El administrador modifica los datos de perfil de la cuenta especificada.
Postcondición	Se modifican los datos de perfil de la cuenta y se muestran actualizados en la página correspondiente.
Escenario Principal	1. Incluye el caso de uso “Consultar datos de cuenta de cliente”. 2. El administrador selecciona la opción que se corresponde con la modificación de los datos. 3. El sistema muestra una nueva página en la que se encuentra un formulario para sustituir parte de la información que se requiera de los datos de perfil del usuario (no será posible modificar el email y el DNI), no es necesario volver a ingresar toda la información correspondiente en cada campo, solo la información que se quiera modificar. 4. El administrador introduce los datos que quiere cambiar dentro del formulario y selecciona la opción para guardarlos. 5. El sistema devuelve la página donde se encuentran los datos de perfil con los nuevos valores actualizados.

Id + Nombre	Identificador: UseCase-020 Nombre: Cambiar información de cuenta.
Actor Principal	Usuario identificado.
Descripción	El usuario modifica los datos personales asociados a su cuenta.
Postcondición	Se modifican satisfactoriamente los datos de perfil del usuario identificado.
Escenario Principal	1. Incluye “Consultar datos de cuenta”. 2. El usuario selecciona la opción que se corresponde con la modificación de los datos de perfil. 3. El sistema muestra un formulario con algunos de los campos que componen la información personal del usuario a fin de ser modificada (no se muestra ni el email, ni el DNI), así como un botón para confirmar dicha información. 4. El usuario cumplimenta el formulario, de forma que los campos que se cumplimentan son modificados y los demás mantienen su valor. El usuario selecciona la opción que permite guardar los cambios introducidos. 5. El sistema actualiza la información y vuelve a la página de perfil.

Id + Nombre	Identificador: UseCase-021 Nombre: Consultar métodos de pago.
Actor Principal	Usuario identificado.
Descripción	El usuario visualiza, como parte de su cuenta, los diferentes métodos de pago que tiene almacenados. Almacenar métodos de pago, como la información de una tarjeta bancaria, facilita el proceso de compra del usuario.
Postcondición	Se muestran los diferentes métodos de pago almacenados en el sistema, tanto tarjetas bancarias, como números de teléfono móvil para Bizum.
Escenario Principal	1. El usuario selecciona el área con donde se encuentra el menú del cliente. 2. El sistema muestra en el desplegable varias opciones entre ellas los métodos de pago. 3. El usuario selecciona en el desplegable la opción que se corresponde con los métodos de pago. 4. El sistema muestra un listado con los métodos de pago del usuario. Por cada método de pago se mostrará un nombre que identifica el método de pago, que le resulte familiar al usuario y que le diferencie del resto, para la tarjeta el nombre del titular, la fecha de vencimiento, y el número de tarjeta, para Bizum el teléfono móvil. Se ofrece la opción de modificar y eliminar cada método de pago listado, así como, la de añadir un nuevo método de pago. Si no hay métodos de pago almacenados se muestra un mensaje informativo.

Id + Nombre	Identificador: UseCase-022 Nombre: Añadir método de pago.
Actor Principal	Usuario identificado.
Descripción	El usuario añade a sus métodos de pago uno nuevo, ya sea una tarjeta bancaria o un Bizum, para poder realizar las compras más cómodamente.
Postcondición	Se añade un nuevo método de pago a los ya existentes, siendo posible ver este entre los demás al realizar su consulta.
Escenario Principal	1. Incluye “Consultar métodos de pago”. 2. El usuario selecciona la opción que se corresponde con la adición de un nuevo método de pago. 3. El sistema muestra una nueva página en la que se encuentra un formulario donde habrá dos opciones mutuamente exclusivas, según las cuales, tras la selección de una de ellas, se mostrarán los campos correspondientes al tipo de método de pago seleccionado. Si se selecciona tarjeta bancaria se mostrarán en el formulario los campos nombre de método de pago, nombre de titular, número de tarjeta, mes y año de vencimiento, y CVC. Si se selecciona Bizum como método de pago se mostrarán los campos nombre de método de pago, y número de teléfono de Bizum. 4. El usuario introduce los datos dentro del formulario y selecciona la opción para guardarlos. 5. El sistema devuelve la página donde se encuentran los diferentes métodos de pago del usuario, encontrándose ya disponible el método de pago introducido.
Escenarios Alternativos	5.b Cierta información introducida del método de pago (nombre del método de pago, número de teléfono o número de tarjeta según corresponda) es igual a la de otro método de pago ya existente para el usuario. 5.b.1 El sistema devuelve la página donde se encuentra el formulario para introducir el nuevo método de pago. 5.b.2 Se muestra un mensaje indicando que no se puede añadir un método de pago con los mismos datos de uno ya existente para el usuario.

Id + Nombre	Identificador: UseCase-023 Nombre: Modificar método de pago
Actor Principal	Usuario identificado.
Descripción	El usuario modifica los datos de un método de pago seleccionado.
Postcondición	Se modifican los datos del método de pago seleccionado pudiéndose confirmar al consultar el mismo. Igualmente, se muestra un mensaje confirmando el éxito de la operación.
Escenario Principal	1. Incluye “Consultar métodos de pago”. 2. El usuario selecciona la opción que se corresponde con la modificación de un método de pago existente. 3. El sistema muestra una nueva página en la que se encuentra un formulario que se corresponde con el tipo de método de pago previo, con los campos propios del tipo para que sean modificados, si se trata de un Bizum se permite modificar solo el nombre del método de pago, si se trata de una tarjeta se permite cambiar el nombre del titular, el nombre del método de pago, y el mes y año de vencimiento. Los campos que no sean modificados mantendrán el valor previo. Se ofrece la opción para guardar los cambios. 4. El usuario introduce los datos dentro del formulario y selecciona la opción para guardarlos. 5. El sistema devuelve la página donde se encuentran los diferentes métodos de pago del usuario, encontrándose el método de pago con los datos actualizados.
Escenarios Alternativos	5.b Cierta información introducida del método de pago (nombre del método de pago) es igual a la de otro método de pago ya existente para el usuario. 5.b.1 El sistema devuelve la página donde se encuentra el formulario para

	modificar el método de pago. 5.b.2 Se muestra un mensaje indicando que no se puede modificar un método de pago con los mismos datos de uno ya existente para el usuario.

Id + Nombre	Identificador: UseCase-024 Nombre: Eliminar método de pago.
Actor Principal	Usuario identificado.
Descripción	El usuario elimina el método de pago de forma que al consultar los métodos de pago este no aparece entre ellos.
Postcondición	Se elimina/inhabilita el método de pago seleccionado, de manera que, al consultar los métodos de pago, este no se encuentra entre ellos. Está disponible para los administradores.
Escenario Principal	1. Incluye “Consultar métodos de pago”. 2. El usuario selecciona la opción que se corresponde con la eliminación de un método de pago existente. 3. El sistema devuelve la página donde se encuentran los diferentes métodos de pago del usuario. El método de pago eliminado no se encuentra entre los demás.

A.4 Casos de uso – Gestión de cesta

Id + Nombre	Identificador: UseCase-027 Nombre: Eliminar producto de la cesta.
Actor Principal	Usuario identificado.
Descripción	El usuario elimina un producto de la cesta el cual desaparece de la misma mostrando el contenido previo sin este producto.
Precondición	Ha de haber algún producto en la cesta que pueda ser eliminado.
Postcondición	Se muestran todos los artículos añadidos a la cesta previamente menos el que se ha eliminado de la misma y se recalcula el coste total de la cesta.
Escenario Principal	1. Incluye “Consultar contenido de la cesta”. 2. El usuario selecciona para cada producto que tiene en la cesta, y que quiere que deje de estarlo, la opción de eliminar. 3. El sistema hace desaparecer de la cesta los productos previamente eliminados actualizando el coste total del contenido de la cesta.

Id + Nombre	Identificador: UseCase-028 Nombre: Modificar contenido de la cesta.
Actor Principal	Usuario identificado.
Descripción	El usuario modifica los parámetros de los artículos añadidos a la cesta, como el número de ítems que se desean para cada artículo y si se desea que sean instalados en su destino. El coste total de la cesta varía en función de estos cambios.
Precondición	Ha de haber algún producto en la cesta para que sus parámetros puedan ser modificados.
Postcondición	Los valores de los parámetros varían según el cambio introducido por el usuario y el coste de la cesta cambia acorde con las modificaciones.
Escenario Principal	1. Incluye “Consultar contenido de la cesta”. 2. El usuario modifica para cada producto el número de unidades que quiere del mismo, así como si quiere o no su instalación. 3. El sistema recalcula el coste del conjunto de toda la cesta y muestra los valores introducidos en la modificación.

A.5 Casos de uso – Gestión de pedidos

Id + Nombre	Identificador: UseCase-031 Nombre: Filtrar pedidos.
Actor Principal	Administrador.
Evento de activación	El caso de uso comienza cuando se cumple la condición de que el administrador desee ver dentro de un conjunto pedidos solo un subconjunto de los mismos, a partir de uno o varios criterios según los cuales se hace el filtrado. Se busca un subconjunto del total de pedidos obtenidos del caso de uso “Consultar pedidos”.
Descripción	El administrador obtiene un subconjunto de pedidos del total filtrando por uno o varios criterios los mismos.
Postcondición	El sistema ha de mostrar los pedidos, según los cuales haya coincidencia con los criterios de filtrado aplicados, en forma de lista. En caso de que no haya ningún pedido coincidente se mostrará un mensaje informando de lo sucedido.
Escenario Principal	1. El administrador dentro de la pantalla donde se muestran pedidos introduce en los campos destinados a recibir los criterios de filtrado los valores deseados (aquellos campos que se desea que se apliquen), y selecciona la opción de filtrar. 2. El sistema muestra una nueva página en la que se listan los pedidos filtrados por uno o varios criterios simultáneamente. Por cada pedido se muestra el identificador, los ítems pedidos para cada artículo, los nombres de los artículos, la fecha, el email y el nombre del comprador. Además, es posible ver en más detalle cada pedido seleccionando la opción correspondiente. En caso de no haber coincidencias se muestra un mensaje informando de lo sucedido.

Id + Nombre	Identificador: UseCase-034 Nombre: Modificar estado de pedido.
Actor Principal	Administrador.
Descripción	El administrador observa todos los pedidos y selecciona uno de ellos para ver información más detallada, desde ahí puede modificar el estado de pedido, cambiando el mismo entre los valores de “en preparación”, “enviado” y “entregado”. El estado de anulado es especial y solo se puede pasar a este estado anulando el pedido, lo cual es inamovible.
Postcondición	El sistema muestra la página con los detalles del pedido actualizados.
Escenario Principal	1. Incluye “Consultar detalles de pedido”. 2. El administrador selecciona la opción que se corresponde con la modificación del estado. 3. El sistema muestra una nueva página con tres opciones de estado, en preparación, enviado y entregado. 4. El administrador selecciona la opción de estado enviado que indica que el pedido le ha recogido la empresa encargada del transporte del pedido y que ha sido enviado, de entregado en caso de que la empresa de transporte le indique al administrador que se ha hecho la entrega, o en preparación en caso de que haya habido algún error en el envío del pedido y se requiera enviar más productos como parte de este, o que aún no está preparado para ser enviado. 5. El sistema cambia el estado del pedido y devuelve la página en la que se muestran los detalles del pedido incluyendo el nuevo estado.

Id + Nombre	Identificador: UseCase-035 Nombre: Consultar historial de pedidos.
Actor Principal	Usuario identificado.

Descripción	El usuario, desde su cuenta, consulta todos los pedidos para conocer sus detalles.
Postcondición	El sistema muestra todos los pedidos del usuario identificado, o un mensaje informativo en caso de que el usuario no haya realizado ningún pedido desde su cuenta.
Escenario Principal	1. El usuario selecciona el área de la página con el menú del cliente. 2. El sistema muestra en el desplegable varias opciones entre ellas los pedidos. 3. El usuario selecciona en el desplegable la opción que se corresponde con la consulta de pedidos. 4. El sistema da como opciones ver los pedidos actuales (cuya entrega no se ha efectuado y no han sido anulados), o ver el historial de pedidos (todos los pedidos realizados por la cuenta). 5. El usuario selecciona la opción que corresponde a todos los pedidos. 6. El sistema muestra todos los pedidos ordenados de más actual a menos, por cada pedido se muestra la información de cada artículo (nombre, número de ítems, si se hace instalación, el precio de la instalación, y el precio del artículo), el coste total del pedido, la dirección de entrega, la fecha en la que se ha realizado el pedido, el método de pago usado (Bizum o tarjeta bancaria), algunos de los datos como el número de tarjeta o número de teléfono según corresponda, y el estado en el que se encuentra el pedido, ya sea pendiente de pago, en preparación, enviado, entregado, o anulado.

Id + Nombre	Identificador: UseCase-036 Nombre: Consultar pedidos pendientes.
Actor Principal	Usuario identificado.
Descripción	El usuario, desde su cuenta, consulta los pedidos que aún no han sido entregados o anulados para conocer sus detalles, así como, el estado en el que se encuentran en relación con el proceso de entrega.
Postcondición	El sistema muestra los pedidos realizados cuyo estado no es entregado o anulado, o un mensaje informativo en caso de que el usuario no haya realizado ningún pedido desde su cuenta.
Escenario Principal	1. El usuario selecciona el área de la página con el menú del cliente. 2. El sistema muestra en el desplegable varias opciones entre ellas los pedidos. 3. El usuario selecciona en el desplegable la opción que se corresponde con la consulta de pedidos. 4. El sistema da como opciones ver los pedidos actuales (cuya entrega no se ha efectuado, y no han sido anulados), o ver el historial de pedidos (todos los pedidos realizados por la cuenta). 5. El usuario selecciona la opción que corresponde a los pedidos actuales. 6. El sistema muestra solo los pedidos que se encuentran pendientes de pago o que están pendientes de entrega, ordenados de más actual a menos. Por cada pedido se muestra la información de cada artículo (nombre, número de ítems, si se hace instalación, el precio de la instalación, y el precio del artículo), el coste total del pedido, la dirección de entrega, la fecha en la que se ha realizado el pedido, el método de pago usado (Bizum o tarjeta bancaria), algunos datos del método de pago como el número de tarjeta o número de teléfono según corresponda, y el estado en el que se encuentra el pedido, siendo en este caso pendientes de pago, en preparación para envíos que aún no han salido de las instalaciones, y enviado para los que están de camino hacia su destino final.

B. Modelo de base de datos

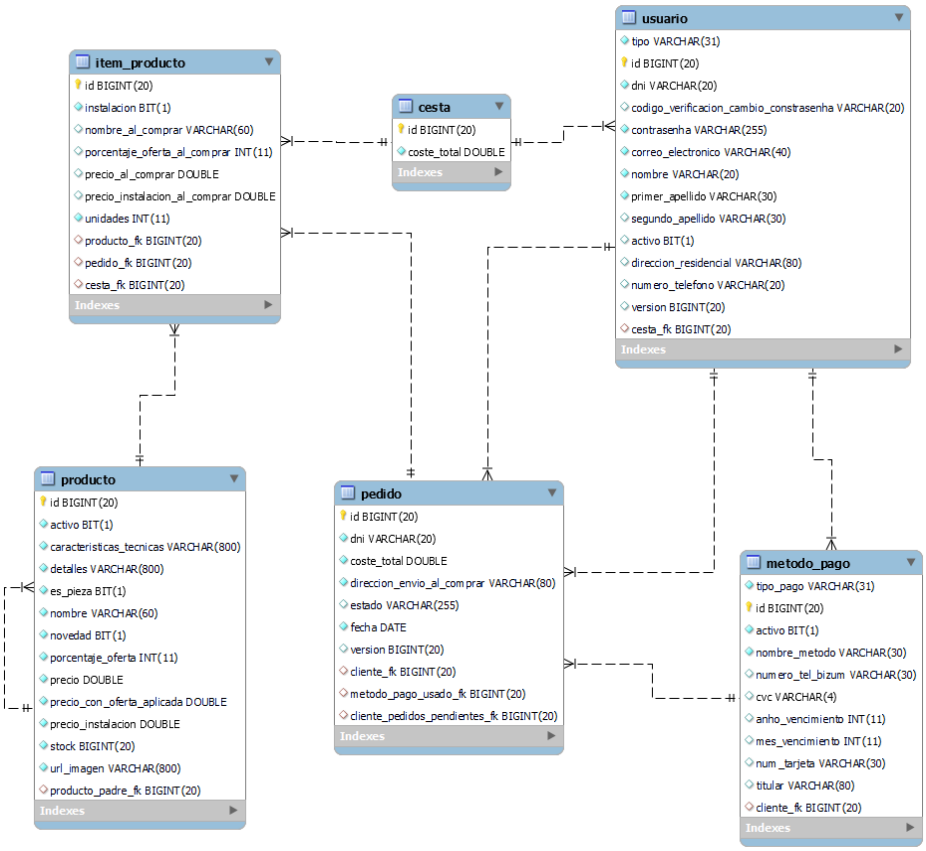

Figura 44: Modelo de base de datos