Memoria práctica 4

Participantes grupo B4

Iker Gálvez Castillo
Álvaro Yuste Moreno
Rocío Gómez Mancebo
Alberto Romero Cantos
Marta González Palmero

Índice

1. URL de la aplicación..... 1
1.1 Usuario de prueba..... 2
2. Tecnologías utilizadas..... 2
3. Requisitos implementados..... 4
3.1. Requisitos funcionales..... 4
3.2 Requisitos no funcionales..... 6
4. Arquitectura de la aplicación..... 6
5. Base de datos..... 6
6. Instrucciones de instalación..... 9
7. Replanteamientos de la API REST..... 9
7.1 Cambios en la base de datos..... 9
8. Servicios externos..... 13
8.1 OpenStreetMap..... 13
8.2 Subir fotos con Cloudinary..... 14
8.3 Pago a través de PayPal..... 14
8.4 Login con Google OAuth..... 15
8.5 Cálculo de la huella de carbono..... 15
8.6 Vercel..... 16
9. Nuevas funcionalidades..... 16
9.1 Cancelar y modificar subasta..... 16
9.2 Mapa en Home y filtro proximidad..... 16
9.3 Carrusel de fotos..... 16
9.4 Cambios en pujas y comentarios..... 16
9.5 Tasa de envío en función de la huella de carbono..... 17
9.6 Subastas desiertas..... 17
9.7 Productos pendientes de pago..... 17

1. URL de la aplicación

Aquí se adjunta la URL pública para poder acceder a la aplicación:

https://frontend-rastro.vercel.app/

1.1 Usuario de prueba

Para facilitar la vista de las distintas acciones que se pueden realizar en la aplicación se ha introducido en la base de datos un usuario con correo pruebaparaingweb@gmail.com que ya tiene un producto comprado, un producto pendiente para comprar y pujas en algunos productos . Además para comprobar las funcionalidades correspondientes a usuarios recién registrado como la solicitud de introducir la ubicación para poder ver el mapa principal, se recomienda iniciar sesión con otro correo distinto que no esté registrado en la base de datos.

(Algunas acciones como crear un producto con varias fotos o cargar la vista de pago pueden llevar unos segundos, pero todo lo incluido en esta memoria funciona correctamente)

2. Tecnologías utilizadas

Hemos utilizado prácticamente las mismas tecnologías que en la entrega anterior:

React.js
Es una biblioteca de JavaScript que nos permite construir el frontend de nuestra aplicación y facilita la creación de componentes reutilizables para tener una estructura más ordenada y reutilizable.

React Leaflet
Es la integración entre React y Leaflet, una biblioteca de mapas que permite mostrar los mapas interactivos que utilizamos en nuestra aplicación.

Node.js
Es un entorno de ejecución basado en JavaScript para la capa del servidor y lo hemos utilizado para implementar nuestro backend.

Express.js
Es un framework de Node.js que permite desarrollar aplicaciones web y APIs y que hemos utilizado en el backend para gestionar las peticiones HTTP de manera eficiente.

Morgan
Es una biblioteca para Node.js que nos da información sobre las peticiones HTTP que realizamos, lo que nos facilita la depuración.

MongoDB
Es una base de datos no relacional que utilizamos para almacenar y gestionar nuestros datos. La hemos elegido porque permite hacer consultas de manera eficiente y por su facilidad para crear una base de datos en la nube.

Mongoose
Es una biblioteca de modelado de objetos orientada a objetos para Node.js que nos permite interactuar con nuestra base de datos de MongoDB.

Nodemon
Es una herramienta que se puede usar en Node.js que nos ha permitido acelerar el proceso de desarrollo porque relanza la aplicación cada vez que guardamos algún cambio sobre algún fichero de ella.

Streamify
Es una biblioteca de Node.js que simplifica el proceso de subir archivos a servicios en la nube como Cloudinary. Proporciona una forma conveniente de convertir un búfer de archivos en un flujo de lectura asíncrono que se puede canalizar directamente al método de carga del servicio en la nube.

Además, hemos utilizado también una serie de servicios externos que facilitan la gestión de las imágenes, las ubicaciones, las sesiones, el pago y el despliegue de la aplicación:

- **Cloudinary**
  Utilizado para el almacenamiento de las imágenes, de forma que nosotros solo guardamos una URL y no las imágenes al completo.

- **OpenStreetMaps**
  Utilizado para mostrar las ubicaciones de forma visual mediante mapas interactivos.

- **Google OAuth 2.0**
  Utilizado para la gestión de las sesiones. De esta forma, los usuarios podrán registrarse con su dirección de correo electrónico de Google y hacer la autenticación con ella de manera segura.

- **PayPal**
  Utilizado para gestionar los pagos realizados en la plataforma con lo referente a tarjetas de crédito y divisas.

- **Vercel**
  Utilizado para desplegar la aplicación en la nube y poder acceder a ella en cualquier momento y desde cualquier lugar con a través de una URL

Detallaremos más estos servicios en el apartado 8 de este documento, haciendo especial énfasis en los que son nuevos de esta entrega: Google OAuth 2.0, PayPal y Vercel.

3. Requisitos implementados

3.1. Requisitos funcionales

Login:
RF 1.1. El usuario podrá iniciar sesión a través de Google (implementado con OAuth 2.0).
RF 1.2. Cuando se inicia sesión, dura 1 hora, cuando expira se mostrará un mensaje y se cerrará automáticamente la sesión.

Página principal:
RF 2.1. El usuario podrá ver un listado de todos los productos que están en venta sin haber iniciado sesión
RF 2.2. Un usuario que no ha iniciado sesión no puede entrar la subasta/ los detalles de un producto específico
RF 2.3. Un usuario que ha iniciado sesión puede entrar en la subasta/ los detalles de un producto desde el listado general.
RF 2.4. El usuario podrá ver en un mapa todos los productos que estén en venta, solo si el usuario tiene su ubicación indicada en su perfil (implementado con OpenStreetMaps).
RF 2.5. El usuario podrá filtrar los productos que desea ver en función de 3 criterios tanto de manera individual como simultánea: proximidad a su ubicación, precio máximo y título o descripción.

Perfil de mi usuario:
RF 3.1. El usuario puede acceder a su perfil desde cualquier vista de la aplicación.
RF 3.2. El usuario puede modificar los datos de su perfil, incluida su foto.
RF 3.3. Al modificar la ubicación del usuario se modificará la latitud y longitud del mismo.
RF 3.4. El usuario puede acceder a un listado de los productos que ha vendido.
RF 3.5. El usuario puede acceder a un listado de los productos que ha puesto a la venta y aún no han sido vendidos.
RF 3.6. El usuario puede acceder a un listado de los productos por los que ha pujado.
RF 3.7. El usuario puede acceder a un listado de los productos que ha comprado.
RF 3.8. El usuario podrá acceder a un listado de los productos que ha ganado la subasta pero aún no ha realizado el pago.

Perfil de otro usuario:
RF 4.1 El usuario podrá acceder al perfil de otro usuario desde un producto, comentario o puja publicada por este.
RF 4.2 El usuario podrá ver la información de otro usuario en su perfil (valoración, su nombre de usuario, número de contacto y su ubicación).
RF 4.3 El usuario podrá visualizar los productos que otro usuario tiene en venta.
RF 4.4 El usuario podrá visualizar los productos que otro usuario ha vendido.

Detalles de producto / subasta:
RF 5.1. Un usuario registrado podrá acceder a un producto y ver toda la información que ofrece: título, descripción, si está vendido o no, la puja más alta hasta el momento y la fecha de cierre.
RF 5.2. El usuario puede ver la ubicación concreta de un producto en el mapa.

RF 5.3. El usuario podrá acceder al perfil del vendedor del producto.
RF 5.4. El usuario podrá modificar la descripción, y la ubicación de su producto en cualquier momento.
RF 5.5. Al actualizar la ubicación de un producto se cambia la latitud y longitud del mismo.
RF 5.6. El usuario podrá modificar la fecha de cierre de la subasta y el precio inicial de esta hasta que se haga la primera puja.
RF 5.7. El usuario podrá cancelar la subasta siempre que no haya pujas hechas.
RF 5.8. El usuario podrá ver un carrusel de fotos de un producto.
RF 5.9. El usuario podrá añadir una foto al carrusel de los productos que haya subastado.
RF 5.10. El usuario podrá escribir un comentario en un producto ajeno.
RF 5.11. El usuario podrá responder a los comentarios que se hayan hecho sobre su producto.
RF 5.12. El usuario podrá borrar los comentarios que haya publicado.
RF 5.13. El usuario podrá pujar por un producto que no le pertenezca.
RF 5.14. El usuario podrá eliminar una puja que haya realizado sobre un producto.
RF 5.15. El usuario podrá cerrar la subasta de sus productos cuando quiera, en ese momento se asignará la venta al mayor pujador. (Este requisito se ha realizado para facilitar la vista de situaciones en las que habría terminado una puja por cumplimiento de la fecha límite)
RF 5.16. La subasta se cerrará automáticamente cuando se cumpla la fecha de cierre indicada en ese producto.

Subastar un producto:
RF 6.1. Un usuario podrá subastar un producto introduciendo la información principal de este (Título, precio inicial, fecha de cierre, foto/s, descripción).

Compra de un producto:
RF 7.1. El usuario con la mayor puja de la subasta podrá comprar el producto cuando se haya cerrado la subasta.
RF 7.2. El usuario podrá elegir cómo será el medio de transporte del envío de su producto.
RF 7.3. El usuario podrá realizar el pago a través de PayPal con la cuenta de sandbox cuyos credenciales aparecen en la vista del pago.
RF 7.4. El usuario podrá ver los gastos extra que se aplican en función de la huella de carbono.
RF 7.5. El usuario podrá ver un resumen del pago.

Valoraciones:
RF 8.1. Una vez un usuario compre un producto, puede valorar al vendedor (según la fiabilidad y la calidad del producto) en la vista de detalles del producto comprado.
RF 8.2. Cuando un usuario venda un producto, puede valorar al comprador en la vista de detalles de ese mismo producto vendido.

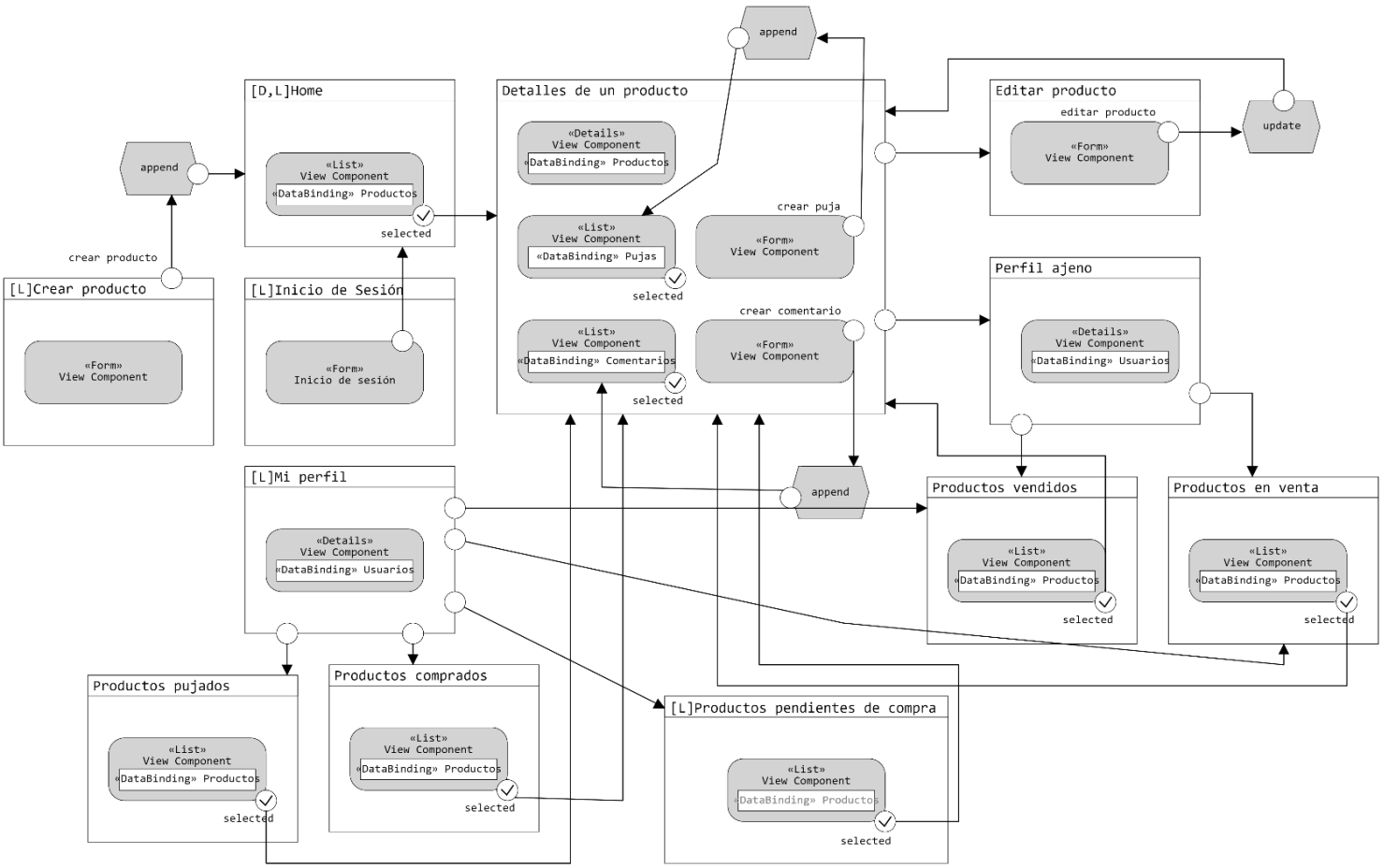
Logout:
RF 9.1 El usuario podrá cerrar sesión desde cualquier vista de la aplicación.

3.2 Requisitos no funcionales

RNF 1. La aplicación está alojada en el cloud Vercel. Ambos, el backend y el frontend están desplegados por separado en Vercel.
RNF 2. La aplicación permite el pago de los productos con el servicio externo Paypal.
RNF 3. Los usuarios podrán iniciar sesión y registrarse con su cuenta de google mediante Oauth 2.0
RNF 4. La aplicación utiliza una base de datos no relacional, en nuestro caso MongoDB
RNF 5. La aplicación utiliza un servicio externo para gestionar el almacenamiento de imágenes, que en nuestro caso es Cloudinary.
RNF 6. La aplicación utilizará un servicio externo para mostrar los mapas que en nuestro caso es OpenStreetMaps.

4. Arquitectura de la aplicación

El esquema de navegación se ha realizado en el lenguaje IFML y es el siguiente:



5. Base de datos

La base de datos es una base de datos en la nube de MongoDB cuya URI es: mongodb+srv://rastroAdmin3:adminadmin@ingweb.xdloocr.mongodb.net/elRastro3

En ella se encuentran dos entidades: usuarios y productos. Esto es así para aprovechar las ventajas de un diseño no relacional. Gracias a eso, la mayoría de la información se encuentra en los productos, ya que es lo que más se va a consultar en la aplicación.

Atributos Producto	**Contenido**
_id	Identificador del objeto
descripcion	Texto con información del producto
fechaCierre	Fecha en la que se cierran las pujas
foto	Imagen principal del producto
imagenes	Imágenes secundarias del producto
precioInicial	Precio inicial del producto
maximaPuja	Precio de la máxima puja realizada
titulo	Nombre del producto
ubicacion	Dirección de la ubicación del producto
vendedor	Id del vendedor del producto
comprador	Id del comprador del producto
vendido	Booleano que indica si el producto se ha vendido
valoracionVendedor	Booleano indicando si el vendedor ha valorado al comprador
valoracionComprador	Booleano indicando si el comprador ha valorado al comprador
comentarios	Lista con los comentarios realizados sobre el producto
pujas	Lista con las pujas realizadas al producto
peso	Peso en gramos del producto

Los comentarios y las pujas son objetos que se encuentran dentro de los productos:

Atributos de los comentarios	**Contenido**
usuario	Id del usuario que realiza el comentario
texto	Contenido del comentario
fecha	Fecha en la se realiza el comentario
respuesta	Contenido de una posible respuesta al comentario
_id	Id del comentario

Atributos de las pujas	**Contenido**
precio	Precio de la puja
usuario	Id del usuario que realizó la puja
_id	Id de la puja

Los usuarios son una entidad aparte:

Atributos de usuario	**Contenido**
_id	Id del usuario
foto	Enlace de la imagen asociada al usuario
username	Nombre del usuario en la aplicación
correo	Dirección de correo electrónico de Google
valoracionMedia	Valoración media calculada a partir de las valoraciones recibidas por compradores.

6. Instrucciones de instalación

Usuario de acceso: Como se explica al inicio del documento, para facilitar la vista de las distintas acciones que se pueden realizar en la aplicación se ha introducido en la base de datos un usuario con correo pruebaparaingweb@gmail.com que ya tiene un producto comprado, un producto pendiente para comprar y pujas en algunos productos . En cualquier caso, para comprobar las funcionalidades correspondientes a usuarios recién registrado como la solicitud de introducir la ubicación para poder ver el mapa principal, se recomienda iniciar sesión con otro correo distinto que no esté registrado en la base de datos.

Acceso desde la nube: El proyecto se encuentra totalmente desplegado en los servidores de Vercel y se puede acceder a la aplicación mediante el siguiente enlace: https://frontend-rastro.vercel.app/

Si se desea hacer consultas al backend subido en la nube se puede hacer mediante el siguiente enlace:
https://backend-rastro.vercel.app/

Ejecución en local: El proyecto entregado contiene un archivo ejecutable “scriptFrontend.bat” que instalará las librerías necesarias para ejecutar el proyecto e iniciará el frontend automáticamente en local. En este caso solo ejecutará el frontend, ya que las llamadas ‘fetch’ que hace este hacia el backend son a su dirección en la nube, por lo tanto no es necesario ejecutarlo en local.

7. Replanteamientos de la API REST

7.1 Cambios en la base de datos

Hemos realizado ciertos cambios para mejorar las consultas de la geolocalización. Aunque OpenStreetMap no tenga límites en cuanto a consultas, hemos añadido tanto en el modelo producto como en el de usuario los atributos lat y lon.
De esta manera sólo hará falta hacer consultas a la api cuando se crea una nueva instancia o cuando se modifica su atributo ubicación.
Por lo tanto, ya no necesitamos la consulta getUbiProducto que teníamos en el controlador de los productos.

Hemos añadido el atributo correo al usuario ya que al hacer el login con google obtenemos el gmail gracias al token. Y ese gmail se almacena en el atributo correo.

Productos (prefijo “/productos”)

Endpoint	**Petición**	**Método del controlador**
/inicio/mostrar	GET	getProductosInicio
Devuelve todos los productos que no han sido vendidos (aún se puede pujar por ellos)		

Endpoint	**Petición**	**Método del controlador**
/:idUsuario/inicio	GET	getProductosInicioConLogin
Devuelve todos los productos que no han sido vendidos y el usuario del id especificado		

Endpoint	**Petición**	**Método del controlador**
/radio	POST	getProductosRadio
Devuelve una lista de productos cuya distancia a la ubicación del usuario especificado sea menor al radio. Pasamos por el body: - radio: distancia max a la que deben estar los productos - idUsuario: id del usuario que aplica el filtro		

Endpoint	**Petición**	**Método del controlador**
/radioPrecio	POST	getProductosPrecioMaxRadio
Devuelve una lista de productos cuya distancia a la ubicación del usuario especificado sea menor al radio y cuya puja más alta sea menor al precio especificado. Pasamos por el body: - radio: distancia max a la que deben estar los productos - idUsuario: id del usuario que aplica el filtro - precio: cantidad que no debe superar la puja más alta actual del producto		

Endpoint	**Petición**	**Método del controlador**
/descripcionRadio	POST	getProductosDescripcionRadio
Devuelve una lista de productos cuya distancia a la ubicación del usuario especificado sea menor al radio y cuyo título o descripción coincida con la cadena especificada. Pasamos por el body: - radio: distancia max a la que deben estar los productos - idUsuario: id del usuario que aplica el filtro - descripcion: cadena que usaremos para compararla con el título y descripción del producto. Si coincide con alguno entra en el filtro.		

Endpoint	**Petición**	**Método del controlador**
/descripcionRadioPrecio	POST	getProductosDescripcionRadioPredio
Devuelve una lista de productos cuya distancia a la ubicación del usuario especificado sea menor al radio, cuyo título o descripción coincida con la cadena especificada y puja más alta sea menor al precio especificado. Pasamos por el body: - radio: distancia max a la que deben estar los productos - idUsuario: id del usuario que aplica el filtro - descripcion: cadena que usaremos para compararla con el título y descripción del producto. Si coincide con alguno entra en el filtro. - precio: cantidad que no debe superar la puja más alta actual del producto		

Endpoint	**Petición**	**Método del controlador**
/huellaCarbonoNuevo	POST	getHuellaCarbono
Esta consulta hace lo mismo que la que ya teníamos pero además de la huella de carbono devuelve el vendedor y el producto.		

Usuarios (prefijo “/usuarios”)

Endpoint	**Petición**	**Método del controlador**
/loginToken/:token	GET	verificarTokenGoogle
Decodificamos el token con jwt y extraemos de él: correo, tiempo expiración del token, id del token y la foto de la cuenta de google del usuario. Si ya existe un usuario cuyo correo sea este,devolvemos su id. En el caso contrario creamos un usuario rellenando solo los atributos username y correo.		

Endpoint	**Petición**	**Método del controlador**
/conexion/:idUsuario/:tokenId/:token	GET	verificarConexion
En este método comprobamos comprobamos que la sesión del usuario siga siendo válida teniendo en cuenta varios casos: 1. Que el token actual del sistema pertenezca al mismo usuario que tiene la sesión iniciada, es decir no ha cambiado la sesión en otra pestaña 2. Que el token actual sea uno válido y coincida con la decodificación obtenida en el login 3. Que no haya expirado en token		

Rutas del frontend

Se han añadido rutas nuevas al frontend:

Rutas	**Función**
/borrarSubasta/:idProducto/:idUsuario	Página intermedia que se genera cuando se borra la subasta de un producto por parte de un usuario.
/pago/:idUsuario/:idProducto	Lleva a la página del pago de un producto por parte del usuario que ha ganado la subasta.
/borrarPuja/:idUsuario/:idProducto/:idPuja	Borrado de la puja especificada de un producto por parte del usuario que la creó.
/borrarComentario/:idUsuario/:idProducto/:idComentario	Borrado del comentario especificado de un producto por parte del vendedor

8. Servicios externos

8.1 OpenStreetMap

Hemos usado leaflet y react-leaflet para la visualización de los mapas. Y para obtener las coordenadas geográficas hemos hecho consultas a la API de Nominatim de OpenStreetMap.

En la página de inicio una vez se ha iniciado sesión podemos ver el mapa centrado en la ubicación del usuario con el que se ha iniciado sesión*. Esa misma ubicación tiene un marcador de color rojo. Además aparecen en el mapa todos los productos por los que aún se pueda pujar. Estos están con un marcador negro. A la derecha del mapa tenemos un formulario que filtra por diferentes criterios (por separado o combinados) uno de esos criterios es la proximidad en kilómetros desde la ubicación del usuario.

Como en el registro de los usuarios nuevos no se introduce la ubicación hasta que estos la modifiquen desde la vista de su perfil. En caso de que la ubicación tenga los valores predeterminados (lat = 0, lon = 0) no aparecerá el mapa. Aparecerá un mensaje indicando que introduzcan su ubicación.

*La única ubicación de usuario que puedes ver en el mapa es la tu propio usuario.

En la vista de un producto concreto vemos su ubicación marcada en el mapa.
Punto de acceso de Nominatim:

```
const nominatimEndpoint ='https://nominatim.openstreetmap.org/search';
const format = 'json';
const response = await axios.get(`${nominatimEndpoint}?q=${ubicacion}&format=${format}`);
```

https://nominatim.openstreetmap.org/search → URL base
format → formato de la respuesta de la consulta, en este caso JSON
ubicacion → parámetro que contiene la dirección que buscamos en una cadena String
response → Contiene la respuesta en formato JSON. Con axios hacemos la consulta HTTP GET a la URL construida.

Punto acceso OpenStreetMap para el mapa:

```
<TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
attribution='&copy;
<a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors'
/>
```

8.2 Subir fotos con Cloudinary

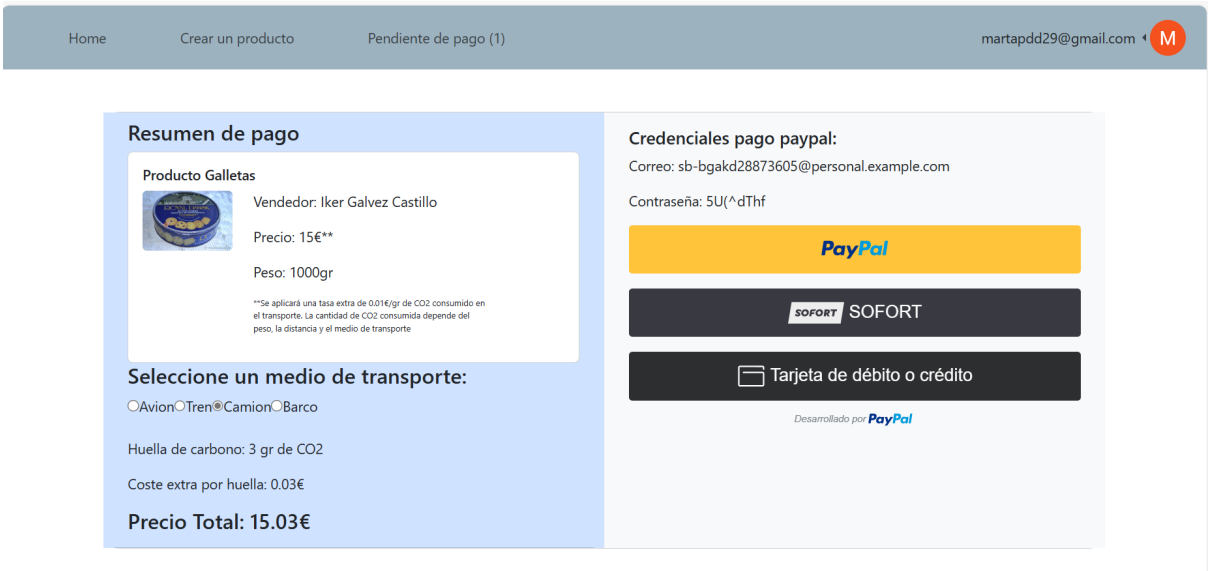
En cuanto a las imágenes un usuario podrá ponerse 1 foto de perfil. Los productos tienen 1 foto principal y después una lista de fotos “secundarias”. La gestión de las imágenes la hemos hecho mediante Cloudinary. De manera que el usuario introduce una imagen almacenada en su dispositivo, nosotros subimos la imagen a Cloudinary y la url la almacenamos en la base de datos.
Hemos aplicado un cambio respecto a la entrega anterior. Y es que en este proceso de subir la foto a Cloudinary, también la almacenamos localmente en una carpeta dentro del propio proyecto. Hemos omitido este almacenamiento interno ya que vemos que no nos aporta gran cosa.

8.3 Pago a través de PayPal

Una vez se cumple la fecha de cierre de una subasta, el usuario con la mayor puja realizada en dicha subasta tiene la oportunidad de comprarlo por ese mismo precio. Cuando un usuario tiene algún producto pendiente de pago le aparecerá un aviso en el menú superior y en el perfil del usuario, indicando además el número de productos que debe pagar.

Cuando el usuario decide comprar el producto, podrá seleccionar el medio de transporte en el que quiere que se le envíe el producto, esto influirá en el precio total que se pague, ya que la huella de carbono varía dependiendo del método de transporte.

Se ofrece, en la misma vista del pago, unas credenciales de una cuenta sandbox de paypal para poder realizar pagos de prueba sin realizar ninguna transferencia real.



8.4 Login con Google OAuth

Se ha hecho uso del servicio de Google OAuth para el inicio de sesión de los usuarios. Para obtener el token cifrado de inicio de sesión se ha usado la librería https://accounts.google.com/gsi/client . Una vez obtenido se ha usado la librería **jwt-decode** para decodificarlo y obtener la información que nos conviene (correo, tokenId, expiración, imagen)**.**

Para tratar esta información se usan dos métodos uno para el inicio de sesión y otro para la verificación de la conexión. Cuando un usuario inicia sesión se y se decodifica se comprueba si en la base de datos hay un usuario con el mismo correo electrónico, si no lo hay se crea uno y en ambos casos se devuelven por el localStorage los datos de sesión del token y el id del Usuario de nuestra base de datos. Con estos datos del localStorage y el id del usuario se comprueba que no se ha cambiado la sesión del usuario y que esta sigue siendo válida, si no es válida en algún momento se cierra la sesión y se redirige a la página de inicio de sesión.

Los métodos mencionados anteriormente están definidos en el archivo “LoginGoogle.js” ubicado en backend/src/controllers . La llamada al endpoint que comprueba la conexión está definida en el useEffect del archivo “navbar.js” por lo que se ejecuta cada vez que se carga el componente.

Por otro lado si localStorage está vacío se tratará al usuario como un invitado que no posee los mismo permisos que un usuario identificado y aunque pueda visualizar cierta información no podrá realizar acciones.

8.5 Cálculo de la huella de carbono

Para obtener la huella de carbono para el envío de un producto se ha hecho uso de la Api de Carbon Interface https://www.carboninterface.com/api/v1/estimates . Esta api nos permite obtener los gramos de CO2 consumidos en el envío de un producto de peso X , en una medio de transporte concreto (plane, truck, train, ship) y una distancia Y.

Este servicio se utiliza en el momento de la compra del producto y para obtener la distancia de envío primero se obtiene la latitud y longitud de ambas ubicaciones (origen y destino) y mediante un método se calcula la distancia en km entre esos dos puntos.
Estos valores se utilizan para añadir el coste extra del envío del producto.

Se está haciendo uso de la versión de prueba de esta API, que solo permite hasta un máximo de 200 consultas por cuenta. Por ello, la entrega de esta práctica se ha hecho con una cuenta nueva con todas las consultas disponibles, para evitar problemas a la hora de probar la aplicación.

8.6 Vercel

Gracias a vercel se han subido a la nube el backend y frontend de la aplicación en servidores independientes. Esto permite un acceso fácil a los servicios de la aplicación a través de URLs públicas, mencionadas previamente en este documento:

https://frontend-rastro.vercel.app/
https://backend-rastro.vercel.app/

9. Nuevas funcionalidades

9.1 Cancelar y modificar subasta.

En la entrega anterior el vendedor de un producto podía modificar la descripción y la ubicación de cualquier producto que vendía en cualquier momento.
Hemos añadido que el vendedor puede modificar el precio inicial y la fecha de cierre de la subasta siempre y cuando no haya ninguna puja hecha. En este mismo periodo de tiempo el vendedor también podrá cancelar la subasta. Lo que implica el borrado del producto de la base de datos.

9.2 Mapa en Home y filtro proximidad

Como hemos explicado en el apartado de OpenStreetMap, en la página de inicio hemos añadido un mapa en el que se muestra la ubicación de todos los productos que están en venta y la ubicación de tu usuario. También hemos añadido a los filtros la opción de filtrar por la proximidad desde tu propia ubicación.

9.3 Carrusel de fotos

En la práctica anterior se implementó un crear producto que permitía crear un producto con un conjunto de fotos y se han añadido en la vista de “Detalles de un producto” un carrousel que nos permite ver todas las fotos de un objeto. Además en tus propios productos puedes añadir nuevas fotos al carrousel.

9.4 Cambios en pujas y comentarios

La implementación anterior permitía el añadido de pujas y comentarios. En esta última versión se ha ampliado sus usos y restricciones. Ahora, para poder pujar el precio debe ser mayor al precio inicial puesto por el vendedor, y no se permitirá pujar o comentar cuando la subasta ya haya terminado. Además, la aplicación permite borrar una puja o comentario una vez la hayas publicado (el propietario del producto no tiene permisos para borrar ninguna puja o comentario).

9.5 Tasa de envío en función de la huella de carbono

Una vez un usuario ha ganado con la puja el producto tiene que realizar el pago, en el apartado de pago. En ese apartado aparece la huella de carbono para el envío entre la ubicación del producto y la del usuario, una vez obtenida en función de la distancia y el tipo de transporte se multiplica por 0.01 y se suma al precio total que se incluirá en el pago de paypal

9.6 Subastas desiertas

Cuando una subasta de un producto alcance su fecha de cierre, pero no contiene ninguna puja, se declara como subasta desierta, entonces automáticamente la aplicación rebaja el precio inicial de la subasta un 10% menos y amplía la fecha de cierre 10 días más.

9.7 Productos pendientes de pago

Una vez termina la subasta de un producto, se selecciona como comprador al usuario al que pertenece la mayor puja. Cuando un usuario gana una subasta y tiene un producto pendiente de pago le aparecerá un aviso en el menú superior y en el perfil del usuario, indicando además el número de productos que debe pagar, ese aviso le lleva directamente a la colección de productos que debe pagar.