LEGIT STYLE

Laravel Api Rest

Alumnes: Rubén Teruel, Brian Vilchez
Profesor: Ramón Cervera
19/12/2023
Curs: DAW2

ÍNDICE

Bienvenido a la API de Legit Style, una robusta plataforma construida sobre el potente framework Laravel, diseñada para facilitar la gestión eficiente de operaciones en una tienda de ropa urbana.

1. Introducción

- Objetivo Principal

El objetivo fundamental de la API Legit Style es proporcionar a la tienda una plataforma integral para la gestión de productos, pedidos y la interacción efectiva con sus clientes.
A través de servicios web RESTful, esta API se convierte en la columna vertebral de la operación, permitiendo a Legit Style mejorar significativamente la eficiencia operativa y ofrecer una experiencia de compra sin contratiempos a sus clientes.

- Tecnologías Utilizadas

La API Legit Style se construye sobre Laravel, aprovechando sus características como eloquent ORM para una gestión eficiente de la base de datos, el sistema de rutas para manejar las solicitudes HTTP y la integración de controladores para gestionar la lógica de la aplicación.

- Seguridad y Escalabilidad

La seguridad es una prioridad clave, con la implementación de prácticas recomendadas de Laravel y la posibilidad de extender las funciones de autenticación y autorización según las necesidades específicas de Legit Style.
Además, la arquitectura de la API se ha diseñado pensando en la escalabilidad, permitiendo un crecimiento sin problemas a medida que la tienda expande su presencia en línea.

2. Estructura del proyecto

2.1 Lista de los nuevos controladores i sus funciones

- **UsuariosControllerApi.php**

1. index()
- Descripción: Muestra un listado paginado de usuarios.
- Lógica: Recupera los usuarios ordenados por la fecha de creación de forma descendente y los devuelve como respuesta JSON.

2. all()
- Descripción: Muestra todos los usuarios sin paginación.
- Lógica: Recupera todos los usuarios y los devuelve como respuesta JSON.

3. create(Request $request)
- Descripción: Crea un nuevo pedido de compra junto con su detalle asociado.
- Lógica:
  - Valida los campos del pedido de compra.
  - Crea un nuevo pedido de compra y su detalle asociado con información proporcionada en la solicitud.
  - Maneja errores y devuelve una respuesta JSON indicando el éxito o el fallo de la operación.

4. store(Request $request)

- Descripción: Almacena un nuevo usuario en la base de datos.
- Lógica:
  - Valida los campos del usuario.
  - Crea un nuevo usuario con la información proporcionada en la solicitud.
  - Maneja errores y devuelve una respuesta JSON indicando el éxito o el fallo de la operación.

5. show($id)

- Descripción: Muestra los detalles de un usuario específico.
- Lógica:
  - Busca un usuario por su ID.
  - Devuelve una respuesta JSON con los detalles del usuario encontrado o un mensaje de error si no se encuentra.

6. edit($id)

- Descripción: No implementado.
- Lógica: Esta función no contiene lógica ya que la descripción indica que no está implementada.

7. update(Request $request, $id)

- Descripción: Actualiza los detalles de un usuario existente.
- Lógica:
  - Busca un usuario por su ID.
  - Valida los campos que se van a actualizar.
  - Actualiza los detalles del usuario con la información proporcionada en la solicitud.
  - Maneja errores y devuelve una respuesta JSON indicando el éxito o el fallo de la operación.

8. destroy($id)

- Descripción: Elimina un usuario existente.
- Lógica:
  - Busca un usuario por su ID.
  - Intenta eliminar el usuario.
  - Maneja errores y devuelve una respuesta JSON indicando el éxito o el fallo de la operación.

- **PedidoCompraControllerApi.php**

1. index()
- Descripción: Muestra un listado paginado de pedidos de compra.
- Lógica: Recupera los pedidos de compra ordenados por la fecha del pedido de forma descendente y los devuelve como respuesta JSON paginada.

2. create()
- Descripción: No implementado.
- Lógica: Esta función no contiene lógica ya que la descripción indica que no está implementada.

3. store(Request $request)
- Descripción: Crea un nuevo pedido de compra junto con su detalle asociado.
- Lógica:
  - Valida los campos del pedido de compra.
  - Crea un nuevo pedido de compra y su detalle asociado con información proporcionada en la solicitud.
  - Maneja errores y devuelve una respuesta JSON indicando el éxito o el fallo de la operación.

4. show($id)
- Descripción: Muestra los detalles de un pedido de compra específico.
- Lógica:
  - Busca un pedido de compra por su ID.
  - Devuelve una respuesta JSON con los detalles del pedido encontrado o un mensaje de error si no se encuentra.

5. edit($id)

- Descripción: Muestra el formulario de edición de un pedido de compra y la lista de usuarios.
- Lógica:
  - Busca el pedido de compra y todos los usuarios.
  - Devuelve una respuesta JSON con los detalles del pedido de compra y la lista de usuarios.

6. update(Request $request, $id)

- Descripción: Actualiza los detalles de un pedido de compra existente y su detalle asociado.
- Lógica:
  - Valida los campos que se van a actualizar.
  - Busca el pedido de compra por su ID junto con su detalle asociado.
  - Actualiza el pedido de compra y su detalle solo si hay cambios.
  - Maneja errores y devuelve una respuesta JSON indicando el éxito o el fallo de la operación.

7. destroy($id)

- Descripción: Elimina un pedido de compra existente junto con su detalle asociado.
- Lógica:
  - Busca el pedido de compra por su ID.
  - Intenta eliminar el detalle del pedido asociado.
  - Intenta eliminar el pedido de compra.
  - Maneja errores y devuelve una respuesta JSON indicando el éxito o el fallo de la operación.

- **ProductoTallaControllerApi.php**

1. index():
- Descripción: Obtiene todos los registros de ProductoTalla con relaciones de Producto y Talla.
- Lógica: Utiliza el método `with` para cargar las relaciones y devuelve la respuesta en formato JSON.

2. store(Request $request):
- Descripción: Almacena un nuevo registro en la tabla ProductoTalla.
- Lógica: Realiza validación de datos utilizando el método `validate`, crea un nuevo registro en la base de datos y devuelve la respuesta en formato JSON.

3. update(Request $request, ProductoTalla $productoTalla):
- Descripción: Actualiza un registro existente en la tabla ProductoTalla.
- Lógica: Realiza validación de datos, loguea información antes de la actualización, actualiza las propiedades del registro y devuelve la respuesta en formato JSON. Maneja excepciones y registra errores si se producen.

4. storeOrUpdate(Request $request):
- Descripción: Intenta crear un nuevo registro o actualizar un registro existente en la tabla ProductoTalla.
- Lógica: Realiza validación de datos, busca un registro existente por id_producto e id_talla, y decide si crear un nuevo registro o actualizar el existente. Devuelve la respuesta en formato JSON. Maneja excepciones y registra errores si se producen.

5. destroy(ProductoTalla $productoTalla):

- Descripción: Elimina un registro de ProductoTalla.
- Lógica: Utiliza el método `delete` en el objeto `ProductoTalla` y devuelve una respuesta sin contenido (204) en formato JSON. Maneja excepciones y registra errores si se producen.

6. show($id):

- Descripción: Obtiene un registro de ProductoTalla por su ID con relaciones de Producto y Talla.
- Lógica: Utiliza el método `findOrFail` para encontrar el registro por ID y cargar las relaciones. Devuelve la respuesta en formato JSON. Maneja excepciones y devuelve un error 404 si el registro no se encuentra.

2.2 Nuevas rutas implementadas (api.php)

- ***Usuarios:***

```
Route::resource('/usuarios', App\Http\Controllers\api\UsuariosControllerApi ::class);

Route::get('/usuarios-pas-a-pas', function () {
   return view('usuarios.api.index');
})->name('usuarios-api');
```

- ***Pedidos Compra:***

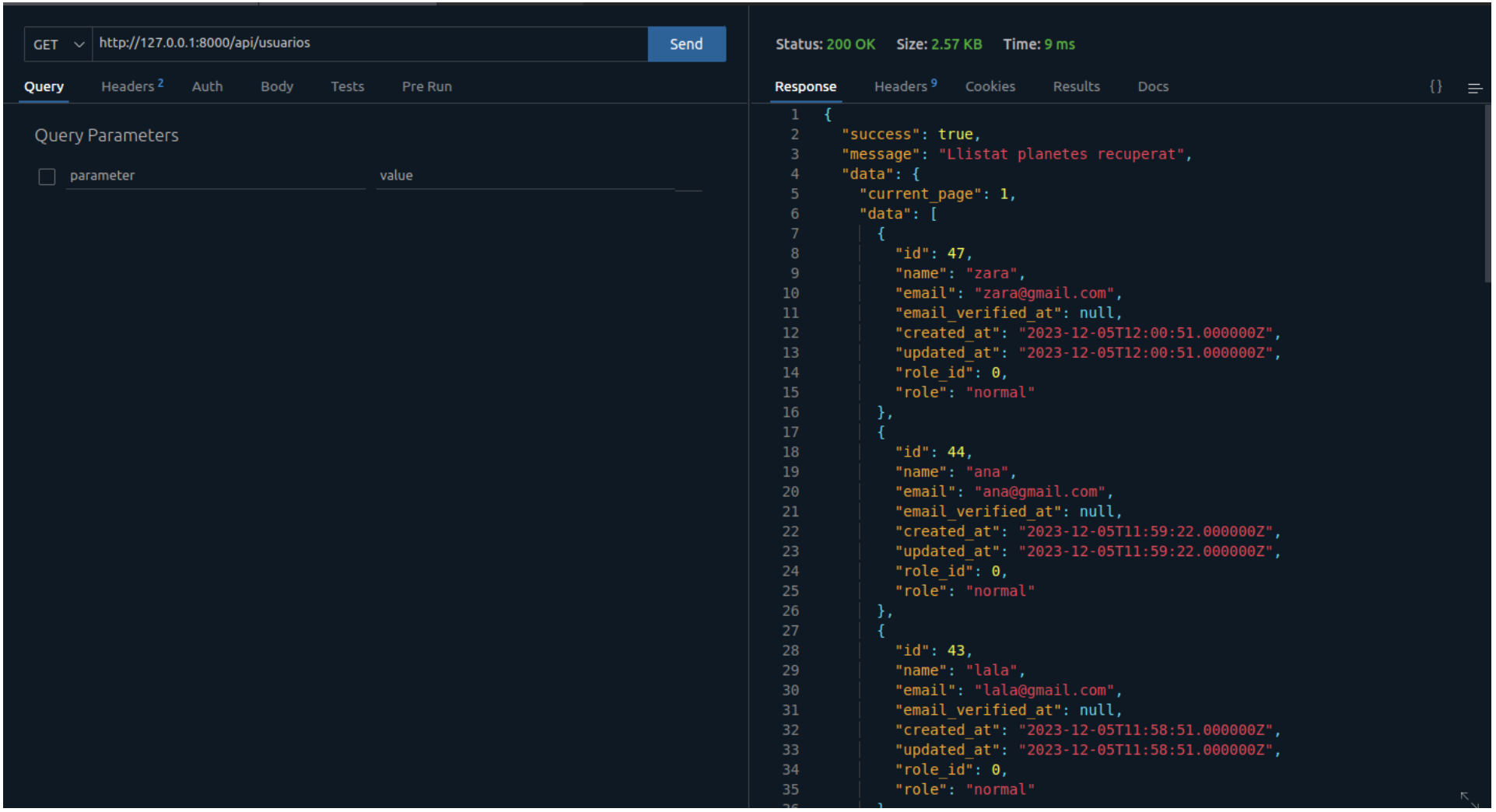
```
Route::resource('/pedidos-compra', App\Http\Controllers\api\PedidoCompraControllerApi::class);

Route::get('/pedidoscompra', function () {
   return view('pedidoscompra.api.index');
})->name('pedidoscompra-api');
```

- ***Productos talla:***

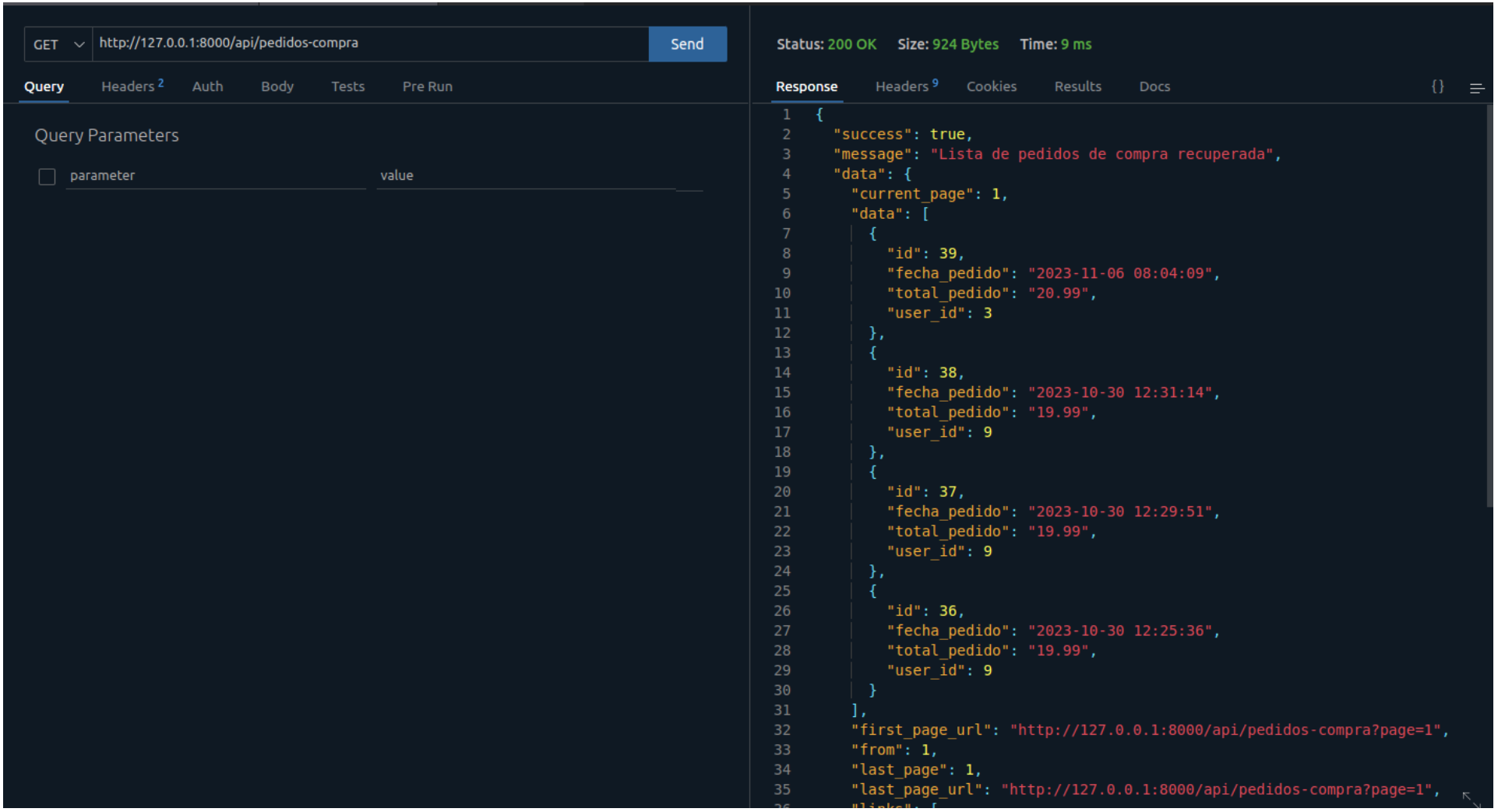
```
Route::resource('/productos-api', App\Http\Controllers\api\ProductoControllerApi::class);
Route::resource('/tallas', App\Http\Controllers\api\TallaControllerApi::class);
Route::resource('/producto-talla', App\Http\Controllers\api\ProductoTallaControllerApi::class);

Route::get('/productotalla', function () {
   return view('producto_talla.api.index');
})->name('producto_talla.api.index');
```

2.3 Pruebas de rutas con cliente REST

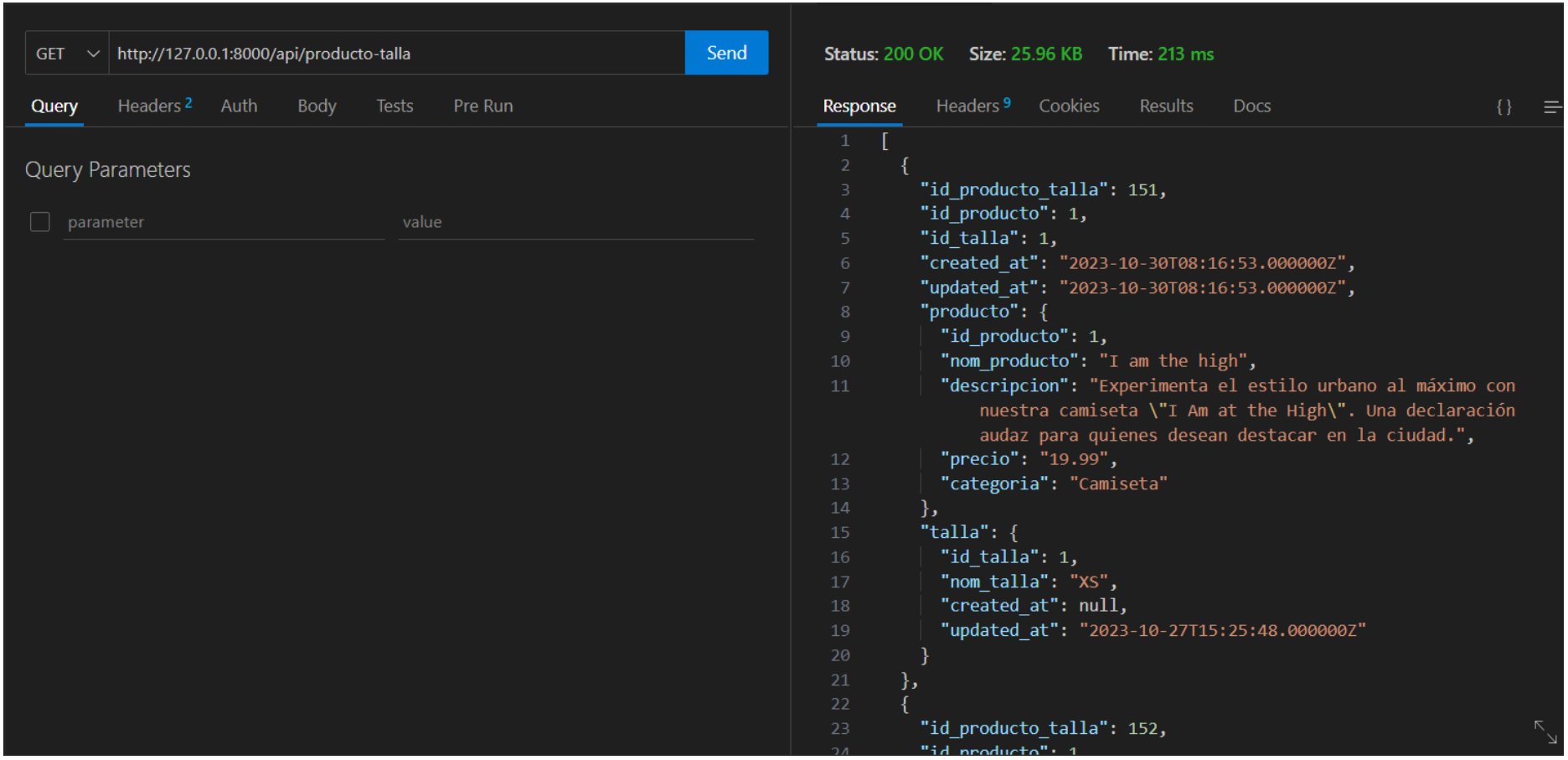
- ***Usuarios***



- ***Pedidos de Compra***



- ***Producto talla***



2.4 Lista de nuevas vistas implementadas

- **resources/views/pedidoscompra/api**

  ❖ ***index.blade.php***

Implementación de la vista de pedidos de compra que permite gestionar y visualizar la información de los pedidos de compra de cada usuario.
La vista incluye un formulario para añadir o actualizar los pedidos de compra, una lista paginada de pedidos de compra existentes, y opciones para eliminar pedidos de compra.
La paginación y la interactividad se han implementado utilizando JavaScript y se ha integrado con una API en el servidor.

- **resources/views/usuarios/api**

  ❖ ***index.blade.php***

Implementación de la vista de usuarios que permite gestionar y visualizar la información de los usuarios.
La vista incluye un formulario para añadir o actualizar usuarios, una lista paginada de usuarios existentes, y opciones para eliminar usuarios.
La paginación y la interactividad se han implementado utilizando JavaScript y se ha integrado con una API en el servidor

- **resources/views/producto_talla/api**

❖ ***index.blade.php***

Implementación de la vista de productos talla que permite gestionar y visualizar la información de los productos talla.
La vista incluye un formulario para añadir o actualizar productos talla, una lista paginada de productos talla existentes, y opciones para eliminar productos talla.
La paginación y la interactividad se han implementado utilizando JavaScript y se ha integrado con una API en el servidor.

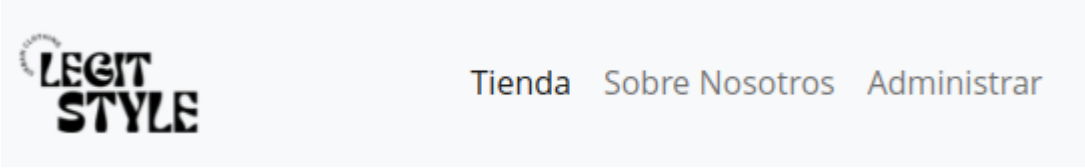
2.5 Enlace github del código fuente

https://github.com/vib-daw2/projecte-2-team-01/tree/master

2.6 Manual de usuario

Accedemos al Panel Administrador iniciando sesión en LEGIT STYLE como admin.

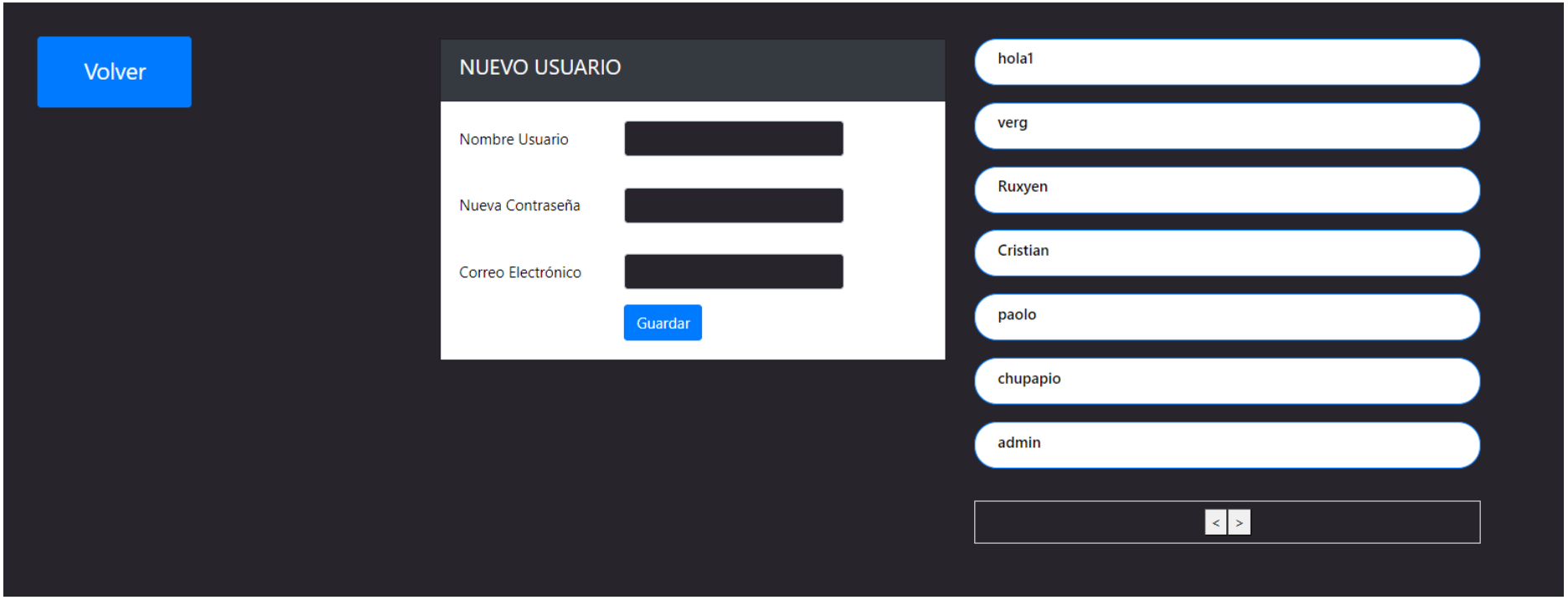
En el menú del navegador de Legit Style aparece una opción extra para acceder a la vista del Panel Administrativo donde encontramos el CRUD.



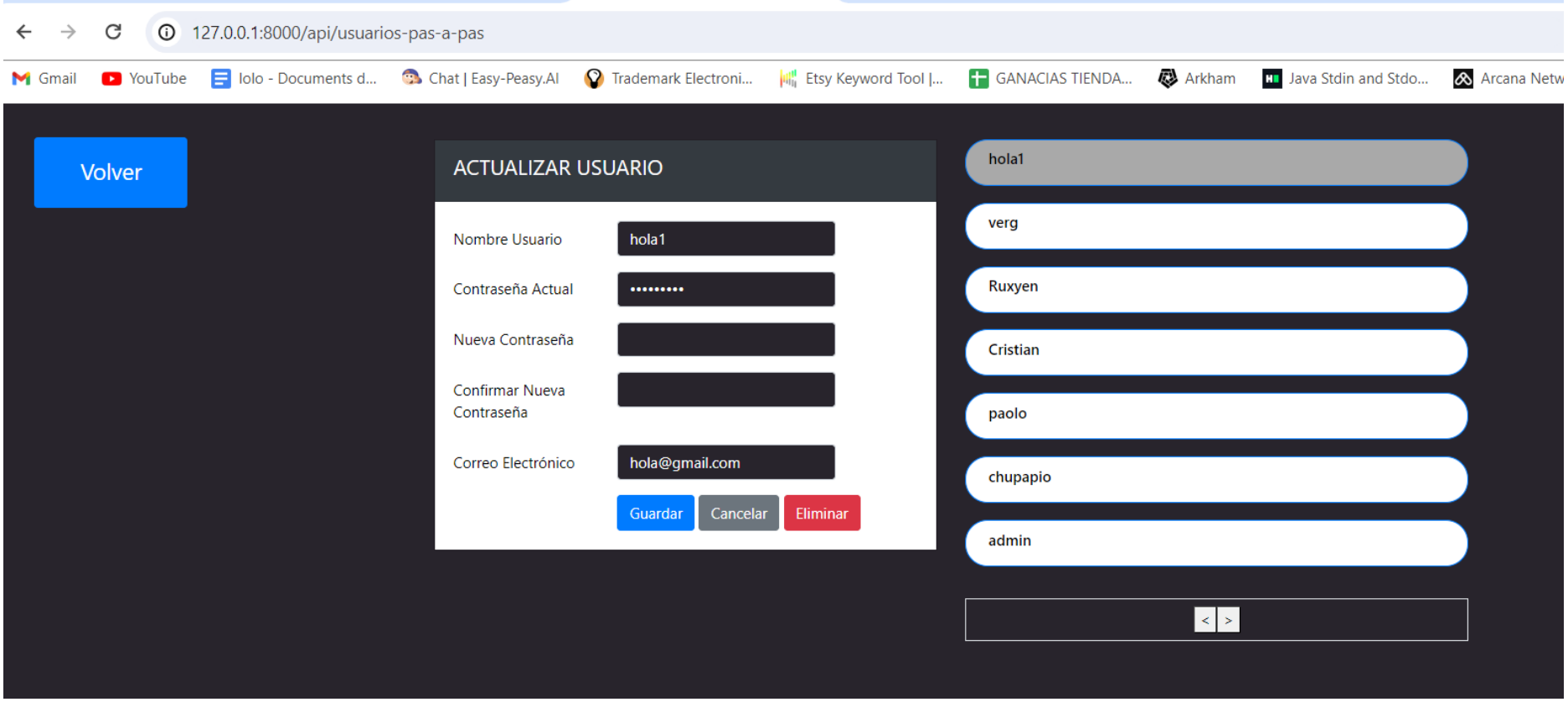
Aquí dentro veremos un nuevo apartado llamado REST, donde nos llevará a las nuevas vistas API REST de Legit Style.

Usuarios:

1. **Crear nuevo usuario**

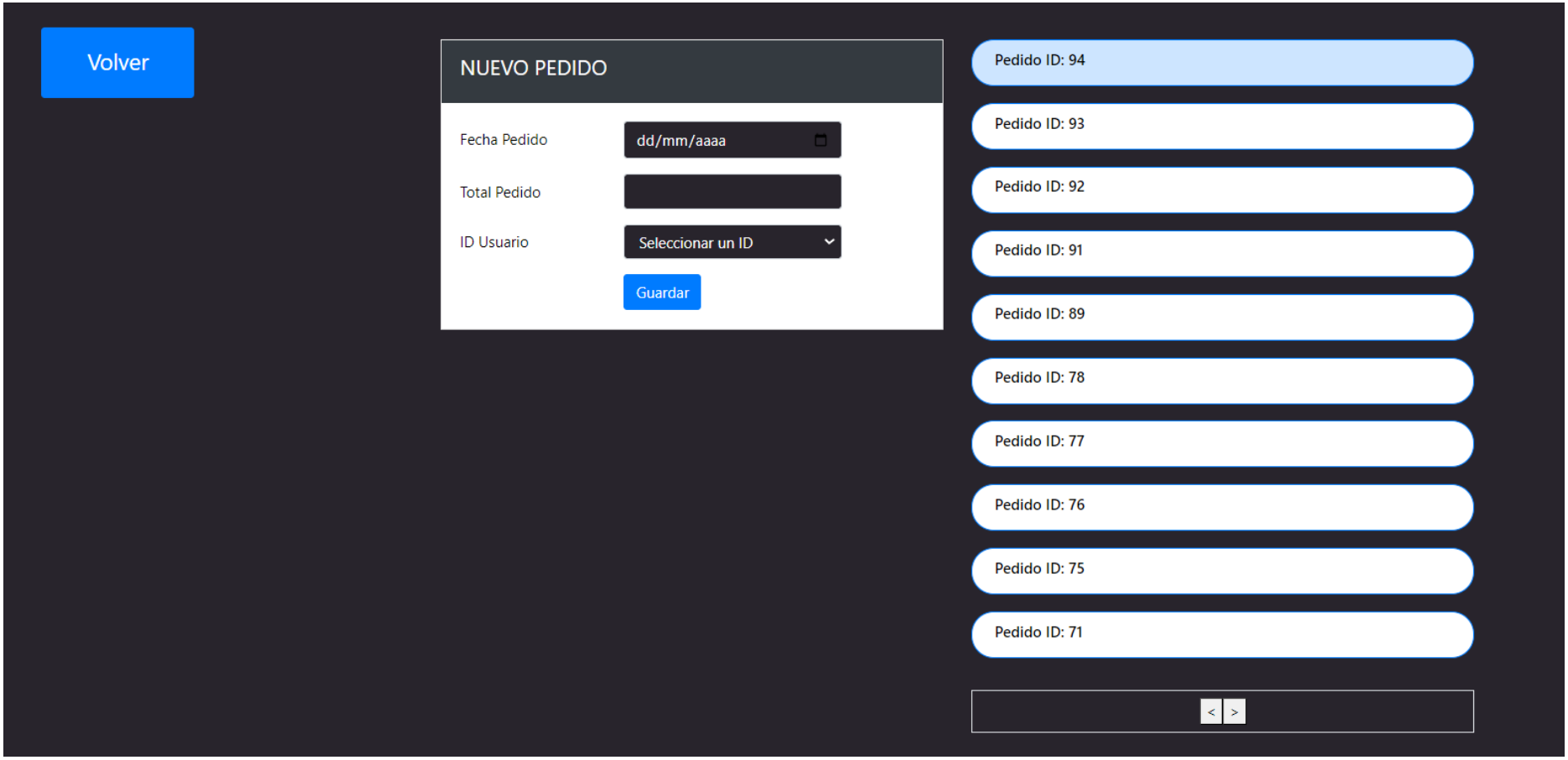


2. Actualizar usuario

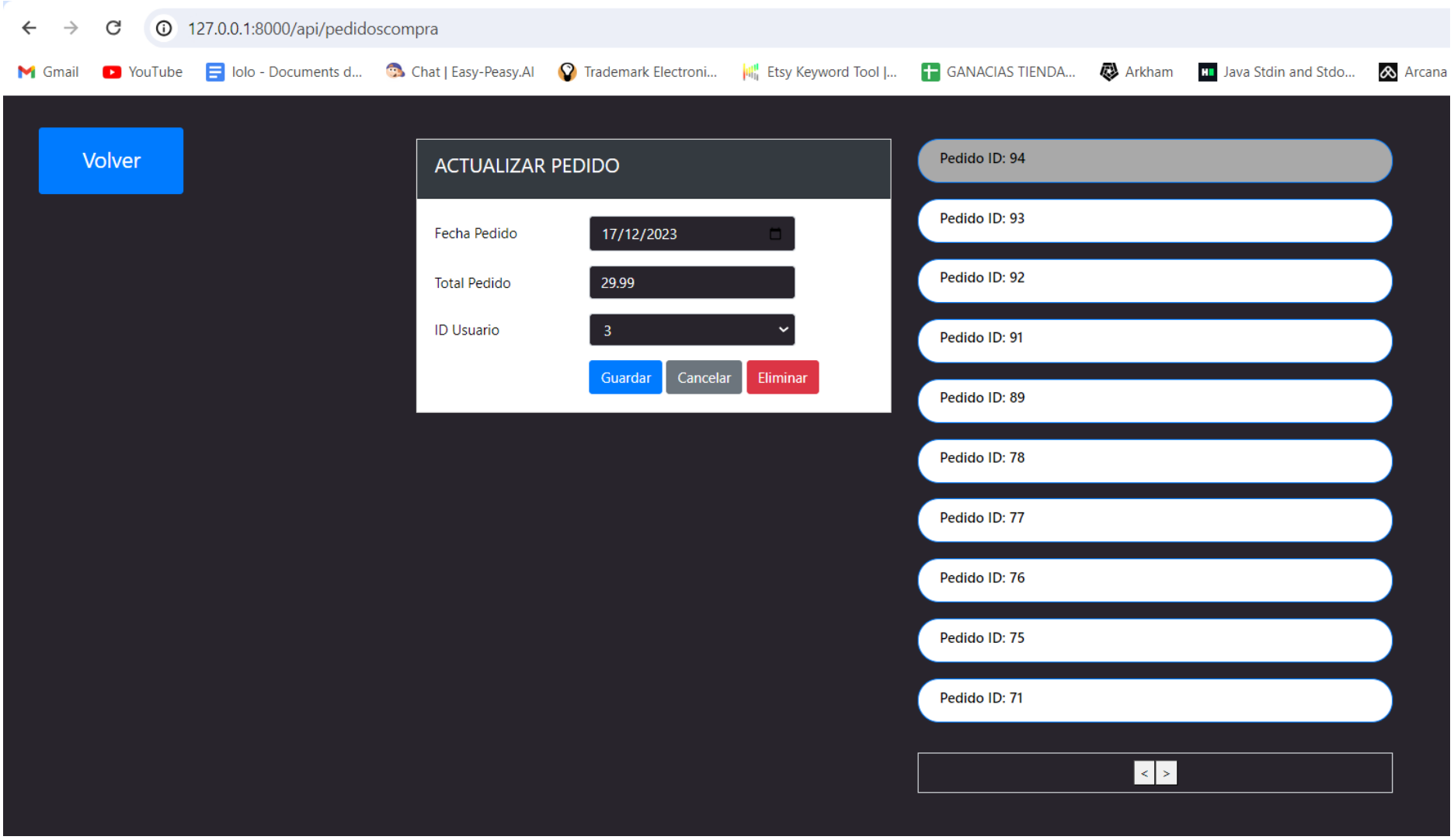


Pedidos de Compra:

1. **Crear pedido de compra**

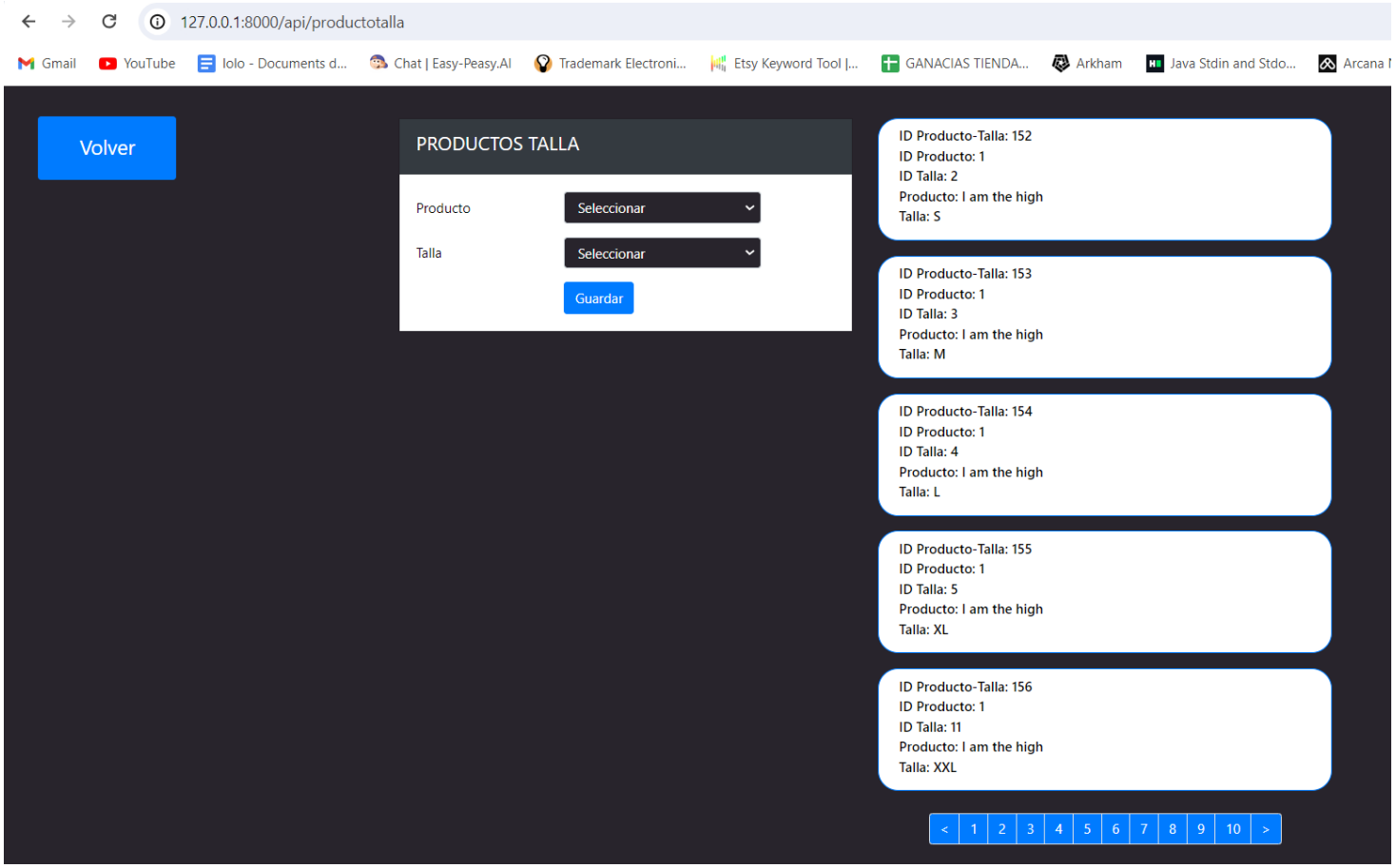


2. Actualizar pedido de compra

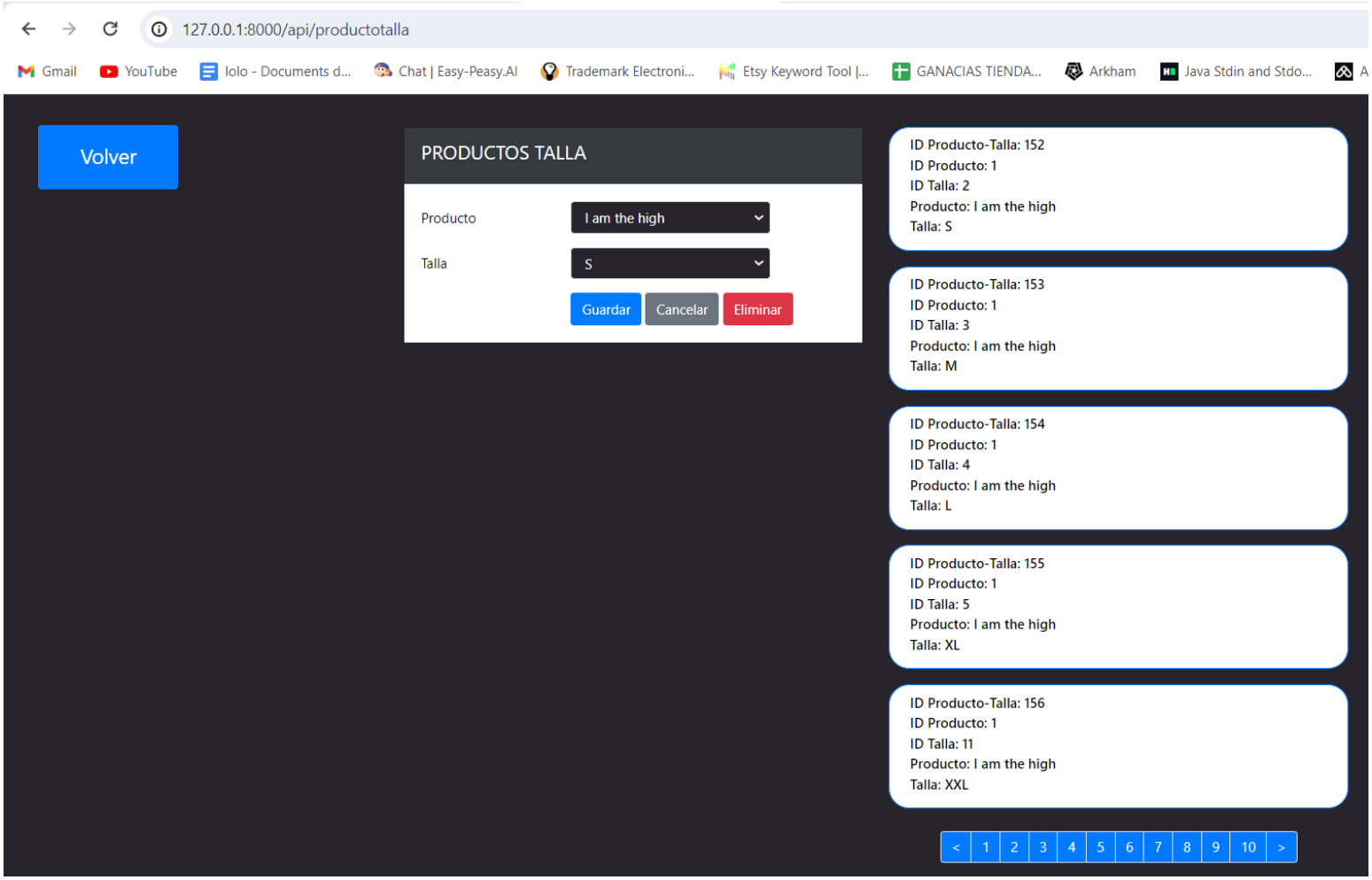


Producto Talla:

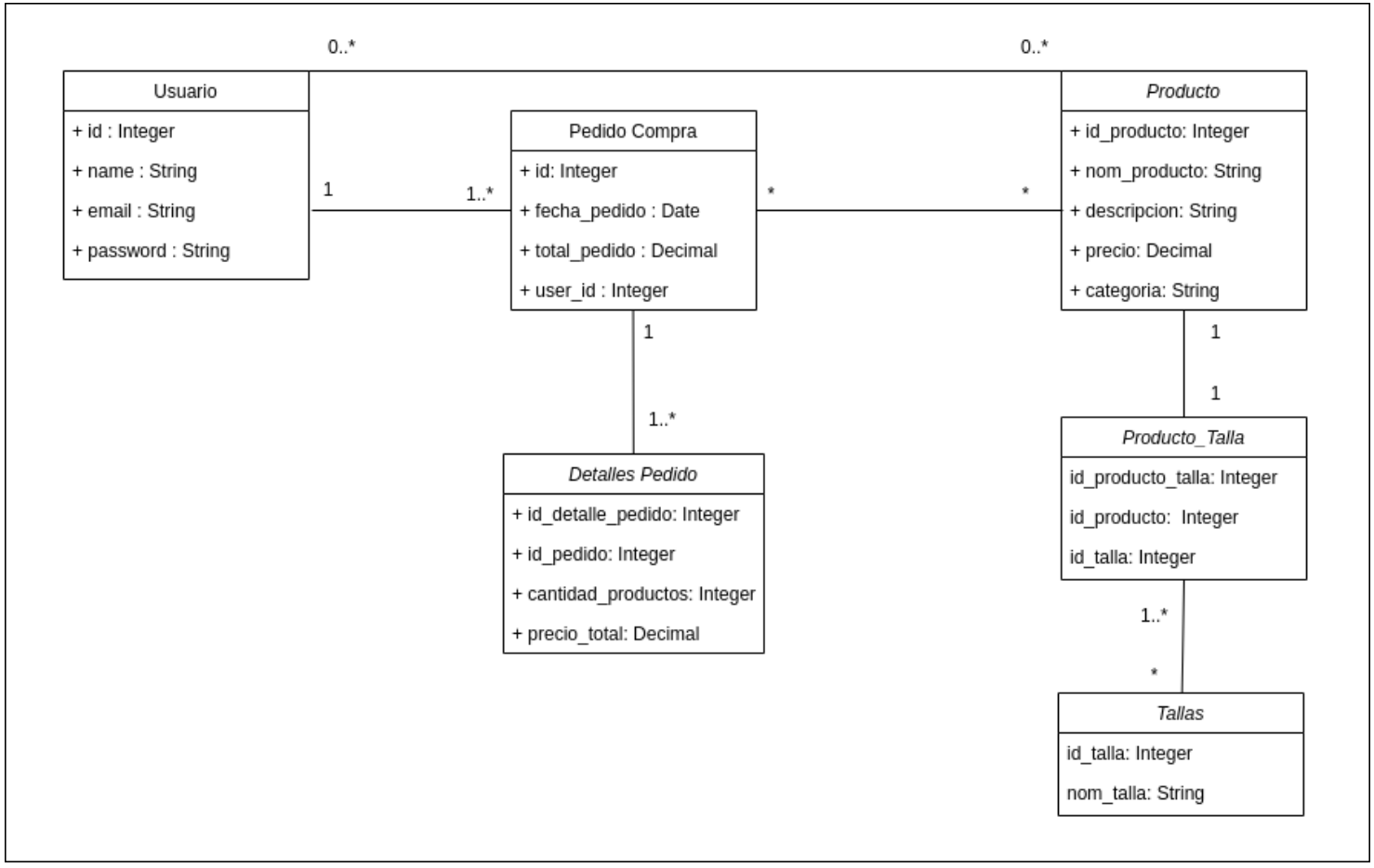
1. **Crear producto talla**



2. Actualizar producto talla



2.7 Diagrama UML



3. Resultados y Valoraciones

- **Implementación exitosa:** El proyecto ha logrado implementar con éxito las funcionalidades esenciales para la gestión de productos, pedidos y usuarios. La integración de Laravel como framework ha demostrado ser una elección acertada para facilitar el desarrollo y la mantenibilidad del sistema.

- **Estructura organizada:** La estructura del proyecto, desde las migraciones hasta los controladores y modelos, está claramente organizada. Esto facilita la comprensión del código y futuras expansiones o modificaciones.

- **Base de datos bien diseñada:** Las migraciones y modelos reflejan una base de datos bien diseñada, con tablas que representan de manera efectiva las entidades del dominio, como productos, pedidos y usuarios.

- **Uso de seeders:** La implementación de seeders para poblar la base de datos con datos de muestra es una práctica sólida. Facilita las pruebas y la demostración del sistema sin la necesidad de ingresar manualmente grandes cantidades de datos.

- **Controladores especializados:** La presencia de controladores específicos para cada entidad, como ProductosController o UsuariosController, indica una separación clara de responsabilidades y facilita el mantenimiento del código.

- **Rutas implementadas:** La inclusión de una sección sobre las rutas implementadas proporciona una visión general de las funcionalidades accesibles a través de la interfaz web.

4. Fuentes informativas

- **Documentación de Laravel:** Exploramos la documentación oficial de Laravel en laravel.com/docs, siendo nuestra guía principal para comprender a fondo las características del framework. Esta referencia nos ayudó a adoptar las mejores prácticas y seguir directrices recomendadas, asegurando una implementación sólida y eficiente.

- **Comunidad de Desarrolladores:** Ambos participamos en la comunidad de desarrolladores de Laravel, especialmente en laracasts.com/discuss. En este espacio, encontramos respuestas a desafíos específicos y valiosas perspectivas de otros desarrolladores, enriqueciendo la calidad de nuestro código mediante el intercambio de experiencias.

- **Recursos Online:** Exploramos varios recursos en línea, como tutoriales y blogs especializados en Laravel y desarrollo web en general. Plataformas como Medium y Dev.to resultaron útiles para acceder a ejemplos prácticos, consejos y trucos que mejoraron nuestra eficacia como equipo de desarrollo.

- **Repositorio de GitHub de Laravel:** Manteníamos nuestro proyecto actualizado revisando el código fuente del repositorio oficial de Laravel en github.com/laravel/laravel. Esto nos permitió conocer las últimas actualizaciones, correcciones de errores y nuevas características del framework.