UNIVERSIDAD GALILEO
Técnico en desarrollo de software
Base de datos III
Sección A

Proyecto Final

Carnet	Nombre
24001350	Dimas Andres Quintana Ramirez
17005131	Jose Carlos Perez Pamech
13005169	Byron Neftalí amírez Rodríguez
24001219	Maria Esther Pineda Izquierdo

Sistema de Gestión de Inventario

Descripción

Este sistema gestiona múltiples almacenes, productos, órdenes de compra y venta, proveedores y usuarios. La base de datos está diseñada para garantizar escalabilidad, seguridad, integridad de datos, eficiencia y flexibilidad.

Entidades

1. Warehouse (bodega)
2. Product (producto)
3. Inventory (inventario)
4. Suppliers (Proveedores)
5. Purchase_oder (Ordenes de compra)
6. Purchase_order_item (Artículos de compra)
7. Sales_order (Ordenes de venta)
8. Sales_order_idem (Articuos de venta)
9. User (Usuarios)

Características

1. Escalabilidad

- Permite la administración de múltiples almacenes y productos.
- Uso de índices en claves primarias y foráneas para optimizar búsquedas.

2. Seguridad

- Autenticación basada en usuarios con roles para restringir accesos.

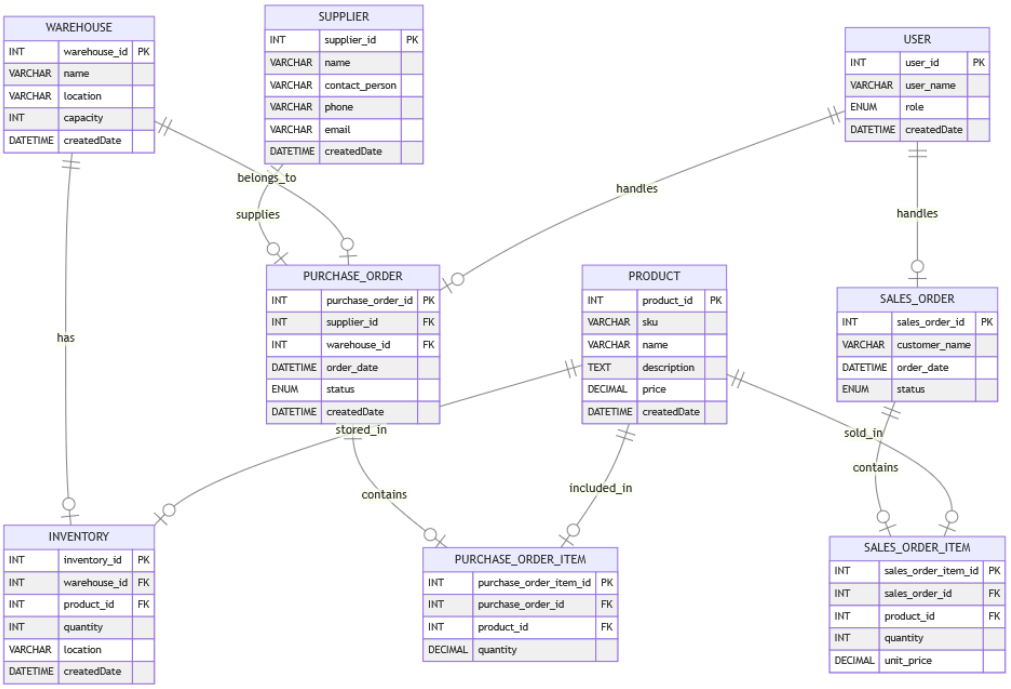
3. Integridad de Datos

- Uso de claves foráneas para garantizar consistencia en relaciones.
- Restricciones de unicidad y valores no nulos donde corresponda.

4. Eficiencia

- Uso de tipos de datos eficientes para almacenamiento y rendimiento.

Diagrama Entidad-Relación



CREACIÓN DE GESTIÓN DE INVENTARIO

```
CREATE DATABASE gestion_inventario;
USE gestion_inventario;

-- Tabla Bodega
CREATE TABLE warehouse (
        warehouse_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        location VARCHAR(100) NOT NULL,
        capacity INT NOT NULL,
        createdDate DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Tabla Productos
CREATE TABLE product (
        product_id INT AUTO_INCREMENT PRIMARY KEY,
        sku VARCHAR(50) UNIQUE NOT NULL,
        name VARCHAR(100) NOT NULL,
        description TEXT NOT NULL,
        price DECIMAL(10,2) NOT NULL,
        createdDate DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Tabla Inventario
CREATE TABLE inventory (
        inventory_id INT AUTO_INCREMENT PRIMARY KEY,
        warehouse_id INT NOT NULL,
        product_id INT NOT NULL,
        quantity INT NOT NULL DEFAULT 0,
        location VARCHAR(100) NOT NULL,
        createdDate DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (warehouse_id) REFERENCES warehouse(warehouse_id),
        FOREIGN KEY (product_id) REFERENCES product(product_id)
```

```
);

-- Tabla Proveedores
CREATE TABLE supplier (
        supplier_id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(150),
        contact_person VARCHAR(150),
        phone VARCHAR(25),
        email VARCHAR(100),
        createdDate DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Tabla Ordenes de Compra
CREATE TABLE purchase_order (
        purchase_order_id INT AUTO_INCREMENT PRIMARY KEY,
        supplier_id INT NOT NULL,
        warehouse_id INT NOT NULL,
        order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        status ENUM('Pendiente', 'Recibida', 'Cancelada') DEFAULT 'Pendiente',
        createdDate DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (supplier_id) REFERENCES supplier(supplier_id),
        FOREIGN KEY (warehouse_id) REFERENCES warehouse(warehouse_id)
);

-- Tabla artículos de compra
CREATE TABLE purchase_order_item (
        purchase_order_item_id INT AUTO_INCREMENT PRIMARY KEY,
        purchase_order_id INT NOT NULL,
        product_id INT NOT NULL,
        quantity DECIMAL(10,2) NOT NULL,
        FOREIGN KEY (purchase_order_id) REFERENCES
purchase_order(purchase_order_id),
        FOREIGN KEY (product_id) REFERENCES product(product_id)
);

-- Tabla Ordenes de Venta
```

```
CREATE TABLE sales_order (
        sales_order_id INT AUTO_INCREMENT PRIMARY KEY,
        customer_name VARCHAR(200) NOT NULL,
        order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        status ENUM('Pendiente', 'Enviada', 'Cancelada') DEFAULT 'Pendiente',
        createdDate DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Tabla de Artículos de la Orden de Venta
CREATE TABLE sales_order_item (
        sales_order_item_id INT AUTO_INCREMENT PRIMARY KEY,
        sales_order_id INT NOT NULL,
        product_id INT NOT NULL,
        quantity INT NOT NULL,
        unit_price DECIMAL(10,2) NOT NULL,
        FOREIGN KEY (sales_order_id) REFERENCES sales_order(sales_order_id),
        FOREIGN KEY (product_id) REFERENCES product(product_id)
);

-- Tabla Usuarios
CREATE TABLE user (
        user_id INT AUTO_INCREMENT PRIMARY KEY,
        user_name VARCHAR(50),
        role VARCHAR(250) NOT NULL,
        createdDate DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Funciones y procedimientos en MySQL

Función para obtener el valor total de un producto (stock * precio)

```
DELIMITER $$

CREATE FUNCTION calculate_product_value(product_id INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total_value DECIMAL(10,2);
    DECLARE price DECIMAL(10,2);
    DECLARE stock INT;

    -- Obtener el precio del producto
    SELECT price INTO price
    FROM product
    WHERE product_id = product_id;

    -- Obtener el stock disponible del producto
    SELECT quantity INTO stock
    FROM inventory
    WHERE product_id = product_id;

    -- Calcular el valor total
    SET total_value = price * stock;

    RETURN total_value;
END $$

DELIMITER ;
```

Ejemplo:

```
SELECT calculate_product_value(1);
```

Procedimiento para eliminar un producto del inventario

```
DELIMITER $$

CREATE PROCEDURE delete_product_from_inventory(
        IN p_product_id INT,
        IN p_warehouse_id INT
)
BEGIN
        DELETE FROM inventory
        WHERE product_id = p_product_id
        AND warehouse_id = p_warehouse_id;
END $$

DELIMITER ;
```

Procedimiento para agregar o actualizar el inventario

```
DELIMITER $$
CREATE PROCEDURE add_or_update_inventory(
        IN p_warehouse_id INT,
        IN p_product_id INT,
        IN p_quantity INT,
        OUT p_result VARCHAR(255)
)
BEGIN
        DECLARE v_existing_id INT;

        -- Verifico si ya existe el producto en el almacén
        SELECT inventory_id INTO v_existing_id
        FROM inventory
        WHERE product_id = p_product_id AND warehouse_id =
p_warehouse_id;

        IF v_existing_id IS NOT NULL THEN
        -- Si ya existe, actualizo la cantidad
           UPDATE inventory
        SET quantity = quantity + p_quantity
        WHERE inventory_id = v_existing_id;
        SET p_result = 'Inventory updated successfully';
```

```
        ELSE
    -- Si no existe, inserto un nuevo registro
    INSERT INTO inventory (product_id, warehouse_id, quantity)
        VALUES (p_product_id, p_warehouse_id, p_quantity);
        SET p_result = 'Item added successfully';
        END IF;
END $$

DELIMITER ;
```

Procedimiento para procesar una venta

```
DELIMITER $$

CREATE PROCEDURE process_sale(
        IN p_user_id INT,
        IN p_product_id INT,
        IN p_warehouse_id INT,
        IN p_quantity INT,
        IN p_unitary_price DECIMAL(10,2),
        OUT p_result VARCHAR(255)
)
BEGIN
        DECLARE v_available_stock INT;
        DECLARE v_total DECIMAL(10,2);

        -- Verificar stock disponible
        SELECT quantity INTO v_available_stock
        FROM inventory
        WHERE product_id = p_product_id AND warehouse_id =
p_warehouse_id;

        IF v_available_stock IS NULL THEN
        SET p_result = 'Producto no encontrado en inventario.';
        ELSEIF v_available_stock < p_quantity THEN
        SET p_result = 'Stock insuficiente.';
        ELSE
    -- Registrar la venta en sales
        INSERT INTO sales (user_id, date, total)
        VALUES (p_user_id, NOW(), 0);
```

```
        SET @sale_id = LAST_INSERT_ID();

        -- Calcular el total de la venta
        SET v_total = p_quantity * p_unitary_price;

        -- Insertar en sales_detail
        INSERT INTO sales_detail (sale_id, product_id, quantity, unitary_price,
warehouse_id)
        VALUES (@sale_id, p_product_id, p_quantity, p_unitary_price,
p_warehouse_id);

        -- Descontar del inventario
        UPDATE inventory
        SET quantity = quantity - p_quantity
        WHERE product_id = p_product_id AND warehouse_id =
p_warehouse_id;

        -- Actualizar total en sales
        UPDATE sales
        SET total = v_total
        WHERE sale_id = @sale_id;

        SET p_result = 'Venta procesada correctamente.';
        END IF;
END $$

DELIMITER ;
```

Procedimiento para obtener inventario

```
DELIMITER $$

DROP PROCEDURE IF EXISTS get_inventory_report; $$

CREATE PROCEDURE get_inventory_report()
BEGIN
        SELECT
        p.product_id,
        p.sku,
        p.name,
        COALESCE(SUM(i.quantity), 0) AS stock,
           p.price,
        (COALESCE(SUM(i.quantity), 0) * p.price) AS total_value
        FROM product p
        LEFT JOIN inventory i ON p.product_id = i.product_id
        GROUP BY p.product_id, p.sku, p.name, p.price;
END $$

DELIMITER ;
```

Procedimiento para procesar una compra

```
DELIMITER $$

DROP PROCEDURE IF EXISTS process_purchase; $$

CREATE PROCEDURE process_purchase(
        IN p_supplier_id INT,
        IN p_warehouse_id INT,
        IN p_product_id INT,
        IN p_quantity INT,
        IN p_unitary_price DECIMAL(10,2),
        IN p_order_date DATETIME,
        OUT p_result VARCHAR(255)
)
BEGIN
        DECLARE v_purchase_order_id INT;
        DECLARE v_existing_inventory INT;
```

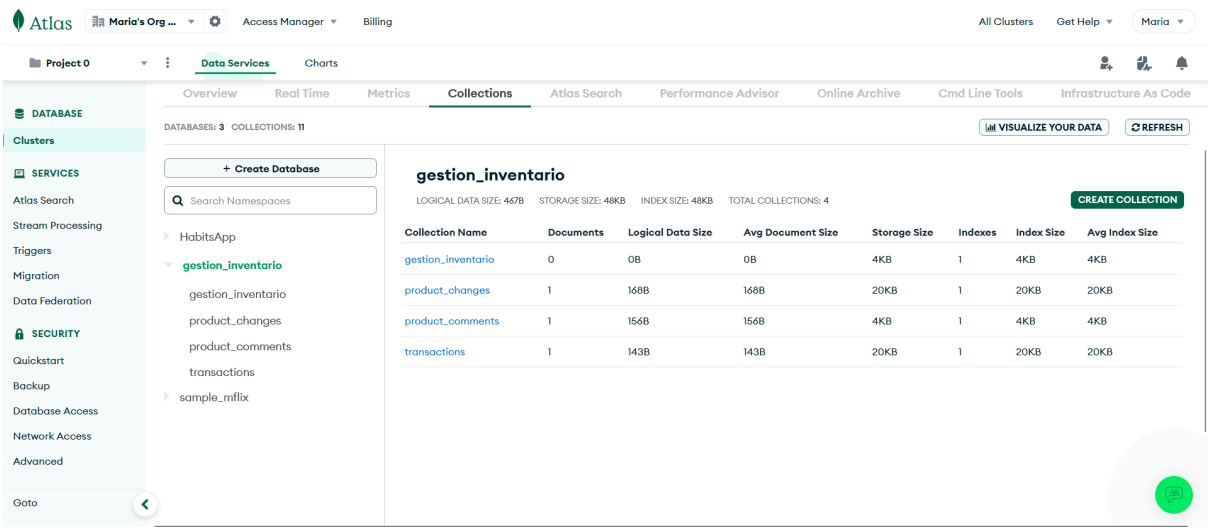
```
        -- Crear la orden de compra
        INSERT INTO purchase_order (supplier_id, warehouse_id, order_date,
total)
        VALUES (p_supplier_id, p_warehouse_id, p_order_date, 0);

        SET v_purchase_order_id = LAST_INSERT_ID(); -- Obtener el ID de la
orden de compra creada

        -- Insertar el detalle de compra
        INSERT INTO purchase_order_item (purchase_order_id, product_id,
quantity, unitary_price)
        VALUES (v_purchase_order_id, p_product_id, p_quantity,
p_unitary_price);

        -- Verificar si el producto ya existe en el inventario del almacén
        SELECT inventory_id INTO v_existing_inventory
        FROM inventory
        WHERE product_id = p_product_id AND warehouse_id =
p_warehouse_id;

        IF v_existing_inventory IS NOT NULL THEN
        -- Si ya existe, actualizar la cantidad sumándola
        UPDATE inventory
        SET quantity = quantity + p_quantity
        WHERE inventory_id = v_existing_inventory;
        ELSE
        -- Si no existe, agregarlo al inventario
        INSERT INTO inventory (product_id, warehouse_id, quantity)
        VALUES (p_product_id, p_warehouse_id, p_quantity);
        END IF;

        -- Actualizar el total en purchase_order
        UPDATE purchase_order
        SET total = (p_quantity * p_unitary_price)
        WHERE purchase_order_id = v_purchase_order_id;

        SET p_result = 'Compra procesada correctamente.';
END $$

DELIMITER ;
```

Trigger para actualizar el valor total en la tabla `sales_order_item` después de realizar una venta:

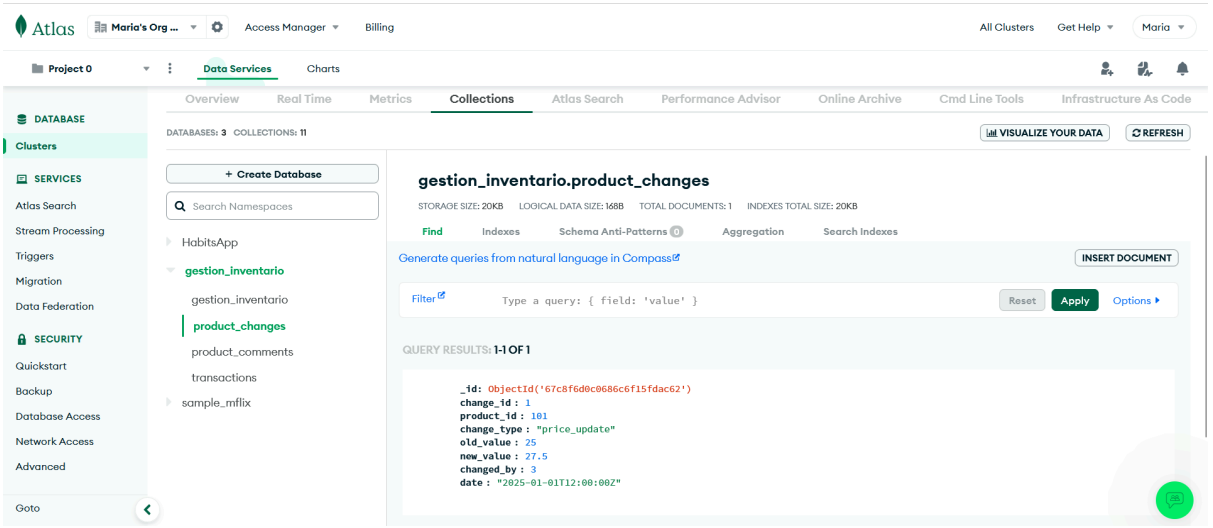
```
DELIMITER $$

CREATE TRIGGER after_sales_order_item_insert
AFTER INSERT ON sales_order_item
FOR EACH ROW
BEGIN
	DECLARE total_value DECIMAL(10,2);

	-- Calcular el valor total (precio unitario * cantidad)
	SET total_value = NEW.unit_price * NEW.quantity;

	-- Actualizar el valor total en la orden de venta
	UPDATE sales_order
	SET total_value = total_value + total_value
	WHERE sales_order_id = NEW.sales_order_id;
END $$

DELIMITER ;
```

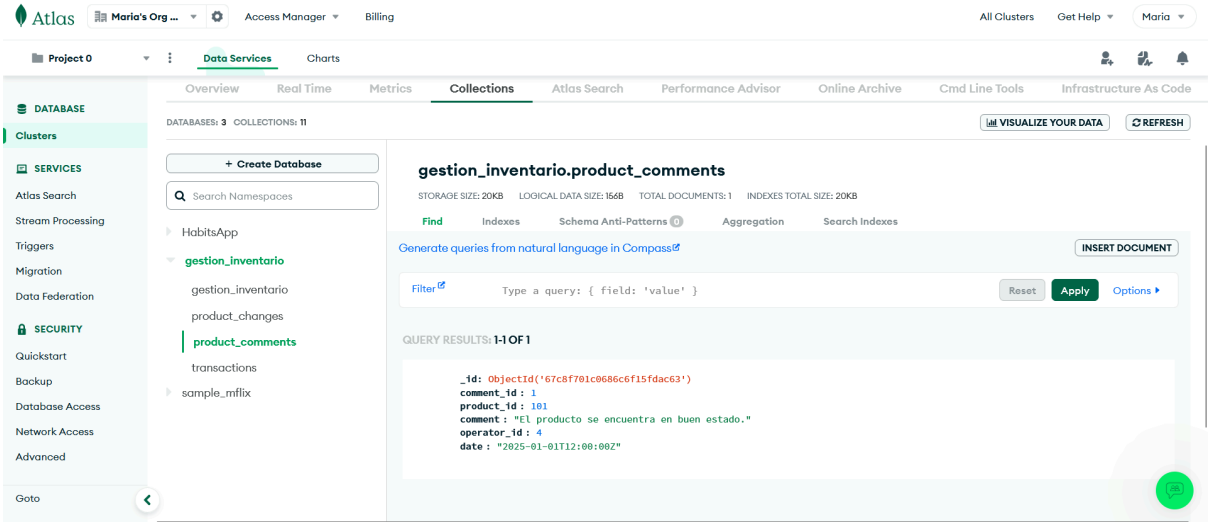
Configuración de bases de datos en MongoDB

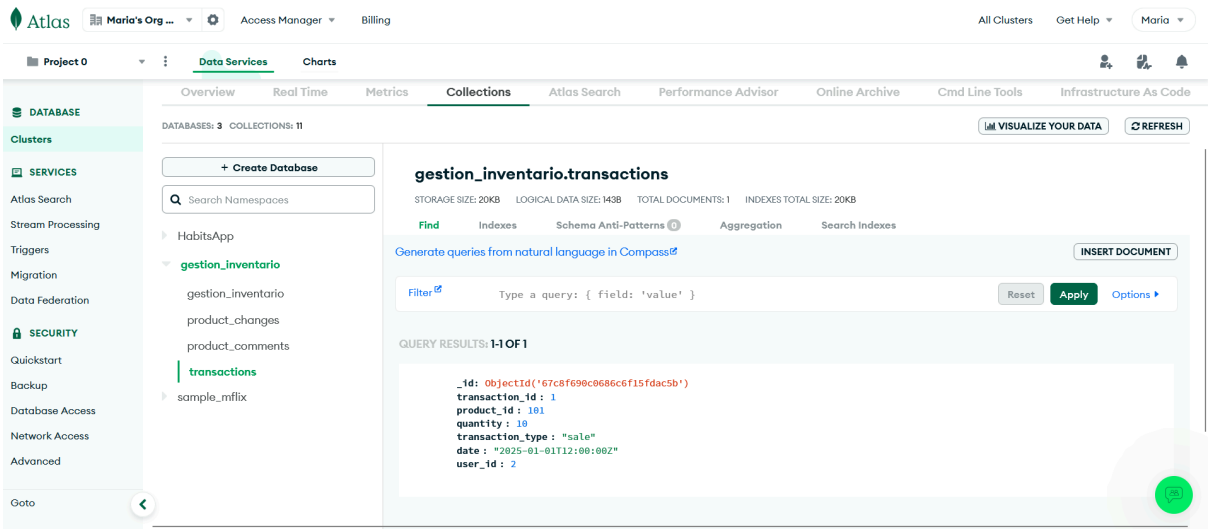
Creación de colecciones
product_changes
product_comments
transactions



Ejemplo de Documentos

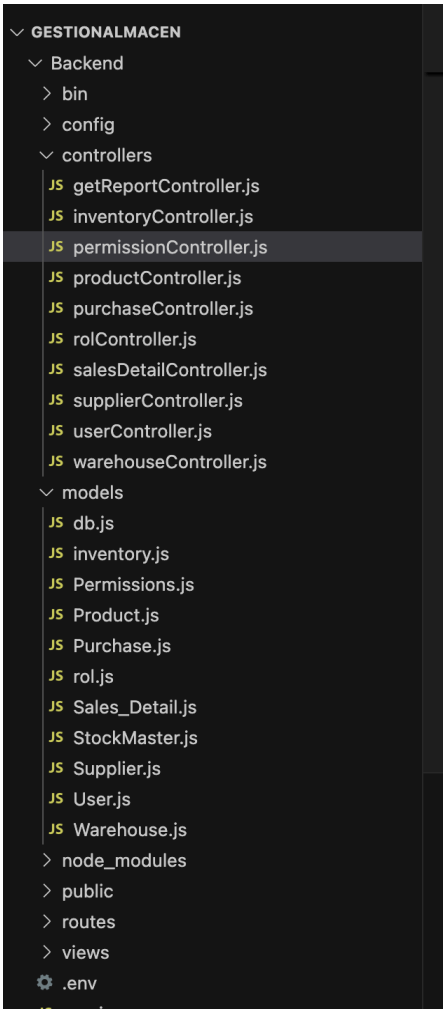




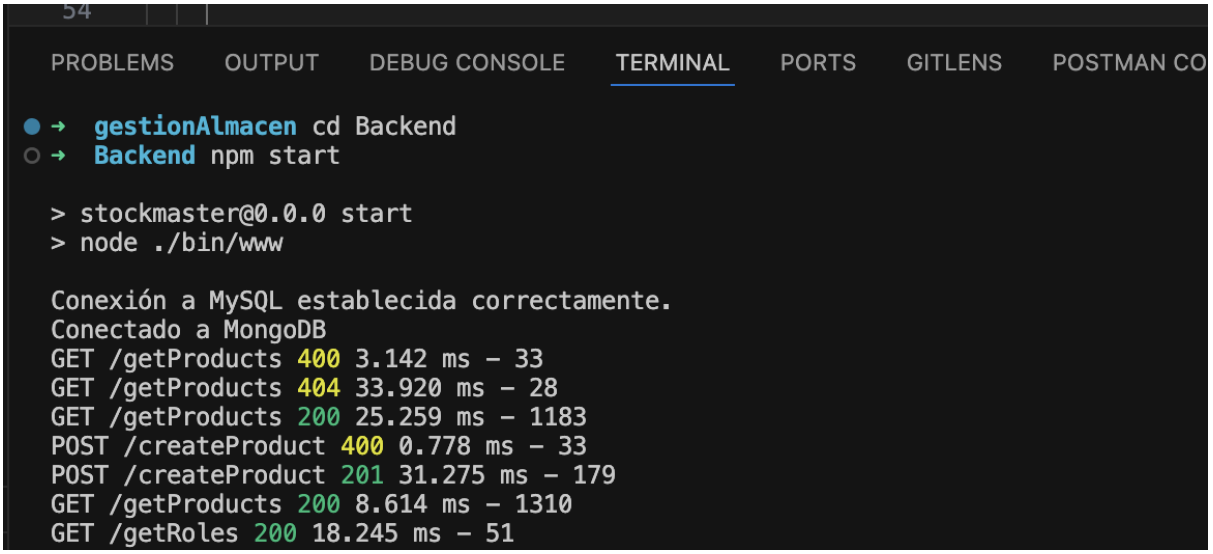


Backend

El Backend se realizó con Node.js y Express, utilizando Modelos y Controladores, para la creación de los endpoints.



Pruebas de conexión:



Pruebas de endpoints
Las pruebas se realizaron con Postman, incluyendo perfiles de seguridad y permisos en cada petición.

```
//NOTE - Endpoints de productos
router.post("/createProduct", checkPermission("GESTIONAR_PRODUCTOS"), createProduct);
router.get("/getProducts", checkPermission("GESTIONAR_PRODUCTOS"), getProducts);
router.delete("/deleteProduct/:product_id", checkPermission("GESTIONAR_PRODUCTOS"), deleteProducts);

//NOTE - Endpoints de almacen
router.post("/createWarehouse", checkPermission("GESTIONAR_ALMACENES"), createWarehouse);
router.get("/getWarehouse", checkPermission("GESTIONAR_ALMACENES"), getWarehouse);
router.delete("/deleteWarehouse/:warehouse_id", checkPermission("GESTIONAR_ALMACENES"), deleteWarehouse);

//NOTE - Endpoints de inventario
router.post("/addInventoryItem", checkPermission("GESTIONAR_INVENTARIO"), addInventoryItem);
router.get("/getInventory", checkPermission("GESTIONAR_INVENTARIO"), getinventory);
router.delete("/deleteInventoryItem/:product_id/:warehouse_id", checkPermission("GESTIONAR_INVENTARIO"), deleteInventoryIter
```

```
const checkPermission = (requiredAction) => {
  return async (req, res, next) => {
    try {
      const { user_id } = req.body;

      if (!user_id) {
        return res.status(400).json({ message: "User ID is required" });
      }

      const [user] = await sequelize.query(
        "SELECT rol_id FROM user WHERE user_id = ?",
        { replacements: [user_id], type: sequelize.QueryTypes.SELECT }
      );

      if (!user) {
        return res.status(404).json({ message: "User not found" });
      }

      const [permission] = await sequelize.query(
        "SELECT accion FROM permissions WHERE rol_id = ? AND accion = ?",
        { replacements: [user.rol_id, requiredAction], type: sequelize.QueryTypes.SELECT }
      );

      if (!permission) {
        return res.status(403).json({ message: "No tienes permisos para esta acción" });
      }

      next();
    } catch (error) {
      console.error("Error en la validación de permisos:", error);
      return res.status(500).json({ message: "Error en el servidor" });
    }
  };
};
```

Permisos, Roles y Usuarios

Se crearon 4 tipos de Roles:

```
"name": "UA" //Administrador
"name": "UO" //Operador de inventarios
"name": "UV" //Ventas
"name": "UAD" //Auditor
```

A ada uno de los Roles se les asignan distintos permisos:

```
"rol_id": 1,
"accion": [
  "GESTIONAR_USUARIOS",
```

```
  "GESTIONAR_PRODUCTOS",
  "GESTIONAR_VENTAS",
  "GESTIONAR_ALMACENES",
  "GESTIONAR_INVENTARIO",
  "VER_REPORTES"
]
```

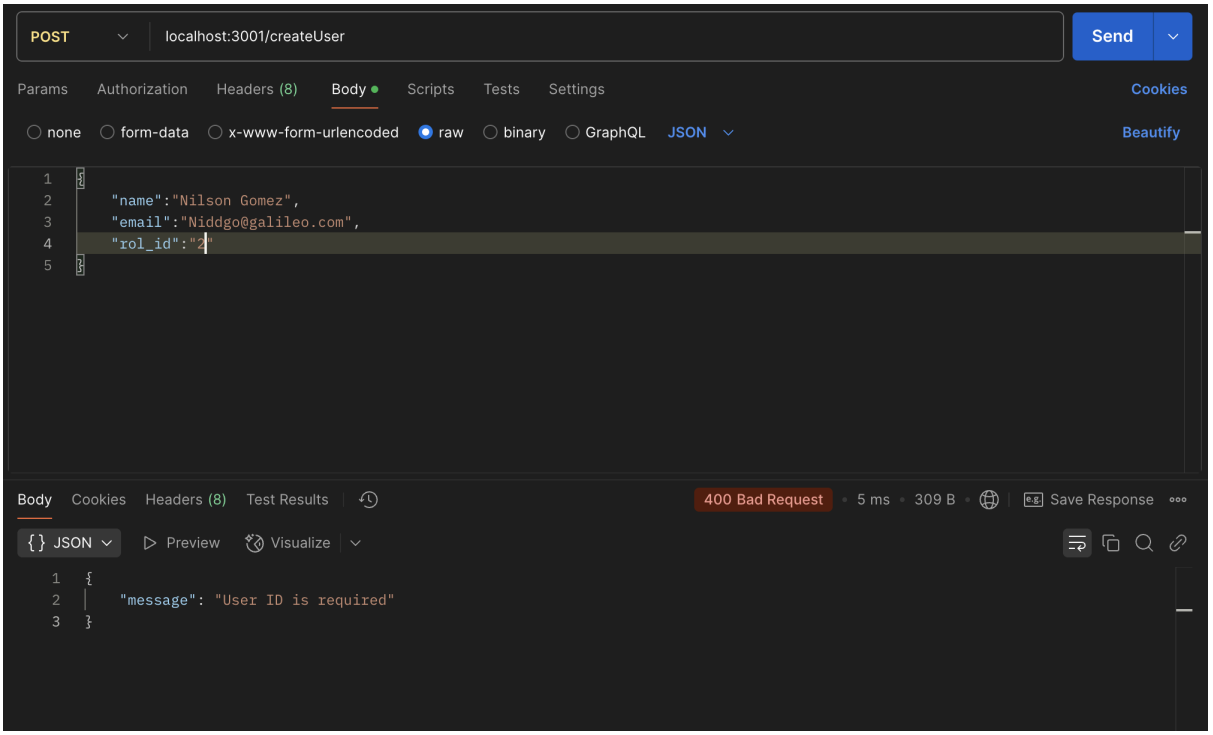
```
"rol_id": 2,
 "accion": [
  "GESTIONAR_PRODUCTOS",
  "GESTIONAR_ALMACENES",
  "GESTIONAR_INVENTARIO"
 ]
```

```
"rol_id": 3,
 "accion": [
  "GESTIONAR_VENTAS"
 ]
```

```
"rol_id": 4,
 "accion": [
  "VER_REPORTES"
 ]
```

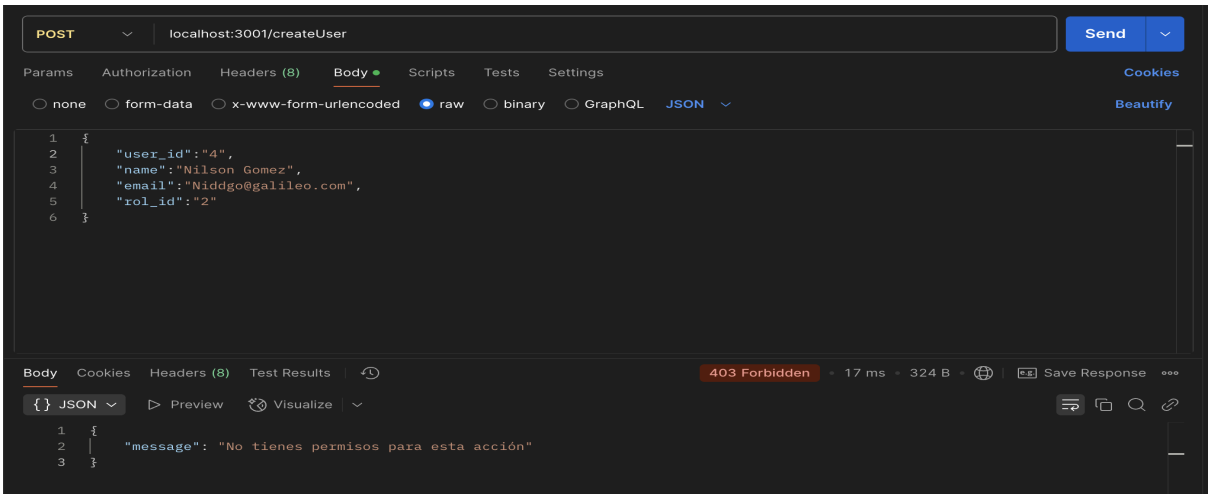
Limitando de esta manera los distintos permisos que puede realizar cada uno de los usuarios, por ejemplo, si un usuario tiene Rol 4, este no podra gestionar usuarios,, productos, ventas, etc.

Ejemplo:



En este caso, se intenta crear un usuario con Rol 2 (UV), sin embargo no se está proporcionando el user_id de quien está creando la solicitud, por lo que procedemos a incluirlo en la solicitud.
Se utiliza el user_id:4, que tiene la siguiente info:

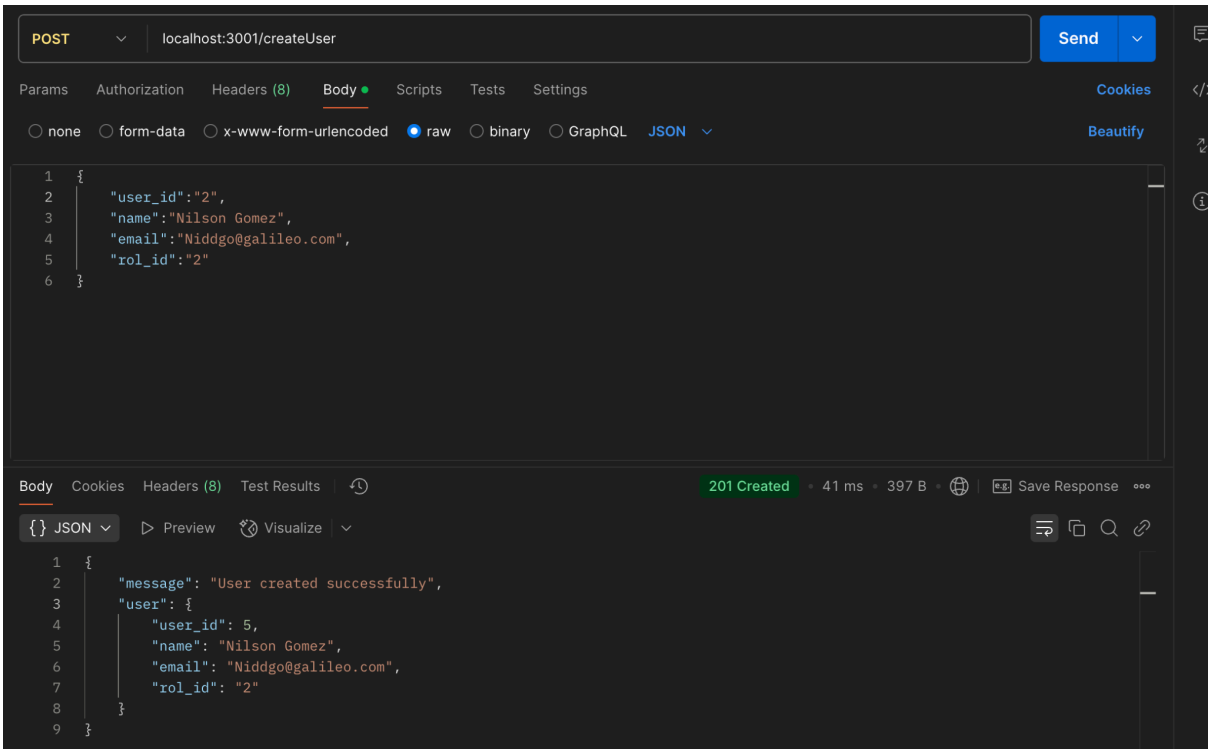

```
    {
        "user_id": 4,
        "name": "Nery Reyes",
        "email": "Guares@galileo.com",
        "rol_id": 2
    }
]
```

al ser un usuario con rol_id 4 (UAD/Usuario administrador), no tiene permisos suficientes, por lo que procedemos a hacer la solicitud con el siguiente usuario, con rol:1 (UA):


```
    {
```

```
    "user_id": 2,
    "name": "Byron Ramírez",
    "email": "bramirez@galileo.com",
    "rol_id": 1
},
```

Acá vemos el funcionamiento puntual, de los permisos y roles.

Fase II

Particiones

Partición Horizontal:
En esta partición, estamos dividiendo la tabla "sales" por año.

Proceso:
Creación de subtablas: Se crean nuevas tablas para cada partición, replicando la estructura de la tabla original (sales). En este ejemplo, se crean tres tablas (sales_2023, sales_2024, sales_2025) para los años 2023, 2024 y 2025, respectivamente. Se realiza de esta forma ya que nuestra tabla "sales" contiene llaves foráneas lo que imposibilita la creación de las particiones en la misma tabla.

Ventaja:
Mejora la gestión y el rendimiento en bases de datos con grandes volúmenes de datos. Optimiza consultas específicas (por ejemplo, consultas por año) sin tener que procesar toda la tabla.

Partición Vertical:
En este caso, la división se realizó por columnas.

Proceso:
La tabla product_details almacena detalles del producto, como la descripción. Esta tabla se relaciona con la tabla product mediante una clave foránea.
La tabla sales_extra_info contiene información adicional sobre las ventas, como el método de pago, descuentos, y notas. Está vinculada con la tabla sales a través de la clave foránea sale_id.

Ventaja: Mejora la organización y facilita el manejo de datos relacionados de manera independiente.

Consultas avanzadas y reportes

```
CREATE OR REPLACE VIEW vista_inventario_general AS
SELECT
        p.product_id AS Codigo,
        p.name AS Nombre,
        SUM(i.quantity) AS Stock_Actual,
        p.price AS Precio_Unitario,
        SUM(i.quantity) * p.price AS Valor_Total
FROM product p
LEFT JOIN inventory i ON p.product_id = i.product_id
GROUP BY p.product_id, p.name, p.price;
```

```
CREATE OR REPLACE VIEW vista_productos_por_ubicacion AS
SELECT
        w.warehouse_id AS ID_Almacen,
        w.name AS Nombre_Almacen,
        w.location AS Ubicacion,
        p.product_id AS Codigo_Producto,
        p.name AS Nombre_Producto,
        i.quantity AS Cantidad
FROM inventory i
INNER JOIN warehouse w ON i.warehouse_id = w.warehouse_id
INNER JOIN product p ON i.product_id = p.product_id;
```

```
CREATE OR REPLACE VIEW vista_ventas_simuladas AS
SELECT
        s.sale_id AS ID_Venta,
        sd.product_id AS Codigo_Producto,
        p.name AS Nombre_Producto,
        sd.quantity AS Cantidad,
        s.date AS Fecha_Venta,
        sd.quantity * sd.unitary_price AS Valor_Total
FROM sales s
INNER JOIN sales_detail sd ON s.sale_id = sd.sale_id
INNER JOIN product p ON sd.product_id = p.product_id;
```

Gestión de usuarios y seguridad

Para garantizar un acceso controlado y seguro al sistema, se implementa un modelo de gestión de usuarios basado en roles y permisos. Cada usuario está asociado a un rol específico, como administrador, operador o supervisor. A su vez, cada rol tiene un conjunto definido de acciones permitidas, que se almacenan en la tabla permissions.

Esta implementación permite:

- Restringir el acceso a determinadas funciones del sistema según el tipo de usuario.
- Mejorar la seguridad y trazabilidad, evitando que usuarios no autorizados accedan o modifiquen información crítica.
- Facilitar la escalabilidad del sistema mediante la adición de nuevos roles y permisos sin necesidad de reestructurar el código base.

Además, las credenciales de los usuarios (especialmente las contraseñas) se manejan de forma segura utilizando técnicas de cifrado como bcrypt, y se utiliza JWT para la autenticación basada en tokens.

Almacenamiento NoSQL

Se configura una base de datos NoSQL, específicamente MongoDB, para el manejo de registros históricos de transacciones. Esta decisión se justifica por las siguientes razones:

- MongoDB permite almacenar grandes volúmenes de datos no estructurados de forma eficiente.
- Los documentos en formato JSON permiten mayor flexibilidad en los registros, ideal para transacciones con diferentes estructuras o niveles de detalle.
- Facilita consultas rápidas por fecha, usuario o tipo de transacción sin necesidad de relaciones complejas.

Para simular los registros históricos:

- Se crea una API que permite insertar y consultar transacciones históricas en MongoDB.
- Las funcionalidades son probadas utilizando la herramienta POSTMAN, la cual permite enviar solicitudes HTTP y verificar la respuesta de la API.

Esto permite tener una separación clara entre los datos operacionales (almacenados en MySQL) y los datos históricos o de auditoría (almacenados en MongoDB), mejorando el rendimiento y facilitando el análisis de datos a largo plazo.

Campos críticos para encriptación y justificación

En el esquema de base de datos, los siguientes campos son considerados sensibles y deben protegerse mediante cifrado o medidas de seguridad:

Tabla	Campo	Motivo de encriptación o protección
user	email	Información personal identificable (PII).
user	name	Información personal del usuario.
user	password	Clave de acceso, debe ser almacenada cifrada.
supplier	contact_person	Información personal del proveedor.
supplier	phone y email	PII de proveedores, puede usarse para contacto no autorizado.

Uso de Bcrypt para encriptación de contraseñas

Bcrypt es una función de hashing adaptativa diseñada para proteger contraseñas. Genera un hash irreversible y seguro, incluso si la base de datos es comprometida.

Proceso paso a paso:

1. Usuario crea cuenta o cambia contraseña:
    - *Se usa bcrypt.hash(password, saltRounds) para generar un hash de la contraseña.*
2. Se almacena el hash en la base de datos.
3. Cuando el usuario inicia sesión:
    - *Se usa bcrypt.compare(inputPassword, storedHash) para validar.*

Ventajas:

1. Genera un salt aleatorio para cada contraseña.
2. Resiste ataques de diccionario y fuerza bruta.
3. El algoritmo puede hacerse más lento aumentando los saltRounds, adaptándose a la evolución de los ataques.

JWT (JSON Web Token) para autenticación

JWT es un estándar abierto (RFC 7519) que permite transmitir información de forma segura entre partes como un objeto JSON.

Proceso paso a paso:

1. El usuario se autentica (email + contraseña).

2. El backend valida la contraseña con bcrypt.

3. Se genera un JWT usando una librería como jsonwebtoken, por ejemplo:

```
const token = jwt.sign({ userId: user.id }, 'secreto', { expiresIn: '1h' });
```

4. El token es enviado al cliente (navegador o app).

5. El cliente incluye el token en cada request, usando headers:

```
Authorization: Bearer <token>
```

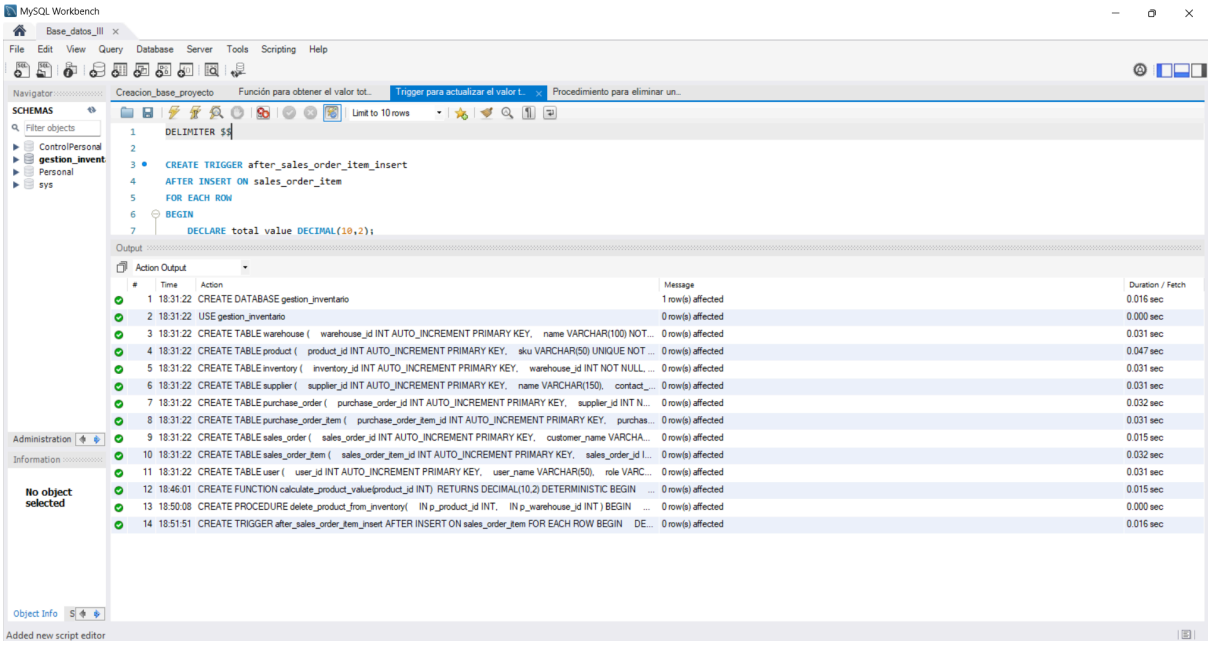
6. El backend valida el token en cada petición con:

```
const decoded = jwt.verify(token, 'secreto');
```

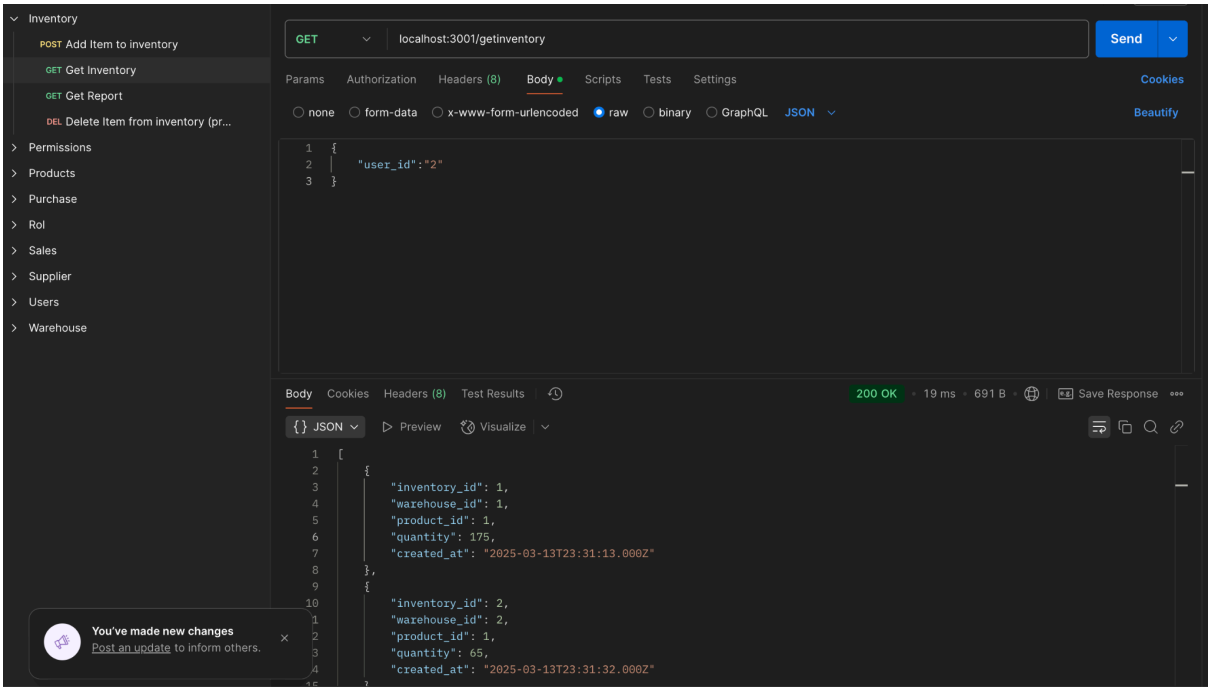
Ventajas:

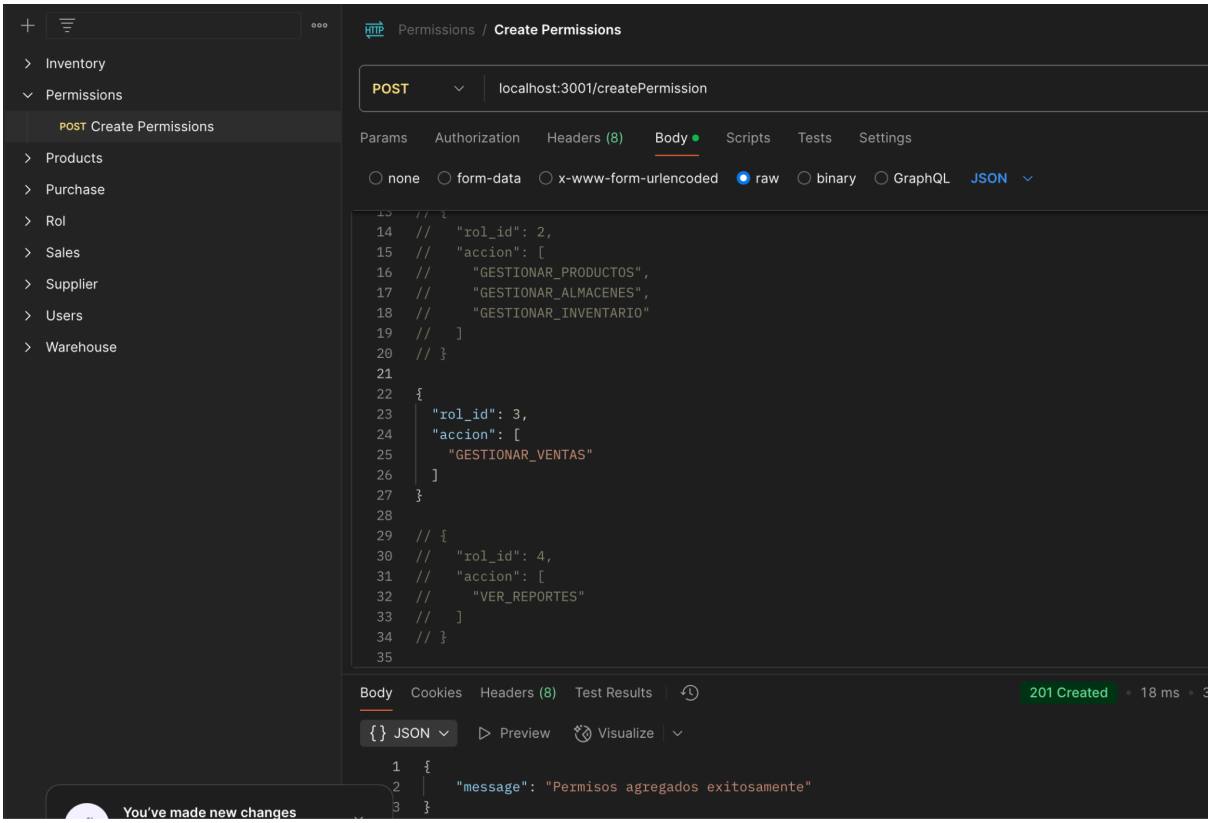
- Stateless: no requiere mantener sesión en el servidor.
- Seguro si se usa HTTPS.
- Permite definir expiración (exp) y payload personalizado.

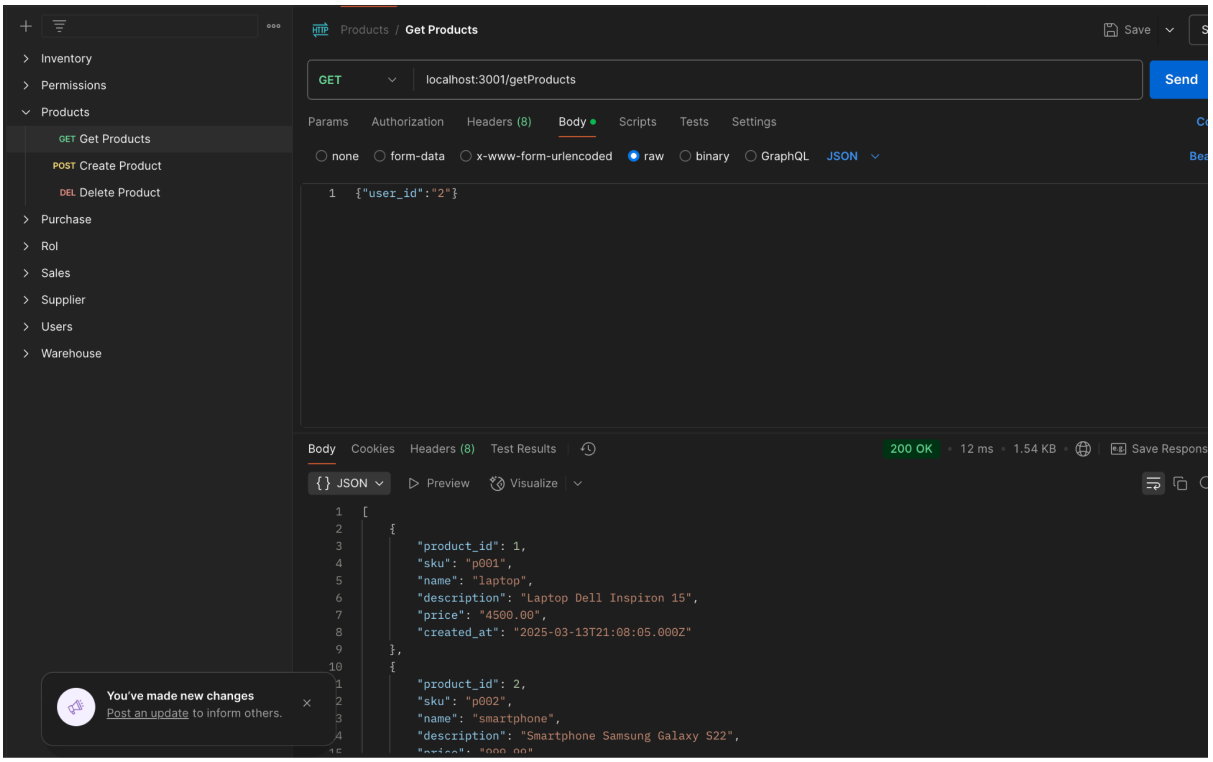
Apéndice

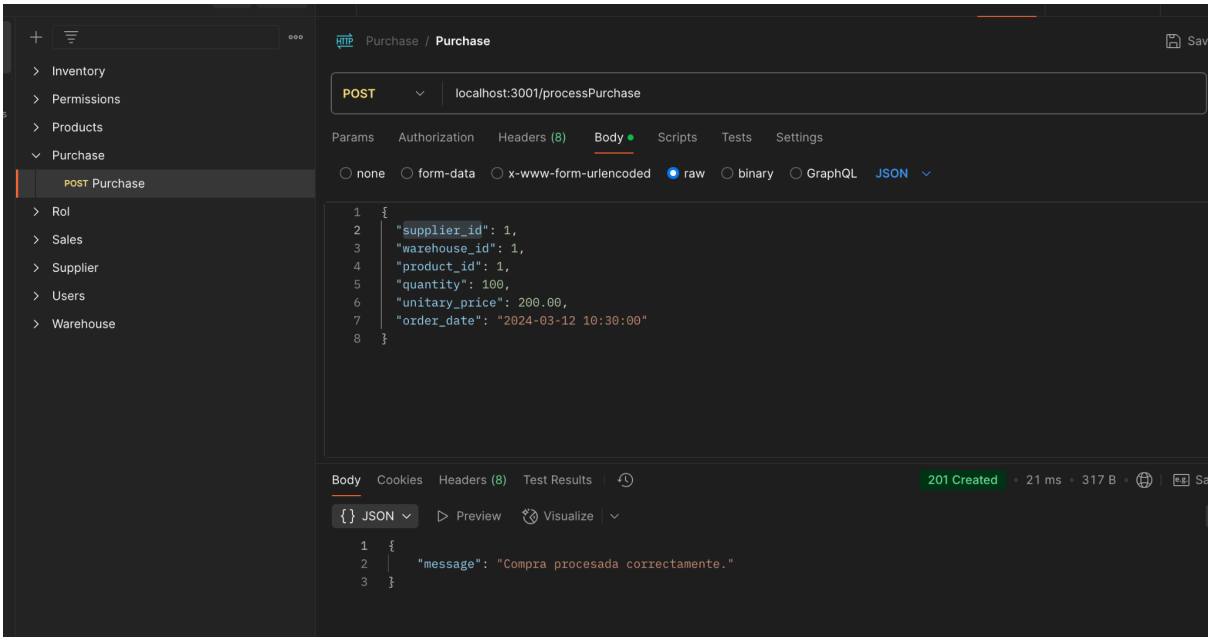


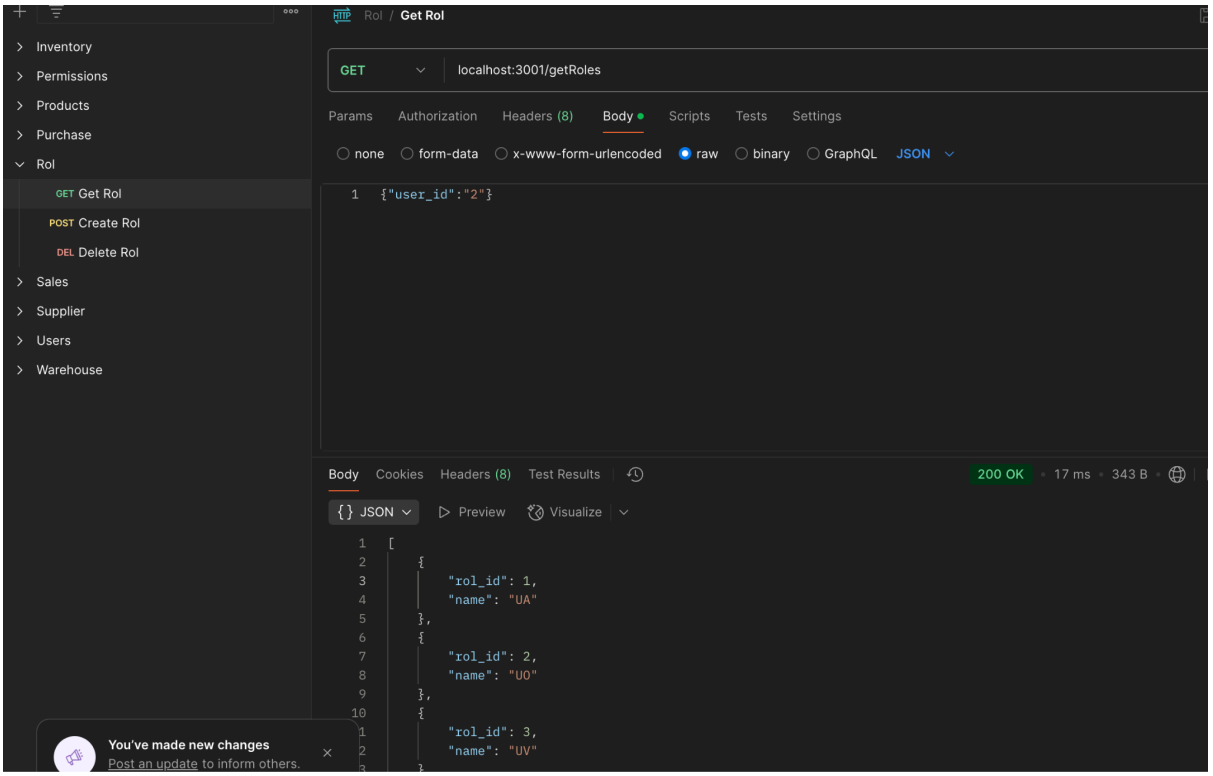
Ejemplos con postman:

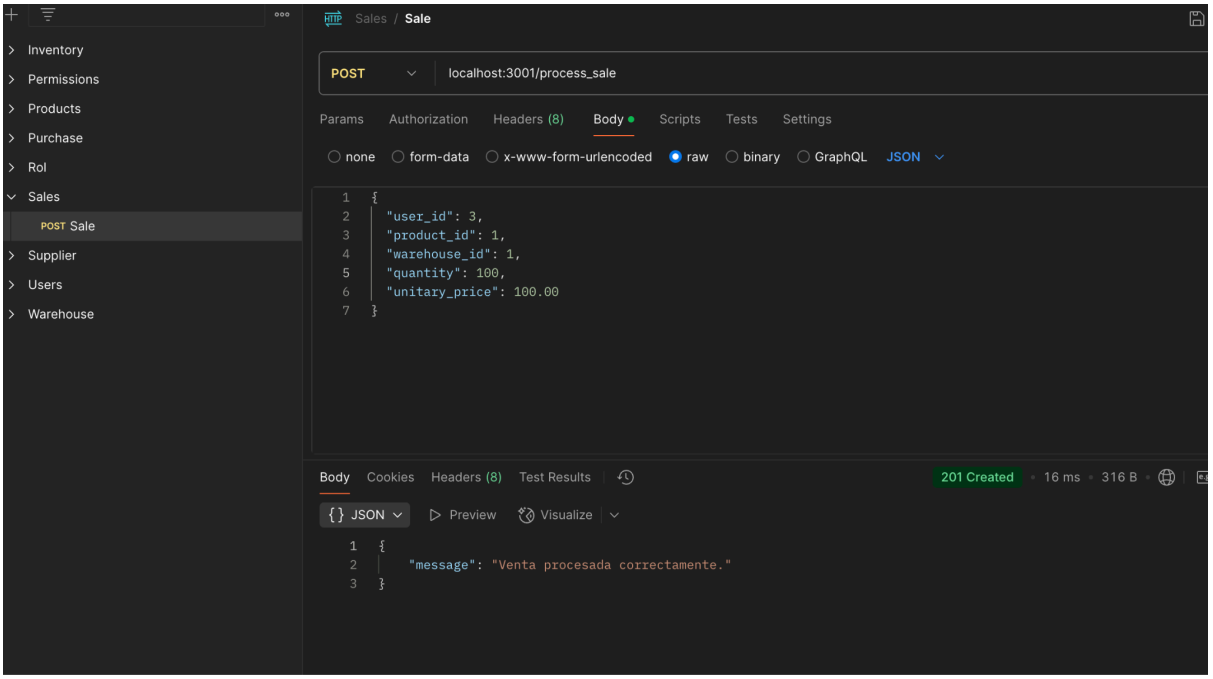

Inventory
POST Add Item to inventory
GET Get Inventory
GET Get Report
DEL Delete Item from inventory (pr...
Permissions
Products
Purchase
RoI
Sales
Supplier
Users
Warehouse
GET
localhost:3001/getinventory
Send
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"user_id":"2"
}
Body
Cookies
Headers (8)
Test Results
200 OK
19 ms
691 B
Save Response
JSON
Preview
Visualize
[
{
"inventory_id": 1,
"warehouse_id": 1,
"product_id": 1,
"quantity": 175,
"created_at": "2025-03-13T23:31:13.000Z"
},
{
"inventory_id": 2,
"warehouse_id": 2,
"product_id": 1,
"quantity": 65,
"created_at": "2025-03-13T23:31:32.000Z"
You've made new changes
Post an update to inform others.

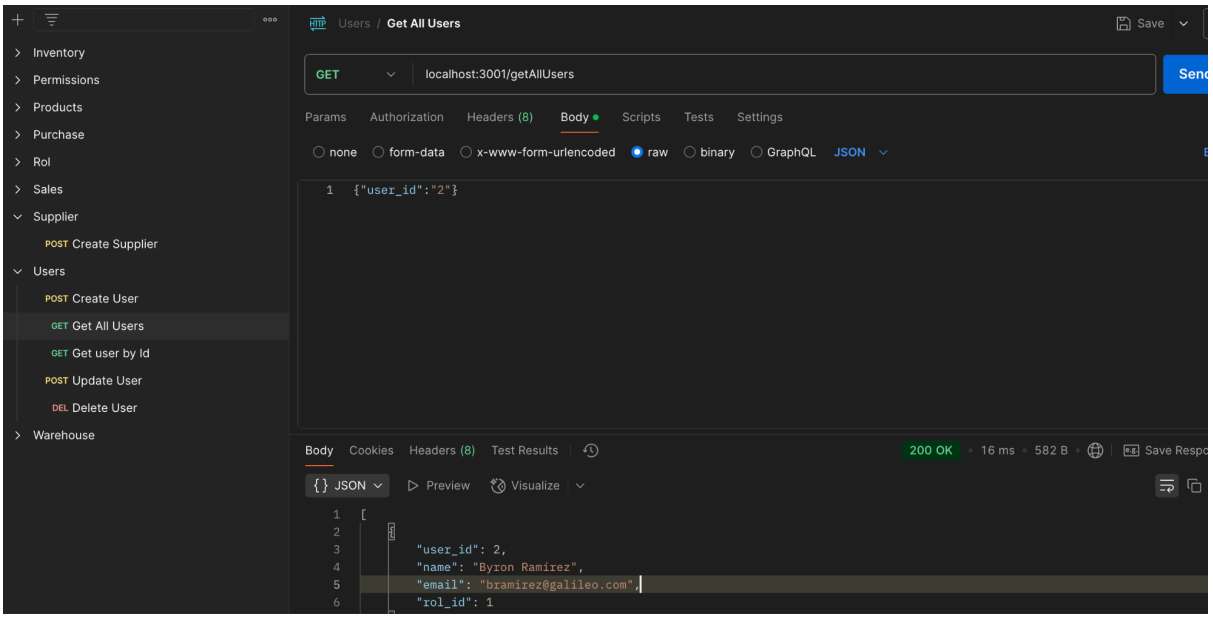

Inventory
Permissions
POST Create Permissions
Products
Purchase
Rol
Sales
Supplier
Users
Warehouse
Permissions / Create Permissions
POST
localhost:3001/createPermission
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
// {
// "rol_id": 2,
// "accion": [
// "GESTIONAR_PRODUCTOS",
// "GESTIONAR_ALMACENES",
// "GESTIONAR_INVENTARIO"
//]
// }
{
"rol_id": 3,
"accion": [
"GESTIONAR_VENTAS"
]
}
// {
// "rol_id": 4,
// "accion": [
// "VER_REPORTES"
//]
// }
Body
Cookies
Headers (8)
Test Results
201 Created
18 ms
{ } JSON
Preview
Visualize
{
"message": "Permisos agregados exitosamente"
}
You've made new changes


Inventory
Permissions
Products
GET Get Products
POST Create Product
DEL Delete Product
Purchase
Rol
Sales
Supplier
Users
Warehouse
Products / Get Products
Save
GET
localhost:3001/getProducts
Send
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{"user_id":"2"}
Body
Cookies
Headers (8)
Test Results
200 OK
12 ms
1.54 KB
Save Respons
{ } JSON
Preview
Visualize
[
{
"product_id": 1,
"sku": "p001",
"name": "laptop",
"description": "Laptop Dell Inspiron 15",
"price": "4500.00",
"created_at": "2025-03-13T21:08:05.000Z"
},
{
"product_id": 2,
"sku": "p002",
"name": "smartphone",
"description": "Smartphone Samsung Galaxy S22",
You've made new changes
Post an update to inform others.


Inventory
Permissions
Products
Purchase
POST Purchase
Rol
Sales
Supplier
Users
Warehouse
Purchase / Purchase
POST
localhost:3001/processPurchase
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"supplier_id": 1,
"warehouse_id": 1,
"product_id": 1,
"quantity": 100,
"unitary_price": 200.00,
"order_date": "2024-03-12 10:30:00"
}
Body
Cookies
Headers (8)
Test Results
201 Created
21 ms
317 B
JSON
Preview
Visualize
{
"message": "Compra procesada correctamente."
}


Inventory
Permissions
Products
Purchase
Rol
GET Get Rol
POST Create Rol
DEL Delete Rol
Sales
Supplier
Users
Warehouse
Rol / Get Rol
GET
localhost:3001/getRoles
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{"user_id":"2"}
Body
Cookies
Headers (8)
Test Results
200 OK
17 ms
343 B
JSON
Preview
Visualize
[
{
"rol_id": 1,
"name": "UA"
},
{
"rol_id": 2,
"name": "UO"
},
{
"rol_id": 3,
"name": "UV"
You've made new changes
Post an update to inform others.


Inventory
Permissions
Products
Purchase
Rol
Sales
POST Sale
Supplier
Users
Warehouse
Sales / Sale
POST
localhost:3001/process_sale
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"user_id": 3,
"product_id": 1,
"warehouse_id": 1,
"quantity": 100,
"unitary_price": 100.00
}
Body
Cookies
Headers (8)
Test Results
201 Created
16 ms
316 B
JSON
Preview
Visualize
{
"message": "Venta procesada correctamente."
}


Inventory
Permissions
Products
Purchase
Rol
Sales
Supplier
POST Create Supplier
Users
POST Create User
GET Get All Users
GET Get user by Id
POST Update User
DEL Delete User
Warehouse
Users / Get All Users
Save
GET
localhost:3001/getAllUsers
Send
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{"user_id":"2"}
Body
Cookies
Headers (8)
Test Results
200 OK
16 ms
582 B
Save Respo
JSON
Preview
Visualize
[
{
"user_id": 2,
"name": "Byron Ramirez",
"email": "bramirez@galileo.com",
"rol_id": 1