

# **Pontificia Universidad Católica del Ecuador**

## **Proyecto Integrador ECOCANJE**

### **Tecnología en Desarrollo de Software**

#### **Integrantes:**

- Cristian Pareja
- Franklin Toro

**Fecha:** 07 de Enero de 2026

#### **1. Introducción**

- **Contexto del problema**

En la actualidad, la gestión inadecuada de residuos reciclables representa un problema ambiental significativo en la ciudad de Quito. Aunque muchas personas están dispuestas a reciclar, existen barreras como la falta de incentivos, el desconocimiento de procesos adecuados y la ausencia de plataformas digitales que faciliten el intercambio responsable de materiales reciclables. Esto provoca que grandes cantidades de residuos reutilizables terminen contaminando el entorno.

Adicionalmente, las iniciativas de reciclaje suelen carecer de sistemas tecnológicos que permitan registrar, controlar y dar seguimiento a los intercambios realizados, lo que limita su alcance y sostenibilidad a largo plazo.

- **Qué es ECOCANJE**

ECOCANJE v1.0 es una aplicación orientada a la gestión de intercambio de productos reciclables, que permite registrar, administrar y controlar productos dentro de un sistema digital. El proyecto busca apoyar iniciativas de economía circular mediante una plataforma tecnológica que facilite la gestión de productos reciclables.

ECOCANJE v1.0 se implementa como una aplicación basada en una arquitectura por capas, que expone una API REST para la administración de productos y operaciones relacionadas, garantizando orden y escalabilidad.

- **A quién va dirigido**

El sistema ECOCANJE en su versión inicial 1.0 está dirigido a estudiantes de la PUCE interesados en participar en procesos de reciclaje y reutilización de productos.

- **Alcance del proyecto**

- Gestión de productos mediante operaciones CRUD (crear, listar, actualizar y eliminar).
- Exposición de una API REST para el acceso a la información de productos.
- Validación básica de datos para garantizar la integridad de la información.
- Conexión a una base de datos PostgreSQL para el almacenamiento persistente.
- Uso de una arquitectura por capas que separa responsabilidades entre rutas, servicios y repositorios.
- Pruebas de funcionamiento de la API mediante herramientas como Postman.

**En esta fase del proyecto, no se contempla:**

- Desarrollo de una interfaz gráfica frontend completa.
- Implementación de doble autenticación de usuarios finales.
- Integración con sistemas externos de pago o logística.
- Despliegue en un entorno de producción real.

Estas funcionalidades quedan previstas como posibles mejoras o futuras versiones del sistema.

## **2. Tecnologías Utilizadas**

- **Lenguaje de Programación**

### **JavaScript (Node.js)**

Se utiliza JavaScript como lenguaje principal del backend mediante el entorno de ejecución Node.js, lo que permite construir aplicaciones del lado del servidor de forma eficiente, asincrónica y con alto rendimiento.

- **Backend**

### **Express.js (Framework)**

Express es un framework minimalista para Node.js que facilita la creación de servidores web y APIs REST. En el proyecto ECOCANJE se utiliza para:

- Definir rutas HTTP (GET, POST, PUT, DELETE).
- Manejar solicitudes y respuestas del cliente.
- Organizar la aplicación siguiendo una arquitectura por capas.

- **Base de Datos**

### **PostgreSQL**

PostgreSQL es el sistema gestor de base de datos relacional utilizado para almacenar la información del sistema, destacando por su confiabilidad, soporte para integridad de datos y escalabilidad.

- **Contenedores**

### **Docker y Docker Compose**

Utilizamos Docker para contener la base de datos PostgreSQL, permitiendo un entorno de desarrollo consistente y facilitar el despliegue y configuración.

Docker Compose utilizamos para administrar los contenedores necesarios del sistema.

- **Acceso a data**

### **Sequelize (ORM)**

Sequelize se utiliza como herramienta ORM para mapear los modelos de la base de datos a objetos JavaScript, facilitando el acceso y la manipulación de datos desde el backend.

### **pg (node-postgres)**

La librería pg permite ejecutar consultas SQL directas sobre PostgreSQL, ofreciendo flexibilidad y control en determinadas operaciones.

- **Frontend**

### **React (Biblioteca)**

React se utiliza para el desarrollo del frontend de ECOCANJE, permitiendo la creación de interfaces de usuario dinámicas y reutilizables mediante componentes.

## **Tailwind CSS (Framework)**

Tailwind CSS se emplea para el diseño visual de la aplicación, proporcionando clases utilitarias que permiten construir interfaces modernas, responsivas y consistentes sin necesidad de escribir grandes hojas de estilo personalizadas.

- **Arquitectura**

### **Arquitectura por Capas**

El sistema se estructura en capas (rutas, servicios y repositorios), lo que mejora la mantenibilidad y escalabilidad del proyecto.

- **Seguridad de la Información**

### **Bcrypt**

Se utiliza la librería bcrypt para el hash de contraseñas de los usuarios, garantizando que las credenciales no se almacenen en texto plano. Y protegiendo la información sensible.

- **Auditoría del Sistema**

Se implementan mecanismos de auditoría a nivel de base de datos mediante funciones y triggers, permitiendo registrar acciones como creación, actualización y eliminación de registros, junto con información del usuario y la fecha de la operación.

- **Respaldo de Información**

### **Cron (tareas programadas en Linux)**

Se emplea el programador de tareas cron para automatizar respaldos periódicos de la base de datos PostgreSQL. Esta automatización garantiza la disponibilidad de copias de seguridad sin intervención manual.

- **Herramientas de pruebas**

#### **Postman**

Postman se utiliza para probar los endpoints de la API REST, validando el funcionamiento de las operaciones y el manejo de errores.

- **Control de versiones**

#### **Git**

Git se utiliza para el control de versiones del código fuente, permitiendo el seguimiento de cambios durante el desarrollo del proyecto.

### **3. Manual de Instalación y Ejecución del Sistema**

El presente manual describe los requisitos, pasos de instalación y ejecución necesarios para el funcionamiento del sistema **ECOCANJE**. Está dirigido a usuarios técnicos, docentes y estudiantes que requieran ejecutar el sistema en un entorno local.

#### **3.1 Requisitos del Sistema**

- Sistema operativo: Windows, Linux o macOS
- **Node.js** (versión 18 o superior)
- **npm** (incluido con Node.js)
- **Docker y Docker Compose**
- **Git**
- Navegador web (Google Chrome, Firefox u otro)
- Herramienta para pruebas de API: **Postman**

## 3.2 Clonar el Proyecto

- Abrir una terminal o consola.
- Clonar el repositorio del proyecto utilizando Git:

```
git clone https://github.com/CristianPareja/Proyecto_final_3_Pareja_Toro.git
```

- Acceder al directorio del proyecto:

```
cd Proyecto_final_3_Pareja_Toro
```

## 3.3 Configuración y Ejecución de la Base de Datos

- Verificar que Docker esté en ejecución.
- Desde la raíz del proyecto, ejecutar:

```
docker compose up -d
```

- Confirmar que los contenedores estén activos:

```
docker ps
```

```
PS C:\Users\crist\OneDrive\Escritorio\PUCE\DesarrolloWeb\Proyecto_final_ECOCANJE> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                                NAMES
2da24d912d9f   proyecto_final_ecocanje-backup-cron  "docker-entrypoint.s..." 21 hours ago  Up About an hour  5432/tcp                                ecocanje_backup_cron
17db753dd7e1   postgres:15                          "docker-entrypoint.s..." 21 hours ago  Up About an hour  0.0.0.0:5435->5432/tcp, [::]:5435->5432/tcp  express_003_postgres
```

Este proceso levantará PostgreSQL con su configuración personalizada, el servicio de cron para respaldos automáticos y los scripts de inicialización de la base de datos con sequelize.

### 3.4 Configuración y Ejecución del Backend

- Desde la raíz del proyecto, instalar las dependencias:

*npm install*

- Configurar el archivo .env con los datos de conexión a la base de datos y variables necesarias.

```
JWT_SECRET=ecocanje_super_secreto_123
DB_HOST=localhost
DB_PORT=5435
DB_NAME=ecocanje_db
DB_USER=postgres
DB_PASSWORD=postgres
PORT=4000
```

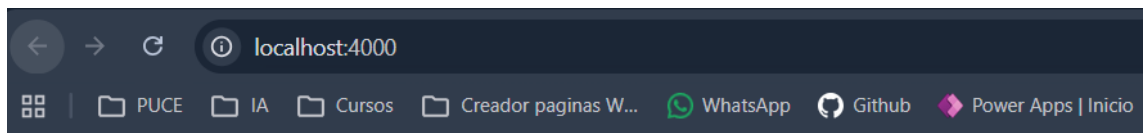
- Iniciar el servidor backend:

*node server.js*

```
PS C:\Users\cris\OneDrive\Escritorio\PUCE\DesarrolloWeb\Proyecto_final_ECOCANJE> node server.js
>>
[dotenv@17.2.3] injecting env (7) from .env -- tip: ⚙ override existing env vars with { override: true }
[dotenv@17.2.3] injecting env (0) from .env -- tip: 🔒 encrypt with Dotenvx: https://dotenvx.com
Conexión a PostgreSQL OK
Tablas creadas / sincronizadas
Servidor corriendo en http://localhost:4000
```

- El backend quedará disponible en:

<http://localhost:4000>



EcoCanje API funcionando



### 3.5 Configuración y Ejecución del Frontend

- Acceder al directorio del frontend:

```
cd ecocanje-frontend
```

- Instalar las dependencias:

```
npm install
```

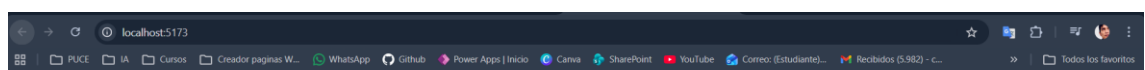
- Iniciar la aplicación frontend:


```
npm run dev
```

```
PS C:\Users\crist\OneDrive\Escritorio\PUCE\DesarrolloWeb\Proyecto_final_ECOCANJE\ecocanje-frontend> npm run dev
>>
  → Local:   http://localhost:5173/
  → Network: use --host to expose
  → press h + enter to show help
```

- Acceder al sistema desde el navegador en una URL similar a:

<http://localhost:5173>



EcoCanje

Ingresa para comprar o vender

Login

Registro

Usuario

Ej: userA

Contraseña

\*\*\*\*\*

Ingresar

### 3.6 Uso del Sistema

- El **frontend** permite al usuario interactuar con el sistema mediante una interfaz desarrollada en React y Tailwind CSS.
- El **backend** expone una API REST que gestiona la lógica del sistema.
- Las operaciones pueden ser probadas mediante herramientas como Postman o directo desde la interfaz de la app.

### 3.7 Backups Automatizados

El sistema incluye respaldos automáticos de la base de datos mediante:

- Script backup.sh
- Programación con cron
- Uso de la herramienta pg\_dump

Las copias de seguridad se almacenan en el directorio backups/

### 3.8 Detención del Sistema

Para detener el sistema:

- Detener frontend y backend con Ctrl + C.
- Apagar los contenedores Docker:

*docker compose down*

## 4. Arquitectura del Sistema

El proyecto ECOCANJE implementa una arquitectura web de tres capas, complementada con una arquitectura por capas en el backend y soportada por una infraestructura basada en contenedores Docker.

PROYECTO_FINAL_ECOCANJE	
— ecocanje-frontend	→ Frontend (React + Tailwind)
— server.js	→ Punto de entrada del backend
— routes/	→ Controladores de la API
— services/	→ Lógica de negocio
— repositories/	→ Acceso a datos
— models/	→ Modelos ORM
— middlewares/	→ Validaciones y seguridad
— database.js	→ Conexión a PostgreSQL
— docker-compose.yml	→ Orquestación de contenedores
— Dockerfile	→ Backend
— Dockerfile.cron	→ Backups automatizados
— initdb/	→ Scripts de inicialización
— postgres-conf/	→ Configuración PostgreSQL
— backups/	→ Copias de seguridad
— backup.sh	→ Script de respaldo
— crontab.txt	→ Programación de cron
— .env	→ Variables de entorno

El sistema se organiza en tres grandes capas:

1. Capa de Frontend
2. Capa de Lógica de Negocio (Backend)
3. Capa de Datos (Base de Datos)

## **4.1 Arquitectura del Frontend**

El frontend del sistema se encuentra en el directorio ecocanje-frontend y está desarrollado utilizando React y Tailwind CSS.

- Arquitectura basada en componentes reutilizables.
- Comunicación con el backend mediante solicitudes HTTP a la API REST.
- Interfaz responsiva y moderna gracias al uso de Tailwind CSS.

## **4.2 Arquitectura del Backend**

El backend de ECOCANJE está desarrollado en Node.js con Express y sigue una arquitectura por capas, organizada de la siguiente manera:

### **4.2.1 Capa de Rutas (routes)**

La carpeta routes actúa como la capa de controladores del sistema.

- Recibir las solicitudes HTTP del frontend.
- Definir los endpoints de la API REST.
- Delegar la lógica de negocio a la capa de servicios.

### **4.2.2 Capa de Servicios (services)**

La carpeta services contiene la lógica de negocio del sistema.

En esta capa se realizan:

- Validaciones de datos.
- Aplicación de reglas del sistema.

- Control de operaciones entre distintas capas.
- Lanzamiento de errores controlados.

Esta capa actúa como intermediaria entre las rutas y los repositorios.

#### **4.2.3 Capa de Repositorios (repositories)**

La carpeta repositories se encarga del acceso a los datos.

Sus funciones principales son:

- Ejecutar operaciones CRUD.
- Interactuar con la base de datos PostgreSQL.
- Abstractar el origen de los datos del resto del sistema

#### **4.2.4 Capa de Modelos (models)**

La carpeta models contiene los modelos definidos mediante Sequelize, los cuales representan las entidades de la base de datos.

- Definen la estructura de las tablas.
- Incluyen restricciones y validaciones básicas.
- Facilitan el mapeo objeto-relacional.

#### **4.2.5 Middlewares**

La carpeta middlewares agrupa funciones que se ejecutan antes o después de procesar una solicitud, tales como:

- Validaciones.
- Seguridad.

- Manejo de errores.
- Protección de rutas.

Esta capa contribuye a la robustez y seguridad del backend.

### 4.3 Arquitectura de Datos

La capa de datos está basada en PostgreSQL, desplegada mediante contenedores Docker.

- Base de datos relacional robusta.
- Configuración personalizada ubicada en postgres-conf.
- Persistencia de datos independiente del ciclo de vida de los contenedores.

### 4.4 Seguridad, Auditoría y Respaldo

- **Hash de contraseñas** para proteger credenciales.
- **Auditorías** a nivel de base de datos para registrar operaciones críticas.
- **Backups automatizados** mediante scripts y tareas programadas con cron.

Con almacenamiento de respaldos en el directorio backups.

### 4.5 Flujo General del Sistema Ecocanje

1. El usuario interactúa con el frontend.
2. El frontend envía una solicitud HTTP al backend.
3. La ruta recibe la solicitud y la envía al servicio correspondiente.
4. El servicio valida y procesa la lógica de negocio.
5. El repositorio accede a la base de datos.
6. La respuesta retorna al frontend.

## **5. Diseño de la Base de Datos**

### **5.1 Descripción General**

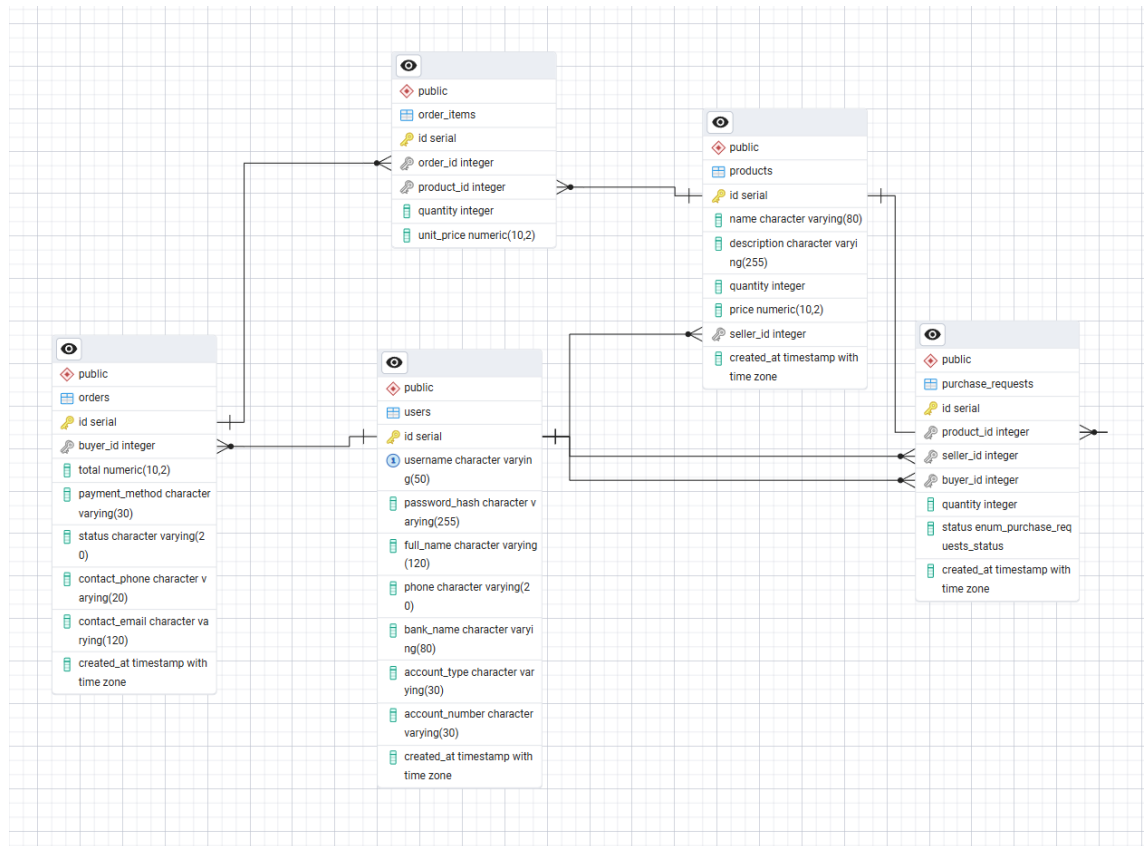
La base de datos del proyecto ECOCANJE está diseñada bajo un modelo relacional, utilizando PostgreSQL como sistema gestor. El diseño permite gestionar usuarios, productos, órdenes de compra y los detalles asociados a cada orden.

### **5.2 Entidades Principales del Sistema**

El sistema está compuesto por las siguientes tablas principales:

- users
- products
- orders
- order\_items
- purchase\_requests

### 5.3 Modelo Entidad–Relación (ERD)



### 5.4 Diccionario de Datos

- **users**

Almacena la información de los usuarios del sistema (compradores y vendedores).

- **PK:** id



<b>Campo</b>	<b>Tipo de dato</b>	<b>Descripción</b>
id	INTEGER (PK)	Identificador único del usuario
username	VARCHAR(50)	Nombre de usuario único en el sistema
password_hash	VARCHAR(255)	Contraseña cifrada mediante bcrypt
full_name	VARCHAR(120)	Nombre completo del usuario
phone	VARCHAR(20)	Número telefónico de contacto
bank_name	VARCHAR(80)	Nombre del banco del usuario
account_type	VARCHAR(30)	Tipo de cuenta bancaria
account_number	VARCHAR(30)	Número de cuenta bancaria
created_at	TIMESTAMP	Fecha y hora de creación del usuario

- **Relaciones:**

- users.id ← FK en products.seller\_id
- users.id ← FK en orders.buyer\_id
- users.id ← FK en purchase\_requests.buyer\_id
- users.id ← FK en purchase\_requests.seller\_id

- **products**

Almacenar los productos ofrecidos por los usuarios vendedores.

- **PK:** id
- **FK:** seller\_id → users.id

Campo	Tipo de dato	Descripción
id	INTEGER (PK)	Identificador único del producto
name	VARCHAR(80)	Nombre del producto
description	VARCHAR(255)	Descripción del producto
quantity	INTEGER	Cantidad disponible en stock
price	NUMERIC(10,2)	Precio unitario del producto
seller_id	INTEGER (FK)	Usuario que publica el producto
created_at	TIMESTAMP	Fecha y hora de creación del producto

### Relaciones

- products.seller\_id → users.id
- products.id → order\_items.product\_id
- products.id → purchase\_requests.product\_id

- **orders**

Registra las órdenes de compra realizadas por los usuarios.

- **PK:** id
- **FK:** buyer\_id → users.id

Campo	Tipo de dato	Descripción
id	INTEGER (PK)	Identificador único de la orden
buyer_id	INTEGER (FK)	Usuario comprador
total	NUMERIC(10,2)	Total de la compra
payment_method	VARCHAR(30)	Método de pago (didáctico)
status	VARCHAR(20)	Estado de la orden
contact_phone	VARCHAR(20)	Teléfono de contacto del comprador
contact_email	VARCHAR(120)	Correo electrónico del comprador
created_at	TIMESTAMP	Fecha y hora de creación de la orden

### Relaciones

- orders.buyer\_id → users.id
- orders.id → order\_items.order\_id

- **order\_items**

Tabla intermedia que relaciona órdenes con productos (relación muchos a muchos).

- **PK:** id
- **FK:**
  - order\_id → orders.id
  - product\_id → products.id

Campo	Tipo de dato	Descripción
id	INTEGER (PK)	Identificador único del detalle
order_id	INTEGER (FK)	Orden asociada
product_id	INTEGER (FK)	Producto comprado
quantity	INTEGER	Cantidad comprada
unit_price	NUMERIC(10,2)	Precio unitario al momento de la compra

## Relaciones

- order\_items.order\_id → orders.id
- order\_items.product\_id → products.id

- **purchase\_requests**

Almacena las solicitudes de compra que un comprador envía a un vendedor.

Funciona como un estado intermedio entre “quiero comprar” y “compra confirmada”.

**PK:** id

- **FK:**

- seller\_id → users.id
- buyer\_id → users.id
- product\_id → products.id

Campo	Tipo	Descripción
id	serial (PK)	Identificador único de la solicitud
product_id	integer (FK → products.id)	Producto que se desea comprar
seller_id	integer (FK → users.id)	Usuario vendedor (dueño del producto)
buyer_id	integer (FK → users.id)	Usuario comprador
quantity	integer	Cantidad solicitada
status	enum_purchase_requests_statuses	Estado de la solicitud
created_at	timestamp with time zone	Fecha y hora de creación

## Relaciones

- `order_items.order_id` → `orders.id`
- `order_items.product_id` → `products.id`
  
- Todas las tablas poseen **clave primaria**.
- Las relaciones se controlan mediante **claves foráneas**.
- Se evita duplicidad de datos mediante normalización.
- Las contraseñas se almacenan únicamente como **hash**.
- Se utilizan estados (status) para control del ciclo de vida de órdenes y solicitudes.

## 6. API REST – Endpoints del Sistema

### 6.1 Módulo de Autenticación (Auth)

**Ruta base:** `/api/auth`

Este módulo permite el registro y autenticación de usuarios, asegurando que las operaciones protegidas solo puedan ser ejecutadas por usuarios autenticados.

Método	Endpoint	Descripción	Doble autenticación
POST	<code>/api/auth/register</code>	Registra un nuevo usuario	No
POST	<code>/api/auth/login</code>	Autentica un usuario y retorna sesión/token JWT	No

## 6.2 Módulo de Productos

**Ruta base:** /api/products

Este módulo gestiona los productos ofrecidos por los usuarios vendedores.

Método	Endpoint	Descripción	Autenticación
GET	/api/products	Lista todos los productos disponibles	No
GET	/api/products/:id	Obtiene un producto por su ID	No
POST	/api/products	Crea un nuevo producto	Sí
PUT	/api/products/:id	Actualiza un producto existente	Sí
DELETE	/api/products/:id	Elimina un producto	Sí

## 6.3 Módulo de Solicitudes de Compra (Purchase Requests)

**Ruta base:** /api/Purchase Requests

Este módulo representa el núcleo del sistema de notificaciones.

Método	Endpoint	Descripción	Autenticación
POST	/api/purchase-requests	Crea una solicitud de compra (estado PENDING)	Sí
GET	/api/purchase-requests/seller/pending	Lista solicitudes pendientes del vendedor (notificaciones)	Sí
POST	/api/purchase-requests/:id/accept	Vendedor acepta solicitud (baja stock y crea orden)	Sí
POST	/api/purchase-requests/:id/reject	Vendedor rechaza solicitud	Sí
GET	/api/purchase-requests/:id/can-see-bank	Comprador consulta estado y datos del vendedor	Sí
Método	Endpoint	Descripción	Autenticación
POST	/api/orders/checkout	Crea una orden de compra a partir del carrito	Sí

<b>GET</b>	/api/orders/my	Lista las compras del usuario autenticado	Sí
------------	----------------	---	----

### 6.3 Módulo de Órdenes

**Ruta base:** /api/orders

Las órdenes ya no se crean directamente por el comprador, sino solo cuando el vendedor acepta una solicitud.

Método	Endpoint	Descripción	Autenticación
GET	/api/orders/my	Lista compras del usuario autenticado	Sí

### 6.4 Flujo de compra implementado:

**Comprador (Usuario B)** selecciona un producto.

Se envía solicitud a:

*POST /api/purchase-requests*

El backend crea un registro en:

- purchase\_requests con estado **PENDING**

**Vendedor (Usuario A):**

- Ve la solicitud mediante polling:
- GET /api/purchase-requests/seller/pending



El vendedor decide:

- **Aceptar**
  - Se reduce el stock del producto
  - Se crea un registro en orders
  - Se crean registros en order\_items
  - La solicitud pasa a **ACCEPTED**
- **Rechazar**
  - No se modifica stock
  - La solicitud pasa a **REJECTED**

**Estados de una solicitud (purchase\_requests.status)**

Estado	Significado
<b>PENDING</b>	Esperando decisión del vendedor
<b>ACCEPTED</b>	Aceptada, stock reducido y orden creada
<b>REJECTED</b>	Rechazada, no hay cambios de stock

## 6.5 Seguridad de los Endpoints

- Los endpoints sensibles utilizan un **middleware de autenticación**.
- El usuario autenticado se obtiene desde el token/sesión.
- Se validan parámetros y datos de entrada antes de procesar las solicitudes.

## 6.6 Códigos de Respuesta HTTP implementados

Código	Significado
200	Operación exitosa
201	Recurso creado correctamente
400	Error de validación
401	No autorizado
404	Recurso no encontrado
500	Error interno del servidor

## 7. Seguridad, Auditoría y Backups

### 7.1 Análisis de riesgos

Se identifican las siguientes vulnerabilidades comunes:

1. **Accesos no autorizados** debido a una mala gestión de usuarios y contraseñas.
2. **Falta de auditoría**, lo que dificulta detectar accesos indebidos o modificaciones no autorizadas.
3. **Pérdida de información** por ausencia de políticas de respaldo.
4. **Ataques de inyección SQL**, si no se controla el acceso desde la aplicación.

La superficie de ataque principal se encuentra en la conexión entre la aplicación web y la base de datos, por lo que se aplican controles de acceso estrictos y auditoría a nivel del DBMS.

## 7.2 Implementación de RBAC (Control de Acceso Basado en Roles)

Se implementó un modelo de **Control de Acceso Basado en Roles (RBAC)** siguiendo el principio de menor privilegio.

### Roles definidos:

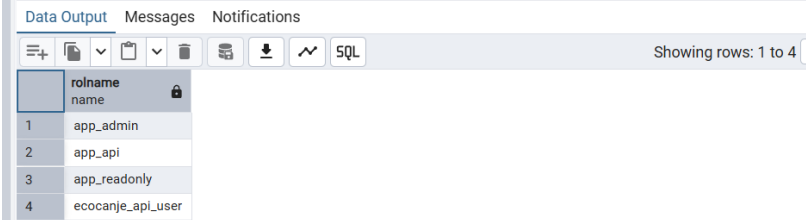
- **app\_admin**: rol administrativo para mantenimiento.
- **app\_api**: rol técnico utilizado por el backend de la aplicación.
- **app\_readonly**: rol de solo lectura para consultas o reportes.

### Usuario técnico:

- **ecocanje\_api\_user**, utilizado exclusivamente por el backend para acceder a la base de datos.

### Roles creados en pgadmin

```
153 -- Roles creados
154 SELECT rolname
155 FROM pg_roles
156 WHERE rolname IN ('app_admin','app_api','app_readonly','ecocanje_api_user')
157 ORDER BY rolname;
```



	rolname	name
1	app_admin	
2	app_api	
3	app_readonly	
4	ecocanje_api_user	

Cada rol cuenta únicamente con los permisos necesarios para cumplir su función, reduciendo el impacto ante un posible compromiso de credenciales.

<i>Rol Aplicación</i>	<b>Rol BD</b>	<b>Permisos</b>
<i>Backend API</i>	app_api	SELECT, INSERT, UPDATE, DELETE
<i>Reportes</i>	app_readonly	SELECT
<i>Administración</i>	app_admin	Gestión controlada

### Permisos asignados a los diferentes roles

```

164 SELECT grantee, privilege_type
165 FROM information_schema.role_table_grants
166 WHERE grantee = 'app_api';
167
168
169

```

Data Output		
	grantee name	privilege_type character varying
1	app_api	INSERT
2	app_api	SELECT
3	app_api	UPDATE
4	app_api	DELETE

## 7.3 Configuración de cifrado

En PostgreSQL, el cifrado se maneja en dos niveles:

- **Cifrado en tránsito:** mediante SSL/TLS, garantizando que la comunicación entre la aplicación y la base de datos no pueda ser interceptada.
- **Cifrado en reposo:** mediante cifrado a nivel de disco y el uso de la extensión **pgcrypto** para operaciones criptográficas.

```
168 SELECT extname FROM pg_extension;
169
170
```

Data Output

Messages

Notifications

≡+

📄

▼

📋

▼

🗑️

🗄️

⬇️

📈

SQL

	extname name 🔒
1	plpgsql
2	pgcrypto

Para el proyecto Ecocanje, se habilitó la extensión pgcrypto, permitiendo el uso de funciones de hashing y cifrado cuando sea necesario para proteger información sensible.

```
🔥 docker-compose.yml M ✕ ⚙️ pg_hba.conf U ✕ ⚙️ postgresql.conf U
postgres-conf > ⚙️ pg_hba.conf
1 # TYPE DATABASE USER ADDRESS METHOD
2
3 # Local (dentro del contenedor)
4 local all all scram-sha-256
5
6 # Permitir conexiones desde la red Docker
7 host all all 0.0.0.0/0 scram-sha-256
8 |
```

Se configuró pg\_hba.conf para aplicar autenticación SCRAM-SHA-256, evitando métodos inseguros como trust o md5, reduciendo el riesgo de accesos no autorizados.

```
postgres.conf U X
postgres-conf > postgresql.conf
1  # =====
2  # Seguridad base (PostgreSQL 15)
3  # =====
4
5  # Escucha en todas las interfaces dentro de Docker
6  listen_addresses = '*'
7  port = 5432
8
9  # Autenticación más segura
10 password_encryption = scram-sha-256
11
12 # Logging (auditoría + trazabilidad)
13 logging_collector = on
14 log_destination = 'stderr'
15 log_directory = 'log'
16 log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
17 log_line_prefix = '%m [%p] %u@%d %h '
18 log_connections = on
19 log_disconnections = on
20 log_statement = 'none'
21 log_min_duration_statement = 500
22
23 # =====
24 # pgAudit
25 # =====
26 shared_preload_libraries = 'pgaudit'
27
28 # Ajustes pgAudit recomendados para auditoría de cambios
29 pgaudit.log = 'write, ddl, role'
30 pgaudit.log_catalog = off
31 pgaudit.log_relation = on
32 pgaudit.log_parameter = on
33 pgaudit.log_statement_once = on
34
```

- Se registran conexiones
- Se registran desconexiones
- Se auditan queries lentas (>500 ms)

## pgAudit (Auditoría avanzada)

Esto habilita auditoría de:

- INSERT / UPDATE / DELETE
- CREATE / ALTER / DROP
- Cambios de roles
- Acceso a tablas

La gestión de claves se propone mediante variables de entorno y buenas prácticas de seguridad, evitando su exposición en el código fuente.

## CONFIGURACIÓN DE AUDITORÍA Y MONITOREO

Se implementó auditoría mediante:

- **Logs nativos de PostgreSQL**, que registran conexiones, desconexiones y consultas.
- **Extensión pgAudit**, configurada para auditar operaciones de escritura y cambios estructurales.

Estos mecanismos permiten detectar accesos indebidos, modificaciones no autorizadas y analizar el comportamiento de los usuarios del sistema.

El monitoreo se realiza a través de logs del contenedor Docker y herramientas administrativas como pgAdmin.

Podemos verificar las auditorias con el comando:

### docker logs express\_003\_postgres

```
2026-02-05 19:54:50.783 UTC [73] user=postgres db=ecocanje_db app=pgAdmin 4 - DB:ecocanje_db client=172.20.0.1 LOG: duration: 6.701 ms statement: /*pgAdash*/
SELECT 'session_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT count(*) FROM pg_catalog.pg_stat_activity WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Total",
  (SELECT count(*) FROM pg_catalog.pg_stat_activity WHERE state = 'active' AND datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Active",
  (SELECT count(*) FROM pg_catalog.pg_stat_activity WHERE state = 'idle' AND datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Idle"
) t
UNION ALL
SELECT 'tps_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(xact_commit) + sum(xact_rollback) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Transactions",
  (SELECT sum(xact_commit) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Commits",
  (SELECT sum(xact_rollback) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Rollbacks"
) t
UNION ALL
SELECT 'ti_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(tup_inserted) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Inserts",
  (SELECT sum(tup_updated) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Updates",
  (SELECT sum(tup_deleted) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Deletes"
) t
UNION ALL
SELECT 'to_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(tup_fetched) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Fetched",
  (SELECT sum(tup_returned) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Returned"
) t
UNION ALL
SELECT 'bio_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(blks_read) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Reads",
  (SELECT sum(blks_hit) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Hits"
) t
```

## **Estrategia de backups y recuperación**

Para el proyecto ECOCANJE se implementó una política de respaldos automáticos utilizando la herramienta **pg\_dump**, ejecutada mediante un servicio **cron** en un contenedor Docker independiente.

**Frecuencia:** cada 1 hora

**Retención:** 7 días

Para el proyecto ECOCANJE se definieron los siguientes objetivos de recuperación:

**RPO:** 1 hora

**RTO :** 30 minutos

### **RPO (Recovery Point Objective):**

El valor de RPO se estableció en una hora debido a que la aplicación ECOCANJE es una plataforma web académica que no maneja transacciones financieras en tiempo real.

Las operaciones principales del sistema (registro de usuarios, publicación de productos y solicitudes de compra) toleran una pérdida limitada de información reciente sin comprometer la integridad general del sistema.

### **RTO (Recovery Time Objective):**

El RTO se fijó en 30 minutos considerando que los respaldos se realizan mediante archivos SQL generados con **pg\_dump**, los cuales permiten una restauración rápida de la base de datos utilizando el comando **psql**



Esta combinación de RPO y RTO equilibra adecuadamente la disponibilidad del sistema, el costo operativo y el nivel de riesgo aceptable para el contexto académico del proyecto ECOCANJE.

**Backup.sh file:** aquí vamos a realizar el backup automático

```

$ backup.sh
6 # =====
7
8 # ☒ Defaults
9 PGHOST="${PGHOST:-postgres}"
10 PGPORT="${PGPORT:-5432}"
11 PGDATABASE="${PGDATABASE:-ecocanje_db}"
12 PGUSER="${PGUSER:-postgres}"
13 PGPASSWORD="${PGPASSWORD:-postgres}"
14
15 export PGPASSWORD
16
17 DATE="$(date +"%Y%m%d_%H%M%S")"
18 BACKUP_DIR="/backups"
19 mkdir -p "$BACKUP_DIR"
20
21 echo "[$(date)] Iniciando backup de ${PGDATABASE} en ${PGHOST}:${PGPORT} con usuario ${PGUSER}"
22
23 pg_dump \
24 -F p \
25 -h "${PGHOST}" \
26 -p "${PGPORT}" \
27 -U "${PGUSER}" \
28 "${PGDATABASE}" > "${BACKUP_DIR}/${PGDATABASE}_${DATE}.sql"
29
30 echo "[$(date)] Backup exitoso: ${PGDATABASE}_${DATE}.sql"
31
32 # Retención: borrar .sql mayores a 7 días
33 find "${BACKUP_DIR}" -type f -name "*.sql" -mtime +7 -delete || true
34
```

**crontab:** seteado para que se realice el backup cada hora

```

$ crontab.txt
1 0 * * * * /bin/bash /backup.sh >> /backups/cron.log 2>&1
2
3
4
```

**Cron.log:** aquí vamos a ver todos los Backups realizados por nuestra base de datos. (al comienzo lo hicimos cada minuto para pruebas)

```

cron.log U X  docker-compose.yml M
backups > cron.log
19 Thu Feb 5 04:43:41 PM UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_164341.sql
20 Thu Feb 5 16:44:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_164401.sql
21 Thu Feb 5 16:45:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_164501.sql
22 Thu Feb 5 16:46:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_164601.sql
23 Thu Feb 5 16:47:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_164701.sql
24 Thu Feb 5 16:48:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_164801.sql
25 Thu Feb 5 16:49:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_164901.sql
26 Thu Feb 5 16:50:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_165001.sql
27 Thu Feb 5 16:51:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_165101.sql
28 Thu Feb 5 16:52:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_165201.sql
29 Thu Feb 5 16:53:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_165301.sql
30 Thu Feb 5 16:54:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_165401.sql
31 Thu Feb 5 16:55:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_165501.sql
32 Thu Feb 5 17:00:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_170001.sql
33 Thu Feb 5 18:00:01 UTC 2026 ✓ Backup OK: /backups/ecocanje_db_20260205_180001.sql
34 |

```

**Docker -compose.yml:** importante agregar un nuevo volumen para realizar los Backups en ese servidor de pruebas de respaldos.

```

cron.log U $ backup.sh U  docker-compose.yml M X
docker-compose.yml
1 services:
2   postgres:
3     image: postgres:15
4     container_name: express_003_postgres
5     ports:
6       - "5435:5432"
7     environment:
8       POSTGRES_USER: postgres
9       POSTGRES_PASSWORD: postgres
10      POSTGRES_DB: ecocanje_db
11    volumes:
12      - ./backups:/backups
13    command: >
14      postgres
15      -c log_destination=stderr
16      -c logging_collector=off
17      -c log_connections=on
18      -c log_disconnections=on
19      -c log_duration=on
20      -c log_min_duration_statement=0
21      -c log_line_prefix='%m [%p] user=%u db=%d app=%a client=%h '
22
23  You, 18 minutes ago • Uncommitted changes
24
25  backup-cron:
26    build:
27      context: .
28      dockerfile: Dockerfile.cron
29    container_name: ecocanje_backup_cron
30    depends_on:
31      - postgres
32    environment:
33      PGHOST: postgres
34      PGPORT: "5432"
35      PGDATABASE: ecocanje_db
36      PGUSER: ecocanje_api_user
37      PGPASSWORD: "123456"
38    volumes:
39      - ./backups:/backups

```

ecocanje\_backup\_cron: container donde vamos a realizar los Backups automáticos.

Containers

Give feedback

Container CPU usage

0.35% / 800% (8 CPUs available)

Container memory usage

88.41MB / 7.43GB

Show charts

Search

Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memo	Actions
<input type="checkbox"/>	<a href="#">examen_final_pareja_cristian</a>	-	-	-	0%	0B / 0B		<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">proyecto_final_ecocanje_test</a>	-	-	-	0%	0B / 0B		<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">downloads</a>	-	-	-	0%	0B / 0B		<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">proyecto_final_ecocanje</a>	-	-	-	0.01%	88.41MB / 15.22Gi		<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">express_003_postgres</a>	1bfbc1d071	<a href="#">postgres:11</a>	<a href="#">5435:5432</a>	0.01%	80.23MB / 7.61Gi		<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>
<input type="checkbox"/>	<a href="#">ecocanje_backup_cron</a>	199703016f88	<a href="#">proyecto-fi</a>		0%	8.18MB / 7.61GB		<a href="#">▶</a> <a href="#">⋮</a> <a href="#">🗑</a>

## 8. Pruebas

### 8.1 POSTMAN

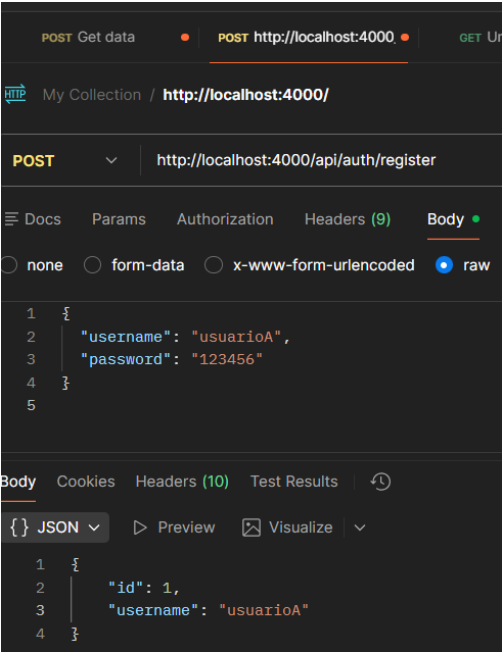
#### Levantamos Docker

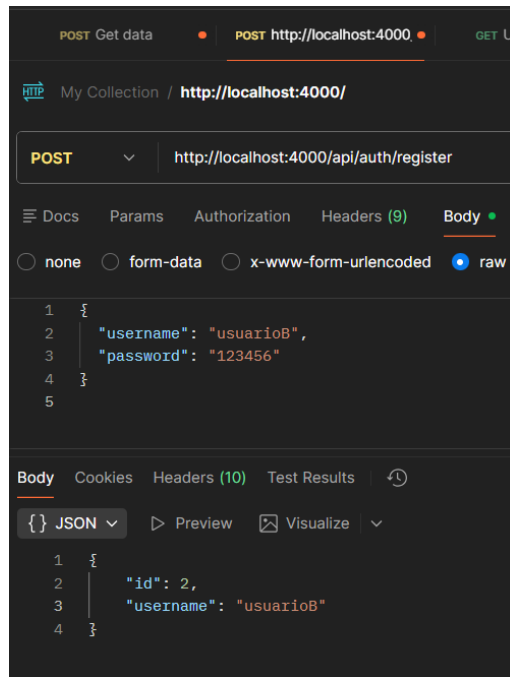
```
>> C:\Users\crist\OneDrive\Escritorio\PUCE\DesarrolloWeb\Proyecto_final_ECOCANJE>
PS C:\Users\crist\OneDrive\Escritorio\PUCE\DesarrolloWeb\Proyecto_final_ECOCANJE> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                    NAMES
2da24d912d9f   proyecto_final_ecocanje-backup-cron "docker-entrypoint.s..." 11 minutes ago Up 11 minutes  5432/tcp              ecocanje_backup_cron
17db753dd7e1   postgres:15                          "docker-entrypoint.s..." 11 minutes ago Up 11 minutes  0.0.0.0:5435->5432/tcp, [::]:5435->5432/tcp express_003_postgres
PS C:\Users\crist\OneDrive\Escritorio\PUCE\DesarrolloWeb\Proyecto_final_ECOCANJE>
```

#### CRUD (Postman)

Registrar usuarioA y usuarioB

POST <http://localhost:4000/api/auth/register>





**pgadmin**

Query

Query History

1

2

3

SELECT id, username, created\_at

FROM users

ORDER BY id DESC;

Data Output

Messages

Notifications

≡

📄

▼

📋

▼




🗑️

🗄️

⬇️

📈

SQL

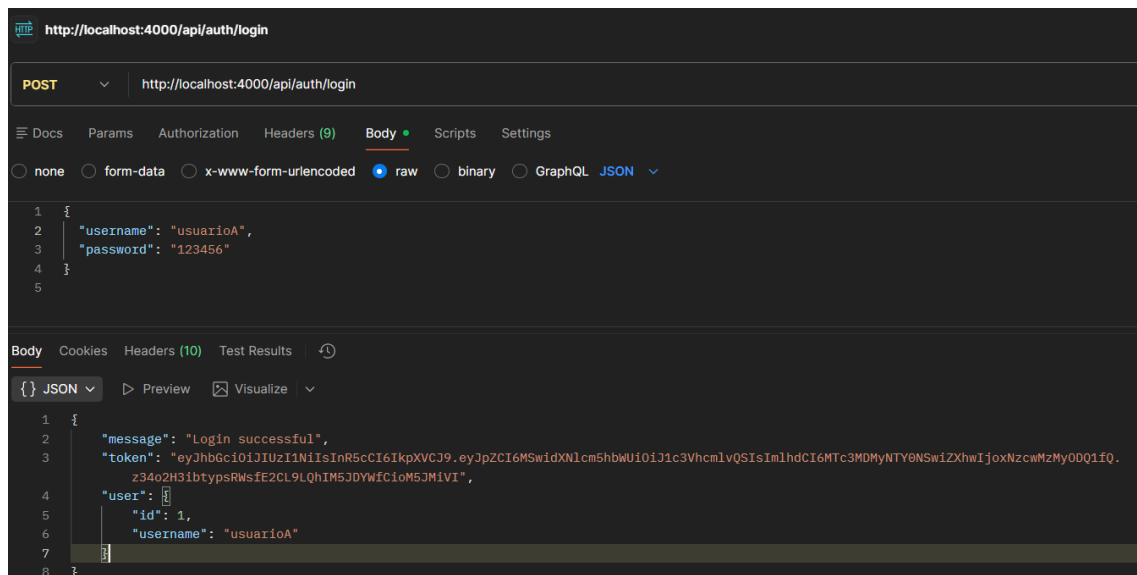
	id [PK] integer 	username character varying (50) 	created_at timestamp with time zone 
1	2	usuarioB	2026-02-05 21:02:50.456+00
2	1	usuarioA	2026-02-05 21:01:59.741+00

## Login usuarioA y usuarioB (guardar los tokens)

POST <http://localhost:4000/api/auth/login>

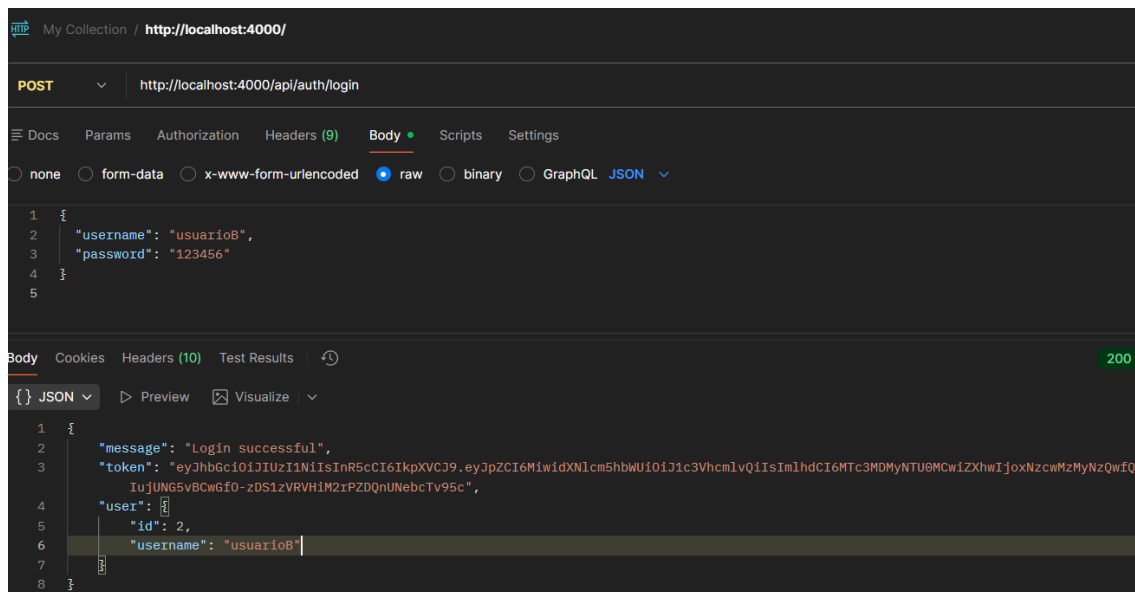
tokenA:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJ1c3VhcmlvQSI6ImVhdCI6MTc3MDMyNTY0NSwiZXhwIjoxNzcwMzM5ODQ1fQ.z34o2H3ibtypsRWsfE2CL9LQhIM5JDYWfCioM5JMiVI



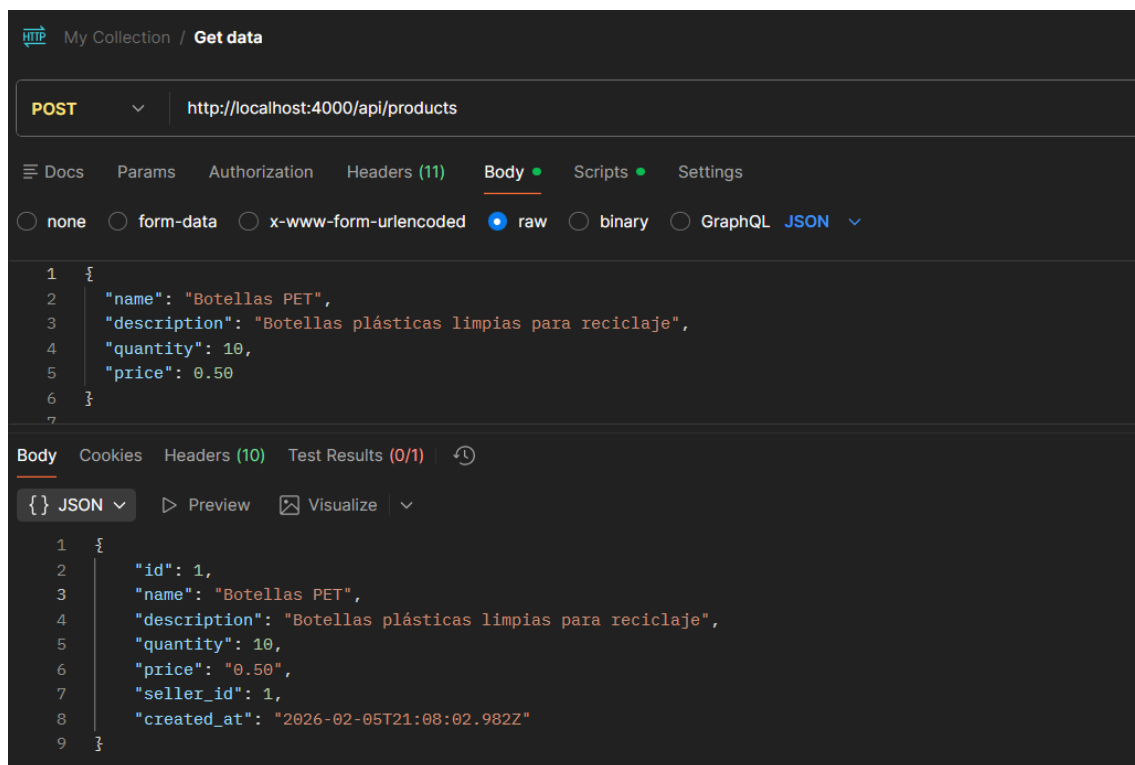
tokenB:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiOiJ1c3VhcmlvQSI6ImVhdCI6MTc3MDMyNTU0MCwiZXhwIjoxNzcwMzM5ODQ1fQ.IujUNG5vBCwGfO-zDS1zVRVHiM2rPZDQnUNebcTv95c



### Crear producto (A)

## POST http://localhost:4000/api/products



pgadmin

```
5 SELECT id, name, quantity, price, seller_id, created_at
6 FROM products
7 ORDER BY id DESC;
8
```

Data Output Messages Notifications

SQL

	id [PK] integer	name character varying (80)	quantity integer	price numeric (10,2)	seller_id integer	created_at timestamp with time zone
1	1	Botellas PET	10	0.50	1	2026-02-05 21:08:02.982+00

### Editorial product (A)

**PUT** http://localhost:4000/api/products/1

The screenshot shows the REST Client interface with a PUT request to `http://localhost:4000/api/products/1`. The request body is a JSON object:

```
{
  "name": "Botellas PET",
  "description": "Botellas plásticas limpias para reciclaje",
  "quantity": 12,
  "price": 0.60
}
```

The interface includes tabs for Docs, Params, Authorization, Headers (11), Body, Scripts, and Settings. The Body tab is selected, and the format is set to raw. The response section shows a 200 OK status with a JSON body:

```
{
  "id": 1,
  "name": "Botellas PET",
  "description": "Botellas plásticas limpias para reciclaje",
  "quantity": 12,
  "price": 0.6,
  "seller_id": 1,
  "created_at": "2026-02-05T21:08:02.982Z"
}
```

pgadmin

9  
10  
11  
12

SELECT id, name, quantity, price, seller\_id

FROM products

WHERE id = 1;

Data Output

Messages

Notifications

SQL

	id [PK] integer	name character varying (80)	quantity integer	price numeric (10,2)	seller_id integer
1	1	Botellas PET	12	0.60	1

Borrar producto (A)

DELETE http://localhost:4000/api/products/1

DELETE

http://localhost:4000/api/products/1

Docs

Params

Authorization

Headers (11)

Body

Scripts

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1 {

2   "name": "Botellas PET",

3   "description": "Botellas plásticas limpias para reciclaje",

4   "quantity": 12,

5   "price": 0.60

6 }

7

Body

Cookies

Headers (10)

Test Results (1/1)

{}

JSON

Preview

Visualize

1 {

2   "message": "Product deleted successfully",

3   "product": {

4     "id": 1,

5     "name": "Botellas PET",

6     "description": "Botellas plásticas limpias para reciclaje",

7     "quantity": 12,

8     "price": "0.60",

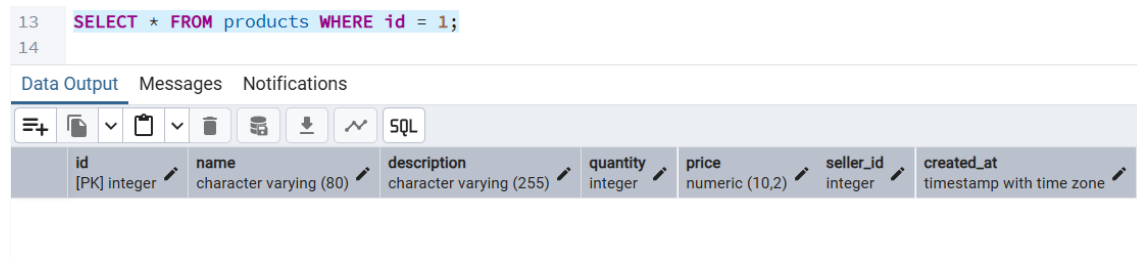
9     "seller\_id": 1,

10    "created\_at": "2026-02-05T21:08:02.982Z"

11 }



pgadmin



## Pruebas de Auditoria

leyendo tablas del sistema y haciendo COPY public.users/products/... para generar el archivo .sql

## Ejecutamos

docker stop ecocanje\_backup\_cron

docker logs -f express\_003\_postgres

## realizamos en postman

register/login usuarioA

create product

update product

delete product

login usuarioB

docker start ecocanje\_backup\_cron

```
2026-02-05 21:14:50.710 UTC [67] user=postgres db=ecocanje_db app=pgAdmin 4 - DB:ecocanje_db client=172.20.0.1 LOG: duration: 5.657 ms statement: /*pgadash*/
SELECT 'session_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT count(*) FROM pg_catalog.pg_stat_activity WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Total",
  (SELECT count(*) FROM pg_catalog.pg_stat_activity WHERE state = 'active' AND datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Active",
  (SELECT count(*) FROM pg_catalog.pg_stat_activity WHERE state = 'idle' AND datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Idle"
) t
UNION ALL
SELECT 'tps_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(xact_commit) + sum(xact_rollback) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Transaction
s",
  (SELECT sum(xact_commit) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Commits",
  (SELECT sum(xact_rollback) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Rollbacks"
) t
UNION ALL
SELECT 'ti_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(tup_inserted) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Inserts",
  (SELECT sum(tup_updated) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Updates",
  (SELECT sum(tup_deleted) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Deletes"
) t
UNION ALL
SELECT 'to_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(tup_fetched) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Fetched",
  (SELECT sum(tup_returned) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Returned"
) t
UNION ALL
SELECT 'bio_stats' AS chart_name, pg_catalog.row_to_json(t) AS chart_data
FROM (SELECT
  (SELECT sum(bio_read) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Reads",
  (SELECT sum(bio_write) FROM pg_catalog.pg_stat_database WHERE datname = (SELECT datname FROM pg_catalog.pg_database WHERE oid = 16384)) AS "Writes"
) t
```

Creación de B

22 ✓  
23  
24  
25

```
SELECT id, username, created_at
FROM users
WHERE username='usuarioB';
```

Data OutputMessagesNotifications

≡+

▼

▼

SQL

	id [PK] integer	username character varying (50)	created_at timestamp with time zone
1	2	usuarioB	2026-02-05 21:02:50.456+00

Creación de producto por parte de A

16  
17 ✓  
18  
19  
20  
21

```
--Auditoria
SELECT p.id, p.name, u.username AS seller, p.created_at
FROM products p
JOIN users u ON u.id = p.seller_id
ORDER BY p.id DESC;
```

Data OutputMessagesNotifications

≡+

▼

▼

SQL

	id integer	name character varying (80)	seller character varying (50)	created_at timestamp with time zone
1	2	Botellas PET	usuarioA	2026-02-05 21:19:38.794+00

### Actualización del producto por parte de A

```
26 SELECT p.id, p.name, p.quantity, p.price, u.username AS seller, p.created_at
27 FROM products p
28 JOIN users u ON u.id = p.seller_id
29 WHERE p.id = 2;
30
```

Data Output Messages Notifications

SQL Showin

	id integer	name character varying (80)	quantity integer	price numeric (10,2)	seller character varying (50)	created_at timestamp with time zone
1	2	Botellas PET	120	0.80	usuarioA	2026-02-05 21:19:38.794+00

### Eliminación de producto por parte de A

```
31 SELECT * FROM products WHERE id = 2;  
32
```

Data Output Messages Notifications

SQL

	id [PK] integer	name character varying (80)	description character varying (255)	quantity integer	price numeric (10,2)	seller_id integer	created_at timestamp with time zone
--	--------------------	--------------------------------	--	---------------------	-------------------------	----------------------	--

## Pruebas de Backup

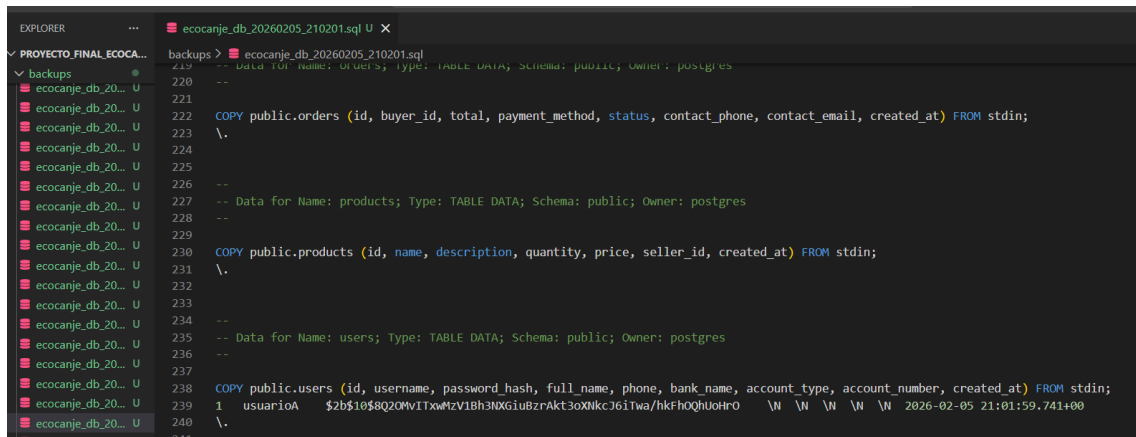
**Antes:**

ecocanje\_db\_20260205\_204757.sql se realizó antes de que se creen los usuarios A y B por eso está vacío

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'EXPLORER' pane shows a tree view with a folder named 'backups'. Below it, a list of backup files is shown, including 'ecocanje\_db\_20260205\_204757.sql'. The right pane shows the content of the selected file, which is a list of backup files. The bottom status bar indicates the current file is 'ecocanje\_db\_20260205\_204757.sql'.

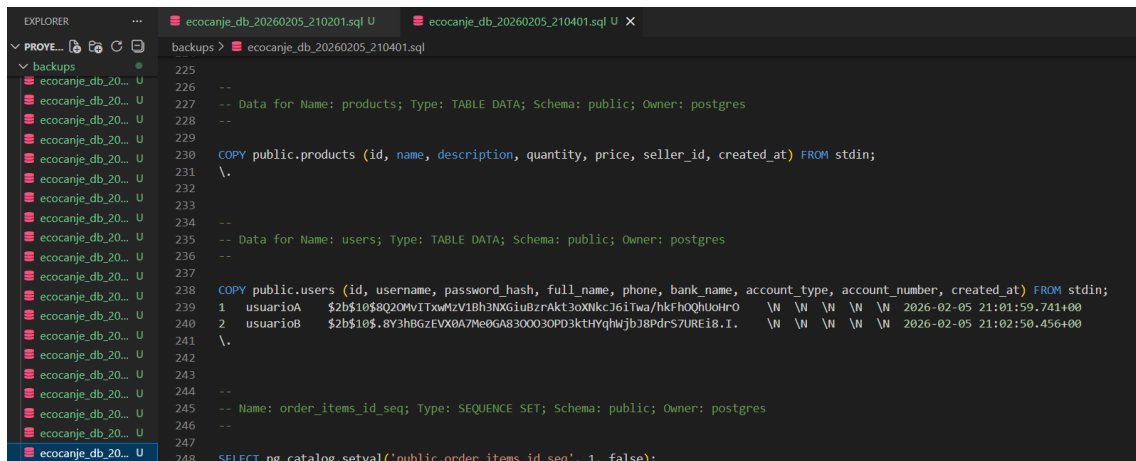
Después:

ecocanje\_db\_20260205\_210201.sql se realizó después de que se creen los usuarios A por lo que ya contiene información



```
EXPLORER    ...    ecocanje_db_20260205_210201.sql U X
backups > ecocanje_db_20260205_210201.sql
220 --
221 -- Data for Name: orders; Type: TABLE DATA; Schema: public; Owner: postgres
222 COPY public.orders (id, buyer_id, total, payment_method, status, contact_phone, contact_email, created_at) FROM stdin;
223 \.
224
225 --
226 -- Data for Name: products; Type: TABLE DATA; Schema: public; Owner: postgres
227 --
228 --
229 COPY public.products (id, name, description, quantity, price, seller_id, created_at) FROM stdin;
230 \.
231
232 --
233 -- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: postgres
234 --
235 --
236 COPY public.users (id, username, password_hash, full_name, phone, bank_name, account_type, account_number, created_at) FROM stdin;
237 1 usuarioA $2b$10$8Q20MvITxw4tZV1Bh3IXGiuBzrAkt3oXNkcJ6iIwa/hkFhoQhUoHrO \N \N \N \N 2026-02-05 21:01:59.741+00
238 \.
239
240 --
241
```

ecocanje\_db\_20260205\_210401.sql se realizó después de que se creen los usuarios B por lo que ya contiene información



```
EXPLORER    ...    ecocanje_db_20260205_210201.sql U    ecocanje_db_20260205_210401.sql U X
backups > ecocanje_db_20260205_210401.sql
225 --
226 -- Data for Name: products; Type: TABLE DATA; Schema: public; Owner: postgres
227 --
228 --
229 COPY public.products (id, name, description, quantity, price, seller_id, created_at) FROM stdin;
230 \.
231
232 --
233 -- Data for Name: users; Type: TABLE DATA; Schema: public; Owner: postgres
234 --
235 --
236 COPY public.users (id, username, password_hash, full_name, phone, bank_name, account_type, account_number, created_at) FROM stdin;
237 1 usuarioA $2b$10$8Q20MvITxw4tZV1Bh3IXGiuBzrAkt3oXNkcJ6iIwa/hkFhoQhUoHrO \N \N \N \N 2026-02-05 21:01:59.741+00
238 2 usuarioB $2b$10$.8Y3hBgzeVX0A7Me0GA8300030PD3kthYqhwjB38PdrS7UREi8.I. \N \N \N \N 2026-02-05 21:02:50.456+00
239 \.
240
241 --
242 -- Name: order_items_id_seq; Type: SEQUENCE SET; Schema: public; Owner: postgres
243 --
244 SELECT pg_catalog.setval('public.order_items_id_seq', 1, false);
245
```

## 8.2 API (FRONTEND)

Registro nuevo usuario frank07

 **EcoCanje**  
Ingresa para comprar o vender

Login

Registro

Usuario

frank07

Contraseña

.....

Nombre completo

Franklin Toro

Teléfono

0992695258

Banco

Pichincha

Tipo de cuenta


Ahorros

Número de cuenta

2345637383

Crear cuenta

Login usuario frank07

 **EcoCanje**  
Ingresa para comprar o vender

Login

Registro

Usuario

frank07

Contraseña

.....

Invalid credentials

Ingresar

Crear nuevo producto frank07 (seller)

Publicar nuevo producto

Cerrar

Completa los datos para vender.

Nombre

Pilas

Descripción

pilas 3A usadas

Cantidad

15

Precio

0,05

Publicar

EcoCanje

Marketplace reciclaje

frank07

Sesión activa

Solicitudes (0)

Vender / Publicar

Salir

Productos disponibles

Refrescar

Pilas

pilas 3A usadas

Vendedor: Franklin Toro

Stock: 15

1

Añadir

\$0.05

plastico

botesllas

Vendedor: Carlos Perez

Stock: 200

1

Añadir


\$0.40

Detalle

Puedes solicitar compra de 1 producto a la vez.

No has añadido ningún producto todavía.

Registro nuevo usuario cris08



EcoCanje

Ingresar para comprar o vender

Login

Registro

Usuario

cris08

Contraseña

.....

Nombre completo

Cristian Pareja

Teléfono

0995163245

Banco

Produbanco

Tipo de cuenta


Corriente

Número de cuenta

39484773773

Crear cuenta

Login usuario cris08



EcoCanje

Ingresar para comprar o vender

Login

Registro

Usuario


cris08

Contraseña

.....


Ingresar

Cris08 compra 5 unidades del producto publicado por frank07 (buyer)



**EcoCanje**  
Marketplace reciclaje

cris08  
Sesión activa

 Solicitudes (0)

Vender / Publicar

Salir

Productos disponibles

Refrescar

**Pilas**  
pilas 3A usadas  
Vendedor: Franklin Toro  
Stock: 15

5

Añadir

**plastico**  
botessllas  
Vendedor: Carlos Perez  
Stock: 200

1

Añadir

Detalle

Quitar

**Pilas**  
pilas 3A usadas  
Cantidad: 5

Total  
**\$0.25**

Solicitar compra

Información del vendedor **PENDIENTE**

Cerrar

Primero el vendedor debe aceptar. La transferencia se habilita solo si acepta.

Usuario	frank07
Nombre	Franklin Toro
Teléfono	0992695258

**Transferencia**

Datos bancarios bloqueados: pendiente de aprobación del vendedor.

\* App académica. El pago/entrega se coordinan fuera de la plataforma.



frank07 (buyer) deberá aceptar o rechazar la venta del producto

Productos disponibles

Refrescar

**Pilas**

pilas 3A usadas

Vendedor: Franklin Toro

Stock: 15

Añadir

\$0.05

**plastico**

botessillas

Vendedor: Carlos Prerez

Stock: 200

Añadir

\$0.40

Solicitudes de compra

Cerrar

Acepta o rechaza solicitudes. El stock solo baja si aceptas.

**Producto: Pilas**

Cantidad solicitada: 5

Comprador: cris08 · Tel: 0995163245

PENDING

Rechazar

Aceptar

\* Al aceptar, el stock se reduce en el backend. Al rechazar, no cambia el stock.

Productos disponibles

Refrescar

**Pilas**

pilas 3A usadas

Vendedor: Franklin Toro

Stock: 10

Añadir

\$0.05

**plastico**

botessillas

Vendedor: Carlos Prerez

Stock: 200

Añadir


\$0.40

Detalle

Puedes solicitar compra de 1 producto a la vez.


No has añadido ningún producto todavía.

Usuario cris08 va a realizar un update y un delete en producto prueba



EcoCanje  
Marketplace reciclaje

cris08  
Sesión activa

 Solicitudes (0)

Vender / Publicar

Salir

Productos disponibles

Refrescar

prueba

prueab CRUD

Vendedor: Cristian Pareja Tu producto

Stock: 201

1

Añadir

Editar

Eliminar

\$90.00

Pilas

pilas 3A usadas

Vendedor: Franklin Toro

Stock: 10

1

Añadir

\$0.05

plastico

boteslillas

Vendedor: Carlos Perez

Stock: 200

1

Añadir

\$0.40

Productos disponibles

Refrescar

prueba

prueba CRUD

Vendedor: Cristian Pareja Tu producto

Stock: 20

1

Añadir

Editar

Eliminar

\$0.90

Pilas

pilas 3A usadas

Vendedor: Franklin Toro

Stock: 10

1

Añadir

\$0.05

plastico

boteslillas

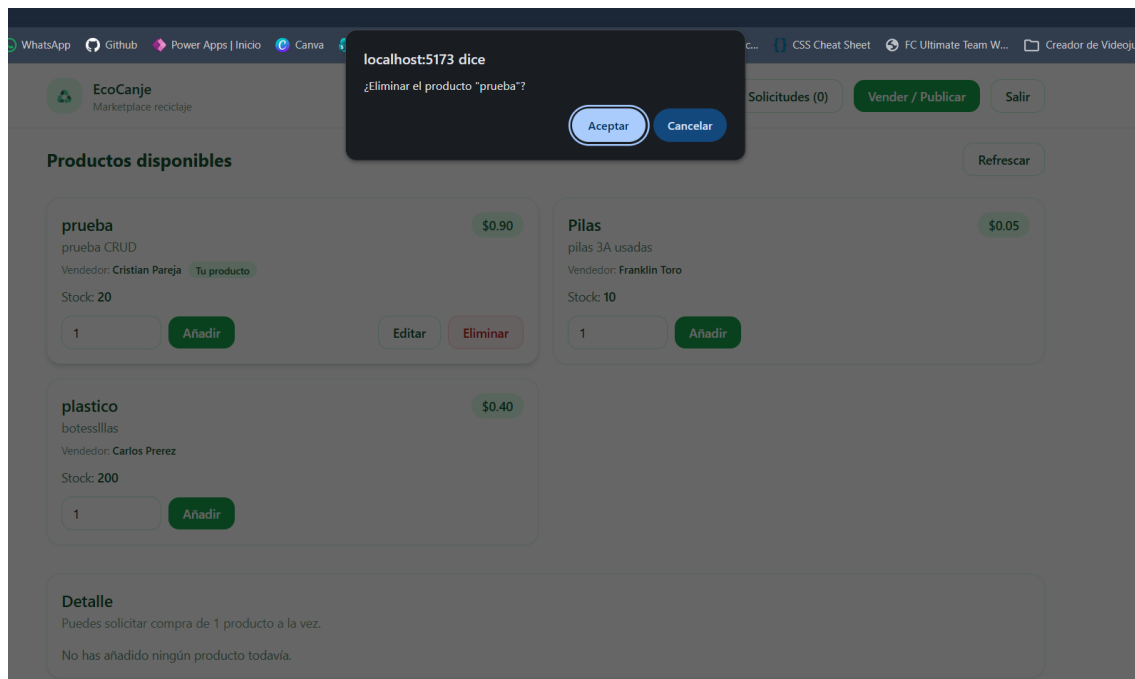
Vendedor: Carlos Perez

Stock: 200

1

Añadir

\$0.40



## 9. Conclusiones

- **La arquitectura por capas (routes, services, repositories)** permitió aislar errores sin afectar la base de datos ni la lógica de negocio.
- **El sistema de notificaciones basado en polling** es simple, efectivo y adecuado para un proyecto académico.
- **La auditoría nativa de PostgreSQL** permitió rastrear todas las operaciones críticas sin agregar complejidad al código.
- **La política de backups automatizados con cron** garantiza recuperación ante fallos y respalda los objetivos RPO y RTO definidos.
- **El control de flujo de compra (PENDING, ACCEPTED or REJECTED)** asegura integridad de stock y coherencia transaccional.