Prueba Técnica QA
Sistema de Gestión

Funcionalidades

API de Usuarios y Transacciones

Pedro Laprea

1. Descripción general del Sistema

- **Tipo de Aplicación:** Sistema de Gestión de usuarios y transacciones (Task Manager)
- **Arquitectura:** API REST pura (Solo Backend)
- **Framework:** Laravel 10.x
- **Base de Datos:** Soporta múltiples (SQLite por defecto en desarrollo)
- **Autenticación:** Token-based (Laravel Sanctum)

2. Funcionalidades del sistema

Autenticación

- **Registro**: Crear nueva cuenta de usuario con nombre, email y contraseña
- **Login**: Iniciar sesión para obtener token de autenticación
- **Logout**: Cerrar sesión
- **Usuario autenticado:** Obtener información del usuario actual

Gestión de Usuarios

- **Listar usuarios:** Ver todos los usuarios registrados con su saldo
- **Crear usuario:** Registrar nuevo usuario con nombre, email y saldo inicial
- **Mostrar usuario:** Ver detalles de un usuario específico por ID
- **Actualizar usuario:** Modificar datos de usuario existente
- **Eliminar usuario:** Remover usuario del sistema

Gestión de Transacciones

- **Listar transacciones:** Ver todas las transferencias registradas
- **Crear transacción:** Realizar transferencia entre usuarios (verifica saldo)
- **Mostrar transacción:** Ver detalles de transacción específica
- **Actualizar transacción:** Modificar monto de transacción existente
- **Eliminar transacción:** Eliminar registro de transacción

Reportes y Exportación

- **Exportar CSV:** Generar archivo CSV con todas las transacciones
- **Resumen estadístico:** Obtener total transferido y promedio por usuario emisor

3. Casos de Prueba

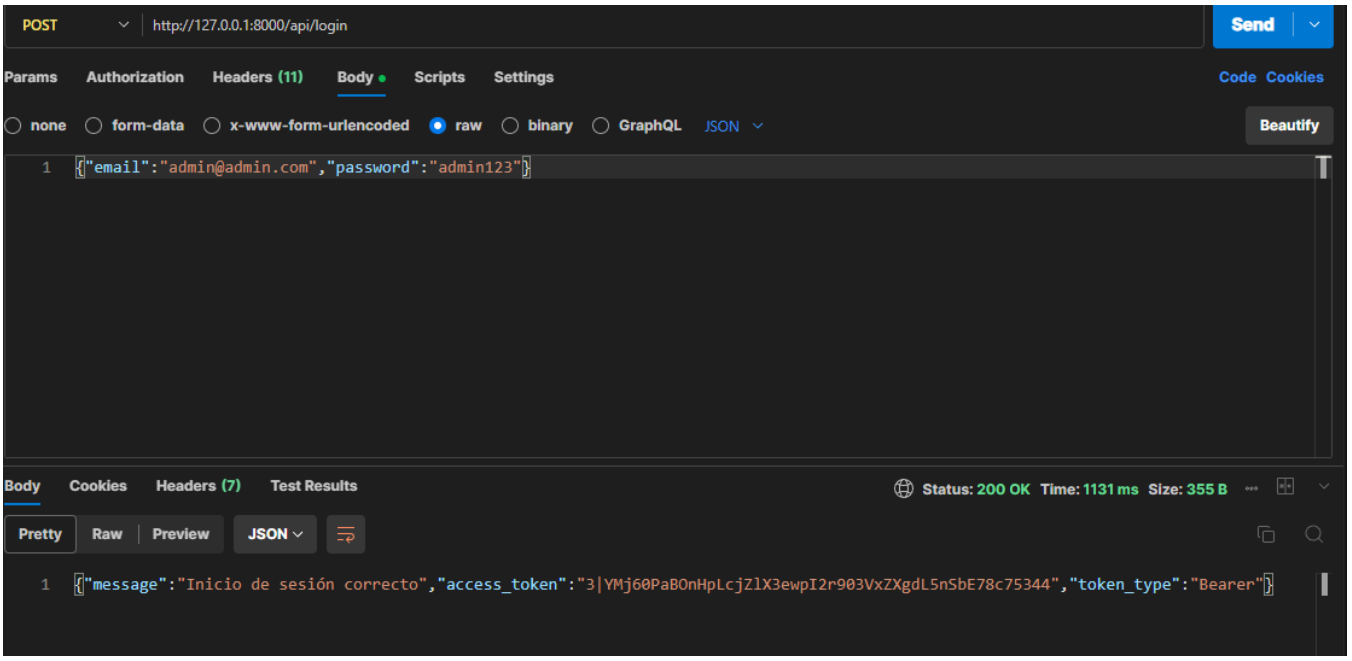
3.1 TC-001: Login exitoso con credenciales válidas

IDENTIFICADOR	TC-001 (Caso Positivo)
DESCRIPCIÓN	Usuario se valida correctamente
PRECONDICIONES	Usuario registrado con email y password

DATOS DE ENTRADA	**json**
	{
	"email": "usuario@ejemplo.com",
	"password": "Password123"
	}

PASOS A SEGUIR	1. Ir al Endpoint **POST api/login** en la colección postman 2. Ingresar datos de entrada en Body 3. Enviar
RESULTADO ESPERADO	**Status: 200 OK**
RESULTADO OBTENIDO	Inicios de sesión correcto y Token de acceso
ESTADO	**Pass**
PRIORIDAD	**Alta**

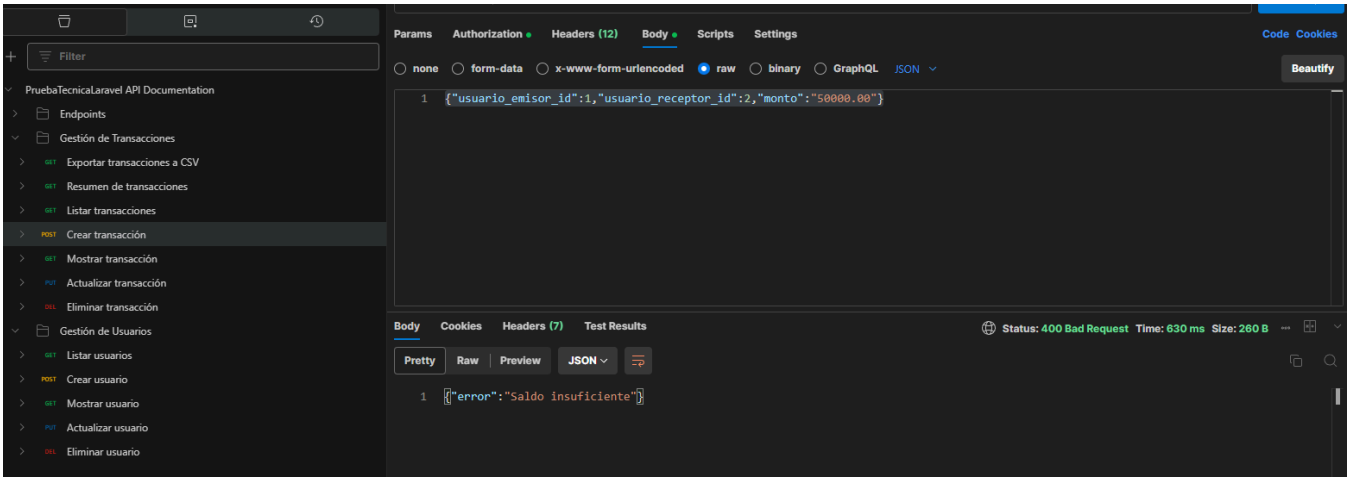
Evidencias de la prueba



3.2 TC-002: Transacción con Saldo insuficiente

IDENTIFICADOR	TC-002 (Caso Negativo)
DESCRIPCIÓN	Se realiza una transacción con saldo menor al indicado
PRECONDICIONES	Usuario registrado emisor con saldo menos a 50000.00
DATOS DE ENTRADA	**json** {"usuario_emisor_id":1,"usuario_receptor_id":2,"monto":"50000.00"}
PASOS A SEGUIR	1. Ir Gestión de transacciones/**POST api/transacciones** en la colección postman 2. Ingresar datos de entrada en Body 3. Enviar
RESULTADO ESPERADO	**Status: 400 Bad Request**
RESULTADO OBTENIDO	Error: Saldo insuficiente
ESTADO	**Pass**
PRIORIDAD	**Media-Alta**

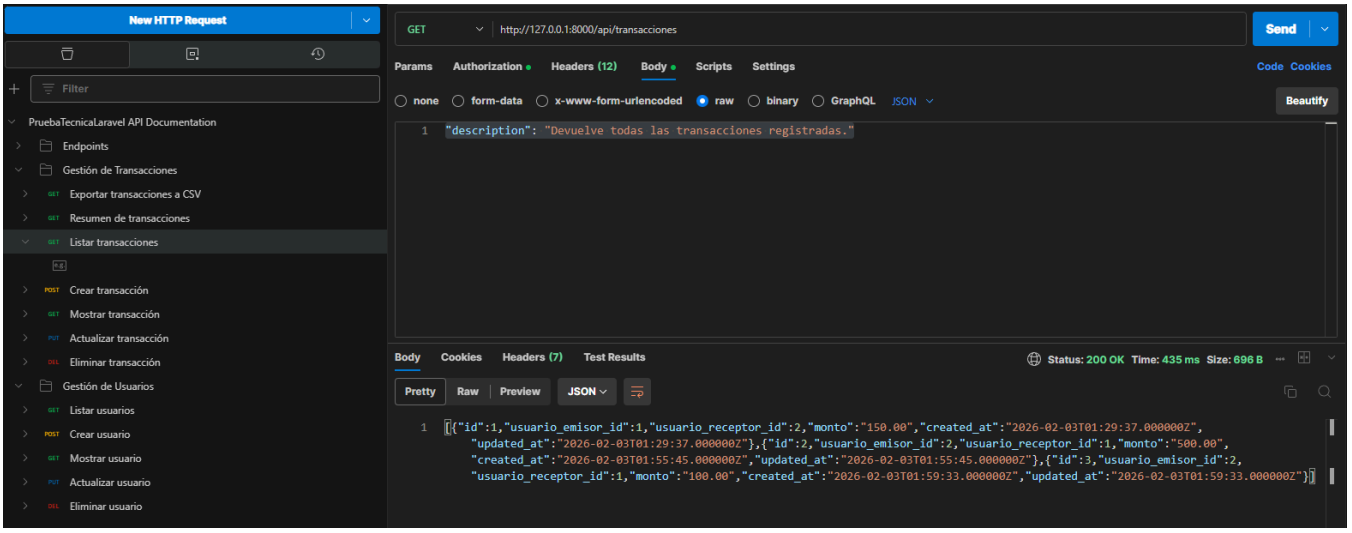
Evidencias de la prueba



3.3 TC-003: Listar transacciones

IDENTIFICADOR	TC-003 (Caso Positivo)
DESCRIPCIÓN	Devuelve todas las transacciones registradas.
PRECONDICIONES	Usuario registrado emisor con saldo menos a 50000.00
DATOS DE ENTRADA	**json** "description": "Devuelve todas las transacciones registradas."
PASOS A SEGUIR	1. Ir Gestión de transacciones/**GET api/transacciones** en la colección postman 2. Ingresar datos de entrada en Body 3. Enviar
RESULTADO ESPERADO	**Status: 200 Ok**
RESULTADO OBTENIDO	Lista de transacciones
ESTADO	**Pass**
PRIORIDAD	**Critica**

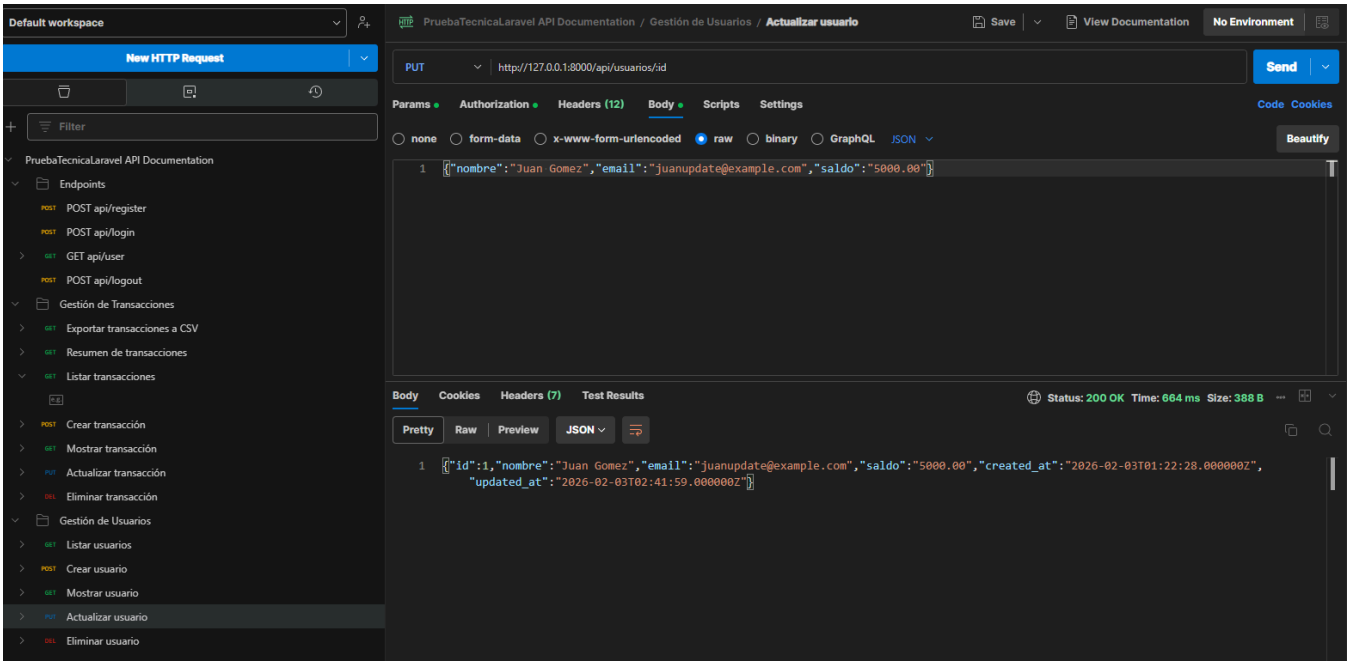
Evidencias de la prueba



3.4 TC-004: Actualizar Usuario

IDENTIFICADOR	TC-004 (Caso Positivo)
DESCRIPCIÓN	Actualiza los datos de un usuario existente.
PRECONDICIONES	El usuario debe existir
DATOS DE ENTRADA	**json** {"nombre":"Juan Gomez","email":"juanupdate@example.com","saldo":"5000.00"}
PASOS A SEGUIR	1. Ir Gestión de transacciones/**PUT api/usuarios/ID** en la colección postman 2. Modificar datos de entrada en Body 3. Enviar
RESULTADO ESPERADO	**Status: 200 Ok**
RESULTADO OBTENIDO	Usuario actualizado
ESTADO	**Pass**
PRIORIDAD	**Alta**

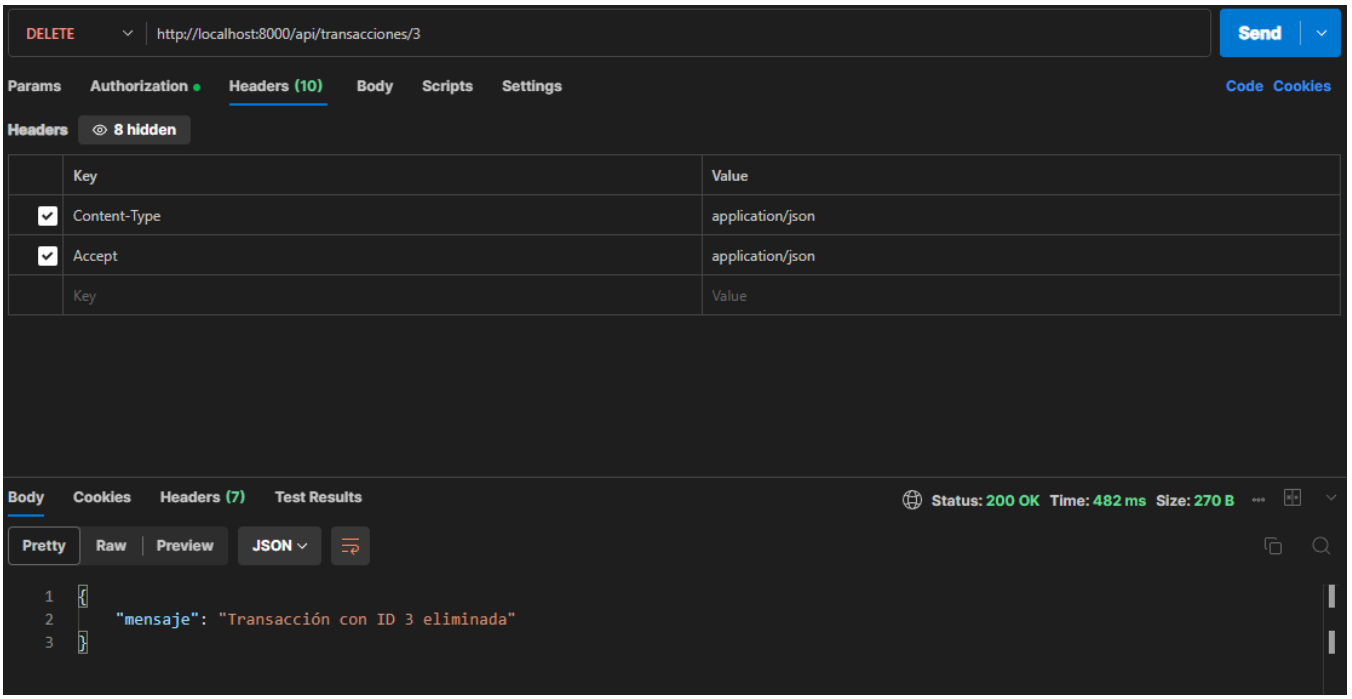
Evidencias de la prueba



3.5 TC-005: Eliminar Transacción

IDENTIFICADOR	TC-005 (Caso Positivo)
DESCRIPCIÓN	Elimina una transacción por ID
PRECONDICIONES	La transacción debe existir
DATOS DE ENTRADA	**URL** {{baseUrl}}/api/transacciones/:id
PASOS A SEGUIR	1. Ir Gestión de transacciones/**DEL api/transacciones/ID** en la colección postman 2. Reemplazar el parametro de ruta :id por el valor correspondiente. Ejm:**/api/transacciones/1** 3. Enviar
RESULTADO ESPERADO	**Status: 200 Ok**
RESULTADO OBTENIDO	Transacción Eliminada
ESTADO	**Pass**
PRIORIDAD	**Media-Alta**

Evidencias de la prueba



4. Base de Datos

Los usuarios se registran correctamente en la base de datos creada

PostMan

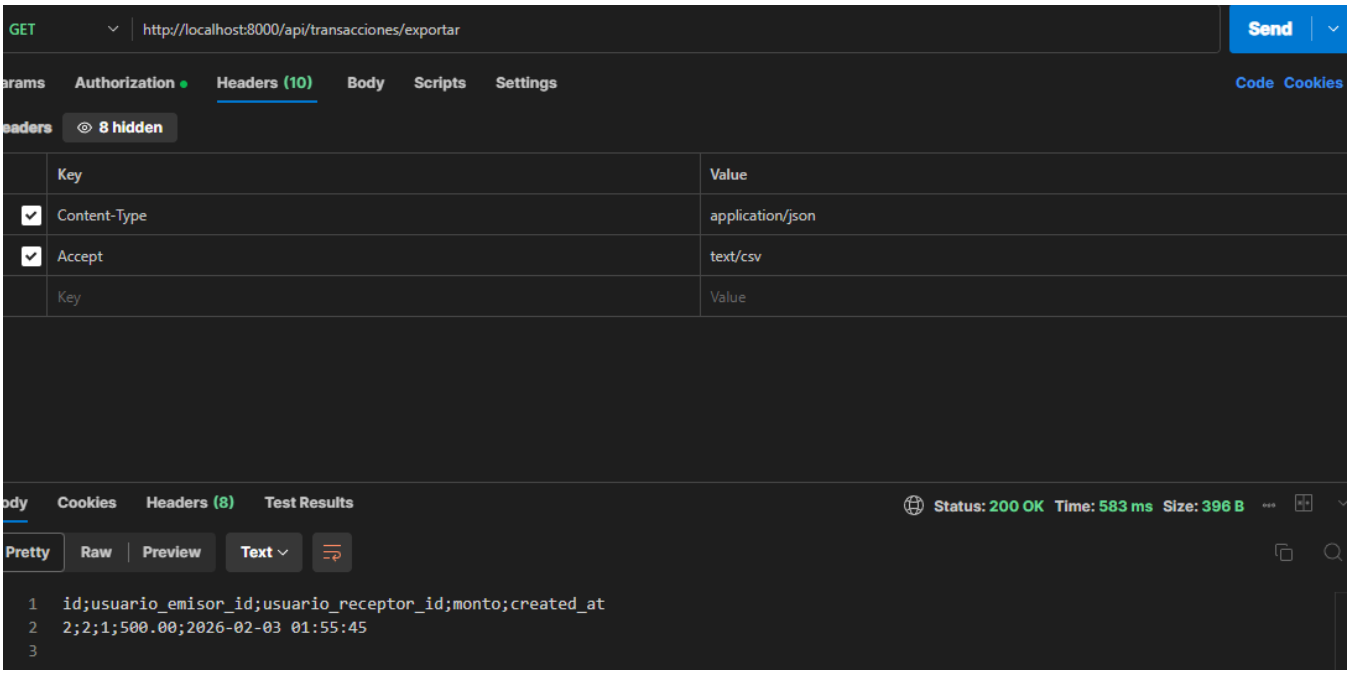

```
[{"id":1,"nombre":"Juan Gomez","email":"juanupdate@example.com","saldo":"5000.00","created_at":"2026-02-03T01:22:28.000000Z",
"updated_at":"2026-02-03T02:41:59.000000Z"},{"id":2,"nombre":"Carlos Ramos","email":"carlos@example.com","saldo":"550.00",
"created_at":"2026-02-03T01:28:21.000000Z","updated_at":"2026-02-03T01:59:33.000000Z"},{"id":3,"nombre":"Fernanda Loprez",
"email":"ferna@example.com","saldo":"1000.00","created_at":"2026-02-03T02:24:10.000000Z","updated_at":"2026-02-03T02:24:10.
000000Z"}]
```

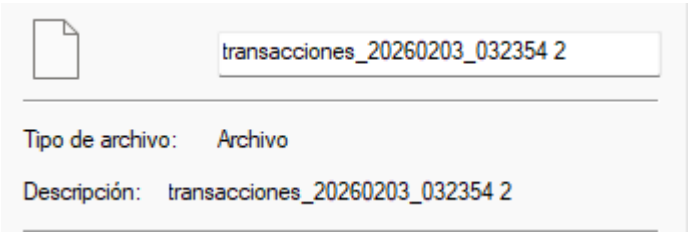

Phpmyadmin

☐	Editar	Copiar	Borrar	1	Juan Pérez	juanupdate@example.com	5000.00	2026-02-03 01:22:28	2026-02-03 02:26:48
☐	Editar	Copiar	Borrar	2	Carlos Ramos	carlos@example.com	550.00	2026-02-03 01:28:21	2026-02-03 01:59:33
☐	Editar	Copiar	Borrar	3	Fernanda Loprez	ferna@example.com	1000.00	2026-02-03 02:24:10	2026-02-03 02:24:10

5. Incidencias Encontradas

Elemento	Descripción
ID de la incidencia	BUG-002 - Error en exportación CSV.
Severidad	Media (Funcionalidad secundaria).
Comportamiento Actual	La API devuelve un status 200 OK pero entrega los datos en formato Texto plano en lugar de un archivo descargable .csv.
Comportamiento Esperado	Al solicitar la ruta con el header adecuado, el sistema debe disparar la descarga de un archivo hoja de calculo.





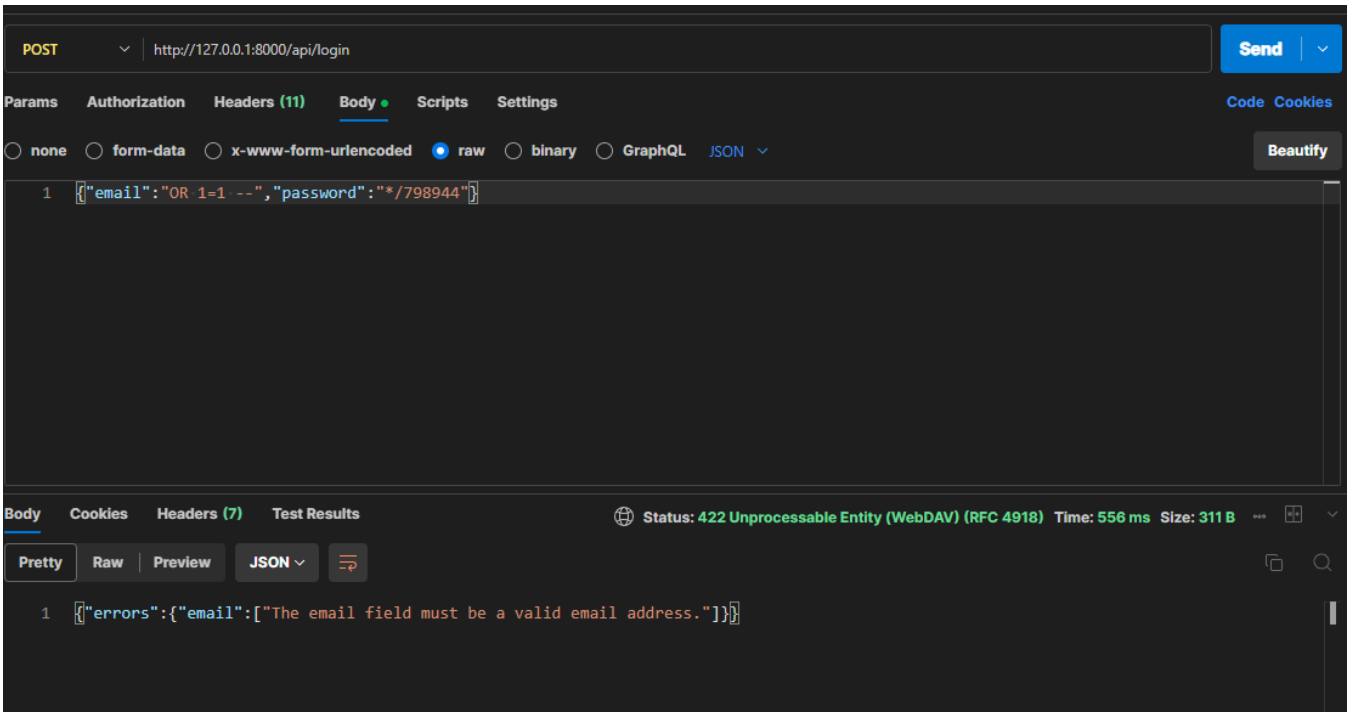
6. Enfoque General

Se implementó una estrategia de testing centrada en API dado que el sistema es una API REST pura (backend-only). El enfoque prioriza riesgo y valor de negocio, comenzando por funcionalidades críticas y seguridad.

TESTING APLICADOS

Testing Funcional (40%)

- Smoke Testing: Validación básica de que el sistema responde
- Happy Path Testing: Flujos exitosos para cada endpoint
- Negative Testing: Manejo de errores y validaciones
- End-to-End Testing: Flujos completos de usuario
- Security Testing (25% del esfuerzo)
- SQL Injection & XSS: Validación de inputs



- Authentication/Authorization: Tokens, rate limiting, control de acceso
- Data Isolation: Usuario A no accede datos de Usuario B

3. API Contract Testing (15% del esfuerzo)

- Schema Validation: Estructura consistente de respuestas JSON
- HTTP Status Codes: Códigos apropiados para cada escenario
- Headers & Content-Type: Configuración técnica correcta

Performance Básico (10% del esfuerzo)

-
- Response Times: < 2 segundos para operaciones CRUD
- Concurrencia básica: Múltiples usuarios simultáneos
- Carga inicial: Comportamiento con volúmenes pequeños de datos

Usabilidad DevEx (10% del esfuerzo)

- Error Messages: Claros y útiles para desarrolladores
- Consistency: Patrones predecibles en toda la API
- Documentación: Evaluación de /docs

PRIORIZACIÓN POR IMPACTO

ALTA PRIORIDAD (Ejecutado primero)

- Validación de saldo en transacciones (crítico negocio)
- Autenticación y tokens (seguridad básica)
- CRUD funcional (operaciones core)

MEDIA PRIORIDAD (Ejecutado después)

- Manejo de errores y validaciones
- Performance básico y tiempos respuesta
- Esquemas de respuesta JSON

BAJA PRIORIDAD (Si hay tiempo)

- Carga y stress testing
- Compatibilidad con diferentes clientes HTTP
- Documentación exhaustiva

HERRAMIENTAS UTILIZADAS

- Postman: Para testing manual organizado y colecciones
- curl: Para pruebas rápidas y scripting
- Terminal/CLI: Para comandos y automatización básica
- Git: Para documentación y control de versiones

MÉTRICAS DE CALIDAD

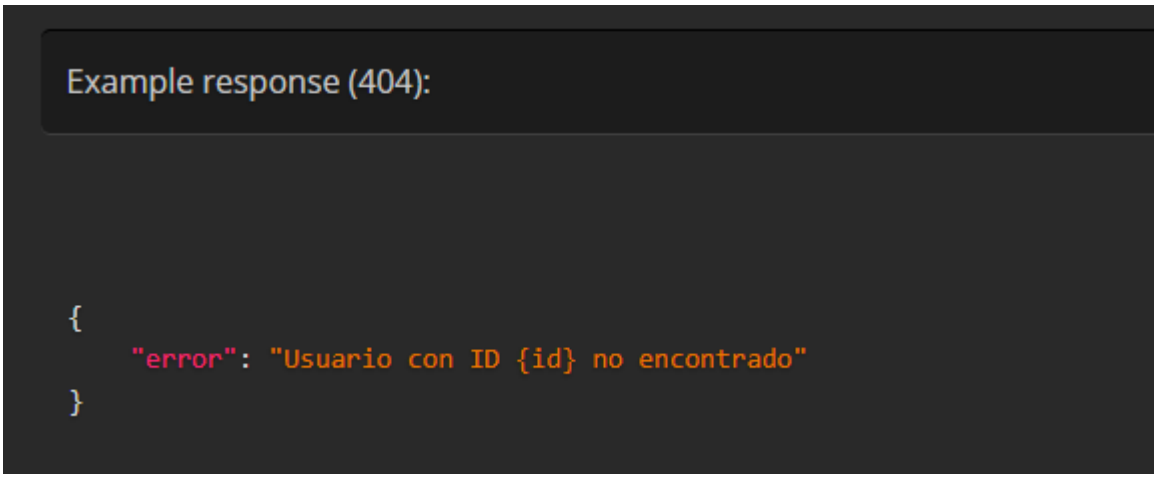
- Cobertura Endpoints: 100% de endpoints principales
- Tasa de éxito: > 90% en casos de prueba
- Tiempo respuesta: < 2s para operaciones críticas

Esta estrategia no solo valida funcionalidad, sino que: Identifica riesgos de seguridad antes de producción, mejora la experiencia del desarrollador con APIs consistentes

7. Propuestas de mejora

1. Mejora en la Semántica de Errores (Capa de UX para Desarrolladores)

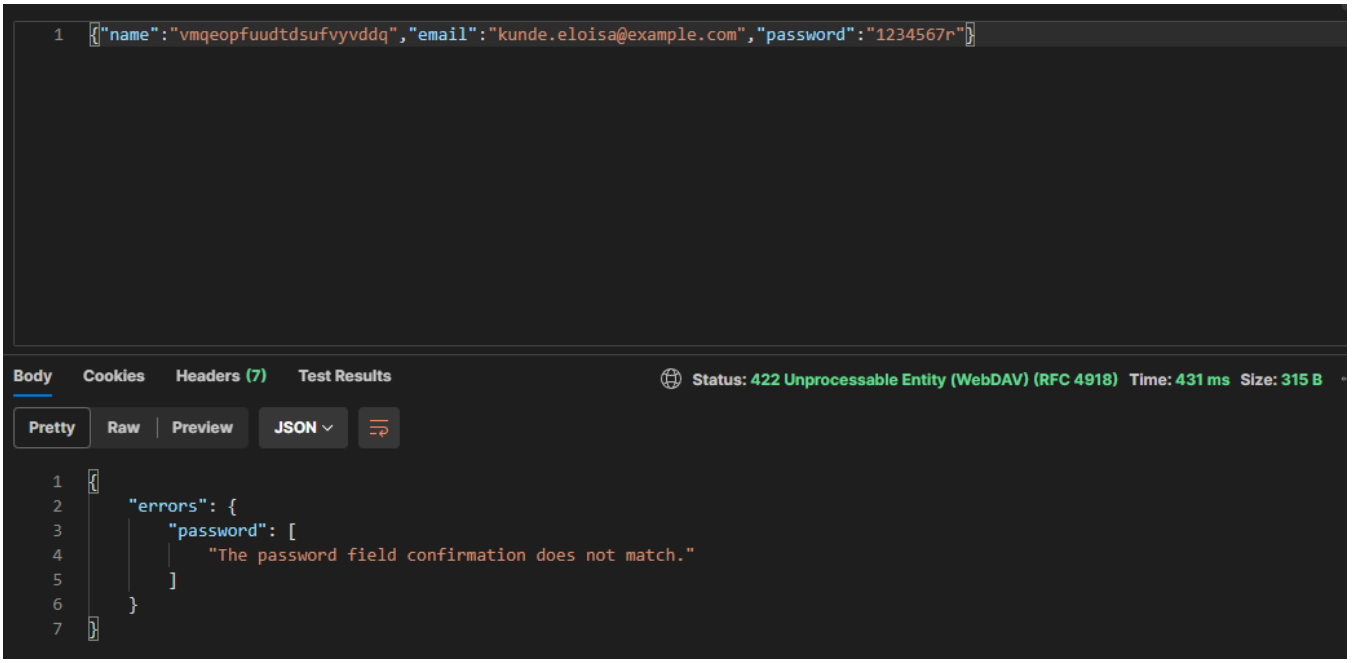
Como observamos en el error **404 disfrazado de 401**, la API confunde al usuario con mensajes de autenticación cuando el problema es de ruta.



- **Propuesta:** Implementar un manejo de excepciones global en Laravel que asegure que si un recurso no es encontrado (ModelNotFoundException), la respuesta sea siempre un **404 Real** con el mensaje "Recurso no encontrado", independientemente de si la ruta está protegida o no.

2. Estandarización de la Documentación de Registro

El fallo por falta de `password_confirmation` reveló una discrepancia entre lo que la API espera y lo que el usuario sabe.



- **Propuesta:** Actualizar la documentación técnica (Swagger/Scribe) para incluir explícitamente todos los campos de validación requeridos. Si se mantiene la regla `confirmed`, el campo

`password_confirmation` debe aparecer como obligatorio en los ejemplos de la documentación web.

3. Automatización de la Configuración de Entorno (CORS)

El error "Failed to fetch" es un bloqueador común en nuevas instalaciones.

- **Propuesta:** Incluir la configuración de CORS permisiva para entornos de desarrollo local en el archivo `.env.example` o mediante un comando de inicialización (`setup.sh`) que limpie las cachés de configuración automáticamente (`php artisan config:clear`) tras cambios en el entorno.

8. Recomendación y sugerencia

Para garantizar la estabilidad y escalabilidad del **Sistema de Gestión de Usuarios y Transacciones**, se recomiendan las siguientes acciones:

- **Implementación de Pruebas de Regresión**: Establecer un ciclo de pruebas manuales y automatizadas que se ejecute ante cada cambio en el código, asegurando que las funcionalidades existentes (como el Login y la Gestión de Transacciones) no se vean afectadas.
- **Integración en Pipeline CI/CD**: Automatizar la ejecución de las colecciones de **Postman** dentro del flujo de Integración y Despliegue Continuo (CI/CD) para validar la integridad de la API de forma automática en cada subida al repositorio.
- **Gestión de Casos de Prueba (Test Cases)**: Mantener una matriz de pruebas técnica dual:
  - **Funcional**: Para flujos de negocio y validaciones de usuario.
  - **Técnica (API)**: Específicamente diseñada en Postman para cubrir casos de borde, códigos de estado (201, 401, 422, 404) y seguridad.
- **Automatización de Interfaz (Frontend)**: Ante el desarrollo de una capa visual, se recomienda la implementación de scripts con **Selenium** o **Cypress** para simular recorridos de usuario final (End-to-End).
- **Pruebas de Rendimiento y Carga**: Ejecutar pruebas de estrés para determinar la capacidad de respuesta de la API de Laravel ante una alta concurrencia de transacciones simultáneas.