UNIVERSIDAD NACIONAL DE INGENIERÍA

Área del Conocimiento de Tecnología de la Información y la comunicación

Entregable BackEnd

Diseño de Sistemas de Internet

Elaborado por:		Tutor:
Br. Gabriel Alejandro Escorcia Chávez.	Br. Andreus Enrique Ramírez Salinas.	Ing. Cristopher Chávez Larios
Carnet: 2021-0338I	Carnet: 2021-0584I	

Noviembre, 2025

Managua, Nicaragua

índice

I. Introducción

Este documento describe la arquitectura técnica y el propósito funcional del SaaS (Software como Servicio) de gestión de tiendas. El proyecto está diseñado para operar como una plataforma multi-inquilino (multi-tenant), permitiendo a diversas empresas administrar sus operaciones de venta, inventario y sucursales de forma independiente y segura bajo una misma infraestructura.

El principal desafío que aborda esta solución es la complejidad de la normativa tributaria en Nicaragua. Las tiendas, especialmente las PYMEs, enfrentan dificultades al aplicar impuestos variables como el Impuesto al Valor Agregado (IVA), que puede ser del 15% o 0% (exento), y el Impuesto Selectivo al Consumo (ISC), que solo aplica a productos específicos.

Nuestra plataforma resuelve esta problemática proporcionando un núcleo de impuestos flexible. A diferencia de los sistemas POS tradicionales con tasas fijas, este SaaS permite a cada tienda asignar múltiples impuestos, con porcentajes personalizados, directamente a productos individuales. Esto asegura el cumplimiento tributario preciso y simplifica la gestión fiscal del comerciante.

El objetivo de esta documentación es detallar la arquitectura del sistema, el flujo de datos y las herramientas tecnológicas seleccionadas para su construcción. Servirá como una guía central para el desarrollo, mantenimiento y despliegue del backend, desde la base de datos hasta la API y su despliegue en la nube.

Si bien este documento se enfoca en el backend, la API resultante es el cerebro único que potenciará una aplicación frontend moderna. Dicha interfaz, que se detallará posteriormente, consumirá estos endpoints para ofrecer una experiencia de usuario intuitiva. El frontend se encargará de traducir la complejidad de la API en un panel de control accesible para los dueños de las tiendas.

II. Justificación

La utilidad de este proyecto radica en su respuesta directa a una necesidad crítica y desatendida en el mercado nicaragüense de software de punto de venta. Los sistemas existentes a menudo son rígidos y fallan en manejar adecuadamente las excepciones y variaciones de la Ley de Concertación Tributaria. Esto obliga a las pequeñas y medianas empresas a realizar cálculos manuales propensos a errores, aumentando su carga administrativa y el riesgo de incumplimiento fiscal.

Este SaaS sirve como una herramienta de gestión integral diseñada para centralizar las operaciones y garantizar la conformidad tributaria. Su propósito es eliminar la incertidumbre fiscal del comerciante, automatizando la aplicación de impuestos complejos. El objetivo es proporcionar una solución accesible que permita a los dueños de tiendas gestionar su inventario, sucursales y ventas con la confianza de que sus reportes fiscales son precisos.

El sistema logra esta precisión mediante su arquitectura de impuestos variables. A diferencia de las soluciones genéricas, la plataforma desvincula el impuesto del producto, permitiendo a la tienda crear un catálogo de impuestos (IVA, ISC) y luego asignarlos a productos específicos con el porcentaje que corresponda, ya sea 15%, 0% o cualquier otra tasa. Esta flexibilidad es el núcleo de la solución.

Además, su modelo como Software como Servicio (SaaS) democratiza el acceso a esta tecnología. Elimina la necesidad de que cada empresa desarrolle o compre costosas licencias de software a medida. A través de una suscripción, cualquier tienda puede acceder a una herramienta potente, segura y siempre actualizada, que escala con su crecimiento y le permite competir en igualdad de condiciones.

III. Objetivos

3.1. Objetivo General

Desarrollar una plataforma SaaS multi-inquilino que simplifique la gestión de tiendas con múltiples sucursales y garantice el cumplimiento preciso de las normativas tributarias variables de Nicaragua.

3.2. Objetivos Específicos

- Implementar el backend multi-inquilino con Strapi y PostgreSQL.
- Crear un módulo de impuestos dinámico para IVA e ISC.
- Asegurar la API con roles y permisos por tienda.
- Documentar la API con Swagger y desplegarla en Render.com.

IV. Requerimientos del Sistema

4.1.Requerimientos Funcionales

- RF01: El sistema debe permitir a los usuarios registrar una nueva cuenta y autenticarse con sus credenciales.
- RF02: El sistema debe validar que solo exista una sesión activa por usuario de forma concurrente.
- RF03: El sistema debe permitir a un usuario autenticado crear tiendas que se asignen automáticamente a su propiedad.
- RF04: El sistema debe restringir a un usuario la visualización, actualización y eliminación de tiendas que no le pertenecen.
- RF05: El sistema debe permitir la creación de múltiples sucursales asociadas a una única tienda.
- RF06: El sistema debe restringir la gestión de sucursales únicamente al usuario propietario de la tienda principal.
- RF07: El sistema debe permitir al usuario gestionar un catálogo maestro de productos asociado a su tienda.
- RF08: El sistema debe restringir la gestión de productos al usuario propietario de la tienda correspondiente.
- RF09: El sistema debe permitir a un superadministrador definir un listado global de tipos de impuestos.
- RF10: El sistema debe permitir al dueño de la tienda asignar uno o más impuestos a un producto individual.
- RF11: El sistema debe permitir al dueño de la tienda especificar un porcentaje numérico personalizado para cada impuesto asignado a un producto.
- RF12: El sistema debe permitir al usuario registrar el inventario (stock) de un producto en una sucursal específica.
- RF13: El sistema debe permitir la actualización de los niveles de stock de un producto en una sucursal.
- RF14: El sistema debe proveer una API RESTful documentada en Swagger para todas las operaciones de gestión.

4.2.Requerimientos No Funcionales

- RNF01: Los mensajes de error de la API deben ser claros, estar en español y utilizar códigos de estado HTTP estándar (ej. 400, 403, 404).
- RNF02 El tiempo de respuesta de la API para peticiones de escritura (POST, PUT, DELETE) debe ser inferior a 5 segundos.
- RNF03 El tiempo de respuesta de la API para peticiones de lectura (GET) no debe exceder los 3 segundos en una conexión estándar.
- RNF04 El sistema debe poder manejar 50 peticiones simultáneas por segundo sin degradación perceptible del servicio.
- RNF05 Las contraseñas de los usuarios deben almacenarse en la base de datos de forma cifrada e irreversible (hashed).
- RNF06 El sistema debe garantizar un aislamiento de datos completo entre inquilinos (Tiendas), impidiendo que un usuario acceda a datos que no le pertenecen.
- RNF07 Toda la comunicación con la API debe requerir un token JWT válido, excepto en los endpoints públicos de autenticación.
- RNF08 El sistema debe invalidar sesiones antiguas cuando un usuario inicia sesión en un nuevo dispositivo (política de sesión única).
- RNF09 La plataforma desplegada (Render.com) debe garantizar una disponibilidad del 99.5% para el servicio de la API.
- RNF10 El servicio de base de datos (Render Postgres) debe incluir copias de seguridad automáticas y diarias.
- RNF11 La arquitectura de despliegue en Render.com debe permitir el escalado de recursos para soportar un número creciente de tiendas.
- RNF12 El código fuente de los controladores personalizados debe estar estructurado de forma modular para facilitar el mantenimiento.
- RNF13 Todas las claves secretas (JWT, BD) deben gestionarse a través de variables de entorno y no estar escritas en el código.
- RNF14 La API debe exponer una documentación Swagger (OpenAPI) actualizada y accesible públicamente.

V. Arquitectura del Sistema

El sistema está diseñado bajo una arquitectura de backend monolítico. Este enfoque consolida toda la lógica de negocio, los endpoints de la API y la gestión de la base de datos en una única aplicación Strapi. Esta decisión centraliza el desarrollo y simplifica el despliegue, permitiendo una gestión unificada de la seguridad y el mantenimiento del sistema.

Strapi es el núcleo de esta arquitectura, funcionando como un Headless CMS. Su principal beneficio es la aceleración radical del desarrollo. Strapi genera automáticamente una API RESTful segura y un panel de administración basándose en los modelos de datos que definimos. Esto nos permite enfocar el esfuerzo de desarrollo en la lógica personalizada, como la seguridad multi-inquilino y el módulo de impuestos.

Para la persistencia de datos, se utiliza PostgreSQL. Esta base de datos relacional es fundamental para manejar las complejas relaciones de un SaaS, como Tiendas, Sucursales, Productos e Inventarios. Su robustez en el manejo de transacciones garantiza la integridad y el aislamiento de los datos entre las diferentes tiendas. Es una solución probada que escala de manera fiable junto con la aplicación.

Render.com es la plataforma de infraestructura (PaaS) que aloja tanto la aplicación Strapi como la base de datos PostgreSQL. El beneficio clave es la abstracción de la infraestructura; Render gestiona los servidores, el escalado y las redes. Al alojar la base de datos y el backend en la misma red privada de Render, se asegura una comunicación de baja latencia y alta seguridad entre los servicios.

GitHub actúa como el repositorio central de control de versiones y el motor de la integración continua. Es la fuente de verdad para todo el código de la aplicación. Está conectado directamente a Render.com, lo que establece un pipeline de CI/CD. Cada vez que se suben cambios al repositorio, se activa automáticamente un nuevo despliegue en el entorno de producción.

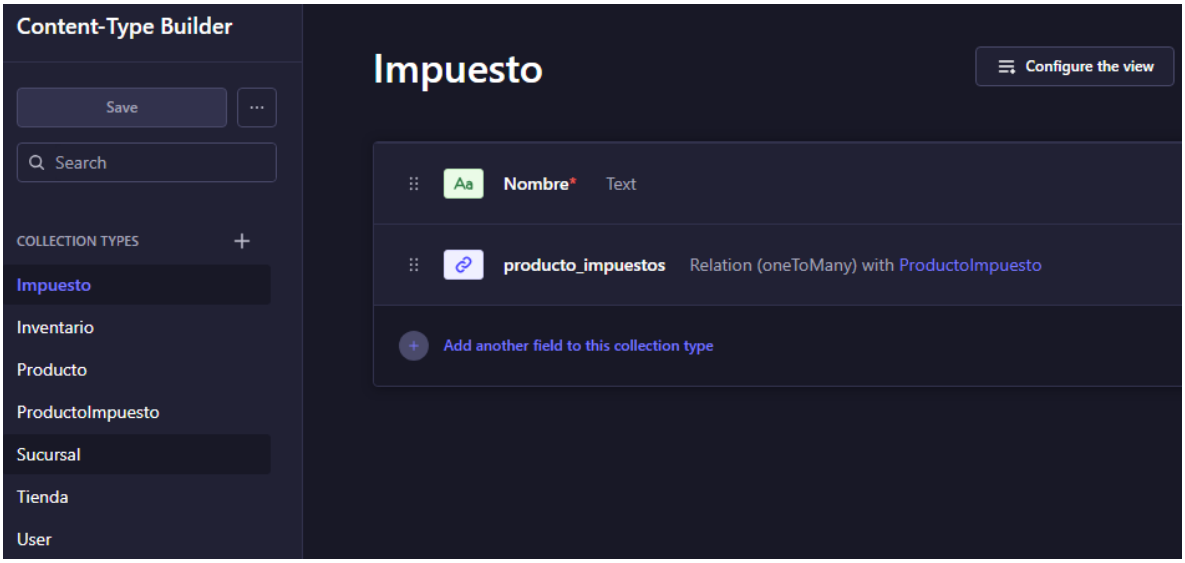
Visual Studio Code es el entorno de desarrollo local (IDE) utilizado para escribir y depurar el código de los controladores personalizados en TypeScript. Para las pruebas de la API, se utiliza Postman. Esta herramienta es crucial para validar el funcionamiento de los endpoints, simular peticiones de un cliente y, lo más importante, verificar que nuestra lógica de seguridad multi-inquilino impide el acceso no autorizado a los datos.

Finalmente, Swagger (OpenAPI) se integra a Strapi mediante su plugin de documentación. Esta herramienta genera automáticamente una documentación interactiva de la API. Este es un beneficio crítico para el desarrollo del frontend, ya que proporciona un "manual de usuario" técnico y siempre actualizado de cada endpoint, los datos que espera y las respuestas que devuelve.

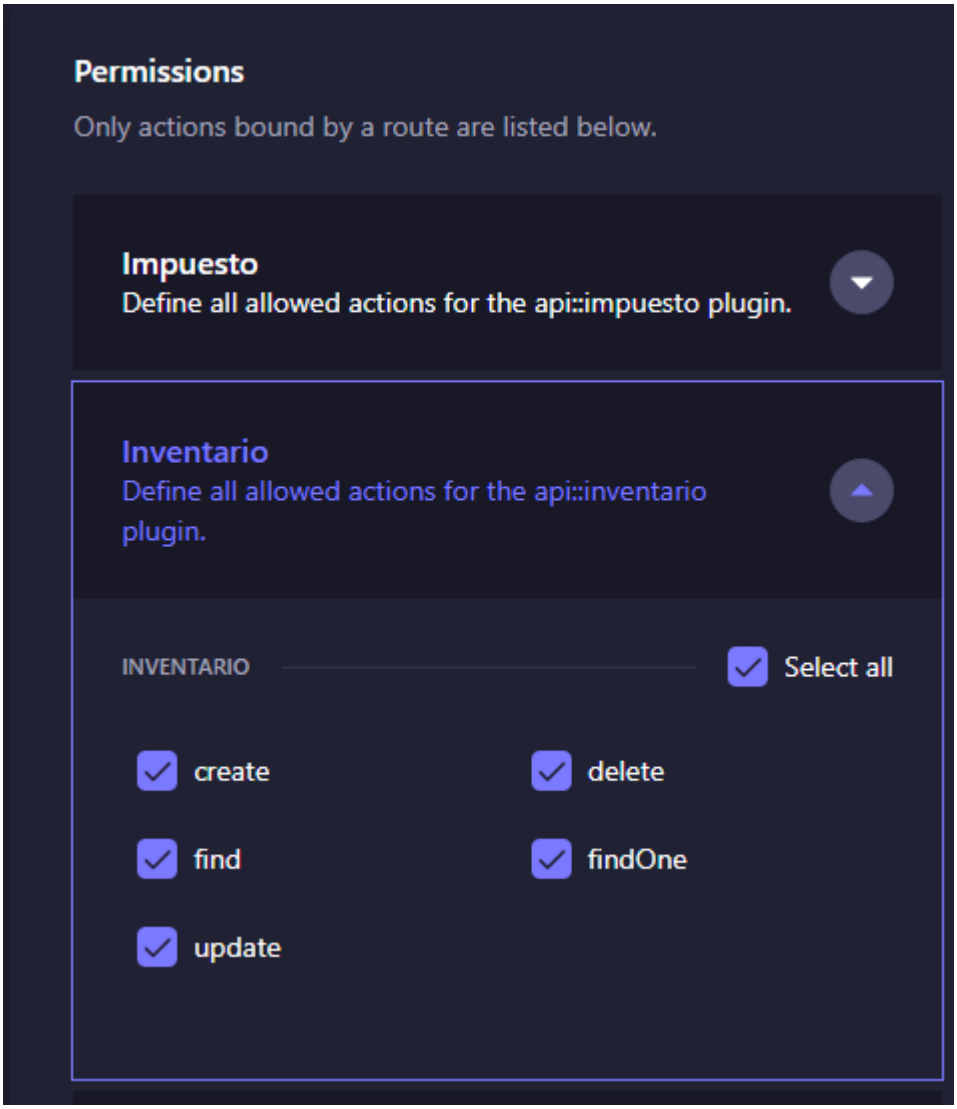
Para completar la solución, se propone el desarrollo de un frontend desacoplado que consuma la API. Esta aplicación cliente se construiría idealmente con un framework moderno como React, permitiendo una experiencia de usuario interactiva y responsiva. El frontend se encargaría exclusivamente de la presentación y la gestión del estado, proporcionando el panel de control donde los dueños de tiendas interactuarán con los datos. Esta separación total garantiza una escalabilidad independiente.

VI. Documentación de Desarrollo

Realización del modelo relacional de la API, hecho en Strapi y PostgreSQL



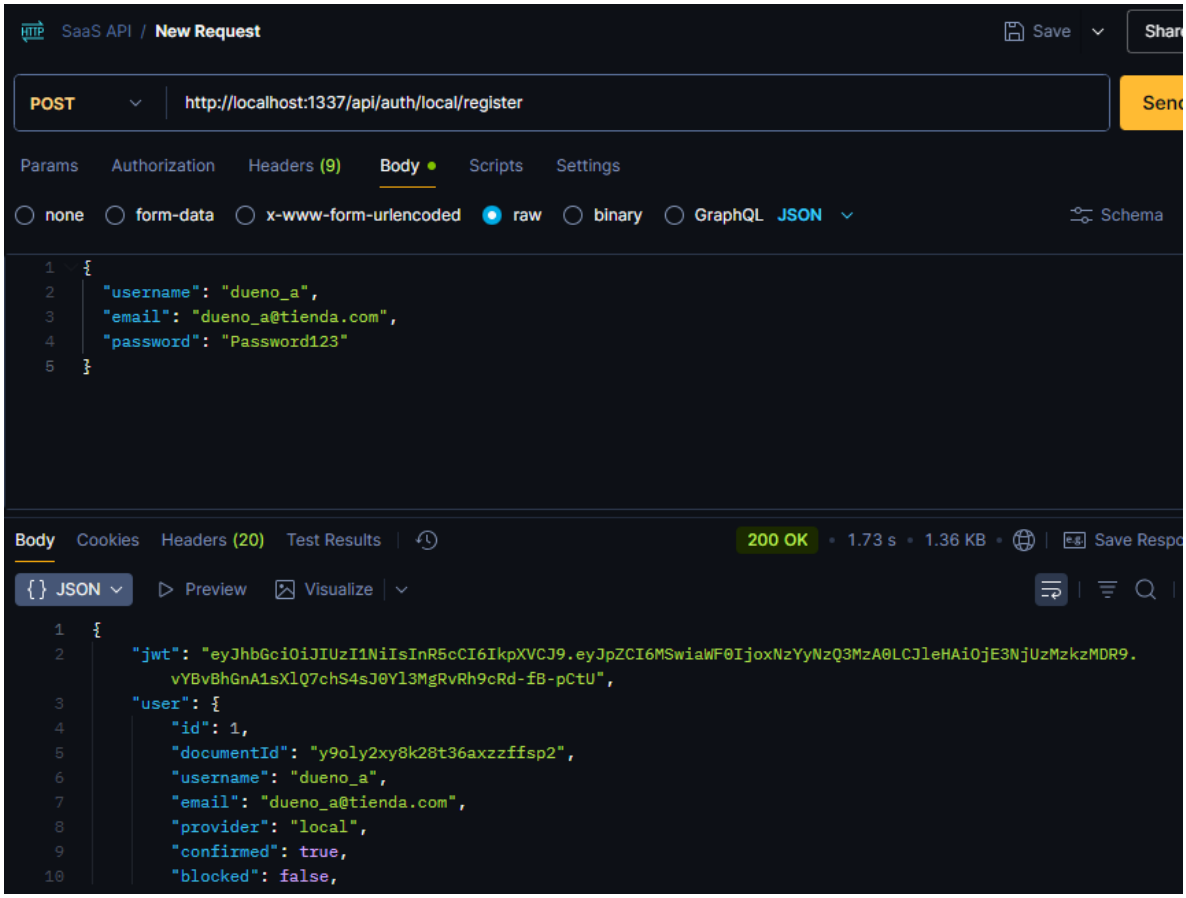
configuración de los permisos y roles.



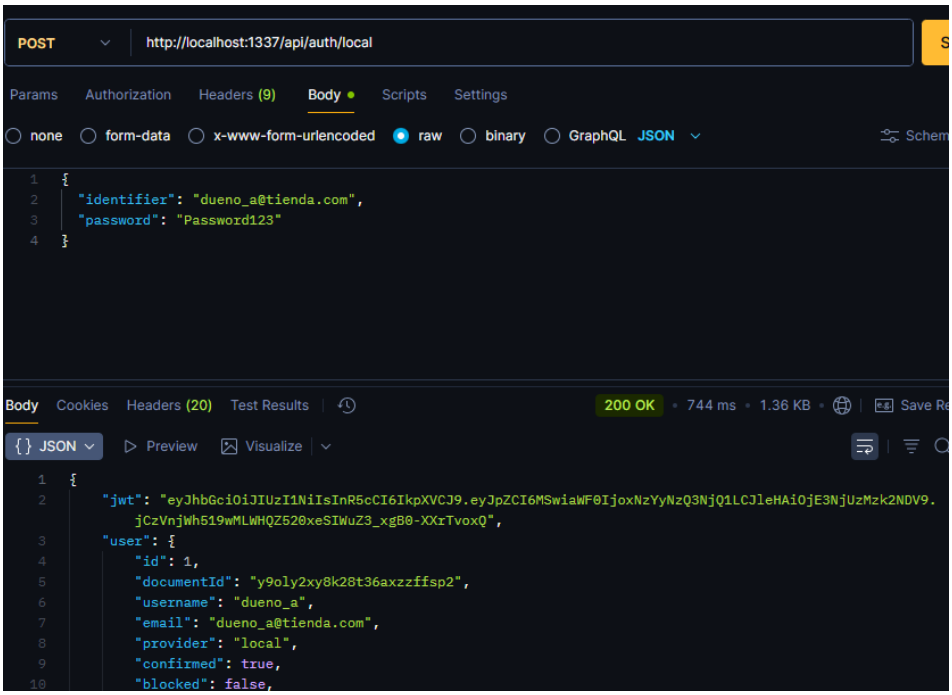
Configuración de permisos y que solo el owner (propietario de la tienda), que se estableció en la relación entre usuario y tienda, pueda ver todas las tiendas del sistema.

```
import { factories } from '@strapi/strapi';

export default factories.createCoreController('api::tienda.tienda', ({ strapi }) => ({
  async create(ctx) {
    const userId = ctx.state.user.id;

    ctx.request.body.data = {
      ...ctx.request.body.data,
      owner: userId,
    };

    const response = await super.create(ctx);
    return response;
  },

  async find(ctx) {
    const userId = ctx.state.user.id;
    ctx.query.filters = {
      ...( (typeof ctx.query.filters === 'object' && ctx.query.filters !== null ? ctx.query.filters : {}) as Record<string, any> ),
      owner: userId,
    };

    const response = await super.find(ctx);
    return response;
  },

  async findOne(ctx) {
    await this.validateOwner(ctx);
    return super.findOne(ctx);
  },

  async update(ctx) {
    await this.validateOwner(ctx);
    return super.update(ctx);
```

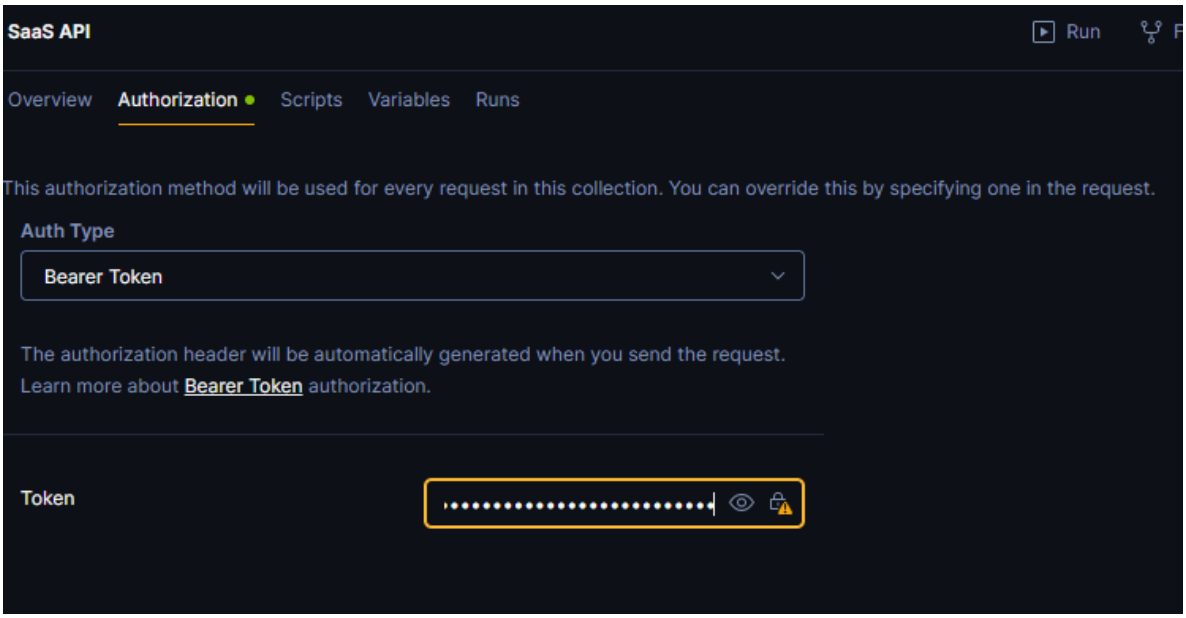
Creación de dueño en postman



Inicio de sesión de dueño a

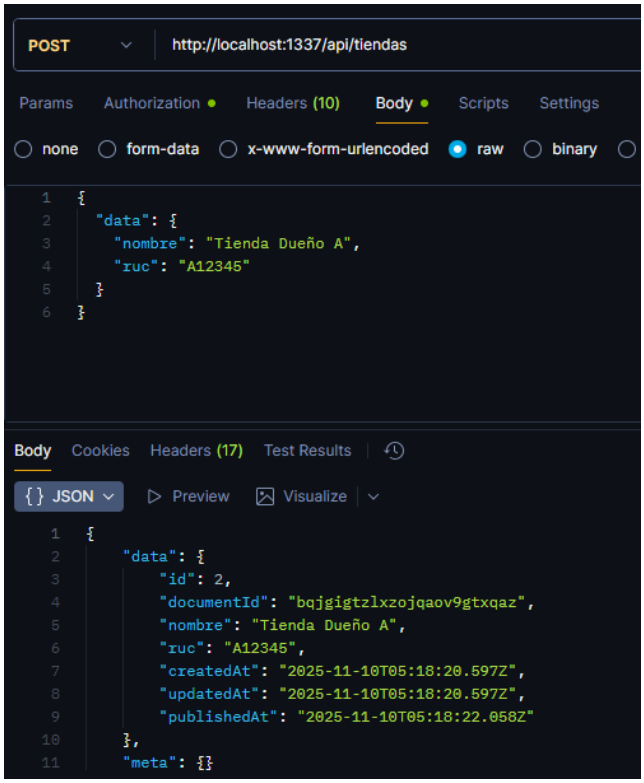


Configuración del token de auth del dueño a

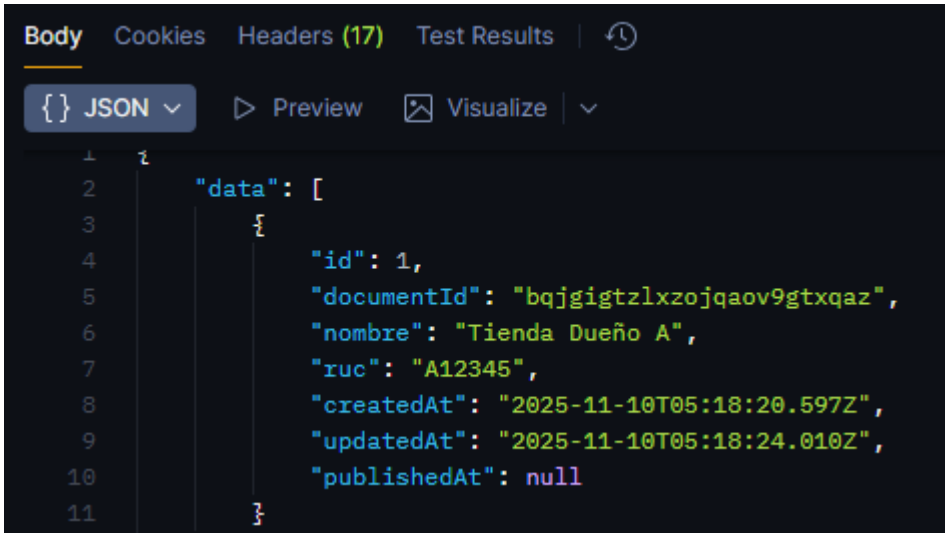


Token jwt: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNzYzMDg4Mjg1LCJleHAiOjE3NjU2ODAyODV9.QFKshFTOllbHlPQO5YlGW-PDO3VOWjS1rTlChO9Dovo

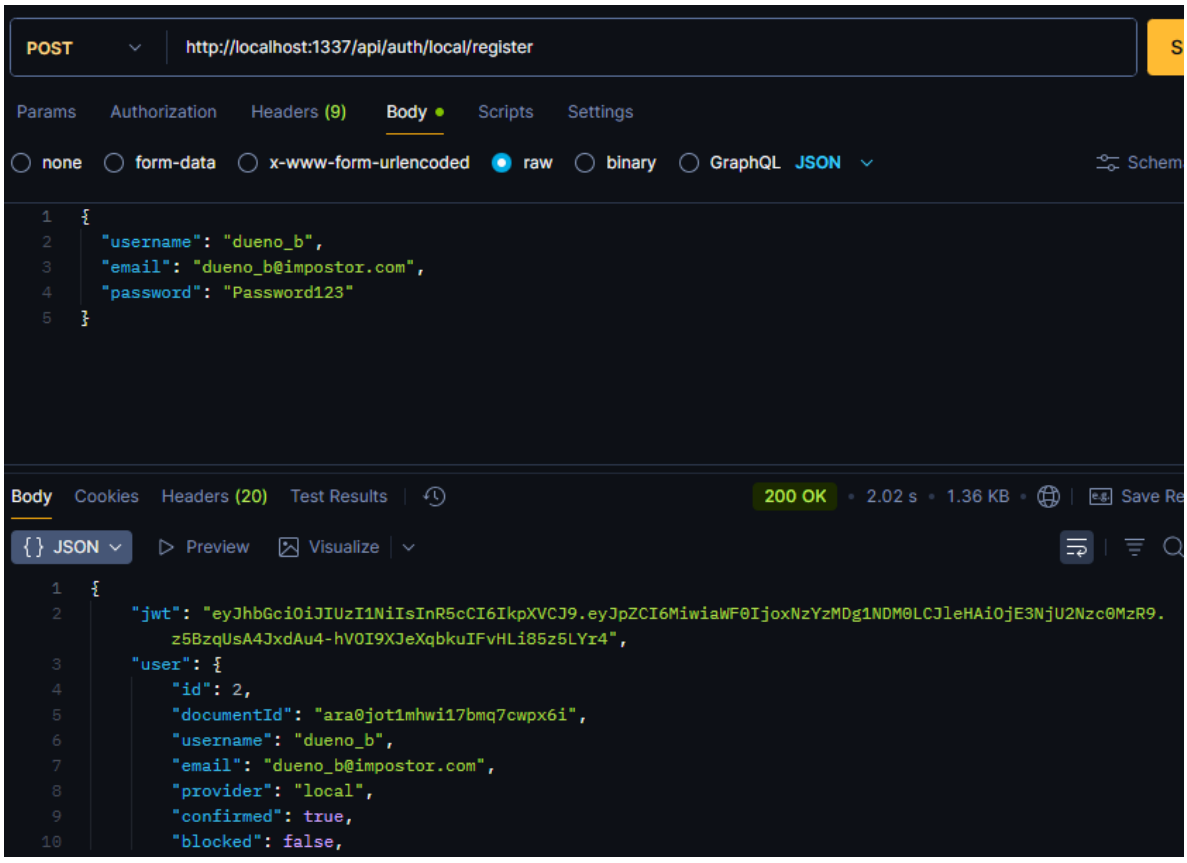
Registro de tienda a nombre del dueño a



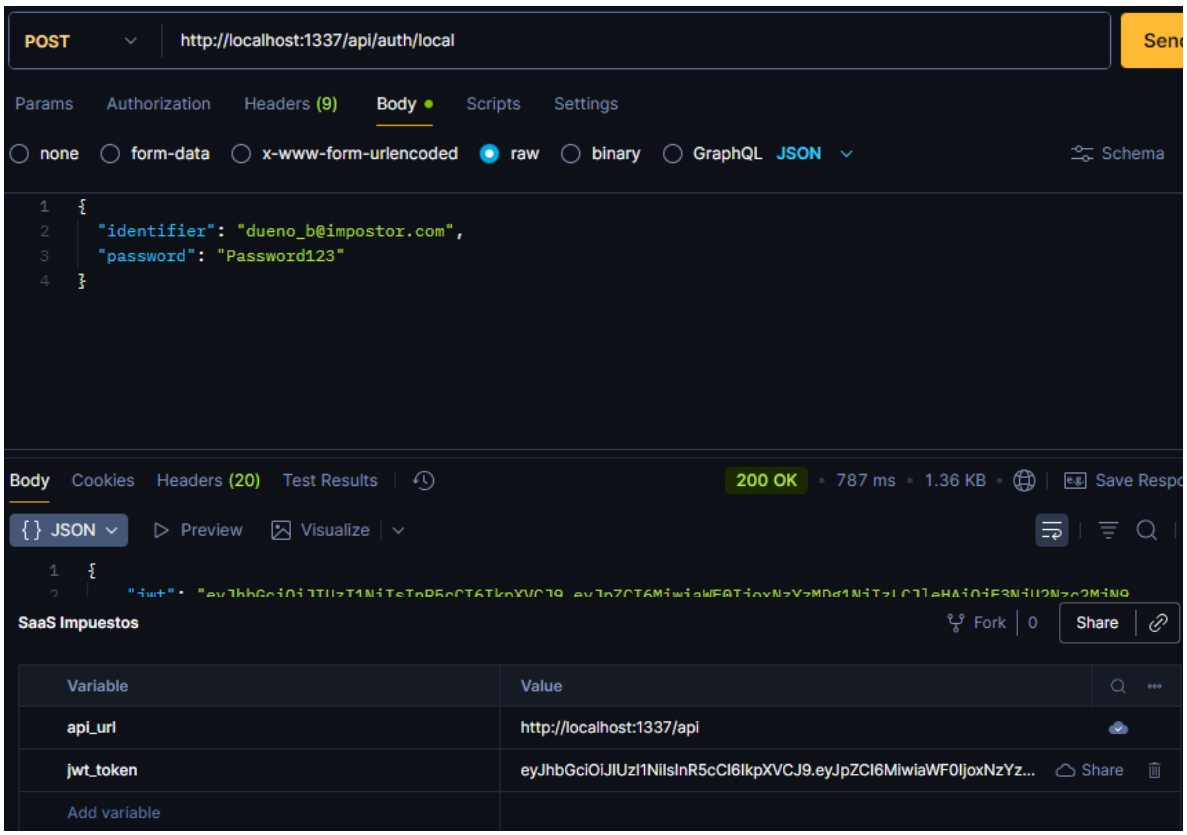
Find (GET) de las tiendas del dueño_a



Para el siguiente caso se requiere de crear un dueño_b, primero se debe poner en Postman que no tiene autorización, y posteriormente se crea el usuario.

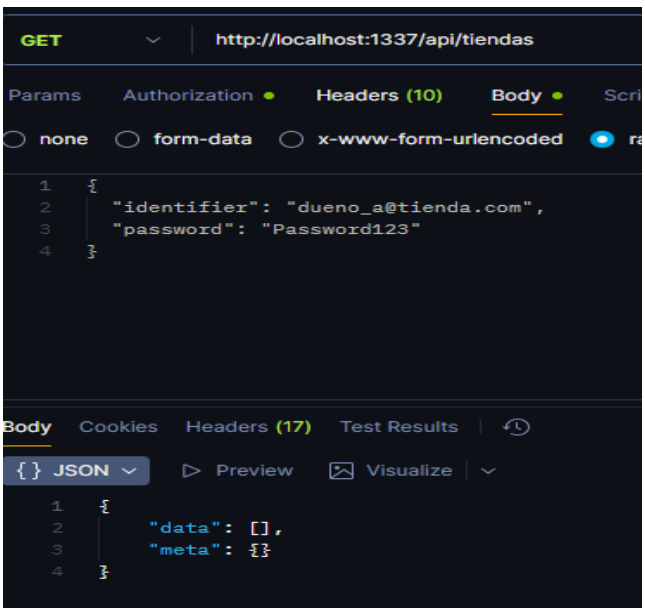


Se hace un log del dueño_b y se copia su token para comprobar si con los tokens del dueño_b se puede ver las tiendas del dueño_a.

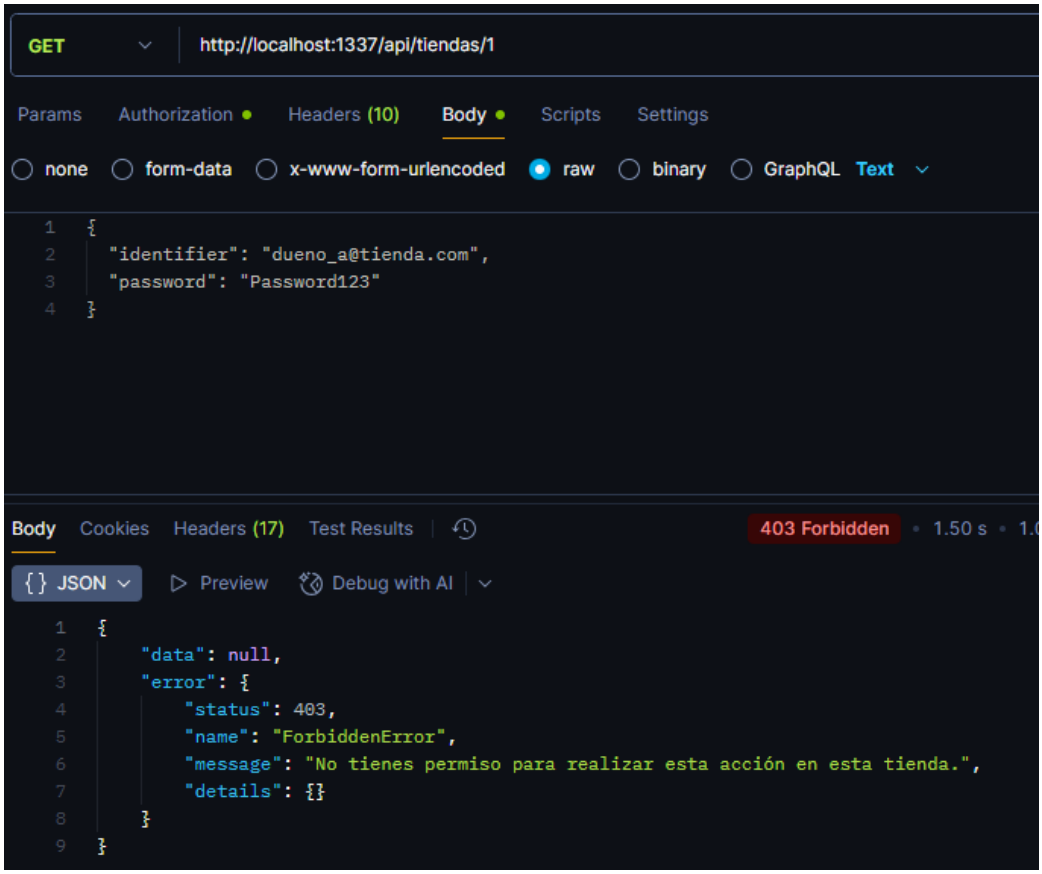


JWT token B: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNzYzMDg1NjIzLCJleHAiOjE3NjU2Nzc2MjN9.ASYBmGCegbv648bGbW9dBbUTqm-eOr3VC0iL6Zg8Jyc

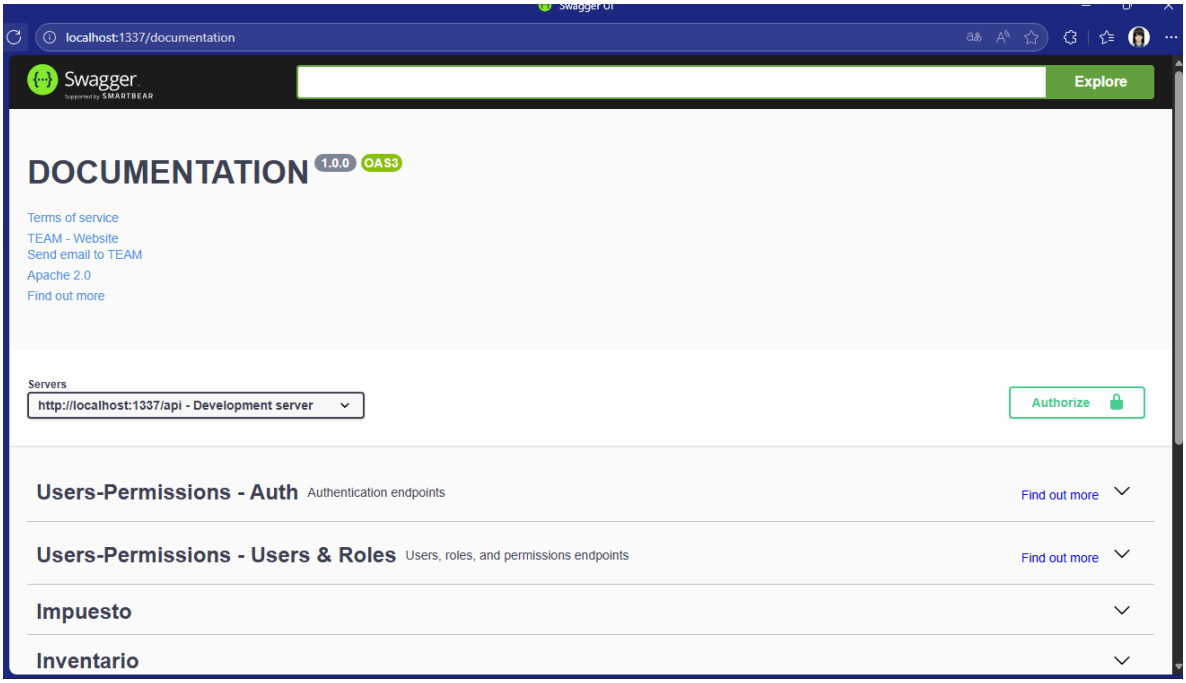
Al intentar ver las tiendas con el token del dueño_b aparece una lista vacia, debido a que no se ha registrado ninguna tienda a este Owner.



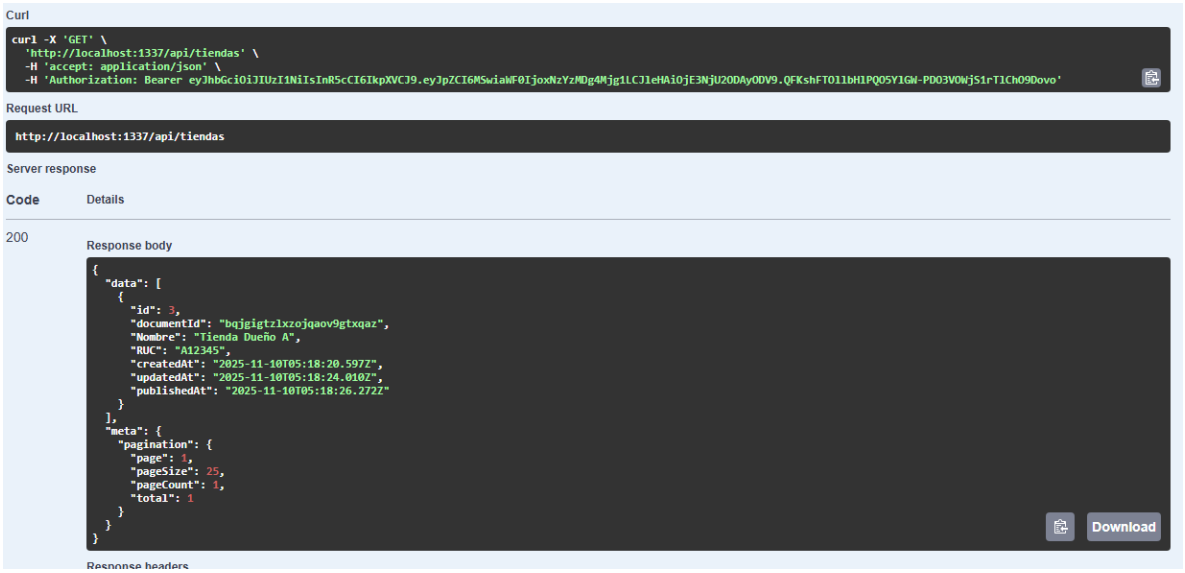
Al solicitar las tiendas del dueño_a con el token del dueño_b esta no te lo permite, debido a que no se tiene permisos.



Para usar la documentación se copio el token del dueño_a



Esto con la finalidad de que también se muestre a detalle lo que mostro Postman.



VII. Repositorio de GitHub

GabrielEscorcia27/saas-impuestos-backend: Repositorio de Proyecto de DSI de Gabriel Escorcia y Andreus Ramírez

7.1. Configuración archivo .env

```
# Server
HOST=0.0.0.0
PORT=1337

# Secrets
APP_KEYS=LZapGnhrJ4ZJcGXJW4W36A==,rVQP1KsXAbdBxvxhkbEAjg==,M9tdyKf
gr5bE/uzoz2/2yg==,oTyiCGaifWs1yjzNtNEvDw==
API_TOKEN_SALT=rnKGPhdo8VuzLRrCISZWXw==
ADMIN_JWT_SECRET=z8jEt8xg7seZz03vuYwWpw==
TRANSFER_TOKEN_SALT=VovGh9KtD19Rh9N0YJSz3w==
ENCRYPTION_KEY=O88PiHKbtQ+R5+YVwT+glQ==

# Database
DATABASE_CLIENT=postgres
DATABASE_HOST=dpg-d47ph53uibrs73d40iag-a.oregon-
postgres.render.com
DATABASE_PORT=5432
DATABASE_NAME=saas_db_jgb3
DATABASE_USERNAME=saas_user
DATABASE_PASSWORD=ykTymP9Lxcud3ANzYUd3uqkYfpii2Vni
DATABASE_SSL=true
DATABASE_FILENAME=
JWT_SECRET=tAFcR53GzqYrf9c6KLBxcA==
```