Universidad Católica Boliviana “San Pablo” Facultad de Ingeniería Carrera de Ingeniería de Sistemas

SEGUNDA EVALUACIÓN

TECNOLOGÍA WEB II 1-2025

MICROSERVICIO DE REPORTES DE DOCUMENTOS NORMATIVOS DEL PROYECTO INTEGRADOS

Presentado por:
IGNACIO RETAMOZO TORREZ

ÍNDICE

CONTENIDO DEL INFORME

ANÁLISIS DEL PROYECTO INTEGRADOR

1

IDENTIFICACIÓN CLARA DEL ENDPOINT

El endpoint consumido por el microservicio es una API REST del proyecto MIGA . Este endpoint proporciona acceso a la base de datos de documentos normativos relacionados con la alimentación saludable en entornos escolares, que es gestionada por el proyecto integrador, devolviendo todos los documentos marcados como vigentes.

GET/ api/documentos

JUSTIFICACIÓN DE LA ELECCIÓN DEL ENDPOINT

El endpoint fue seleccionado porque es la fuente principal de datos del proyecto integrador, ofreciendo información estructurada sobre documentos normativos (leyes, decretos, resoluciones, etc.). Su acceso mediante una API REST permite una integración sencilla con el microservicio, garantizando la disponibilidad de datos esenciales para las funcionalidades de búsqueda, filtrado y generación de informes requeridas por el módulo MIGA.

DESCRIPCIÓN TÉCNICA DEL ENDPOINT

- **URL**: http://localhost:4000/api/documentos
- **Método**: GET
- **Formato de datos**: La respuesta es un objeto JSON con la estructura {ok: boolean, documentos: [{id_documento: Int, nombre: String, tipo: String, fuente_origen: String, descripcion: String, importancia: String, anio_publicacion: String, enlace: String, aplicacion: String, cpe: String, jerarquia: String, concepto_basico: String, USUARIO_id_usuario: Int, vistas: Int, palabras_clave_procesadas: String, isDeleted: Boolean, usuario: { id_usuario: Int} }] }.
- **Estructura:** El endpoint devuelve una lista de documentos con campos que cumplen con los requerimientos del módulo MIGA, incluyendo identificadores únicos y tipos de documento.

DISEÑO DEL MICROSERVICIO

2

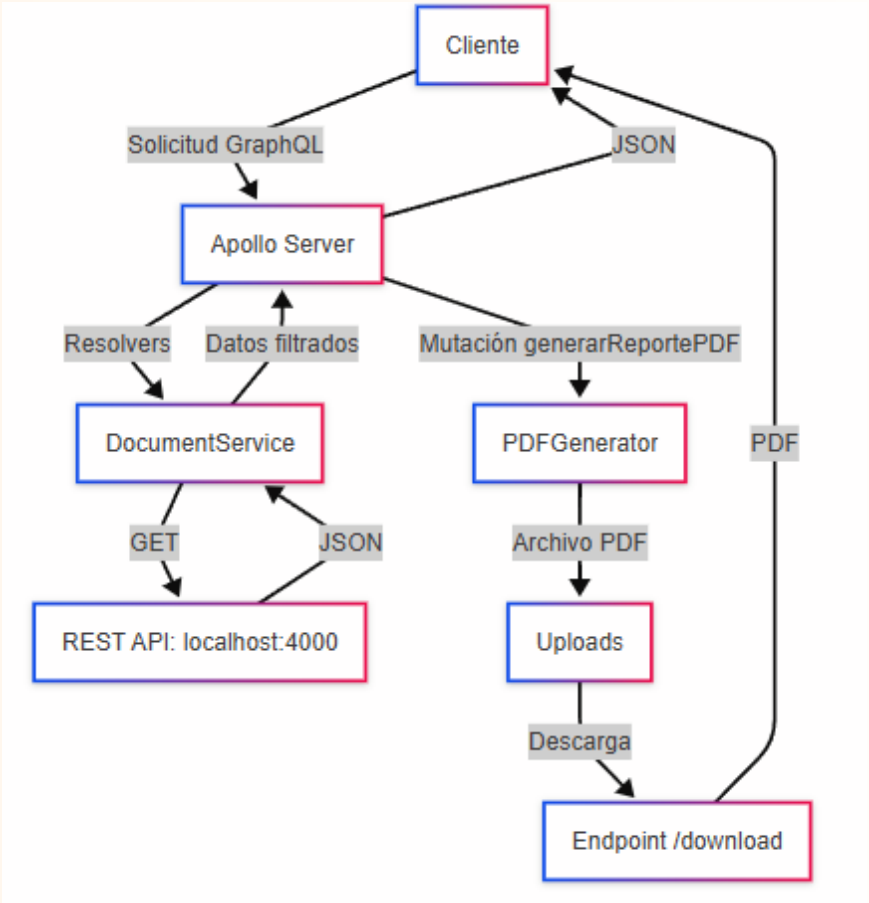
OBJETIVO DEL MICROSERVICIO

Construir un microservicio para que usuarios MIGA puedan consultar y filtrar documentos normativos del proyecto integrador, así como generar informes en formato PDF basados en criterios de búsqueda específicos.

ELECCIÓN Y JUSTIFICACIÓN DE LA TECNOLOGÍA DE COMUNICACIÓN

Se eligió GraphQL como tecnología de comunicación debido a su flexibilidad para definir consultas personalizadas, permitiendo a los clientes solicitar solo los campos necesarios de los documentos. Comparado con REST, GraphQL reduce el sobreconsumo de datos y simplifica la gestión de filtros complejos (por tipo, año, orden). La integración con Apollo Server asegura un desarrollo robusto y escalable, mientras que el consumo del endpoint REST se realiza con axios por su simplicidad y soporte para solicitudes HTTP.

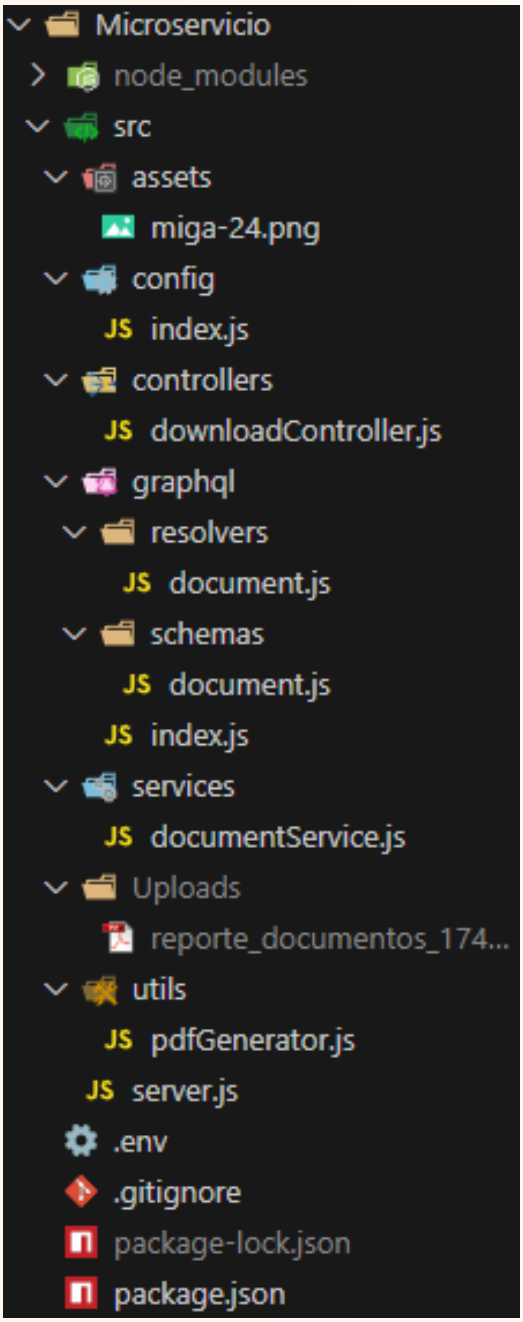
DIAGRAMA DEL FLUJO DE INTEGRACIÓN



IMPLEMENTACIÓN TÉCNICA

3

ESTRUCTURA DEL PROYECTO



- assets/
  - miga-24.png: Logo de MIGA
- config/
  - index.js: Contiene configuraciones generales del proyecto
- controllers/
  - downloadController.js: Controlador de rutas HTTP para permitir la descarga de archivos generados, como reportes PDF.
- graphql/
- resolvers/
  - document.js: Define las funciones que resuelven las operaciones declaradas en el schema de GraphQL.
- schemas/
  - document.js: Define el esquema GraphQL usando gql.
  - index.js: Usado para combinar y exportar todos los schemas y resolvers en uno solo.
- services/
  - documentService.js: Aquí se definen las funciones para obtener los documentos desde el endpoint rest y realizar filtros, ordenamientos, etc.
- Uploads/
  - Archivos generados y listos para descargar, como reportes PDF.
- utils/
  - pdfGenerator.js: Utilidad para generar reportes PDF en base a filtros y campos .

server.js
Archivo principal que levanta el servidor.

.env
Contiene variables de entorno.

.gitignore
Indica qué archivos deben ser ignorados por Git.

package.json & package-lock.json
Manejan las dependencias y scripts del proyecto.

LIBRERIAS NECESARIAS

"apollo-server": "^3.13.0", "apollo-server-express": "^3.13.0", "axios": "^1.7.7", "dotenv": "^16.4.5", "express": "^4.21.2", "moment": "^2.30.1", "pdfkit": "^0.17.1"

CODIGO BASE

CONFIG/INDEX.JS

```
Lrequire('dotenv').config();

module.exports = {
 PORT: process.env.PORT || 4001,
 REST_API_URL: process.env.REST_API_URL || 'http://localhost:4000/api/documentos',
};
```

CONTROLLERS/DOWNLOADCONTROLLER

```
const express = require('express');
const fs = require('fs');
const path = require('path');

function setupDownloadRoutes(app) {

 app.get('/download/:fileName', (req, res) => {
  try {
   const fileName = req.params.fileName;

   if (!fileName || fileName.includes('..') || !fileName.endsWith('.pdf')) {
    return res.status(400).send('Nombre de archivo inválido');
   }
   const filePath = path.join(__dirname, '../Uploads', fileName);

   if (!fs.existsSync(filePath)) {
    return res.status(404).send('Archivo no encontrado');
   }

   res.setHeader('Content-Type', 'application/pdf');
   res.setHeader('Content-Disposition', `attachment; filename="${fileName}"`);

   const fileStream = fs.createReadStream(filePath);
   fileStream.pipe(res);

  } catch (error) {
   console.error(`Error en la descarga del archivo: ${error.message}`);
   res.status(500).send('Error al descargar el archivo');
  }
 });
}
module.exports = { setupDownloadRoutes };
```

CODIGO BASE

GRAPHQL/RESOLVERS/DOCUMENT.JS

```
const documentService = require('../../services/documentService');
const pdfGenerator = require('../../utils/pdfGenerator');

module.exports = {
 Query: {
  documentos: async (_, { filter = {} }) => {
   try {
    const documentos = await documentService.getFilteredDocuments(filter);
    return documentos;
   } catch (error) {
    throw new Error(`Error al consultar documentos: ${error.message}`);
   }
  },
 },
 Mutation : {
  generarReportePDF: async (_, { filter = {}, fields = null }) => {
   try {
    const documentos = await documentService.getFilteredDocuments(filter);

    const options = {
                    fields: fields || ['id_documento', 'nombre','tipo','fuente_origen',
'anio_publicacion','aplicacion', 'vistas']
    };

    const result = await pdfGenerator.generatePDFReport(documentos, options, filter);

    return {
     fileName: result.fileName,
     downloadUrl: result.downloadUrl
    };
   } catch (error) {
    console.error(`Error al generar PDF: ${error.message}`);
    throw new Error(`Error al generar PDF: ${error.message}`);
   }
  },
 }
};
```

GRAPHQL/INDEX.JS

```
const documentSchema = require('./schemas/document');
const documentResolvers = require('./resolvers/document');

module.exports = {
 typeDefs: [documentSchema],
 resolvers: [documentResolvers],
};
```

CODIGO BASE

GRAPHQL/SCHEMAS/DOCUMENT.JS

```
const { gql } = require('apollo-server');

const documentSchema = gql`
 type User {
  id_usuario: Int
  tipo: String
  correo: String
  isDeleted: Boolean
 }

 type Document {
  id_documento: Int
  nombre: String
  tipo: String
  fuente_origen: String
  descripcion: String
  importancia: String
  anio_publicacion: String
  enlace: String
  alcance: String
  concepto_basico: String
  USUARIO_id_usuario: Int
  isfavorite: Boolean
  aplicacion: String
  cpe: String
  jerarquia: String
  isDeleted: Boolean
  vistas: Int
  palabras_clave_procesadas: String
  usuario: User
 }

 input DocumentFilter {
  tipo: String
  anio: Int
  orderBy: String
  orderDirection: String
 }
 type Query {
  documentos(filter: DocumentFilter): [Document]
 }
 type PDFGenerationResponse {
  fileName: String!
  downloadUrl: String!
 }

 type Mutation {
  generarReportePDF(
   filter: DocumentFilter,
   fields: [String]
  ): PDFGenerationResponse
 }
`;

module.exports = documentSchema;
```

CODIGO BASE

SERVICES/DOCUMENTSERVICE.JS

```
const axios = require('axios');
const { REST_API_URL } = require('../config');

class DocumentService {
 async getAllDocuments() {
  try {
   const response = await axios.get(REST_API_URL);
   if (!response.data.ok) {
    throw new Error('Fallo al consumir documentos de REST API');
   }
   return response.data.documentos;
  } catch (error) {
   if (error.response) {
    throw new Error(`Error ${error.response.status}: ${error.response.statusText}`);
   }
   throw error;
  }
 }

 async getFilteredDocuments(filter = {}) {
  try {
   let documentos = await this.getAllDocuments();

   if (filter.tipo) {
    documentos = documentos.filter(doc => doc.tipo === filter.tipo);
   }

   if (filter.anio) {
    documentos = documentos.filter(doc => {
     if (!doc.anio_publicacion || isNaN(new Date(doc.anio_publicacion).getTime())) {
      return false;
     }
     return new Date(doc.anio_publicacion).getUTCFullYear() === filter.anio;
    });
   }

   // Ordenar por el campo especificado
   if (filter.orderBy) {
    const orderField = filter.orderBy;
    documentos.sort((a, b) => {
     if (a[orderField] === null || a[orderField] === undefined) return 1;
     if (b[orderField] === null || b[orderField] === undefined) return -1;

     // Ordenamiento para campos num[ericos
     if (typeof a[orderField] === 'number') {
      return filter.orderDirection === 'DESC'
       ? b[orderField] - a[orderField]
       : a[orderField] - b[orderField];
     }

     // Ordenamiento para campos de texto
     return filter.orderDirection === 'DESC'
      ? b[orderField].localeCompare(a[orderField])
      : a[orderField].localeCompare(b[orderField]);
    });
   }

   return documentos;
  } catch (error) {
   console.error(`Error al filtrar documentos: ${error.message}`);
   throw error;
  }
 }
}

module.exports = new DocumentService();
```

CODIGO BASE

SERVERS.JS

```
const { typeDefs, resolvers } = require('./graphql');
const { PORT } = require('./config');
const { ApolloServer } = require('apollo-server-express');
const express = require('express');
const { setupDownloadRoutes } = require('./controllers/downloadController');

async function startServer() {
 const app = express();

 const server = new ApolloServer({ typeDefs, resolvers });

 //Servidor de GraphQL
 await server.start();

 // Middleware de Apollo a Express
 server.applyMiddleware({ app });

 setupDownloadRoutes(app);

 // Servir archivos estáticos
 app.use('/static', express.static('public'));

 app.listen(PORT, () => {
                    console.log(`Servidor ejecutándose en
http://localhost:${PORT}${server.graphqlPath}`);
  console.log(`Descargas disponibles en http://localhost:${PORT}/download/`);
 });
}

startServer().catch(error => {
 console.error('Error al iniciar el servidor:', error);
});
```

.ENV

```
PORT= 4001
REST_API_URL= 'http://localhost:4000/api/documentos'
```

UTILS/PDFGENERATOR

El archivo es muy grande por lo cual se obviara su anotacion aqui por razones practicas.

DESARROLLO DEL MICROSERVICIO FUNCIONAL

REQUISITO

Implementar un microservicio completamente funcional que cumpla con los objetivos del proyecto, es decir, proporcionar una interfaz para consultar reportes documentos normativos y generar informes PDF basados en los datos del proyecto integrador.

IMPLEMENTACIÓN TÉCNICA

El microservicio está desarrollado en Node.js utilizando un stack tecnológico que incluye Express para la gestión del servidor, Apollo Server para la API GraphQL, y PDFKit para la generación de informes. La arquitectura está modularizada en componentes bien definidos, cada uno con responsabilidades específicas, lo que asegura mantenibilidad y escalabilidad. A continuación, se describen los elementos clave:

- Configuración del Servidor (index.js):
  - El servidor se inicializa con Express y Apollo Server, configurando un puerto dinámico (PORT o 4001 por defecto) definido en config.js mediante dotenv para cargar variables de entorno.
  - Apollo Server se integra como middleware (server.applyMiddleware({ app })), exponiendo la API GraphQL en /graphql.
  - Se configura un endpoint adicional para descargas (/download/:fileName) mediante setupDownloadRoutes.
  - Archivos estáticos (PDFs generados) se sirven desde el directorio public (app.use('/static', express.static('public'))).

- Esquema y Resolutores GraphQL (graphql/index.js, documentSchema.js):
  - El esquema GraphQL define tipos (Document, User, PDFGenerationResponse), una consulta (documentos) y una mutación (generarReportePDF).
  - Los resolutores manejan la lógica de negocio:
    - Query.documentos: Invoca documentService.getFilteredDocuments para obtener documentos filtrados según el input DocumentFilter (tipo, año, orden).
    - Mutation.generarReportePDF: Llama a pdfGenerator.generatePDFReport para crear un informe PDF, retornando el nombre del archivo y la URL de descarga.
  - La integración con Apollo Server asegura tipado fuerte y validación automática de consultas.

- Generación de Informes PDF (utils/PDFGenerator.js):
  - La clase PDFGenerator utiliza PDFKit para crear documentos PDF con un diseño estructurado (encabezado, tabla de datos, pie de página).
  - Inicializa un directorio Uploads para almacenar PDFs temporales y programa limpiezas periódicas (_startCleanupScheduler) para eliminar archivos antiguos.
  - Genera tablas dinámicas basadas en los campos seleccionados, con estilos personalizados (colores, fuentes, bordes) y soporte para paginación automática.
- Gestión de Descargas (controllers/downloadController.js):
  - Implementa un endpoint GET (/download/:fileName) que valida el nombre del archivo (debe ser PDF, sin caracteres no permitidos) y lo sirve con encabezados adecuados (Content-Type: application/pdf, Content-Disposition: attachment).
  - Utiliza streams (fs.createReadStream) para una transferencia eficiente de archivos.
- Estructura Modular:
  - config.js: Centraliza la configuración de entorno (PORT, REST_API_URL).
  - services/DocumentService.js: Encapsula la lógica de consumo y filtrado de datos.
  - utils/PDFGenerator.js: Maneja la generación de PDFs.
  - controllers/downloadController.js: Gestiona descargas.
  - Esta separación de responsabilidades facilita pruebas, mantenimiento, y escalabilidad.

CUMPLIMIENTO

- El microservicio es completamente funcional, iniciándose en un puerto configurable y exponiendo endpoints para consultas GraphQL y descargas de PDFs.
- Cumple con el objetivo de consultar documentos normativos (vía documentos) y generar informes (vía generarReportePDF).
- La modularidad y el uso de tecnologías modernas (GraphQL, PDFKit) aseguran robustez y extensibilidad.

CONSUMO CORRECTO DEL ENDPOINT EXTERNO

REQUISITO

Consumir correctamente el endpoint externo del proyecto integrador (http://localhost:4000/api/documentos), garantizando una integración confiable y manejo adecuado de respuestas y errores.

IMPLEMENTACIÓN TÉCNICA

El consumo del endpoint externo está implementado en la clase DocumentService (services/DocumentService.js), que utiliza la librería axios para realizar solicitudes HTTP. A continuación, se detalla la implementación:

- Método getAllDocuments:
  - Realiza una solicitud GET a REST_API_URL (http://localhost:4000/api/documentos) usando axios.get.
  - Verifica que la respuesta tenga la propiedad ok: true (if (!response.data.ok) throw new Error(...)).
  - Retorna el arreglo response.data.documentos, que contiene los documentos normativos.
  - Maneja errores genéricos con un try-catch, registrando el mensaje de error en la consola y relanzándolo.
- Integración con Configuración:
  - La URL del endpoint se obtiene de REST_API_URL en config.js, permitiendo configuraciones dinámicas mediante variables de entorno (process.env.REST_API_URL).
  - Si no se define REST_API_URL, se usa un valor por defecto (http://localhost:4000/api/documentos).
- Manejo de Errores:
  - Captura errores de red o del servidor (catch (error)) y lanza un nuevo error con un mensaje descriptivo (Error al consumir: ${error.message}).
  - Registra errores en la consola para depuración (console.error).
- Uso en Resolutores:
  - La consulta documentos (graphql/index.js) invoca documentService.getFilteredDocuments, que a su vez llama a getAllDocuments para obtener los datos iniciales.
  - Esto asegura que el consumo del endpoint sea el punto de partida para todas las operaciones de datos.

CUMPLIMIENTO:

- El endpoint se consume correctamente mediante solicitudes GET, y los datos se procesan según la estructura esperada ({ ok: boolean, documentos: Array }).
- La configuración dinámica (config.js) permite adaptar el microservicio a diferentes entornos.

PROCESAMIENTO Y TRANSFORMACIÓN DE LOS DATOS

REQUISITO

Procesar y transformar los datos obtenidos del endpoint externo para cumplir con los requerimientos del módulo MIGA, incluyendo filtrado por tipo, año, nombre, palabras clave, fuente origen, y generación de informes estructurados.

IMPLEMENTACIÓN TÉCNICA

El procesamiento y transformación de datos se realiza principalmente en DocumentService para el filtrado y en PDFGenerator para la generación de informes. A continuación, se detalla cada aspecto:

- Filtrado de Documentos (DocumentService.getFilteredDocuments):
  - Filtros Implementados:
    - Tipo (filter.tipo): Filtra documentos cuya propiedad tipo coincida con el valor proporcionado (documentos.filter(doc => doc.tipo === filter.tipo)).
    - Año (filter.anio): Filtra documentos cuyo anio_publicacion (interpretado como fecha) coincida con el año especificado. Incluye validación para fechas inválidas (if (!doc.anio_publicacion || isNaN(new Date(doc.anio_publicacion).getTime())) return false).
    - Ordenación (filter.orderBy, filter.orderDirection): Ordena documentos por un campo especificado (numérico o texto) en dirección ASC o DESC. Maneja valores nulos (if (a[orderField] === null) return 1) y usa localeCompare para texto.
  - Lógica:
    - Obtiene todos los documentos con getAllDocuments.
    - Aplica filtros secuencialmente, reduciendo el conjunto de datos.
    - Ordena los resultados si se especifica orderBy.
- Transformación para Informes (PDFGenerator.generatePDFReport):
  - Estructura del PDF:
    - Encabezado: Incluye un logo (miga-24.png), título (“Reporte de Documentos”), y fecha de generación (moment().format('DD/MM/YYYY HH:mm')).
    - Información de Filtros: Muestra los criterios de filtrado aplicados (tipo, anio, orderBy) y el total de documentos.
    - Tabla de Datos: Crea una tabla dinámica con columnas basadas en los campos seleccionados (options.fields). Calcula anchos de columna proporcionales (_calculateColumnWidths) y maneja paginación automática (if (startY + estimatedRowHeight + footerHeight > doc.page.height)).
    - Pie de Página: Agrega un número de página y un texto de sistema (Sistema de Gestión Documental | Página ${currentPage}).
  - Formateo de Datos (_formatFieldValue):
    - Convierte valores nulos a “N/A”.
    - Formatea años (anio_publicacion) extrayendo el año de fechas válidas.
    - Acorta URLs largas (enlace) para mejor visualización (value.substring(0, 27) + '...').
    - Convierte booleanos a “Sí”/“No”.
  - Estilos: Usa colores personalizados (colors.primary, colors.light), fuentes (Helvetica, Helvetica-Bold), y alterna colores de filas para mejorar la legibilidad.
- Integración con GraphQL:
  - La mutación generarReportePDF recibe un filtro (DocumentFilter) y una lista de campos (fields), pasa los datos filtrados a PDFGenerator, y retorna el nombre del archivo y la URL de descarga.
  - El esquema GraphQL asegura que los datos transformados se alineen con los tipos definidos (Document, PDFGenerationResponse).

CUMPLIMIENTO

- El filtrado por tipo, anio, y ordenación está correctamente implementado, cubriendo los requerimientos de búsqueda para reportes.
- La generación de informes transforma los datos en tablas estructuradas con formato profesional, soportando campos personalizados y paginación.

EXPOSICIÓN DE ENDPOINT PROPIO

IMPLEMENTACIÓN TÉCNICA

El microservicio expone dos endpoints principales, ambos funcionales y alineados con los objetivos del proyecto:

- Endpoint GraphQL (http://localhost:4001/graphql):
  - Implementación:
    - Configurado mediante Apollo Server en index.js (server.applyMiddleware({ app })).
    - Soporta consultas y mutaciones definidas en documentSchema.js:
      - Query.documentos(filter: DocumentFilter): [Document]: Recupera documentos filtrados.
      - Mutation.generarReportePDF(filter: DocumentFilter, fields: [String]): PDFGenerationResponse: Genera un informe PDF y retorna su nombre y URL de descarga.
    - El esquema GraphQL (documentSchema.js) define tipos fuertemente tipados (Document, User, PDFGenerationResponse, DocumentFilter), asegurando validación automática de entradas y respuestas.
  - Funcionalidad:
    - Permite a los clientes especificar filtros (tipo, anion, orderBy) y campos deseados, aprovechando las ventajas de GraphQL para consultas flexibles.
    - Retorna datos en formato JSON, accesible vía herramientas como Postman, GraphiQL, o aplicaciones frontend.
- Endpoint de Descarga (http://localhost:4001/download/:fileName):
  - Implementación:
    - Definido en controllers/downloadController.js (setupDownloadRoutes).
    - Valida el nombre del archivo para prevenir accesos no autorizados (if (!fileName.endsWith('.pdf') || fileName.includes('..'))).
    - Sirve archivos PDF desde el directorio Uploads usando streams (fs.createReadStream) para eficiencia.
    - Configura encabezados HTTP adecuados (Content-Type: application/pdf, Content-Disposition: attachment).
  - Funcionalidad:
    - Permite a los clientes descargar los PDFs generados por la mutación generarReportePDF.
    - Los archivos están disponibles temporalmente (1 hora, según fileRetentionTime en PDFGenerator).

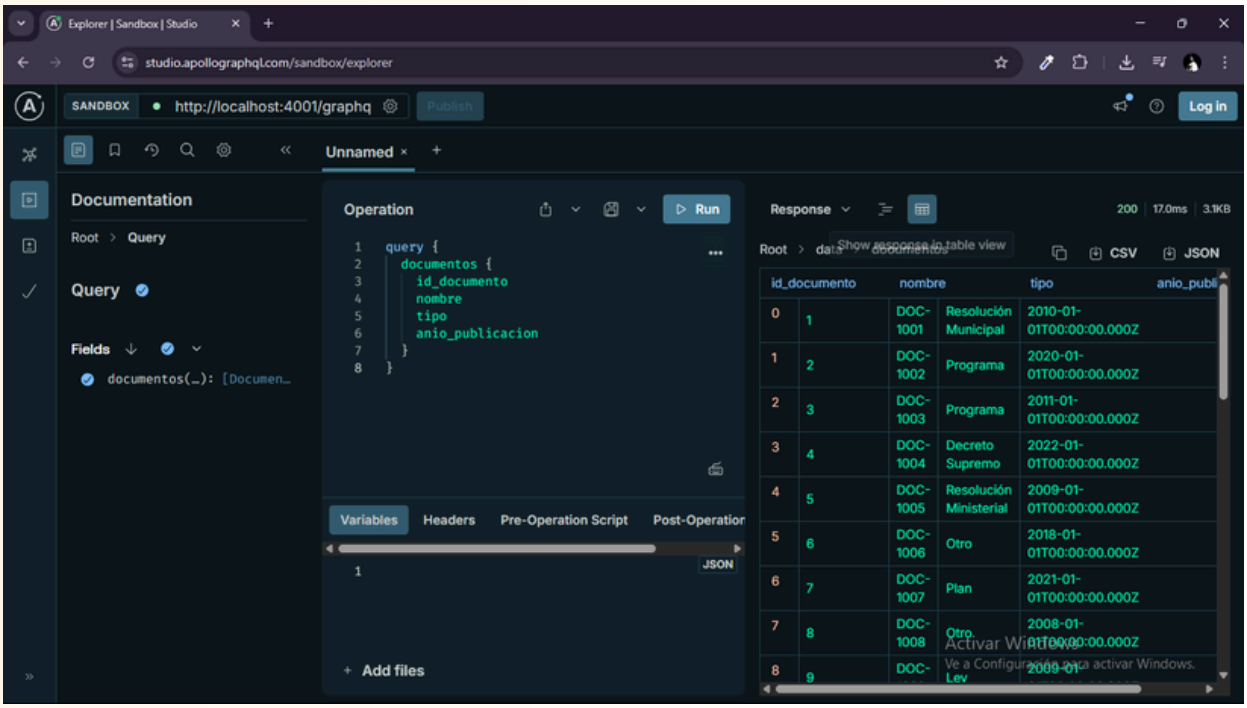
PRUEBAS Y DOCUMENTACIÓN

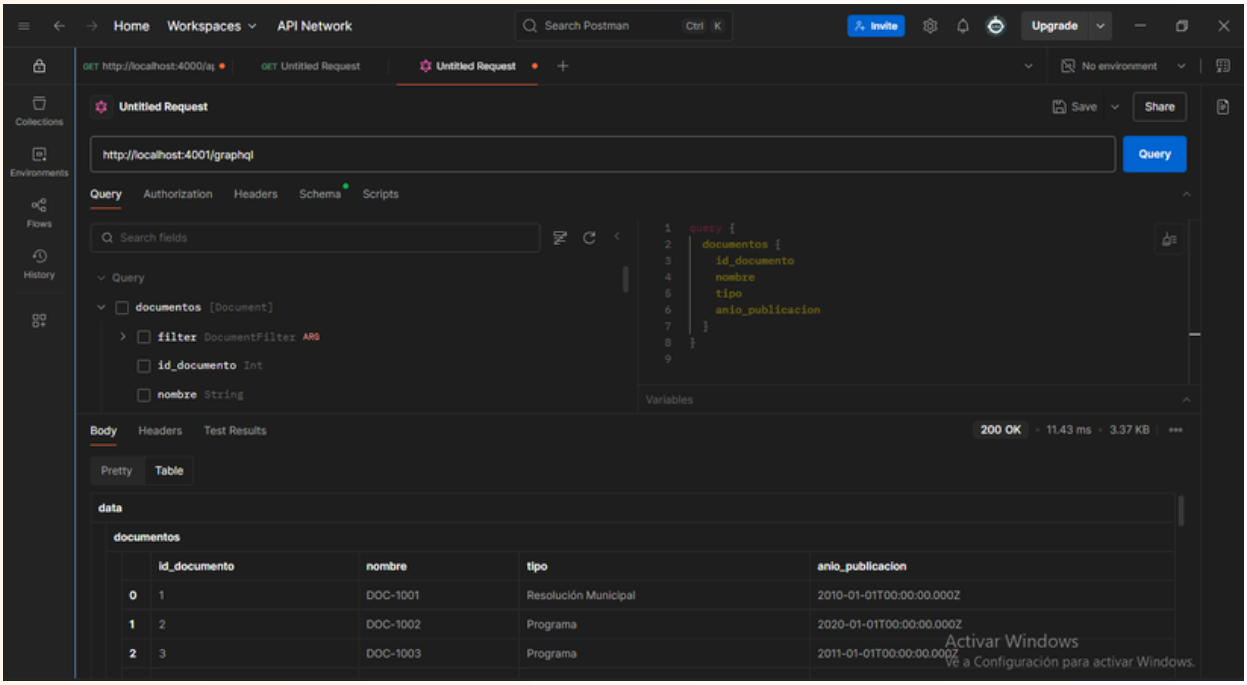
4

CASOS DE PRUEBA Y EVIDENCIAS FUNCIONALES

CASO 1: CONSULTA DE DOCUMENTOS SIN FILTRO

Propósito: Verificar que la consulta documentos retorna todos los documentos disponibles cuando no se aplican filtros. Se utilizara GraphQL Playground y PostMan

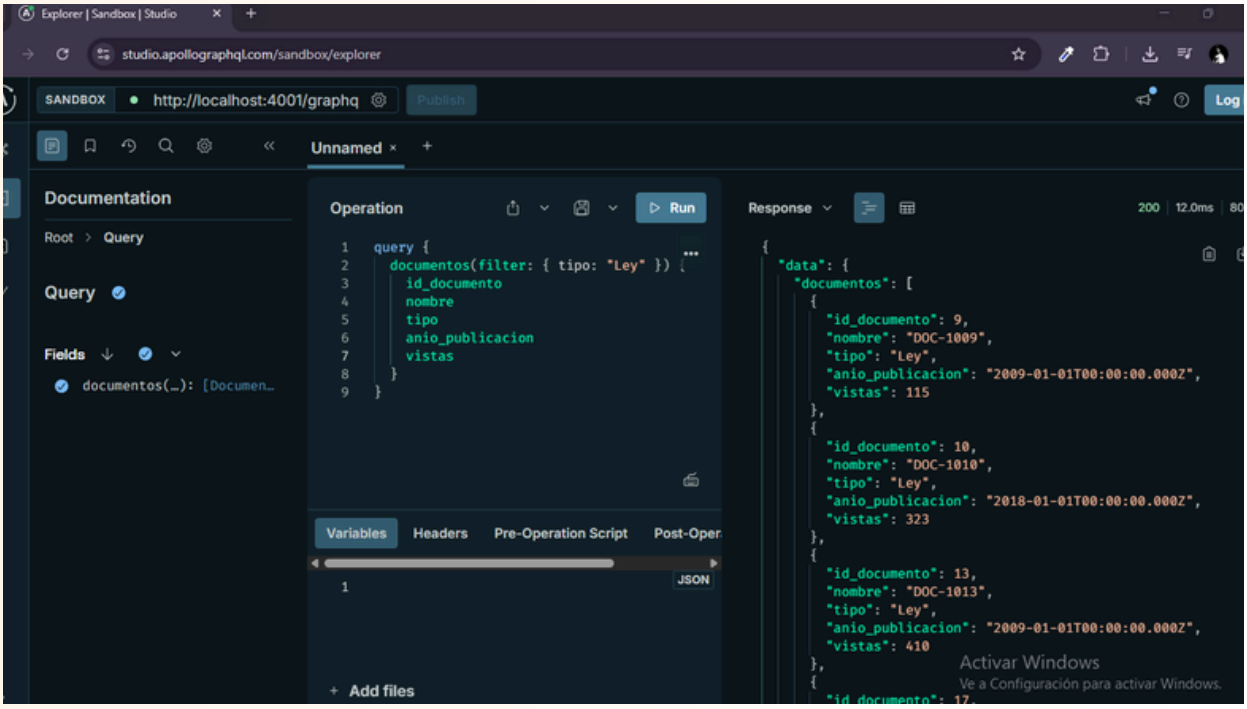


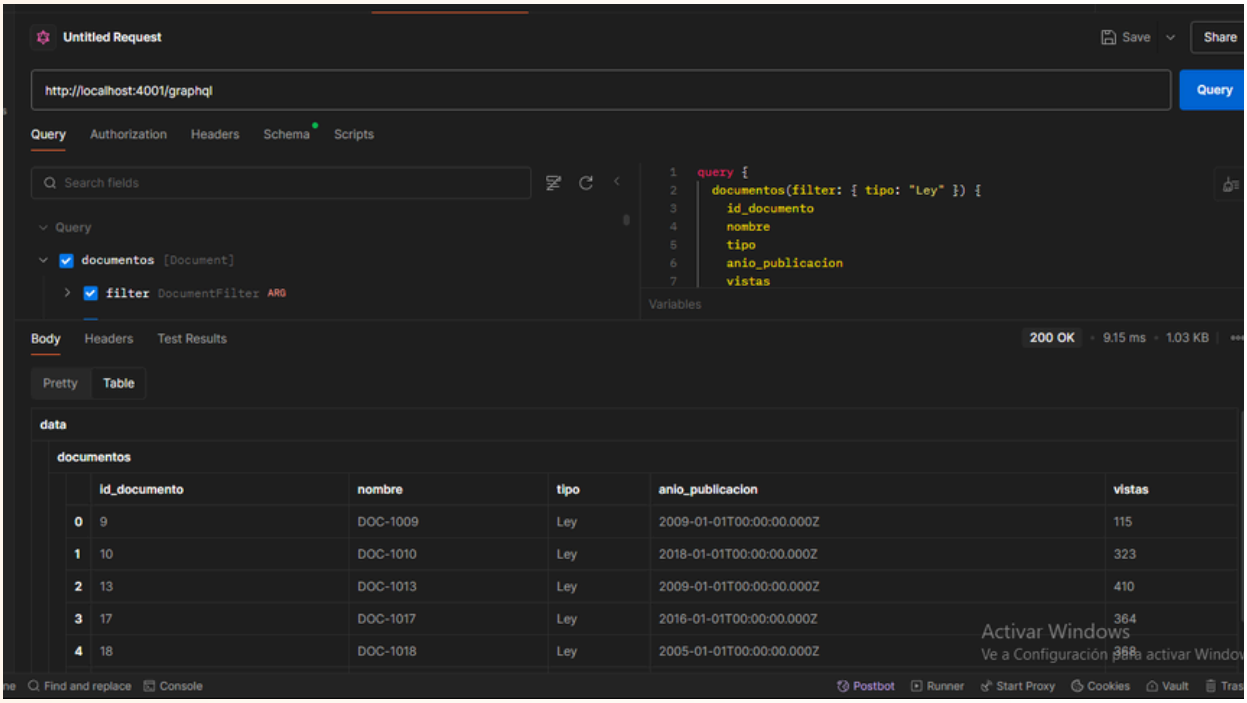


CASOS DE PRUEBA Y EVIDENCIAS FUNCIONALES

CASO 2: CONSULTA DE DOCUMENTOS CON FILTRO POR TIPO

Propósito: Verificar que la consulta documentos filtra correctamente los documentos por el campo tipo (por ejemplo, "Ley"). Se utilizara GraphQL Playground y PostMan

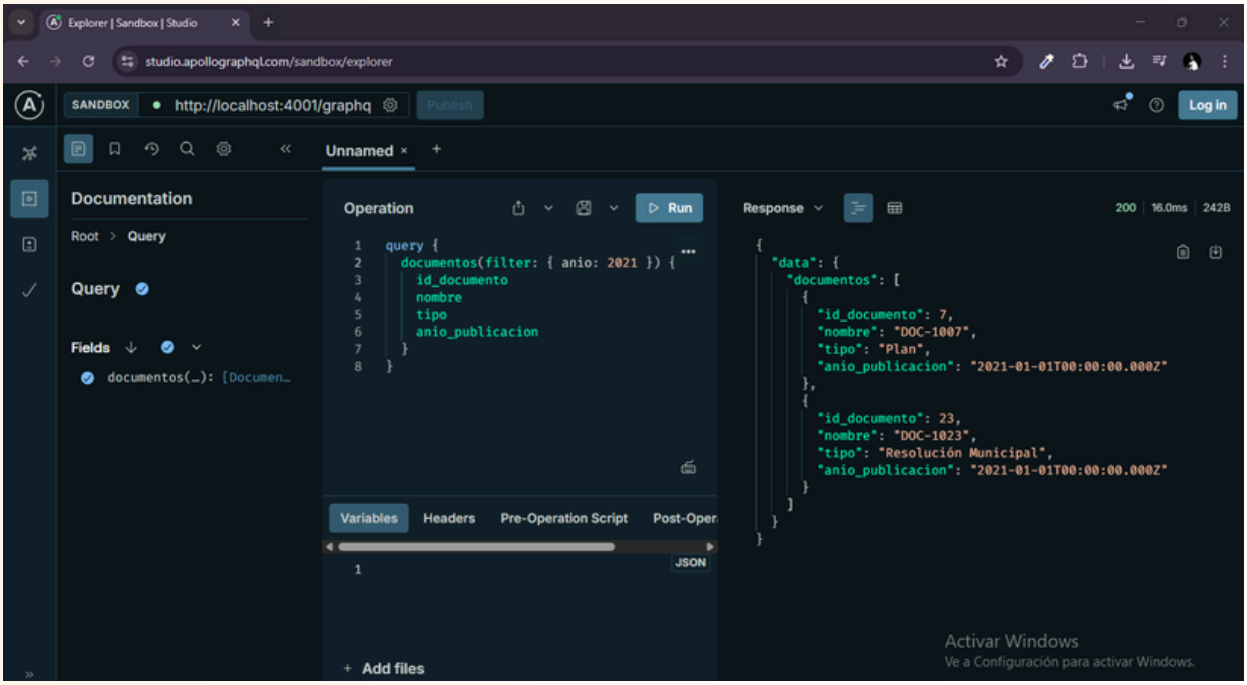


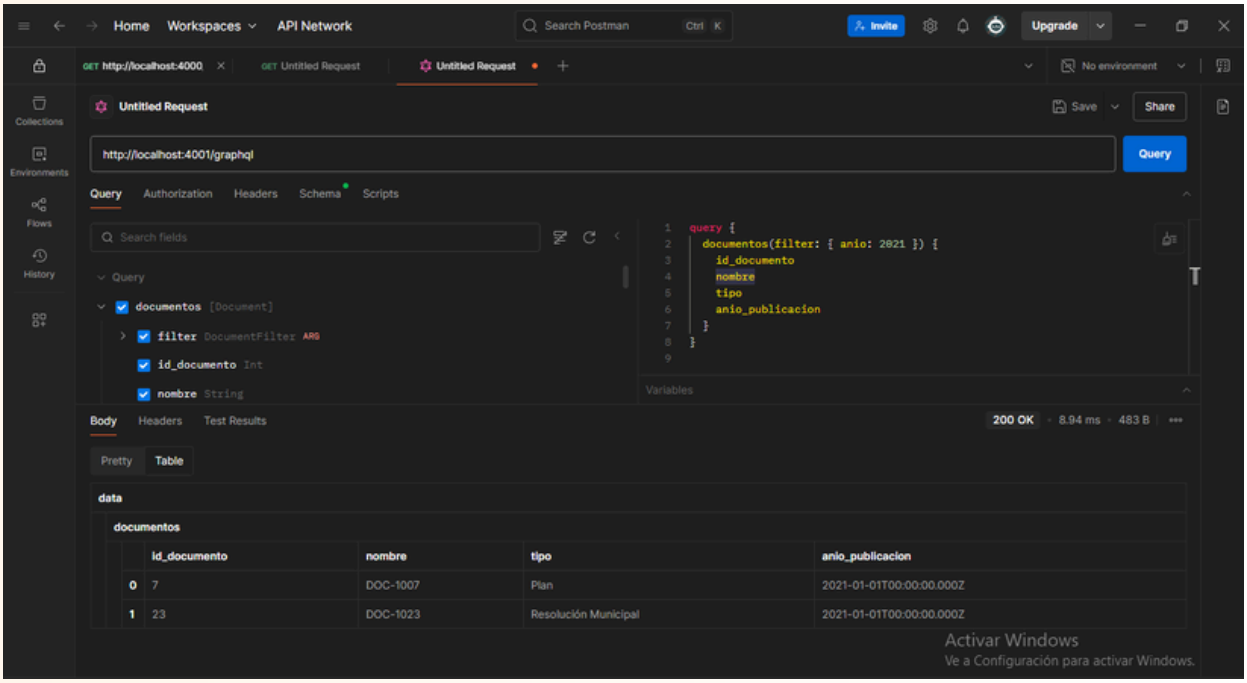


CASOS DE PRUEBA Y EVIDENCIAS FUNCIONALES

CASO 3: CONSULTA DE DOCUMENTOS CON FILTRO POR AÑO

Propósito: Verificar que la consulta documentos filtra correctamente por anio_publicacion (por ejemplo, 2021). Se utilizara GraphQL Playground y PostMan

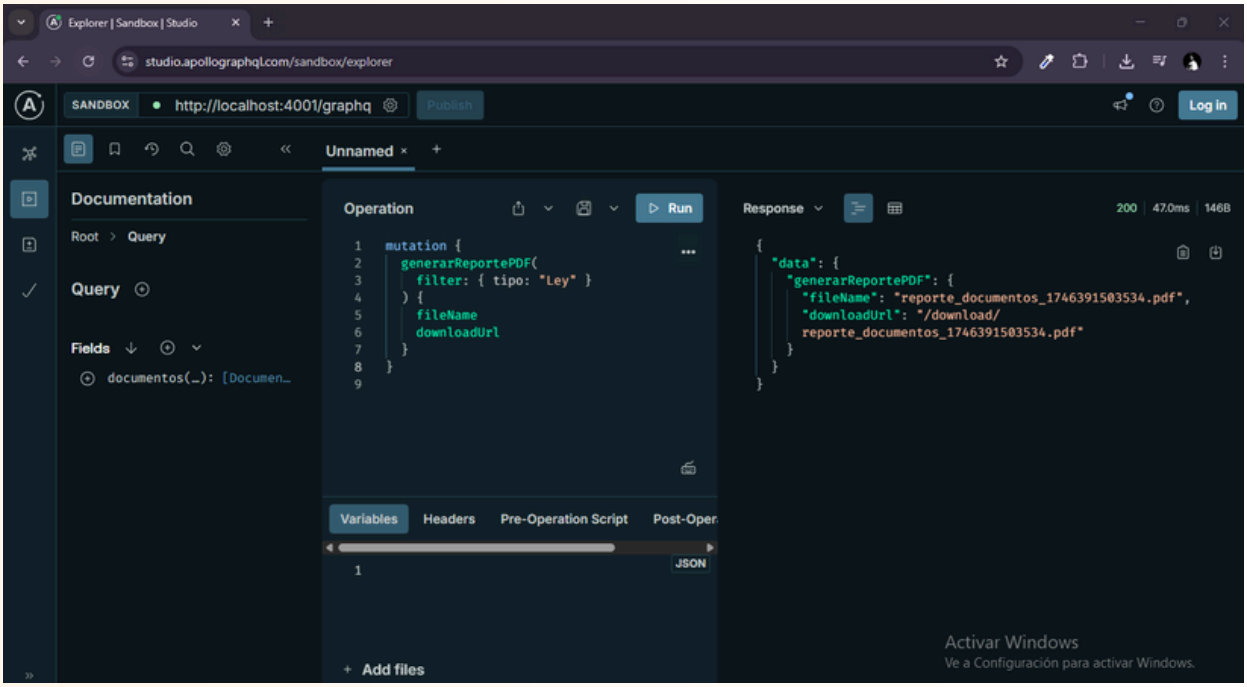


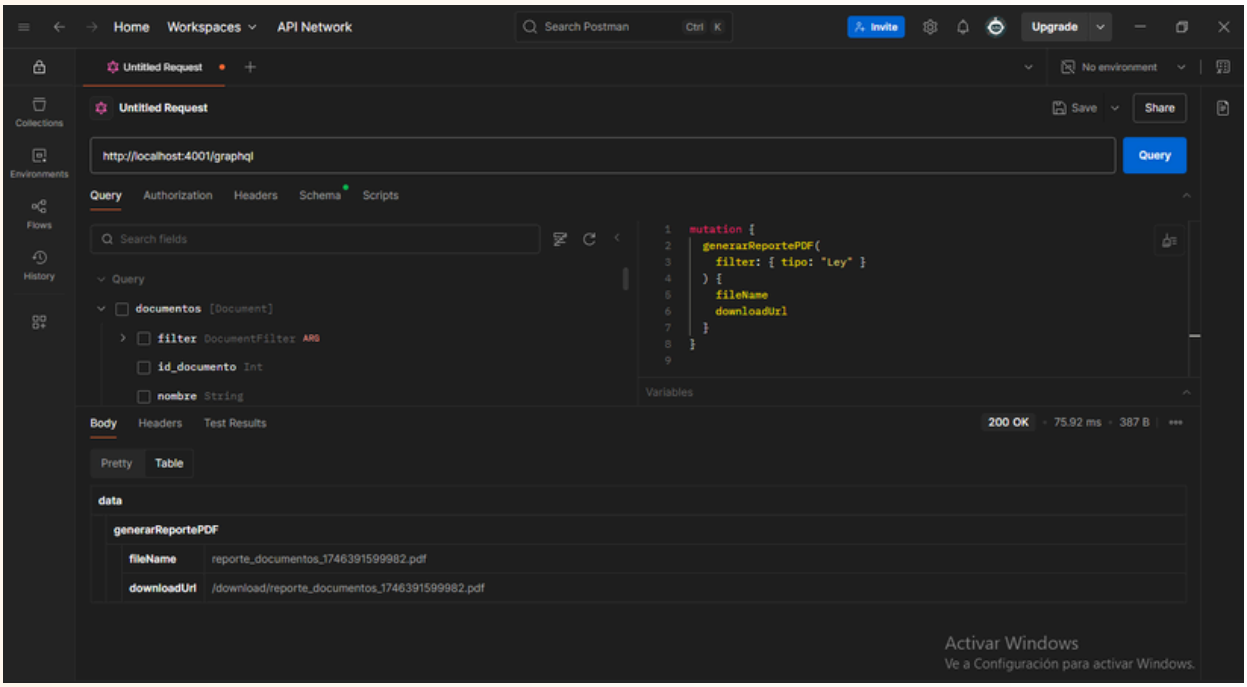


CASOS DE PRUEBA Y EVIDENCIAS FUNCIONALES

CASO 4: GENERACIÓN DE INFORME PDF

Propósito: Verificar que la mutación generarReportePDF crea un informe PDF con los documentos filtrados y los campos especificados. Se utilizara GraphQL Playground y PostMan





PRESENTACIÓN FINAL Y ANEXOS

5

RESUMEN

El microservicio MIGA, desarrollado con Node.js, Apollo Server, Axios, Express y PDFKit, cumple el objetivo de facilitar el acceso a documentos normativos sobre alimentación saludable en entornos escolares, permitiendo consultas y filtrado por tipo y año para poder generar informes PDF, y descargas, todo consumiendo un endpoint REST (http://localhost:4000/api/documentos). Se implementó una estructura modular con archivos como schema.js (tipos GraphQL), resolvers.js (lógica), documentService.js (consumo/filtrado), pdfGenerator.js (PDFs), y index.js (servidor), alojados en src/, junto con Uploads y .env. Se realizaron pruebas manuales (consultas sin filtro, por tipo/año, generación/descarga de PDF) con evidencias en capturas y un video adjunto para demostrar su funcionamiento.

ANEXOS

REPOSITORIO DE GIT HUB: https://github.com/Alkima-hydra/2do-Examen-TECWEB2-Microservicios
VIDEO EN DRIVE: https://drive.google.com/file/d/1UXtO_e7i9BSx24-ABUGqR6etNopjiqdw/view?usp=sharing