Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas - IPN

Aplicaciones Distribuidas

INTEGRANTES:

Flores Chavarría Diego

Gallegos Ruiz Diana Abigail

Informe de pruebas y resultados

Grupo: 4TV3

Fecha de entrega 09/01/2024

Índice

1. Introducción

Este proyecto es una implementación de una aplicación distribuida basada en microservicios. Este sistema integral incluye servicios web y un balanceador de carga y trabajando en conjunto para permitir la transferencia simultánea de archivos desde 1 MB hasta 100 MB. El caso de estudio se enfoca en un servicio de Streaming.

Dentro de este catálogo de servicios básicos, se consideran diferentes aspectos, desde la transferencia de archivos de múltiples tipos (texto, audio, video) ya sea carga o descarga, creación y visualización de usuarios, hasta la gestión de la concurrencia.

2. Herramientas y Tecnologías utilizadas

Lenguaje de Programación

Java: Lenguaje de programación versátil y orientado a objetos utilizado como base para el desarrollo del proyecto debido a su robustez, portabilidad y amplia comunidad de soporte.

Frameworks de Servicios Web

Spring Boot: Framework de Spring que simplifica la configuración y el desarrollo de aplicaciones basadas en Java. Se utilizó para crear rápidamente microservicios y gestionar servicios web de manera eficiente.

Framework de Mapeo Objeto-Relacional (ORM)

Hibernate: Framework de mapeo objeto-relacional para Java que facilita el trabajo con bases de datos relacionales. Se empleó para gestionar la capa de persistencia y mapear objetos Java a tablas de base de datos y viceversa.

Protocolos de Transferencia de Archivos

HTTP: Protocolo estándar para transferir datos en la World Wide Web. Fue el protocolo utilizado para la comunicación entre los distintos componentes del sistema distribuido, permitiendo la transferencia segura y confiable de información.

Herramientas de Testing

POSTMAN: Herramienta de colaboración para el desarrollo de API que se utilizó para realizar pruebas exhaustivas y validaciones de los servicios web desarrollados. Permitió verificar la correcta funcionalidad de los endpoints y garantizar la integridad de las operaciones realizadas a través de peticiones HTTP.

3. Patrón de Diseño

Controlador-Servicio-Repositorio

El patrón Controlador-Servicio-Repositorio es común en muchas aplicaciones de Spring Boot. Destaca por su clara separación de responsabilidades:

- El Controlador se encarga de exponer la funcionalidad para ser consumida externamente
- El Repositorio gestiona el almacenamiento y recuperación de datos
- La capa de Servicio aloja la lógica de negocio, coordinando la interacción entre el Controlador y el Repositorio.

Esta división permite organizar el código de manera clara y efectiva. Las operaciones de almacenamiento y recuperación de datos se manejan en el Repositorio, la exposición de funcionalidades en el Controlador y cualquier lógica específica de la aplicación en la capa de Servicio. El Repositorio opera independientemente de quién lo invoque, mientras que la capa de Servicio realiza su trabajo utilizando el Repositorio según sea necesario. Por su parte, el Controlador simplemente dirige las operaciones a la capa de Servicio, manteniéndose ágil y enfocado.

Este enfoque beneficia especialmente a las pruebas unitarias, ya que permite la simulación de capas adyacentes, facilitando las pruebas centradas en cada capa de manera aislada. Las pruebas del Controlador se enfocan en códigos de respuesta y valores, las pruebas del Servicio pueden realizarse como objetos regulares, y las pruebas de lógica de negocio pueden ejecutarse sin la necesidad de pasar por el Controlador.

4. Implementación del Balanceador de carga

El balanceador de cargar - gateway, es el punto de acceso a los diferentes microservicios, concentrando todas las peticiones hacia el, para posteriormente redireccionarlas a la instancia del microservicio correspondiente, este gateway puede manejar tolerancia a fallos mediante resiliencia, haciendo que una instancia similar a un microservicio responda con las peticiones en caso de que la principal se caiga o tarde en responder.

La configuración proporcionada emplea la anotación @Bean para crear un componente de tipo RouteLocator. Este componente, una interfaz de Spring Cloud Gateway, se emplea para establecer las reglas de enrutamiento. Para construir este componente, se utiliza el RouteLocatorBuilder, un elemento de Spring Cloud Gateway.

Se encuentra implementado en el siguiente código del proyecto:

```java
package com.appdist.ms.msgateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class MsGatewayApplication {

  public static void main(String[] args) {
    SpringApplication.run(MsGatewayApplication.class, args);
  }

  @Bean
  public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {
    return builder.routes()
    .route("users-service", r -> r.path("/v1/api/user/**")
    .uri("http://localhost:8081"))
    .route("files-service", r -> r.path("/v1/api/files/**")
    .uri("http://localhost:8082"))
    .route("files-reports-service", r -> r.path("/**")
    .uri("http://localhost:8080"))
    .build();
  }

```

SpringBootApplication

Anotación que indica que esta es la clase principal de la aplicación Spring Boot. Configura automáticamente la aplicación, escanea componentes y habilita la autoconfiguración.

public RouteLocator customRouteLocator(RouteLocatorBuilder builder)

Método anotado con *@Bean* que crea un componente `RouteLocator`. Este método define las reglas de enrutamiento para las diferentes rutas de la aplicación.

Se utiliza el RouteLocatorBuilder para definir las rutas y sus destinos (uri). En este caso, se definen tres rutas:

1. **users-service:** Cualquier solicitud que coincida con **/v1/api/user/**** se redirige al servicio en `http://localhost:8081`.
2. **files-service:** Las solicitudes que coinciden con **/v1/api/files/**** se redirigen al servicio en `http://localhost:8082`.
3. **files-reports-service:** Cualquier solicitud que no coincida con las rutas anteriores se redirige al servicio en `http://localhost:8080`.

5. Casos de Prueba

El sistema se compone de 5 servicios que se exponen a continuación:

5.1. Creación de un usuario

Este endpoint permite crear un nuevo usuario. Ver figura 1.

```
http://localhost:8765/v1/api/user/createUser
```

El contenido del body para esta prueba fue:

```
curl --location 'http://localhost:8765/v1/api/user/createUser' \
--data-raw '{
  "firstName":"Diego",
  "lastName":"Flores",
  "isActive":"1",
  "password":"040201",
  "email":"diegoflowers444@gmail.com",
  "suscription":"cargar|20"
}'
```

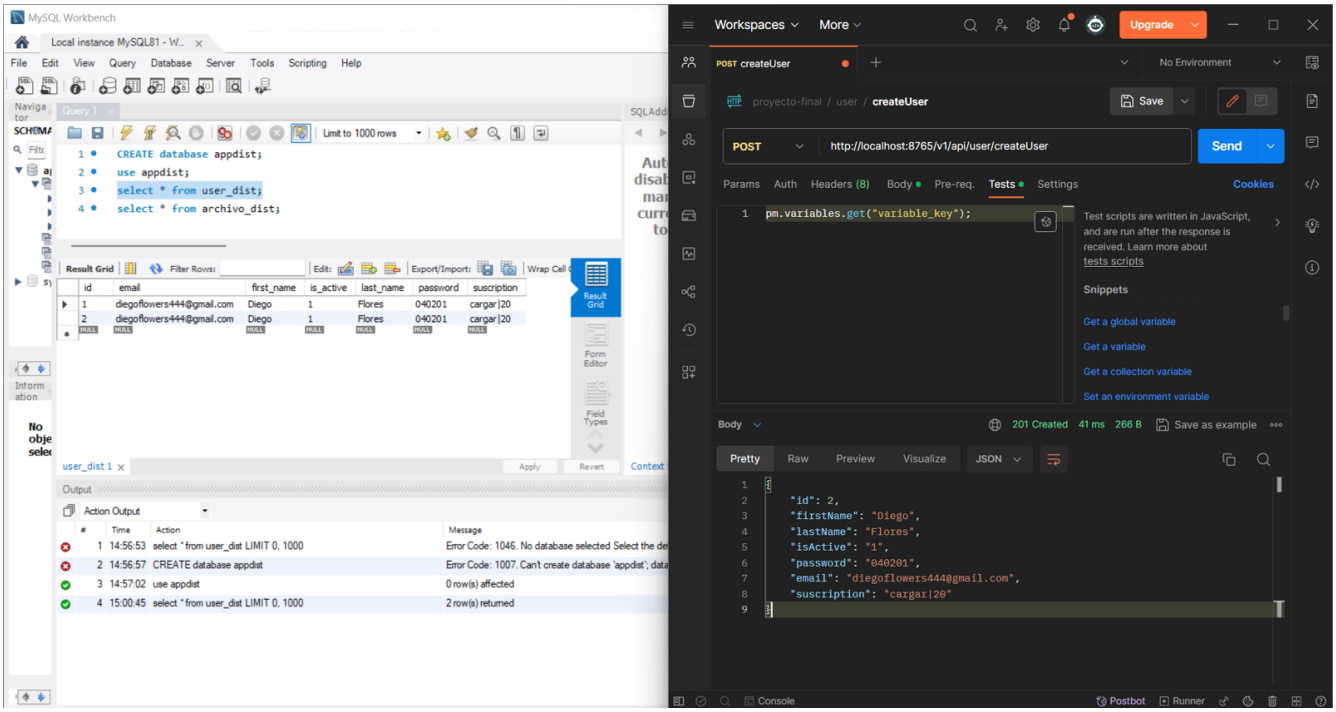


Figura 1: Visualización a la izquierda de la base de datos en MySQL del registro del nuevo usuario a través del consumo de la API en postman

5.2. Visualización de un usuario

Endpoint que realiza una solicitud HTTP GET para obtener información sobre un usuario específico a partir de un id de usuario. La solicitud debe incluir el ID del usuario en la ruta del URL. Ver figura 2.

```
http://localhost:8765/v1/api/user/findUser/{idUsuario}
```

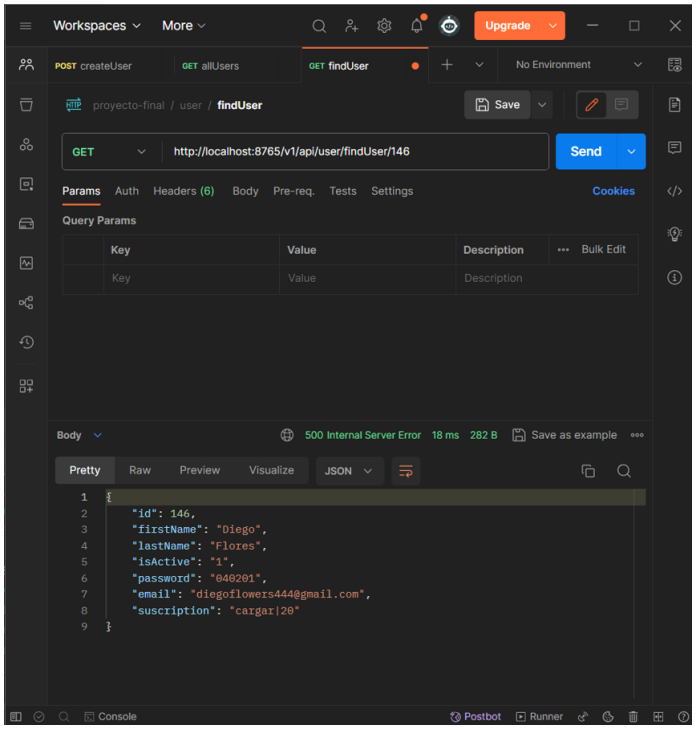


Figura 2: Respuesta del servicio de `findUser` en POSTMAN del usuario con id 146.

5.3. Creación de múltiples usuarios

Se ejecutó el servicio de usuarios en el runner de POSTMAN, haciendo 400 iteraciones del body proporcionado. Figura3.

5.4. Visualización de múltiples usuarios

Endpoint que realiza una solicitud HTTP GET para obtener una lista de usuarios. La solicitud no contiene un cuerpo de solicitud. Figura 4 .

```
http://localhost:8765/v1/api/user/listUsers
```

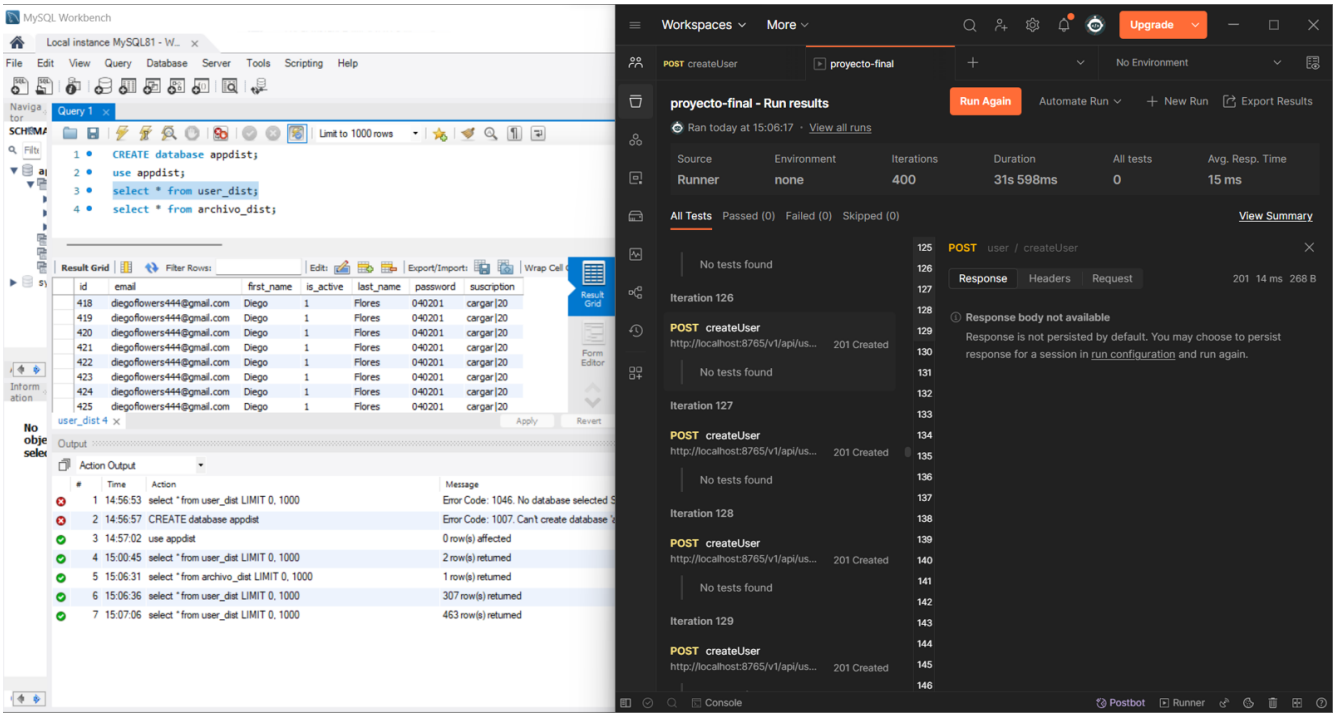


Figura 3: Visualización a la izquierda de la base de datos en MySQL del registro de 400 usuarios a través del consumo de la API en postman

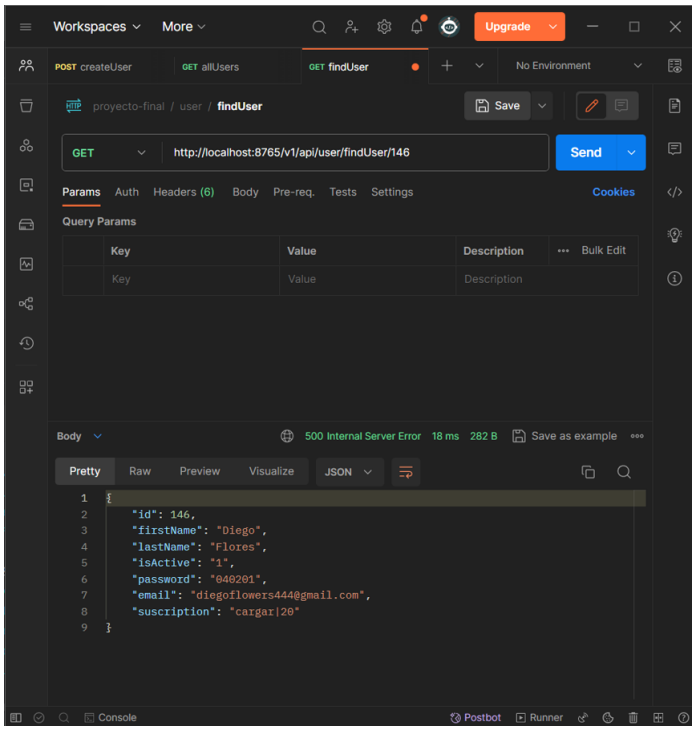
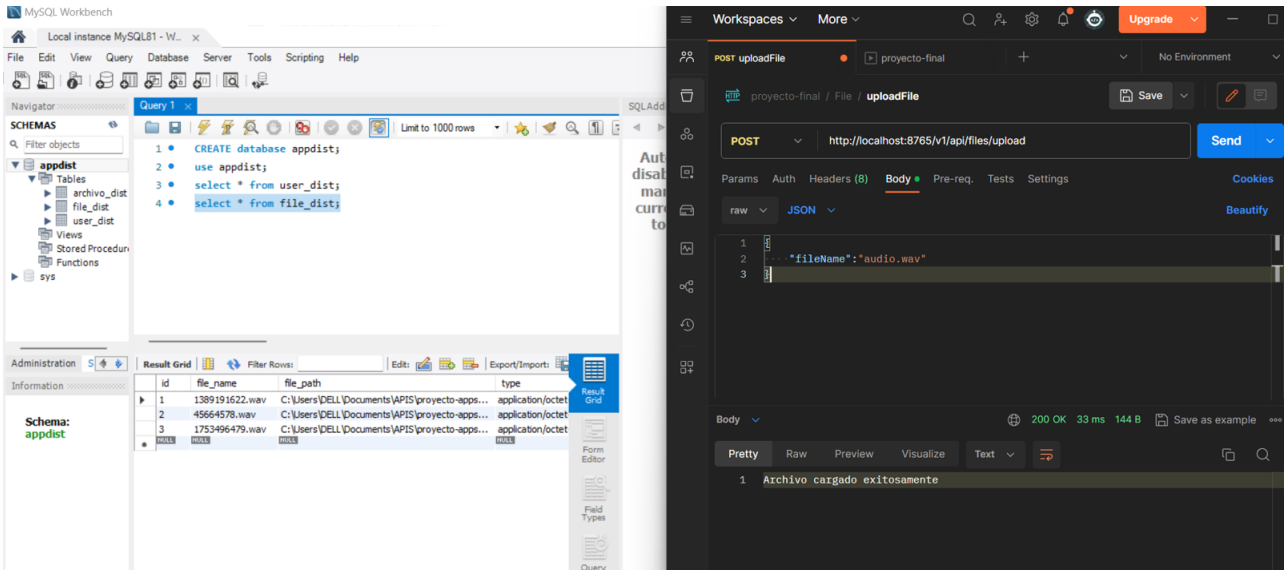


Figura 4: Respuesta del servicio de `listUsers` en POSTMAN

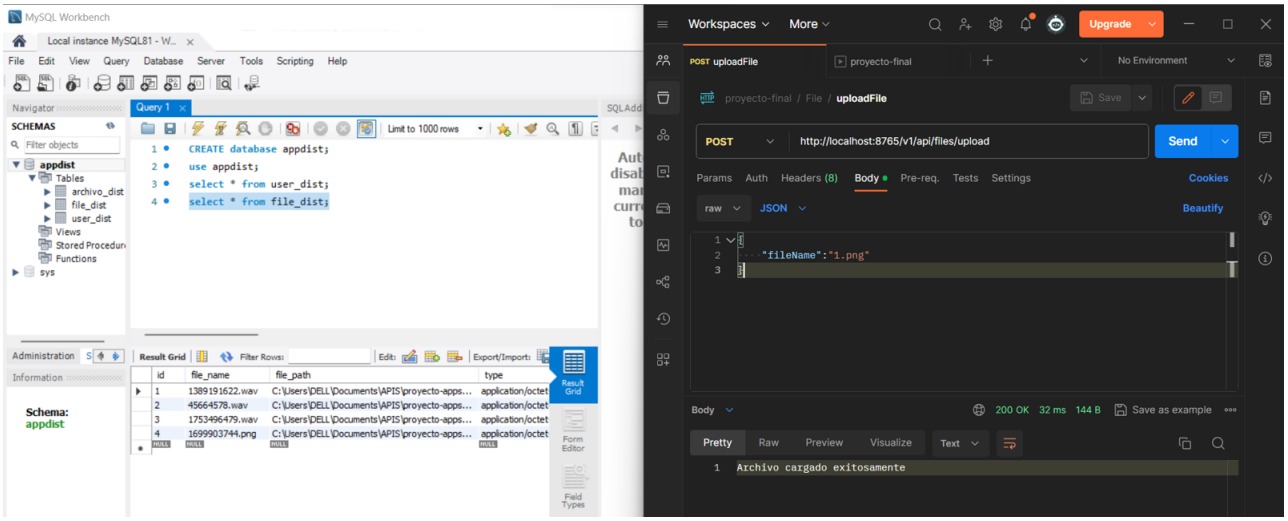
5.5. Cargar un archivo

Endpoint que realiza una solicitud HTTP POST para cargar un archivo. Ver figura 5.

```
http://localhost:8765/v1/api/files/upload
```



(a) Consumo del servicio para un archivo tipo wav, a la izquierda su registro en la base de datos de MySQL.



(b) Consumo del servicio para un archivo tipo png, a la izquierda su registro en la base de datos de MySQL.

Figura 5: Consumo del servicio `upload` en POSTMAN.

5.6. Visualización (Descarga) de un archivo

Endpoint que realiza una solicitud HTTP GET para descargar un archivo específico a partir de un id de archivo. La solicitud debe incluir el ID del archivo en la ruta del URL. Ver figura 6.

```
http://localhost:8765/v1/api/files/download/{fileID}
```

5.7. Cargar múltiples archivos

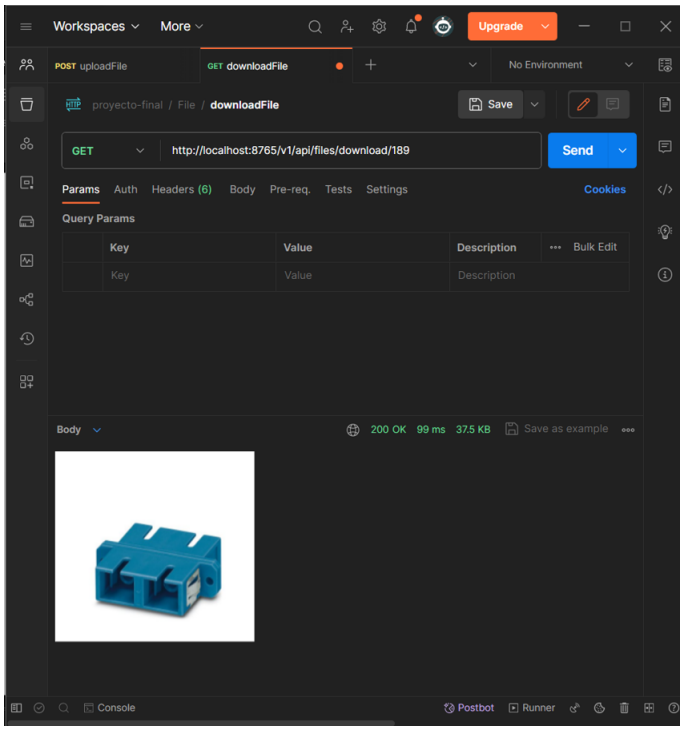
Se ejecutó el servicio de archivos en el runner de POSTMAN, haciendo 400 iteraciones del body proporcionado. En esta prueba, se observa un tiempo de ejecución de 34 s 880 ms, que es considerablemente mayor al de los usuarios por el tipo de archivo. Ver figura 7.

5.8. Visualización de reporte PDF

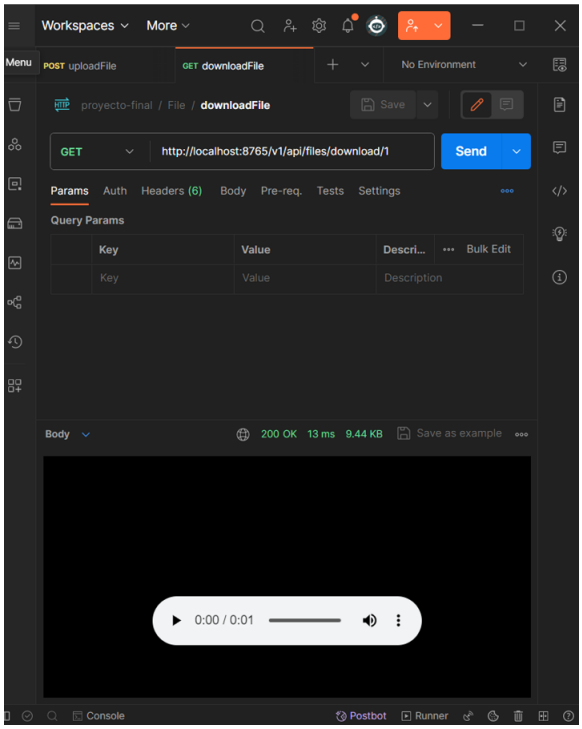
Endpoint que realiza una solicitud HTTP GET para descargar un archivo PDF con el reporte de la información de un usuario a partir de un id de usuario. La solicitud debe incluir el id del usuario en la ruta del URL. Ver figura 8.

6. Documentación de APIS

La documentación detallada de todo el proyecto se encuentra publicada en https://documenter.getpostman.com/view/17468240/2s9YsKfXDx?fbclid=IwAR06VjQ71Er3cfkka5-0bIyap73vlpfQ1aQXM3ppF2822c236-aa7e-4d9a-bc98-fb6b1881b290



(a) Respuesta del servicio para un archivo tipo wav



(b) Respuesta del servicio para un archivo tipo png

Figura 6: Respuesta del servicio `download` en POSTMAN.

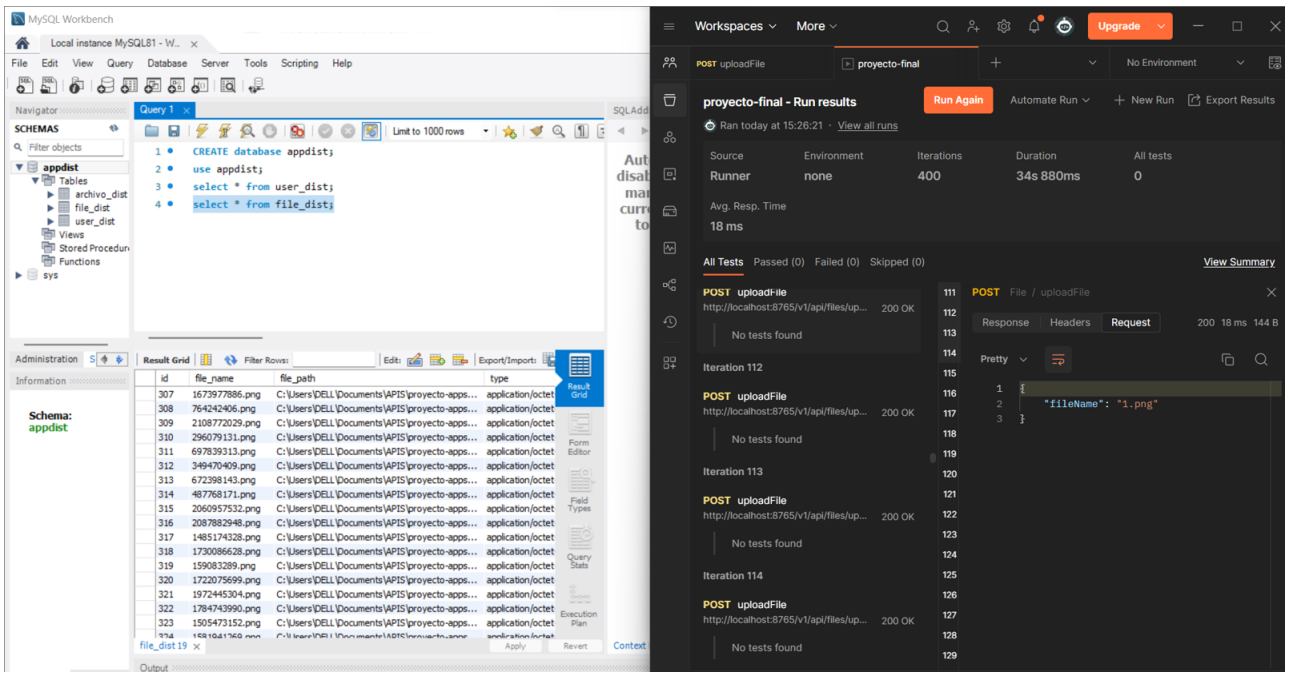
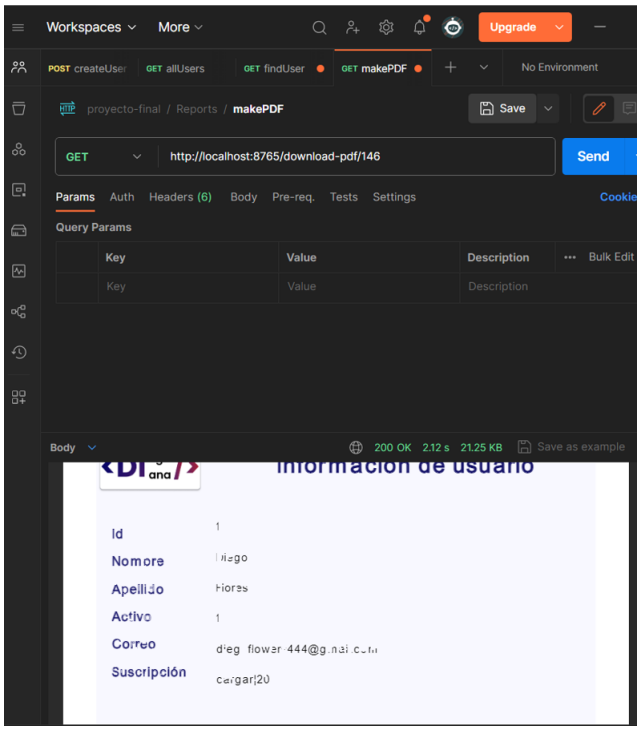
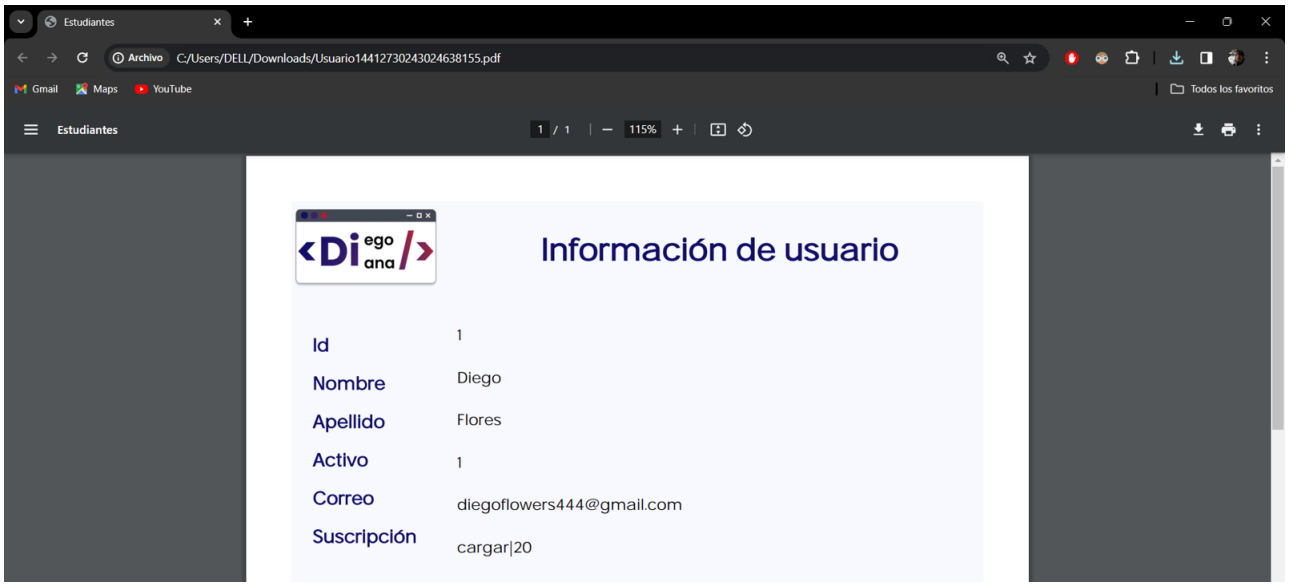


Figura 7: Consumo del servicio de `upload` en POSTMAN, a la izquierda la visualización de los 400 registros en la base de datos de MySQL.



(a) Respuesta del servicio en POSTMAN



(b) Visualización del PDF generado en el navegador de Chrome

Figura 8: Respuesta del servicio `download-pdf` en POSTMAN.