Integrantes: Leonardo Espinoza, Santiago Jiménez, Isabel Morocho

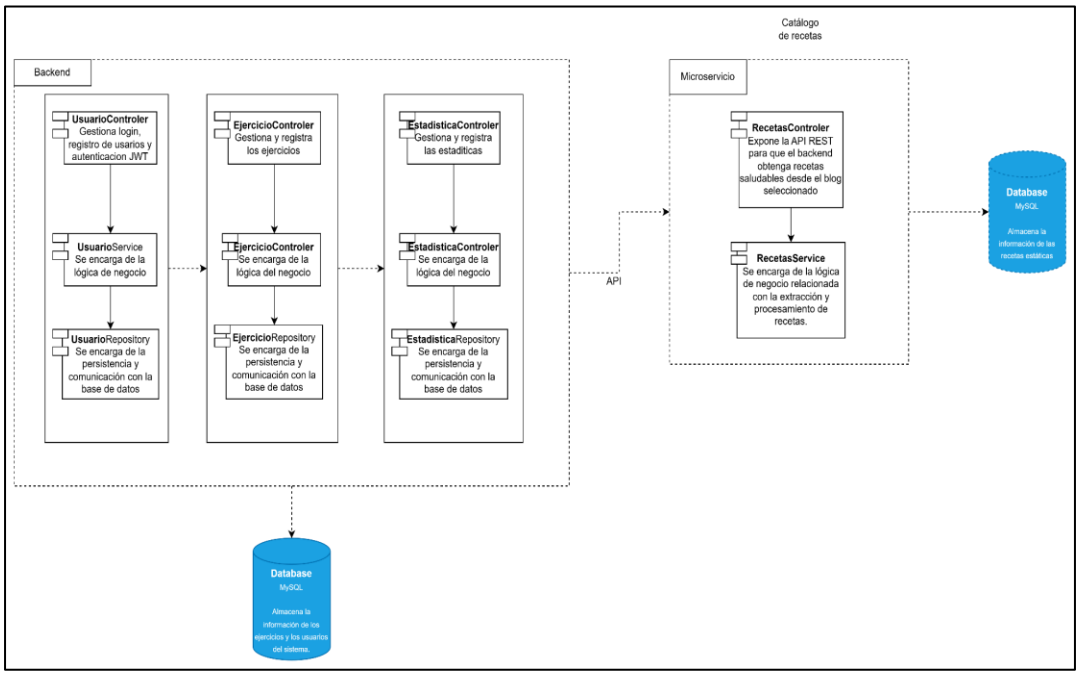
INFORME

Documentar el desarrollo de microservicios y su integración mediante herramientas colaborativas y APIs.

1. **Resumen del microservicio creado**
   - **Nombre del microservicio:** Microservicio de Recetas: Catálogo de recetas (contenido estático/semi-estático, recomendaciones)
   - **Propósito:** El microservicio implementado tiene como objetivo brindar a los diferentes usuarios del sistema planes nutricionales saludables previamente establecidos según la meta de cada usuario, donde cada uno de estos les ayudaran a complementar su rutina de manera óptima. Estas recetas permanecerán almacenadas estáticamente en formato JSON y pueden ser accedidas mediante las rutas o endpoints.
     El objetivo de la implementación de este microservicio es mejorar la calidad de experiencia de los usuarios dentro de nuestro sistema principal, permitiéndoles optimizar sus rutinas de ejercicio con una alimentación adecuada.
   - **Relación con el sistema principal:** Este microservicio cuenta como parte del ecosistema modular que sigue el sistema principal planificador de rutinas, como un servicio independiente que se comunica con el backend del sistema principal mediante API Rest. El microservicio catálogo de recetas nutricionales viene siendo un complemento alimenticio para el módulo de ejercicios, cada usuario puede consultar algunos planes establecidos y elegir el de su preferencia promoviendo su alimentación acorde al tipo de entrenamiento que se encuentre realizando.
2. **Integración mediante APIs**
   - **Descripción del método de comunicación (REST, API Gateway o endpoints compartidos).**

DIAGRAMA C4 ACTUALIZADO

Componentes



Para la comunicación con el Microservicio (Spring Boot), con el sistema principal FitSIL (Spring Boot) se implementó un nuevo módulo: Gestión de Nutrición, en el cual encontramos controller, dto, y service (Recetas desde el microservicio)

A continuación **RecetaController.java** para la conexión con el microservicio:

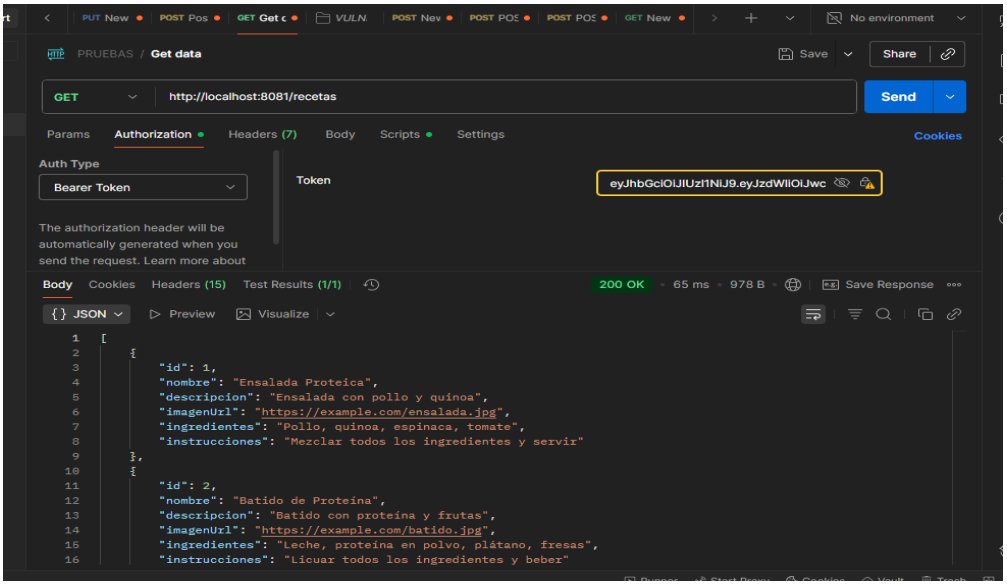
```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import sistema_FitSIL.GestionNutricion.dto.RecetaDTO;
import sistema_FitSIL.GestionNutricion.service.RecetaClientService;

import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/recetas")
@CrossOrigin(origins = "*")
public class RecetaController {

    @Autowired
    private RecetaClientService recetaClientService;

    /**
     *  Obtener todas las recetas
     * GET http://localhost:8081/recetas
     */
    @GetMapping
    public ResponseEntity<?> obtenerRecetas() {
        try {
            List<RecetaDTO> recetas = recetaClientService.obtenerTodasLasRecetas();
            return ResponseEntity.ok(recetas);
        } catch (RuntimeException e) {
            return ResponseEntity.status(503)
                    .body(Map.of(
                        k1:"error", v1:"Microservicio no disponible",
                        k2:"mensaje", e.getMessage()
                    ));
        }
    }

    /**
     *  Obtener una receta específica por ID
     * GET http://localhost:8081/recetas/1
     */
```

Agregación del localhost del microservicio al sistema general:

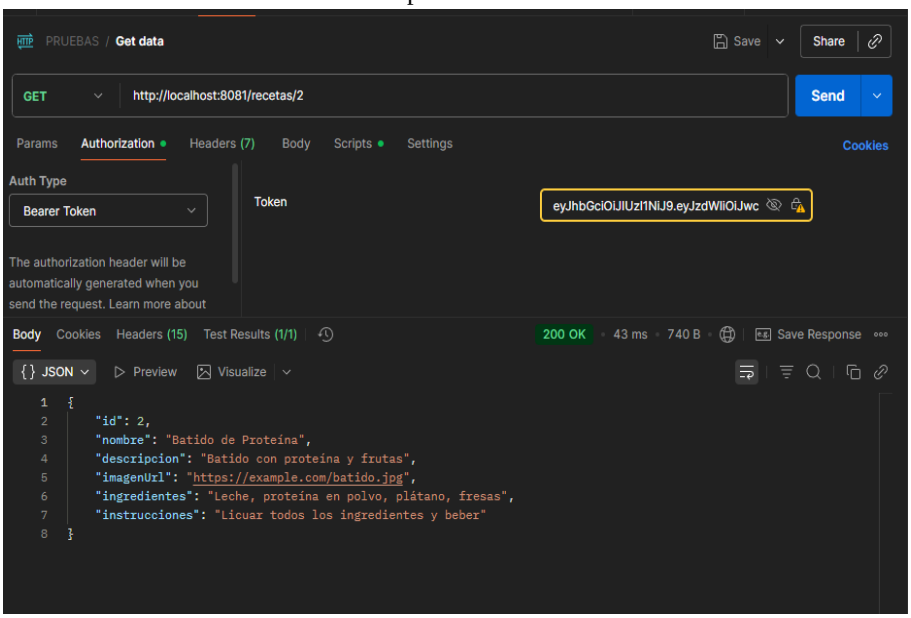
sistema-FitSIL > src > main > resources > application.properties

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Que Hibernate cree/actualice automáticamente las tablas según tus modelos
spring.jpa.hibernate.ddl-auto=update

# Habilitar consola H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console


spring.config.import=optional:file:.env
jwt.secret=MI_SECRETO_AQUI


# URL del microservicio de recetas
microservicio.recetas.url=http://localhost:8082
```

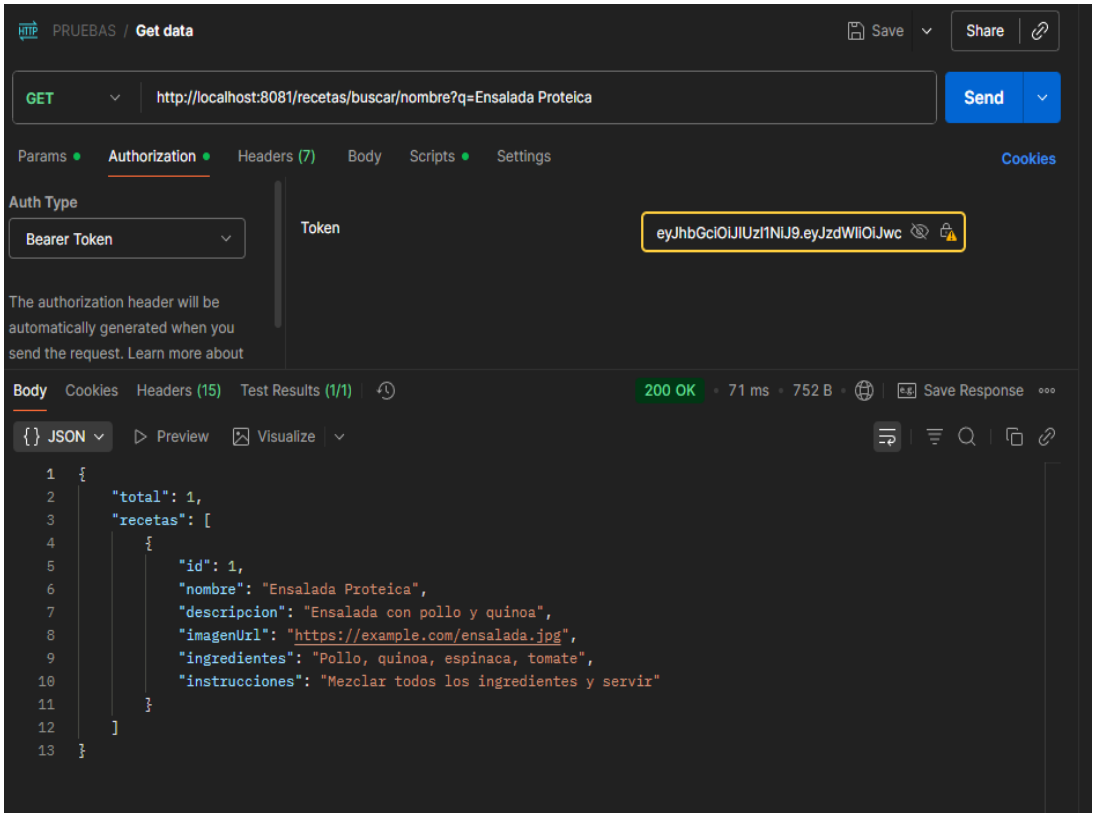
- **Ejemplos de llamadas y respuestas (fragmentos Postman o Swagger).**
  Para las peticiones en Postman, se debe estar logueado correctamente, usando el token de autenticación, para acceder a las recetas:
  - Obtener todas las recetas



- Obtener una receta específica

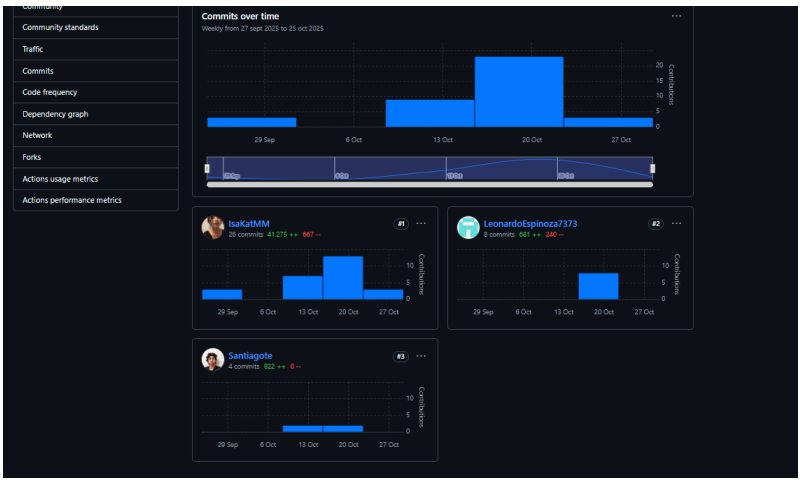


- Buscar por nombre

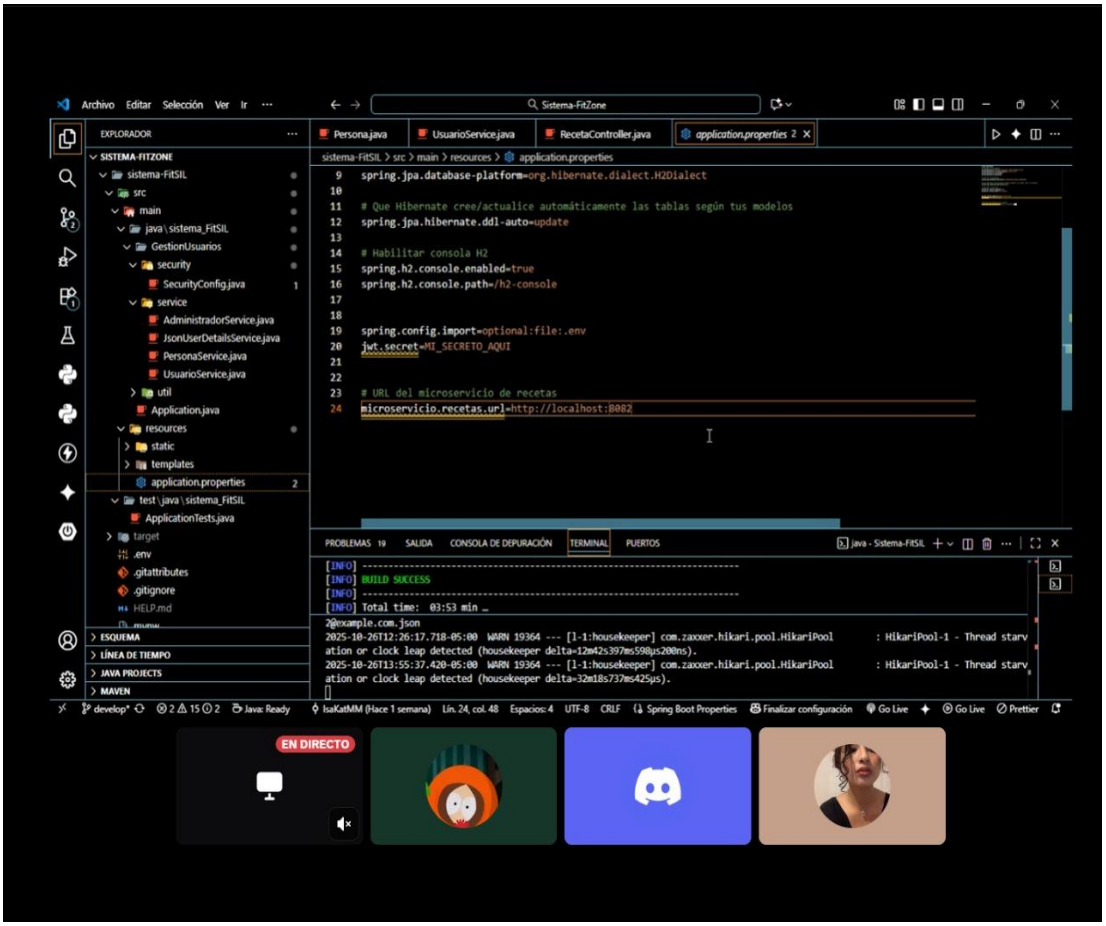


3. Herramientas colaborativas utilizadas

- **Describir el uso de Trello, Taiga, GitHub Projects, Discord u otras herramientas para la coordinación del equipo.**
  Para el desarrollo de esta actividad se utilizaron las herramientas GitHub y Discord. **GitHub** se empleó principalmente para el control de versiones del código fuente, y la implementación del nuevo microservicio, y **Discord** para la comunicación facilitando las reuniones de nuestro equipo y la discusión de avances.
- **Evidenciar cómo estas herramientas contribuyeron a la planificación y seguimiento de tareas.**
  - GitHub:



  - Discord:



4. Buenas prácticas y aprendizajes

- I**dentificar dos buenas prácticas aplicadas durante el desarrollo.**
    1. **Uso de arquitectura modular (Microservicios):** Se aplicó la buena práctica de desacoplar la funcionalidad de recetas en un microservicio independiente, lo cual permite una mayor escalabilidad, mantenibilidad y reutilización del código. El nuevo microservicio desarrollado se comunica con el sistema principal FitSIL mediante consumo de APIs REST.
    2. **Uso de control de versiones y colaboración con GitHub:** Se utilizó GitHub para gestionar el código fuente del proyecto, aplicando commits frecuentes y descriptivos, uso de ramas para el desarrollo de nuevas funcionalidades y pull requests para integrar los cambios de forma controlada.
- **Reflexión personal sobre cómo la modularización mejora la escalabilidad y mantenimiento del proyecto.**

  La modularización aplicada en el desarrollo del microservicio de recetas para el sistema FitSIL nos permitió comprender la importancia de dividir un sistema grande en componentes más pequeños y específicos (en este caso únicamente el de recetas). Al separar la gestión de recetas en un microservicio independiente, el proyecto se volvió más organizado, escalable y fácil de mantener, por ello gracias a esta estructura modular, ahora podemos actualizar la lógica del microservicio sin afectar directamente al resto del sistema, lo que reduce riesgos y facilita el trabajo en equipo. Además, si en el futuro deseamos ampliar dicha funcionalidad, por ejemplo, permitir que otros módulos o aplicaciones consuman las recetas, la integración será más sencilla gracias a la independencia y comunicación mediante APIs REST.

Preguntas de reflexión

1. **¿Qué ventajas observas en la integración mediante APIs REST respecto a un monolito tradicional?**

   Las ventajas que influyen sobre la integración de API Rest sobre un modelo monolito tradicional son escalabilidad, flexibilidad, separación de componentes y mantenimiento ágil, un monolito es una aplicación centralizada donde todos los componentes están estrechamente acoplados por lo tanto es mucho más difícil llegar a desarrollar nuevas funcionalidades para el sistema además de que estamos obligados a cambiar gran parte del código del sistema ya existente lo que dejaría a todo el sistema sin funcionar, esto no pasaria gracias al uso de microservicios y la conexión mediante las APIs.

2. **¿Cómo aportan las herramientas colaborativas a la gestión técnica de los microservicios?**

   El uso de herramientas colaborativas es una gran técnica para la gestión de los microservicios pues ayudan a facilitar la comunicación y la automatización que son elementos importantes para el correcto manejo de arquitecturas distribuidas. Además aceleran el desarrollo en equipos de trabajo por lo que cada grupo podrá encargarse de un servicio a la vez, esto genera un sistema totalmente independiente y robusto.

3. **¿Qué riesgos existen al distribuir un sistema en varios microservicios y cómo pueden mitigarse?**

   Algunos riesgos se incluyen como el aumento de la complejidad la gestión y manejo de los datos y la comunicación, mientras que existen algunas opciones con la que podemos mitigar estos riesgos son: la automatización mediante herramientas de orquestación como los kubernetes, gestión de la consistencia de los datos mediante patrones distribuidos, además podemos reforzar la seguridad del diseño del sistema con autenticacion y autorizacion robusta, manejo de API Gateway y el cifrado de información sensible.

Bibliografía

[1] Newman, S. (2021). Building Microservices. O'Reilly Media.

[2] Richardson, C. (2022). Microservices Patterns. Manning Publications.

[3] OWASP Foundation (2023). OWASP Secure Microservices Design.

[4] Docker Inc. Docker Documentation.

[5] GitHub Docs. Collaborative Development Workflows.