



unl

Universidad
Nacional
de Loja

FEIRNNR - Carrera de computación

UNIVERSIDAD NACIONAL DE LOJA

**Facultad De Industria, Energía y Recursos
Naturales No Renovables**

Carrera: Computación

Tema:

APE-004

Asignatura:

Desarrollo Basado en Plataformas

Docente:

Ing. Edison Coronel

Integrantes:

- Santiago Jimenez
- Leonardo Espinoza
- Isabel Morocho

Curso:

5to

LOJA – ECUADOR

2025

Guía de Actividades

Práctico-Experimentales Nro. 004

1. Datos Generales

Asignatura	Desarrollo Basado en Plataformas
Ciclo	5 A
Unidad	1
Resultado de aprendizaje de la unidad	Discute cómo los estándares Web influyen en el desarrollo de software, bajo los principios de solidaridad, transparencia, responsabilidad y honestidad.
Práctica Nro.	004
Título de la Práctica	Despliegue de un microservicio independiente e integración básica con el servicio principal del proyecto.
Nombre del Docente	Edison Leonardo Coronel Romero
Fecha	Viernes 24 de octubre
Horario	07h30 – 10h30
Lugar	Aula 232
Tiempo planificado en el Sílabo	3 horas

2. Objetivo(s) de la Práctica:

- Diseñar y desplegar un microservicio funcional que se comuniquen con el backend principal del proyecto.
- Implementar comunicación REST entre servicios utilizando un API Gateway o endpoint compartido.
- Documentar la arquitectura de microservicios en el modelo C4 y registrar evidencias del despliegue.

3. Materiales y reactivos:

- Computador con acceso a Internet.
- Docker / Docker Compose o entorno virtual local.
- Repositorio del proyecto (GitHub/GitLab).
- Postman / Swagger.
- Framework backend (Django REST Framework / Express / Spring Boot).

4. Equipos y herramientas

- Laboratorio de Desarrollo de Software o equipo personal.
- IDE: Visual Studio Code / IntelliJ IDEA.
- Git y GitKraken (flujo GitFlow).
- Terminal de comandos y contenedores Docker.

- Herramientas de modelado C4 (PlantUML, Mermaid, Draw.io).

5. Procedimiento / Metodología

Inicio

- Contextualización de la práctica: introducción a microservicios, serverless y patrones de comunicación.
- Revisión del modelo C4 actual (backend monolítico) y planificación de su modularización.

Desarrollo

1. Diseño del microservicio

- **Seleccionar una funcionalidad del proyecto que pueda desacoplarse (por ejemplo: módulo de notificaciones, gestión de equipos o usuarios).**

Se ha seleccionado la funcionalidad: **Microservicio de Recetas - Catálogo de recetas**, el cual 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.

- **Definir su API interna y endpoints principales.**

Local host - Microservicio: <http://localhost:8082/>

Método	Ruta	Descripción
Get	http://localhost:8082/api/recetas	Obtener todas las recetas disponibles en el catálogo

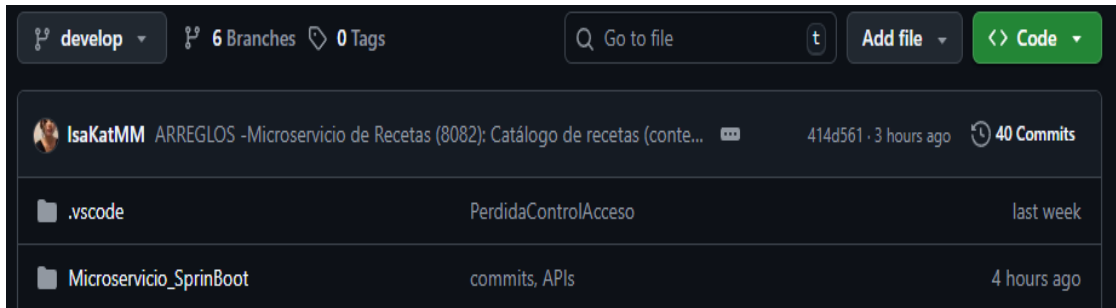
Local host -Sistema FitSIL: <http://localhost:8081/>

Método	Ruta	Descripción
GET	http://localhost:8081/recetas	Obtener todas las recetas disponibles en el catálogo del microservicio
GET	http://localhost:8081/recetas/#	Obtener una receta específica por id
GET	http://localhost:8081/recetas/buscar/nombre?q=EnsaladaProteica	Buscar por nombre

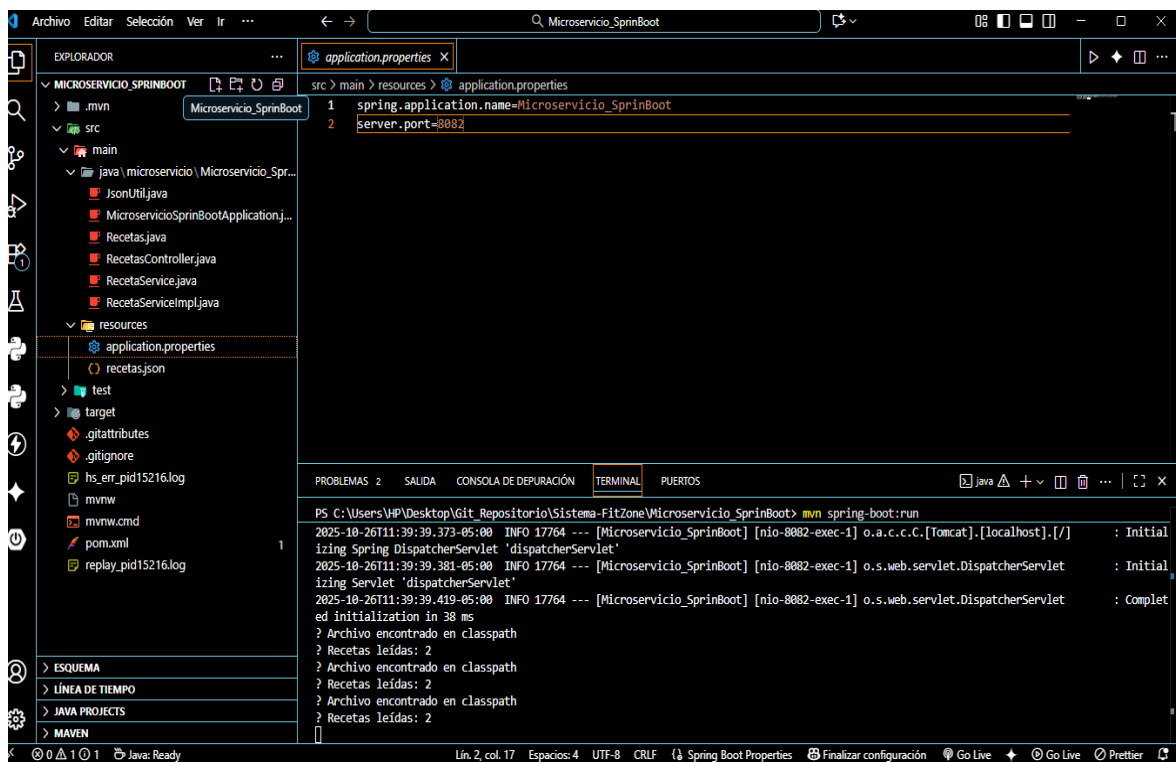
2. Configuración del entorno

- **Crear una nueva carpeta o repositorio** `microservice_<nombre>` y configurarlo como módulo independiente.

Carpeta en el repositorio del microservicio:



Configuración del módulo independiente:



- **Implementar la estructura base con controladores, servicios y rutas.**

Controlador:

The screenshot shows an IDE with the file explorer on the left displaying the project structure for 'MICROSERVICIO_SPRINGBOOT'. The main editor displays the code for 'RecetasController.java'. The code imports Spring Web annotations and defines a REST controller with a single endpoint for obtaining recipes. The terminal at the bottom shows the command 'mvn spring-boot:run' and the application's startup logs.

```

4
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.RestController;
8
9 @RestController
10 @RequestMapping("/api/recetas")
11 public class RecetasController {
12
13     private final RecetaService recetaService;
14
15     public RecetasController(RecetaService recetaService) {
16         this.recetaService = recetaService;
17     }
18
19     @GetMapping
20     public List<Recetas> obtenerRecetas() {
21         return recetaService.obtenerRecetas();
22     }

```

Terminal output:

```

PS C:\Users\VP\Desktop\Git_Repository\Sistema-FitZone\Microservicio_SpringBoot> mvn spring-boot:run
2025-10-26T11:39:39.373-05:00 INFO 17764 --- [Microservicio_SpringBoot] [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initial
izing Spring DispatcherServlet 'dispatcherServlet'
2025-10-26T11:39:39.381-05:00 INFO 17764 --- [Microservicio_SpringBoot] [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Initial

```

Servicio:

The screenshot shows the same IDE with the file 'RecetaService.java' selected. The code defines a package, imports a List, and creates an interface 'RecetaService' with a method 'obtenerRecetas()'. The terminal shows the application running successfully after the 'mvn spring-boot:run' command.

```

1 package microservicio.Microservicio_SpringBoot;
2
3 import java.util.List;
4
5 public interface RecetaService {
6     List<Recetas> obtenerRecetas();
7 }
8
9
10

```

Terminal output:

```

PS C:\Users\VP\Desktop\Git_Repository\Sistema-FitZone\Microservicio_SpringBoot> mvn spring-boot:run
2025-10-26T11:39:39.373-05:00 INFO 17764 --- [Microservicio_SpringBoot] [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initial
izing Spring DispatcherServlet 'dispatcherServlet'
2025-10-26T11:39:39.381-05:00 INFO 17764 --- [Microservicio_SpringBoot] [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Initial

```

- **Configurar variables de entorno, dependencias y documentación básica (README).**
- **Variables de entorno:** Las variables de entorno son usadas para no exponer contraseñas o datos sensibles

```

application.properties X
src > main > resources > application.properties
1 spring.application.name=${APP_NAME:Microservicio_SprinBoot}
2 server.port=${PORT:8082}
3

```

- **Dependencias:**

The screenshot shows an IDE with the following components:

- EXPLORADOR (Left):** A file explorer showing the project structure. The 'resources' folder is expanded, showing 'application.properties' and 'recetas.json'.
- Editor (Center):** The 'pom.xml' file is open, showing XML content. The content includes a parent POM for 'spring-boot-starter-parent' and a child POM for 'Microservicio_SprinBoot'.
- TERMINAL (Bottom):** A terminal window showing the output of the command 'mvn spring-boot:run'. The output indicates that the application started successfully on port 8082.

pom.xml Content:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.4.10</version> Upgrade to the Latest Patch
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.microservicio</groupId>
12  <artifactId>Microservicio_SprinBoot</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>Microservicio_SprinBoot</name>
15  <description>Microservicio para el sistema fitsil</description>
16  <url/>
17  <licenses>
18    <license/>
19  </licenses>
20  <dependencies>

```

Terminal Output:

```

PS C:\Users\HP\Desktop\Git_Repository\Sistema-FitZone\Microservicio_SprinBoot> mvn spring-boot:run
2025-10-26T15:03:48.524-05:00 INFO 21736 --- [Microservicio_SprinBoot] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat
started on port 8082 (http) with context path '/'
2025-10-26T15:03:48.559-05:00 INFO 21736 --- [Microservicio_SprinBoot] [ restartedMain] m.M.MicroservicioSprinBootApplication : Started
MicroservicioSprinBootApplication in 5.856 seconds (process running for 7.218)
2025-10-26T15:03:48.665-05:00 INFO 21736 --- [Microservicio_SprinBoot] [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initial
izing Spring DispatcherServlet 'dispatcherServlet'
2025-10-26T15:03:48.680-05:00 INFO 21736 --- [Microservicio_SprinBoot] [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Initial
izing Servlet 'dispatcherServlet'

```

3. Comunicación entre servicios

- **Configurar API Gateway o endpoint del backend principal para enlazar el nuevo servicio.**

API Gateway: <http://localhost:8082/api/recetas> → Microservicio de recetas

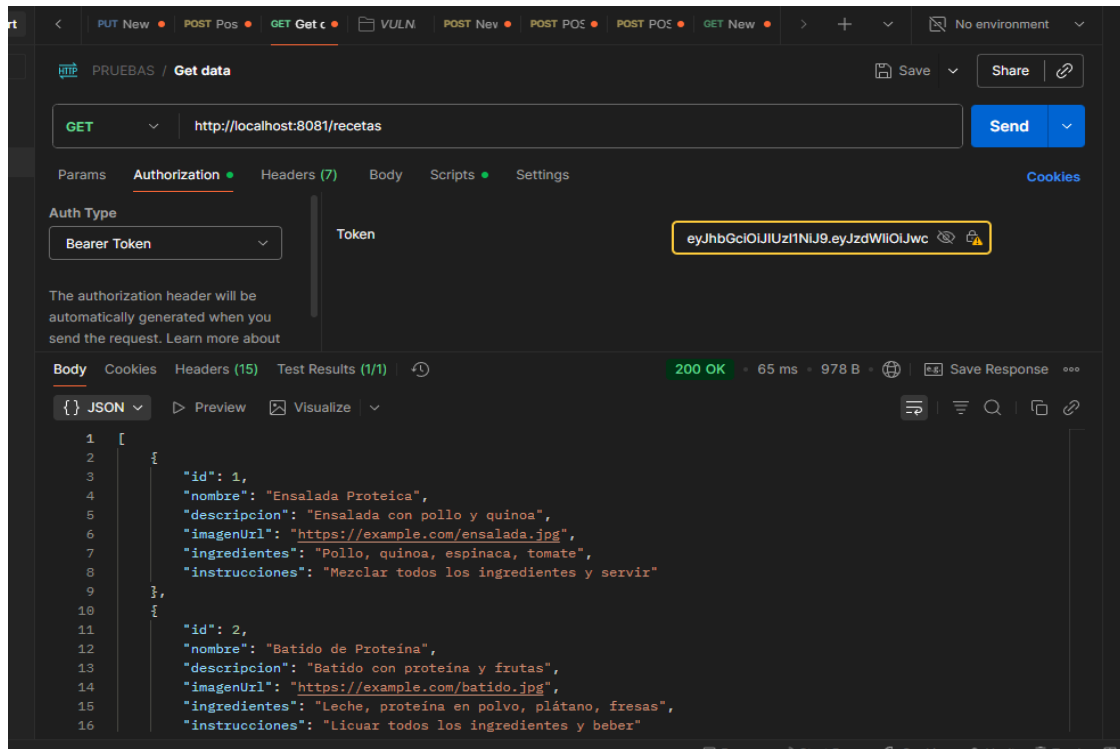
The screenshot shows an IDE with the following components:

- Explorador (Ctrl+Mayús+E):** Displays the project structure for 'SISTEMA-FITZONE'. The 'RecetaController.java' file is selected under the 'controller' directory.
- RecetaController.java:** The code for the controller is visible. It includes a `@CrossOrigin(origins = "**")` annotation, an `@Autowired` `RecetaClientService`, and a `@GetMapping` method `obtenerRecetas()` that returns a `ResponseEntity` containing a list of recipes or an error message.
- TERMINAL:** Shows the build output and runtime logs. The build was successful. The runtime logs show warnings from HikariPool-1 regarding clock leap detection.

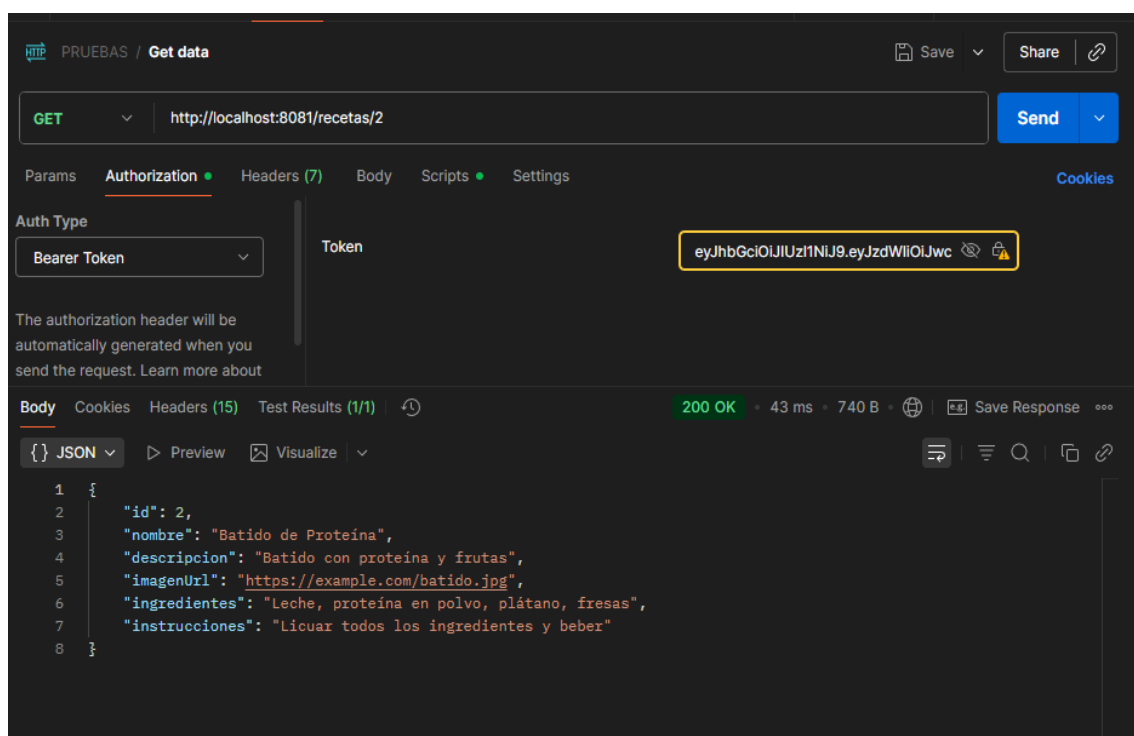
The screenshot shows the same IDE environment as the previous one, but with different code in the `RecetaController.java` file and the terminal output.

- RecetaController.java:** The code now includes two `@GetMapping` methods. The first, `buscarPorNombre()`, takes a query parameter `q` and returns a `ResponseEntity` with a map containing the total number of recipes and the list of recipes. The second, `buscarPorIngrediente()`, is partially visible.
- TERMINAL:** Shows the build output and runtime logs. The build was successful. The runtime logs show warnings from HikariPool-1 regarding clock leap detection.

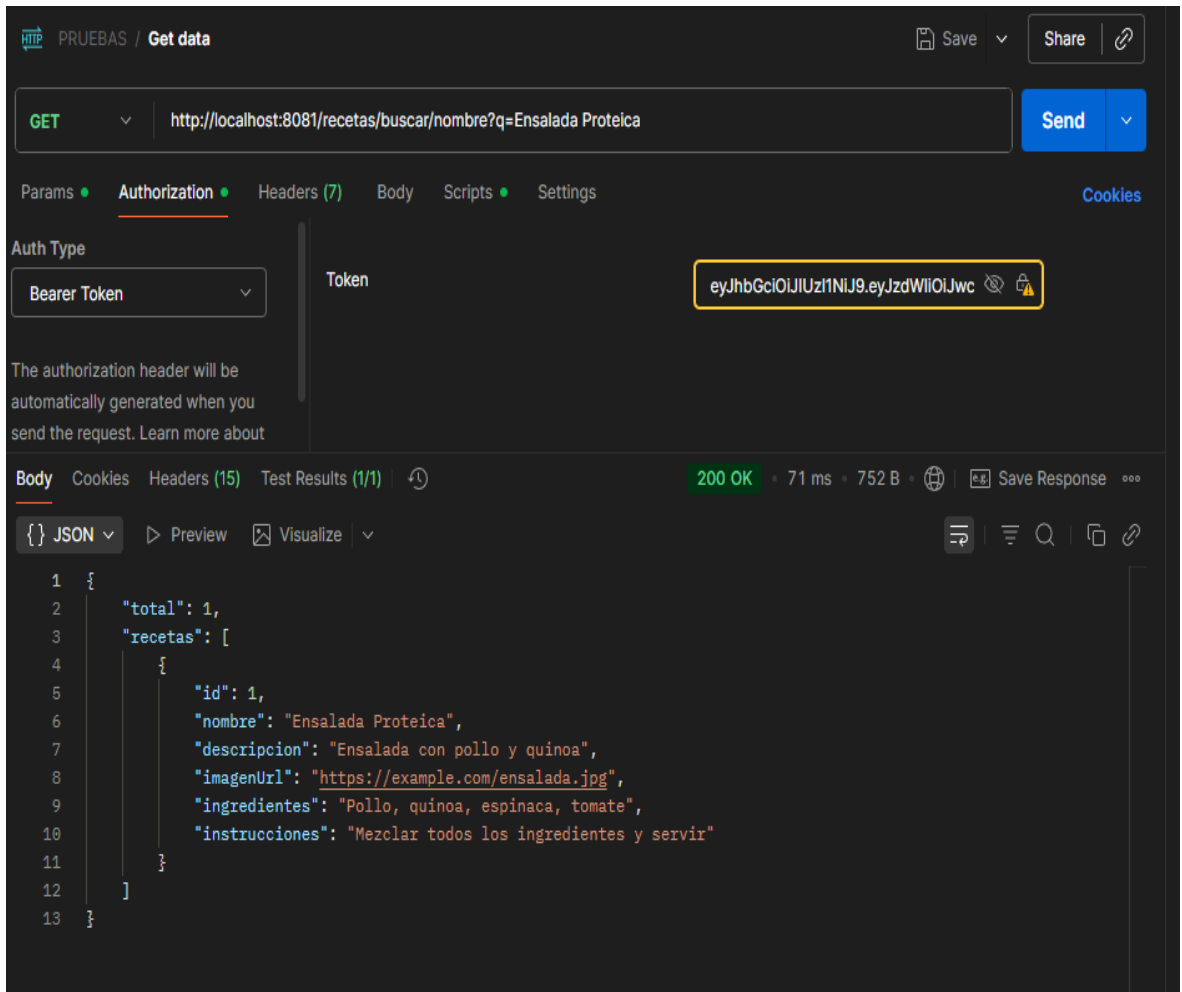
- **Probar el intercambio de datos mediante peticiones HTTP.**
Para las peticiones en Postman, se debe estar logueado correctamente, usando el token de autenticación, para acceder a las recetas:
 - Obtener todas las recetas



- Obtener una receta específica



- Buscar por nombre



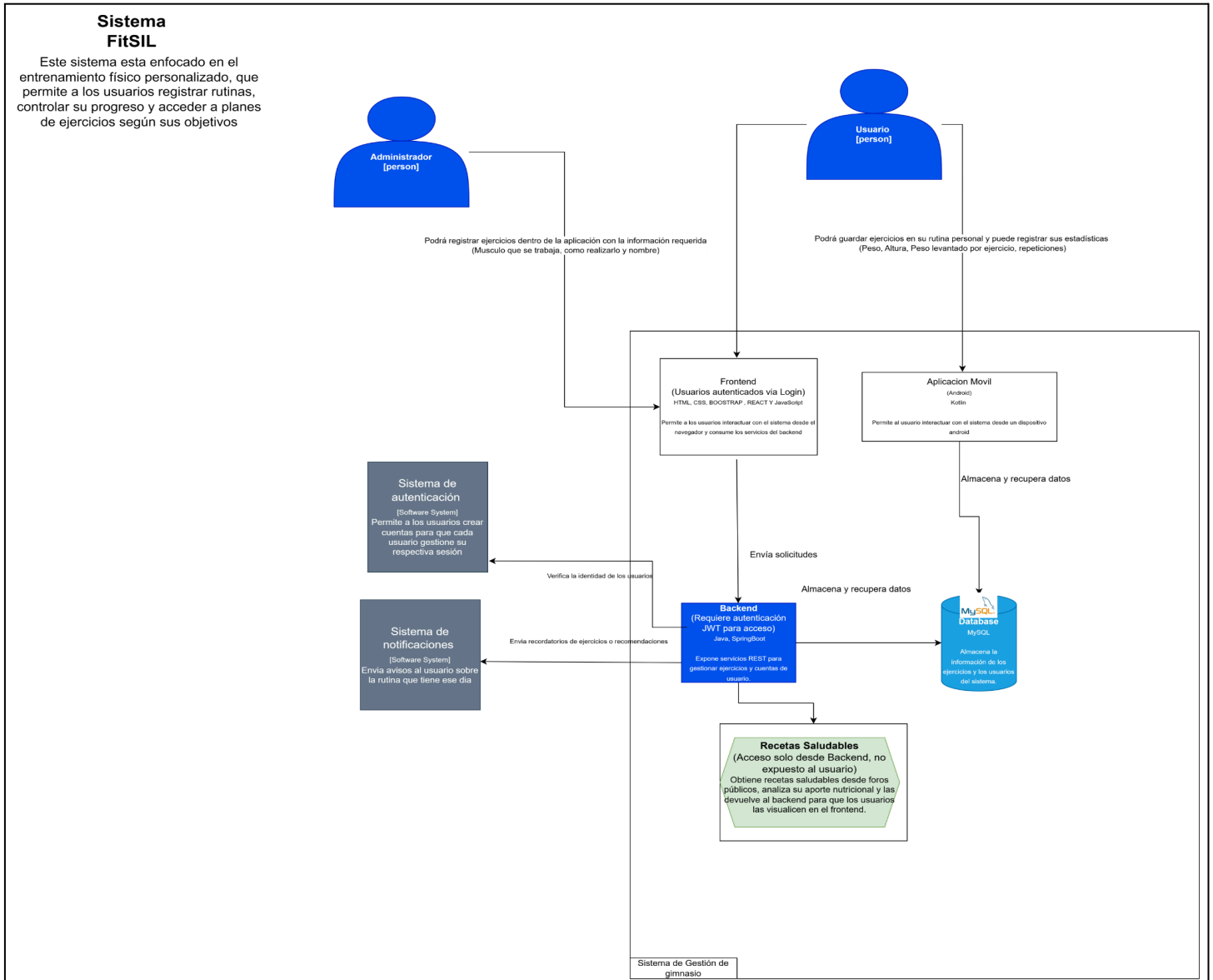
4. Despliegue y validación

- Ejecutar el microservicio en contenedor Docker o entorno local.
- Validar la conexión y registrar logs de comunicación.
- Documentar resultados en [/docs/architecture/container-diagram-v2.png](#).

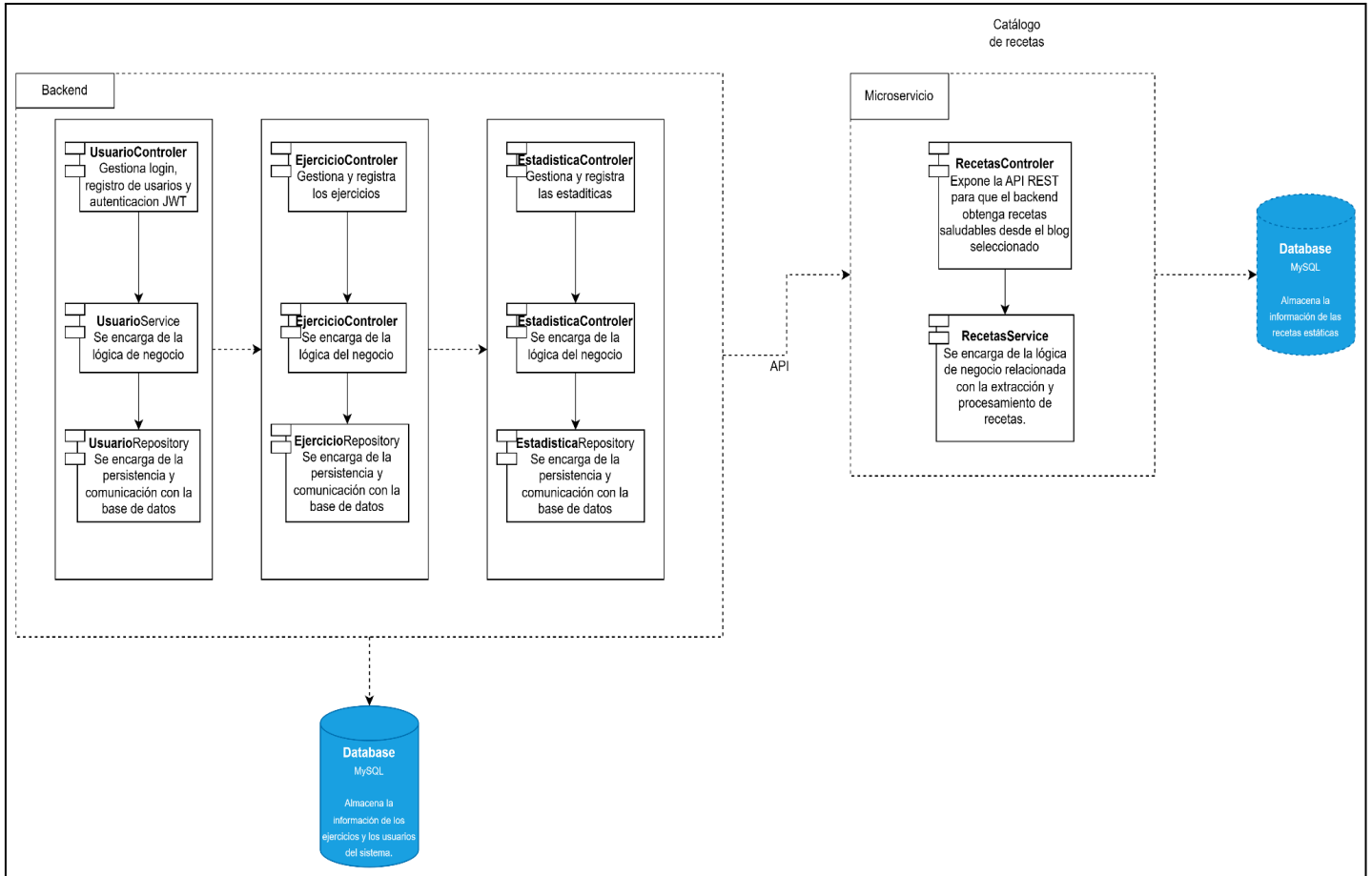
5. Registro en modelo C4

- Actualizar los diagramas **Container** y **Component** incorporando el microservicio y el gateway.
- Registrar capturas o exportar los diagramas actualizados.

DIAGRAMA C4 ACTUALIZADO Contenedores



Componentes



6. Resultados esperados:

- Microservicio desplegado y comunicándose correctamente con el backend principal.
- API Gateway o punto de integración funcional.
- Documentación técnica y diagramas C4 actualizados.
- Logs y capturas de comunicación.

7. Preguntas de Control:

- **¿Qué diferencia principal existe entre una arquitectura monolítica y una basada en microservicios?**

La arquitectura monolítica centra toda la lógica del sistema en una única aplicación y cada uno de sus módulos poseen un alto grado de acoplamiento y son desplegados en conjunto. Por el contrario una arquitectura basada en microservicios divide todo el sistema en servicios independientes, cada uno implementa su propia lógica, base de datos y su ejecución por separado.

- **¿Qué beneficios aporta el uso de un API Gateway en la integración de servicios?**

Un api Gateway funciona como un único punto de acceso para todas las peticiones por parte del usuario, algunos de sus beneficios son:

- seguridad todo en uno aplicando autenticación, autorización y un límite en cuanto a peticiones
- orquestación y enrutamiento con esto nos referimos a que cada solicitud se va a dirigir a un microservicio en específico.
- monitoreo para centralización de métricas y facilitar el mantenimiento y seguimiento de errores.
- desacoplamiento puesto a que los clientes no necesitan conocer las direcciones ni los detalles por los que están compuestos cada microservicio interno.

- **¿Qué riesgos se presentan al no manejar correctamente la comunicación entre microservicios?**

Algunos errores que pueden surgir cuando los microservicios no son manejados correctamente son:

- latencias elevadas: configuración incorrecta y dependencias circulares lo que desencadena un sistema lento
- fallas en cascada: si un servicio falla y no existen mecanismos de tolerancia, esto podría afectar al resto de módulos del sistema.
- inconsistencia en los datos: si no existe la sincronización entre los distintos microservicios del sistema se podrá generar estados incoherentes entre servicios
- vulnerabilidades en la seguridad: este punto se da generalmente cuando no se validan correctamente los datos y las credenciales entre servicios

- **¿Cómo se refleja la modularidad en los niveles Container y Component del modelo C4?**

Nivel Container: cada microservicio o aplicación como el backend o la base de datos se representan mediante un contenedor con

unas responsabilidades y objetivos específicos.
 Nivel Component: dentro de cada contenedor el código generalmente se divide en módulos, controladores, clases, paquetes, etc, donde cada uno es responsable de una parte específica de la lógica de negocio.

- ¿Qué consideraciones de seguridad se deben mantener al exponer endpoints entre servicios?
 La autenticación y autorización mediante el uso de tokens JWT o OAuth2 que ayuden a validar la identidad del servicio que realiza la petición
 Cifrado en las comunicaciones con el uso de métodos HTTP para interceptar los datos en tránsito y protegerlos
 Validaciones de entrada para gestionar y validar los datos recibidos por ejemplo desde un formulario y que no sean ataques de inyección o datos corruptos.

8. Evaluación

Criterio	2 – Logro Alto	1 – Logro Medio	0 – Bajo / Sin Evidencia
1. Diseño e implementación del microservicio	Servicio funcional, correctamente desacoplado y con endpoints operativos.	Servicio funcional con errores menores o acoplamiento parcial.	Servicio no funcional o ausente.
2. Integración con el backend principal	Comunicación fluida entre servicios y validación en Postman / Swagger.	Comunicación parcial o errores en la respuesta.	No hay evidencia de integración.
3. Documentación técnica y C4 actualizado	Diagramas Container y Component actualizados y coherentes.	Diagramas incompletos o desactualizados.	Sin actualización del modelo C4.
4. Despliegue y pruebas	Despliegue funcional en Docker o entorno local con logs claros.	Despliegue parcial o sin validaciones suficientes.	No se logra el despliegue.

5. Organización del repositorio y evidencias	Estructura ordenada, README actualizado, capturas y registros claros.	Estructura parcial o documentación incompleta.	Sin documentación ni evidencias.
---	---	--	----------------------------------

9. Bibliografía

- Newman, S. (2021). *Building Microservices*. O'Reilly Media.
- OWASP Foundation (2023). *OWASP Secure Microservices Design*.
- Docker Inc. *Docker Documentation*.
- Richardson, C. (2022). *Microservices Patterns: With examples in Java*. Manning Publications.


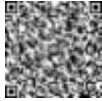


unl

Universidad
Nacional
de Loja

10. Elaboración y Aprobación

- Elaborado por: nombre del Docente responsable de la práctica.
- Revisado por: nombre del Técnico de laboratorio (solo si aplica).
- Aprobado por: nombre del Director de Carrera.

Elaborado por	Edison L Coronel Romero Docente	 Firmado electrónicamente por: EDISON LEONARDO CORONEL ROMERO Validar únicamente con FirmaBC
Aprobado por	Edison L Coronel Romero Director de Carrera	 Firmado electrónicamente por: EDISON LEONARDO CORONEL ROMERO Validar únicamente con FirmaBC