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

Daniel Aldazosa Mirnada

Segunda Evaluación de Tecnologías Web II 1-2025 – Primera Opción

---

Objetivo General

Diseñar e implementar un microservicio independiente de mensajería Telegram que consuma endpoints REST del proyecto monolito, validando usuarios y permitiendo envíos individuales y masivos. También debe ser usable desde el gateway para exponerse hacia frontend.

---

1. Análisis del Proyecto Integrador (15 pts)

1.1 Identificación clara de los endpoint(s) (5 pts)

Nombre	Método	Ruta	Descripción
Obtener lista de usuarios	GET	`/collections/usuarios`	Retorna todos los registros de usuario.
Obtener idTelegram por email	GET	`/collections/usuarios?filter=email="XXX"`	Recupera usuario(s) filtrando por correo.
Obtener un usuario concreto	GET	`/collections/usuarios/{id}`	Detalles de un usuario por su ID interno.
Actualizar idTelegram	PATCH	`/collections/usuarios/{id}`	Modifica el campo `idTelegram` de un usuario.

Ejemplo de request/response

GET /collections/usuarios?filter=email="juan@ejemplo.com" HTTP/1.1
Authorization: Bearer <token>

```
{
  "items": [
    { "id": "abc123", "email": "juan@ejemplo.com", "idTelegram": 123456789 }
  ]
}
```

1.2 Justificación de la elección (5 pts)

- Estos endpoints permiten al microservicio:
    1. **Validar existencia** de usuario (GET con filtro `email`).
    2. **Recuperar el `idTelegram`** necesario para enviar mensajes.
    3. **Actualizar** el registro cuando el usuario vincula su bot.
- Son suficientes para soportar tanto envío individual como masivo.

---

2. Diseño del Microservicio (15 pts)

2.1 Objetivo del microservicio (5 pts)

Construir un microservicio en **Spring-Boot** que:

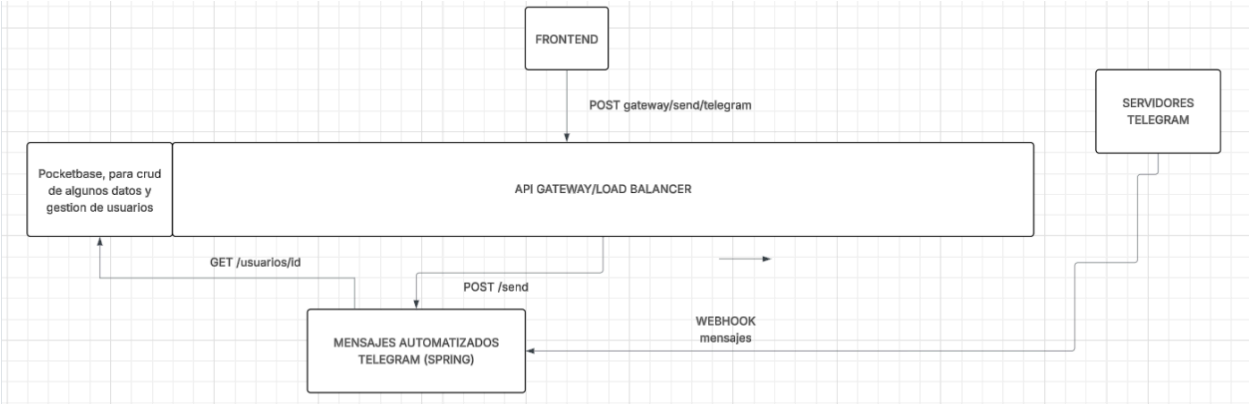
- Aloja un bot de Telegram (token en `.properties`).
- Permite:
    - **Enviar un mensaje individual.**
    - **Enviar mensajes masivos** con hilos.
- Consulta el monolito vía **REST/HTTP** para:
    - Validar usuarios.

- Obtener/actualizar `idTelegram`.

2.2 Elección y justificación de la tecnología de comunicación (5 pts)

- **REST sobre HTTP(S)**:
  - Ligero y compatible con el monolito (PocketBase + Go).
  - Gestionado por un API Gateway que maneja autenticación y balanceo.
  - Es posible usar con túneles http
- **Telegram-Spring-Bot**:
  - Librería madura para integración con la Bot API.
  - Facilita Long Polling

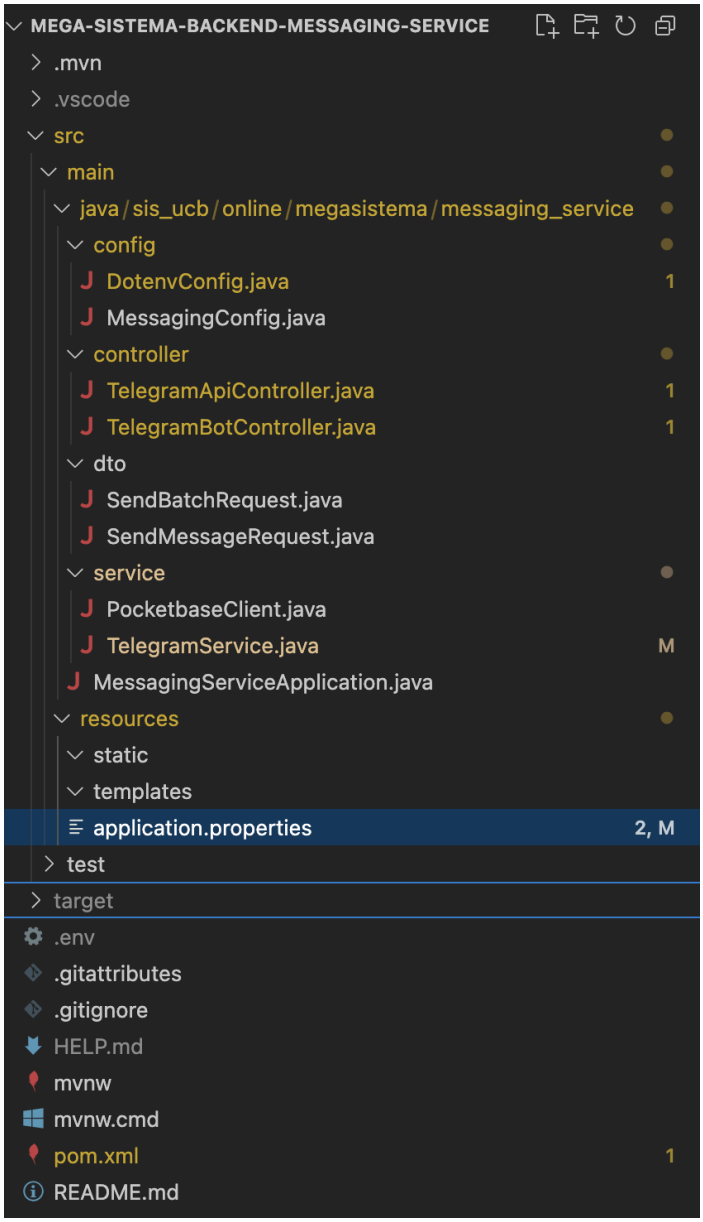
2.3 Diagrama del flujo de integración (5 pts)



---

3. Implementación Técnica (40 pts)

3.1 Estructura del proyecto



3.2 Desarrollo del microservicio funcional (15 pts)

- **Spring-Boot** (Java 17)
- **Dependencias**:
    - `spring-boot-starter-web`
    - `org.telegram:telegrambots-spring-boot-starter`

- `spring-boot-starter-webflux` (para hilos masivos)
- `spring-boot-starter-test`

```java
package sis_ucb.online.megasistema.messaging_service.controller;



import org.telegram.telegrambots.meta.api.objects.User;



import io.github.drednote.telegram.core.annotation.TelegramCommand;

import io.github.drednote.telegram.core.annotation.TelegramController;

import io.github.drednote.telegram.core.annotation.TelegramMessage;

import io.github.drednote.telegram.core.annotation.TelegramPatternVariable;

import io.github.drednote.telegram.core.annotation.TelegramRequest;

import io.github.drednote.telegram.core.request.UpdateRequest;

import io.github.drednote.telegram.response.GenericTelegramResponse;

import io.github.drednote.telegram.response.TelegramResponse;

import sis_ucb.online.megasistema.messaging_service.service.PocketbaseClient;



@TelegramController

public class TelegramBotController {

   private final PocketbaseClient pocketbaseClient;



   public TelegramBotController(PocketbaseClient pocketbaseClient) {

       this.pocketbaseClient = pocketbaseClient;
```

```
    }



    @TelegramCommand("/inicio")

    public String alIniciar(User usuario) {

        System.out.println("Usuario: " + usuario);

        return "Hola " + usuario.getFirstName();

    }



    @TelegramCommand("/inicio {email:.*}")

    public String alIniciarConEmail(@TelegramPatternVariable("email") String email,
User usuario) {




        System.out.println("Usuario: " + usuario);



        Boolean resultado = pocketbaseClient.updateUserTelegramId(usuario.getId(),
email).block();

        if (resultado == null || !resultado) {

            return "Error al actualizar el ID de Telegram. Por favor, intenta
nuevamente.";

        }
```

```
        return "Hola " + usuario.getFirstName() + ", tu ID de Telegram ha sido
actualizado con el email: " + email;



    }









    @TelegramCommand("/ayuda")

    public String alAyuda() {

        return "Comandos disponibles:\n" +

                "/inicio - Iniciar una conversación con el bot\n" +

                "/inicio {email} - Iniciar con tu email\n";

    }



    @TelegramRequest

    public TelegramResponse aTodos(UpdateRequest peticion) {

        return new GenericTelegramResponse("Comando no soportado. Prueba /ayuda para
ver los comandos disponibles.");

    }

}
```

```
package sis_ucb.online.megasistema.messaging_service.config;
```

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.reactive.function.client.WebClient;

@Configuration
public class MessagingConfig {

   @Value("${pocketbase.url}")
   private String pocketbaseUrl;

   @Value("${pocketbase.apiKey:}")  // cadena vacía si no existe
   private String pocketbaseApiKey;

   @Bean
   public WebClient pocketbaseWebClient() {
       return WebClient.builder()
               .baseUrl(pocketbaseUrl)
               .defaultHeader("Authorization", "Bearer " + pocketbaseApiKey)
               .build();
   }
}
```

3.3 Consumo correcto del endpoint externo (10 pts)

```java
package sis_ucb.online.megasistema.messaging_service.service;



import org.springframework.stereotype.Component;

import org.springframework.web.reactive.function.client.WebClient;

import reactor.core.publisher.Mono;



import java.util.List;

import java.util.Map;
```

```java
@Component

public class PocketbaseClient {

   private final WebClient webClient;



   public PocketbaseClient(WebClient webClient) {

       this.webClient = webClient;

   }



   public Mono<Boolean> checkUserPermission(Long telegramId, String permiso) {

       String uri =
String.format("/api/collections/usuarios/records?filter=telegramId=\"%d\"",
telegramId);

       return webClient.get()

               .uri(uri)

               .retrieve()

               .bodyToMono(Map.class)

               .doOnNext(resp -> System.out.println("Response CheckPermission: " +
resp))

               .map(resp -> {

                   @SuppressWarnings("unchecked")

                   List<Map<String, Object>> items = (List<Map<String, Object>>)
resp.get("items");

                   if (items == null || items.isEmpty()) return false;

                   // Based on the error, "rol" is a String, not a List

                   String rol = (String) items.get(0).get("rol");

                   return rol != null && rol.equals(permiso);
```

```
                });



    }



    public Mono<Boolean> updateUserTelegramId(Long telegramId, String email){



        String uri =
String.format("/api/collections/usuarios/records?filter=email=\"%s\"", email);

        return webClient.get()

                .uri(uri)

                .retrieve()

                .bodyToMono(Map.class)

                .doOnNext(resp -> System.out.println("Response UpdateTelegramId: " +
resp))

                .flatMap(resp -> {

                    @SuppressWarnings("unchecked")

                    List<Map<String, Object>> items = (List<Map<String, Object>>)
resp.get("items");

                    if (items == null || items.isEmpty()) return Mono.just(false);

                    String id = (String) items.get(0).get("id");

                    return webClient.patch()

                            .uri("/api/collections/usuarios/records/" + id)

                            .bodyValue(Map.of("telegramId", telegramId))

                            .retrieve()

                            .bodyToMono(Map.class)
```

```
                        .map(response -> true)

                        .onErrorReturn(false);

                });

    }



    public Mono<Long> findChatIdByIdentifier(String identifier) {

        String uri =
String.format("/api/collections/usuarios/records?filter=id=\"%s\"", identifier);



        return webClient.get()

                .uri(uri)

                .retrieve()

                .bodyToMono(Map.class)

                .doOnNext(resp -> System.out.println("Response: " + resp))

                .flatMap(resp -> {

                    @SuppressWarnings("unchecked")

                    List<Map<String, Object>> items = (List<Map<String, Object>>)
resp.get("items");

                    if (items == null || items.isEmpty()) return Mono.error(new
IllegalArgumentException("Usuario no encontrado: " + identifier));

                    Object telegramIdObj = items.get(0).get("telegramId");

                    if (telegramIdObj == null) {

                        return Mono.error(new IllegalArgumentException("telegramId no
encontrado para el usuario: " + identifier));

                    }
```

```
                try {
                    return Mono.just(Long.parseLong(telegramIdObj.toString()));
                } catch (NumberFormatException e) {
                    return Mono.error(new IllegalArgumentException("telegramId no
válido: " + telegramIdObj));
                }
            });
    }


}
```

3.4 Procesamiento y transformación de los datos (10 pts)

- **Individual**: invocar `sendMessage(chatId, texto)` de la librería.

```
public void sendTo(String identifier, String text) {
        SendMessage msg = SendMessage.builder()
                .chatId(identifier.toString())
                .text(text)
                .build();

        // Ahora sí podemos llamar a execute()
        try {
            telegramClient.execute(msg);
```

```
        } catch (TelegramApiException e) {
            // TODO Auto-generated catch block
            System.out.println("Error al enviar el mensaje: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- **Masivo**:

```
public void sendBatch(List<String> identifiers, String text) {
        identifiers.parallelStream().forEach(identifier -> {
            SendMessage msg = SendMessage.builder()
                    .chatId(identifier)
                    .text(text)
                    .build();

            try {
                telegramClient.execute(msg);
            } catch (TelegramApiException e) {
                System.out.println("Error al enviar el mensaje a " + identifier
+ ": " + e.getMessage());
                e.printStackTrace();
            }
        });
    }
```

- **Transformación**: extraer `idTelegram` del JSON, construir payload.

```
return webClient.get()

                .uri(uri)

                .retrieve()

                .bodyToMono(Map.class)

                .doOnNext(resp -> System.out.println("Response: " + resp))

                .flatMap(resp -> {

                    @SuppressWarnings("unchecked")

                    List<Map<String, Object>> items = (List<Map<String,
Object>>) resp.get("items");

                    if (items == null || items.isEmpty()) return Mono.error(new
IllegalArgumentException("Usuario no encontrado: " + identifier));

                    Object telegramIdObj = items.get(0).get("telegramId");

                    if (telegramIdObj == null) {

                        return Mono.error(new
IllegalArgumentException("telegramId no encontrado para el usuario: " +
identifier));

                    }



                    try {

                        return
Mono.just(Long.parseLong(telegramIdObj.toString()));

                    } catch (NumberFormatException e) {
```

```
                        return Mono.error(new
IllegalArgumentException("telegramId no válido: " + telegramIdObj));

                    }

                });
```

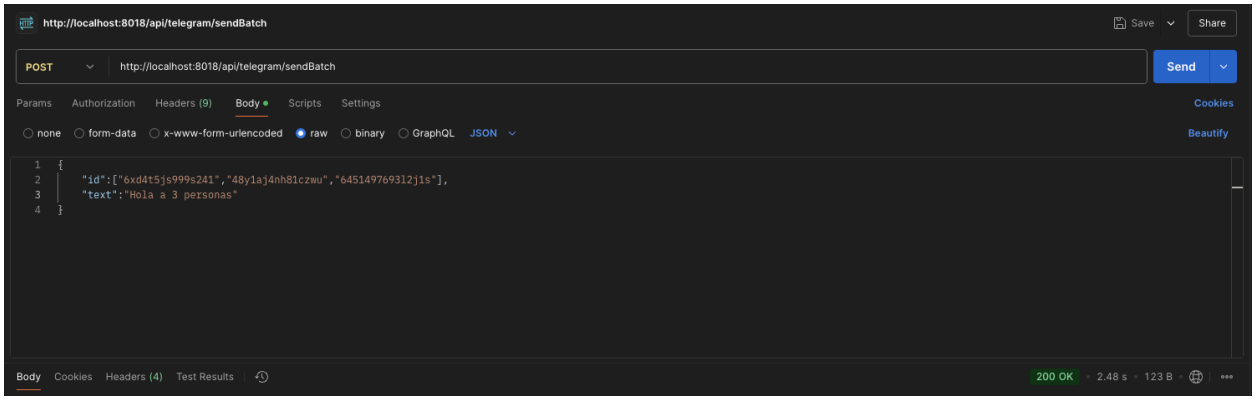
3.5 Exposición de endpoint propio/documentación (5 pts)

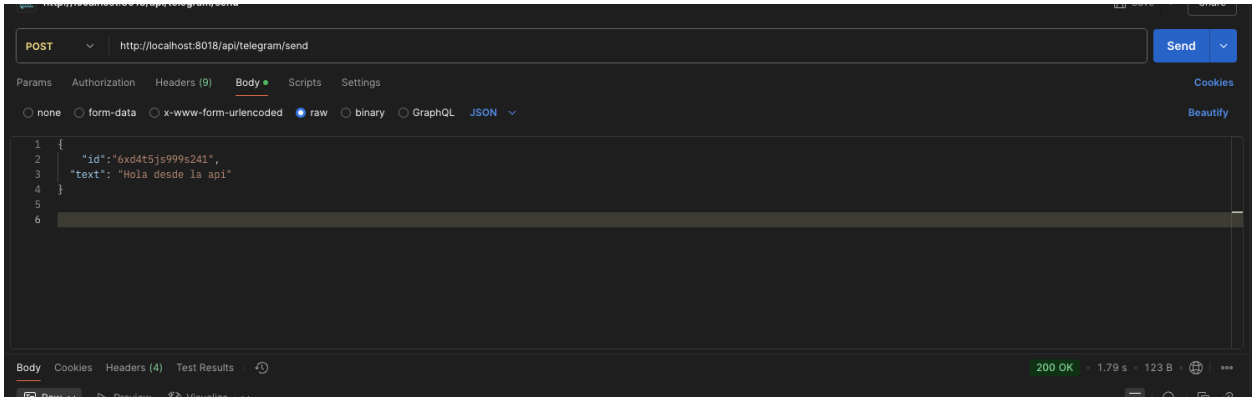
Controller:

```
package sis_ucb.online.megasistema.messaging_service.controller;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import reactor.core.publisher.Mono;
import sis_ucb.online.megasistema.messaging_service.dto.SendBatchRequest;
import sis_ucb.online.megasistema.messaging_service.dto.SendMessageRequest;
import sis_ucb.online.megasistema.messaging_service.service.PocketbaseClient;
import sis_ucb.online.megasistema.messaging_service.service.TelegramService;

@RestController
@RequestMapping("/api/telegram")
public class TelegramApiController {
   private final TelegramService telegramService;
   private final PocketbaseClient pocketbaseClient;

   public TelegramApiController(TelegramService telegramService, PocketbaseClient
pocketbaseClient) {
       this.telegramService = telegramService;
       this.pocketbaseClient = pocketbaseClient;
   }

   @PostMapping("/send")
   public ResponseEntity<Void> sendMessage(@RequestBody SendMessageRequest req) {
       Long chatId = pocketbaseClient.findChatIdByIdentifier(req.getId()).block();
       //Boolean permissionCheck = pocketbaseClient.checkUserPermission(chatId,
"docente").block();
       //if (permissionCheck == null || !permissionCheck) {
         //  return ResponseEntity.status(403).build(); // Forbidden
```

```
      // }
       if (chatId == null) {
           return ResponseEntity.badRequest().build(); // Bad Request
       }
       telegramService.sendTo(chatId.toString(), req.getText());
       return ResponseEntity.ok().build();
   }

   @PostMapping("/sendBatch")
   public ResponseEntity<Void> sendBatch(@RequestBody SendBatchRequest req) {
       for (String id : req.getId()) {
           Long chatId = pocketbaseClient.findChatIdByIdentifier(id).block();
           //Boolean permissionCheck = pocketbaseClient.checkUserPermission(chatId,
"docente").block();
           //if (permissionCheck == null || !permissionCheck) {
             //  return ResponseEntity.status(403).build(); // Forbidden
          // }
           if (chatId == null) {
               return ResponseEntity.badRequest().build(); // Bad Request
           }
           telegramService.sendTo(chatId.toString(), req.getText());
       }
       return ResponseEntity.ok().build();
   }
}
```

-
- **Documentar** con **Swagger/OpenAPI** o **Postman Collection**.

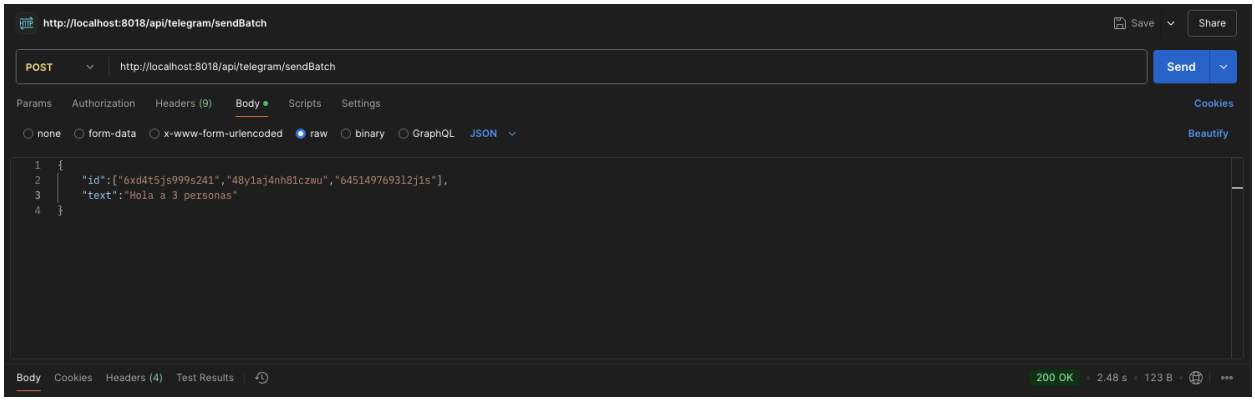


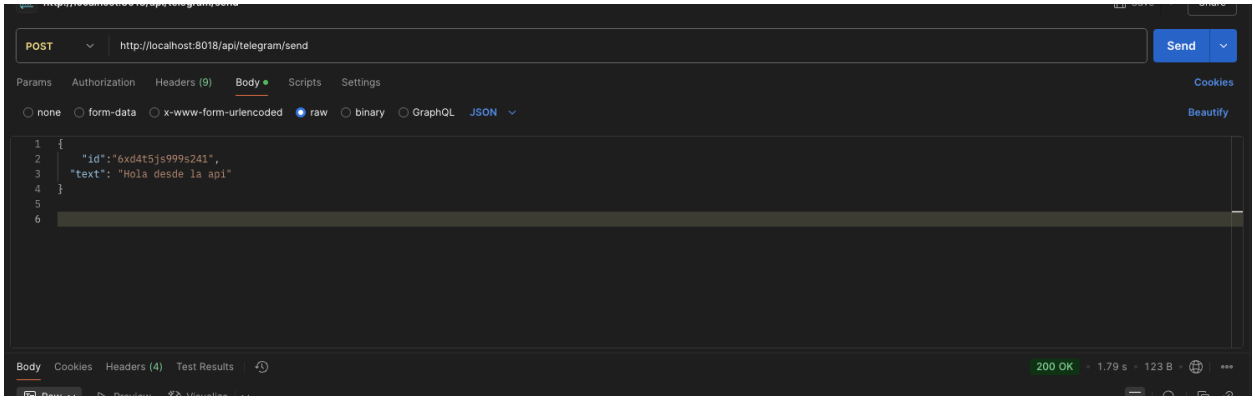


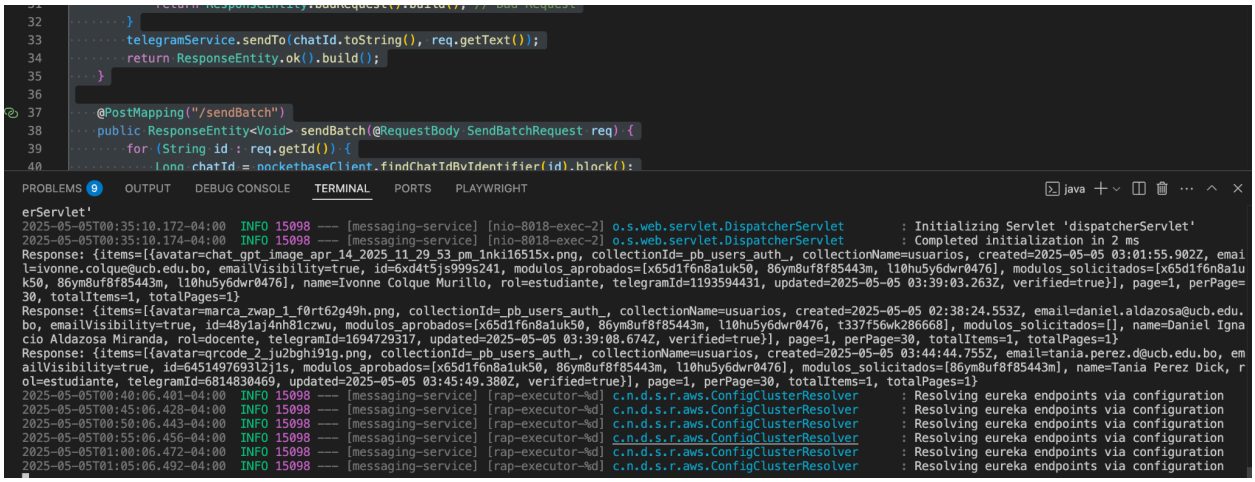
---

4. Pruebas y Documentación (15 pts)

4.1 Evidencias funcionales (5 pts)

- 

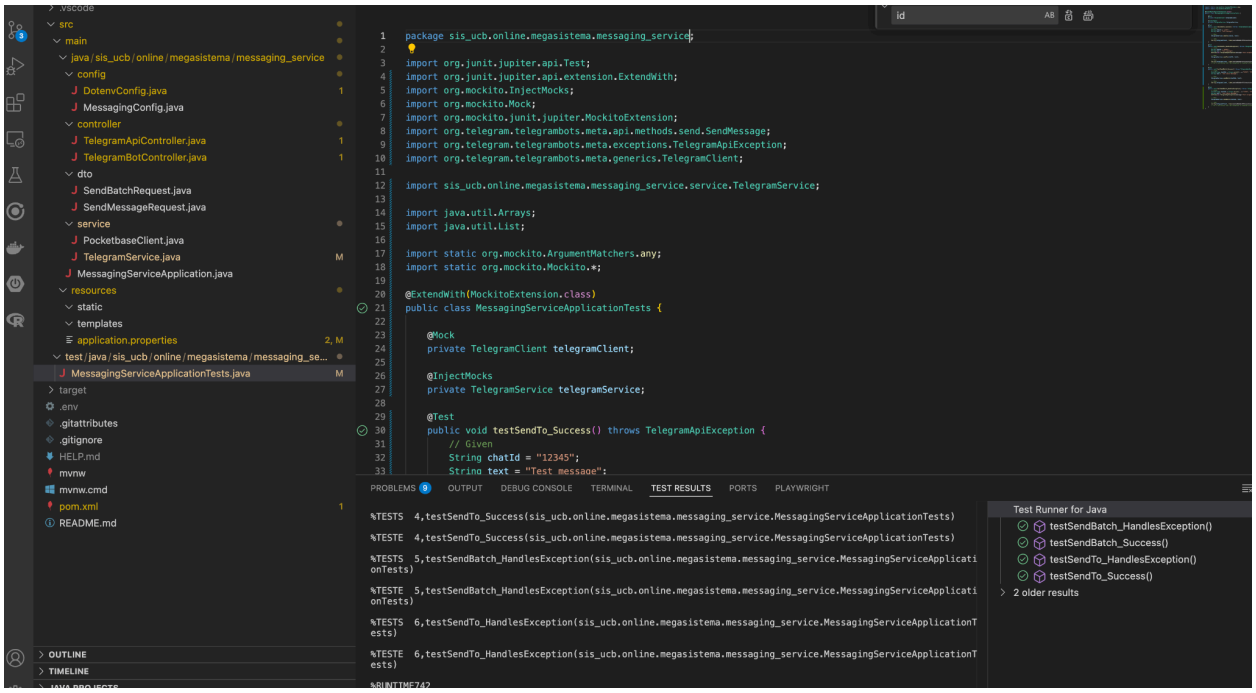




En estas imágenes se puede ver el funcionamiento y conexión con el monolito

4.2 Pruebas unitarias o manuales explicadas (5 pts)

- **Unitarias**:



Con los siguientes tests

```
package sis_ucb.online.megasistema.messaging_service;
```

```
import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import org.telegram.telegrambots.meta.api.methods.send.SendMessage;

import org.telegram.telegrambots.meta.exceptions.TelegramApiException;

import org.telegram.telegrambots.meta.generics.TelegramClient;



import sis_ucb.online.megasistema.messaging_service.service.TelegramService;



import java.util.Arrays;
```

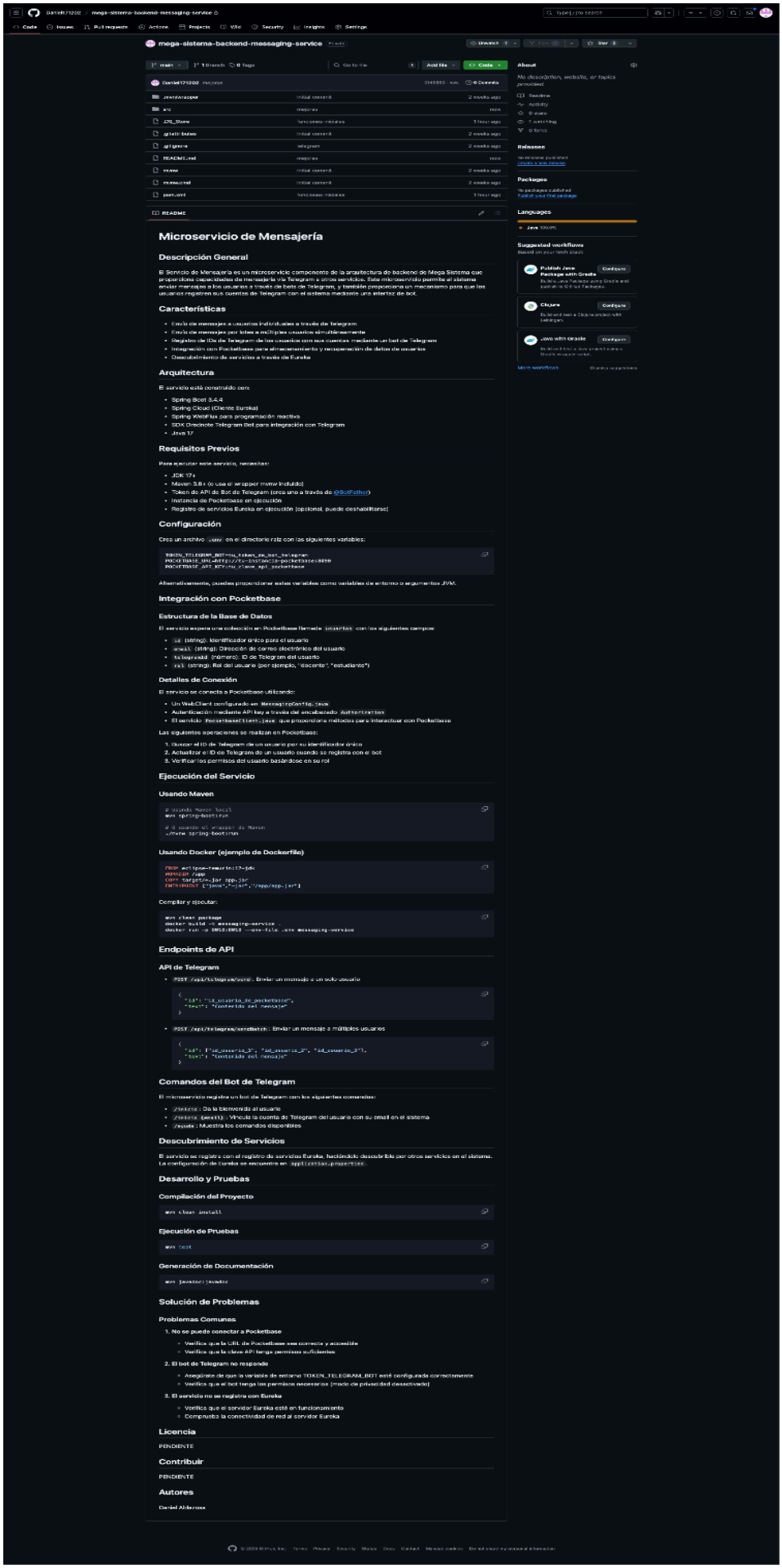
```java
import java.util.List;



import static org.mockito.ArgumentMatchers.any;

import static org.mockito.Mockito.*;



@ExtendWith(MockitoExtension.class)

public class MessagingServiceApplicationTests {



   @Mock

   private TelegramClient telegramClient;



   @InjectMocks

   private TelegramService telegramService;



   @Test

   public void testSendTo_Success() throws TelegramApiException {

       // Given

       String chatId = "12345";

       String text = "Test message";



       // When

       telegramService.sendTo(chatId, text);



       // Then
```

```
        verify(telegramClient, times(1)).execute(any(SendMessage.class));
    }

    @Test
    public void testSendTo_HandlesException() throws TelegramApiException {
        // Given
        String chatId = "12345";
        String text = "Test message";
        doThrow(new TelegramApiException("Test exception")).when(telegramClient).execute(any(SendMessage.class));

        // When
        telegramService.sendTo(chatId, text);

        // Then
        verify(telegramClient, times(1)).execute(any(SendMessage.class));
        // We're verifying that the exception is handled without being propagated
    }

    @Test
    public void testSendBatch_Success() throws TelegramApiException {
        // Given
        List<String> chatIds = Arrays.asList("12345", "67890");
        String text = "Test batch message";
```

```
        // When
        telegramService.sendBatch(chatIds, text);

        // Then
        verify(telegramClient, times(2)).execute(any(SendMessage.class));
    }

    @Test
    public void testSendBatch_HandlesException() throws TelegramApiException {
        // Given
        List<String> chatIds = Arrays.asList("12345", "67890");
        String text = "Test batch message";
        doThrow(new TelegramApiException("Test
exception")).when(telegramClient).execute(any(SendMessage.class));

        // When
        telegramService.sendBatch(chatIds, text);

        // Then
        verify(telegramClient, times(2)).execute(any(SendMessage.class));
        // We're verifying that the exception is handled without being
propagated
    }
}
```

4.3 Documentación clara del microservicio (5 pts)

- **README.md**:



Microservicio de Mensajería

Descripción General

El Servicio de Mensajería es un microservicio componente de la arquitectura de backend de Mega Sistema que proporciona capacidades de mensajería vía Telegram a otros servicios. Este microservicio permite al sistema enviar mensajes a los usuarios a través de bots de Telegram, y también proporciona un mecanismo para que los usuarios registren sus cuentas de Telegram con el sistema mediante una interfaz de bot.

Características

- Envío de mensajes a usuarios individuales a través de Telegram
- Envío de mensajes por lotes a múltiples usuarios simultáneamente
- Registro de IDs de Telegram de los usuarios con sus cuentas mediante un bot de Telegram
- Integración con Pocketbase para almacenamiento y recuperación de datos de usuarios
- Descubrimiento de servicios a través de Eureka

Arquitectura

El servicio está construido con:

- Spring Boot 3.4.4
- Spring Cloud (Cliente Eureka)
- Spring WebFlux para programación reactiva
- SDK Onecnote Telegram Bot para integración con Telegram
- Java 17

Requisitos Previos

Para ejecutar este servicio, necesitas:

- JDK 17+
- Maven 3.6+ (o usa el wrapper mvnw incluido)
- Token de API de Bot de Telegram (crea uno a través de @BotFather)
- Instancia de Pocketbase en ejecución
- Registro de servicios Eureka en ejecución (opcional, puede deshabilitarse)

Configuración

Crea un archivo `.env` en el directorio raíz con las siguientes variables:

```
TOKEN_TELEGRAM_BOT=su_token_de_bot_telegram
POCKETBASE_URL=http://ts-instancia-pocketbase:8090
POCKETBASE_API_KEY=su_clave_api_pocketbase
```

Alternativamente, puedes proporcionar estas variables como variables de entorno o argumentos JVM.

Integración con Pocketbase

Estructura de la Base de Datos

El servicio espera una colección en Pocketbase llamada `usuarios` con los siguientes campos:

- `id` (string): Identificador único para el usuario
- `email` (string): Dirección de correo electrónico del usuario
- `telegramId` (número): ID de Telegram del usuario
- `rol` (string): Rol del usuario (por ejemplo, "docente", "estudiante")

Detalles de Conexión

El servicio se conecta a Pocketbase utilizando:

- Un WebClient configurado en `MessagingConfig.java`
- Autenticación mediante API key a través del encabezado `Authorization`
- El servicio `PocketbaseClient.java` que proporciona métodos para interactuar con Pocketbase

Las siguientes operaciones se realizan en Pocketbase:

1. Buscar el ID de Telegram de un usuario por su identificador único
2. Actualizar el ID de Telegram de un usuario cuando se registra con el bot
3. Verificar los permisos del usuario basándose en su rol

Ejecución del Servicio

Usando Maven

```
# Usando Maven local
mvn spring-boot:run

# O usando el wrapper de Maven
./mvnw spring-boot:run
```

Usando Docker (ejemplo de Dockerfile)

```
FROM eclipse-temurin:17-jdk
WORKDIR /app
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar","/app/app.jar"]
```

Compilar y ejecutar:

```
mvn clean package
docker build -t messaging-service .
docker run -p 8082:8082 --env-file .env messaging-service
```

Endpoints de API

API de Telegram

- `POST /api/telegram/send`: Enviar un mensaje a un solo usuario

```
{
  "id": "id_usuario_de_pocketbase",
  "text": "Contenido del mensaje"
}
```

- `POST /api/telegram/sendBatch`: Enviar un mensaje a múltiples usuarios

```
{
  "id": ["id_usuario_1", "id_usuario_2", "id_usuario_3"],
  "text": "Contenido del mensaje"
}
```

Comandos del Bot de Telegram

El microservicio registra un bot de Telegram con los siguientes comandos:

- `/inicio`: Da la bienvenida al usuario
- `/inicio {email}`: Vincula la cuenta de Telegram del usuario con su email en el sistema
- `/ayuda`: Muestra los comandos disponibles

Descubrimiento de Servicios

El servicio se registra con el registro de servicios Eureka, haciéndolo descubrible por otros servicios en el sistema. La configuración de Eureka se encuentra en `application.properties`.

Desarrollo y Pruebas

Compilación del Proyecto

```
mvn clean install
```

Ejecución de Pruebas

```
mvn test
```

Generación de Documentación

```
mvn javadoc:javadoc
```

Solución de Problemas

Problemas Comunes

1. **No se puede conectar a Pocketbase**
   - Verifica que la URL de Pocketbase sea correcta y accesible
   - Verifica que la clave API tenga permisos suficientes
2. **El bot de Telegram no responde**
   - Asegúrate de que la variable de entorno TOKEN_TELEGRAM_BOT esté configurada correctamente
   - Verifica que el bot tenga los permisos necesarios (modo de privacidad desactivado)
3. **El servicio no se registra con Eureka**
   - Verifica que el servidor Eureka esté en funcionamiento
   - Comprueba la conectividad de red al servidor Eureka

Licencia

PENDIENTE

Contribuir

PENDIENTE

Autores

Daniel Alderoza

---

5. Presentación Final (15 pts)

VIDEO FUNCIONAMIENTO Y CÓDIGO

Screen Recording 2025-05-05 at 1.52.11 AM.mov

https://drive.google.com/file/d/19dk7d7eG6HR4UNY6VvN8LNg7dQiWFgyx/view?usp=sharing