

**Universidad Católica Boliviana “San Pablo”**

**Facultad de Ingeniería**

**Carrera de Ingeniería de Sistemas**

**Microservicio de Envío de Correos Personalizados**



**Segunda Evaluación de Tecnologías Web II 1-2025**

**Docente:** Ing. Miguel Pacheco

**Paralelo:** 1

**Integrante grupo:**  
Zarate Luque Jhessika Yancarla

**Fecha entrega:**

*La Paz - Bolivia*  
*2025*

# 1. Análisis del Proyecto Integrador (15 pts)

## 1.1 Identificación clara del endpoint (5 pts)

El endpoint utilizado está expuesto por el servidor principal [main.go](#) del proyecto integrador mediante la ruta:

- [/api/custom/email](#)
- [/api/custom/emails](#)

## 1.2 Justificación de la elección del endpoint (5 pts)

Este endpoint permite enviar correos electrónicos personalizados a los usuarios registrados en el sistema. Se seleccionó porque es crítico para la comunicación institucional, y se puede extender para notificaciones como recordatorios de eventos, anuncios o entrega de certificados.

## 1.3 Descripción técnica del endpoint (5 pts)

Ambas rutas aceptan solicitudes POST con un JSON que incluye:

- **recipients:** lista de correos electrónicos
- **subject:** asunto
- **body:** contenido (texto o HTML)

Ejemplo:

- [/api/custom/email](#)

```
{  
  
  "recipients": ["daniel.aldazosa@gmail.com",  
  
    "daniel.alsa@gmail.com",  
  
    "jhessikazarate@gmail.com",  
  
    "tania.perez.d@ucb.edu.bo"],  
  
  "subject": "NOTASSSS",  
  
  "body": ""  
}
```

Característica	Descripción
Método	POST (email)
URL	<code>/api/custom/email</code>
Formato de datos (email)	JSON ( <code>recipients</code> , <code>subject</code> , <code>body</code> )
Formato de respuesta	JSON( <code>invalidRecipients</code> , <code>microserviceReply</code> , <code>success</code> )
Autenticación	JWT en headers ( <code>application/json</code> )

- `/api/custom/emails`

```
{
  "recipients": ["jhessikazarate@gmail.com", "correo2@example.com"],
  "subject": "Asunto del correo 5",
  "body": "<!DOCTYPE html>\n<html lang=\"es\">\n  <head>\n    <meta\ncharset=\"UTF-8\">\n    <meta name=\"viewport\"\ncontent=\"width=device-width, initial-scale=1.0\">\n    <style>\n      body\n      {\n        font-family: Arial, sans-serif;\n        background-color:\n#f3f4f6;\n        margin: 0;\n        padding: 0;\n      }\n      .container\n      {\n        max-width: 600px;\n        margin: 0 auto;\n        background-color: #ffffff;\n        border: 1px solid #e2e8f0;\n        border-radius: 8px;\n        overflow: hidden;\n      }\n      .header {\n        background-color: #000f4a;\n        padding: 20px;\n        text-align:\ncenter;\n      }\n      .header img {\n        width: 100%;\n        max-width: 560px;\n        height: auto;\n        border-radius: 8px;\n      }\n      .welcome {\n        font-size: 1.5rem;\n        color: #00197c;\n        text-align: center;\n        margin-top: 20px;\n      }\n      .message {\n        padding: 20px;\n        font-size: 1rem;\n        color: #333;\n        text-align: justify;\n      }\n      .footer {\n        padding: 20px;\n        background-color: #f3f4f6;\n        text-align: center;\n        font-size:\n0.9rem;\n        color: #666;\n      }\n      .footer a {\n        color:\n#0026c3;\n        text-decoration: none;\n      }\n    </style>\n  </head>\n  <body>\n    <div class=\"container\">\n      <div class=\"header\">\n        <img\nsrc=\"https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRDOIyeUw-xVtSHzh\no0un9hE3r-gNXb_pF9WQ&s\" alt=\"Imagen de cabecera\" />\n      </div>\n      <div class=\"welcome\">Bienvenida Jhessika!</div>\n      <div
```

```
class="message">\n      Este es un mensaje de prueba para verificar que  
el diseño del correo se visualiza correctamente. Aquí puedes colocar  
cualquier contenido importante, como noticias, recordatorios o información  
relevante para los destinatarios.\n    </div>\n    <div  
class="footer">\n      © 2025 Universidad Católica Boliviana | <a  
href="https://ucb.edu.bo">Visita nuestro sitio web</a>\n    </div>\n  </div>\n</body>\n</html>"  
}
```

Característica	Descripción
Método	POST (email)
URL	<code>/api/custom/emails</code>
Formato de datos (email)	JSON ( <code>recipients</code> , <code>subject</code> , <code>body</code> )
Formato de respuesta	JSON( <code>invalidRecipients</code> , <code>microserviceReply</code> , <code>success</code> )
Autenticación	JWT en headers ( <code>application/json</code> )

El endpoint verifica que los destinatarios existan en la tabla `usuarios` de PocketBase. Luego reenvía la información de los destinatarios que si existen al microservicio responsable de enviar los correos.

## 2. Diseño del Microservicio (15 pts)

### 2.1 Objetivo del microservicio claramente definido (5 pts)

El microservicio tiene como objetivo:

- Enviar correos de manera concurrente a los usuarios de la carrera de Ingeniería de Sistemas.
- Reintentar el envío de correos fallidos cada 3 minutos.

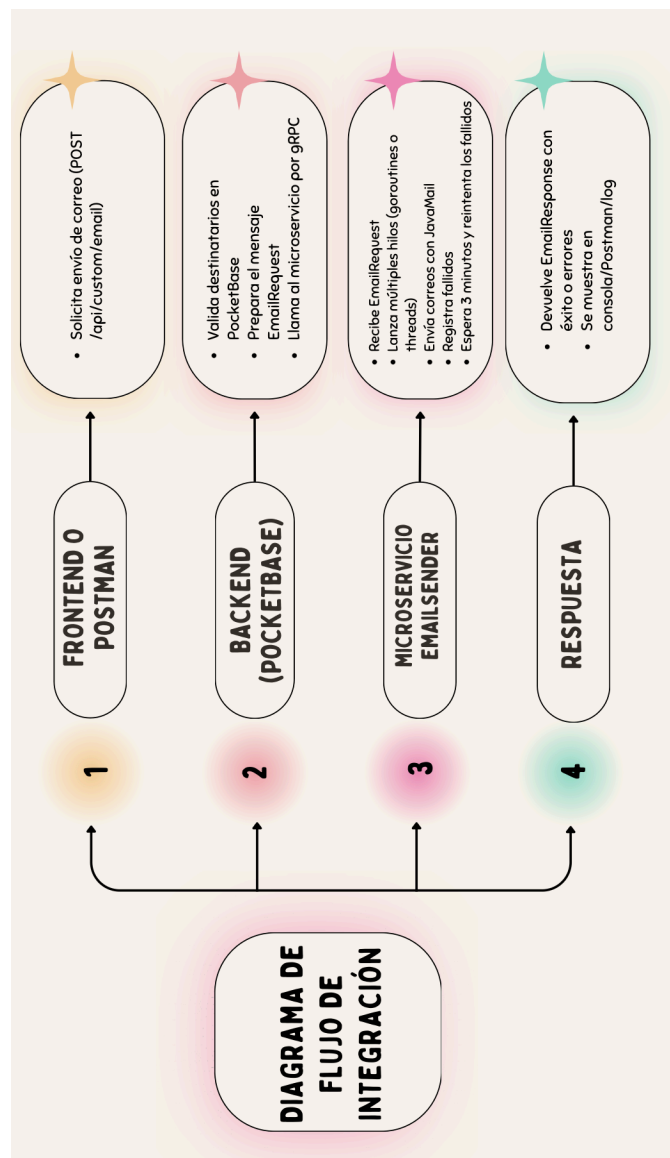
### 2.2 Elección y justificación de la tecnología de comunicación (5 pts)

Se eligió gRPC por su eficiencia y rendimiento, ideal para sistemas distribuidos. Su uso permite la comunicación cliente-servidor con soporte para concurrencia y reintentos automáticos.

## 2.3 Diagrama del flujo de integración (5 pts)

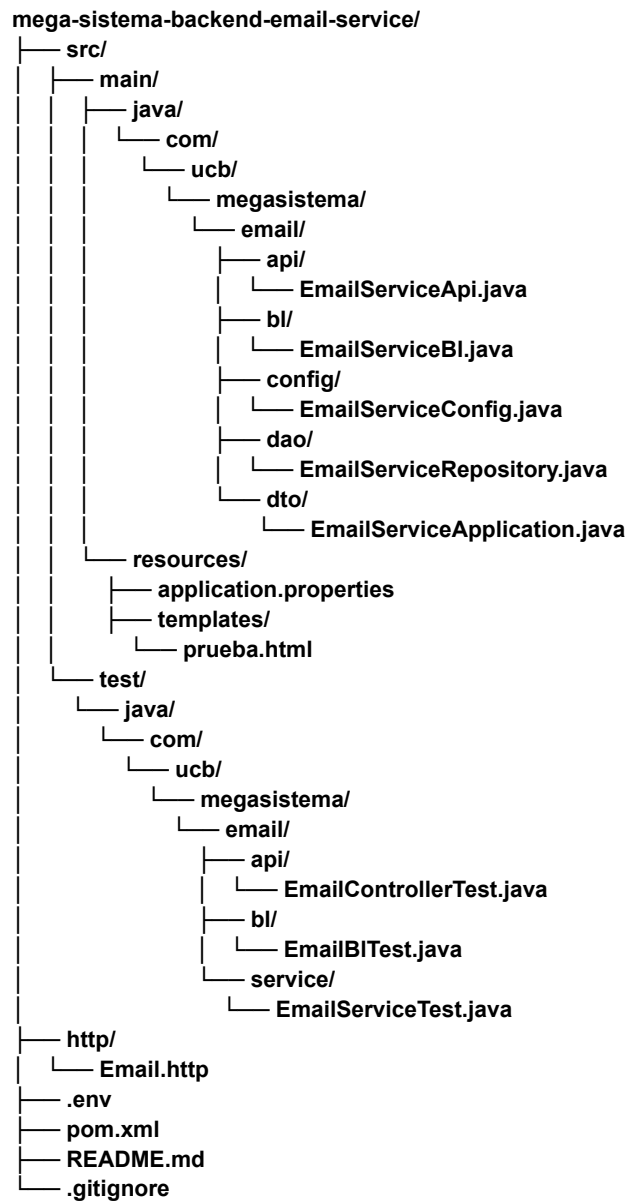
Flujo resumido:

1. Cliente envía solicitud POST a `/api/custom/email`.
2. El backend verifica en PocketBase si los correos existen.
3. Se envían los datos al microservicio por gRPC.
4. El microservicio procesa los envíos concurrentemente.
5. Correos fallidos se almacenan y se reintentan cada 3 minutos.



## 3. Implementación Técnica (40 pts)

### 3.1.1 Estructura del proyecto



### 3.1.2 Código base

#### EmailServiceApi.java

```
package sis_ucb.online.megasistema.email_service.api;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
```

```

import org.springframework.web.bind.annotation.RestController;

import sis_uch.online.megasistema.email_service.dto.EmailServiceDTO;
import sis_uch.online.megasistema.email_service.bl.EmailServiceBl;

@RestController
@RequestMapping("/api/v1/auth")
public class EmailServiceApi {

    @Autowired
    private EmailServiceBl emailServiceBl;

    @PostMapping("/send-email-invitation")
    public ResponseEntity<String> sendMailInvitation(@RequestBody EmailServiceDTO
email) {
        try {
            emailServiceBl.sendMailInvitation(email);
            return new ResponseEntity<>("Email enviado exitosamente",
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>("Error sending email: " + e.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @PostMapping("/send-email-html")
    public ResponseEntity<String> sendEmailWithHTML(@RequestBody EmailServiceDTO
email) {
        try {
            emailServiceBl.sendEmailWithHTML(email);
            return new ResponseEntity<>("Email enviado exitosamente",
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>("Error sending email: " + e.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}

```

## EmailServiceBl.java

```

package sis_uch.online.megasistema.email_service.bl;

import java.io.IOException;

```

```

import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;
import org.thymeleaf.TemplateEngine;
import org.thymeleaf.context.Context;

import sis_uchb.online.megasistema.email_service.dao.EmailServiceRepository;
import sis_uchb.online.megasistema.email_service.dto.EmailServiceDTO;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.*;

@Service
public class EmailServiceBl implements EmailServiceRepository {
    private final JavaMailSender javaMailSender;
    private final TemplateEngine templateEngine;

    public EmailServiceBl(JavaMailSender javaMailSender, TemplateEngine
templateEngine) {
        this.javaMailSender = javaMailSender;
        this.templateEngine = templateEngine;
    }

    @Override
    public void sendMailInvitation(EmailServiceDTO emailServiceDTO) {
        List<String> recipients = emailServiceDTO.getRecipients();
        int totalRecipients = recipients.size();

        // Máximo 20 hilos para evitar sobrecarga
        int threadCount = Math.min(totalRecipients, 20);
        ExecutorService executor = Executors.newFixedThreadPool(threadCount);

        // Lista de correos que fallaron (segura para hilos)
        List<String> failedRecipients = Collections.synchronizedList(new ArrayList<>());

```



```

CountDownLatch latch = new CountDownLatch(totalRecipients);

for (String recipient : recipients) {
    executor.submit(() -> {
        try {
            sendSingleEmail(recipient, emailServiceDTO);
        } catch (Exception e) {
            System.err.println("Error al enviar a " + recipient + ": " +
e.getMessage());
            failedRecipients.add(recipient);
        } finally {
            latch.countDown();
        }
    });
}

executor.shutdown();

try {
    latch.await(); // Esperar que terminen todos
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
}

// Intentar reenviar los fallidos una vez más

// Intentar reenviar los fallidos después de 3 minutos
if (!failedRecipients.isEmpty()) {
    System.out.println("Esperando 3 minutos antes de reintentar correos
fallidos...");

    new Thread(() -> {
        try {
            Thread.sleep(180_000); // Esperar 3 minutos (180,000 milisegundos)
            System.out.println("Reintentando correos fallidos...");

            ExecutorService retryExecutor =
Executors.newFixedThreadPool(Math.min(failedRecipients.size(), 10));
            List<String> stillFailed = Collections.synchronizedList(new
ArrayList<>());
            CountDownLatch retryLatch = new CountDownLatch(failedRecipients.size());

            for (String recipient : failedRecipients) {
                retryExecutor.submit(() -> {

```

```

        try {
            sendSingleEmail(recipient, emailServiceDTO);
        } catch (Exception e) {
            System.err.println("Reintento fallido para: " + recipient);
            stillFailed.add(recipient); // guardar definitivamente
        } finally {
            retryLatch.countDown();
        }
    });
}

retryExecutor.shutdown();
retryLatch.await();

if (!stillFailed.isEmpty()) {
    System.err.println("Los siguientes correos fallaron
permanentemente:");
    stillFailed.forEach(System.err::println);
}

System.out.println("Reintento de correos finalizado.");

} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    System.err.println("Reintento interrumpido.");
}

}).start(); // Lanzar hilo separado para esperar y reintentar
}

System.out.println("Proceso de envío de correos finalizado.");
}

// Método para enviar un solo correo
private void sendSingleEmail(String recipient, EmailServiceDTO dto) throws
Exception {
    var mimeType = javaMailSender.createMimeType(); // NUEVO: crear mensaje
    aquí
    MimeMessageHelper helper = new MimeMessageHelper(mimeType, true, "UTF-8");
    helper.setTo(recipient);
    helper.setSubject(dto.getSubject());

    Context context = new Context();
    context.setVariable("mensaje", dto.getBody());
    String contentHTML = templateEngine.process("prueba", context);

```

```

        System.out.println "[" + Thread.currentThread().getName() + "] Enviando correo
a: " + recipient);

        helper.setText(contentHTML, true);
        javaMailSender.send(mimeMessage); // usar el mensaje de este hilo
    }

public void sendEmailWithHTML(EmailServiceDTO emailServiceDTO) {
    List<String> recipients = emailServiceDTO.getRecipients();
    int totalRecipients = recipients.size();

    int threadCount = Math.min(totalRecipients, 20);
    ExecutorService executor = Executors.newFixedThreadPool(threadCount);

    List<String> failedRecipients = Collections.synchronizedList(new ArrayList<>());
    CountdownLatch latch = new CountdownLatch(totalRecipients);

    for (String recipient : recipients) {
        executor.submit(() -> {
            try {
                sendSingleEmailWithHTML(recipient, emailServiceDTO);
            } catch (Exception e) {
                System.err.println("Error al enviar a " + recipient + ": " +
e.getMessage());
                failedRecipients.add(recipient);
            } finally {
                latch.countDown();
            }
        });
    }

    executor.shutdown();

    try {
        latch.await();
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }

    // Reintento después de 3 minutos
    if (!failedRecipients.isEmpty()) {
        System.out.println("Esperando 3 minutos antes de reintentar correos fallidos
(HTML) ...");

        new Thread(() -> {

```

```

        try {
            Thread.sleep(30_000); // 3 minutos

            System.out.println("Reintentando correos fallidos (HTML)...");
            ExecutorService retryExecutor =
Executors.newFixedThreadPool(Math.min(failedRecipients.size(), 10));
            CountdownLatch retryLatch = new
CountdownLatch(failedRecipients.size());
            List<String> stillFailed = Collections.synchronizedList(new
ArrayList<>());

            for (String recipient : failedRecipients) {
                retryExecutor.submit(() -> {
                    try {
                        sendSingleEmailWithHTML(recipient, emailServiceDTO);
                    } catch (Exception e) {
                        System.err.println("Reintento fallido para: " +
recipient);

                        stillFailed.add(recipient);
                    } finally {
                        retryLatch.countDown();
                    }
                });
            }

            retryExecutor.shutdown();
            retryLatch.await();

            if (!stillFailed.isEmpty()) {
                System.err.println("Los siguientes correos HTML fallaron
permanentemente:");
                stillFailed.forEach(System.err::println);
            }

            System.out.println("Reintento de correos HTML finalizado.");
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.err.println("Reintento interrumpido.");
        }
    }).start();
}

System.out.println("Proceso de envío de correos HTML finalizado.");
}

```

```

        // Método para enviar un solo correo con HTML directamente
        private void sendSingleEmailWithHTML(String recipient, EmailServiceDTO dto)
throws Exception {
            var mimeType = javaMailSender.createMimeMessage();
            MimeMessageHelper helper = new MimeMessageHelper(mimeType, true,
"UTF-8");
            helper.setTo(recipient);
            helper.setSubject(dto.getSubject());

            // Enviar el HTML directamente
            String contentHTML = dto.getBody(); // Usar el HTML proporcionado en el body
            System.out.println("[ " + Thread.currentThread().getName() + " ] Enviando
correo a: " + recipient);

            helper.setText(contentHTML, true);
            javaMailSender.send(mimeType); // Enviar el correo
        }

        @Override
        public void sendEmail(EmailServiceDTO emailServiceDTO) {
            // TODO Auto-generated method stub
            throw new UnsupportedOperationException("Unimplemented method 'sendEmail'");
        }
    }
}

```

## EmailServiceConfig.java

```

package sis_ucb.online.megasistema.email_service.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ResourceLoader;
import org.springframework.core.io.DefaultResourceLoader;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.JavaMailSenderImpl;

import java.util.Properties;

```

```

@Configuration
@PropertySource("classpath:email.properties")
public class EmailServiceConfig {
    @Value("${email.username}")
    private String email;

    @Value("${email.password}")
    private String password;

    private Properties getMailProperties(){
        Properties properties = new Properties();
        properties.put("mail.smtp.auth", "true");
        properties.put("mail.smtp.starttls"+"enable", "true");
        properties.put("mail.smtp.host", "smtp.gmail.com");
        properties.put("mail.smtp.port", "587");
        return properties;
    }

    @Bean
    public JavaMailSender getJavaMailSender(){
        JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
        mailSender.setJavaMailProperties(getMailProperties());
        mailSender.setUsername(email);
        mailSender.setPassword(password);

        return mailSender;
    }

    @Bean

    public ResourceLoader resourceLoader(){
        return new DefaultResourceLoader();
    }
}

```

## EmailServiceRepository.java

```

package sis_uchb.online.megasistema.email_service.dao;

import sis_uchb.online.megasistema.email_service.dto.EmailServiceDTO;

public interface EmailServiceRepository {
    public void sendEmail(EmailServiceDTO emailServiceDTO) ;

    public void sendMailInvitation(EmailServiceDTO email);
}

```

```
}
```

## EmailServiceApplication.java

```
package sis_ucb.online.megasistema.email_service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@SpringBootApplication
public class EmailServiceApplication {

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

    @Bean
    public WebMvcConfigurer corsConfigurer () {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**").allowedOrigins("*")
                    .allowedMethods("GET", "POST", "PUT", "DELETE")
                    .allowedHeaders("*");
            }
        };
    }
}
```

## prueba.html

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css"
    />

    <meta charset="UTF-8" />
```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<style>
    body {
        font-family: "Arial", sans-serif;
        background-color: #f3f4f6;
        margin: 0;
        padding: 0;
        color: #333;
    }
    .container {
        width: 100%;
        max-width: 600px;
        margin: 0 auto;
        background-color: #ffffff;
        box-shadow: 10px 0 10px rgba(0, 0, 0, 0.1);
        border-radius: 8px;
        overflow: hidden;
        border: 1px solid #e2e8f0;
    }
    .header {
        background-image:
url(https://psicopedagogia-backend.serverbb.online/api/v1/public/file/download/pp2_
20240529-235622081.jpg);
        background-size: 100% auto;
        background-position: center;
        padding: 20px;
        width: 100%;
        text-align: center;
    }

    .header img {
        max-height: 70px;
        margin: 0 10px;
    }
    /****888
background-image:
url(https://psicopedagogia-backend.serverbb.online/api/v1/public/file/download/fond
obanner_20240530-000038245.jpg);
    */
    .header1 {
        background-size: 100% auto;
        background-position: center;
        height: 13rem;
        width: 100%;
        text-align: center;

```



```
}

.header2 {
  position: relative;
  display: flex;
  justify-content: center;
  align-items: center;
  flex-wrap: wrap;
  gap: 10px;
  padding: 10px;
  background-color: #f3f4f6;
  border-top: 1px solid #e2e8f0;
}

.header1 img {
  max-width: 180px;
}

.welcome {
  color: #00197c;
  text-align: center;
  margin-top: 20px;
  font-size: 1.5rem;
}

.welcome1 {
  color: #fefeff;
  text-align: center;
  margin-top: 20px;
  font-size: 2rem;
  box-shadow: 10px 10px 10px rgba(0, 0, 0, 0.1);
}

.message {
  color: #4a5568;
  padding: 20px;
  text-align: justify;
  font-size: 1rem;
}

.message span {
  display: block;
  font-size: 1.2rem;
  color: #e53e3e;
  margin: 20px 0;
}

.footer {
  text-align: center;
  padding: 20px;
  background-color: #f3f4f6;
}
```

```
border-top: 1px solid #e2e8f0;
}
.footer a {
  color: #0026c3;
  text-decoration: none;
  font-weight: bold;
}
.valores {
  background-color: #00c1b1;
  color: #fefeff;
  border: none;
  border-radius: 25px;
  padding: 10px 20px;
  margin: 10px;
  cursor: pointer;
  font-size: 0.8rem;
  width: 4rem;
  border: 4px solid #00c1b1;
}
.valores:hover {
  background-color: #fdfdfd;
  color: #00c1b1;
  width: 4.2rem;
}
.valores.a {
  background-color: #d2dd02;
  border: 4px solid #d2dd02;
}
.valores:hover.a {
  background-color: #fdfdfd;
  color: #d2dd02;
  width: 4.2rem;
}
.valores.b {
  background-color: #00da2c;
  border: 4px solid #00da2c;
}
.valores:hover.b {
  background-color: #fdfdfd;
  color: #00da2c;
  width: 4.2rem;
}
.valores.c {
  background-color: #f8ac07;
  border: 4px solid #f8ac07;
```

```
}

.valores: hover.c {
    background-color: #fdfdfd;
    color: #f8ac07;
    width: 4.2rem;
}

.header{
    background-color: #000f4a;
}

.header .imagen{
    width: 14rem;
    height: 14rem;

}

</style>
</head>
<body>
    <div class="container">
        <div class="header">
            
```

```

```

```
</div>
```

```
<p class="message">
```

```
Bienvenido al Sistema de Información de la Carrera de Ingeniería de Sistemas
```

```
<br /><br />
```

```
<span>Estimado estudiante:</span>
```

```
<br /><br />
```

```
Te damos la bienvenida al Sistema de Información de la Carrera de Ingeniería
de Sistemas. Este sistema está diseñado para brind
```

```
</p>
```

```
<h3 style="margin-left: 1rem; color: #00197c">
```

```
Atención de Dirección de Carrera
```

```
</h3>
```

```
<p class="message">
```

```
Av. 14 de Septiembre y Calle 2 de Obrajes \
```

```
<br /><br />
```

```
Departamento de Psicopedagogía Bloque N
```

```
<br /><br />
```

```
Horarios de atención: de lunes a viernes de 08:30 a 16:30<br /><br />
```

```
Contacto: 2782222 Int. 2872<br /><br />
```

```
WhatsApp: 73231313 - 77294940
```

```
</p>
```

```
<p class="message">Atentamente, Direccion de Carrera de Ing. de Sistemas</p>
```

```
</div>
<div class="footer">
  <p>
    Visita nuestro sitio web en
    <a href="https://sis-ucb.online"
      >https://sis-ucb.online</a>
  >
</p>
</div>
</body>
</html>
```

### 3.2 Desarrollo del microservicio funcional (15 pts)

El microservicio está implementado en Go, con soporte para gRPC y goroutines para el envío concurrente de correos. Las tareas principales:

- Recibir estructura **EmailRequest** con destinatarios, asunto y cuerpo.
- Lanzar hilos para enviar cada correo.
- Registrar los fallidos.

### 3.3 Consumo correcto del endpoint externo (10 pts)

El servidor principal (**main.go**) prepara el **EmailRequest** tras validar los correos en PocketBase. Luego, invoca al microservicio usando su cliente gRPC con la información preparada.

### 3.4 Procesamiento y transformación de los datos (10 pts)

Los datos se convierten de JSON (HTTP) a la estructura **EmailRequest** de gRPC. El microservicio convierte el cuerpo HTML si es necesario y maneja los estados de éxito o error.

El microservicio gRPC se basa en la comunicación eficiente entre servicios a través de un contrato bien definido utilizando archivos **.proto**. Este contrato especifica el servicio que proporciona el microservicio de envío de correos masivos, permitiendo la comunicación entre el backend y el microservicio mediante un protocolo binario optimizado.

El servicio gRPC se define mediante el archivo **.proto**, que incluye el método **SendBulkEmail**, el cual acepta un mensaje de tipo **EmailRequest** y retorna una respuesta de tipo **EmailResponse**. La estructura de estos mensajes está

completamente documentada, asegurando que las solicitudes y respuestas sean claras y fácilmente consumibles por el sistema.

- **EmailRequest:** Define los parámetros necesarios para enviar un correo, incluyendo el asunto, cuerpo del mensaje y una lista de destinatarios.
- **EmailResponse:** Retorna el resultado de la operación, indicando si el envío fue exitoso o si hubo algún error, junto con un mensaje adicional que explica el estado de la operación.

La estructura de este archivo `.proto` permite una comunicación eficiente y sin ambigüedades entre el cliente y el microservicio, facilitando la integración y mantenimiento a largo plazo.


Este enfoque asegura que el microservicio esté completamente documentado, y su implementación cumpla con los principios de claridad, escalabilidad y mantenibilidad, esenciales en la arquitectura de microservicios.

## 4. Pruebas y Documentación (15 pts)

### 4.1 Evidencias funcionales (5 pts)


Se incluyen:

- Video haciendo solicitud a `/api/custom/email`

 api-email.mov

[https://drive.google.com/file/d/1t-RGi3\\_2iotX9tYvpC2GLtYSUSLSCdFh/view?usp=sharing](https://drive.google.com/file/d/1t-RGi3_2iotX9tYvpC2GLtYSUSLSCdFh/view?usp=sharing)

- Video haciendo solicitud a `/api/custom/emails`


 api-emails.mov

<https://drive.google.com/file/d/1uzwg-DedUQUMPzgOZ0IJ730JGA0TubxG/view?usp=sharing>

### 4.2 Pruebas unitarias o manuales explicadas (5 pts)


Pruebas manuales:

- Solicitud con usuarios válidos: éxito

 api-email.mov


[https://drive.google.com/file/d/1t-RGi3\\_2iotX9tYvpC2GLtYSUSLSCdFh/view?usp=sharing](https://drive.google.com/file/d/1t-RGi3_2iotX9tYvpC2GLtYSUSLSCdFh/view?usp=sharing)

- Solicitud con correos inválidos: muestra los rechazados

 api-emails.mov


<https://drive.google.com/file/d/1uzwg-DedUQUMPzgOZ0IJ730JGA0TubxG/view?usp=sharing>

- Error al mandar ciertos correos: Volverlo a intentar luego de 3 minutos

 pruebaUnitaria.mov

<https://drive.google.com/file/d/175Blm-rUa6JTFYXt6RDDHQgJxIYOG5Ge/view?usp=sharing>

- Apagado del microservicio: error controlado

 microservicioApagado.mov

<https://drive.google.com/file/d/1UhGQ-2hAtTagaCA9LclGITjoTL1IVRY8/view?usp=sharing>

### **4.3 Documentación clara del microservicio (README.md o PDF) (5 pts)**

Se proporciona este PDF como documentación técnica. Incluye descripción general, tecnología usada, pruebas y flujo del sistema.



# mega-sistema-backend-email-service

## Servicio de Envío de Correos (Microservicio con Spring Boot)

Este repositorio contiene un **microservicio de envío de correos electrónicos** desarrollado con **Java y Spring Boot**. Forma parte de un sistema distribuido donde cada componente tiene una responsabilidad específica. Este servicio en particular se encarga del envío masivo de correos utilizando múltiples hilos para mejorar la eficiencia y tolerancia a fallos.

### ¿Qué es un Microservicio?

Un **microservicio** es una pequeña aplicación autónoma que realiza una única función dentro de un sistema más grande. En este caso, este microservicio gestiona únicamente el **envío de correos electrónicos**. Su diseño permite que sea escalable, mantenible y que pueda ser desplegado de forma independiente.

### Integración con el API Gateway

Este microservicio **no recibe las peticiones directamente del frontend**, sino a través de un **API Gateway** (`main.go` en este caso), el cual actúa como punto de entrada único. El API Gateway enruta la solicitud al microservicio adecuado según el endpoint.

### Funcionalidad Principal

Este servicio permite enviar correos electrónicos de forma paralela (multi-threading), mejorando la velocidad y eficiencia del envío masivo. En caso de que algún correo falle al enviarse, se intenta nuevamente **después de 3 minutos** automáticamente.

## 🔧 Endpoints Disponibles

### 1. POST /api/v1/auth/send-email-invitation

Envía un correo de invitación con un mensaje HTML predeterminado.

### 2. POST /api/v1/auth/send-email-html

Envía un correo de con un mensaje HTML personalizado que manda el client desde la petición.

#### 👤 Ejemplo de solicitud

```
POST http://localhost:8017/api/v1/auth/send-email-invitation
Content-Type: application/json
```

```
{
  "recipients": [
    "daniel.aldazosa@gmail.com",
    "daniel.alsa@gmail.com",
    "jhessikazarate@gmail.com",
    "tania.perez.d@ucb.edu.bo"
  ],
  "subject": "NOTASSSS",
  "body": ""
}
```

```
POST http://localhost:8017/api/v1/auth/send-email-html
Content-Type: application/json
```

```
{
  "recipients": [
    "daniel.aldazosa@gmail.com",
    "daniel.alsa@gmail.com",
    "jhessikazarate@gmail.com",
    "tania.perez.d@ucb.edu.bo"
  ],
  "subject": "NOTASSSS",
  "body": "<!DOCTYPE html>\n<html lang=\"es\">\n  <head>\n    <meta charset=\"UTF-8\">\n  "
}
```

## 5. Presentación Final (15 pts)

### 5.1 Explicación técnica del microservicio y su integración (10 pts)

El `main.go` verifica que cada correo pertenezca a un usuario registrado (de la carrera de Sistemas). El microservicio realiza:


- Envío concurrente usando goroutines
- Manejo de errores y almacenamiento temporal de fallos
- Reintentos cada 3 minutos mediante un timer

Este diseño desacopla la validación de usuarios del envío masivo de correos, haciendo el sistema escalable.


## 5.2 Demostración funcional clara (5 pts)

Se realiza la demostración desde Postman + logs en consola. Se muestran:

- Solicitudes exitosas


 api-email.mov

[https://drive.google.com/file/d/1t-RGi3\\_2iotX9tYvpC2GLtYSUSLSCdFh/view?usp=sharing](https://drive.google.com/file/d/1t-RGi3_2iotX9tYvpC2GLtYSUSLSCdFh/view?usp=sharing)


 api-emails.mov

<https://drive.google.com/file/d/1uzwg-DedUQUMPzgOZ0IJ730JGA0TubxG/view?usp=sharing>

- Fallos controlados con reintentos

 pruebaUnitaria.mov

<https://drive.google.com/file/d/175Blm-rUa6JTFYXt6RDDHQgJxIYOG5Ge/view?usp=sharing>

 microservicioApagado.mov

<https://drive.google.com/file/d/1UhGQ-2hAtTagaCA9LclGITjoTL1IVRY8/view?usp=sharing>

## Anexo: Fragmentos Clave del Código

### Validación de Correos en **main.go**

```
// Obtener todos los emails de la colección 'usuarios'
userRecords, err := e.App.FindAllRecords("usuarios");
if err != nil {
    return e.InternalServerError("Error al obtener los correos válidos",
err)
}

validEmails := map[string]bool{}
for _, record := range userRecords {
    email := record.Get("email").(string)
    validEmails[email] = true
}
```

```
}
```

## Envío concurrente en el microservicio (resumen)

```
public void sendMailInvitation(EmailServiceDTO emailServiceDTO) {
    List<String> recipients = emailServiceDTO.getRecipients();
    int totalRecipients = recipients.size();

    // Máximo 20 hilos para evitar sobrecarga
    int threadCount = Math.min(totalRecipients, 20);
    ExecutorService executor = Executors.newFixedThreadPool(threadCount);

    // Lista de correos que fallaron (segura para hilos)
    List<String> failedRecipients = Collections.synchronizedList(new ArrayList<>());

    CountdownLatch latch = new CountdownLatch(totalRecipients);

    for (String recipient : recipients) {
        executor.submit(() -> {
            try {
                sendSingleEmail(recipient, emailServiceDTO);
            } catch (Exception e) {
                System.err.println("Error al enviar a " + recipient + ": " +
e.getMessage());
                failedRecipients.add(recipient);
            } finally {
                latch.countDown();
            }
        });
    }

    executor.shutdown();

    try {
        latch.await(); // Esperar que terminen todos
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    }
}
```

```
// Método para enviar un solo correo
private void sendSingleEmail(String recipient, EmailServiceDTO dto) throws
Exception {
    var mimeMessage = javaMailSender.createMimeMessage(); // NUEVO: crear mensaje
aquí
```

```
MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, true, "UTF-8");
helper.setTo(recipient);
helper.setSubject(dto.getSubject());
```

```
Context context = new Context();
context.setVariable("mensaje", dto.getBody());
String contentHTML = templateEngine.process("prueba", context);
System.out.println "[" + Thread.currentThread().getName() + "] Enviando correo a: " + recipient);
```

```
helper.setText(contentHTML, true);
javaMailSender.send(mimeMessage); // usar el mensaje de este hilo
}
```

## Reintento cada 3 minutos

```
// Intentar reenviar los fallidos después de 3 minutos
if (!failedRecipients.isEmpty()) {
    System.out.println("Esperando 3 minutos antes de reintentar correos fallidos...");

    new Thread(() -> {
        try {
            Thread.sleep(180_000); // Esperar 3 minutos (180,000 milisegundos)
            System.out.println("Reintentando correos fallidos...");

            ExecutorService retryExecutor =
                Executors.newFixedThreadPool(Math.min(failedRecipients.size(), 10));
            List<String> stillFailed = Collections.synchronizedList(new
                ArrayList<>());

            CountdownLatch retryLatch = new CountdownLatch(failedRecipients.size());

            for (String recipient : failedRecipients) {
                retryExecutor.submit(() -> {
                    try {
                        sendSingleEmail(recipient, emailServiceDTO);
                    } catch (Exception e) {
                        System.err.println("Reintento fallido para: " + recipient);
                        stillFailed.add(recipient); // guardar definitivamente
                    } finally {
                        retryLatch.countDown();
                    }
                });
            }

            retryExecutor.shutdown();
            retryLatch.await();
        }
    });
}
```

```
        if (!stillFailed.isEmpty()) {  
            System.err.println("Los siguientes correos fallaron  
permanentemente:");  
            stillFailed.forEach(System.err::println);  
        }  
  
        System.out.println("Reintento de correos finalizado.");  
  
    } catch (InterruptedException e) {  
        Thread.currentThread().interrupt();  
        System.err.println("Reintento interrumpido.");  
    }  
}).start(); // Lanzar hilo separado para esperar y reintentar  
}
```

## Conclusión

El microservicio implementado cumple satisfactoriamente con los objetivos planteados, al encargarse del envío masivo y concurrente de correos electrónicos a usuarios verificados del sistema. Su diseño modular, basado en gRPC, permite un alto rendimiento y una clara separación de responsabilidades entre el backend principal y el servicio de mensajería. Además, la capacidad de reintento automático ante fallos fortalece la confiabilidad del sistema. Esta arquitectura no solo mejora la escalabilidad y mantenibilidad del proyecto integrador, sino que también sienta las bases para futuras extensiones, como la integración de notificaciones en tiempo real o servicios multiplataforma. En conjunto, el microservicio no solo aporta una solución técnica robusta, sino que también mejora significativamente la experiencia de usuario final..