Mail-service

1. Seeder (EmailTemplateSeeder)

Qué hace:

- Al iniciar la aplicación, crea y guarda las plantillas de email predeterminadas (bienvenida, recuperación, notificación) usando el EmailTemplateService.
- Solo se ejecuta una vez al arranque.
- No envía correos, solo inicializa los textos base.

Por qué existe:

- Permite que el sistema tenga plantillas listas para usar desde el inicio, sin depender de una base de datos externa.

---

2. Servicio de Plantillas (EmailTemplateService)

Qué hace:

- Administra las plantillas de email: guardar, buscar por ID o tipo, activar/desactivar, procesar (reemplazar variables), obtener asunto, etc.
- No envía correos, solo gestiona el contenido y los datos de las plantillas.

Por qué existe:

- Centraliza la lógica de plantillas, permitiendo que cualquier parte del sistema pueda obtener y procesar plantillas de forma consistente.
- Facilita cambios en el formato o contenido de los emails sin tocar la lógica de envío.

---

3. Estrategias (EmailStrategy y subclases)

Qué hacen:

- Cada estrategia representa la lógica para armar y enviar un tipo específico de correo (bienvenida, recuperación, notificación).

- Usan el EmailTemplateService para obtener la plantilla adecuada y rellenarla con los datos del usuario.
- Configuran el destinatario, asunto y cuerpo del mensaje.

Por qué existen:

- Permiten separar la lógica de cada tipo de correo, facilitando la extensión (agregar nuevos tipos) y el mantenimiento.
- Si cambias la lógica de un solo tipo de correo, no afectas a los demás.

---

4. Servicio de Envío (EmailService)

Qué hace:

- Es el punto de entrada para enviar correos desde los controladores.
- Recibe la petición del controlador y decide qué estrategia usar según el tipo de correo.
- Llama a la estrategia correspondiente, que arma el mensaje y lo envía usando JavaMailSender.
- Tiene métodos para enviar correos individuales (sendWelcomeEmail, sendRecoveryEmail, etc.) y masivos (sendBulkEmail).

Por qué existe:

- Centraliza la lógica de envío, desacoplando los controladores de los detalles técnicos.
- Permite manejar logs, errores y lógica común de envío en un solo lugar.

---

5. Controladores (`WelcomeEmailController` , etc.)

Qué hacen:

- Reciben las peticiones HTTP (por ejemplo, desde el frontend).
- Validan los datos y llaman al método correspondiente de EmailService.
- No contienen lógica de armado ni envío de correos, solo orquestan la petición.

Por qué existe:

- Mantienen la API limpia y sencilla, delegando la lógica compleja a los servicios.

FLUJO COMPLETO

1. **Inicio de la app:**
   - El seeder carga las plantillas en memoria usando EmailTemplateService.
2. **Petición de envío de correo:**
   - Un usuario hace una petición (por ejemplo, registro).
   - El controlador recibe la petición y llama a EmailService.sendWelcomeEmail().
3. **Selección de estrategia:**
   - EmailService crea una instancia de la estrategia adecuada (WelcomeEmailStrategy).
   - Llama a sendEmail() pasando la estrategia y los datos del usuario.
4. **Armado y envío:**
   - La estrategia toma la plantilla desde EmailTemplateService, la rellena con los datos y arma el mensaje.
   - Usa JavaMailSender para enviar el correo al destinatario.
5. **Respuesta:**
   - El controlador responde al usuario según el resultado.

¿Por qué cada parte tiene solo esa función?

- **Seeder:** Solo inicializa datos, no envía ni procesa correos.
- **EmailTemplateService:** Solo gestiona plantillas, no sabe nada de envío.
- **Estrategias:** Solo arman el mensaje para un tipo de correo, no gestionan plantillas ni envían directamente.
- **EmailService:** Solo orquesta el envío, no sabe de plantillas ni de HTTP.

- **Controladores:** Solo reciben peticiones y delegan, no arman ni envían correos.

Ventajas de este diseño:

- Fácil de mantener y escalar.
- Cada parte es testeable por separado.
- Puedes cambiar la lógica de plantillas, envío o controladores sin afectar el resto.
- Permite agregar nuevos tipos de correos fácilmente.