## Envío de Correos

Desde la versión 4.3, symfony gestiona el envío de correos a través del componente Mailer.

Para instalarlo con Flex:

composer require symfony/mailer

Antes de la versión 4.3 se gestionaban mediante el bundle **SwiftMailerBundle**.

composer require mailer

# Configuración

## Mailer

El componente Mailer permite utilizar envío de correos configurando el transporte con la variable MAILER_DSN en el fichero .env:

```
MAILER_DSN=smtp://user:pass@smtp.example.com
```

Pero si vamos a utilizar alguno de los siguientes proveedores, conviene instalar el provider asociado, que viene ya preconfigurado:

- Amazon SES: composer require symfony/amazon-mailer
- Gmail composer: require symfony/google-mailer
- MailChimp: composer require symfony/mailchimp-mailer
- Mailgun: composer require symfony/mailgun-mailer
- Postmark: composer require symfony/postmark-mailer
- SendGrid: composer require symfony/sendgrid-mailer

Se añadirá la configuración necesaria en el fichero .env y solamente tendremos que indicar los datos de nuestra cuenta (usuario/contraseña para gmail, key para sendgrid, etc)

## SwiftMailer

- url

Permite poner la configuración completa utilizando una URL de tipo DSN:

```
smtp://user:pass@host:port/?timeout=60&encryption=ssl&auth_mode=login&...
```

Normalmente estará configurada en una variable de entorno.

```
swiftmailer:
    url: '%env(MAILER_URL)%'
    spool: { type: 'memory' }
```

```
MAILER_URL=smtp://user:pass@host:port/?timeout=60&encryption=ssl&auth_mod
e=login
```

La lista de opciones de configuración es la siguiente

- transport

Los posibles valores son: - smtp - gmail - mail (obsoleto desde la versión 5.4.5) - sendmail - null (es lo mismo que poner disable_delivery a true)

- username
- password
- command

Comando que será ejecutado por sendmail. Por defecto */usr/sbin/sendmail -bs*

- host
- port
- timeout

Timeout en segundos cuando se utiliza smtp.

- source_ip

Únicamente válido en smpt

- local_domain

El nombre de dominio para utilizar en el comando *HELO*.

- encryption

Los posibles valores son *tls*, *ssl* o *null*.

- auth_mode

Autenticación a utilizar para smtp. Los posibles valores son:

```
- plain
- login
- cram-md5
- null
```

- spool

Para la configuración de gestión de envío de mensajes. Tiene dos subopciones:

```
- type: que puede valer *memory* o *file*
- path: directorio de los archivos si el tipo de spool es file.
```

- sender_address

Si se establece esta opción, todos los mensajes tendrán esta dirección como la "dirección de respuesta".

- antiflood

- threshold: número de emails enviados antes resetear el transporte.
- sleep: número de segundos a esperar durante el reseteo del transporte.
- delivery_addresses

Si se establece un valor, TODOS los correos se enviarán a estas direcciones en lugar de enviarse a las direcciones originales. Es una opción útil durante el desarrollo.

- delivery_whitelist

Si se establece, los emails con direcciones incluidas en esta lista, serán enviados a dichas direcciones además de ser enviados a las direcciones indicadas en delivery_addresses.

- disable_delivery

Si se establece a *true*, no se enviarán correos.

- logging

Si se establece a *true*, el data collector asociado recogerá información de Swift Mailer y la mostrará en el profiler. Por defecto tiene el valor *%kernel.debug%*.

## Envío de emails

### Mailer

El componente Mailer trabjaja con la clase **Email** para construir un mensaje/correo y la clase/servicio **MailerInterface** para enviar los mensajes/correos.

```
// src/Controller/MailerController.php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Mailer\MailerInterface;
use Symfony\Component\Mime\Email;

class MailerController extends AbstractController
{
    /**
     * @Route("/email")
     */
    public function sendEmail(MailerInterface $mailer)
    {
        $email = (new Email())
            ->from('hello@example.com')
            ->to('you@example.com')
            ->cc('cc@example.com')
            ->bcc('bcc@example.com')
            ->replyTo('fabien@example.com')
            ->priority(Email::PRIORITY_HIGH)
            ->subject('Time for Symfony Mailer!')
            ->text('Sending emails is fun again!')
            ->html('<p>See Twig integration for better HTML
integration!</p>');

        /** @var Symfony\Component\Mailer\SentMessage $sentEmail */
        $sentEmail = $mailer->send($email);
        // $messageId = $sentEmail->getMessageId();

        // ...
    }
}
```

Más información en: https://symfony.com/doc/current/mailer.html#creating-sending-messages

## SwiftMailer

La librería de Swift Mailer trabjaja con la clases **Swift_Message** para construir un mensaje/correo y la clase/servicio **Swift_Mailer** para enviar los mensajes/correos.

```
public function index($name, \Swift_Mailer $mailer)
{
    $message = (new \Swift_Message('Hello Email'))
        ->setFrom('send@example.com')
        ->setTo('recipient@example.com')
        ->setBody(
            $this->renderView(
                // templates/emails/registration.html.twig
                'emails/registration.html.twig',
                array('name' => $name)
            ),
            'text/html'
        )
        /*
         * Se puede incluir una versión en texto plano del mensaje
        ->addPart(
            $this->renderView(
                'emails/registration.txt.twig',
                array('name' => $name)
            ),
            'text/plain'
        )
        */
    ;

    $mailer->send($message);

    return $this->render(...);
}
```

Para mantener los elementos desacoplados, el cuerpo del mensaje se puede programar en una plantilla de Tiwg y renderizarlo con la función **renderView()**.

El fichero de Twig registration.html.twig podría ser algo parecido a esto:

```
{# templates/emails/registration.html.twig #}
<h3>¡Enhorabuena!</h3>

Hola {{ name }}. Te has registrado con éxito.

Para hacer login, ve a: <a href="{{ url('login') }}">Login</a>.

¡Gracias!

<img src="{{ absolute_url(asset('images/logo.png')) }}">
```

La clase Swift_Message soporta más opciones, como por ejemplo, adjuntar archivos. El resumen de los métodos de dicha clase es el se puede ver a continuación:

```
$message = (new Swift_Message())

  // Asunto del mensaje
  ->setSubject('Your subject')

  // Dirección o direcciones de correo para el From
  ->setFrom(['john@doe.com' => 'John Doe'])

  // Direcciones de correo para el To, Cc y Bcc (setTo/setCc/setBcc)
  ->setTo(['receiver@domain.org', 'other@domain.org' => 'A name'])

  // Cuerpo del mensaje
  ->setBody('Here is the message itself')

  // Cuerpo alternativo del mensaje
  ->addPart('<q>Here is the message itself</q>', 'text/html')

  // Archivos adjuntos
  ->attach(Swift_Attachment::fromPath('my-document.pdf'))
  ;
```

Más información en: https://swiftmailer.symfony.com/docs/messages.html

## Cómo trabajar con emails durante el desarrollo

### Deshabilitar envío de correos

- Con Mailer:

```
# config/packages/dev/mailer.yaml
framework:
    mailer:
        dsn: 'null://null'
```

- Con SwiftMailer:

```
# config/packages/test/swiftmailer.yaml
swiftmailer:
    disable_delivery: true
```

### Enviar todos los correos a la cuenta del desarrollador

Podemos configurar el mailer para que envíe los correos a unas direcciones determinadas en lugar de a sus destinatarios originales.

De esta forma, durante el desarrollo, no bombardearemos a los usuarios con correos de testeo. Tampoco necesitaremos molestarles para preguntarles si les han llegado los correos.

- Con Mailer:

Se realiza configurando un listener pre-programado.

```
# config/services_dev.yaml
services:
    mailer.dev.set_recipients:
        class: Symfony\Component\Mailer\EventListener\EnvelopeListener
        tags: ['kernel.event_subscriber']
        arguments:
            $sender: null
            $recipients: ['youremail@example.com']
```

- Con SwiftMailer:

Se realiza en la configuración del bundle.

```
# config/packages/test/swiftmailer.yaml
swiftmailer:
    delivery_addresses: ['dev@example.com']
```

## Enviar todos los correos a la cuenta del desarrollador con excepciones

- Con SwiftMailer:

La opción **delivery_whitelist** permite configurar expresiones regulares. Si el destinatario de un correo cumple alguna de las expresiones regulares, se le entregará el correo, además de entregarse también a los correos configurados en *delivery_addresses*.

```
# config/packages/dev/swiftmailer.yaml
swiftmailer:
    delivery_addresses: ['dev@example.com']
    delivery_whitelist:
       # all email addresses matching these regexes will be delivered
       # like normal, as well as being sent to dev@example.com
       - '/@specialdomain\.com$/'
       - '/^admin@mydomain\.com$/'
```

La opción *delivery_whitelist* solamente funciona si está definida la opción *delivery_addresses*.

## Ver correos en la barra de depuración y en el profiler

En la barra de depuración aparece un número indicando cuántos correos se han enviado. En el profiler se muestran los detalles de dichos correos.

Si enviamos un email y acto seguido redirigimos a otra página, en la barra de depuración no se mostrará el email enviado.

Tenemos la opción de buscar el profiler de la petición anterior, en el buscador del profiler, pero también podemos establecer la opción **intercept_redirects** a *true* en el entorno de *dev*. Esto provoca que la redirección se pare y se pueda ver en el profiler la información de los emails enviados.

```
# config/packages/dev/web_profiler.yaml
web_profiler:
    intercept_redirects: true
```

## Cómo testear si un email ha sido enviado

Se puede testear que un email ha sido enviado y comprobar cualquiera de sus propiedades (asunto, cuerpo, destinatario...) a través del profiler.

Veámoslo con un ejemplo.

Tenemos un controlador muy sencillo que simplemente envía un email.

```
public function sendEmail($name, \Swift_Mailer $mailer)
{
    $message = (new \Swift_Message('Hello Email'))
        ->setFrom('send@example.com')
        ->setTo('recipient@example.com')
        ->setBody('You should see me from the profiler!')
    ;

    $mailer->send($message);

    return $this->render(...);
}
```

Lo podríamos testear de la siguiente forma:

```
// tests/Controller/MailControllerTest.php
namespace App\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class MailControllerTest extends WebTestCase
{
    public function testMailIsSentAndContentIsOk()
    {
        $client = static::createClient();

        // Habilita el profiler para la petición que vamos a realizar.
        $client->enableProfiler();

        $crawler = $client->request('POST', '/path/to/above/action');

        $mailCollector = $client->getProfile()-
>getCollector('swiftmailer');

        // Comprueba si el email fue enviado
        $this->assertSame(1, $mailCollector->getMessageCount());

        $collectedMessages = $mailCollector->getMessages();
        $message = $collectedMessages[0];
```

```
        // Más comprobaciones
        $this->assertInstanceOf('Swift_Message', $message);
        $this->assertSame('Hello Email', $message->getSubject());
        $this->assertSame('send@example.com', key($message->getFrom()));
        $this->assertSame('recipient@example.com', key($message-
>getTo()));
        $this->assertSame(
            'You should see me from the profiler!',
            $message->getBody()
        );
    }
}
```

La clave está en acceder al data collector del mailer (Mailer o SwiftMailer) y revisar la información recogida.

- Con Mailer: $mailCollector = $client->getProfile()->getCollector('mailer');
- Con SwiftMailer: $mailCollector = $client->getProfile()->getCollector('swiftmailer');

## Notas

Al testear correos debemos prestar atención a dos cosas:

- Que el profiler esté habilitado
- No seguir la redirección en caso de que hubiera alguna

## Cómo configurar varios mailers

Configurar varios mailers es muy sencillo. Basta poner un nombre a cada mailer, y a partir del nombre la configuración de cada uno.

- Con Mailer:

```
# config/packages/mailer.yaml
framework:
    mailer:
        transports:
            main: '%env(MAILER_DSN)%'
            important: '%env(MAILER_DSN_IMPORTANT)%'
```

Por defecto se utilizará el primer mailer de la lista.

Si se quiere utilizar otro de los mailers configurados se debe indicar con el Header *X-Transport*:

```
// Utiliza el mailer por defecto
$mailer->send($email);

// Utiliza el mailer 'important'
$email->getHeaders()->addTextHeader('X-Transport', 'important');
$mailer->send($email);
```

- Con SwiftMailer:

```
swiftmailer:
    default_mailer: primer_mailer
    mailers:
        primer_mailer:
            # ...
        segundo_mailer:
            # ...
```

Por defecto se utilizará el mailer indicado en *default_mailer*:

Cada mailer es registrado automáticamente como un servicio con estos IDs:

```
// ...

// devuelve el primer_mailer
$container->get('swiftmailer.mailer.primer_mailer');

// devuelve el mailer por defecto (primer_mailer)
$container->get('swiftmailer.mailer');
```

```
// devuelve el segundo_mailer
$container->get('swiftmailer.mailer.segundo_mailer');
```

Si estamos utilizando *autowiring*, el servicio que se inyectará será siempre el mailer que se haya configurado como *default_mailer*. Si necesitamos inyectar otro de los mailers, debemos configurarlo manualmente:

```
# config/services.yaml
services:
    _defaults:
        bind:
            # inyecta segundo_mailer en los argumentos de constructor
tipados con \Swift_Mailer
            \Swift_Mailer: '@swiftmailer.mailer.segundo_mailer'
            # inyecta segundo_mailer en los argumentos de servicios cuyo
nombre sea $specialMailer
            $specialMailer: '@swiftmailer.mailer.segundo_mailer'

    App\Some\Service:
        # inyecta segundo_mailer en el argumento $differentMailer del
constructor del servicio App\Some\Service
        $differentMailer: '@swiftmailer.mailer.segundo_mailer'

    # ...
```

## Envío asíncrono de mails

- Con Mailer:

El envío asincrono con Mailer se realiza mediante el componente MESSENGER. Un componente nuevo incluido en las últimas versiones de Symfony.

Se instala mediante el siguiente comando:

composer require messenger

```
# config/packages/messenger.yaml
framework:
    messenger:
        transports:
            async: "%env(MESSENGER_TRANSPORT_DSN)%"
```

Y se envían con el siguiente comando

php bin/console messenger:consume async

en el que 'async' es el nombre del transport configurado para utiliar MESSENGER.

Más información sobre MESSENGER:
https://symfony.com/doc/current/components/messenger.html

- Con SwiftMailer:

Con SwiftMailer el envío asíncrono se configura indicando en la propiedad *spool* el valor *file*.

```
# config/packages/swiftmailer.yaml
swiftmailer:
    # ...
    spool:
        type: file
        path: /path/to/spooldir
```

Y se envían con el siguiente comando:

APP_ENV=prod php bin/console swiftmailer:spool:send

Se puede configurar límite de mensajes

APP_ENV=prod php bin/console swiftmailer:spool:send --message-limit=10

Y se puede configurar límite de tiempo (en segundos)

APP_ENV=prod php bin/console swiftmailer:spool:send --time-limit=10