

# **Le composant Mailer**

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Installation du composant Mailer et configuration de l'application</b>	<b>3</b>
A. Installation du composant Mailer et configuration de l'application .....	3
B. Exercice : Quiz .....	5
<b>III. Utilisation avancée et affichage d'email</b>	<b>6</b>
A. TemplatedEmail .....	6
B. Ajouter un fichier ou d'une image .....	9
C. Affichage d'e-mails .....	9
D. Exercice : Quiz .....	12
<b>IV. Essentiel</b>	<b>12</b>
<b>V. Auto-évaluation</b>	<b>13</b>
A. Exercice .....	13
B. Test .....	13
<b>Solutions des exercices</b>	<b>14</b>

## I. Contexte

**Durée :** 1 h

**Environnement de travail :** un ordinateur connecté à Internet

**Prérequis :**

Notion de PHP

Notion de POO

### Contexte

Aujourd'hui, beaucoup de sites internet ont intégré la possibilité de renvoyer aux utilisateurs des réponses automatiques à un formulaire de contact. De la même façon, beaucoup de sociétés ont besoin d'envoyer des e-mails, ne serait-ce que pour valider une inscription ou modifier un mot de passe, de manière automatique. C'est devenu un élément indispensable dans les applications web modernes. Cependant, l'envoi d'e-mails nécessite l'utilisation d'un serveur dédié, appelé serveur SMTP. C'est là qu'intervient le composant Mailer de Symfony, qui facilite grandement cette tâche.

Le composant Mailer de Symfony est un outil performant et flexible qui permet de gérer et d'envoyer des e-mails de manière simple. Avec ce composant, vous pouvez facilement créer et envoyer des courriels, y compris des templates HTML, des pièces jointes, des adresses de réponse, des en-têtes et des signatures. Vous pouvez également configurer facilement les paramètres d'envoi d'e-mail tels que les serveurs SMTP, les adresses d'envoi et de réponse.

Dans ce cours, nous expliquerons comment fonctionne le composant Mailer et comment l'intégrer dans votre application Symfony 5.4.

## II. Installation du composant Mailer et configuration de l'application

### A. Installation du composant Mailer et configuration de l'application

#### Installation Symfony/Mailer

Pour commencer, il faut installer le composant Mailer à l'aide de composer. Pour cela, tapez la ligne de commande ci-dessous à l'aide du terminal dans le dossier de votre application :

```
1 $ composer require symfony/mailer
```

#### Paramétrage du serveur SMTP

Maintenant, il va falloir configurer le serveur SMTP. Un serveur SMTP (Simple Mail Transfer Protocol) est un serveur informatique qui gère l'envoi et la réception des e-mails. Il s'agit du protocole standard utilisé pour envoyer des e-mails via internet. Ainsi, lorsque vous enverrez un e-mail, il sera transféré de votre application de messagerie vers le serveur SMTP qui gère le domaine de l'adresse de destination.

Pour le configurer avec Symfony, c'est assez simple. Cela se passe un peu comme avec la configuration de votre application vers votre base de données. Il faut dans votre fichier `".env"`, à la racine de votre projet, configurer la variable d'environnement `MAILER_DSN`.

En fait, de par l'installation du package mailer, un nouveau fichier YAML qui se nomme `mailer.yaml` a été créé. Vous pouvez trouver le fichier `mailer.yaml` dans le répertoire `config/packages` de votre application. Ce fichier indique que mailer a besoin d'un DSN qu'il récupère par la variable d'environnement `MAILER_DSN`.

```
1 # .env
2 MAILER_DSN=smtp://user:pass@smtp.example.com:port
```

Bien sûr, dans la variable d'environnement MAILER\_DSN, il vous faudra rentrer les paramètres de votre serveur SMTP. Voici deux manières différentes de procéder, soit avec Mailtrap, soit directement avec une boîte mail :

Mailtrap : beaucoup de développeurs utilisent en mode développement un service en ligne comme Mailtrap. Celui-ci permet aux développeurs d'envoyer et de recevoir des e-mails de test sans envoyer des e-mails avec de véritables destinataires.

Pour utiliser mailtrap c'est très simple vous vous rendez sur mailtrap.io et vous ouvrez un compte. Une fois votre compte créé vous aurez une boîte de réception virtuelle, et dans les réglages SMTP, choisissez votre langage de programmation. Donc pour nous ce sera Symfony 5+ puis copier et coller votre nouvelle variable d'environnement.

#### Exemple

```
1 MAILER_DSN=smtp://code_de_votre_boite@sandbox.smtp.mailtrap.io:2525?
  encryption=tls&auth_mode=login.
```

Code\_de\_votre\_boite étant le code qui vous sera attribué.

Boîte Gmail ou autre : récupérer les paramètres SMTP de votre boîte et rentrez-les dans la variable d'environnement MAILER\_DSN. Sur le site officiel de Symfony<sup>1</sup>, vous pourrez trouver de l'aide pour configurer selon le service que vous utilisez votre DSN.

### Découverte du composant Mailer et ses méthodes

Voici comment vous pouvez utiliser mailer dans un contrôleur. (Il est conseillé de plutôt le gérer dans un service que vous pourriez appeler MailerService par exemple, mais par souci de clarté nous utiliserons Mailer directement dans le contrôleur).

Pour faire fonctionner Mailer, il faut importer dans le contrôleur MailerInterface et Mime\Email ainsi qu'injecter MailerInterface dans la méthode du contrôleur.

#### Exemple

```
1 <?php
2
3 namespace App\Controller;
4
5 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
6 use Symfony\Component\HttpFoundation\Request;
7 use Symfony\Component\HttpFoundation\Response;
8 use Symfony\Component\Mailer\MailerInterface;
9 use Symfony\Component\Mime\Email;
10
11 class MyController extends AbstractController
12 {
13     public function sendEmailAction(Request $request, MailerInterface $mailer): Response
14     {
15
16         $email = (new Email())
17             ->from('didierdeschamps@example.com')
18             ->to('zinedine@example.com')
19             ->subject('Coupe du monde')
20             ->text('vous êtes invité à la prochaine coupe du monde');
21
22         $mailer->send($email);
23     }
```

<sup>1</sup> <https://symfony.com/doc/current/mailer.html>

```

24
25 return $this->redirectToRoute('homepage');
26 }
27 }

```

Dans l'exemple ci-dessus, on constate que l'objet Email possède plusieurs méthodes :

- La méthode `from()` : spécifie l'adresse e-mail de l'expéditeur de l'e-mail. En option on peut ajouter le nom de l'expéditeur.  
Par exemple : `->from('didierdeschamps@exemple.com', 'Didier Deschamps')`
- La méthode `to()` : spécifie l'adresse e-mail du destinataire. En option on peut ajouter le nom du destinataire. Alternative équivalente `addTo()`.  
Par exemple : `->to('zinedine@exemple.com', 'Zinedine Zidane')`
- La méthode `subject()` : spécifie le sujet de l'e-mail.
- La méthode `text()` : spécifie le contenu texte de l'e-mail.
- La méthode `end()` : permet d'envoyer l'e-mail avec tous les éléments récoltés des différentes méthodes utilisées.

Notez que pour le FROM et TO, vous pouvez récupérer cette variable depuis le fichier de config `service.yaml` et `.env` si vous avez une adresse constante, qui ne changera jamais (type formulaire de contact).

L'objet Email est la classe principale utilisée pour créer des e-mails avec Mailer de Symfony. Voici d'autres méthodes fréquemment utilisées disponibles sur cet objet :

- `html()` : spécifie le contenu HTML de l'e-mail. Cela permet de créer des e-mails plus riches et plus visuellement attrayants en utilisant le balisage HTML. Vous l'utiliserez souvent en alternative à la méthode `text()`.
- `addCc()` ou `cc()` sont utilisés pour ajouter des destinataires en copie (CC) à un courrier électronique.
- `addBcc()` ou `bcc()` sont utilisés pour ajouter des destinataires en copie cachée (BCC) à un courrier électronique. Les destinataires en copie cachée recevront également le courrier électronique, mais leur présence ne sera pas visible par les autres destinataires.
- `replyTo()` : spécifie l'adresse e-mail et le nom de la personne à qui les réponses à l'e-mail doivent être envoyées.
- `priority()` : spécifie le niveau d'importance de l'e-mail envoyé.  
Exemple : `->priority(Email::priority_high)`.
- `getHeaders()` : permet de récupérer les en-têtes d'e-mail. On peut y ajouter des fonctions.  
Exemple : `->getHeaders()`
- `->addTextHeader('X-Auto_Response-Suppress', 'OOF, DR, RN, NRN, AutoReply')`  
`addTextHeader()` étant une méthode qui permet d'ajouter un message en en-tête afin que les serveurs ne renvoient pas de réponse automatique.
- `charset()` : permet de définir le jeu de caractères à utiliser pour le contenu de l'e-mail.  
Exemple : `->charset('UTF-8')`.

Il existe également d'autres méthodes pour ajouter des en-têtes personnalisés, des pièces jointes à partir d'un flux de données, ainsi que pour modifier les options de l'e-mail comme la date d'envoi.

## B. Exercice : Quiz

[solution n°1 p.15]

### Question 1

Qu'est-ce qu'un serveur SMTP ?

- ☐ Un serveur de messagerie utilisé pour transporter les e-mails
- ☐ Un serveur qui stocke les données utilisateur
- ☐ Un serveur qui gère les paiements

Question 2

Comment installer le composant Mailer de Symfony à l'aide de Composer ?

- ☐ Composer add symfony/mailer
- ☐ Composer install symfony/mailer
- ☐ Composer require symfony/mailer

Question 3

Quel est l'objectif de la variable d'environnement Mailer\_DSN ?

- ☐ Configurer le serveur SMTP
- ☐ Configurer le template HTML de l'e-mail
- ☐ Configurer l'adresse de réponse

Question 4

Qu'est-ce que Mailtrap ?

- ☐ Un service en ligne pour recevoir des e-mails de test
- ☐ Un serveur SMTP
- ☐ Un outil de configuration de base de données

Question 5

Quelle méthode de l'objet Email permet d'ajouter des destinataires en copie cachée ?

- ☐ add();
- ☐ bcc();
- ☐ cc();

### III. Utilisation avancée et affichage d'email

#### A. TemplatedEmail

##### Méthode

Pour une meilleure intégration du HTML/CSS il est possible d'utiliser l'objet TemplatedEmail plutôt que l'objet Email. Il faudra bien sûr appeler la classe TemplatedEmail en faisant cet import :

```
1 use Symfony\Bridge\Twig\Mime\TemplatedEmail;
```

Assurez-vous de bien avoir Twig-Bundle dans votre projet. Sinon, il vous faudra l'installer à l'aide de composer. Il faut aussi changer l'objet Email par TemplatedEmail et modifier la méthode `html()` par `htmlTemplate()` dans laquelle vous insérerez le chemin de votre fichier twig. Puis, vous devez ajouter la méthode `context()`.

Grâce à cette méthode `context()`, il est possible de passer des paramètres à Twig avec des variables personnalisées. Vous pourrez également utiliser dans Twig la variable spéciale `'email'` qui permet d'accéder aux propriétés de l'e-mail (par exemple, le destinataire, l'expéditeur, l'objet, etc).

### Exemple

Ci-dessous le nouvel objet `$email` avec `TemplatedEmail`.

```
1 $email = (new TemplatedEmail())
2     ->from('monAdresseEmail@example.com')
3     ->to('AdresseEmailUtilisateur@example.com')
4     ->subject('Demande de renseignement')
5     ->htmlTemplate('emails/test.html.twig')
6     ->context([
7         'username' => 'Didier Deschamps'
8     ]);

1 #emails\test.html.twig
2
3 <!DOCTYPE html>
4 <html lang="en">
5 <head>
6     <meta charset="UTF-8">
7 </head>
8 <body>
9     <h1>Bonjour {{ username }} , bienvenue sur notre plateforme !</h1>
10    <p>Nous allons vous répondre le plus rapidement possible sur {{ email.subject }} </p>
11 </body>
12 </html>
```

Ci-dessus l'exemple d'un fichier twig qui sera utilisé pour la réponse. Puisque dans `context` nous avons passé `username`, nous pouvons le récupérer dans notre fichier twig.

La variable `email`, utilisée dans cet exemple, permet de récupérer toutes les propriétés souhaitées de l'objet `TemplatedEmail` créé dans le contrôleur. La variable `email` est une instance de `WrappedTemplatedEmail`, une classe qui est héritée de `TemplatedEmail`.

## Préparation du projet

Nous allons créer un projet dont le but sera de répondre automatiquement avec un fichier PDF associé à une demande de contact sur notre site. Pour cela nous allons créer un formulaire de contact et une application qui puisse le gérer.

Description des éléments dont nous aurons besoin :

- Une entité : pour les besoins de ce cours, elle sera nommée `contact` et aura comme propriété `username`, `email`, `subject`, `message` et `createAt`.
- Une définition de formulaire : vous pouvez définir votre formulaire soit dans un `FormType` ou directement dans le contrôleur.

Pour ce cours, nous utiliserons un `FormType` c'est-à-dire une classe nommée `ContactType` qui contiendra la définition du formulaire pour notre exemple.

- Un template : c'est-à-dire une page Twig qui contiendra le formulaire que l'utilisateur devra remplir.
- Un contrôleur : il sera nommé `ContactController` pour notre exemple.

## Configuration du Contrôleur

```

1 <?php
2
3 namespace App\Controller;
4
5 use App\Entity>Contact;
6 use App\Form\ContactType;
7 use App\Repository>ContactRepository;
8 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
9 use Symfony\Component\HttpFoundation\Request;
10 use Symfony\Component\HttpFoundation\Response;
11 use Symfony\Component\Routing\Annotation\Route;
12 use Symfony\Component\Mailer\MailerInterface;
13 use Symfony\Bridge\Twig\Mime\TemplatedEmail;
14
15
16 #[Route('/contact')]
17 class ContactController extends AbstractController
18 {
19     #[Route('/new', name: 'app_contact_new', methods: ['GET', 'POST'])]
20     public function new(Request $request, ContactRepository $contactRepository, MailerInterface
21 $mailer): Response
22     {
23         $contact = new Contact();
24         $form = $this->createForm(ContactType::class, $contact);
25         $form->handleRequest($request);
26
27         if ($form->isSubmitted() && $form->isValid()) {
28             $contact->setCreatesAt(new DateTimeImmutable());
29             $contactRepository->save($contact, true);
30             $address = $contact->getEmail();
31             $username = $contact->getUsername();
32
33             $email = (new TemplatedEmail())
34                 ->from('hello@example.com')
35                 ->to($address)
36                 ->subject('Test Symfony Mailer!')
37                 ->text('Envoi en utilisant Symfony Mailer')
38             ->htmlTemplate('emails/response.html.twig');
39             ->context([
40                 'username' => $username
41             ]);
42         };
43
44         $mailer->send($email);
45
46         return $this->redirectToRoute('app_contact_index', [], Response::HTTP_SEE_OTHER);
47     }
48
49     return $this->renderForm('contact/new.html.twig', [
50         'contact' => $contact,
51         'form' => $form,
52     ]);
53 }

```



Dans le code ci-dessus, nous avons récupéré dans la variable `$address`, l'adresse e-mail de l'utilisateur via le formulaire de contact que nous réutilisons pour répondre dans la méthode `->to()`. Ainsi que l'username issue de ce même formulaire, dans la variable `$username` qui est passée dans `context()` pour être utilisé dans le fichier twig de la méthode `htmlTemplate()`.

## B. Ajouter un fichier ou d'une image

### Ajouter un fichier

Aujourd'hui, il est courant d'ajouter des pièces jointes à un e-mail. Avec le composant Mailer, c'est la fonction `attachFromPath()` qui vous permettra de faire cela. Pour que le fichier puisse être lié à l'e-mail, il faudra que celui-ci soit présent sur le serveur qui enverra l'e-mail. Le chemin que vous indiquerez dans la méthode `attachFromPath()` par défaut démarre du dossier public.

#### Exemple

```
1 ->attachFromPath('pdf/Reponse.pdf');
```

En deuxième paramètre, vous pouvez renommer l'image et en troisième paramètre, préciser le type, même si Symfony en principe le gère très bien.

### Ajouter une image

Pour ajouter une image il est nécessaire avant d'ajouter dans le fichier twig.yaml le chemin pour que twig récupère des images.

```
1 #config/packages/twig.yaml
2 twig:
3     default_path: '%kernel.project_dir%/templates'
4     paths:
5     '%kernel.project_dir%/public/images' : images
```

Puis, appelez l'image dans le fichier twig en utilisant la variable email dont on a déjà parlé :

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5 </head>
6 <body>
7     <h1>Bonjour {{ username }} bienvenue sur notre application</h1>
8     <h2> Nous sommes très heureux de vous compter parmi nous </h2>
9     <p>Vous nous avez contacté pour {{ email.subject}}</p>
10    
11 </body>
12 </html>
```

## C. Affichage d'e-mails

### HTML web

La problématique du HTML web dans les e-mails est liée à la façon dont les clients de messagerie électronique rendent le code HTML. Contrairement aux navigateurs web, les clients de messagerie électronique ont des capacités de rendu HTML limitées et varient considérablement selon leur interprétation du code HTML. Certains clients de messagerie électronique ne prennent pas en charge les styles CSS, les images ou les balises HTML courantes, tandis que d'autres ont des restrictions sur la taille des fichiers ou des polices de caractères.

Cela peut rendre difficile la création de templates d'e-mails qui s'affichent correctement sur tous les clients de messagerie électronique.

### Méthode Tables HTML

Pour résoudre cette problématique, les développeurs d'e-mails utilisent souvent une approche basée sur les tables HTML.

Les tables HTML permettent de structurer les e-mails en utilisant des cellules qui peuvent être alignées horizontalement et verticalement. Cette approche est utile pour créer des mises en page complexes avec des colonnes, des en-têtes, des pieds de page et des images alignées avec du texte.

En utilisant exclusivement les tables HTML, les développeurs d'e-mails peuvent s'assurer que les e-mails seront affichés correctement sur tous les clients de messagerie électronique, car les tables sont largement prises en charge par tous les clients de messagerie électronique. En outre, cette approche permet de s'assurer que les e-mails sont également accessibles aux personnes qui utilisent des lecteurs d'écran ou des clients de messagerie électronique avec une prise en charge limitée des styles CSS.

### Exemple

```
1 <table cellpadding="0" cellspacing="0" border="0" width="100%">
2 <tr>
3   <td align="center">
4     
5   </td>
6 </tr>
7 <tr>
8   <td align="center">
9     <h1>Titre de l'email</h1>
10  </td>
11 </tr>
12 <tr>
13   <td>
14     <p>Contenu de l'email.</p>
15     <p>Contenu de l'email.</p>
16     <p>Contenu de l'email.</p>
17   </td>
18 </tr>
19 <tr>
20   <td align="center">
21     <p><a href="http://example.com">Lien vers notre site web</a></p>
22   </td>
23 </tr>
24 </table>
```

Dans cet exemple, une table HTML est utilisée pour structurer le contenu de l'email en utilisant des cellules alignées horizontalement et verticalement. Les attributs "cellpadding", "cellspacing" et "border" sont utilisés pour spécifier la mise en forme de la table, tandis que l'attribut "width" est utilisé pour définir la largeur de la table.

Chaque élément de contenu de l'email est placé dans une cellule de la table. Les balises HTML courantes telles que les images, les titres et les liens peuvent être utilisées à l'intérieur de chaque cellule. Les attributs "align" sont utilisés pour aligner les éléments de contenu horizontalement.

En utilisant cette approche basée sur les tables HTML, les développeurs peuvent créer des templates d'e-mails qui s'affichent correctement sur tous les clients de messagerie électronique, quelle que soit leur capacité à interpréter le code HTML.

## Inky

L'utilisation du filtre Inky peut aider à résoudre certains problèmes de compatibilité des emails avec les clients de messagerie électronique.

Inky est un langage de balisage HTML développé par la société ZURB pour aider à simplifier la création de modèles d'e-mails compatibles avec tous les clients de messagerie électronique. Le filtre Inky permet de convertir le code Inky en HTML standard, en ajoutant automatiquement les tables HTML nécessaires pour garantir la compatibilité avec les clients de messagerie électronique.

Notez que certains clients de messagerie électronique peuvent encore avoir des problèmes d'affichage avec des e-mails créés à l'aide d'Inky, en particulier sur les anciennes versions de clients de messagerie.

### Méthode

Pour pouvoir utiliser le filtre Inky il est nécessaire d'installer les packs ci-dessous :

```
1 $ composer require twig/inky-extra
2 $ composer require twig/extra-bundle
```

### Exemple

Voici un exemple d'utilisation du filtre Inky dans un template d'e-mail :

```
1 {% apply inky_to_html %}
2 <container>
3   <row>
4     <columns small="12" large="6">
5       <h1>Bienvenue !</h1>
6       <p>Merci de vous être inscrit à notre service. Nous sommes ravis de vous compter parmi
7       nos utilisateurs.</p>
8     </columns>
9     <columns small="12" large="6">
10      
11    </columns>
12  </row>
13 </container>
14 {% endapply %}
```

Dans cet exemple, nous utilisons la balise `apply` avec le filtre `inky_to_html` pour convertir le code Inky en HTML standard. Le code Inky est utilisé pour structurer le contenu de l'email, en utilisant des balises Inky comme `container`, `row` et `columns`. Le filtre Inky se charge de convertir le code Inky en HTML standard, en ajoutant automatiquement les tables HTML nécessaires pour garantir la compatibilité avec tous les clients de messagerie électronique.

Notez que dans cet exemple, nous utilisons également des classes de colonnes (`small` et `large`) pour contrôler la mise en page de l'e-mail sur différents appareils.

## Style/CSS

Le support des styles et du CSS dans les e-mails est assez limité par rapport à ce que l'on peut faire dans les pages web. Les clients de messagerie électronique ont tendance à désactiver ou à supprimer complètement les styles CSS dans les e-mails pour des raisons de sécurité ou de compatibilité.

Cependant, il est possible d'utiliser des styles CSS de base pour personnaliser la présentation des e-mails. Les styles doivent être définis en ligne (c'est-à-dire directement dans les balises HTML) plutôt que dans un fichier externe, car les clients de messagerie ne peuvent pas charger de fichiers CSS externes.

## D. Exercice : Quiz

[solution n°2 p.16]

### Question 1

Quel est l'avantage d'utiliser la classe TemplatedEmail plutôt que la classe Email ?

- ☐ Elle permet d'envoyer des pièces jointes
- ☐ Elle permet d'utiliser un page twig comme modèle
- ☐ Elle permet d'envoyer des e-mails en masse

### Question 2

Comment faut-il modifier l'utilisation de l'objet Email pour utiliser l'objet TemplatedEmail ?

- ☐ Changer la méthode text() ou html() par htmlTemplate() et insérer le chemin du fichier twig
- ☐ Ajouter la méthode context() pour passer des paramètres à Twig
- ☐ Les deux réponses précédentes sont correctes

### Question 3

Quelle est la variable spéciale de Twig permettant d'accéder aux propriétés de l'email ?

- ☐ Email
- ☐ Message
- ☐ Subject

### Question 4

Dans l'exemple du contrôleur, comment est envoyé l'e-mail de réponse ?

- ☐ En utilisant la méthode send() de l'objet Email
- ☐ En utilisant la méthode send() de l'objet MailerInterface
- ☐ En utilisant la méthode send() de l'objet WrappedTemplatedEmail

### Question 5

Que doit-on ajouter à notre projet pour pouvoir utiliser l'objet TemplatedEmail ?

- ☐ Twig-Bundle
- ☐ Doctrine-Bundle
- ☐ Security-Bundle

## IV. Essentiel

Le composant Mailer de Symfony est un outil puissant pour la gestion et l'envoi d'e-mails dans les applications web modernes. Il facilite grandement la tâche des développeurs en permettant une configuration simple et flexible des paramètres d'envoi d'e-mails tels que les serveurs SMTP, les adresses d'envoi et de réponse, les en-têtes, etc.

L'installation du composant Mailer se fait facilement à l'aide de Composer, et la configuration du serveur SMTP se fait via une variable d'environnement dans le fichier .env à la racine du projet. Il est également possible d'utiliser des services en ligne comme Mailtrap pour tester l'envoi et la réception d'e-mails sans envoyer de véritables e-mails à des destinataires.

Un autre atout du composant Mailer est qu'il offre une grande flexibilité pour créer et envoyer des e-mails avec des templates HTML, des pièces jointes et des adresses de réponse. Il est également possible de configurer des en-têtes et des signatures pour personnaliser les e-mails envoyés.

En conclusion, le composant Mailer est très performant et adapté pour les applications Symfony. Il offre une intégration facile avec d'autres composants de Symfony. En utilisant ce composant, les développeurs peuvent créer des applications web modernes avec une gestion efficace et flexible de l'envoi d'e-mails.

## V. Auto-évaluation

### A. Exercice

Vous êtes un développeur junior dans une agence web. Vous avez été formé au PHP et vous maîtrisez le framework de Symfony. On vous confie différents projets sur lesquels vous devez apporter des modifications ou de nouvelles fonctionnalités.

#### Question 1

[solution n°3 p.17]

Dans le premier projet du jour, on vous demande de créer une réponse automatique à un formulaire de demande de service. Dans cette réponse, vous préciserez que nous les recontacterons dans les 24 h en précisant le numéro de téléphone que le client a renseigné. Décrivez uniquement le contrôleur.

#### Question 2

[solution n°4 p.17]

Votre chef de service vient vous voir et vous demande de modifier la réponse automatique de l'e-mail, car il souhaite une meilleure mise en forme, et d'utiliser twig pour cela. Précisez ci-dessous uniquement le nouvel objet \$email.

### B. Test

#### Exercice 1 : Quiz

[solution n°5 p.18]

##### Question 1

Quelle est la différence entre l'objet Email et l'objet TemplatedEmail ?

- ☐ L'objet Email est utilisé pour envoyer des e-mails sans modèle tandis que l'objet TemplatedEmail est utilisé pour envoyer des e-mails avec un modèle Twig
- ☐ L'objet Email est utilisé pour envoyer des e-mails avec un modèle Twig tandis que l'objet TemplatedEmail est utilisé pour envoyer des e-mails sans modèle
- ☐ Il n'y a pas de différence entre l'objet Email et l'objet TemplatedEmail

##### Question 2

Quelle méthode est utilisée pour récupérer les en-têtes d'e-mail ?

- ☐ getHeaders()
- ☐ addHeaders()
- ☐ Headers()

##### Question 3

Comment peut-on passer des paramètres à Twig avec des variables personnalisées ?

- ☐ En utilisant la méthode "send()" dans l'objet TemplatedEmail
- ☐ En utilisant la méthode "context()" dans l'objet TemplatedEmail
- ☐ En utilisant la méthode "html()" dans l'objet TemplatedEmail

##### Question 4

Comment peut-on modifier la méthode `html()` en méthode `htmlTemplate()` dans l'objet `TemplatedEmail` ?

- ☐ En écrivant simplement `"htmlTemplate()"` au lieu de `"html()"` dans le code
- ☐ En appelant la méthode `htmlTemplate()` avec le chemin du fichier Twig
- ☐ En utilisant la méthode `getHtmlTemplate()` au lieu de `html()`

Question 5


Quelle méthode spécifie l'adresse e-mail de l'expéditeur de l'e-mail ?

- ☐ `addFrom()`
- ☐ `from()`
- ☐ `addTo()`

## Solutions des exercices


**Exercice p. 5 Solution n°1****Question 1**

Qu'est-ce qu'un serveur SMTP ?

- ☒ Un serveur de messagerie utilisé pour transporter les e-mails
- ☐ Un serveur qui stocke les données utilisateur
- ☐ Un serveur qui gère les paiements
-  Un serveur SMTP (Simple Mail Transfer Protocol) est un serveur de messagerie qui est utilisé pour transporter l'e-mail d'un expéditeur vers un ou plusieurs destinataires.


**Question 2**

Comment installer le composant Mailer de Symfony à l'aide de Composer ?

- ☐ Composer add symfony/mailer
- ☐ Composer install symfony/mailer
- ☒ Composer require symfony/mailer
-  Composer utilise « *require* » pour installer des packages dans votre application.


**Question 3**

Quel est l'objectif de la variable d'environnement Mailer\_DSN ?

- ☒ Configurer le serveur SMTP
- ☐ Configurer le template HTML de l'e-mail
- ☐ Configurer l'adresse de réponse
-  Tout comme nous avons besoin de définir le serveur de notre base de données dans une variable d'environnement, Mailer\_DSN définit le serveur SMTP en accord avec le bon hébergeur / transporteur.


**Question 4**

Qu'est-ce que Mailtrap ?

- ☒ Un service en ligne pour recevoir des e-mails de test
- ☐ Un serveur SMTP
- ☐ Un outil de configuration de base de données
-  Mailtrap est un service en ligne pour recevoir des e-mails de test. Il est très populaire et facile à mettre en place.

**Question 5**


Quelle méthode de l'objet Email permet d'ajouter des destinataires en copie cachée ?

- ☐ add();
- ☒ bcc();
- ☐ cc();
-  Avec cette méthode, les autres destinataires recevront aussi le courrier, mais ne connaîtront pas les autres destinataires qui sont dans cette fonction.

## Exercice p. 12 Solution n°2


### Question 1

Quel est l'avantage d'utiliser la classe TemplatedEmail plutôt que la classe Email ?

- ☐ Elle permet d'envoyer des pièces jointes
- ☒ Elle permet d'utiliser un page twig comme modèle
- ☐ Elle permet d'envoyer des e-mails en masse
-  La classe TemplatedEmail dans Symfony est utilisée pour envoyer des e-mails à partir d'un modèle prédéfini (template) à l'aide de Twig, le moteur de template utilisé dans Symfony.


### Question 2

Comment faut-il modifier l'utilisation de l'objet Email pour utiliser l'objet TemplatedEmail ?

- ☐ Changer la méthode text() ou html() par htmlTemplate() et insérer le chemin du fichier twig
- ☐ Ajouter la méthode context() pour passer des paramètres à Twig
- ☒ Les deux réponses précédentes sont correctes
-  Effectivement, il faut modifier la méthode pour le contenu ainsi qu'utiliser la méthode context() qui permet d'utiliser la puissance de twig.


### Question 3

Quelle est la variable spéciale de Twig permettant d'accéder aux propriétés de l'email ?

- ☒ Email
- ☐ Message
- ☐ Subject
-  Email permet d'afficher de nombreux éléments de l'e-mail, par exemple pour afficher le sujet, on l'appellera avec email.subject.

### Question 4


Dans l'exemple du contrôleur, comment est envoyé l'e-mail de réponse ?

- ☐ En utilisant la méthode send() de l'objet Email
- ☒ En utilisant la méthode send() de l'objet MailerInterface
- ☐ En utilisant la méthode send() de l'objet WrappedTemplatedEmail
-  send() est une méthode de MailerInterface qui, dans les exemples du cours, est appelée \$mailer.



**Question 5**

Que doit-on ajouter à notre projet pour pouvoir utiliser l'objet TemplatedEmail ?

- ☒ Twig-Bundle
- ☐ Doctrine-Bundle
- ☐ Security-Bundle
-  L'objet TemplatedEmail a pour objectif d'envoyer le rendu de l'email avec un moteur de template comme twig.

**p. 13 Solution n°3**

```

1  #[Route('/contact', name: 'contact', methods: ['GET', 'POST'])]
2  public function contact(Request $request, ContactRepository $contactRepository,
3  MailerInterface $mailer): Response
4  {
5      $contact = new Contact();
6      $form = $this->createForm(ContactType::class, $contact);
7      $form->handleRequest($request);
8
9      if ($form->isSubmitted() && $form->isValid()) {
10
11          $contactRepository->save($contact, true);
12          $address = $contact->getEmail();
13          $username = $contact->getUsername();
14          $phone = $contact->getPhone();
15
16          $email = (new Email())
17              ->from('thebestfirm@example.com')
18              ->to($address)
19              ->subject('réponse à votre demande de service')
20              ->html(
21                  '<h1> Merci ' . $username . 'de vous tourner vers nos services.</h1>'
22                  . '<p>Nous ne manquerons pas de vous rappeler au ' . $phone . ' dans les 24'
23                  . ' heures </p>'
24              );
25
26          $mailer->send($email);
27
28          return $this->redirectToRoute('app_contact_index', [], Response::HTTP_SEE_OTHER);
29      }
30
31      return $this->render('contact/new.html.twig', [
32          'contact' => $contact,
33          'form' => $form->createView(),
34      ]);
35  }

```

**p. 13 Solution n°4**

```

1 $email = (new TemplatedEmail())
2     ->from('thebestfirm@example.com')
3     ->to($address)
4     ->subject('réponse à votre demande de service')
5     ->htmlTemplate('emails/reponse.html.twig')

```

```

6         ->context([
7             'username' => $username,
8             'phone' => $phone
9         ])
10    ;
11
12    $mailer->send($email);

```

## Exercice p. 13 Solution n°5

### Question 1

Quelle est la différence entre l'objet Email et l'objet TemplatedEmail ?

- ☒ L'objet Email est utilisé pour envoyer des e-mails sans modèle tandis que l'objet TemplatedEmail est utilisé pour envoyer des e-mails avec un modèle Twig
- ☐ L'objet Email est utilisé pour envoyer des e-mails avec un modèle Twig tandis que l'objet TemplatedEmail est utilisé pour envoyer des e-mails sans modèle
- ☐ Il n'y a pas de différence entre l'objet Email et l'objet TemplatedEmail
- ☒ L'objet TemplatedEmail permet d'envoyer des e-mails plus complexes que l'objet Email avec des modèles Twig.

### Question 2

Quelle méthode est utilisée pour récupérer les en-têtes d'e-mail ?

- ☒ getHeaders()
- ☐ addHeaders()
- ☐ Headers()
- ☒ Souvent, cette méthode utilisera à son tour d'autres méthodes telles que setHeaders().

### Question 3

Comment peut-on passer des paramètres à Twig avec des variables personnalisées ?

- ☐ En utilisant la méthode "send()" dans l'objet TemplatedEmail
- ☒ En utilisant la méthode "context()" dans l'objet TemplatedEmail
- ☐ En utilisant la méthode "html()" dans l'objet TemplatedEmail
- ☒ La méthode context() de l'objet TemplatedEmail permet, tout comme la méthode render() utilisée dans un contrôleur, de passer des données à un template Twig.

### Question 4

Comment peut-on modifier la méthode html() en méthode htmlTemplate() dans l'objet TemplatedEmail ?

- ☐ En écrivant simplement "htmlTemplate()" au lieu de "html()" dans le code
- ☒ En appelant la méthode htmlTemplate() avec le chemin du fichier Twig
- ☐ En utilisant la méthode getHtmlTemplate() au lieu de html()

🔍 La méthode `htmlTemplate()` a besoin de connaître le chemin pour trouver le fichier Twig correspondant.

### Question 5

---

Quelle méthode spécifie l'adresse e-mail de l'expéditeur de l'e-mail ?

- ☐ `addFrom()`
- ☒ `from()`
- ☐ `addTo()`

🔍 La méthode `from()` indique l'e-mail de l'expéditeur. Cette adresse peut directement être spécifiée ou dans une variable préalablement instanciée.