

# Le Profiler

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. À la découverte du Profiler de Symfony</b>	<b>3</b>
A. À la découverte du Profiler de Symfony .....	3
B. Exercice : Quiz .....	6
<b>III. Les onglets du Profiler</b>	<b>6</b>
A. Les différents onglets .....	6
B. DataCollector .....	12
C. Exercice : Quiz .....	15
<b>IV. Essentiel</b>	<b>15</b>
<b>V. Auto-évaluation</b>	<b>16</b>
A. Exercice .....	16
B. Test .....	16
<b>Solutions des exercices</b>	<b>17</b>

## I. Contexte

**Durée :** 1 h

**Prérequis :**

- Notion de PHP
- Notion de POO

**Environnement de travail :** un ordinateur

### Contexte

Développer une application web avec un framework comme Symfony 5.4 apporte un vrai confort, car c'est une version stabilisée. Elle est LTS, pour *Long-Term Support*, c'est-à-dire qu'elle a un support à long terme. En effet, SensioLabs s'engage à corriger les bugs de cette version jusqu'en novembre 2024 et à suivre la sécurité jusqu'en novembre 2025. À savoir que les LTS sortent tous les 2 ans lorsqu'une nouvelle version majeure de Symfony voit le jour. La LTS sera toujours la dernière version mineure au bout des deux ans (3,4, 4,4, 5,4, 6,4, etc.). C'est ce que l'on appelle le Symfony LifeCycle.

Symfony offre beaucoup d'outils afin de faciliter la tâche des développeurs. Dans ce cours, nous allons découvrir la Debug Toolbar, qui se nomme Profiler, terme anglais qui ne nécessite pas de traduction dans le milieu de la programmation.

Le Profiler est un outil de débogage qu'il faut installer. Il est à la fois très simple et très puissant. C'est véritablement un outil indispensable pour développer une application web, car il est essentiel d'être informé de l'état de notre application dès que nous ajoutons de nouvelles fonctionnalités ou que nous apportons des modifications.

Comment déboguer une application avec Symfony ? Comment utiliser le Profiler ? Que signifie chaque icône de celui-ci ? Peut-on ajouter d'autres fonctionnalités ? C'est ce que nous allons découvrir dans ce cours sur le Profiler.

## II. À la découverte du Profiler de Symfony

### A. À la découverte du Profiler de Symfony

Le Profiler ainsi que d'autres outils de débogage ne sont pas installés par défaut dans le framework de Symfony. Il va donc nous falloir l'installer. Avant de le faire, il faut toutefois bien comprendre que c'est un outil de développement, voire de test, et qu'il ne faut en aucun l'utiliser en mode production. Car des outils de débogage tel que le Profiler, non seulement jouent sur les performances, mais surtout peut ouvrir à des failles de sécurité.

Par défaut, la variable d'environnement APP\_ENV se trouve dans le fichier .env et c'est elle qui va définir si on est en mode Développement, Test ou Production en mettant cette variable égale à dev, test ou prod.

### Installation

Si vous souhaitez ajouter uniquement le pack Profiler, vous pouvez l'installer avec cette ligne de commande dans le projet que vous venez de créer :

```
1 $ composer require --dev symfony/profiler-pack
```

Nous vous conseillons de plutôt installer debug qui offre d'autres outils de débogage même si nous ne les détaillerons pas tous dans ce cours ainsi que des fonctionnalités plus poussées au Profiler qu'il installera de toute façon.

```
1 $ composer require debug --dev
```

Si vous venez d'effectuer cette ligne de commande, vous pouvez noter tous les packs qui ont été installés et configurés, à savoir :

- twig/twig
- symfony/translation-contracts
- symfony/twig-bridge
- symfony/twig-bundle
- symfony/web-profiler-bundle
- symfony/stopwatch
- symfony/profiler-pack
- monolog/monolog
- symfony/monolog-bridge
- symfony/monolog-bundle
- symfony/debug-bundle
- symfony/debug-pack

Maintenant, lancez le serveur avec la commande ci-dessous :

```
1 $ symfony serve
```

Cliquez sur l'adresse qu'il vous propose comme `http://127.0.0.1:8000` et constatez le résultat.

## Le Profiler

Voici, à quelque chose près, ce que vous devriez avoir en bas de votre page :

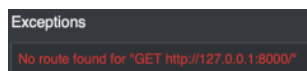


### Toolbar après installation

Cette Toolbar donne des informations précieuses que nous pouvons visualiser d'un seul coup d'œil, en « *Hover* » ou survol vous aurez quelques informations de plus mais tout cela n'est qu'un moyen visuel et rapide d'avoir accès aux informations les plus importantes. Nous décortiquons plus précisément ces informations dans la partie 2 « *Les onglets du Profiler* » de ce cours.

La première impression déjà c'est qu'il y a du rouge, alors que le projet vient juste d'être installé et que tout à l'air de fonctionner. Cet outil est là pour vous aider et vous apporter des solutions, alors avant même de commencer à découvrir les icônes de cette Toolbar, autre nom que l'on donne au Profiler, nous allons commencer à déboguer ou du moins à comprendre ce qu'il se passe. Pour cela, cliquez sur le 404 ou n'importe où sur la barre afin que le Profiler vous ouvre une page de débogage. Vous devriez voir deux 1 dans des carrés rouges. Un dans Exceptions et un dans Logs. Cliquez sur Exceptions.

Voici le message que vous devriez avoir :



### Message Exceptions

Comme toujours dans la programmation et le monde de l'informatique, il est nécessaire de connaître un minimum d'Anglais. Le message est clair, le profiler n'a pas trouvé de route pour `GET http://127.0.0.1:8000/`. C'est normal, nous n'en avons pas encore créé et Symfony, par l'intermédiaire du Profiler, nous dit que ce n'est pas normal, parce que pour toute adresse URL il faut une route. Si vous cliquez sur Logs ou que vous regardez votre console, vous trouverez un message similaire.

Puisque nous sommes sur Symfony Profiler, découvrons cette page avec les mines d'informations qui nous sont fournies.

### Méthode Barre de recherche

Tout en haut à droite nous avons un outil de recherche qui nous emmène directement sur la documentation officielle de Symfony. Essayez par exemple avec le mot « *Profiler* ». Vous pouvez choisir parmi les différentes propositions de recherche la référence qui vous paraît la plus opportune. La première proposition est intéressante. Ensuite, sur votre droite, choisissez la version de Symfony que vous utilisez.

### Méthode URL

http://127.0.0.1:8000/  
Method: GET - HTTP Status: 404 - IP: 127.0.0.1 - Profiled on: Wed, 18 Jan 2023 08:24:00 +0000 - Token: a33736

#### Informations URL

Maintenant, juste en dessous, en rouge dans notre cas parce que nous sommes en HTTP Status 404, bien sûr nous serions en vert si nous étions en HTTP Status 200, vous trouverez des informations sur l'URL et la connexion. Rien de bien compliqué, vous devriez pouvoir comprendre ces informations. Notez tout de même le *token* qui est généré par le Profiler. Grâce à celui-ci, vous pourrez par exemple accéder aux données du Profiler dans votre programme, par exemple avec ce code ci-dessous :

```
1 $token = $response->headers->get('X-Debug-Token');  
2 $profile = $container->get('profiler')->loadProfile($token);
```

Profile Search :

The screenshot shows the Symfony Profiler search interface. At the top, there are three tabs: 'Last 10', 'Latest', and 'Search'. Below these, there are several input fields for filtering search results: 'IP', 'Method' (set to POST), 'Status', 'URL', 'Token', 'From' (set to 20/01/2023), 'Until' (set to 20/01/2023), and 'Results' (set to 10). A 'Search' button is located at the bottom right of the search area.

#### Recherche dans les résultats de Profiler

Descendons un peu plus bas et sur la gauche, vous devriez voir cet outil de recherche qui vous permet de retrouver les 10 derniers résultats, le dernier résultat ou d'ouvrir un outil de recherche afin de retrouver un résultat précis. Tapez par exemple l'adresse IP, et si vous avez lancé plusieurs fois une requête, vous devriez avoir une liste de résultats avec un token différent à chaque fois.

## B. Exercice : Quiz

[solution n°1 p.19]

### Question 1

La Toolbar permet de connaître la version de Symfony avec lequel l'application est développée.

- ☐ Vrai
- ☐ Faux

### Question 2

Quelle est la bonne ligne de commande pour installer le Profiler ?

- ☐ \$ composer require debug --dev
- ☐ \$ composer require debug/profiler --dev
- ☐ \$ composer require --dev profiler-pack

### Question 3

Où peut-on accéder à l'outil Profiler dans Symfony ?

- ☐ Dans la console Symfony
- ☐ Dans le navigateur web
- ☐ Dans un IDE

### Question 4

Le Profiler génère un token pour chaque requête.

- ☐ Vrai
- ☐ Faux

### Question 5

Dans Profile Search, il est possible de rechercher par numéro de statut HTTP un résultat de requête.

- ☐ Vrai
- ☐ Faux

## III. Les onglets du Profiler

### A. Les différents onglets

#### Méthode Request/Response

Cet onglet est très utile. Si vous cliquez dessus, vous aurez une page centrale qui s'affiche avec 6 onglets : Request, Response, Cookies, Session, Flashes, Server Parameters.



### Onglet Profiler Request/Response

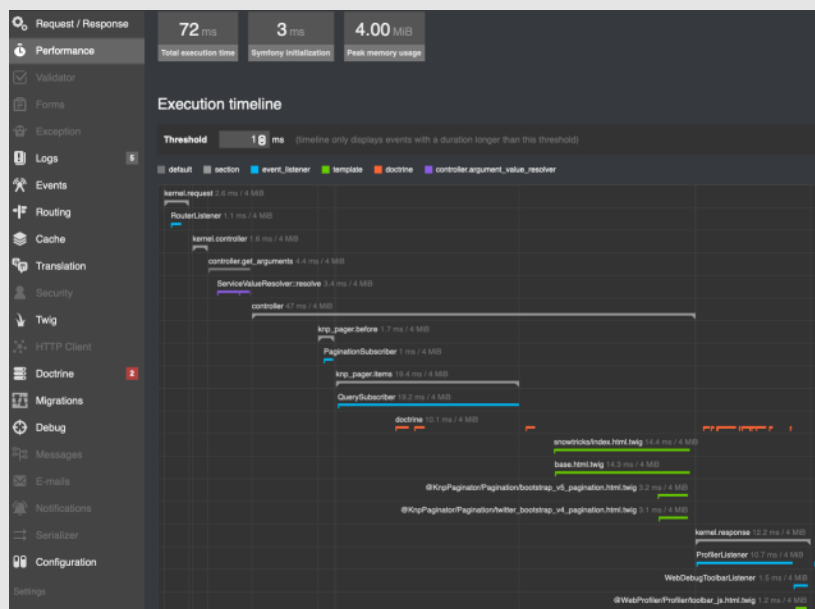
#### • Request :

- GET et POST Parameters : vous permet de connaître les paramètres GET ou POST. Par exemple, si vous validez un formulaire, vous aurez dans POST Parameters des informations sur le nom du formulaire utilisé et les valeurs récupérées.
- Uploaded Files : vous renseignera dans « *key* » à partir de quel formulaire le ou les fichiers ont été téléchargés et dans « *Value* », vous aurez les informations sur l'objet créé.
- Request Attributes : vous y retrouvez par exemple les informations sur le nom du contrôleur utilisé ainsi que le nom de la route.
- Request Headers : contient des informations sur l'en-tête de la requête HTTP.
- Request Content : contient des informations sur le contenu de la requête HTTP.

Notez que dans la Toolbar, si vous survolez le 404 de notre exemple de départ, vous retrouverez en plus de ce code de réponse HTTP des informations issues de Request telles que le nom de la route et le nom du contrôleur mais aussi l'état de la session et du pare-feu.

- **Response** : donne l'en tête de la réponse.
- **Cookies** : donne des informations sur les cookies et les redirections.
- **Session** : donne les informations sur la super globale Session.
- **Flashes** : donne les informations sur les messages flashes qui ont été retransmis.
- **Server Parameters** : donne des informations sur les paramètres du serveur et des variables d'environnement.

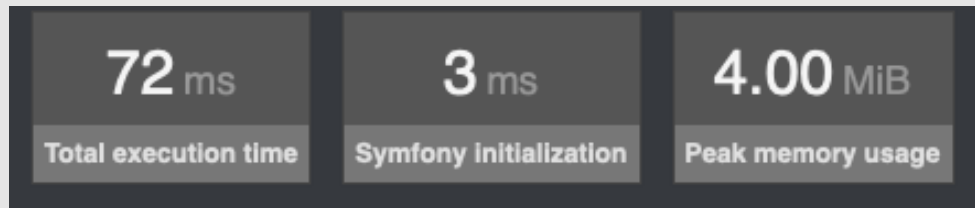
### Méthode Onglet Performance



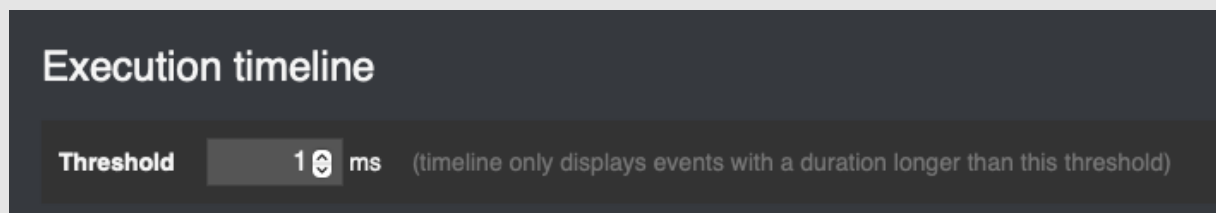
### Onglet Performance

Pour que l'exemple soit un peu plus parlant, voici ci-dessus les performances d'une petite application web fonctionnelle.

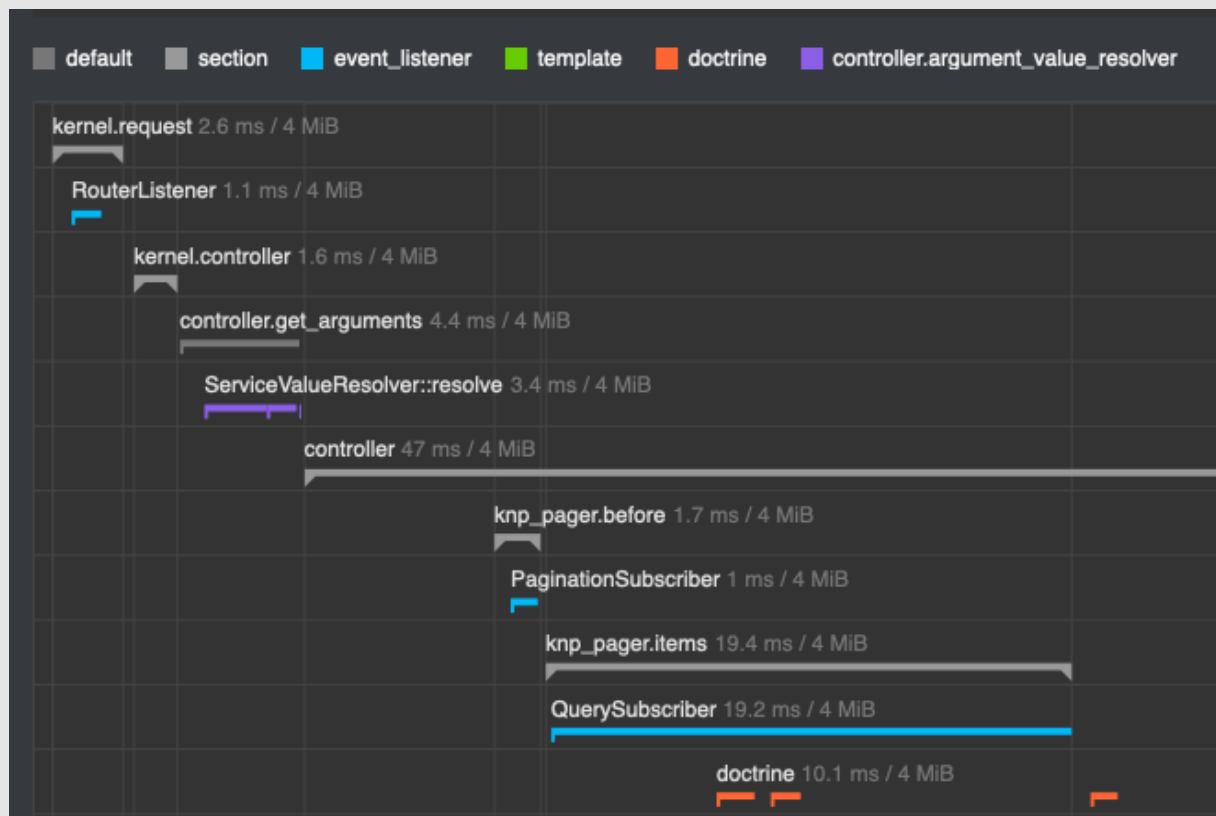
Les trois chiffres du haut nous donnent le temps d'exécution, la mémoire utilisée et nous retrouvons ces deux chiffres dans la Toolbar et au milieu le temps initialisation de Symfony.



Sous « *Execution timeline* », il y a un filtre qui se nomme « *Threshold* », on peut augmenter ou diminuer la valeur. Cela permet de mieux identifier les éléments qui utilisent le plus les ressources.



Puis, en dessous, nous avons une représentation graphique avec le temps d'exécution de chaque élément qui constitue la réponse HTTP. Les couleurs permettent de mieux identifier les catégories d'éléments. Vous noterez peut-être que la Toolbar utilise aussi les performances de l'application. Les performances du rendu, qui est généralement en vert, sont cumulées et données en informations dans la Toolbar.

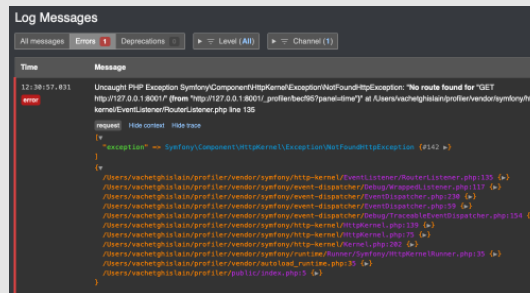




## Méthode Onglet Exception

Dans cet onglet, vous retrouverez toutes les informations concernant les erreurs survenues durant l'exécution de la requête. C'est celui que nous avons regardé en premier dans ce cours. Généralement il affiche un message explicite, dans notre cas c'était qu'il n'avait pas trouvé la route. En dessous, dans le sous-onglet « *Exceptions* », l'endroit où l'erreur s'est déclenchée s'affiche. Cette partie est souvent intéressante mais pas toujours révélatrice du vrai problème. Les sous-onglets Logs et Stack Traces permettent de comprendre tout ce qu'il s'est passé durant la requête mais il s'adresse à un public averti, si vous êtes développeur junior, vous mettrez un peu de temps à trouver ces renseignements utiles.

## Méthode Onglet Logs



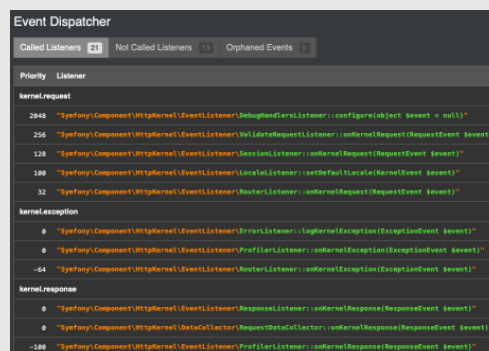
### Onglet Logs avec un « context » et « trace » ouvert

Cet onglet est l'une des principales caractéristiques du Profiler de Symfony. Il montre non seulement les journaux de débogage d'erreur mais aussi toutes les dépréciations déclenchées par l'application. Depuis la version 5.4 de Symfony, cette section a été entièrement repensée. Chaque message de journal a été réorganisé pour que les informations soient plus faciles à comprendre. Par exemple, sous l'horodateur, on voit immédiatement si c'est un debug, une info ou une erreur et dans la partie message, la catégorie. Le système de filtrage par type (level) ou catégorie (channel) est plus avancé et plus rapide. Par défaut, Symfony propose ses propres channels (debug, info, etc.), et priorités. Sachez que par la suite, vous pouvez créer néanmoins vos propres catégories de logs et mener des actions spécifiques (alerting, etc.) selon leurs levels. Vous les retrouverez également ici, dans la debug toolbar.

Les boutons « *Show context* » et « *Show trace* » vous apporteront d'autres informations dont vous avez besoin. Cet onglet est vraiment bien pensé pour vous aider à diagnostiquer les problèmes. Il est représenté dans la Toolbar par une icône avec un point d'exclamation. En survolant l'icône, vous pourrez savoir combien il y a d'erreurs, d'alertes et de dépréciations.

## Méthode Onglet Events

Cet onglet liste tous les événements qui ont été lancés durant la requête. En effet, Symfony travaille pour vous, le framework « *écoute* » tout ce qui se passe dans l'application. Regardez déjà tout ce qui est listé sur cette première page, alors que nous n'avons pas encore commencé à coder.



## Onglet Events

### Méthode Onglet Routing

Cet onglet Profiler va lister toutes les routes qui ont été trouvées dans votre application. Vous y trouverez son nom et son chemin URL. Le Profiler mettra en vert, comme dans l'exemple ci-dessous, ce qui est issu d'une application fonctionnelle, la route qui « matches », c'est-à-dire qui correspond à la requête dans la liste mais aussi dans un encadré en haut de la page. Si des paramètres ont été passés à la route, vous les trouverez dans la première section.

27	security_registration	/inscription	Path does not match
28	verify_user	/verif/{token}	Path does not match
29	resend_verif	/renvoi-verif	Path does not match
30	security_login	/connection	Path does not match
31	security_logout	/deconnection	Path does not match
32	forgotten_password	/oubli-pass	Path does not match
33	reset_pass	/oubli-pass/{token}	Path does not match
34	home	/	Path does not match
35	add_figure	/snowtricks/addfigure	Route matches!

## Onglet Routing

### Méthode Onglet Cache

L'utilisation d'un cache est un excellent moyen de rendre une application plus rapide. Lorsque vous maîtriserez cette partie, vous y trouverez les informations sur le cache dans cet onglet.

#### Cache

2	0.01 ms	1	1	0	0	1	0 %
Total calls	Total time	Total reads	Total writes	Total deletes	Total hits	Total misses	Hits/reads

#### Pools

cache.app	0	cache.system	0	cache.validator	0	cache.serializer	0	cache.annotations	0	cache.property_info	0
cache.messenger.restart_workers_signal	0	cache.validator_expression_language	0	cache.doctrine.orm.default.result	0	cache.doctrine.orm.default.query	2	cache.security_expression_language	0		

#### Metrics

2	0.01 ms	1	1	0	0	1	0 %
Calls	Time	Reads	Writes	Deletes	Hits	Misses	Hits/reads

#### Calls

#	Time	Call	Hit
1	0.01 ms	getItem()	[ "a2733d3b60885675a89532f6e9c78b18" => false ]
2	0.01 ms	save()	[ "a2733d3b60885675a89532f6e9c78b18" => true ]

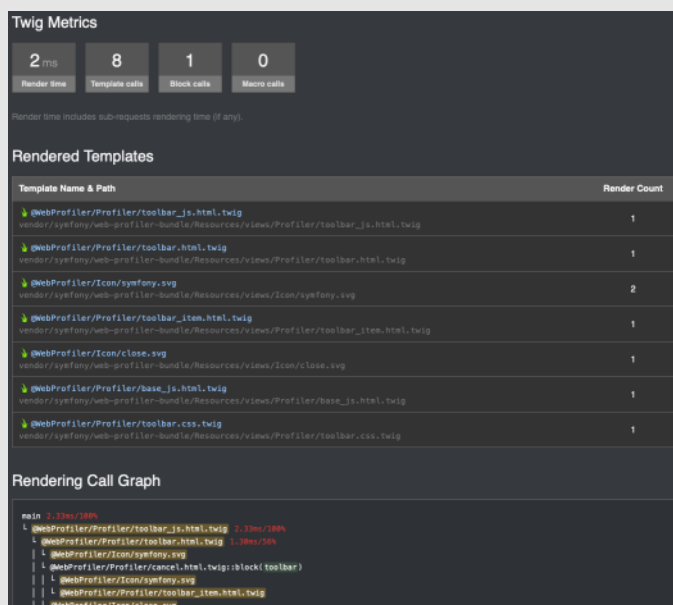
## Onglet Cache

**Méthode** Onglet Twig Metrics

Comme son nom l'indique, cet onglet va vous donner des mesures sur le moteur de template Twig. Vous retrouverez dans la Toolbar sous la forme d'une icône représentant une brindille les valeurs clés à retenir. À savoir :

- Le temps du rendu de la page en millisecondes.
- Les éléments de template appelés, cela peut être la Toolbar par exemple.
- Le nombre de blocs Twig appelés. Imaginez que vous avez un formulaire Twig, un bloc sera par exemple un `form_row`.
- Le nombre de macros appelées, à savoir qu'une macro Twig est comparable à une fonction en PHP.

Vous pouvez retrouver tous ces éléments de la Toolbar en haut de la page Profiler dans l'onglet Twig Metrics. En dessous, dans « *Rendered Templates* », il y a le descriptif de chaque élément Twig appelé. Dans « *Rendering Call Graph* », vous trouverez le graphique d'appel complet généré par Twig de la page.



## Onglet Twig Metrics

**Méthode** Debug

Vous connaissez certainement la fonction `var_dump()` de PHP qui affiche les informations structurées d'une variable avec son type et sa valeur, et explore les tableaux et les objets. La fonction `dump()` provient de Symfony et permet d'afficher les informations de `var_dump` de façon plus visuelle et graphique.

`Dump()`, lorsqu'elle est utilisée dans le code PHP de l'application Symfony, se verra représentée dans la Toolbar par une icône représentant une cible, suivie du nombre de fois que cette méthode a été utilisée. Au survol, vous aurez le visuel des informations récoltées par `Dump()`. Il existe également la méthode `dd` (*dump and die*) qui permet de dump grâce au composant debug de Symfony et par la suite d'arrêter l'exécution du code. C'est très utile lorsque vous voulez déboguer une ligne sans exécuter des actions suivantes.

Dans le Profiler, vous aurez tous les résultats des `dump()` affichés dans l'encadré Dumped Contents.

Voici un exemple avec une application fonctionnelle où 2 dump() ont été générés, un renvoie 10 qui est la valeur d'une variable et l'autre tous les éléments d'une entité nommée Category :

```

IP: 127.0.0.1 Profiled on: Thu, 19 Jan 2023 16:22:50 +0000 Token: 7526df

Dumped Contents
In SnowtricksController.php line 34:
10

In SnowtricksController.php line 34:
array (
  0 => App\Entity\Category (#764)
  -id: 1
  -name: "minim enim eu"
  -description: "Quosquet dignissimos autem quisquam dolores voluptas vel alias. Ea qui atque utam euismod. Minima fugiat sed fugit et facillia voluptas quibusdam. Quosquet dignissimos autem Ea qui atque enim aut. Sed ipsum nequequam aut autem euismod. Minima fugiat sed fugit et facillia."
  -figures: Doctrine\ORM\PersistentCollection (#778)
  -collection: Doctrine\Common\Collections\ArrayCollection (#789)
  -elements: []
  -initialized: false
  -snapshot: []
  -owner: App\Entity\Category (#764)
  -association: V\api::[] (1)
  -em: Doctrine\ORM\EntityManager (#774 ..1)
  -backedFieldName: "groups"
  -typeClass: Doctrine\ORM\Mapping\ClassMetadata (#766 ..)
  -isDirty: false
)
1 => App\Entity\Category (#772)
  -id: 5
  -name: "est ullam esse"
  -description: "Adipisci eaque esse delectati plerumque deserunt adipisci sed. Reprehenderit ipsa! Iste quidem laboriosam x est ut ipsa! Iste."
  -figures: Doctrine\ORM\PersistentCollection (#778)
  -collection: Doctrine\Common\Collections\ArrayCollection (#782)
  
```

### Onglet Debug

Ce ne sont là que les descriptions des onglets fournis par défaut par le profiler Web. Toutefois, il en existe beaucoup d'autres. D'ailleurs, dès que vous allez construire votre application web, vous verrez apparaître l'onglet Doctrine où vous découvrirez toutes les requêtes faites pour vous vers la base de données ainsi que la description de toutes les entités mappées. Pour vos formulaires, c'est l'onglet Forms qui vous aidera à identifier les formulaires utilisés ainsi que leurs champs.

Parlons aussi de l'onglet sécurité. Dès que vous aurez mis un système de connexion et d'authentification en place avec Security, vous aurez un onglet dédié ainsi qu'une icône représentant une personne dans la Toolbar où vous pourrez voir si vous êtes connecté ainsi que votre rôle et d'autres informations. Dans l'onglet Security, vous aurez des informations sur le token, le firewall et les décisions d'accès pour ne parler que des sous-onglets les plus importants.

Ainsi, vous aurez un nouvel onglet dans le Profiler la plupart du temps après avoir installé un package avec de nouvelles fonctionnalités.

## B. DataCollector

Il est possible de créer soit même un outil Profiler. Symfony nous fournit DataCollector pour cela.

Imaginons que nous souhaitons ajouter une icône à la Toolbar qui nous permettrait de connaître le nombre d'utilisateurs inscrits. Pour cela, créons un fichier PHP UserListCollector que nous pourrions mettre dans un dossier nommé DataCollector, « src\DataCollector\UserListCollector ».

```

1 <?php
2
3 namespace App\DataCollector;
4
5 use App\Repository\UserRepository;
6 use Symfony\Component\HttpFoundation\Request;
7 use Symfony\Component\HttpFoundation\Response;
8 use Symfony\Component\HttpKernel\DataCollector\DataCollector;
9
10 class UserListCollector extends DataCollector
11 {
12

```

```

13 public function __construct(
14     private UserRepository $userRepository,
15 ) {}
16
17 public function collect(Request $request, Response $response, \Throwable $exception = null)
18 {
19     $this->data = [
20         'method' => $request->getMethod(),
21         'acceptable_content_types' => $request->getAcceptableContentTypes(),
22         'users' => $this->userRepository->findAll(),
23         'countUsers' => count($this->userRepository->findAll())
24     ];
25 }
26
27 public function reset()
28 {
29     $this->data = [];
30 }
31 public function getName()
32 {
33     return 'toolbar';
34 }
35 public function getMethod()
36 {
37     return $this->data['method'];
38 }
39
40 public function getAcceptableContentTypes()
41 {
42     return $this->data['acceptable_content_types'];
43 }
44 public function getCountUsers()
45 {
46     return $this->data['countUsers'];
47 }
48 public function getUsers()
49 {
50     return $this->data['users'];
51 }
52 }

```

Décortiquons le code ci-dessus :

Nous créons une classe `UserListCollector` qui hérite de `DataCollector` qui fournit 3 méthodes :

- `collect()` : méthode qui stocke les données récupérées dans le tableau `$this->data`. « *method* » et « *acceptable\_content\_types* » doivent être présentés comme décrit dans l'exemple. Vous devez ensuite définir vos besoins. Ici, nous récupérons grâce à `UserRepository` que nous avons appelé dans le constructeur la liste des utilisateurs dans « *users* » et le nombre d'utilisateurs dans « *countUsers* ».
- `reset()` : sert à réinitialiser l'état du Profiler à chaque rafraîchissement de la page.
- `getName()` : retourne l'identifiant unique du collecteur dans l'application.

Pour le Web Profiler, il faut ajouter pour chaque variable qu'il y a dans `$this->data` une méthode `get` du même nom qui renvoie le tableau de `$this->data`. Dans notre exemple, `getMethod()`, `getAcceptableContentTypes()`, `getUsers()` et `getCountUsers()`.

Dans le fichier services.yaml de config, ajoutez cela :

```
1 App\DataCollector\UserListCollector:
2   arguments: [ '@App\Repository\UserRepository' ]
3   tags:
4     -
5       name:      data_collector
6       template:  'data_collector/countUsers.html.twig'
7       id:        'toolbar'
```

Dans ce fichier yaml, nous avons déclaré les arguments que nous avons utilisés, le template et l'id que nous avons retournés dans UserListCollector.

La dernière étape est donc de créer le template :

```
1 {% extends '@WebProfiler/Profiler/layout.html.twig' %}
2
3 {% block toolbar %}
4   {% set icon %}
5     
6     <span class="sf-toolbar-value">{{ collector.countUsers }}</span>
7   {% endset %}
8
9   {{ include('@WebProfiler/Profiler/toolbar_item.html.twig') }}
10 {% endblock %}
11
12 {% block menu %}
13   <span class="label">
14     <span class="icon">  </span>
15     <strong>CountUsers</strong>
16   </span>
17 {% endblock %}
18
19 {% block panel %}
20   <div class="row">
21     <div class="col-md-6">
22       <h2>User</h2>
23       <table>
24         <tr>
25           <th>Id</th>
26           <th>Email</th>
27         </tr>
28         {% for user in collector.users %}
29           <tr>
30             <td>{{ user.id }}</td>
31             <td>{{ user.email }}</td>
32           </tr>
33         {% endfor %}
34       </table>
35     </div>
36   </div>
37 {% endblock %}
```

Dans ce template, nous appelons @WebProfiler pour travailler avec le Profiler. Nous avons différents blocs :

- Bloc toolbar : nous appelons une icône ainsi que le nombre d'utilisateurs à l'aide de collector.countUsers, élément que nous avons fait passer dans \$this-data.
- Bloc menu : représente le menu que nous avons à gauche dans Web Profiler, nous appelons l'icône ainsi que son nom.

- Bloc panel : page de visualisation dans Web Profiler. Nous créons un tableau qui liste les utilisateurs avec leur id et email.

### C. Exercice : Quiz

[solution n°2 p.20]

#### Question 1

Quelle information retrouve-t-on dans l'encadré Request Attributes ?

- ☐ Le nom de la route de la requête
- ☐ Le Token de la requête
- ☐ La méthode de la requête

#### Question 2

L'onglet Routing du Profiler contient toutes les routes de l'application.

- ☐ Vrai
- ☐ Faux

#### Question 3

Les méthodes `var_dump()` et `dump()` n'ont rien à voir.

- ☐ Vrai
- ☐ Faux

#### Question 4

Le Profiler influence-t-il les performances de l'application ?

- ☐ Vrai
- ☐ Faux

#### Question 5

La méthode `dump()` retransmet les informations de façon visuelle et graphique.

- ☐ Vrai
- ☐ Faux

## IV. Essentiel

L'information est essentielle pour un développeur et il est difficile de devoir tout surveiller en même temps. Nous voulons avoir à la fois un code de qualité et performant. Nous voulons connaître, lorsque nous ajoutons de nouvelles fonctionnalités à notre application web, les répercussions sur celle-ci.

Le Profiler nous offre tout cela de façon claire, graphique et intuitive. La Toolbar nous tient au courant des informations essentielles en un clin d'œil et lorsque l'on ouvre la page du Profiler, nous avons accès à un nombre d'informations incroyable. À travers ce cours, nous avons commencé à nous familiariser avec le Profiler, découvert la façon de l'utiliser et d'accéder à toutes ces informations et nous avons découvert les onglets installés par défaut par Symfony et les plus utilisés.

## 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 en version LTS 5.4. Pour la première fois, on vous confie à la formation d'un tout nouveau développeur junior qui a des notions de PHP mais qui n'a jamais travaillé sur Symfony.

#### Question 1

[solution n°3 p.21]

Pour son premier jour avec vous, vous avez pour mission d'expliquer à ce nouveau développeur junior comment installer le Profiler. Expliquez comment vous allez vous y prendre.

#### Question 2

[solution n°4 p.21]

Votre deuxième objectif est de lui expliquer à quoi sert le Profiler de Symfony et comment lire la Toolbar. Expliquez comment vous pourriez vous y prendre.

### B. Test

#### Exercice 1 : Quiz

[solution n°5 p.21]

##### Question 1

Qu'indique l'encadré « *macro calls* » de Twig Metrics ?

- ☐ Le nombre de pages Twig
- ☐ Le nombre de doubles parenthèses Twig
- ☐ Le nombre de fonctions Twig

##### Question 2

Dans quel onglet du Profiler trouve-t-on des informations sur les variables de session ?

- ☐ Events
- ☐ Security
- ☐ Request/response

##### Question 3

Les logs sont-ils des fichiers qui contiennent l'historique des événements ?

- ☐ Vrai
- ☐ Faux

##### Question 4

Est-ce que le sous-onglet Flashes existe dans le Profiler Web ?

- ☐ Vrai
- ☐ Faux

##### Question 5



Quelle méthode pour supprimer un objet de la base de données Doctrine fournit-il lors de création du Repository ?


- ☐ dump()
- ☐ remove()
- ☐ log()

## Solutions des exercices




**Exercice p. 6 Solution n°1****Question 1**

La Toolbar permet de connaître la version de Symfony avec lequel l'application est développée.

- ☒ Vrai
- ☐ Faux
-  Vous retrouverez cette information à droite de la Toolbar.


**Question 2**

Quelle est la bonne ligne de commande pour installer le Profiler ?

- ☒ `$ composer require debug --dev`
- ☐ `$ composer require debug/profiler --dev`
- ☐ `$ composer require --dev profiler-pack`
-  Cette ligne de commande permet d'installer plusieurs outils de débogage dont le Profiler. Il est possible d'installer seul le Profiler avec cette ligne de commande : « `$ composer require --dev symfony/profiler-pack` ».


**Question 3**

Où peut-on accéder à l'outil Profiler dans Symfony ?

- ☐ Dans la console Symfony
- ☒ Dans le navigateur web
- ☐ Dans un IDE
-  La barre de débogage ainsi que les autres outils de débogage ne sont accessibles qu'à partir d'un navigateur web.


**Question 4**

Le Profiler génère un token pour chaque requête.

- ☒ Vrai
- ☐ Faux
-  Vous pourrez ainsi récupérer des informations du Profiler dans votre code PHP de par ce token.

**Question 5**


Dans Profile Search, il est possible de rechercher par numéro de statut HTTP un résultat de requête.

- ☒ Vrai
- ☐ Faux
-  Il existe de nombreux filtres de recherche dans cette section.

## Exercice p. 15 Solution n°2


### Question 1

Quelle information retrouve-t-on dans l'encadré Request Attributes ?

- ☐ Le nom de la route de la requête
- ☒ Le Token de la requête
- ☐ La méthode de la requête
-  Vous y trouverez aussi le(s) paramètre(s) de la route s'il y en a.


### Question 2

L'onglet Routing du Profiler contient toutes les routes de l'application.

- ☒ Vrai
- ☐ Faux
-  Y compris la route qui matches avec la requête.


### Question 3

Les méthodes var\_dump() et dump() n'ont rien à voir.

- ☐ Vrai
- ☒ Faux
-  La méthode dump() qu'utilise Symfony est une méthode donnée à var\_dump().


### Question 4

Le Profiler influence-t-il les performances de l'application ?

- ☒ Vrai
- ☐ Faux
-  Le Profiler est utile en mode de développement, même s'il utilise des ressources de l'application.

### Question 5

La méthode dump() retransmet les informations de façon visuelle et graphique.

- ☒ Vrai
- ☐ Faux
-  Les informations sont présentées de façon beaucoup plus visuelle avec des jeux de couleurs pour aider à une meilleure compréhension de celles-ci.

**p. 16 Solution n°3**

Déjà, vous allez lui rappeler que le Profiler est un outil de développement qui ne doit s'utiliser qu'en mode de développement voire de test, car on ne souhaite pas qu'un utilisateur puisse avoir des informations sur l'application, cela nuirait aux performances de celle-ci, de plus, cela pourrait ouvrir à des failles de sécurité.

Vous pourrez lui montrer dans le fichier `.env` qu'il existe une variable d'environnement qui définit le mode de travail, à savoir développement, test ou production. Puis, vous expliquerez comment installer le Profiler à l'aide du terminal. Vous insisterez sur l'importance de vérifier dans la ligne de commande qu'il y a bien « `- dev` » et qu'il est possible bien sûr de voir dans le fichier `composer.json` les bundles et packages installés dans « `require-dev` », tableau qui contient uniquement les librairies qui ne seront donc utilisées qu'en mode de développement.

**p. 16 Solution n°4**

Vous pourriez commencer à lui dire que le Profiler de Symfony est un outil d'aide au développement. Son but est de donner le maximum d'informations sur l'application et d'aider au débogage. Le Profiler est composé de deux éléments principaux : une barre de débogage appelée Toolbar et une application web de profilage, appelée Profiler Web.

La Toolbar apporte de nombreuses informations que l'exécution de l'application a récoltées. Plus l'application possède de fonctionnalités, plus il est probable qu'elle aura de plus en plus d'icônes. Il est possible de survoler chaque icône et elles donneront davantage d'informations.

Pour ouvrir le Profiler Web, il suffit de cliquer sur la Toolbar. La version de Symfony est toujours inscrite à droite de la barre, et à sa gauche il y a le numéro de la réponse HTTP en couleur vert ou rouge suivant son numéro. Vous pourrez lui décrire une à une chaque icône ou du moins les plus importantes informations à retenir de la Toolbar.

**Exercice p. 16 Solution n°5****Question 1**

Qu'indique l'encadré « *macro calls* » de Twig Metrics ?

- ☐ Le nombre de pages Twig
- ☐ Le nombre de doubles parenthèses Twig
- ☒ Le nombre de fonctions Twig
- ☐ Les macros sont des équivalents de fonctions PHP en Twig.


**Question 2**

Dans quel onglet du Profiler trouve-t-on des informations sur les variables de session ?

- ☐ Events
- ☐ Security
- ☒ Request/response
- ☐ Vous retrouverez ces informations dans le sous-onglet Session.

**Question 3**


Les logs sont-ils des fichiers qui contiennent l'historique des événements ?

- ☒ Vrai
- ☐ Faux
-  Les logs s'apparentent à un journal de bord horodaté.

#### Question 4

---


Est-ce que le sous-onglet Flashes existe dans le Profiler Web ?

- ☒ Vrai
- ☐ Faux
-  Flashes indique tous les messages flashes que l'application vient de retransmettre.

#### Question 5

---

Quelle méthode pour supprimer un objet de la base de données Doctrine fournit-il lors de création du Repository ?

- ☐ dump()
- ☒ remove()
- ☐ log()
-  remove() est une méthode qui va permettre de supprimer un objet de la base de données.