e-application - 2020

4. Plus loin avec Symfony5

Matthieu Kowalczyk
Ingénieur en technologies de l'information @CGI

Nathan Levy
Ingénieur en technologies de l'information @CGI

matthieu.kowalczyk@cgi.com
nathan.levy@cgi.com

CGI

UNIVERSITÉ DE MONTPELLIER

Agenda

1. Sécurité
2. Performance
3. Formulaires
4. Validation
5. TP n°6
6. Encore plus loin...

A rendre pour le :
04/01/2021
2 personnes max

Description du projet final

Dans le cadre de ce cours et afin de mettre en pratique l'utilisation du framework Symfony5.
Celui-ci devra être héberger sur la **plateforme Cloud Heroku**.

Le blog devra être en HTLM5 (**sémantique** et **valide**). Utilisant un **framework CSS** (Bootstrap, pure, etc).

Le blog comprend 2 contrôleurs (Blog & Crud) et 5 routes (Action) :

- **indexAction =>** page d'accueil (/) listant les x derniers billets (page à page),
- **postAction =>** page billet (/post/$id) affichant le contenu d'un bille
- **newAction =>** Ajout
- **editAction =>** Edition
- **deleteAction =>** Suppression

Le blog comprend une Entity Article contenant les champs suivants :

- **title** (titre de l'article)
- **slug** (le slug qui va servir d'alias d'url basé sur le titre de l'article)
- **content** (le contenu de l'article)
- **published** (la date de création de l'article)

Le blog rends disponible une API REST qui permet de récupérer les 5 derniers articles de notre blog.
Le blog utilise une API REST afin de récupérer des données (les 5 derniers articles du blog d'un autre groupe par exemple).

Le blog implémente **le SecurityBundle de Symfony** pour se connecter (la page de login **doit être habillée**) et les routes CRUD doivent être protégées.
Le blog implémente **StofDoctrineExtensionsBundle** pour rentre vos articles timestampable et sluggable.

Quoi rendre ?

Ajouter sur Heroku dans la rubrique « Access », les 2 collaborateurs :

- nathan.levy.um2@gmail.com
- matthieu.kowalczyk.um2@gmail.com

- Ajouter dans votre projet un fichier README.md :
  - Nom & Prénom des participants au projet (**2 maximum**),
  - Compte et accès (pour la partie administrateur),
  - Commentaires,
  - Appréciation sur le cours pour l'année suivante (facultatif).

Si vous ne réussissez pas à déployer votre blog sur la plateforme Heroku **(Le déploiement sur heroku compte dans la note final)** vous pouvez nous l'envoyer par mail.

- Un fichier um_2020_NOM1_NOM2.tar.gz avec :
  - Sources (sans cache mais avec les vendors)
  - Dump SQL (MySQL, PostGre, etc.) ou fichier SQLite.

Security

- Symfony intègre un bundle Security qui permet de gérer la sécurité de notre application.
- Il permet de créer facilement l'entité User.
- Il permet de créer facilement des formulaire d'authentification (Form).
- Il permet de créer et définir les différents rôles de nos utilisateurs (security.yml)
- Il permet de gérer facilement les droits et accès aux différentes parties de notre application (security.yml, annotations sur nos routes).
- Le Symfony maker permet de génerer facilement les différentes classes necessaire a l'authentification.
- Plus sur le documentation

security.yml

```yaml
# config/packages/security.yaml
security:
    # ...

    firewalls:
        # ...
        main:
            # ...

    access_control:
        # require ROLE_ADMIN for /admin*
        - { path: '^/admin', roles: ROLE_ADMIN }

        # or require ROLE_ADMIN or IS_AUTHENTICATED_FULLY for /admin*
        - { path: '^/admin', roles: [IS_AUTHENTICATED_FULLY, ROLE_ADMIN] }

        # the 'path' value can be any valid regular expression
        # (this one will match URLs like /api/post/7298 and /api/comment/528491)
        - { path: ^/api/(post|comment)/\d+$, roles: ROLE_USER }
```

Sécurité : Injection SQL

- Injection SQL : consiste à altérer une requête SQL pour en modifier l'impact :

SELECT uid FROM Users WHERE name = '(nom)' AND password = '(mot de passe hashé)';

Devient :

SELECT uid FROM Users WHERE name = 'Dupont' -- ' AND password = '4e383a1918b432a9bb7702f086c56596e';

La réponse de Symfony : utilisation de Doctrine dans les règles de l'art (*setParameter*).

Sécurité : XSS

Le cross-site scripting (abrégé *XSS*), est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, permettant ainsi de provoquer des actions sur les navigateurs web visitant la page.

Les possibilités des XSS sont très larges puisque l'attaquant peut utiliser tous les langages pris en charge par le navigateur (JavaScript, Java, Flash...) et de nouvelles possibilités sont régulièrement découvertes notamment avec l'arrivée de nouvelles technologies comme HTML5.

Il est par exemple possible de rediriger vers un autre site pour de l'hameçonnage ou encore de voler la session en récupérant les cookies.

La réponse de Symfony :

- **Dans Twig :** l'échappement est **par défaut**. Pour le désactiver :
{{ article.body | raw }}

Ne jamais faire confiance à une donnée de type user input !

Sécurité : CSRF

Les attaques de type Cross-Site Request Forgery (abrégées CSRF prononcées sea-surfing ou parfois XSRF) utilisent l'utilisateur comme déclencheur, celui-ci devient complice sans en être conscient. L'attaque étant actionnée par l'utilisateur, un grand nombre de systèmes d'authentification sont contournés.

La réponse de Symfony : là encore, si Symfony est utilisé dans les règles de l'art en utilisant le composant « *form* », la protection et la génération de token est automatique.

Aller plus loin.

Plus loin dans la sécurité : OWASP

- Ne péchez jamais par orgueil ! Vérifiez que vous n'êtes vulnérable à aucune faille !
- Il existe des standards pour le web : OWASP , qui propose un TOP 10 des vulnérabilités.
- Il existe des outils : Zed Attack Proxy (ZAP) qui va se comporter comme un proxy entre vous et votre site afin de le tester. Le programme est en mesure également d'agir seul et de faire du « *brute force* ».
- Une faille de sécurité peut avoir des effets néfastes sur le business ou la réputation :
  - LinkedIn manque de sel et stock les mot de passe presque en clair
  - Parti politique qui autorise le « Directory indexes » sur son répertoire d'upload et laisse fuiter la liste de ses donateurs.

2 – Performance

- L'OPCache
- Le cache utilisateur
- Le cache HTTP

Le cache HTTP

- Le moyen le plus efficace d'améliorer les performances d'une application est de mettre en cache l'intégralité d'une réponse pour ne plus avoir à rappeler l'application pour les requêtes suivantes. Bien sûr, ce n'est pas toujours possible pour les sites web fortement dynamiques, ou peut être que si...
- Le système de cache de Symfony5 est différent, car il se base sur la simplicité et la puissance du **cache HTTP** tel qu'il est défini dans la spécification HTTP. Au lieu de réinventer un processus de mise en cache, Symfony4 adopte la norme qui définit la communication de base sur le Web. Une fois que vous avez compris les fondamentaux de la validation HTTP et de l'expiration de la mise en cache, vous serez prêts à maîtriser le système de cache de Symfony2.

Le cache HTTP & la passerelle

- Une passerelle de cache, ou reverse proxy, est une couche indépendante qui se trouve devant votre application.
- La passerelle met en cache les réponses telles qu'elles sont retournées par l'application et répond aux requêtes avec les réponses qui sont en cache avant qu'elles n'atteignent l'application.
- Symfony5 possède sa propre passerelle par défaut.
- Mais n'importe quelle autre peut être également utilisée :
  - Varnish,
  - Squid.

Le cache HTTP & les entêtes

- Les entêtes du cache HTTP sont utilisés pour communiquer avec la passerelle de cache et tout autre cache entre votre application et le client. Symfony5 en propose par défaut et fournit une interface puissante pour interagir avec eux.
- HTTP définit quatre entêtes de cache :
  - Cache-Control
  - Expires
  - Etag
  - Last-Modified

```php
// ...

use Symfony\Component\HttpFoundation\Response;

$response = new Response();

// marque la réponse comme publique ou privée
$response->setPublic();
$response->setPrivate();

// définit l'âge max des caches privés ou des caches partagés
$response->setMaxAge(600);
$response->setSharedMaxAge(600);

 // définit une directive personnalisée du Cache-Control
$response->headers->addCacheControlDirective('must-revalidate', true);
```

Le cache HTTP & l'expiration

- L'expiration et la validation HTTP sont les deux modèles utilisés pour déterminer si le contenu d'un cache est valide (peut être réutilisé à partir du cache) ou périmé (doit être régénéré par l'application).
- Avec le modèle d'expiration, on spécifie simplement combien de temps une réponse doit être considérée comme « valide » en incluant un entête Cache-Control et/ou Expires. Les systèmes de cache qui supportent l'expiration enverront la même réponse jusqu'à ce que la version en cache soit expirée et devienne « invalide ».

```php
$date = new DateTime();
$date->modify('+600 seconds');

$response->setExpires($date);
```

Le cache HTTP & la validation

- Quand une page est dynamique (c-a-d quand son contenu change souvent), le modèle de validation est souvent nécessaire. Avec ce modèle, le système de cache stocke la réponse, mais demande au serveur à chaque requête si la réponse est encore valide. L'application utilise un identifiant unique (l'entête Etag) et/ou un timestamp (l'entête Last-Modified) pour vérifier si la page a changé depuis sa mise en cache.

```php
public function indexAction()
{
    $response = $this->render('MyBundle:Main:index.html.twig');
    $response->setETag(md5($response->getContent()));
    $response->setPublic(); // permet de s'assurer que la réponse est publique, e
    $response->isNotModified($this->getRequest());

    return $response;
}
```

ESI et Hinclude

- Les Edge Side Includes (ESI) autorisent le cache HTTP à mettre en cache des fragments de pages (voir des fragments imbriqués) de façon indépendante. Avec les ESI, vous pouvez même mettre en cache une page entière pendant 60 minutes, mais un bloc imbriqué dans cette page uniquement 5 minutes.
- Les contrôleurs peuvent être imbriqués de façon asynchrone avec la bibliothèque javascript hinclude.js. Comme le contenu imbriqué vient d'une autre page (et d'un autre contrôleur), Symfony4 utilise le helper standard render pour configurer les tags hinclude.

Aller plus loin :
•Cache component de Symfony
•Contenu asynchrone avec hinclude.js

3 – Formulaires

- Création de formulaire
- Lien avec les entités
- Passage au templates

Le form FormBuilder en relation avec l'entité

- Symfony5 propose un « *FormBuilder* » qui, relié à une entité, va facilement vous permettre de construire un formulaire de création ou d'édition (CRUD).
- Prenons l'exemple de l'entité suivante :

```php
// src/Entity/Task.php
namespace App\Entity;

class Task
{
    protected $task;
    protected $dueDate;

    public function getTask()
    {
        return $this->task;
    }

    public function setTask($task)
    {
        $this->task = $task;
    }

    public function getDueDate()
    {
        return $this->dueDate;
    }

    public function setDueDate(\DateTime $dueDate = null)
    {
        $this->dueDate = $dueDate;
    }
}
```

Utiliser la méthode createFormBuilder

- Une fois l'entité correctement écrite, il ne reste plus qu'à invoquer la méthode *createFormBuilder()*,
- Puis de passer le formulaire à la vue,
- Et pour finir afficher le formulaire dans la vue via une fonction Twig :
  `{{ form(form) }}`

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

use App\Entity\Task;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\Form\Extension\Core\Type\DateType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;
use Symfony\Component\Form\Extension\Core\Type\TextType;
use Symfony\Component\HttpFoundation\Request;

class TaskController extends AbstractController
{
    public function new(Request $request)
    {
        // creates a task object and initializes some data for this example
        $task = new Task();
        $task->setTask('Write a blog post');
        $task->setDueDate(new \DateTime('tomorrow'));

        $form = $this->createFormBuilder($task)
            ->add('task', TextType::class)
            ->add('dueDate', DateType::class)
            ->add('save', SubmitType::class, ['label' => 'Create Task'])
            ->getForm();

        // ...
    }
}
```

Aller plus loin

- Sf5 intègre nativement un grand nombre de FormType.
- Le layout général des formulaires est personnalisable.
- Il est possible de faire des formulaires imbriqués.
- Le composant Form est intimement lié à Doctrine.
- La fonction Twig *form* vue précédemment peut également être découpée en plusieurs fonctions :

```
{{ form_start(form) }}
    {{ form_errors(form) }}

    {{ form_row(form.name) }}
    {{ form_row(form.dueDate) }}

    {{ form_row(form.submit, { 'label': 'Submit me' }) }}
{{ form_end(form) }}
```

Aller plus loin : Forms

4 – Validation

- La validation au sein des entités
- Validation au sein des formulaires

Le lien entité / validation

- Lors de la sauvegarde d'une entité en base, il est possible de contrôler que les informations renseignées sont correctes. C'est ce procédé que Symfony5 appelle la validation.
- Il existe plusieurs formats de validation (YAML, Annotations, XML & PHP). La validation étant liée à l'entité, je vous conseille fortement d'utiliser les annotation au sein de vos entités.

```php
// src/Entity/Author.php
namespace App\Entity;

// ...
use Symfony\Component\Validator\Constraints as Assert;

class Author
{
    /**
     * @Assert\NotBlank
     */
    private $name;
}
```

Valider une entité

- Une fois l'entité créé, il devient simple de la vérifier à partir d'un contrôleur.
- Pour cela on utilise le service validator.

```php
// ...
use App\Entity\Author;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Validator\Validator\ValidatorInterface;

// ...
public function author(ValidatorInterface $validator)
{
    $author = new Author();

    // ... do something to the $author object

    $errors = $validator->validate($author);

    if (count($errors) > 0) {
        /*
         * Uses a __toString method on the $errors variable which is a
         * ConstraintViolationList object. This gives us a nice string
         * for debugging.
         */
        $errorsString = (string) $errors;

        return new Response($errorsString);
    }

    return new Response('The author is valid! Yes!');
}
```

Rediriger les erreurs dans l'IHM

- Il est également possible de rediriger les erreurs dans la vue pour les afficher :

```php
if (count($errors) > 0) {
    return $this->render('author/validation.html.twig', [
        'errors' => $errors,
    ]);
}
```

```twig
{# templates/author/validation.html.twig #}
<h3>The author has the following errors</h3>
<ul>
{% for error in errors %}
    <li>{{ error.message }}</li>
{% endfor %}
</ul>
```

Validation au sein des formulaires

- Lorsque vous utilisez le composant formulaire pour générer la couche CRUD de votre entité, la validation de l'entité est automatiquement prise en compte.
- Vous n'avez rien de plus à faire !

Aller plus loin

- Sf5 intègre nativement un grand nombre de contraintes.
- Les messages d'erreurs sont traduisibles.
- Il est possible de configurer des groupes de validation.

Aller plus loin : Validation.

5 – TD n°6

- Création de l'entité User
- Création du formulaire d'authentification
- Mise en place du CRUD pour nos articles

Mise en place du security.yml

- Lisez la documentation Security de Symfony,
- Appliquez la et suivez les liens et instructions pour la créations de form login, user (pensez au maker bundle)

Buts :

- Avoir une entité User enregistrée en base avec des rôles, et droits d'accès particuliers.
- Apprendre à lire / utiliser / appliquer une documentation.

6 – Encore plus loin !

- Les tests dans Sf5
- Les Commands
- Envoyer des mails
- Loguer avec monolog
- Webpack
- Fixtures
- Bonnes pratiques

Les tests

- L'une des force de Symfony est sa couverture de tests et le fait de pouvoir facilement tester nos applications via l'encapsulation du framework de tests PHP Unit.
- Sans surprise Sf5 permet la réalisation de tests unitaires.
- Mais la force de Symfony est d'embarquer un client permettant de faire des requêtes et par là des tests fonctionnels (simuler des clicks vers des liens ou des soumission de formulaire) !

Aller plus loin : Les Tests.

Console Commands

- Symfony permet d'ajouter vos propres commands que vous pourrez utiliser grâce à la console Symfony.
- Cela peut notamment vous êtes utiles lors de la réalisation de CRON.
- Le bundle embarque un systeme de barre de progression et autre différents éléments utiles en output.

Aller plus loin : Comment créer une commande pour la Console.

Envoyer des mails avec SwiftMailer

- Pour ce qui est de l'envoi de mail, Symfony5 utilise la librairie SwiftMailer.
- Cette librairie permet de s'interfacer avec un serveur SMTP.
- Elle permet d'envoyer des mails complexes avec pièces jointes très facilement.

Aller plus loin : Comment envoyer un Email

Monolog et les logs

- Symfony utilise la librairie monolog pour la gestion de ses logs.
- Cette solution est complètement paramétrable.
- Elle peut être configurée tant au niveau de la verbosité que de la sévérité.
- La rotation des logs est possible.

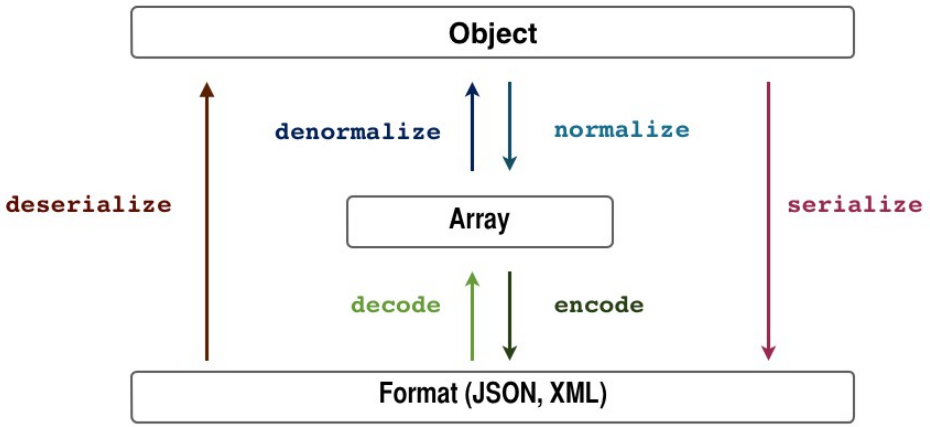
Aller plus loin : Comment utiliser Monolog pour écrire des Logs.

Webpack Encore

- Symfony embarque une library JS appelée Webpack Encore qui vous permet de facilement gérer votre CSS et votre JS.
- Webpack intégère aussi un preprocesseur et un compilateur CSS / JS.
- Webpack Encore fonctionne avec npm (ou yarn).
- Voir la documentation

Serializer

- Un service (désactivé par défaut) pour *serialiser*.
- Possibilité de normaliser.
- Possibilité de convertir (ex : camel_case_to_snake_case).
- Possibilité de créer ses propres encodeurs.
- Possibilité de mettre en cache (APCu).



Aller plus loin : Serializer.

Les événements

- Symfony offre la possibilité d'écouter (*listener*) ou de souscrire (*subscriber*) à des évènements (*event*).
- Symfony intègre par défaut plusieurs évènements.
- Mais il est également possible de créer ses propres évènements

Aller plus loin : Event dispatcher.

Bonne pratiques Sf5

Pour finir, je vous conseille cette lecture :

Bonnes pratiques pour Symfony 5

Avec un focus sur

- Ne créez pas de multiple bundle pour structurer la logique de votre application,
- Utilisez les paramètres d'environnements,
- N'oubliez pas le fichier .dist,
- Stockez vos assets dans le répertoire web.

Webographie

- Site officiel de Symfony : http://symfony.com/
- Le manuel PHP : http://php.net/manual/fr/index.php
- La doc : http://symfony.com/doc/current/book/index.html

Bibliographie

- Toutes la documentation et les « books » officiels sont librement téléchargeables
  http://symfony.com/doc/current/index.html

Licence

La documentation Symfony5 étant bien faite et sous Licence libre, des illustrations tout comme des bouts de textes de ce cours en sont extraits.