Travaux dirigés

Happy - API - Birthday

Partie 1

Objectifs :

Création d'une API de gestion de dates d'anniversaire qui fonctionnera sur le même principe que l'application développée précédemment.

Prérequis :

Connaissance de la théorie sur les API REST

Simple CRUD

Etapes de réalisation du CRUD

Installation framework symfony

En mode skeleton

Création d'une entité « Birthday »

2 champs : name et birthday (type date mutable)

Création des fixtures

En s'appuyant sur le bundle tiers foundry

```
composer require zenstruck/foundry --dev
```

Utilitaires

Postman

On utilisera postman pour tester les requêtes

Profiler

A tout moment, installer le profiler pour débugger.

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

Visualiser les url à l'adresse : http://127,0,0,1:8000/_profiler

Création des routes

- ### List

Utiliser JMS Serializer pour envoyer les données en Json

- ### Show

Utiliser le ParamConverter de Symfony

- ### Delete

Utiliser le verbe HTTP delete

- ### Create

Options d'envoi des données dans postman

- raw
- form-data
- x-www-form-url-encoded

Récupérer les données postées en PHP

```
$data = json_decode($request->getContent(), true);
```

ou

```
$data = $request→getPayload()→all();
```

Hydrater l'objet en PHP

Manuellement ou via un formulaire, comme ceci :

```
$birthday = new Birthday();
$form = $this->createForm(BirthdayType::class, $birthday);
$form->submit($data);
```

Dans ce cas :

Créer préalablement la classe de formulaire **BirthdayType** et préciser dans les options

```
'csrf_protection' => false,
```

- ### Update

2 méthodes possibles : PUT ou PATCH

Hydrater l'objet avec un formulaire

```
$birthday = new Birthday();
$form = $this->createForm(BirthdayType::class, $birthday);
```

```
        $form->submit($data, false);
```

clearMissing
NB : le second paramètre (false) de la méthode submit permet de conserver les données de l'objet originel, si celles-ci ne sont pas fournies dans la requête.

Gestion des erreurs HTTP : 404, 500

Implémenter un Event subscriber sur l'événement kernel.exception afin de retourner une réponse au format Json

Types d'erreurs	Code
Symfony\Component\HttpKernel\Exception\HttpExceptionInterface	401, 403, 404 ...
Doctrine\DBAL\Exception\NotNullConstraintViolationException	500
\ErrorException	500
A compléter ...	

```
public function onKernelException(ExceptionEvent $event): void
{
  $exception = $event->getThrowable();
  $statusCode = $exception instanceof HttpExceptionInterface ? $exception->getStatusCode() : 500;
  $event->setResponse(new JsonResponse(['error' => $exception->getMessage()], $statusCode));
}
```

Aller plus loin :

- Loguer les exceptions

Contrôle de validation des données

Par défaut, les erreurs SQL seront détectés (champs non null), mais on peut mettre en place un contrôle de validation personnalisé. Exemples

Autoriser les noms ayant au moins deux lettres ----- facile -----

Modifier l'entité pour poser la contrainte et modifier le controleur pour vérifier que la contrainte est respectée
https://symfony.com/doc/current/validation.html

Auoriser uniquement les dates passées ----- difficile -----

Créer une contrainte personnalisée

https://symfony.com/doc/current/validation/custom_constraint.html

Les variables d'environnement dans Postman

Remplacer l'url de base http://127.0.0.1:8000 par une variable d'environnement

Etapes

- Créer un environnement « API - Birthday »
- Y ajouter une variable (par ex base_uri), avec la valeur http://127.0.0.1:8000
- Rassembler les routes dans un dossier « api ». Associer un environnement à ce dossier
- Associer ce dossier à l'environnement ainsi créé
- Enfin, utiliser {{base_uri}} dans les adresses de requête

Authentification des utilisateurs

Objectif :
S'authentifier pour utiliser l'application.
Les utilisateurs ne peuvent pas consulter les anniversaires créés par d'autres utilisateurs.

User Entity

Installation du security bundle

Création de l'entité : Lancer la commande make:user et répondre aux questions suivantes, en laissant les valeurs par défaut :

```
The name of the security user class (e.g. User) [User]:
>

Do you want to store user data in the database (via Doctrine)? (yes/no) [yes]:
>

Enter a property name that will be the unique "display" name for the user (e.g. email, username, uuid) [email]:
>

Will this app need to hash/check user passwords? Choose No if passwords are not needed or will be checked/hashed by some other system (e.g. a single sign-on server).

Does this app need to hash/check user passwords? (yes/no) [yes]:
>
```

Lancer les migrations

```
bin/console make:migration
bin/console doctrine:migrations:migrate
```

Création de fixtures

Penser à chiffrer les mots de passe avec le service UserPasswordHasher.

```
bin/console make:factory
bin/console make:fixtures
bin/console doctrine:fixtures:load
```

Login

Json Web Token

Lorsu'un utilisateur envoi ses informations de login, le serveur va générer un JWT et le transmettre au client. Celui-ci devra le joindre à chaque requête afin d'être authentifié par le serveur.

Pour en savoir plus sur les JWT :

https://openclassrooms.com/fr/courses/7709361-construisez-une-api-rest-avec-symfony/7795148-authentifiez-et-autorisez-les-utilisateurs-de-l-api-avec-jwt
https://www.vaadata.com/blog/fr/jetons-jwt-et-securite-principes-et-cas-dutilisation/

Lexik jwt bundle

https://symfony.com/bundles/LexikJWTAuthenticationBundle/current/index.html
https://helmi-bejaoui.medium.com/a-beginners-guide-on-jwt-authentication-symfony-5-api-based-bd6622bfe975

Installer

```
composer require "lexik/jwt-authentication-bundle"
```

Créer une paire de clef

```
bin/console lexik:jwt:generate-keypair
```

Modifier la configuration du fichier **security.yaml** ainsi que celle du fichier **routing.yaml** comme indiqué dans la doc

A ce stade les routes d'API retournent une erreur HTTP 401: « unauthorized »

Login

Envoyer une requête POST sur l'url http://127.0.0.1:8000/api/login_check
en indiquant les paramètres issus des fixtures :

```
{
    "username": "sporer.dedrick@corwin.com",
    "password": "password"
```

```
}
```

Le serveur retourne le JWT

Analyser le JWT

https://jwt.io/

Ajouter le JWT aux requêtes sur Postman

Manuellement

Dans l'en-te de la requête ajouter le champ
« Authorization » avec la valeur « bearer xxxxxxxxxxxxxxxxxxxxxxxx »

Automatiquement

1. Créer une variable d'environnement « token » vide
2. Dans le dossier « api » configurer l'authorization en spécifiant Bearer token
3. Dans la zone « Test » ajouter ce script :

```
const {token} = JSON.parse(responseBody)
postman.setEnvironmentVariable("token",token)
```

DOC : https://apidog.com/articles/how-to-set-bearer-token-in-postman/

Register

Implémenter une classe de formulaire pour l'entité User
Implémenter une route de contrôleur pour gérer la création d'un utilisateur
Tester dans Postman

Le problème du champ « rôles »

Si les rôles sont envoyés sous forme d'**Array :**

```
"roles": ["ROLE_ADMIN"]
```

le formulaire n'accepte pas le type (uniquement **String** ou **Number**) => **le formulaire n'est pas validé.**
=> Il est nécessaire d'envoyer une **String :**

```
 "roles": "['ROLE_ADMIN']"
```

Le formulaire est valide, mais … l'entité demande un type **Array** => **l'entité n'est pas validée.**

Solution :

Il faut envoyer les valeurs sous forme de chaîne :

```
"roles": " ROLE_ADMIN, ROLE_USER "
```

Et créer un **data Transformer** https://symfony.com/doc/current/form/data_transformers.html, dans le formType :

```
$builder->get('roles')
       ->addModelTransformer(new CallbackTransformer(
           function ($rolesAsArray): string {
               // transform the array to a string
               return implode(', ', $rolesAsArray);
           },
           function ($rolesAsString): array {
               // transform the string back to an array
               return explode(', ', $rolesAsString);
           }
       ))
   ;
```

A venir dans la seconde partie ...

Relation OneToMany

Associer un utilisateur aux dates d'anniversaire qu'il ajoute dans l'application

Contrôle d'accès

Permettre à l'utilisateur d'interagir uniquement avec les dates d'anniversaires qu'il a ajouté dans l'application.

Hyperliens

Rendre sont API auto-découvrable, c'est à dire de retourner, en plus des données des liens vers les « endpoints » qui permettront de naviguer dans l'application

Documentation

Rendre son application développeur – friendly pour celleux qui ont la charge de développer le « front »

Documentation

Sur la sécurité en général

https://symfonycasts.com/screencast/symfony-security

https://symfony.com/doc/current/security.html

https://symfony.com/doc/current/security/access_token.html
https://openclassrooms.com/fr/courses/5489656-construisez-un-site-web-a-l-aide-du-framework-symfony-5/5654131-securisez-lacces-de-votre-site-web

Sur les API
https://openclassrooms.com/fr/courses/7709361-construisez-une-api-rest-avec-symfony

Pour s’entraîner

Création d’une API de l’application TODO