TP3 : Formulaires / Back-office / Crud

Objectif : Création des CRUD pour gestion des entités déjà créées. Modification des formulaires, rendu des formulaires en TWIG etc …

1. Créer les formulaire CRUD permettant de gérer les entités de types DISH. Ces formulaires (actions de controller) devront :
    a. gérer les opérations suivantes : list, show, new, update, and delete.
    b. être accessible via des urls préfixées en /admin/dish
    c. être configurable via annotations

2. Localiser les éléments générés par Symfony :
    a. Controller : repérer les différentes actions auto-générées, essayer de comprendre le fonctionnement de ces actions.
    b. Views : repérer les templates auto-générés pour le rendu de ces formulaires et page de liste.
    c. FormType : repérer dans App/Forms le FormType auto-généré pour les entités Dish.

3. Ouvrir dans un navigateur les urls correspondantes aux CRUD ainsi générés :
    a. /admin/dish/ : liste les entités DISH présentes en base de données
    b. /admin/dish/new - edit - delete : permettent la création ou l'édition de ces même entités.
    c. Essayer de créer un plat via la fonctionnalité : Create a new dish
    d. En cas d'erreur : tenter de comprendre l'exception obtenue.
    e. Implémenter les méthodes manquantes

4. Modifier les éléments suivants :
    a. Dans les formulaire de création et d'édition d'un Plat (DISH) : Faire en sorte que les champs : name / description / Category / Prix soient obligatoires.
    b. Dans les formulaire de création et d'édition d'un Plat (DISH) : faire en sorte que le champs prix n'accepte que les chiffres (entiers ou float).
    c. A l'enregistrement ou la mise à jour d'un contenu de type DISH, si le champs Image est null ou vide, renseigner automatiquement une image par défaut : http://via.placeholder.com/360x225
    d. Dans les pages de liste (admin/dish) et de visualisation (admin/dish/<ID>/show). afficher l'image au lieu de l'url pour le champs Image.

e. A l'aide d'annotations dans l'entité : Dans les formulaire de création et d'édition d'un Plat (DISH) : Faire en sorte que le champs description compte au minimum 10 caractères, mais pas plus de 100. Un/des message(s) doivent s'afficher si ce n'est pas le cas.

f. Dans les formulaire de création et d'édition d'un Plat (DISH) : faire en sorte que le champs "Calories" ne soit pas un champs libre, mais une liste déroulante à choix unique, contenant des choix possibles allant de 10 à 300 (de 10 en 10). ***Idéalement, cette liste doit être fournie par le controller au formType dans une Option.***

Exemple de code pour générer les options :

```
function _availableCalories()
{
  $calories = array();
  for($i=10;$i<=300;$i+=10)
     $calories[$i]=$i;
  return $calories;
}
```

5. Répéter l'opération de création de CRUD pour les entités : User / Category / Allergen afin d'obtenir tous les CRUD nécessaire à la gestion de vos contenus.
6. En utilisant les flashbag messages, ajoutez à votre back-office un système de message indiquant à l'utilisateur lorsque tout s'est bien passé :
    a. "Le plat <nom du plat a bien été créé>"
    b. "Le plat <nom du plat a bien été modifié>"
    c. "Le plat <nom du plat a bien été supprimé>"
    d. etc ...
7. Avec les connaissances Symfony acquises dans les exercices précédents :
    a. créer une nouvelle route dans le controller Admin du App
        i. Cette route répondra sur l'url : /admin.
        ii. Cette route servira un template TWIG proposant une liste de liens vers les fonctionnalités du "Back-office".
    b. Ajouter au site du Restau, un lien "gérer" permettant d'accéder à cette nouvelle route depuis les pages du site en "Front-office".
8. Avec vos connaissances TWIG, faire hériter l'ensemble des views générés par les CRUD d'un layout HTML/CSS dédié à ce que nous appellerons désormais le "Back-office". Ce layout, peut (selon vos préférences) hériter lui même du layout utilisé sur le front-office.

TP4 : Services / DI / Commands

Objectif : Utilisation des commandes Symfony. Injection de dépendances + Création d'un service.

1. Créer une commande console Symfony permettant d'importer l'ensemble de la liste du personnel.
    a. Dans le Bundle Restau créez un Dossier Command.
    b. Dans ce dossier créez une nouvelle classe PHP (consultez la doc Symfony de création d'une commande).
        i. Qui porte le nom : **ImportPeopleCommand**
        ii. Qui étend la classe ContainerAwareCommand
        iii. Qui implémente les méthodes configure & execute
    c. cette commande devra s'exécuter en console de la façon suivante :
        i. php bin/console restau:people:import
    d. Le restaurant dispose d'un outil de RH, qui met à disposition certains services via une API.
    e. L'api est disponible ici : http://35.180.128.163/ (http://bit.ly/2W8Z2nl)
    f. La commande **ImportPeopleCommand** doit importer le Personnel du restaurant dans la Base de données en se basant sur les données fournies par le webService.
    g. Si vous avez le temps, a l'aide de la documentation symfony, ajoutez à votre commande la possibilité de recevoir une option : URL

2. Créer un Service symfony permettant de connecter l'outil de RH du restaurant.
    a. Dans le Bundle App, créez un dossier Services.
    b. Dans ce dossier, créez une nouvelle classe : RhService
    c. Cette classe devra fournir 2 méthodes principales (vous les implémenterez plus tard) :
        i. **getPeople** : qui après consultation de l'aPI renverra l'ensemble du personnel.
        ii. **getDayTeam** ($date) : qui après consultation de l'aPI retournera l'équipe présente au restaurant à la date donnée.
    d. Dans le Bundle App, dans le dossier Resources/config repérez et éditez le fichier services.yml . Configurez votre service afin que :
        i. il soit accessible sous le nom suivant : restau.services.rh
        ii. il recoive les dépendances suivantes :

1. Le Logger par défaut de symfony. @logger
2. URL Racine de l'api : %rh_api_endpoint% (à configurer dans vos parameters)

Exemple de constructeur pour le service (en SF3) :

```php
class RhService
{
    private $logger;
    private $api_endpoint;

    public function __construct(LoggerInterface $logger,$api_endpoint)
    {
        $this->logger = $logger;
        $this->api_endpoint = $api_endpoint;
    }
```

Resources/config/services.yml :

services.yml

```yaml
services:
    restau.services.rh:
        class: RestauBundle\Services\RhService
        arguments: ["@logger",  "%rh_api_endpoint%"]
```

App/config/parameter.yml :

services.yml | parameters.yml

```yaml
# This file is auto-generated during the composer install
parameters:
    database_host: 127.0.0.1
    database_port: null
    database_name: iut
    database_user: root
    database_password: root
    mailer_transport: smtp
    mailer_host: 127.0.0.1
    mailer_user: null
    mailer_password: null
    secret: ThisTokenIsNotSoSecretChangeIt
    rh_api_endpoint: "https://master-7rqtwti-vobaeam2aenxq.eu.platform.sh/"
```

3. Implémentez la méthode "**getDayTeam**" dans le service.
4. Déplacez le code de récupération de la liste du personnel du code de votre Commande "**ImportPeopleCommand**" vers la méthode "**getPeople**" de votre service de façon à ce que la Commande utilise le Service.

5. Mettez à jour la page "Equipe" sur le site web, pour qu'elle affiche l'équipe présente au restaurant à la date du jour. Les client pourront savoir avant de venir qui les servira.

TP5 : EventListeners

Objectif : Comprendre les Events Symfony et implémenter un listener / un subscriber et un Event custom. Utiliser le Dispatcher.

1. A l'aide des mécanismes vu ensemble dans les TP précédents, créer les entités suivantes :
    a. Un entité "Table" représentant les différentes tables du restaurants :
        i. nom de la table (ou numéro de la table)
        ii. nom de la salle (grande salle / petite salle)
    b. Insérez manuellement en base de données (ou via une commande) les différentes tables de votre restaurant.
    c. Une entité "Order" :
        i. numéro de commande (ID)
        ii. date/heure de prise de commande
        iii. Utilisateur (serveur) à l'origine de la commande (ManyToOne vers User)
        iv. Table concernée par la commande (ManyToOne vers Table)
        v. Plats contenus dans la commande (ManyToMany vers Dish)
        vi. Prix total de la commande
        vii. Status de la commande (Prise / Préparée / Servie / Payée)
2. A l'aide des mécanismes vu dans les TP précédent, créez le formulaire de prise de commande dans le back-office.

3. Créez un EventSubscriber permettant de calculer le montant total de la commande avant que celle ci soit enregistrée en base de donnée.
    a. A lire :
        i. http://symfony.com/doc/3.3/event_dispatcher.html

ii. https://symfony.com/doc/current/doctrine/event_listeners_subscribers.html

b. A l'aide de la documentation : Créez sous la forme d'un service, un EventSubscriber réagissant aux évènements doctrine : PrePersist et PreUpdate

c. Au PrePersist, ce service renseignera automatiquement

i. la date/heure de prise de commande.

ii. le status de la commande à "Prise".

iii. l'utilisateur / serveur ayant pris la commande (utilisateur connecté).

d. Au PrePersist ET au PreUpdate, ce service calculera automatiquement le montant total de la commande en additionnant les prix des plats contenus dans la commande.

4. Créez 2 EventListeners permettant de gérer la gestion de la commande dans le restaurant. Nous utiliserons ici des emails mais cela pourrait être des notifications mobiles ou tout autre système de message.

a. Vous devrez ici, injecter dans vos services, le service 'Mailer'

b. Un EventListener réagissant au PostPersist Doctrine sur l'entité Order.

i. Un email est envoyé aux cuisines indiquant qu'une nouvelle commande a été prise en salle.

c. Un EventListener réagissant au PostUpdate Doctrine sur l'entité Order

i. Si la commande est au statut "Préparée" un email est envoyé au serveur pour lui indiquer qu'il peut servir la commande.

ii. Si la commande est au statut "Servie" un email est envoyé à l'accueil pour indiquer que la commande pourra être encaissée.

iii. Si la commande est au statut "Payée" un email est envoyé au serveur pour lui indiquer que sa commande a été facturée.

5. Lorsqu'une commande est payée, l'application Restau doit aussi indiquer au système RH que l'argent a été encaissé. Nous pourrions réaliser cela dans les Listeners créés ci dessus, mais nous souhaitons que l'application Restau soit évolutive, et puisse être compatible si le restaurant mets en place d'autres mécanismes Digitaux à l'avenir. Nous allons donc créer notre propre évènement : **order.payed** qui pourra être utilisé par n'importe quel bundle à l'avenir.

6. A l'aide de la documentation, créez cet évènement

a. Documentation : https://symfony.com/doc/3.3/components/event_dispatcher.html

b. Créez une classe d'évènement dans App/Event qui s'appellera OrderPayedEvent

7. Ajoutez dans le Listener PostUpdate de l'entité Order, le dispatch de l'event order.payed (pour ce faire, vous devrez injecter l'EventDispatcher en dépendance de votre service Listener PostUpdate).

8. En utilisant les mécanismes déjà vu précédemment, créez un listener sur votre propre évènement **order.payed**.

    a. Ce listener devra disposer en dépendance du service d'accès à l'API du système RH développé au TP4. Ce dernier devra contenir une nouvelle méthode **setOrderPayed($order)** . Exemple de code rapide :

```
47      /**
48       * Helper to declare a payed order in RH system
49       */
50      public function setOrderPayed($order)
51      {
52          $url = $this->api_endpoint.
53              "?method=order&order=".$order->getId().
54              "&amount=".$order->getAmount().
55              "&server=".$order->getOwner()->getUsername();
56          return json_decode( file_get_contents($url) );
57      }
58  }
```

    b. Lors du déclenchement de l'évènement **order.payed** sur une commande, la méthode **setOrderPayed** sera déclenchée. Et la commande sera donc automatiquement comptabilisée dans l'outil RH de l'établissement.

TP6 : Sécurité / Firewall

Objectif : Sécuriser le back-office par Login/Mot de passe

En vous aidant de la documentation symfony 4.3, configurez la sécurité de votre symfony en définissant des Firewall.

Source : https://symfony.com/doc/master/security.html

1. Installez les bundles de sécurité.
2. Liez le security Bundle à votre entité User existante afin que chaque membre de l'équipe puisse avoir son compte (Utilisez le Maker)
3. Configurez les firewall nécessaire :
    a. Les urls de back-office (/admin/*) doivent être restreintes au rôle ROLE_ADMIN
    b. Les urls de backoffice de gestion de utilisateurs (/admin/users) doivent être retreintes au role ROLE_SUPER_ADMIN
    c. Les autres url sont publiques
4. Créez une commande symfony, qui crée automatiquement des comptes utilisateurs utilisables pour tous les nouveaux utilisateurs arrivant par l'API RH (implémentée au TP4)

TP7 : API

Objectif : Maîtriser l'ensemble de la stack, être autonome dans la recherche et dans l'évolution de la solution technique.

Sujet : Le restaurant a investi dans le développement d'une solution mobile.

Chaque personnel de salle du restaurant dispose d'une application mobile dédiée installée sur un smartphone. Un Ipad est accroché au mur en cuisine pour les suivi des plats a préparer. La caisse du restaurant dispose aussi d'une application permettant de traiter le paiement au départ des clients.

L'application doit permettre de prendre des commandes et de les suivre de la prise de la prise de commande jusqu'au paiement du client.

Pour cela, elle doit interagir avec notre système.

Le partenaire externe qui a développé l'application a fourni la documentation suivante :

GET /api/1.0/service/dishes/list.jon?category={id}

- **Documentation**

Requirements

Name	Requirement	Type	Description
category	facultatif	string	ID de catégorie de plat

Return

liste des plats au format JSON

GET /api/1.0/service/dishes/{id}/detail.jon

Requirements

Name	Requirement	Type	Description
id	required	int	ID du plat

Return

retourne le détail d'un plat avec les allergènes

GET /api/1.0/service/team/today.jon

Return

Retourne la liste du personnel présent aujourd'hui dans l'établissement.

GET /api/1.0/service/team/today.jon

Return

Retourne la liste du personnel présent aujourd'hui dans l'établissement.

POST /api/1.0/service/orders.json

Requirements

Name	Requirement	Type	Description
order	required	Order	commande à insérer

Action

Créé une commande.

Return

La commande ainsi créée

PUT /api/1.0/service/orders/{ID}.json

Requirements

Name	Requirement	Type	Description

order	required	Order	commande à insérer
id	required	int	id de la commande à modifier

Action

Modifie la commande.

Return

La commande ainsi créée

API Status Codes

Status Code	Description
200	• OK
404	• Object not found
403	• Malformed request

---

1. A l'aide des mécanismes vu ensemble dans les TP précédents, installez les bundles de contributions :
    a. FOS RestBundle (+ JMS Serializer Bundle)
    b. NelmioApiDocBundle
2. Alternative : ApiPlatform Bundle
3. A l'aide de la documentation des différents bundle, implémentez l'API nécessaire au fonctionnement des applications : configuration / routing / controllers / serialization.