Documentation Technique - Facturation

1) Informations techniques.

L'application est réalisée avec le framework Symfony (2.6.11).
Elle tourne avec une base MySQL.
Langages utilisés : PHP (5.4 minimum), HTML , Twig , JavaScript , CSS , SQL.

Liste des bundles utilisés :

- « friendsofsymfony/jsrouting-bundle » : facilite l'appel aux routes.
- « obtao/html2pdf-bundle » : Facilite l'intégration de HTML2PDF.
- « beberlei/DoctrineExtensions » : Ajoute nombre de fonctions SQL.
- « friendsofsymfony/user-bundle » : Gère la connexion des utilisateurs.

Liste des commandes à entrer dans la console **à l'installation** de l'application :

En admettant que vous vous trouvez dans /var/www/html.

Récupérer la dernière version de l'application via GitHub
- git clone lien vers le git

Aller dans à la racine
- cd *racine*

Récupérer composer.phar
- curl -sS https://getcomposer.org/instaler | php

Installer les bundles et dépendances
- php composer.phar install

Donner les droits pour les caches
- HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' | grep -v root | head -1 | cut -d\ -f1`

- sudo chmod +a "$HTTPDUSER allow delete,write,append,file_inherit,directory_inherit" app/cache app/logs

- sudo chmod +a "`whoami` allow delete,write,append,file_inherit,directory_inherit" app/cache app/logs

Créer la base de données et entrer les tables

- php app/console doctrine:database:create
- php app/console doctrine:schema:update —force

Liste des commandes à entrer dans la console **après un changement** dans l'application :

En admettant que vous vous trouvez dans /var/www/html/Facturation.

Récupérer la dernière version de l'application via GitHub

- git pull

Vider les caches dev et prod

- php app/console cache:warmup —env=dev —no-debug
- php app/console cache:warmup —env=prod —no-debug
- rm -rf app/cache/dev/*
- rm -rf app/cache/prod/*

Créer les liens

- php app/console asstic:dump
- php app/console assets:install

Mettre à jour la base de données

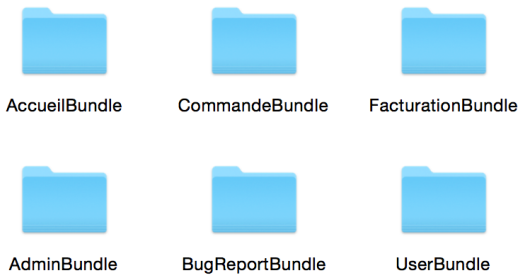
- php app/console doctrine:schema:update —force

2) Arborescence.

Chaque type de fichier est bien séparé en respectant les règles de Symfony.

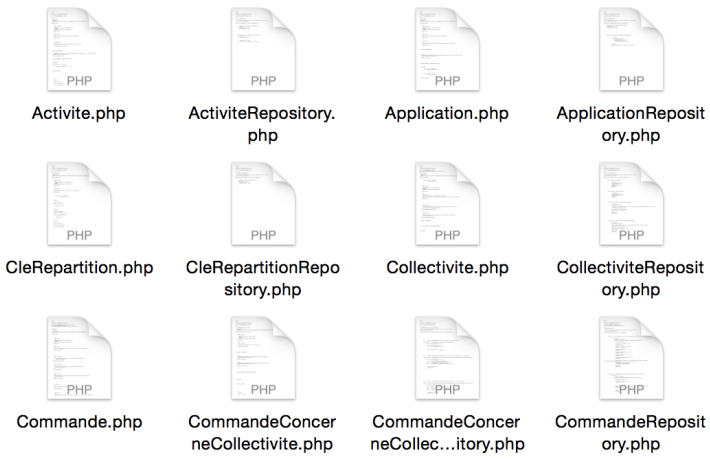
1) Les Bundles

L'application est séparée en 4 Bundles principaux : Accueil, Commande, Facturation et Admin.
Chaque bundle se trouve dans **src / JC /** :



2) Les Entités

Les entités sont toutes regroupées dans le Bundle « Commande ».
—> **src / JC / CommandeBundle / Entity /**

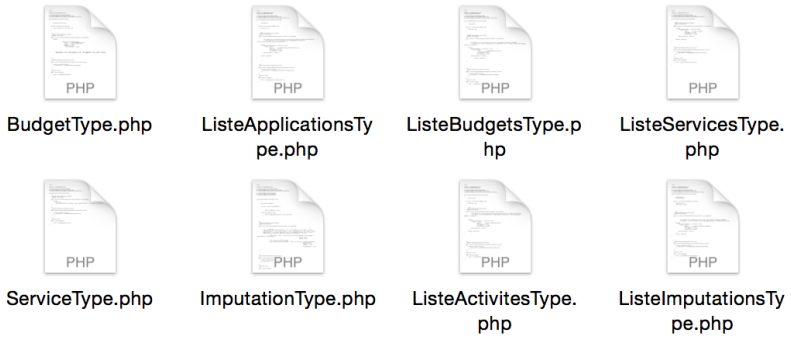


Elles ont (presque) toutes un Repository associé dans ce même dossier.

Les entités commençant par « Liste… », sont des entités qui ne servent qu'à stocker une collection d'Entity. Transformées en formulaires, ces entités permettent de gérer la page Admin.

3) Les Formulaires

Les formulaires sont tous regroupés dans le Bundle « Commande ».
—> **src / JC / CommandeBundle / Form /**



Le contrôle des formulaires se fait grâce à au Validator de Symfony

—> **« use Symfony\Component\Validator\Constraints »** . Les contraintes sont placées au-dessus de chaque attribut de chaque entité.

4) Les Views

Les views se trouvent dans leur bundle correspondant.
—> **src / JC / XBundle / Resources / views / X**

4) CSS et JS

Les bundles comportent leurs propres CSS et JS
—> **src / JC / XBundle / Resources / public / JS ou CSS /**

Note : L'application comporte un fichier CSS et un fichier JS « global », qui s'applique à tous les bundles. Ces deux fichiers servent à ne pas répéter des classes ou fonctions qui sont utilisées dans plusieurs bundles.
—> **app / Resources / Public / JS ou CSS /**

3) Les Bundles.

L'application se divise en 4 Bundles : Accueil, Commande, Facturation et Admin.

1) Le bundle Accueil

Le bundle Accueil n'est utilisé que dans l'affichage de la page d'accueil. C'est elle qui donne un récapitulatif des budgets des services.

C'est dans ce Bundle que la NavBar est créée.
—> **src / JC / AccueilBundle / Resources / views / Accueil**

Cette NavBar est appelée dans le layout principal de l'application, afin de ne pas avoir à l'inclure sur chaque page.
—> **app / Resources / views / layout.html.twig**

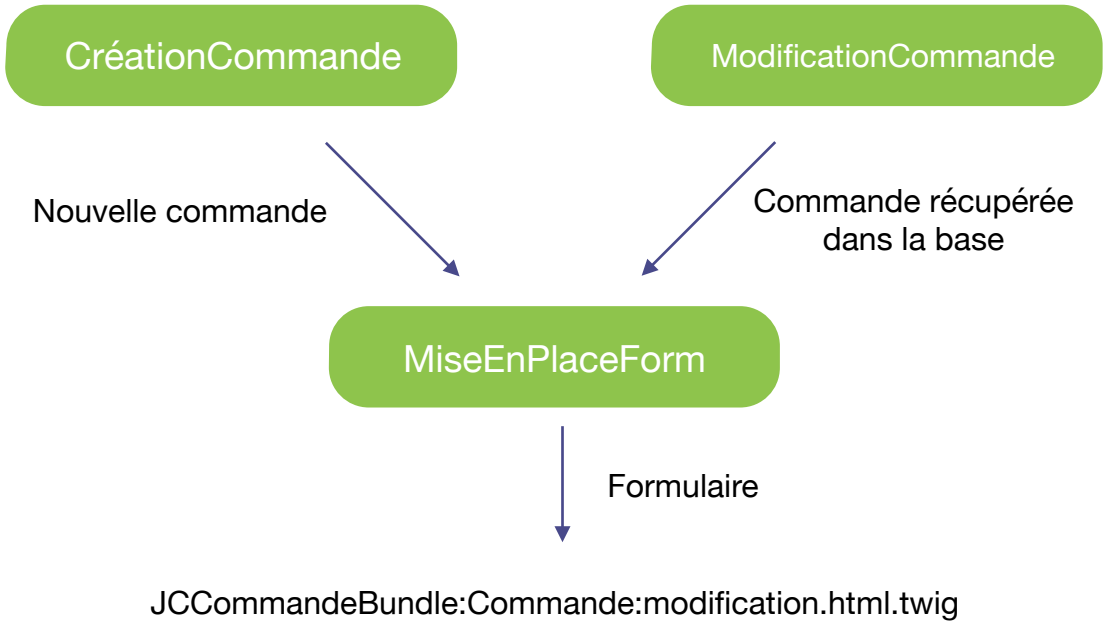
2) Le bundle Commande

Le bundle Commande concerne la gestion des commandes : l'affichage de la liste des commandes, la création et la modification des commandes.

L'affichage de la liste des commandes se fait avec la fonction « listeAction() ». La pagination est gérée dans le repository de l'entité Commande.

L'affichage du détail d'une commande s'effectue grâce à la fonction « detailAction() ».

Création et modification d'une commande sont liées. Les deux feront appel à la même fonction « MiseEnPlaceFrom() » qui va mettre en place le formulaire de création ou de modification.



Dans les fonctions « detailAction() » et « modificationAction() » un contrôle est fait au début de la fonction pour être sur que l'on ne modifie pas une fonction qui est engagée, en paiement ou terminée ; et inversement dans la fonction « detailAction() ».

Par exemple, la fonction « modificationAction() » est appelée avec l'ID d'une commande terminée en paramètre, elle renvoie automatique sur « detailAction() ».

En plus des routes évoquées, le bundle Commande contient des routes uniquement appelées en Ajax.

- jc_commande_get_tva : renvoie la liste des montants de TVA
- jc_commande_change_etat : change l'état d'une commande
- jc_commande_paiement_commande : enregistre le paiement d'une commande
- jc_commande_supprimer_paiement_facture : supprime le paiement d'une commande

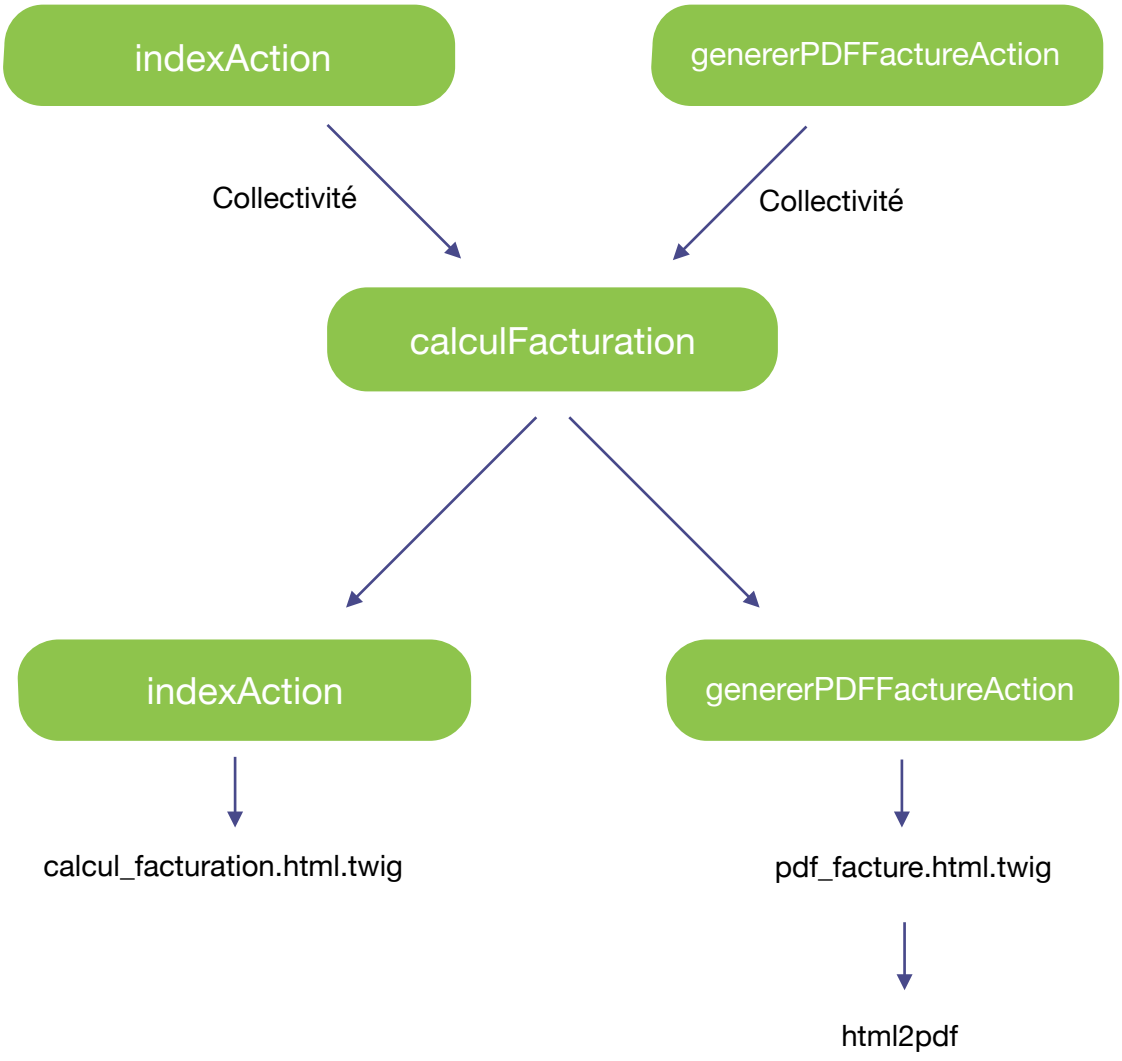
Les trois dernières routes appellent des fonctions qui renvoient « true » si l'action s'est bien déroulée, « false » sinon.

3) Le bundle facturation

Le bundle Facturation concerne la facturation annuelle faite aux collectivités. Il ne contient que 3 routes, une route qui affiche une page de présentation (route : jc_facturation_homepage), une pour afficher le détail de la facturation pour une collectivité donnée (route : jc_facturation_calcul), et la dernière qui appelle la fonction qui crée le PDF à envoyer aux collectivités.

La fonction « indexAction() » se contente de compter, pour chaque collectivité, le nombre de commandes mutualisées et le nombre de commandes directes.

Les fonctions « calculAction() » et « genererPDFFactureAction() » appellent toutes deux la même fonction « calculFacturation() » afin, pour la première, d'afficher le détail de la facturation pour la collectivité donnée, et pour la deuxième, de créer le PDF à envoyer à la collectivité.



La fonction « calculFacturation() » est très imposante, c'est elle qui calcule ce que doit payer une collectivité pour une année donnée. Les calculs sont expliqués dans la documentation utilisateur de l'application, dans la partie « Calculs ».

Note : Si le calcul de la facturation doit changer, c'est cette fonction qui doit être modifiée.

- Cette fonction construit un tableau pour la collectivité donnée. Le tableau contient les index suivants :

- « nom » —> le nom de la collectivité

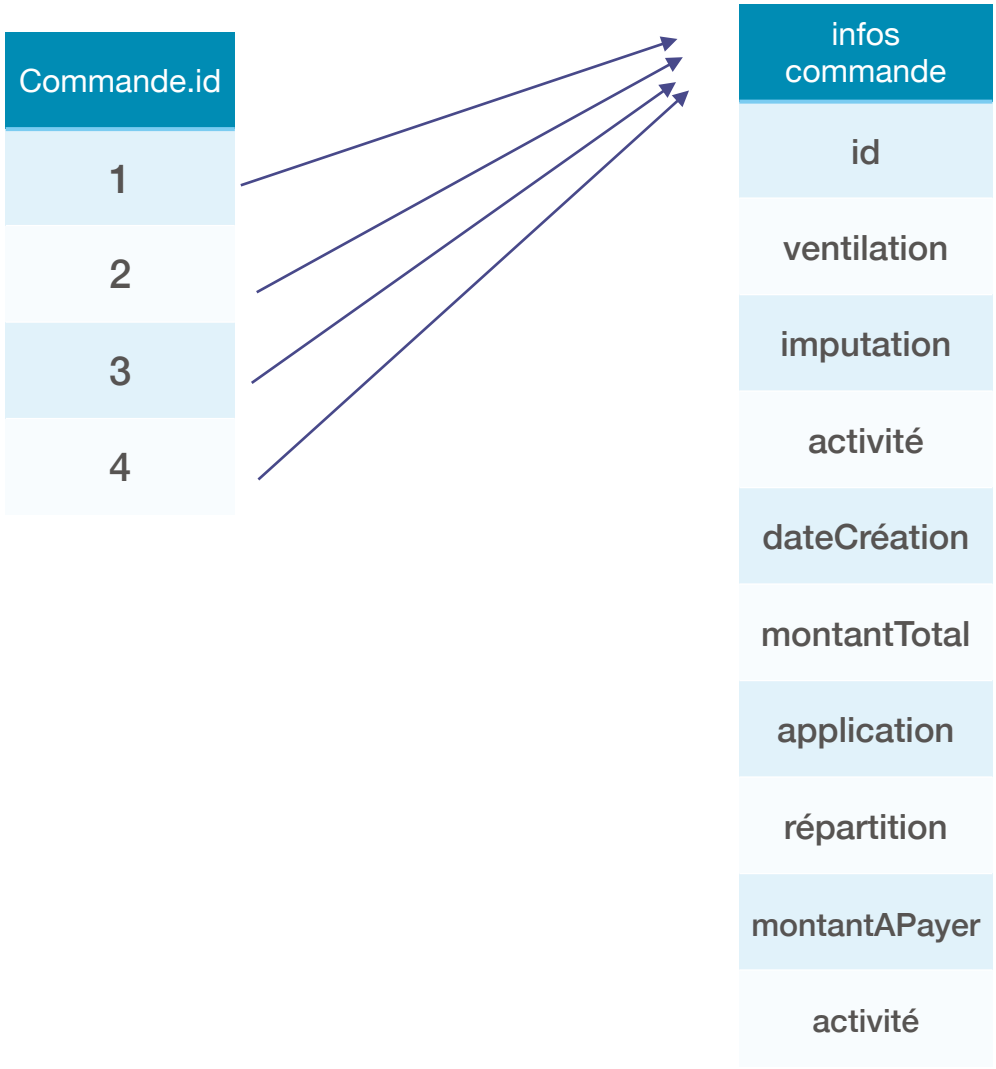
- « nbMutualisees » —> le nombre de commandes mutualisées
- « nbDirectes » —> le nombre de commandes directes

- « montantMutualisees » —> le montant des commandes mutualisées
- « montantDirectes » —> le montant des commandes directes

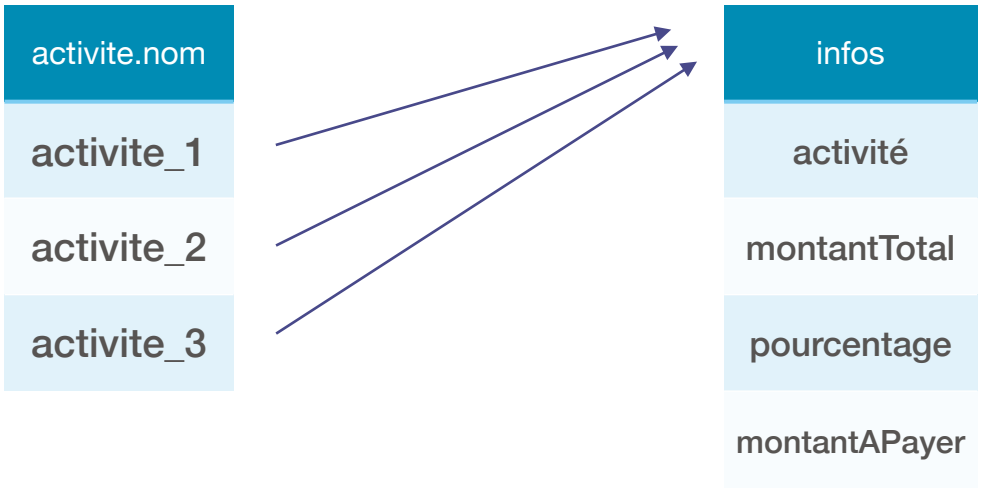
- « nbForfaits » —> le nombre de forfaits
- « montantForfaits » —> le montant des forfaits

- « montantInvestissement » —> montant des commandes investissement
- « montantFonctionnement » —> montant des commandes fonctionnement
- « montantMassesSalariales » —> montant des masses salariales cumulées

- Elle construit également un tableau qui stocke les commandes. Ce tableau se construit comme cela :



- Enfin, elle construit un tableau qui stocke les masses salariales. Ce tableau se construit comme cela :

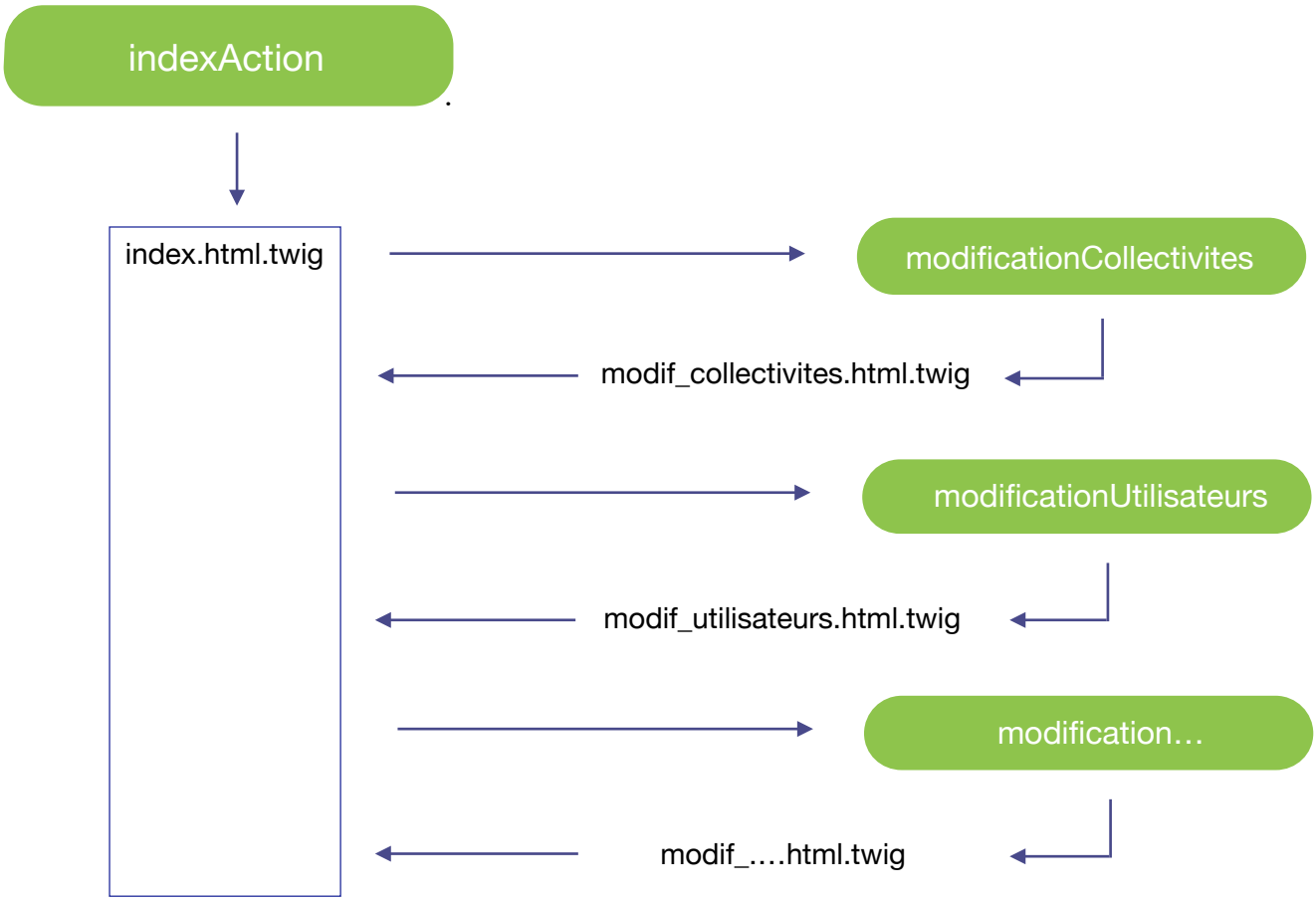


Note : On stocke des totaux pour les commandes bien que l'on stocke les commandes en détail, cela afin de ne pas faire des additions dans le Twig (ce qui pourrait prendre plus de temps)

4) Le bundle admin

Le bundle Admin sert à la gestion interne de l'application. Il sert par exemple à ajouter une application dans la base, ajouter un utilisateur, etc …

Cette partie se présente sous la forme d'une seule page, dans laquelle sont chargées toutes les informations. Grâce à la classe « tab-pane », lorsque l'utilisateur clique sur un « li » dans le menu sur le coté gauche, seul le « tab-pane » correspondant s'affiche.



Toutes les « tab-pane » contiennent donc des formulaires. Ce sont ces formulaires qui sont générés grâce aux entités « ListeX »

Important : Dans la page admin, l'utilisateur a la possibilité de créer des utilisateurs, des applications, etc . Lors de l'installation de l'application (lorsque la base est vide), dans les formulaires contenant des « Select » ces ajouts ne pourront pas être effectués. En effet, pour ne pas avoir à faire de requête au chaque fois que l'on clique sur « Ajoute un(e) X », une fonction JavaScript va récupérer les options d'un Select déjà entré pour les mettre dans le nouveau. Donc si la base est vide, la fonction ne trouvera aucun Select pour copier les « Options ».

Il faut donc passer par PHPMyAdmin pour entrer les premières données .

Un fichier SQL est fourni contenant des informations types.

4) Sécurité.

La connexion des utilisateurs est gérée par le bundle « friendsofsymfony/user-bundle ».

Trois rôles sont définis :
- ROLE_SUPER_ADMIN
- ROLE_ADMIN
- ROLE_COMPTA

Le rôle SUPER_ADMIN n'est pas utilisé, contrairement aux rôles ADMIN et COMPTA.

Le rôle ADMIN peut modifier les informations dans la partie « Admin » sur site, peut changer le statut d'une commande et peut effectuer des paiements.

Le rôle COMPTA peut changer le statut d'une commande et peut effectuer des paiements, mais ne peut faire des modifications que sur la partie « Imputation » de la section ADMIN.

Note : La page de connexion est redéfinie dans le bundle UserBundle
—> **src / JC / UserBundle / Security**

Ce bundle n'a pas été mentionné avant, car il n'est utilisé que pour ça.