

Annexe

Test alternance

JANIO Mathieu

Table des matières

[Question 1 : Liste des 20 dernières commandes 3](#_Toc29929989)

[Question 2 : Liste optimisée des 20 dernières commandes 3](#_Toc29929990)

[Question 3 : Enregistrer de nouvelles commandes 4](#_Toc29929991)

[Question 4 : Enregistrer des nouvelles commandes en utilisant les services 4](#_Toc29929992)

[Question 5 : Charger des commandes en javascript 5](#_Toc29929993)

[Question 6 : Charger des commandes en Vanilla javascript (sans aucune librairie) 5](#_Toc29929994)

[Autres soucis : 5](#_Toc29929995)

[Conclusion : 5](#_Toc29929996)

# Question 1 : Liste des 20 dernières commandes

Difficulté : Facile

Il était question de retourner les 20 dernières commandes à l’aide d’un QueryBuilder dans la méthode « getLastNewOrders » du repository « OrderRepository ».

La méthode ne prend aucun paramètre et retourne une liste d’« Order » filtré pour les nouvelles commandes, ordonnées par date pour obtenir les dernières commandes pour retourner seulement les 20 dernières.

Tous ces paramètres sont définis à l’aide de méthodes intégrés dans QueryBuilder.

La seule difficulté que j’ai pu rencontrer était la réalisation d’un select dans le QueryBuilder puisque ce dernier gérait déjà cette méthode. Après 1 an sans faire de Symfony cela a pu me rappeler les bases que j’avais durant mon cursus à l’EPSI. Ce qui m’a permis par la suite de produire efficacement le projet que vous m’avez confié.

La vue et le contrôleur n’ont pas eu besoin de modifications.

# Question 2 : Liste optimisée des 20 dernières commandes

Difficulté : Facile

Comme pour la première question le QueryBuilder a été créé, mais à l’instar de la première question nous effectuons directement dans le QueryBuilder la jointure avec le Customer afin que Symfony n’ait pas à récupérer, avec des requêtes supplémentaires, ces éléments-là.   
Libérant ainsi des échanges inutiles avec la base de données.

Pour ajouter une couche d’optimisation il pourrait être judicieux d’intégrer une couche de cache et d’ajouter une option pour forcer le rechargement (bouton actualiser).

# Question 3 : Enregistrer de nouvelles commandes

Difficulté : Moyenne

De la même façon que les premières questions, une QueryBuilder où l’ont précise que lors du select nous voulons le « count() » afin d’obtenir directement le nombre de commandes.

Pour ce qui est de récupérer les valeurs de l’api de la route « api/orders/new » je n’ai malheureusement pas réussi à réaliser l’appel api. Voici ce que j’ai réalisé :

* Appel de mon contrôleur vers le contrôler de l’api KO ❌
* Appel d’un fichier \*.php vers le contrôler api OK ✅
* Appel de mon contrôler vers une api externe (ex : <https://pokeapi.co/>) OK ✅
* Appel api twig vers contrôler api OK ✅

La version du projet Symfony est 4.1.7, je n’ai donc pas accès à « HttpClient » qui est disponible à partir de la version 4.3 ni la possibilité d’installer des modules complémentaires. Les requêtes ont donc été réalisées à l’aide de Curl. Et la requête qui ne fonctionne pas ce finit sur un CURLOPT\_TIMEOUT.

Je diagnostiquerai donc un problème avec Symfony ?

Pour outrepasser cette problématique j’ai donc utilisé « $this->forward() » où l’on passe en paramètre la route :: la classe puis un getter pour récupérer les informations. Il n’y avait plus qu’à transformer le JSON en tableau compréhensible pour le php, puis boucler sur ces éléments afin de créer les objets adéquats.

A chaque fois que l’on crée un objet avec ses informations correct nous préparons la requête.  
Puis lorsque tous les « Object » ont été insérés dans la requête nous exécutons celle-ci.

Pour savoir si la requête c’est bien déroulé nous exécutons à deux reprises le count des nouvelles commandes afin d’obtenir un avant/ après.

Paramètre de retour et vu modifié afin de mieux visualiser le résultat.

# Question 4 : Enregistrer des nouvelles commandes en utilisant les services

Difficulté : difficile

Même problématique que pour la question 3 sur l’appel api.

Le service retournerait le contenu de l’url puis dans le contrôler nous aurions un switch case qui nous permettrait de déterminer le type de donnée reçu et ainsi effectuer les traitements adéquats. Le service serait donc général et n’aurait en aucun cas besoin de plus amples modifications.

# Question 5 : Charger des commandes en javascript

Difficulté : facile

L’appel au contrôleur à l’aide de jQuery n’a pas été compliqué, en revanche je ne connaissais pas le système de Template. Après quelques recherches il s’est avéré simple de le mettre en place. Pour le système de filtre en javascript j’ai pensé à effectuer une nouvelle recherche en fonction de la valeur du select.

J’ai donc modifié la requête pour lui faire un « WHERE » si la valeur passée est contenue dans une liste de paramètres définit afin de limiter le plus possible les risques. Il aurait été possible, si la liste des « status » était évolutive, de créer celle-ci à l’aide d’une requête mais dans notre cas cela aurait fait des requêtes supplémentaires inutiles. Bien sûr le paramètre est défini à l’aide d’un « :status » afin qu’une fois de plus rajouter une couche de sécurité et éviter les injections SQL.

Pour finir en regardants la documentation de UiKit qui est utilisé dans le projet j’ai pu voir que la librairie possède un système de filtre des éléments présents dans le dom. J’ai donc également implémenté ce filtre qui était simple à utiliser et laisser l’ancien que j’avais réalisé.

# Question 6 : Charger des commandes en Vanilla javascript (sans aucune librairie)

Difficulté : facile

Le système de Template à été plus fastidieux à implémenter en Vanilla javascript. Mais il n’a pas non plus été compliqué à réaliser avec quelques recherches sur l’idée de la réalisation ainsi que le traitement des nodes du dom. Le filtre à également été fait rapidement.

# Autres soucis :

Version de MySQL requise : 5.6 alors que je possédais la dernière.  
Puis conflit entre mes MySQL d’installer.

# Conclusion :

Le test m’a paru relativement simple à réaliser après m’être re-imprégné de Symfony. La seule exception fut le problème de l’appel api sur lequel j’ai dû passer 70% du temps du projet et je n’en ai pas pu déterminer son origine. C’est ce qui m’a le plus frustré.