Refresh Technique

Xavier POLLEZ

Préambule

Le but de ce document est de présenter un ensemble d'exercice à faire sur le projet e-commerce initialement développé par les stagiaires en CDS durant l'année 2018.
Le projet a été déposé sur le GIT NEXTOO (https://git.nextoo.fr/nextoo-academy) pour lequel un accès a été donné.
De plus, une documentation technique est présente à l'adresse suivante https://taiga.nextoo.fr/project/chocapigs-e-commerce/wiki/home (de même un accès a été donné)

Chaque exercice a été estimé par un profil junior avec 1 an et demi d'expérience disposant déjà de compétences en JEE. Cela permet ainsi de se situer par rapport au temps de référence (le formé devra communiquer de manière transparente le temps passé par exercice ce qui permettra de situer où il est important d'insister lors des corrections).

Pour toute question/support, Olivier Delbouve (odelbouve@nextoo.fr) et Christophe Messiaen (cmessiaen@nextoo.fr) se tiennent à entière disposition.

Note: pour les exercices sur le projet e-commerce, se baser sur la branche git "refresh_tech_caracteristiques" pour avoir la version la plus à jour.
Aucun exercice ne doit être merge sur la branche principale du projet : chacun doit travailler sur sa propre branche de travail nouvellement créé (ex: "refresh_odelbouve").

Exercice 1 (estimation: 2j)

Spring Boot, Maven, Java 8, REST (entrant + sortant)

Objectif: Créer un projet de toute pièce, offrant un service REST user-friendly pour consulter la météo récupérée depuis une source externe, le tout en un temps record.

Pour cela, commencer par lire les références ci-dessous avant de faire l'exercice. Demander au formateur en cas de besoin.

- Spring Boot

Présentation: http://www.baeldung.com/spring-boot-starters
Initialisation d'un projet: https://start.spring.io/
Puis pour approfondir (plus tard, si intéressé et le temps):
https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

- API météo

https://openweathermap.org/appid
Ou
https://www.prevision-meteo.ch/services

- JSON / jackson (ex: JsonIgnoreProperties, JsonProperty…)

https://github.com/FasterXML/jackson-annotations

- RestTemplate

https://www.baeldung.com/rest-template

Commencer par créer un serveur REST:

- le créé via Spring Boot (voir le site)
- puis définir un point d'entrée REST qui:
  - accepte un nom de ville en paramètre (pour commencer)
  - récupère les prévisions du jour et de ceux à venir pour la ville depuis une des API externe, et génère les objets correspondants (uniquement les champs utiles, et avec des noms ayant un sens)
  - répond une liste des jours à venir avec les infos qui vont bien

- Ajouter les sources à Git

Suivre la procédure décrite en annexe "Relier des nouvelles sources à un git existant" pour créer une branche avec vos sources (elles sont indépendantes des autres branches, donc pas de clone nécessaire).
Par la suite, pensez à bien commiter et pousser vos modifications dessus.

- Java 8

https://programminghints.com/2015/12/moving-from-java-6-to-java-8/

Après avoir lu les nouveautés liées à Java 8, faire évoluer l'application en ajoutant des services supplémentaires (**ne pas écraser le(s) ancien(s)**). Utiliser les stream et autres nouveautés Java 8 pour parser/trier/transformer les prévisions pour ces nouveaux services:

- afficher le jour le plus chaud de la semaine
- afficher uniquement les jours où il pleut
- afficher les villes dans un périmètre de 50km où il fait plus chaud que la ville demandée (peut être pas possible, expliquer pourquoi si c'est le cas).
- afficher l'humidité actuelle, celle moyenne pour chaque jour de la semaine, et le détail du jour le plus sec (peut être pas possible, expliquer pourquoi si c'est le cas).

Exercice 2 (estimation: 2j)

VueJS

Objectif: Créer un front pour l'API météo de l'exercice 1

Conseils de développement:

- Il est possible d'utiliser le même IDE que le reste du projet (Eclipse/IntelliJ), mais on peut aussi utiliser VSCode pour faire du front
- Installer le plugin VueJS de Chrome

Lire les vidéos:
https://www.grafikart.fr/formations/vuejs

Jeter un oeil aux bonnes pratiques, a minima les "essentielles" (et ne pas hésiter à y revenir plus tard)
https://fr.vuejs.org/v2/style-guide/index.html
Et en support:
https://fr.vuejs.org/v2/guide/

Suivre les tutoriels ci-dessous en adaptant les exercices pour créer un front à votre API météo de l'exercice 1:
https://blog.ippon.fr/2017/04/24/vue-js-2-0-petit-tutoriel-volume-1/
https://blog.ippon.fr/2017/05/02/vue-js-2-0-petit-tutoriel-volume-2/
https://blog.ippon.fr/2017/05/09/vue-js-2-0-petit-tutoriel-volume-3/
https://blog.ippon.fr/2017/05/29/vue-js-2-0-petit-tutoriel-volume-4/
https://blog.ippon.fr/2017/07/24/vue-js-2-0-petit-tutoriel-volume-5/

Exercice 3 (estimation: 1j)

Spring, Hibernate/JPA, Mockito/PowerMock, Git

Pré-requis: récupérer le projet e-commerce depuis git sur le poste (créer un compte et demander les droits aux formateurs si besoin). **Utiliser la branche (ou tag) "refresh_tech_caracteristiques"** pour partir sur les bonnes bases.
https://git.nextoo.fr/nextoo-academy

Puis installer en suivant la procédure suivante (idem, demander les droits si besoin):
https://taiga.nextoo.fr/project/chocapigs-e-commerce/wiki/installationinitialisation-du-projet-e-commerce
Note: Docker n'est pas nécessaire en local, mais quand il est possible de l'utiliser, il est conseillé de l'utiliser pour la BDD (les autres sont inutiles car ils sont remplacés par IntelliJ). Le principal avantage est d'éviter tout soucis de versions/paramétrages, et/ou de conflit avec d'autres projets (ex: une mauvaise version peut bloquer l'installation, l'alimentation ou l'exécution de la base et faire perdre un temps précieux).

Objectif: sur l'application e-commerce, créer la fonctionnalité permettant d'ajouter une caractéristique sur un produit.
Les caractéristiques se composent d'un libellé (type) et d'une valeur, par exemple :

- **Broché**: 223 pages
- **Editeur** : Flammarion
- **Langue** : Français
- **Dimensions du produit**: 22 x 15,5 x 1,7 cm

Les types de caractéristiques doivent être une liste finie, éditable et commune à tous les produits.
Un type de caractéristique n'est pas obligatoire pour un produit, et ne peut être définie au maximum qu'une seule fois par produit.

Notions abordées: (si besoin/question, contacter le formateur)

- Spring: annotation (bean, autowired…), couches, DO/DTO, Maven…
- JPA: entité+liaisons, repository, annotations, HQL, méthodes auto générées, customRepository...
- Tests: properties+profil, test unitaire, mock, injectMock, test intégration…
- PAS de front
- PAS de Controller / REST, donc PAS de GraphQL!!

Important: qu'il soit prévu ou non de faire les services et/ou le front, il est important de s'adapter au code existant et de choisir si on modifie ce code ou on ajoute du nouveau. Il faut également orienter son développement sur ce qui est utile uniquement, et pas bêtement faire tout ce qu'il est possible de faire. Ex:

- Il est probable que l'on modifie les caractéristiques d'un produit directement dans celui-ci. Les méthodes seront donc certainement liées/intégrées à celles des produits existants.
- Il est certainement utile de lister tous les types de caractéristique afin de choisir laquelle on veut renseigner.
- Par contre, cela n'a pas vraiment de sens de lister toutes les valeurs que peut avoir un type de caractéristique.

Exercice:
- créer une branche pour garder le travail et penser à push une fois par jour pour vérification du travail (notifier les formateurs quand les pushs sont fait)
- créer/maj le modèle BDD pour avoir les caractéristiques, ainsi que les scripts correspondants
- créer les repository (utiles!) + Entity
- créer les services métier (utiles aussi) + DTO + transformer
- créer des tests unitaires et d'intégration (= tests fonctionnels par rapport aux cas d'utilisation prévu) afin de tester vos services et bdd

Exercice 4 (estimation: 1j)

JPA avancé (Criterias)

Objectif: Créer une requête complexe en utilisant les criterias JPA.

Sur le projet e-commerce, créer une méthode de repository qui permet de récupérer une liste de produit en utilisant un CriteriaBuilder.

Celle-ci devra accepter les filtres optionnels suivant:
- La note moyenne des avis clients est > paramA
- La note moyenne des avis clients est < paramB
- Le nom du produit = paramC
- Le nom du produit contient la valeur paramD
- La catégorie du produit = paramE (précisément, pas de notion de hiérarchie)

Le format des paramètres paramA à paramE est libre mais doit être justifiée.

Pensez à respecter la structure JEE, et à faire des TUs (attention aux valeurs bizarres, vides, injection… ;-)).
L'exercice peut être développé en suivant la méthode TDD (demander au formateur pour savoir)

Bonus: Pour aller plus loin

- OWASP

https://www.owasp.org
Top 10:
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

Les projets front e-commerce sont fonctionnels, il y a peut être des failles (ou pas, j'ai pas encore essayé)

- Analyse et débuggage

Basculer sur la branche refresh_tech_debug du projet e-commerce (backend) et refaire fonctionner les tests unitaires.
Il est possible de les voir tous avec un maven test, mais sinon voici la liste:
Facile:

- ProduitTransformerTests
- UtilisateurTransformerTests
- CategorieSupprimeBusinessTest
- PhotoBusinessTests
- ProduitBusinessTests
- UtilisateurBusinessTests
- UtilisateurMutationTests
- CategorieRestaurationTests

Compliqué:

- UtilisateurRepositoryTests
- CategorieBusinessTest
- CategorieRepositoryTests

- Docker

https://docker-training.jug-montpellier.org/
Parties utiles pour commencer (je pense que c'est déjà pas mal):

1. Requirements
2. The Basics
3. Images & build
4. Images from container
5. Volumes
6. Docker-compose

- AOP (programmation orientée aspect)

Objectif: Ajouter un logger au site e-commerce qui affiche le nom et les paramètres (avec leur valeur) de toutes les méthodes de business utilisées par l'application.

Présenter:

- Le concept: https://en.wikipedia.org/wiki/Aspect-oriented_programming
- Spring AOP & AspectJ: https://docs.spring.io/spring/docs/2.5.x/reference/aop.html

Puis faire l'objectif ci dessus sur le projet e-commerce.

Annexes

Stack standard

Back:
- Rest
- Java 8
- Suite Spring (Web, Core, MVC, Security...)
- JUnit (Mockito)
- SQL + Hibernate/JPA
- Maven/Nexus

Front:
- Angular 6 ou VueJS

Outils:
- Linux + IntelliJ
- Debugger Chrome
- GIT
- GitLab + Sonar (CI/CD = intégration/livraison continue)

Relier des nouvelles sources à un git existant

Par exemple pour l'exercice 1

```
cd /<projet>
git init .
git remote add origin https://git.nextoo.fr/nextoo-academy/weather-forecast.git
git checkout -b refresh_<nom>
git add .
git commit -m "Premiere version"
git push origin refresh_<nom>
```