Analyse et programmation orientée objet
Travail pratique

Haute école spécialisée de Suisse occidentale

Chargé de cours
Raphaël P. Barazzutti
raphael.barazzutti@heig-vd.ch

Travail réalisé par
Jämes Ménétrey
james.menetrey@heig-vd.ch

Date de rendu du projet: 11 février 2017

Table des matières

1 Introduction

Ce projet est réalisé pour la deuxième évaluation de l'unité AProgOO. Les critères d'évaluation sont :

— l'utilisation de la programmation orientée objet,
— l'utilisation de technologies de l'écosystème Java, tel que Spring et
— l'utilisation de JavaDoc pour la documentation des classes et méthodes.

Pour cela, le sujet du travail est une plateforme de blogging permettant à un utilisateur de créer des articles qui sont ensuite affichés sur la page d'accueil. Les figures suivantes représentent les quelques interfaces réalisées.

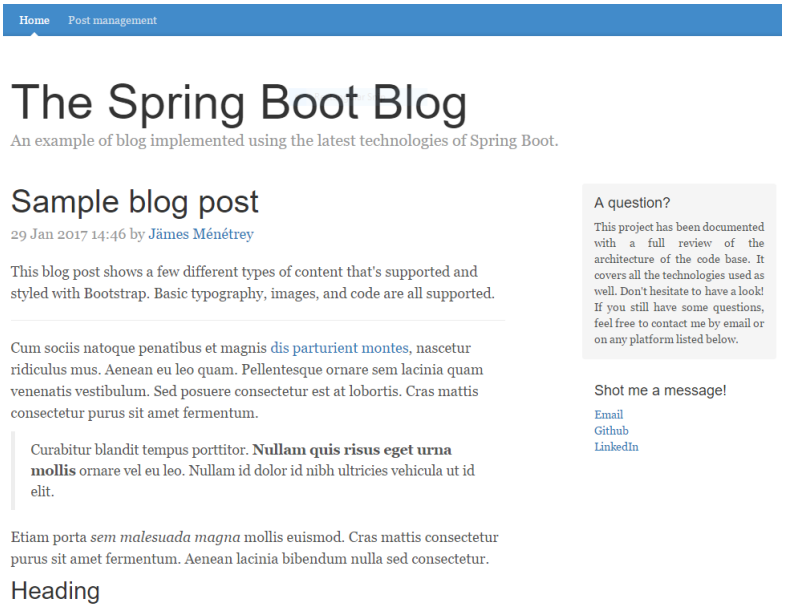

FIGURE 1 – Page d'accueil du blog.

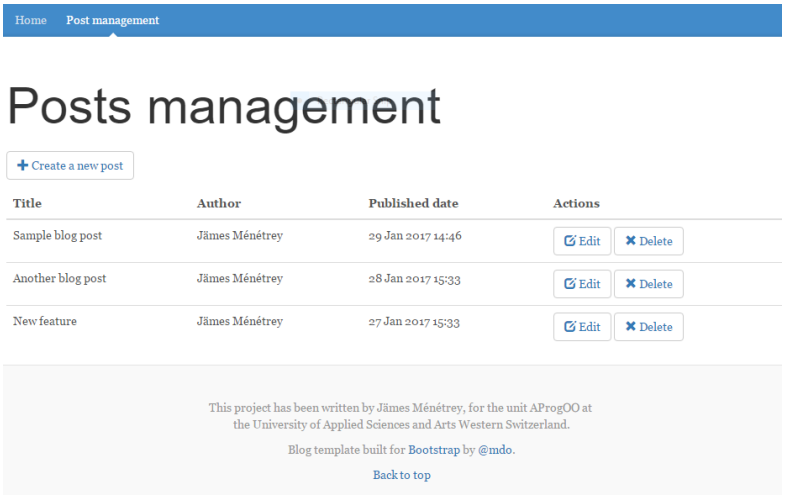

FIGURE 2 – Page gérant les posts écrits.

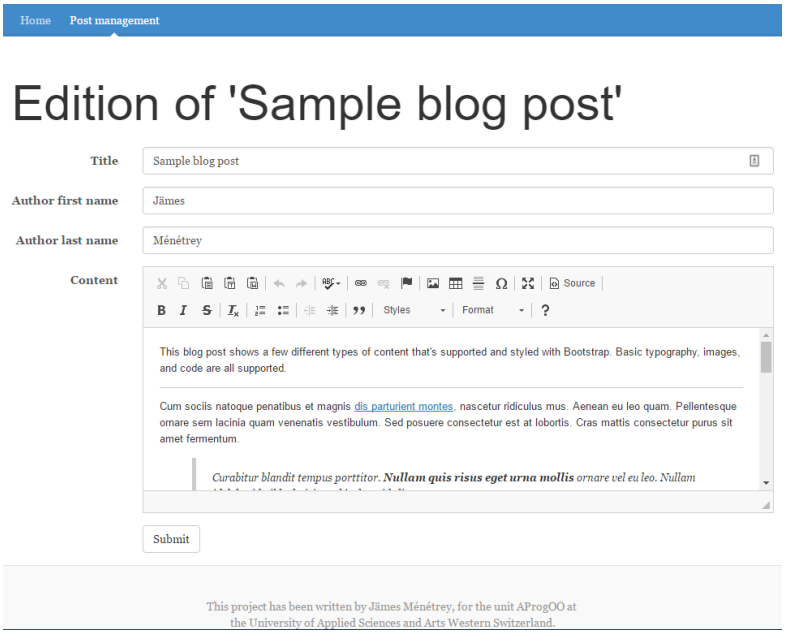

FIGURE 3 – Page éditant le contenu d'un post.

2 Architecture

2.1 N-tier architecture

Le projet a subdivisé selon une architecture n-tier. Le code est ainsi séparé en trois couches :

— **Data** : interaction avec le système de stockage des données (comme une base de données).
— **Domain** : contiens les modèles de données ainsi que la logique Business.
— **Presentation** : technologie manipulant les données avec la logique Business (comme une application Web).

La couche *Presentation* ne peut interagir qu'avec la couche *Domain* et celle-ci ne peut interagir qu'avec la couche *Data*, comme illustré sur le diagramme 4.

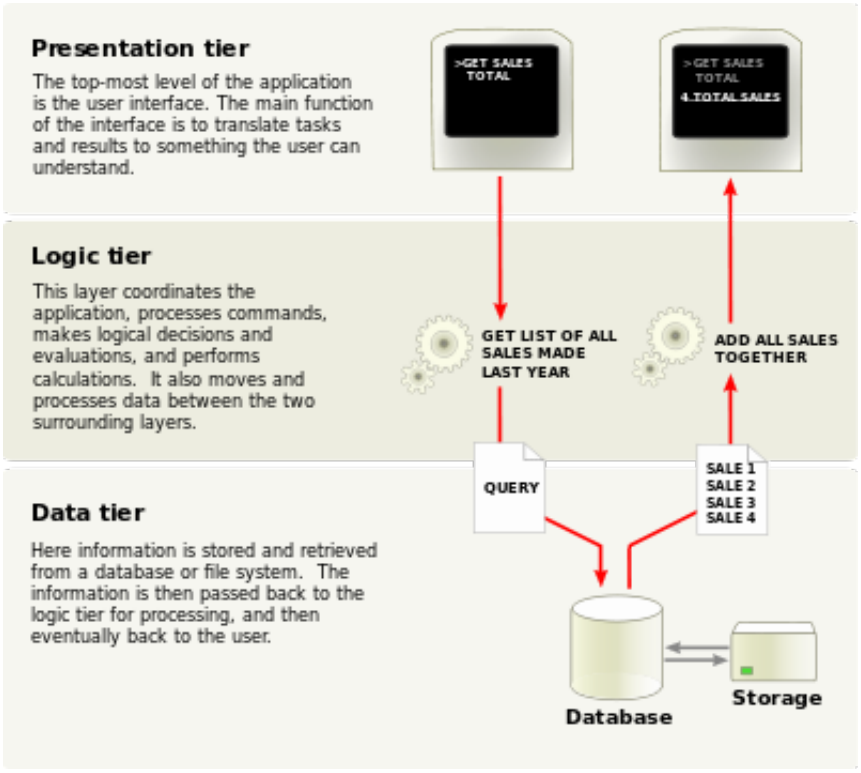

FIGURE 4 – Architecture n-tier en détail.

2.2 Structure du projet

La solution de développement est découpée en plusieurs modules Maven, comme détaillée dans le paragraphe suivant et dans la figure 5. La structure est composée ainsi par couche :

— Data
  — **data** : module d'abstraction de données. Il contient uniquement les interfaces liées à la manipulation des données.
  — **data-jpa** : module d'implémentation de manipulation des données, avec le framework *Spring Data JPA*.
— Domain
  — **domain** : module contenant les modèles de données que les autres couches manipulent.
  — **services** : module d'abstraction des services. Il contient les contrats de la logique et processus Business.

    - **services-impl** : module d'implémentation des services. Il contient la logique et processus Business.
- Presentation
    - **presentation-web** : module contenant l'application Web.
- Integration tests
    - **integration-tests** : module contenant les tests d'intégrations de la solution.

Un répertoire supplémentaire *docs* est présent, contenant le fichier LaTeX de ce document ainsi que la JavaDoc du projet.

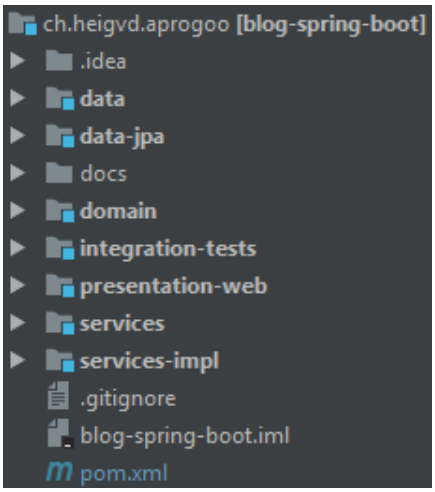

FIGURE 5 – La structure de 1er et de 2ème niveau du projet.

3 Compiler le projet

3.1 Création des packages

Le projet peut être compilé en exécutant la commande Maven suivante :

```
mvn clean package
```

Maven va invoquer son mon mécanisme nommé *Ractor* [1]. Il s'agit d'un système automatiquement utilisé lorsque Maven doit compiler un projet composé de plusieurs modules. Pour résoudre les dépendances entre ces modules (qui ne sont pas installé dans le store *.m2* local), Maven construit l'arborescence des dépendances et place les modules compilés dans un emplacement temporaire, servant ainsi à compiler les modules qui en sont dépendants.

3.2 Exécution de l'application Web

L'application Web peut ensuite être exécutée avec la commande suivante :

1. Maven Reactor : https://maven.apache.org/guides/mini/guide-multiple-modules.html

```
java -jar presentation-web/target/presentation-web-1.0-SNAPSHOT-exec.jar
```

3.3 Création de la JavaDoc

Il est possible de construire la JavaDoc avec la commande Maven suivante :

```
mvn javadoc:aggregate
```

Le paramètre `aggregate` [2] permet de construire la documentation basée sur un projet possédant plusieurs modules.

4 Technologies utilisées

4.1 Couche Data

Spring Data JPA Bibliothèque permettant de manipuler une base de données en fournissant plusieurs outils, tel qu'un CRUD. Il s'agit de la couche de données recommandée par Spring. Une fois les interfaces déclarées, il suffit de configurer l'application (à l'aide du fichier *application.yml*) en lui fournissant les détails de connexion à la base de données pour s'y connecter et y effectuer des requêtes.

Flyway DB Outil permettant d'effectuer des migrations pour une base de données. Lorsque l'application Web démarre, Flyway DB compare automatiquement l'état de la base de données avec tous les scripts de migration et aligne la base de données si nécessaire. Les fichiers de migration sont disponibles dans *data-jpa/src/main/resources/db/migration*.

H2 Database Base de données embarquée dans l'application. Celle-ci est démarrée en même temps que l'application et génère un fichier de stockage dans *~/blog-spring-boot*. L'interface de gestion de la base de données a été configurée dans le fichier *application.yml* pour répondre à l'URL `http://localhost:8080/h2`, à condition que l'application soit démarrée. Lorsque l'application est déployée sur un environnement de production, le fichier de configuration *application-production.yml* réécrit certaines configurations du fichier de base, ne rendant plus disponible l'interface d'administration de la base de données H2, par exemple.

4.2 Couche Domain

Hibernate validator Collection d'annotations permettant d'augmenter les modèles de données avec des informations de validation. Les autres couches peuvent ainsi déduire si l'état d'un modèle est valide pour ensuite être persisté.

2. Maven JavaDoc aggregate : `http://maven.apache.org/plugins-archives/maven-javadoc-plugin-2.10.2/examples/aggregate.html`

4.3 Couche Presentation

Spring MVC Framework fournissant un ensemble de technologies et de bonnes pratiques afin de concevoir un site Web avec une infrastructure MVC.

Thymeleaf Template engine pour le rendu de vues dans le site Web.

DevTools Outils facilitant la création d'une application Web avec Spring Boot. Ceux-ci peuvent recompiler automatiquement les modules si des modifications y sont apportées lorsque le serveur Web fonctionne et rafraichit automatiquement les pages impactées grâce à l'extension LiveReload [3].

Bootstrap Framework front-end rendant le site responsive pour être compatible avec toutes les tailles de périphériques.

CKEditor Outil WYSIWYG permettant à l'utilisateur de formater correctement ses écrits lors de la création des posts.

5 Héritage de configuration

Tous les modules de la solution de développement utilisent Maven, permettant ainsi l'héritage de configuration entre les fichiers *pom.xml*. Le fichier *pom.xml* parent est celui situé à la racine de la solution. Comme ce projet a été développé avec le framework Spring Boot, le fichier parent utilise lui-même le fichier parent de Spring Boot [4].

6 JAR et WAR

Les applications Web traditionnelles sont exportées en une archive WAR, permettant leur exécution dans un serveur Web séparé. Grâce à Spring Boot, il est possible de créer un JAR intégrant le serveur Web ainsi que le système de base de données. L'application peut ensuite être déployée dans un conteneur Docker et dans un Cloud tel que AWS. Pour ce projet, il a donc été choisi d'utiliser l'optique du JAR afin de rendre l'application facilement extensible.

3. Plus d'informations sur LiveReload : `http://livereload.com`
4. Il s'agit d'une configuration recommandée pour Spring Boot : `http://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-maven`.

7 Template engine

7.1 JSP et Thymeleaf

Lors de la réalisation du projet, il a fallu utiliser un système de rendu des vues. Spring MVC utilise la technologie JSP par défaut. Cependant, le framework Spring Boot recommande l'utilisation de la technologie Thymeleaf, qui permet d'effectuer des opérations plus étendues. Pour les amoureux de JSP, un des repositories Github officiel de Spring Boot [5] intègre JSP. Malgré cela, le site officiel de Spring Boot indique que l'utilisation de JSP possède ces limitations [6], par exemple, l'impossibilité de créer un JAR unique pour une application Web.

Pour cette raison, Thymeleaf a été la technologie choisie pour la réalisation de ce projet.

7.2 Thymeleaf layout

Thymeleaf intègre la notion de layouts (ou templates) de deux manières différentes [7] :

— utilisation de système d'inclusion, et
— utilisation d'un système hiérarchique.

Le premier permet de facilement d'inclure un fichier Thymeleaf dans un autre fichier grâce à l'attribut `th:include`. Il s'agit de l'équivalent de la fonction `include` de PHP. Cela peut-être pratique pour un contenu isolé à réutiliser.

> Spring Boot offre le support de beaucoup de technologies pour le rendu de pages. [8]

La deuxième méthode est plus intéressante (mais plus complexe) pour la gestion d'un template qui sera utilisé pour plusieurs pages. Il s'agit de la méthode du donut. Le template définit un cadre d'éléments, comme le préambule HTML, l'en-tête et le pied de page. Au milieu du template, on définit où est le corps de la page et le moteur de rendu de Thymeleaf injectera le contenu de la page en cours à cet endroit. Ce projet utilise cette approche avec le fichier *presentation-web/src/main/resources/templates/layouts/main.html*.

8 Java 8 : Stream

Java 8 introduit une nouvelle fonctionnalité appelée *Stream*. Elle permet d'appliquer des projections ou des sélections sur une collection de données. Il est possible d'en avoir un aperçu dans le service `PostService`. Ce dernier reçoit une liste de posts (dans le contexte d'un blog) d'un repository. Il est donc nécessaire de les trier du plus récent au plus vieux. Voici un aperçu du code permettant d'ordonner les posts selon leur date de publication :

5. Spring Boot avec JSP : https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples/spring-boot-sample-web-jsp
6. Limitation JSP dans Spring Boot : http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-developing-web-applications.html#boot-features-jsp-limitations
7. Plus d'informations sur les layouts en Thymeleaf : http://www.thymeleaf.org/doc/articles/layouts.html
8. Les templating engines supportés par Spring Boot : http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-developing-web-applications.html#boot-features-spring-mvc-template-engines

```java
/**
* Retrieves all the posts.
*
* @return The return value is a collection of {@link Post}.
*/
@Override
public Iterable<Post> findAll() {
return StreamSupport.stream(repository.findAll().spliterator(), false)
        .sorted((o1, o2) -> o2.getPublishedDate()
        .compareTo(o1.getPublishedDate()))
        .collect(Collectors.toList());
}
```

9 Testing

9.1 Tests unitaires

Les technologies suivantes ont été utilisées pour la réalisation de tests unitaires (implémentés dans le module *services-impl*). Ces dernières sont disponibles dans la dépendance `spring-boot-starter-test` pour Spring Boot.

junit Outil fournissant l'infrastructure pour l'exécution de tests unitaires.

hamcrest Fluent API pour l'assertion des tests unitaires. Les assertions deviennent plus lisibles.

mockito Outil de mocking afin d'écrire des tests unitaires en injectant des dépendances fictives basées sur des interfaces.

De nombreux goals Maven exécutent les tests unitaires. Celui spécialisé dans l'exécution de ces tests est : `mvn test`.

9.2 Tests d'intégration

À l'opposé des tests unitaires, les tests d'intégration permettent de tester l'application sans couche d'isolation. Spring Boot supporte aisément les tests d'intégration en décorant les classes de tests par deux annotations :

```java
@RunWith(SpringRunner.class)
@SpringBootTest
```

Cela a pour effet de démarrer l'application (on peut voir le démarrage complet dans la console de log). Les tests d'intégration sont ensuite exécutés sur l'application démarrée. Il est à noter que les tests d'intégration sont localisés dans un module séparé, car teste l'intégralité des autres modules. Cela réduit aussi le problème de dépendances circulaires entre les modules de la solution.

10 JavaDoc

Le plugin Maven de génération de la JavaDoc du projet a été intégré au fichier *pom.xml* parent. Il suffit d'exécuter la commande suivante pour générer la documentation :

```
mvn javadoc:aggregate
```

Le paramètre `aggregate` permet de générer la documentation pour tous les modules d'un projet. La documentation de ce projet a été générée dans le dossier *docs/*.

11 Bonus : Docker and Amazon Web Services

11.1 Création du conteneur

Grâce à la création d'un JAR avec Spring Boot, il est facile d'en créer un conteneur Docker. Le fichier parent Maven contient le plugin permettant de créer des conteneurs Docker avec la commande suivante (pour autant que Docker soit installé sur la machine) :

```
mvn docker:build
```

La commande doit être exécutée dans le module où se trouve le point d'entrée de l'application Web, c'est-à-dire dans le dossier *presentation.web/*. Le conteneur est maintenant disponible dans Docker. Tous les conteneurs peuvent être affichés à l'aide de la commande suivante :

```
docker images
```

Finalement, le conteneur peut être exécuté avec la commande suivante :

```
docker run -it -p 80:8080 blog-spring-boot
```

11.2 Envoi sur Amazon Web Services

Il est possible de créer un compte gratuit et de profiter de quelques services d'Amazon Web Services en créant un compte sur http://aws.amazon.com. Après inscription, le service hébergeant les conteneurs Docker se nomme *EC2 Container Service*. En suivant les quelques étapes, il est possible de créer un repository où il sera nécessaire de s'authentifier, de tag le conteneur sur la machine locale afin de le rendre unique puis finalement de l'upload avec les commandes suivantes :

```
aws ecr get-login --region eu-central-1
docker tag blog-spring-boot:latest
↪ 245527330870.dkr.ecr.eu-central-1.amazonaws.com/blog-spring-boot:latest
docker push
↪ 245527330870.dkr.ecr.eu-central-1.amazonaws.com/blog-spring-boot:latest
```

Ces commandes contiennent des données liées au repository utilisé lors de la réalisation de ce projet. Elles sont données par Amazon au moment de l'upload du conteneur. Amazon indique ensuite les étapes à suivre pour la mise en ligne du conteneur et propose un sous-domaine unique pour l'application, comme illustré sur la figure 6.

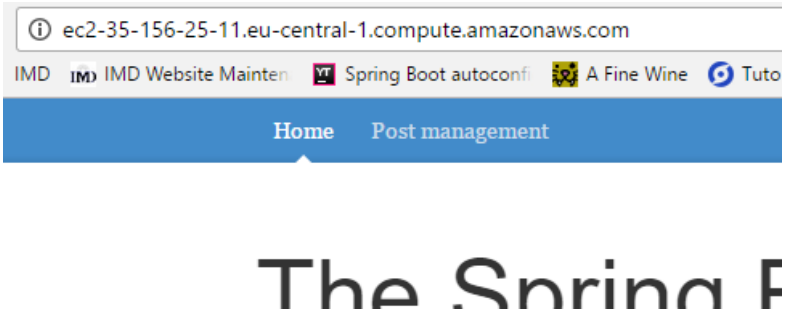

FIGURE 6 – Un exemple de sous-domaine proposé par Amazon pour un conteneur.

12 Sources d'information

Pour la réalisation de ce travail, les cours de l'unité AProgOO ont été la base sur laquelle j'ai fondé mon savoir sur les technologies Java. J'ai ensuite pris la liberté de consulter les vidéos suivantes disponible sur Pluralsight pour en apprendre plus sur l'environnement Spring :

— Spring Fundamentals [9]
— Introduction to Spring MVC [10]
— Introduction to Spring MVC 4 [11]
— Creating Your First Spring Boot Application [12]
— Spring Boot : Efficient Development, Configuration, and Deployment [13]

J'ai ainsi passé plusieurs jours a étudier ce framework qui offre de très bonnes fonctionnalités et possède une grande communauté. Grâce à ces connaissances et maintenant que je suis à l'aise avec la JVM, je peux à présent en apprendre plus sur Scala ainsi que son écosystème.

9. https://app.pluralsight.com/library/courses/spring-fundamentals/
10. https://app.pluralsight.com/library/courses/springmvc-intro/
11. https://app.pluralsight.com/library/courses/spring-mvc4-introduction/
12. https://app.pluralsight.com/library/courses/spring-boot-first-application/
13. https://app.pluralsight.com/library/courses/spring-boot-efficient-development-configuration-deployment/