SITE DE BLOGGING

Dossier de réalisation

FLORIAN DE SAINT LEGER
FLORENCE DELBARRE
LUDOVIC MANTOVANI

A l'issue d'un cours consacré au NoSQL et à la prise en main de MongoDB, il nous a été demandé de créer un site de blogging utilisant une base de données MongoDB.

Table des matières

Contexte

Notre blog « **L'Antre du NoSQL** » est un blog dédié au NoSQL et à son intérêt dans le monde professionel. Nous avons voulu, via des articles complets et construits, donner envie aux lecteurs de s'intéresser davantage à cette nouvelle catégorie de SGBD.

Nous avons privilégié une interface :

- simple
- epurée

En effet, nous avons la volonté d'attirer le lecteur sur le contenu et que son regard soit le moins « parasité » possible par des fonctionnalités inutiles.

.

Les fonctionnalités attendues

Par défaut, il est possible de naviguer dans le blog en cliquant sur le « preview » de l'article pour le lire dans son intégralité.

Pour chaque article, il est possible de laisser un commentaire sans fournir d'adresse mail ou de créer un compte

Nous avons mis en place un système de connexion pour que les administrateurs puissent se connecter et ainsi supprimer, ajouter et modifier des articles. Ces fonctionnalités sont ne sont pas visible pour les visiteurs

Les différentes technologies utilisées

J2ee :

- J2EE ou Java EE fait référence à Java Enterprise Edition et comporte la plate-forme standard (à savoir le Java Stanard Edition) mais aussi un nombre de bibliothèques conséquente et remplissant davantage de fonctionnalité.
- Notre choix s'est porté sur ce langage pour son utilisation grandissante dans le milieu professionnel et aussi, via des frameworks, la simplicité et les apports supplémentaires

Spring :

- Spring est une framework permettant le développement d'applications Java et est devenu un standard dans le monde professionnel. Spring permet de réduire les dépendances, faciliter les tests, simplifier le code et organiser le développement.
- Notre choix s'est porté sur Spring Boot puisque celle-ci propose une approche plutôt dogmatique et permet de s'affranchir des difficultés de configuration de Spring. De plus, cette mouture du framework permet d'embarquer un serveur Tomcat nous permettant de nous affranchir de la configuration parfois lourde d'un serveur Tomcat, configuration pouvant parfois être lourde.

Maven :

- Maven permet entre autre une gestion simple des dépendances via l'ajout de celles-ci dans le fichier pom.xml à la racine de l'application.

MongoTemplate :

- Spring met à disposition une librairie permettant la communication entre une application J2EE et nos données MongoDB.

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
</dependency>
```

Figure 1 : dépendance à ajouter dans pom.xml

- **TinyMCE** : éditeur WYSIWYG (What you see is what you get) opensource basé sur les technologies Javascript et HTML

L'architecture des données et les requêtes

Le développement de l'application avec J2EE et Spring permet d'organiser notre code avec trois types de classes distincts

Pour chaque collection :

- Création des entités .java avec les attributs et les accesseurs en lecture et en écriture.
-

```
@Document
public class User {

    @Id
    private String id;
    private String pseudonyme;
    private String password;
```

Figure 2: Entité avec les annotations @Document et @Id

- Création des classes permettant les requêtes.

```
@Service
public class UserRepository {

    @Autowired
    MongoTemplate mongoTemplate;

    static final Logger logger = LoggerFactory.getLogger(User.class);

    public User getUser(String pseudonyme, String password)
    {
        if(mongoTemplate.collectionExists(User.class))
        {
            Query query =  new Query(Criteria.where("pseudonyme").is(pseudonyme).and("password").is(password));
            User result = mongoTemplate.findOne(query, User.class);
            return result;
        }
        else
        {
            //TODO raise error
            logger.error(Post.class.toString() + " collection not found");
        }
        return(null);
    }

}
```

Figure 3 : méthode pour effectuer des requêtes

- ✓ On recherche l'instance MongoTemplate afin d'effectuer les requêtes.
- ✓ La liste des méthodes qui peuvent être utilisées dans le reste du code.

Pour l'ensemble des collections :

- Création des **contrôleurs** afin de gérer les données reçus et envoyés par la vue.

```
//affiche la page post.jsp pour lire un post dans son intégralité
@RequestMapping( value = "post/{id}" , method = RequestMethod.GET )
    public ModelAndView afficherPost( @PathVariable( "id" ) final String id ){
        final ModelAndView mav = new ModelAndView ("post");
        Post post=postRepo.getPostById(id);
        mav.addObject( "post", post );
        return mav;
    }
```

Figure 4: la méthode afficherPost

- Lorsque l' URI est **/post/{id}** (id étant une chaîne de caractère) avec une méthode **GET,** nous faisons appel à la méthode **getPostbyId** défini dans la classe PostRepository. L'objet Post ainsi récupéré est envoyé dans la vue post.jsp.

Les sauvegardes/restaurations

MongoDB met à disposition des outils pour sauvegarder et restaurer des bases de données

- **Mongodump** pour la sauvegarde : via la commande mongodumb

  mongodump --collection post --db test : permet donc de créer une sauvegarde de la collection post de la base de données test dans le sous repertoire dump/ du répertoire de travail courant.
- **Mongorestore** pour la restauration : par defaut mongorestore cherche les fichiers de sauvegarde dans le repértoire dump/ évoqué ci-dessus

  mongorestore --dbpath "chemin_db" "chemin_backup"

Le manuel d'administration

- Il est inutile de configurer une base de données MongoDB en local puisque le code fait appel à une base distante.

  En effet dans le fichier application.properties référençant les diverses propriétés de notre application :

  spring.data.mongodb.host=92.155.20.108
  spring.data.mongodb.port=27017

  Si on souhaite changer de base de données et travailler en local, il suffit de préciser la valeur « **localhost** » pour spring.data.mongodb.host et de démarrer MongoDB.

- Afin de modifier le code source, il faudra importer un projet dans Eclipse.

  File > Import > Existing Maven Projects

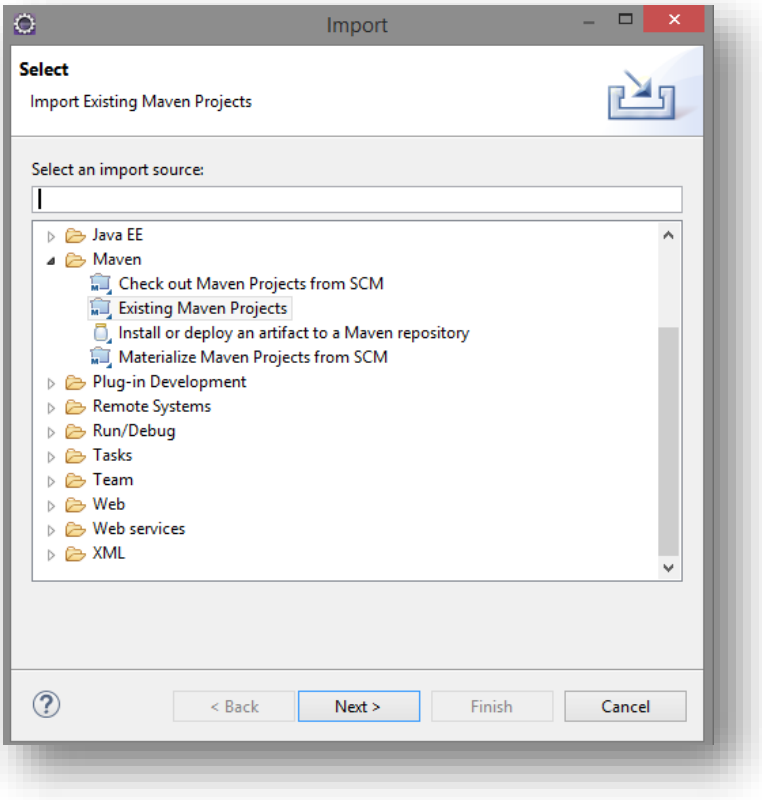

Figure 5 : Importation d'un projet MAVEN - Etape 1

Puis sélectionner le dossier contenant la source du code. Il faut alors cocher le fichier pom.xml. Puis cliquer sur « Finish »

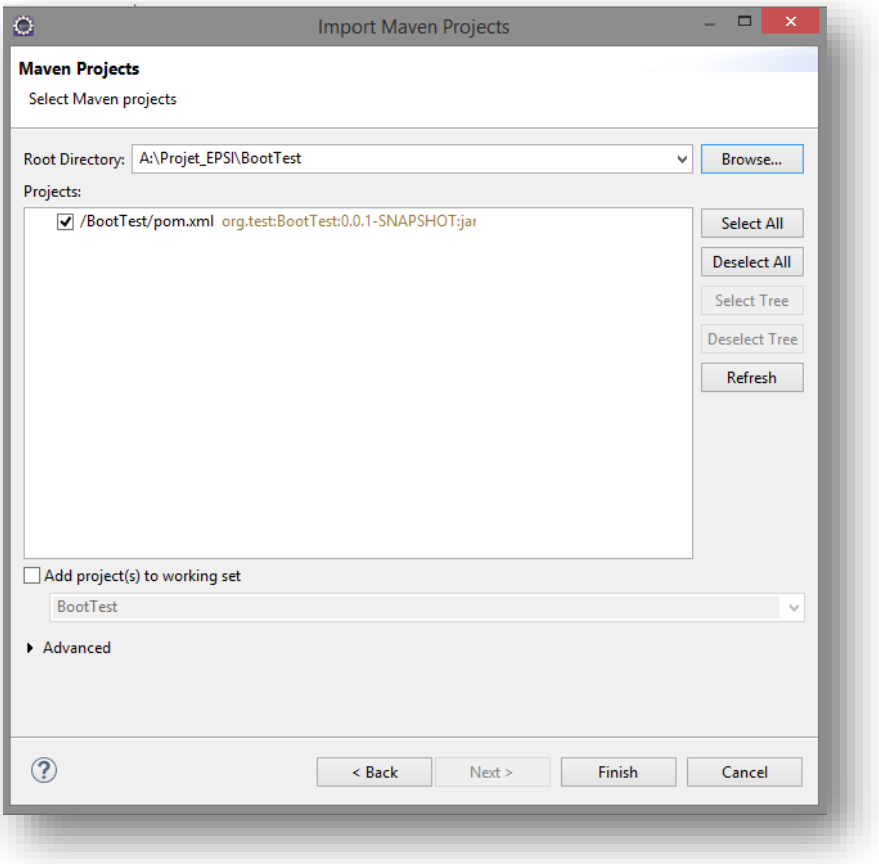

Figure 6: Importation d'un projet MAVEN - Etape 2

Il ne faut pas oublier d'effectuer un Maven Update.
Pour lancer l'application, il faut effectuer un clic droit sur le projet et Run As … Java Application.

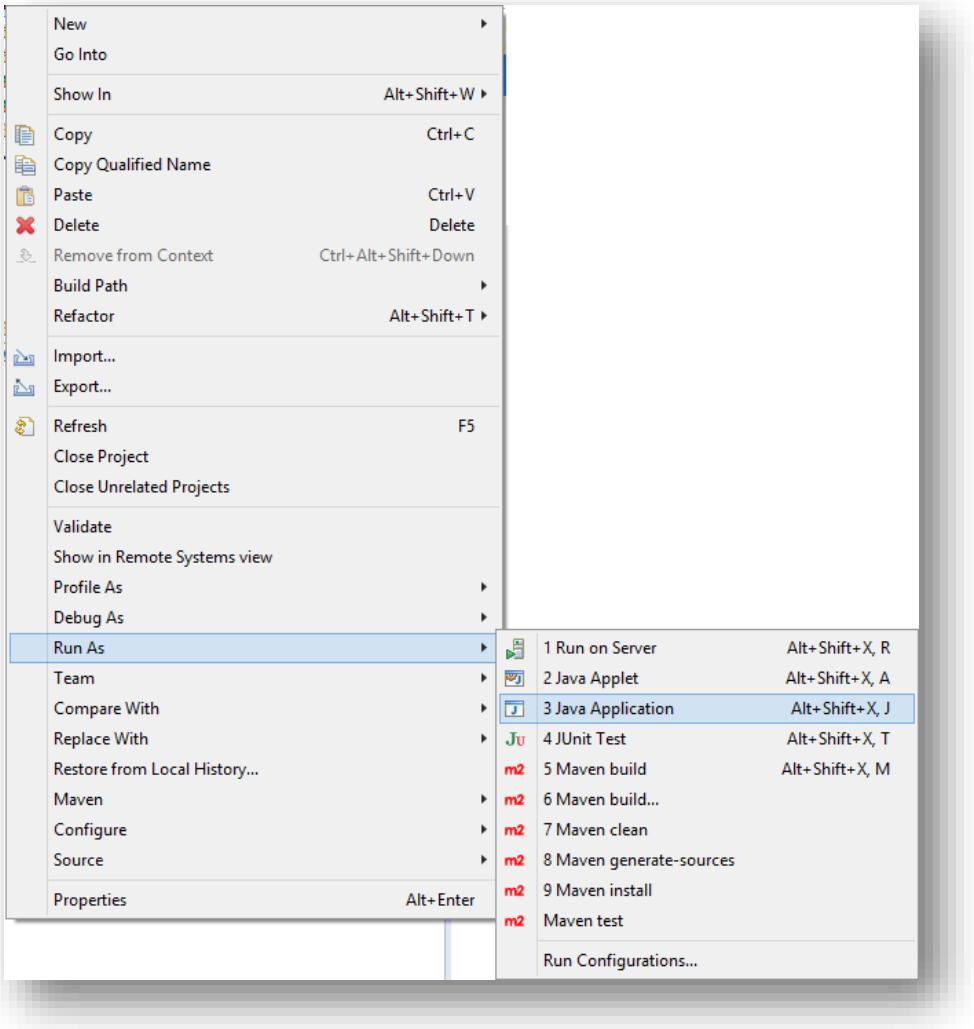

Figure 7 : Lancer l'application

Puis sélectionner **BootTestApplication**.

Vérifier dans la console que l'application a correctement démarré.

```
2015-04-30 09:48:08.318  INFO 3736 --- [           main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2015-04-30 09:48:08.337  INFO 3736 --- [           main]
BootTest.BootTestApplication             : Started BootTestApplication in 53.514
seconds (JVM running for 56.316)
```

L'application sera ainsi donc visible via un explorateur web à l'adresse **http://localhost:8080**.

Il est utile de rappeler que notre application exige l'installation d'un environnement java (Java SE Development Kit 8 disponible ici).

Les administrateurs du blog sont les suivants :

Username	Password
admin	nosql
jack	pwd

john	password
kate	mdp