

Guide : SimpleChat avec Spring

Important : Tout ce que je dis ici, est le fruit de mes recherches et de mon ressenti personnel. Je suis loin d'être un expert en Spring, et j'ai le même niveau que vous en Java. Il peut donc arriver que des erreurs soient présentes dans ce document, mais j'espère qu'il permettra de vous aider dans la compréhension de ce framework et qu'il nous permettra d'être efficient dans la réalisation de SimpleChat et du projet de Gestion de Projet.

Objectif de ce guide :

En résumé : combiner Spring et OCSF

Ici, nous allons voir ensemble comment créer un projet console en Java, avec Spring. Ainsi, nous pourrons utiliser Spring pour réaliser SimpleChat. En effet, utiliser Spring n'empêche pas d'utiliser OCSF.

En plus, Spring permet aussi de faire du REST, ça peut vous être utile pour le cours de Gestion de Projet !

Pourquoi Spring et pas autre chose ?

En résumé : Spring est un des plus gros Framework Java, stable et puissant.

Spring fait parti des Framework Java les plus répandu, si ce n'est pas LE plus répandu. Il s'utilise avec les gestionnaires de dépendances Maven ou Gradle. Spring est constitué de plein d'éléments. Il n'est pas obligatoire d'utiliser tout Spring (il est très peu probable qu'un projet nécessite toutes les fonctionnalités couvertes par Spring, et certainement pas SimpleChat).

Ainsi, Spring est constitué d'un core, un noyau obligatoire, autour duquel gravite une nébuleuse d'extension. Le core va s'occuper de créer le contexte de Spring avec l'injection de dépendance, et les extensions permettent par exemple de gérer les requêtes à la base de données, ou bien d'implémenter des fonctions de sécurité, ou encore d'implémenter une architecture MVC... bref, les options sont nombreuses, rendez-vous sur le site officiel pour en apprendre plus.

SpringBoot permet de gérer les dépendances entre les différents modules de Spring. Donc ce n'est pas tout à fait un projet Spring qu'on va faire, mais un projet SpringBoot, c'est bien plus facile.

Spring est beaucoup utilisé en Java EE (JEE ou J2E). Donc, souvent utilisé pour faire du Web : soit en REST (coucou cours de Gestion de Projet) soit en MVC avec un contexte classique ou on produit des pages

JSP. Ici, on va faire ni l'un ni l'autre : nous allons utiliser Spring pour faire un projet standalone, qui tourne dans la console.

Qu'est ce que ça va nous apporter ?

En résumé : Spring peut déjà nous servir pour : les requêtes à la base de données avec des Repositories, les fichiers de configurations, l'injection de dépendance.

Comme dit plus haut, *Spring* fait pleins de choses. Mais pour vous montrer ce qu'il peut nous apporter, prenons quelques tâches à réaliser dans *SimpleChat* :

Persistence des données : Il nous faudra une base de données. Donc des requêtes *SQL*, certainement du *CRUD*, créer des objets Java avec les objets de la *BDD* (avec *JPA* comme on faisait en M1) etc... Pour cette tâche, il existe *Spring Data Jpa* ! Cette extension *Spring* inclut *JPA*, *Hibernate*, et un mécanisme qui crée la majorité des requêtes *SQL* tout seul (oui oui, tout seul, pour de vrai, vous n'avez ni à configurer ni à coder). En gros : vous annotez vos classes métier comme on faisait avec *JPA*, et *Spring* va s'occuper de créer une classe appelée « Repository », dont les méthodes seront créées au moment du build, et qui s'occupera de la majorité des interactions entre la classe métier et la *BDD*. En bonus, on pourrait utiliser un gestionnaire de migration de la *BDD* comme *Flyway* ou *Liquibase*, pour garder notre code *SQL* dans notre projet Java, et ainsi si je vous donne mon projet, vous aurez également ma base de données (pour peu que vous ayez le même *SGBD* que moi d'installer).

Internationalisation : Il faut pouvoir définir plusieurs langages. Pour ça, on a l'instinct d'utiliser des bundles. Pour ceux qui ne savent pas ce que c'est : ceux sont des fichiers dans lequel on associe des valeurs à des mots clefs. Par exemple, on pourrait avoir un *bundle.fr-FR* qui contiendrait *title = mon titre*. Un *bundle.en-EN* qui contiendrait *title = my title*. Et ainsi de suite. C'est bien beau tout ça, mais comment on va taper dans ce fichier ? Comment va-t-on passer d'un fichier à un autre en fonction de la langue choisi ? Hé ben... *Spring* s'en occupe ! *Spring* fonctionne par défaut avec des fichiers *.properties* qui permettent de configurer ce que bon nous semble, et dispose de profils permettant de passer d'un fichier à un autre.

Pour le code en général : L'injection de dépendance. Ce n'est pas demandé dans le projet, mais ceux qui ont déjà travaillé avec de l'injection de dépendances savent comme c'est agréable. Et avec *Spring*, c'est très facile. Je ne vais pas expliquer ce que c'est ici, puisque c'est assez complexe, mais je réaliserai un exemple très simple à ce sujet dans la suite de ce guide.

Qu'est ce qu'on va faire dans ce guide ?

Créer un projet *SpringBoot* en version console (standalone) dans lequel vous pourrez utiliser *OCSF*. Il est important de savoir qu'il existe des outils sur le site officiel de *Spring* permettant de générer des projets. Mais ici, on va le faire à l'ancienne avec nos mains. C'est un choix complètement personnel, je trouve qu'on comprend mieux ainsi.

Donc dans l'ordre, nous allons :

1. Créer un projet *Maven* (ça aurait pu marcher avec *Gradle*)
2. Modifier le *pom.xml* pour inclure *SpringBoot*.

3. Créer le *main* permettant de lancer l'application.
4. Créer une classe annoté *@Service* pour démontrer l'intérêt de l'injection de dépendance.
5. Créer un fichier *.properties* et voir comment récupérer ces propriétés en Java.
6. Inclure *Spring Data JPA*, la connexion à une base *PostgreSQL*, et créer une *BDD* vide dans *PostGre*.
7. Mettre en place *FlyWay* pour taper notre code *SQL* dans le projet Java.
8. Créer une petite classe métier, avec un *repository* pour illustrer la persistance des données dans *Spring*.

Important : Avant de démarrer, rendez vous sur le repo git :

https://github.com/CedricMtta/Guide_SimpleChatSpring

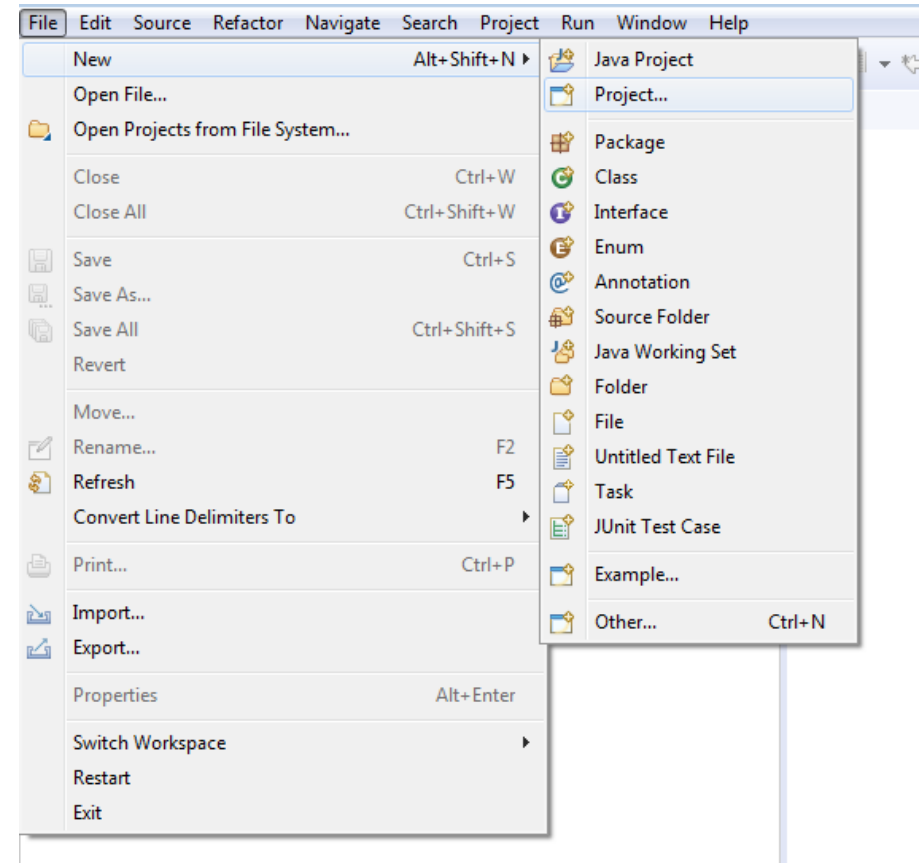
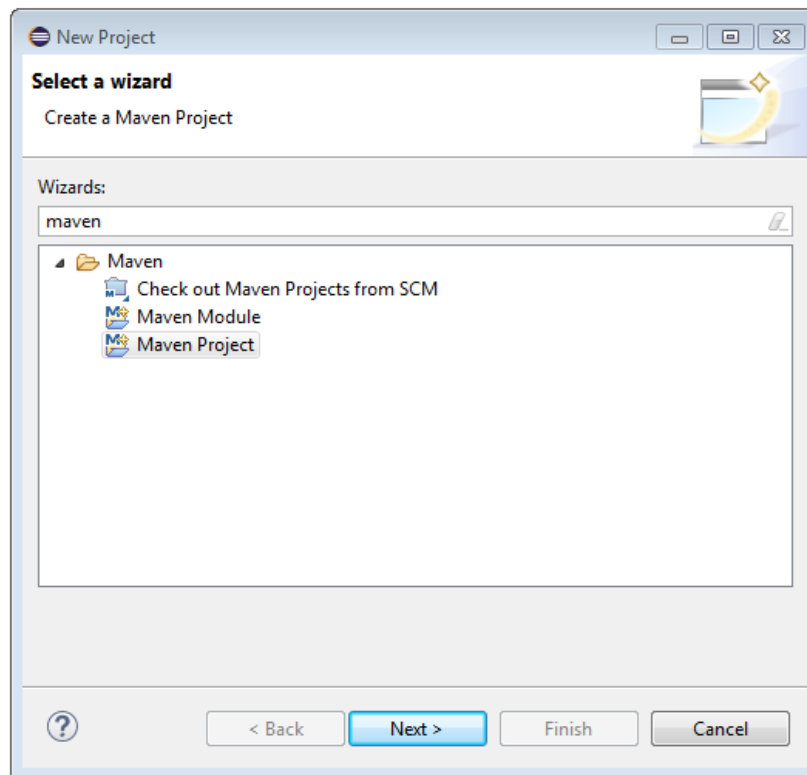
Cela vous évitera de retaper ce qui est contenu dans les capture d'écran, ou de récupérer directement le projet complet ! 😊

Réalisation :

Création du projet :

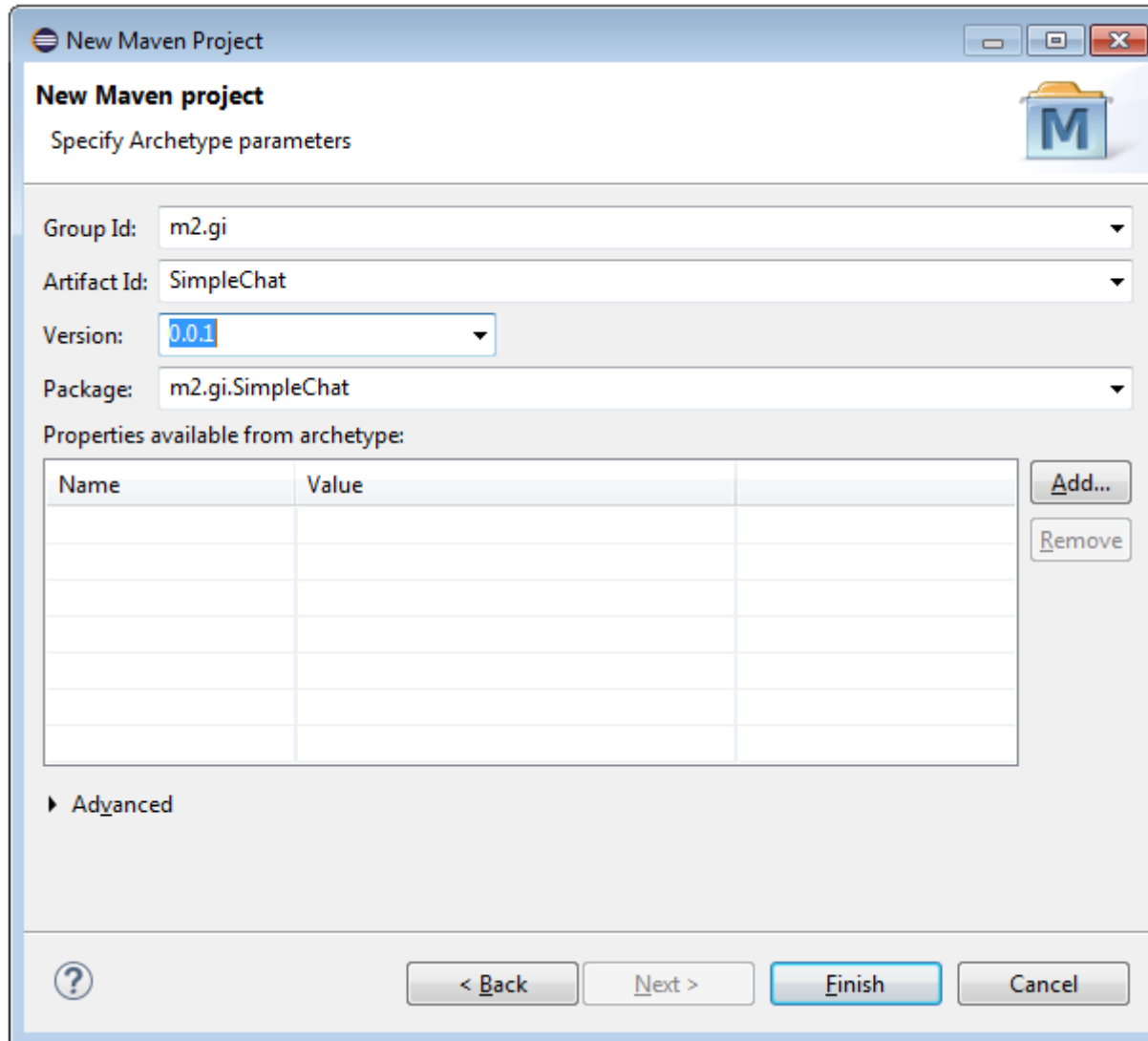
On ouvre Eclipse, direction :

File > New > Project > Maven Project



Cédric MOSCHETA

Cliquez ensuite sur suivant jusqu'à tomber sur la page de configuration du projet Maven, et complétez le formulaire comme ci-dessous :



New Maven Project

Specify Archetype parameters

Group Id: m2.gi

Artifact Id: SimpleChat

Version: 0.0.1

Package: m2.gi.SimpleChat

Properties available from archetype:

Name	Value

► Advanced

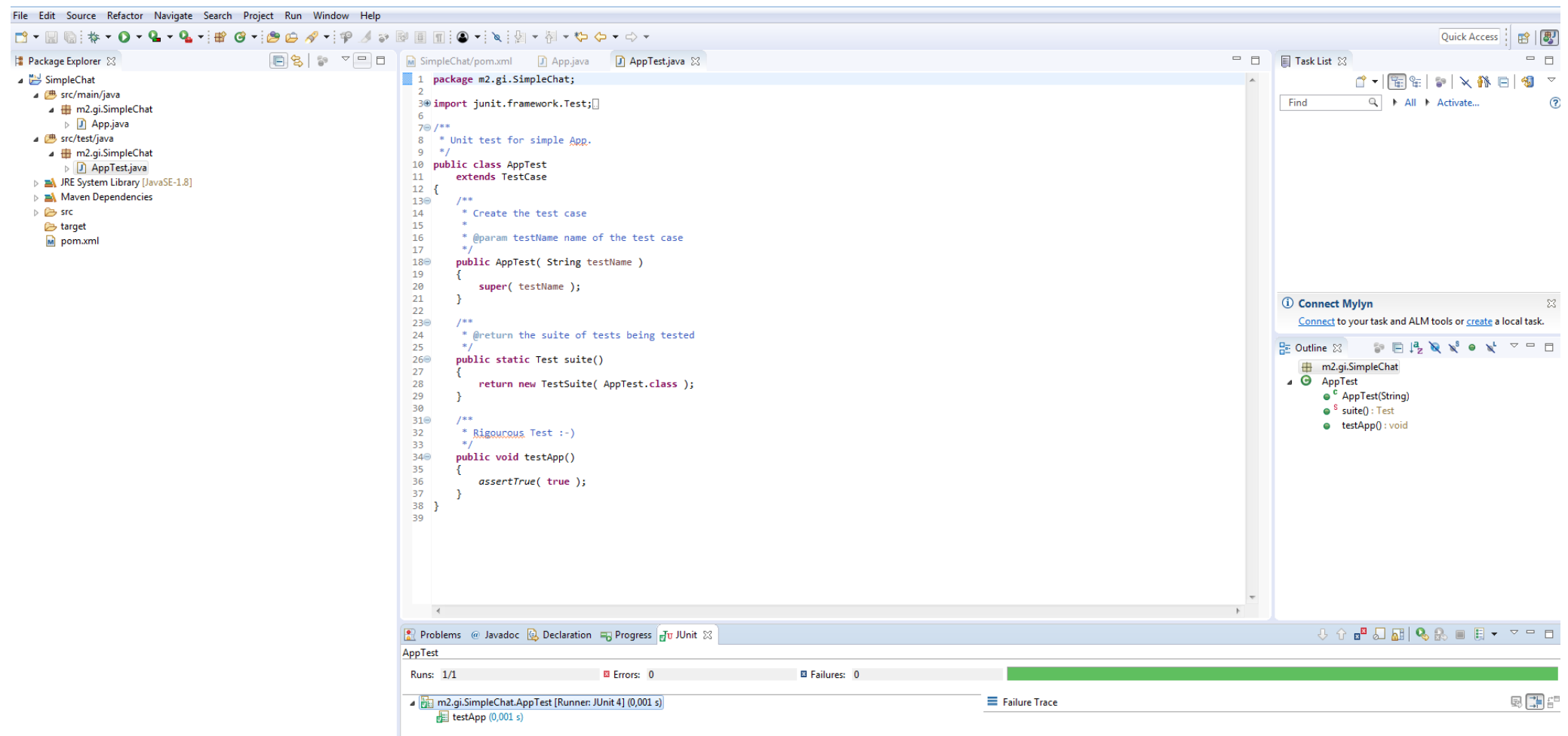
< Back Next > Finish Cancel

Remplacez le contenu du pom.xml par celui-ci-dessous :

```
SimpleChat/pom.xml App.java AppTest.java
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4
5   <groupId>m2.gi</groupId>
6   <artifactId>SimpleChat</artifactId>
7   <version>0.0.1</version>
8   <packaging>jar</packaging>
9
10  <name>SimpleChat</name>
11  <url>http://maven.apache.org</url>
12
13  <properties>
14    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15    <java.version>1.8</java.version>
16  </properties>
17
18  <parent>
19    <groupId>org.springframework.boot</groupId>
20    <artifactId>spring-boot-starter-parent</artifactId>
21    <version>1.5.9.RELEASE</version>
22  </parent>
23
24  <dependencies>
25    <!-- le core de Spring -->
26    <dependency>
27      <groupId>org.springframework.boot</groupId>
28      <artifactId>spring-boot-starter</artifactId>
29    </dependency>
30    <!-- Spring a aussi une extension pour faire des test unitaire : elle regroupe plusieurs framework de test : JUnit, Mockito, Harmcrest. -->
31    <dependency>
32      <groupId>org.springframework.boot</groupId>
33      <artifactId>spring-boot-starter-test</artifactId>
34      <scope>test</scope>
35    </dependency>
36  </dependencies>
37
38  <build>
39    <plugins>
40      <!-- Permet de packager le projet pour la prod. SpringBoot dispose d'un plugin exprès pour ça avec Maven. -->
41      <plugin>
42        <groupId>org.springframework.boot</groupId>
43        <artifactId>spring-boot-maven-plugin</artifactId>
44      </plugin>
45    </plugins>
46  </build>
47 </project>
48
```

Cédric MOSCHETA

Vous devriez voir Eclipse qui commence à télécharger les dépendances une fois le pom sauvegardé. On lance le test généré par Maven pour vérifier que tout va bien !



Remplaçons le contenu de la classe App par celui-ci, qui constituera notre main avec Spring !

```

1  package m2.gi.SimpleChat;
2
3  import static java.lang.System.exit;
4
5  import org.springframework.boot.Banner;
6  import org.springframework.boot.CommandLineRunner;
7  import org.springframework.boot.SpringApplication;
8  import org.springframework.boot.autoconfigure.SpringBootApplication;
9  /**
10   * SimpleChat with SpringBoot.
11   *
12   */
13
14  @SpringBootApplication
15  public class App implements CommandLineRunner {
16
17
18
19      public static void main(String[] args) throws Exception {
20          // disabled banner, don't want to see the spring logo
21          SpringApplication app = new SpringApplication(App.class);
22          app.setBannerMode(Banner.Mode.OFF);
23          app.run(args);
24      }
25
26      @Override
27      public void run(String... args) throws Exception {
28
29          System.out.println("Hello from SimpleChat !");
30          exit(0);
31      }
32  }

```

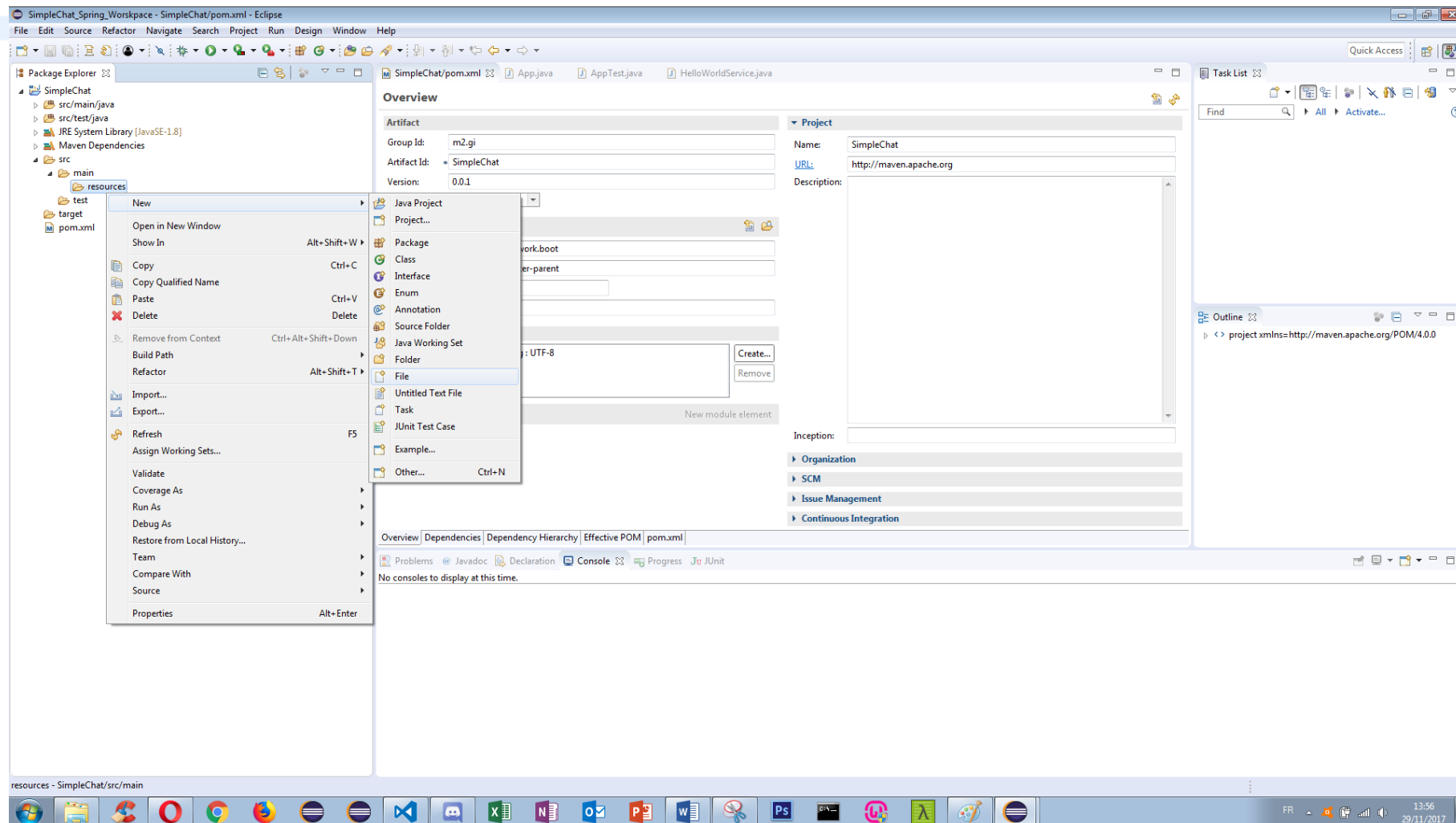
Tips : Utilisez **CTRL+SHIFT+O** pour manager les imports 😊 Vous pouvez lancer l'application si vous voulez, vous verrez alors dans la console le message *Hello from SimpleChat !* apparaître.

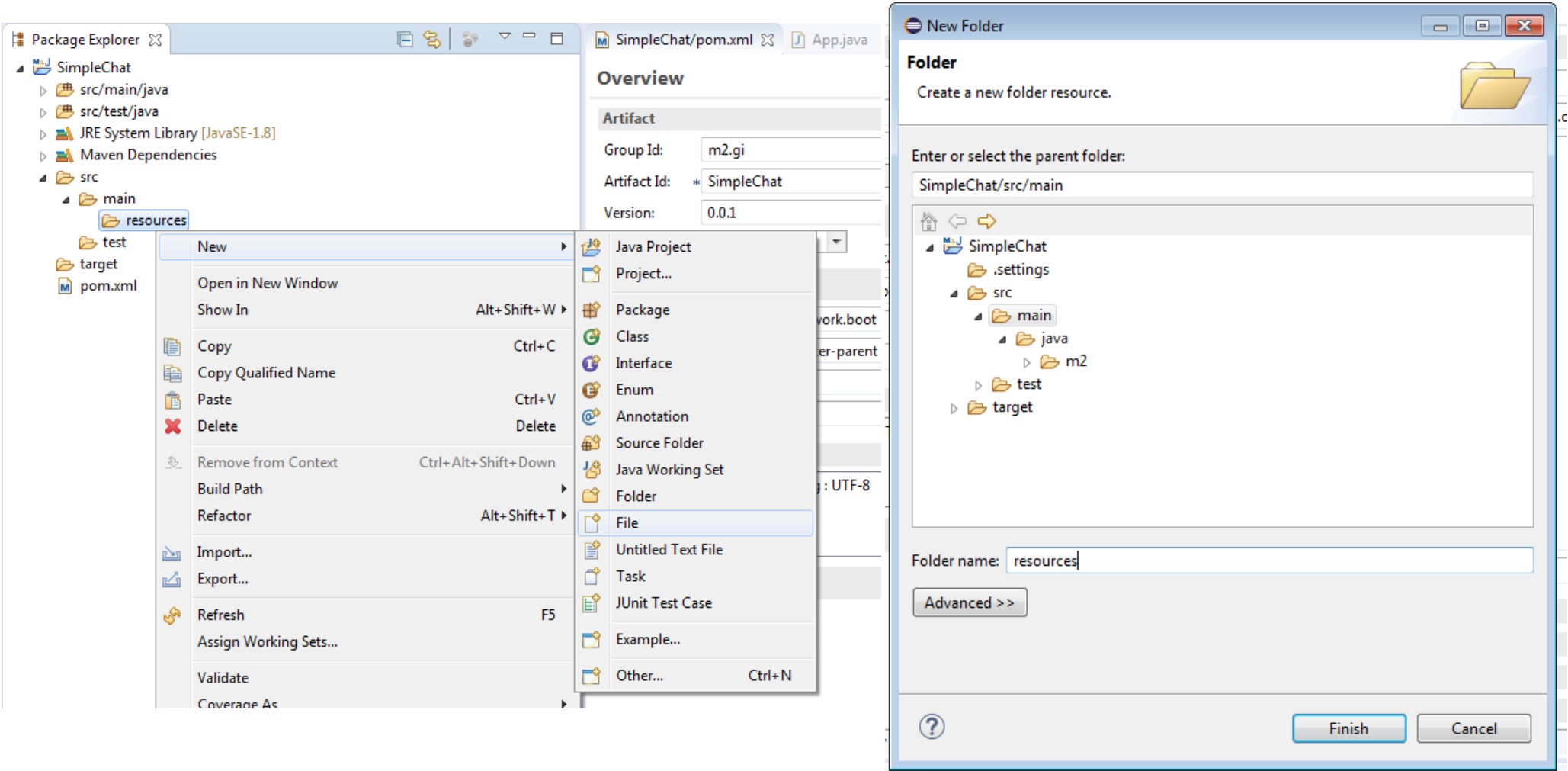

```

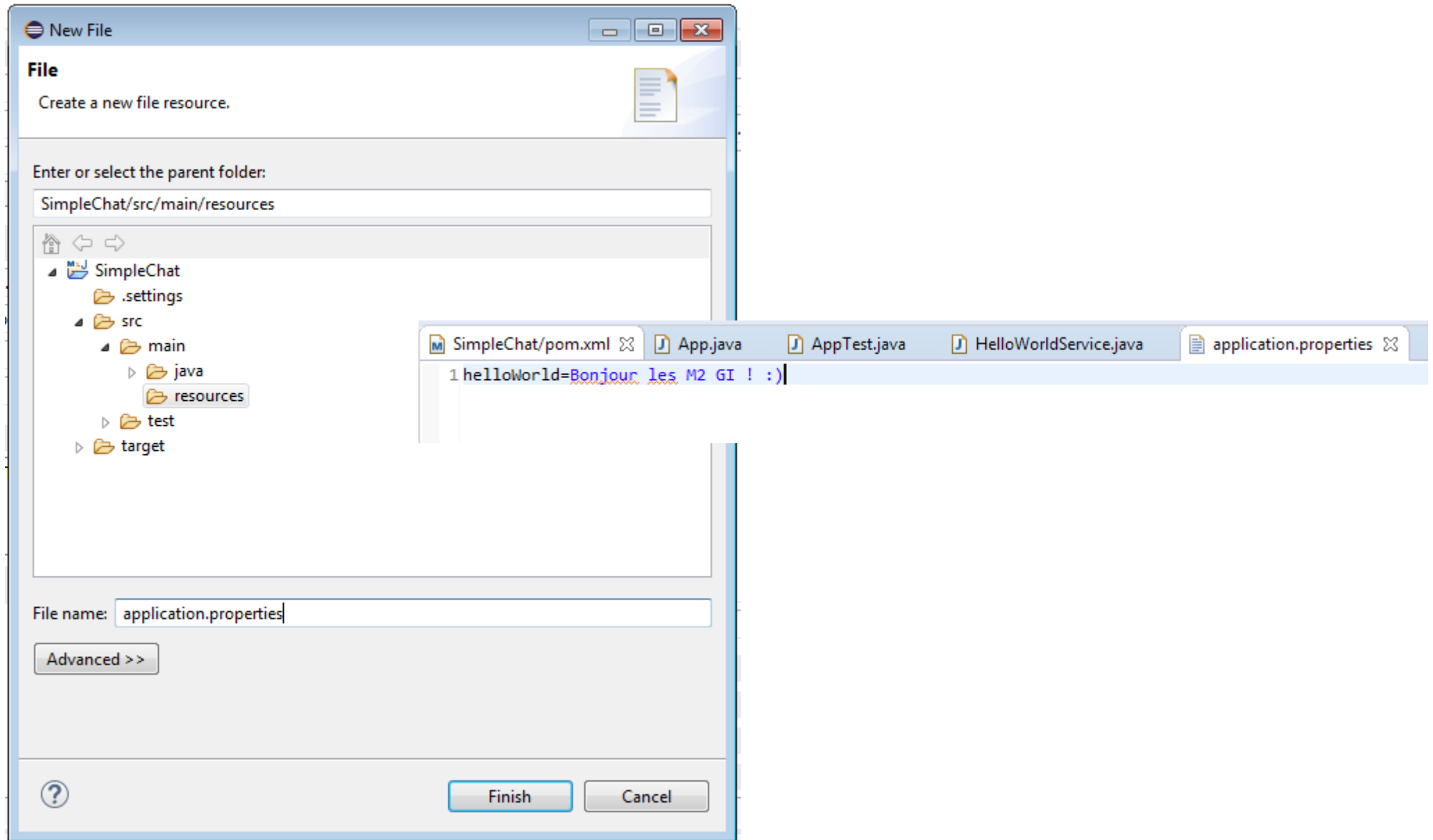
1 package m2.gi.SimpleChat;
2
3 import org.springframework.beans.factory.annotation.Value;
4
5 @Service
6 public class HelloWorldService {
7
8     @Value("${helloWorld:unknown}")
9     private String message;
10
11     public String getMessage() {
12         return message;
13     }
14 }
15
16 }

```

On crée un service bidon qui fait appel à une propriété, et le fichier propriété qui lui est associé !







The screenshot shows an IDE with the following tabs: SimpleChat/pom.xml, App.java, AppTest.java, HelloWorldService.java, and application.properties. The main editor displays the code for App.java, which is a Spring Boot application. The code includes imports for Spring framework components, a package declaration, and a main method that runs the application. The console output at the bottom shows the execution of the application, including the startup of the Spring container and the output of the main method.

```

1 package m2.gi.SimpleChat;
2
3 import static java.lang.System.exit;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.Banner;
7 import org.springframework.boot.CommandLineRunner;
8 import org.springframework.boot.SpringApplication;
9 import org.springframework.boot.autoconfigure.SpringBootApplication;
10 /**
11  * SimpleChat with SpringBoot.
12  *
13  */
14
15 @SpringBootApplication
16 public class App implements CommandLineRunner {
17
18     @Autowired
19     HelloWorldService hw;
20
21     public static void main(String[] args) throws Exception {
22         // disabled banner, don't want to see the spring logo
23         SpringApplication app = new SpringApplication(App.class);
24         app.setBannerMode(Banner.Mode.OFF);
25         app.run(args);
26     }
27
28     @Override
29     public void run(String... args) throws Exception {
30
31         System.out.println(hw.getMessage());
32         exit(0);
33     }
34 }

```

Console Output:

```

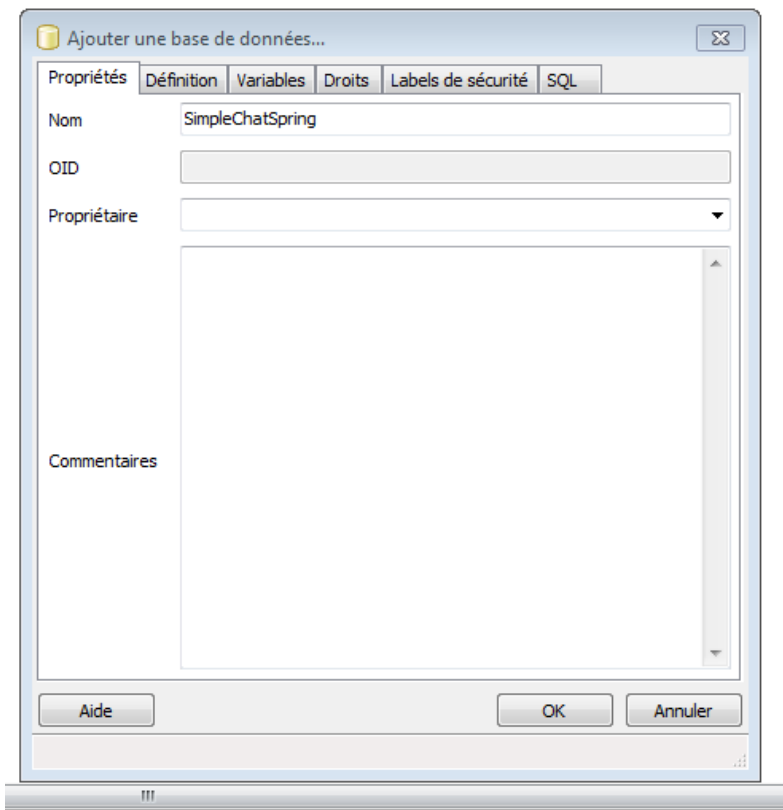
<terminated> App [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (29 nov. 2017 à 14:03:28)
2017-11-29 14:03:29.492 INFO 7300 --- [main] m2.gi.SimpleChat.App : Sta
2017-11-29 14:03:29.496 INFO 7300 --- [main] m2.gi.SimpleChat.App : No
2017-11-29 14:03:29.548 INFO 7300 --- [main] s.c.a.AnnotationConfigApplicationContext : Ref
2017-11-29 14:03:30.146 INFO 7300 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Reg
Bonjour les M2 GI ! :)
2017-11-29 14:03:30.159 INFO 7300 --- [Thread-2] s.c.a.AnnotationConfigApplicationContext : Clo
2017-11-29 14:03:30.160 INFO 7300 --- [Thread-2] o.s.j.e.a.AnnotationMBeanExporter : Unr

```

Cédric MOSCHETA

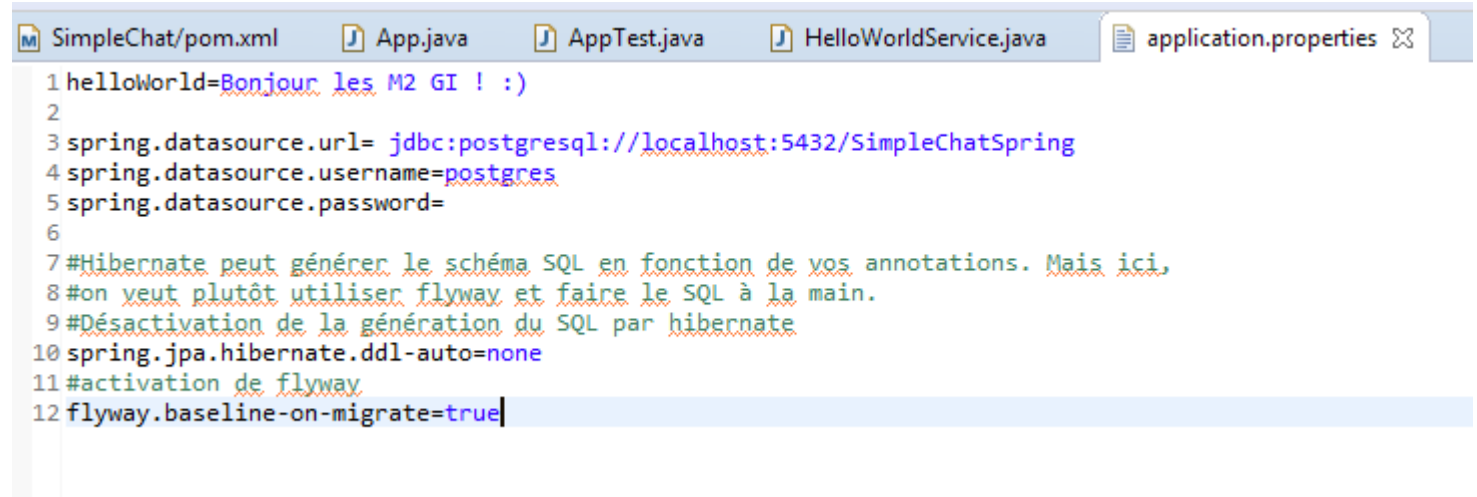
On récapitule : Nous avons notre *main* qui initialise le *core* de *Spring*. Ce même *main* va appeler une méthode de notre Service *HelloWorldService*, qui lui-même affiche une valeur contenue dans le fichier *.properties*. Vous voyez l'annotation *@Autowired* dans le *main* ? C'est ça, l'injection de dépendances 😊

Maintenant, on peut passer à la partie la plus fun : la persistance des données. On va donc, comme promis, créer une *BDD* dans *PostgreSQL* qui sera vide. On va rajouter les dépendances de *Spring Data Jpa*, de *PostgreSQL* et de *FlyWay* à notre *pom.xml* et enfin on va créer une classe métier appelée « *Message* » qui contiendra seulement une *String*, on va également créer son schéma *SQL* avec *FlyWay*, et enfin on va se servir de notre *main* pour rendre un objet de la classe *Message* persistant ! C'est parti 😊



<= La création de la BDD, nommée SimpleChatSpring

```
</dependency>
<!-- Spring Data Jpa, pour la persistance des données. Utilise Hibernate & JPA -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<!-- La connexion à PostgreSQL -->
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
<!-- FlyWay : gestionnaire de migration de la bdd -->
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-core</artifactId>
</dependency>
..
```

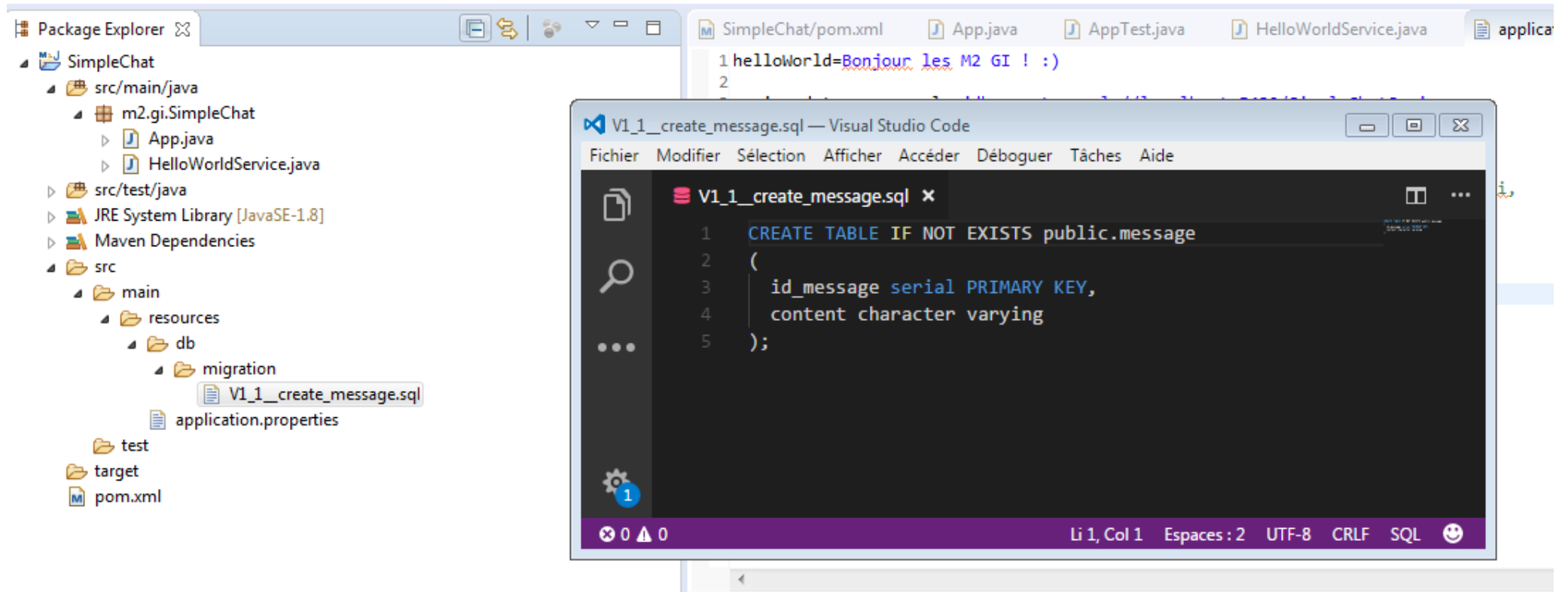


```
1 helloWorld=Bonjour les M2 GI ! :)
2
3 spring.datasource.url= jdbc:postgresql://localhost:5432/SimpleChatSpring
4 spring.datasource.username=postgres
5 spring.datasource.password=
6
7 #Hibernate peut générer le schéma SQL en fonction de vos annotations. Mais ici,
8 #on veut plutôt utiliser flyway et faire le SQL à la main.
9 #Désactivation de la génération du SQL par hibernate
10 spring.jpa.hibernate.ddl-auto=none
11 #activation de flyway
12 flyway.baseline-on-migrate=true
```

Dans le dossier *resources*, rajoutez un dossier *db*, et dans ce dossier *db* rajoutez un dossier *migration*.

C'est ici qu'on va créer nos schémas SQL !

Respectez bien la notation : `V1_1__create_message.sql` signifie : version 1.1 de la BDD, création de message.



Quand on relance notre application, une table « schema_version » est créée.

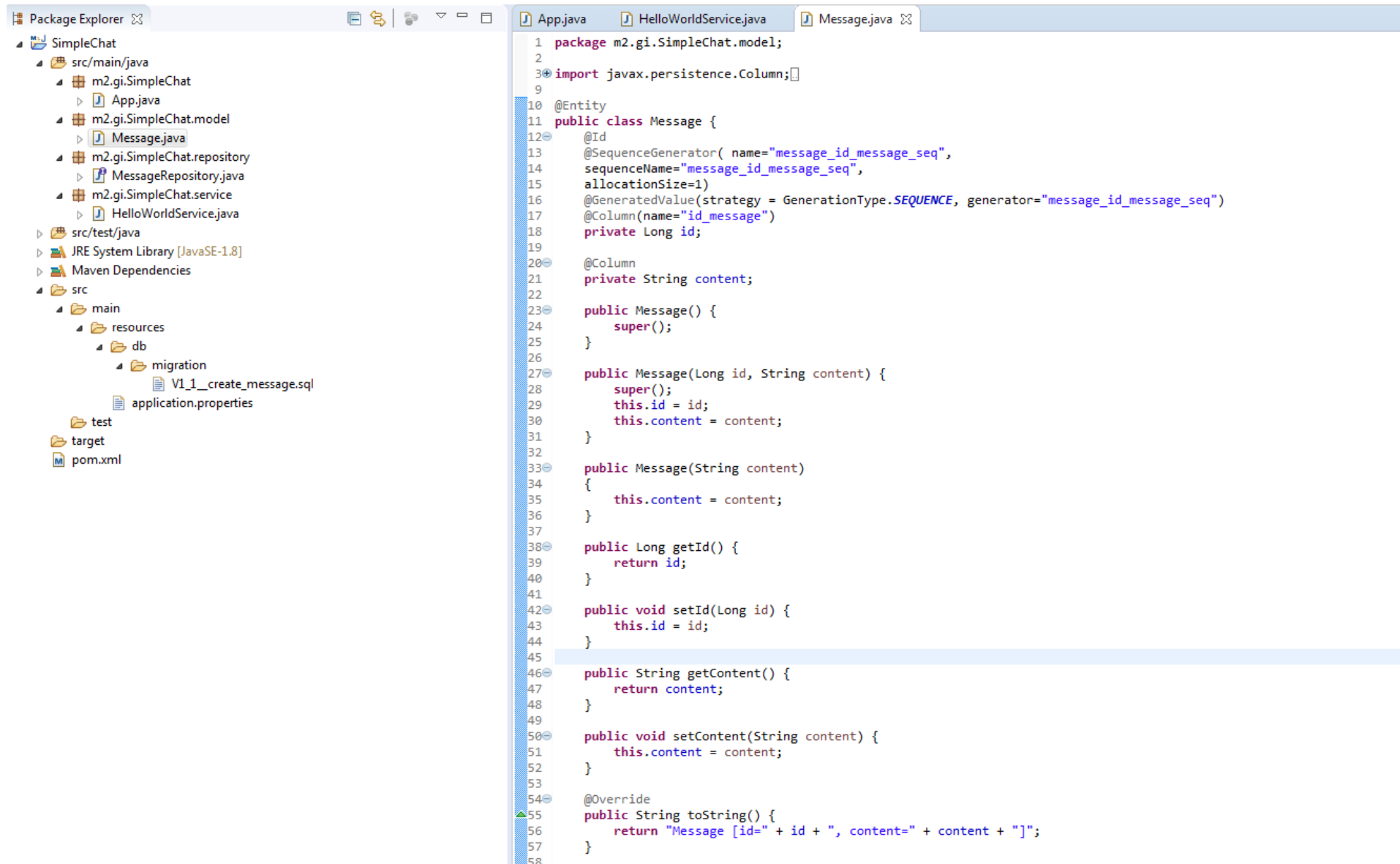
S'ils n'apparaissent pas dans `schema_version`, ils sont exécutés et ajouté à la table.

The screenshot shows the pgAdmin 4 interface. On the left, the 'public' schema is selected, showing its contents: 'message' (with columns 'id_message' and 'content'), 'schema_version' (with columns 'version_rank', 'installed_rank', 'version', 'description', 'type', 'script', 'checksum', 'installed_by', 'installed_on', 'execution_time', 'success'), and 'views'. The 'schema_version' table is highlighted. On the right, the 'Édition des données' window for 'localhost (localhost:5432) - SimpleChatSpring - public.schema_version' is open, displaying the table's data. The table has 11 columns: 'version_rank' (integer), 'installed_rank' (integer), 'version' ([PK] character varying(50)), 'description' (character varying(200)), 'type' (character varying(20)), 'script' (character varying(1000)), 'checksum' (integer), 'installed_by' (character varying(100)), 'installed_on' (timestamp without time zone), 'execution_time' (integer), and 'success' (boolean). The first row shows data for version 1.1, installed on 2017-11-29 14:17:41.432399, with a success status of TRUE.

[illegible]

Cédric MOSCHETA

On réorganise nos fichiers. Vérifiez bien que tous les dossiers et fichiers que vous créez sont en dessous de App.java ! En effet, Spring scan les classes à partir de l'endroit où se trouve App.java. Donc, si vous mettez vos packages au-dessus d'App.java ou au même niveau, Spring ne les détectera pas. On en profite pour créer notre classe Message !

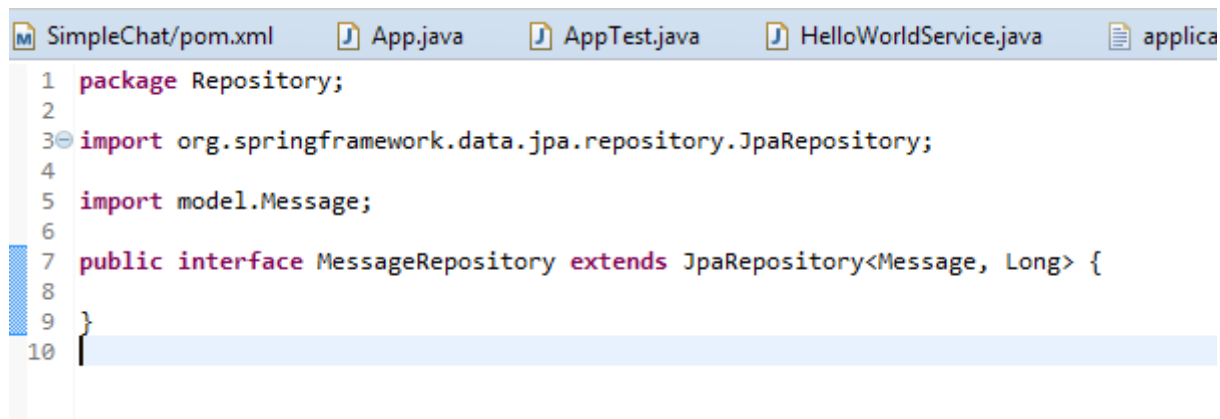


The screenshot displays an IDE interface with two main panels. The left panel, titled 'Package Explorer', shows a project structure for 'SimpleChat'. It includes a 'src/main/java' directory with packages 'm2.gi.SimpleChat' and 'm2.gi.SimpleChat.model'. The 'm2.gi.SimpleChat.model' package contains 'App.java' and 'Message.java'. Other files like 'MessageRepository.java' and 'HelloWorldService.java' are also visible. The right panel shows the code of 'Message.java'. It is a JPA entity class with a 'Long id' and a 'String content' attribute. The class includes a constructor, getters, setters, and a 'toString()' method. The package is 'm2.gi.SimpleChat.model' and it imports 'javax.persistence.Column'.

```
1 package m2.gi.SimpleChat.model;
2
3 import javax.persistence.Column;
4
5
6
7
8
9
10 @Entity
11 public class Message {
12     @Id
13     @SequenceGenerator( name="message_id_message_seq",
14         sequenceName="message_id_message_seq",
15         allocationSize=1)
16     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator="message_id_message_seq")
17     @Column(name="id_message")
18     private Long id;
19
20     @Column
21     private String content;
22
23     public Message() {
24         super();
25     }
26
27     public Message(Long id, String content) {
28         super();
29         this.id = id;
30         this.content = content;
31     }
32
33     public Message(String content)
34     {
35         this.content = content;
36     }
37
38     public Long getId() {
39         return id;
40     }
41
42     public void setId(Long id) {
43         this.id = id;
44     }
45
46     public String getContent() {
47         return content;
48     }
49
50     public void setContent(String content) {
51         this.content = content;
52     }
53
54     @Override
55     public String toString() {
56         return "Message [id=" + id + ", content=" + content + "]";
57     }
58 }
```

Cédric MOSCHETA

C'est quoi ça, une classe vide ? Hé ben non ! Ce joli petit repository va être complété par Spring au build-time. Ainsi, vous aurez toutes les méthodes CRUD de votre modèle qui seront déjà créées, sans rien faire !



```
SimpleChat/pom.xml App.java AppTest.java HelloWorldService.java applica
1 package Repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import model.Message;
6
7 public interface MessageRepository extends JpaRepository<Message, Long> {
8
9 }
10
```

Cédric MOSCHETA

Et voilà, après modification du main, vous avez une application Spring avec injection de dépendances, gestion de fichier .properties, migrations de base de données, et un modèle CRUD opérationnel !

The screenshot displays an IDE environment with the following components:

- Package Explorer:** Shows the project structure for 'SimpleChat', including source files like 'App.java', 'HelloWorldService.java', and 'Message.java', as well as test files and dependencies.
- Code Editor:** Displays the 'App.java' file, which includes a main method and a 'run' method. The 'run' method calls 'saveMessage' and 'printSavedMessages'.
- Database Editor:** Shows a table named 'public.message' with columns 'id_message' (PK) and 'content'. The table contains one record with 'id_message' 1 and 'content' 'Hello I wanna be persisted !'.
- Console:** Displays the output of the application, showing logs from Hibernate and Spring, and the execution of the 'run' method.

```
App.java
33 app.setBannerMode(Banner.Mode.UPT);
34 app.run(args);
35 }
36
37 @Override
38 public void run(String... args) throws Exception {
39     System.out.println(hw.getMessage());
40     saveMessage("Hello I wanna be persisted !");
41     exit(0);
42 }
43
44 private void printSavedMessages()
45 {
46     List<Message> msgs = msgRep.findAll();
47     for (Message message : msgs) {
48         System.out.println(message);
49     }
50 }
51
52 private void saveMessage(String msg)
53 {
54     System.out.println("Persisted messages before save : ");
55     this.printSavedMessages();
56
57     Message msgPersisted = msgRep.save(new Message(msg));
58     System.out.println(msgPersisted + " has been persisted");
59
60     System.out.println("Persisted messages after save : ");
61     this.printSavedMessages();
62 }
63
64 }
65 }
```

Console Output:

```
<terminated> App [Java Application] C:\Program Files\Java\jdk1.8.0_151\bin\javaw.exe (29 nov. 2017 à 14:44:34)
name: default
...
2017-11-29 14:44:36.869 INFO 7688 --- [main] org.hibernate.Version : HHH000412: Hibernate Core {5.0.12.Final}
2017-11-29 14:44:36.870 INFO 7688 --- [main] org.hibernate.cfg.Environment : HHH000206: hibernate.properties not found
2017-11-29 14:44:36.871 INFO 7688 --- [main] org.hibernate.cfg.Environment : HHH000021: Bytecode provider name : javassist
2017-11-29 14:44:36.949 INFO 7688 --- [main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.0.1.Final}
2017-11-29 14:44:37.128 INFO 7688 --- [main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
2017-11-29 14:44:37.291 INFO 7688 --- [main] o.h.e.j.e.i.LobCreatorBuilderImpl : HHH000424: Disabling contextual LOB creation as createClob() method threw error : java.lang.reflec
2017-11-29 14:44:37.291 INFO 7688 --- [main] org.hibernate.type.BasicTypeRegistry : HHH000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryTy
2017-11-29 14:44:37.720 WARN 7688 --- [main] org.hibernate.orm.deprecation : HHH9000014: Found use of deprecated [org.hibernate.id.SequenceHiLoGenerator] sequence-based id ge
2017-11-29 14:44:37.987 INFO 7688 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2017-11-29 14:44:38.330 INFO 7688 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
Bonjour les M2 GI ! :)
Persisted messages before save :
2017-11-29 14:44:38.382 INFO 7688 --- [main] o.h.h.i.QueryTranslatorFactoryInitiator : HHH000397: Using ASTQueryTranslatorFactory
Message [id=1, content=Hello I wanna be persisted !] has been persisted
Persisted messages after save :
Message [id=1, content=Hello I wanna be persisted !]
2017-11-29 14:44:38.519 INFO 7688 --- [Thread-3] s.c.a.AnnotationConfigApplicationContext : Closing org.springframework.context.annotation.AnnotationConfigApplicationContext@50de0926: startu
2017-11-29 14:44:38.519 INFO 7688 --- [Thread-3] o.s.j.e.a.AnnotationMBeanExporter : Unregistering JMX-exposed beans on shutdown
2017-11-29 14:44:38.519 INFO 7688 --- [Thread-3] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
```

Cédric MOSCHETA

Et voilà ! Vous avez donc un projet qui tourne. J'espère que ce guide vous aura permis de mieux appréhender comment réaliser une appli standalone avec Spring et que vous appréciez ses fonctionnalités.