



Spring - Spring BOOT

M2I Formations 2022



Objectifs de la formation

- Créer une application avec des vues, une base de données etc...
- Rappel sur les concepts POO liés à Spring
- Découverte des fondamentaux Spring
- Découverte des fondamentaux Spring Boot
- Utilisation et configuration de Spring Initializr
- Configurations de base avec Maven
- Mise en place des principes de modularité
- Développement d'une application Rest
- Utilisation d'une base de données relationnelles avec JDBC



Les points de la formation

- Pourquoi a-t-on besoin de Spring?
- Le Jargon indispensable lié à la POO
- Les fondamentaux en Spring
- La mise en place de Spring Boot
- Les applications web avec Spring Boot
- Spring MVC
- Développer une API Rest avec Spring
- Gérer une base de données relationnelle avec JDBC
- Générer une application avec Spring Initializr



Pré-requis pour ce cours

- Avoir des connaissances en JAVA SE
- Avoir des connaissances en JAVA EE
- Avoir des connaissances en HTML - CSS - JS
- Avoir des connaissances en POO (Programmation Orientée Objet)
- Avoir des connaissances de base en Maven



A qui est destiné ce cours

- Développeurs full-stack
- Architectes

Spring - Spring Boot

Créer des applications avec le framework le plus populaire en Java



Installation


Environnement Spring



Installation de la JDK 17



Installation de l'IDE IntelliJ IDEA



Version: 2022.2
Build: 222.3345.118
26 juillet 2022

[Notes de publication](#)

[Configurations système](#)

[Instructions d'installation](#)

[Autres versions](#)

[Logiciels tiers](#)

Télécharger IntelliJ IDEA

Windows macOS Linux

Ultimate

Pour le développement web et entreprise

Télécharger .dmg (Intel)

Essai gratuit de 30 jours disponible

Community

Pour le développement JVM et Android

Télécharger .dmg (Intel)

Gratuit, conçu à partir de code open source

Sélectionnez l'outil d'installation adapté pour Intel ou Apple Silicon

	IntelliJ IDEA Ultimate	IntelliJ IDEA Community Edition
Java, Kotlin, Groovy, Scala	✓	✓
Maven, Gradle, sbt	✓	✓
Git, GitHub, SVN, Mercurial, Perforce	✓	✓



1.

**Pourquoi a-t-on
besoin de Spring?**



Synopsis et fil conducteur

Nous allons créer une même application pour différents clients d'abord en Java ce qui va nous permettre de soulever des problématiques.

Nous allons dans un premier temps voir comment avec nos connaissances en objet on peut résoudre ces problématiques, puis en comprendre les limites et ensuite comprendre pourquoi Spring est apparu et comment il répond de manière simple à ces problématiques.

Considérons 3 clients :

- Un client de l'automobile, qui veut gérer ses factures en console, les factures vont de 1 à x.
- Un client d'un supermarché qui veut gérer ses factures via une page web, et ses factures sont préfixées de INV_* pour chaque facture
- Un client qui vend de la peinture, et lui il a besoin d'une page web, mais il génère ses factures avec une douchette (un Bip comme dans les magasins)

L'idée générale va être de montrer comment répondre à ces besoins en essayant de centraliser le code et ça va s'avérer compliqué sans utiliser Spring car le code de base n'est pas flexible !



Présentation - Spring c'est quoi?

- ▶ Framework Java créé par Rod Johnson en 2003
- ▶ Révolution dans le monde Java, aujourd'hui Spring est très utilisé par les entreprises qui souhaitent développer des applications puissantes de manière simple



Présentation - Spring Boot c'est quoi?

- ▶ Spring Boot est une solution de 'convention plutôt que de configuration' destinée à l'infrastructure logicielle Java Spring.
- ▶ Spring Boot a été créé 10 ans après la création de Spring en 2012. C'est une seconde révolution pour le monde Java.
- ▶ Spring Boot vise à réduire la complexité de la configuration de nouveaux projets Spring.



Pourquoi est né Spring Boot?

- ▶ Simplifier le développement des projets du framework Spring
- ▶ Simplifier la gestion des dépendances grâce aux starters qui regroupent plusieurs dépendances
- ▶ Homogénéiser les versions
- ▶ Augmenter la productivité lors de la création d'API
- ▶ Permettre de se concentrer sur l'aspect métier et laisser de côté la complexité liée à la création d'une API



Pourquoi l'utilise-t-on?

- ▶ Créer des applications complexes de manière simple !
- ▶ Pour créer des applications que l'on peut découper en micro services
- ▶ Pour créer des applications REST de manière simple
- ▶ Pour administrer de manière simple des base de données relationnelles
- ▶ Se concentrer sur l'aspect métier et laisser de côté la complexité du code



Différence entre Spring et Spring Boot

- ▶ Spring Boot est basé sur toutes les fonctionnalités par défaut de Spring
- ▶ Core Spring et MVC peuvent gérer des fonctions complètes de n'importe quelle application Java.
- ▶ Spring Boot nous aide à réduire la complexité liée à la configuration du Spring
- ▶ Spring Boot est donc une extension du framework Spring
- ▶ Il adopte une vision Spring et un éco-système de développement plus rapide et plus efficace



Les avantages de Spring Boot

- Dépendances de type 'starter' pour simplifier la construction et la configuration d'une application
- Serveur intégré pour éviter la complexité lors du déploiement d'applications
- Métriques, vérification et configuration externalisée
- Configuration automatique



Le concurrent de Spring





Spring Boot répond à des besoins particuliers

Côté entreprise :

- ▶ Se concentrer sur l'aspect métier

Côté application :

- ▶ Réduire la complexité liée à la mise en place d'architecture, de configuration, de gestion de dépendances, des persistances de données...



Spring ou Spring Boot?

Spring Boot !

Spring Boot contient toutes les fonctionnalités du framework Spring tout en rendant le développement d'application plus simple ! En comparaison avec Spring, vous pouvez créer une application et la démarrer en beaucoup moins de temps car tous les attributs Spring Boot sont auto-configurés par défaut.



2.

**Le Jargon POO et
ses limites,
solutionnées par
Spring**



La programmation par contrat

La programmation par contrat est un paradigme de programmation dans lequel le déroulement des traitements est régi par des règles. Ces règles, appelées des assertions, forment un contrat qui précise les responsabilités entre le client et le fournisseur d'un morceau de code logiciel. C'est une méthode de programmation semi-formelle dont le but principal est de réduire le nombre de bugs dans les programmes.



Les interfaces

Une interface de programmation (aussi nommée API pour Application Programming Interface) est un ensemble standardisé de méthodes, de classes, de fonctions et de constantes.

Une API concède à donner un certain degré d'abstraction au programmeur. C'est-à-dire qu'elle lui cache la difficulté d'accès à un système ou à un logiciel, en présentant un jeu de fonctions standards, dont uniquement les valeurs retournées et les paramètres sont établis.



Injection de dépendances

L'injection de dépendances (dependency injection en anglais) est un mécanisme qui permet d'implémenter le principe de l'inversion de contrôle.

Il consiste à créer dynamiquement (injecter) les dépendances entre les différents objets en s'appuyant sur une description (fichier de configuration ou métadonnées) ou de manière programmatique.



Coupling

Un couplage fort signifie que les classes et les objets dépendent les uns des autres. En général, le couplage fort n'est pas bon car il réduit la flexibilité et la réutilisation du code, tandis que le couplage faible signifie la réduction des dépendances d'une classe qui utilise directement les différentes classes.

Une des solutions pour réduire le couplage de classe est l'implémentation d'interfaces !

Un couplage faible permet de réduire les interdépendances entre les composants d'un système dans le but de réduire le risque que les changements dans un composant nécessitent des changements dans tout autre composant. Il permet donc d'augmenter la flexibilité du code, le rendre plus maintenable et on le verra plus tard, rendre l'ensemble du framework plus stable.



Inversion de contrôle

L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les frameworks qui fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du framework ou de la couche logicielle sous-jacente.

Il existe différentes formes, ou représentation d'IoC, le plus connu étant l'injection de dépendances (dependency injection) qui est un patron de conception permettant, en programmation orientée objet, de découpler les dépendances entre objets.



Réflexivité

La **réflexivité** consiste à découvrir de façon dynamique des informations relatives à une classe ou à un objet. C'est utilisé au niveau de la machine virtuelle Java lors de l'exécution du programme. La machine virtuelle stocke les informations relatives à une classe dans un objet.

La réflexivité n'est que le moyen de connaître toutes les informations concernant une classe donnée pour par exemple créer des instances de classe de façon dynamique grâce à cette notion.



La flexibilité du code

- ▶ Le code de base n'est pas flexible. En tentant de créer notre application de facturation nous nous sommes rendus compte qu'il s'avérait très difficile de répondre aux besoins de chaque client sans impacter de manière considérable notre code. C'est parce que notre code n'est pas flexible qu'un tel framework comme Spring est né !



En résumé Spring va nous permettre de..

- Simplifier la programmation par contrat
- Augmenter l'abstraction du code sans avoir à implémenter des interfaces complexes
- Faciliter l'injection de dépendance
- Réduire le couplage de classe
- Procéder à l'inversion de contrôle
- Améliorer la reflexivité du code
- Augmenter la flexibilité du code

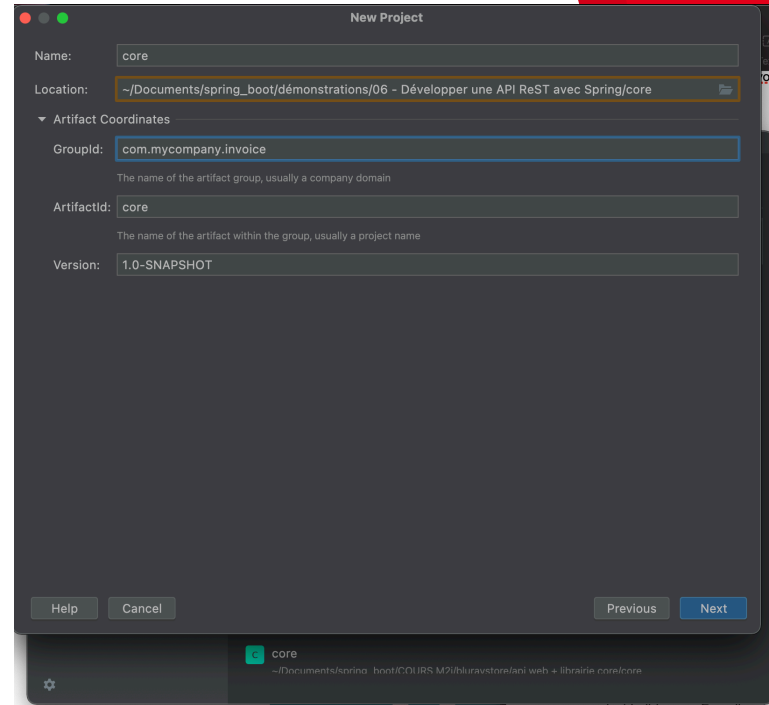
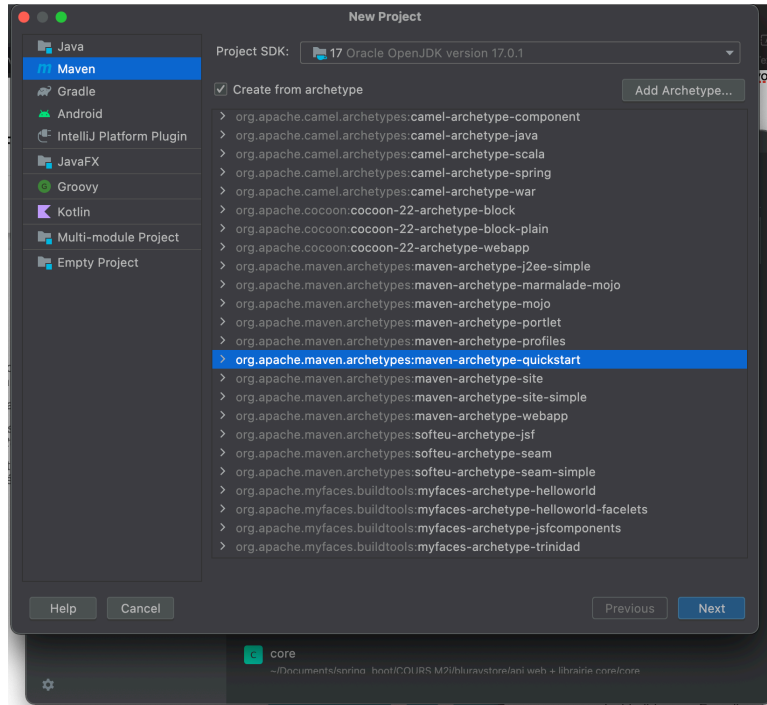


3.

Les fondamentaux du framework Spring



Créer un projet Spring





Créer un projet Spring

- ▶ Créer un fichier application.xml
- ▶ Ajouter la dépendance suivante :

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-context</artifactId>  
  <version>5.3.20</version>  
</dependency>
```

- ▶ Loader les changements maven



Lancer un projet Spring

- ▶ Sur IntelliJ :

CONFIGURATION / EDIT RUN CONFIGURATION / NEW RUN CONFIGURATION / MAVEN /
COMMAND LINE : `spring-boot-run`
- ▶ Depuis le projet sans passer par un IDE : `mvnw spring-boot:run` ou `mvn spring-boot:run` sur
MAC



Conteneurs légers Spring

Bon dans la vraie vie on aurait pas intérêt à demander à l'utilisateur de rentrer manuellement les classes à invoquer, cela n'aurait pas de sens.

Et puis soyons honnête, la configuration d'un projet ne va pas non plus changer tous les 4 matins.

Pour gérer les différentes instanciations des classes, on pourrait plutôt créer un fichier de configuration dans lequel on mettrait les classes à instancier, et ce fichier varierait et serait configuré en fonction du client !

On le lierait, et il instancierait les bonnes classes.

C'est le cœur de SPRING, on appelle cela le conteneur léger de Spring (SPRING LIGHTWEIGHT CONTAINER)



Le context Spring

- Nous avons vu que le framework Spring fournit un conteneur IoC. Ce conteneur permet de définir ce que l'on appelle un contexte d'application (ApplicationContext). Un contexte d'application contient la définition des objets que le conteneur doit créer ainsi que leurs interdépendances.

- Le fichier applicationContext.xml définit les Beans pour le “contexte Web racine”, c’est-à-dire le contexte associé à l’application Web.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- name="service" fait référence à l'attribut repository dans MovieController

    Ici j'injecte dans mon controller le service
    -->
    <bean class="org.mycompany.bluraystore.controller.MovieController">
        <property name="service" ref="service"/>
    </bean>

    <!-- name="repository" fait référence à l'attribut repository dans DefaultMovieService

    Ici j'injecte dans mon service le repository
    -->
    <bean id="service" class="org.mycompany.bluraystore.service.DefaultMovieService">
        <property name="repository" ref="repository"/>
    </bean>

    <!-- Ici donc j'instancie ma classe -->
    <bean id="repository" class="org.mycompany.bluraystore.repository.FileMovieRepository">
        <!-- Application des modifications pour l'exercice 6 :) -->
        <property name="file" value="/Users/samihhabani/Documents/spring_boot/movies.csv"/>
    </bean>

</beans>
```



Ajouter le context Spring à notre projet

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-context</artifactId>  
  <version>5.3.20</version>  
</dependency>
```

Récupération du context

```
public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        MovieController controller = context.getBean(MovieController.class);
        MovieServiceInterface service = context.getBean(MovieServiceInterface.class);
        MovieRepositoryInterface repository = context.getBean(MovieRepositoryInterface.class);

        controller.setService(service);
        service.setRepository(repository);
        controller.addUsingConsole();
    }
}
```



Autowiring

- L'autowiring est une fonctionnalité du framework Spring qui permet l'injection de dépendance entre objets.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd" default-autowire="byType">

    <bean class="org.mycompany.bluraystore.controller.MovieController">
    </bean>

    <bean id="service" class="org.mycompany.bluraystore.service.DefaultMovieService">
    </bean>

    <bean id="repository" class="org.mycompany.bluraystore.repository.FileMovieRepository">
        <property name="file" value="/Users/samihhabbani/Documents/spring_boot/movies.csv"/>
    </bean>

</beans>
```



Autowiring

- ▶ On retrouve deux types d'autowiring :
- ▶ `byName` : l'autowiring se fera par le nom des propriétés et l'id des Beans correspondants
- ▶ `byType` : l'autowiring se fera par type (Classe)



Property-placeholder

- ▶ `<context:property-placeholder location="classpath:application.properties"/>`
- ▶ Permet de charger les fichiers de configuration à l'endroit désigné



Configuration par annotation

- ▶ `<context:annotation-config/>`
- ▶ Permet d'activer les annotations dans des beans qui sont déjà enregistrés dans le contexte de l'application



Détection automatique des beans

- `<context:annotation-config/>`
- `@Component`
- `@Service`
- `@Repository`
- `<context:component-scan base-package='com.company.invoice'/>`



Valorisation des propriétés par annotation

- ▶ @Value
- ▶ Permet l'injection de valeur dans les propriétés de nos beans Java



@Autowired

- ▶ L'annotation @Autowired **permet d'activer l'injection automatique de dépendance**. Contrairement au mode autowiring en XML, il n'est pas possible de définir une stratégie à appliquer. Cette annotation peut être placée sur un constructeur, une méthode ou directement sur un attribut.



Gestion des conflits de dépendances

- Répartition par librairie
- **Répartition par package**
- Répartition par annotation @primary
- Autowiring byName
- @profile ou @conditionnal

‣ Par package :

```
<context:component-scan base-package="com.company.invoice.repository"/>  
<context:component-scan base-package="com.company.invoice.controller"/>  
<context:component-scan base-package="« com.company.invoice.service"/>
```



Regrouper la configuration en package

```
<context:component-scan base-package="com.company.invoice.repository.web"/>  
<context:component-scan base-package="com.company.invoice.controller.prefix"/>  
<context:component-scan base-package="com.company.invoice.service.memory"/>
```



Déplacer la configuration vers la classe d'exécution

- @Configuration
- @ConfigurationScan

```
@Configuration
@ComponentScan(basePackages = { "org.mycompany.bluraystore.controller",
                                "org.mycompany.bluraystore.repository.file",
                                "org.mycompany.bluraystore.service" })
@PropertySource("classpath:application.properties")
public class AppConfig {
}
```




@Configuration & @ConfigurationScan

- ▶ L'annotation @Configuration **permet de déclarer pour Spring un composant qui ne sert qu'à configurer le contexte de l'application**. Normalement ce composant n'est pas destiné à être injecté comme dépendance mais à déclarer des méthodes de fabrique annotées avec @Bean.
- ▶ L'annotation @ConfigurationScan permet de spécifier les packages qui doivent être scannés. Sans argument, Spring scannera le package courant et tous les sous package à l'intérieur



Fichier xml en complément

► @ImportResource

```
@SpringBootApplication
@ImportResource({"classpath*:applicationContext.xml"})
public class Springboot2XmlConfigApplication {

    public static void main(String[] args) {
        ApplicationContext applicationContext = SpringApplication.run(Springboot2XmlConfigApplication.class, args);

        MessageProcessor userService = applicationContext.getBean(MessageProcessor.class);
        userService.processMsg("twitter message sending ");
    }
}
```



4.

La mise en place de Spring Boot



`application.properties`

- ▶ Ce fichier permet de garder x propriétés dans un seule et unique fichier pour exécuter votre application sur différents environnements.
- ▶ En Spring Boot, les propriétés sont gardées dans un fichier `application.properties` sous le classpath



Classpath

- ▶ Classpath est un paramètre passé à une machine virtuelle Java qui définit le chemin d'accès au répertoire où se trouvent les classes et les packages Java afin qu'elle les exécute.



Starters

- ▶ Les starters en Spring sont des ensembles de dépendances que l'on peut implémenter dans notre application.
- ▶ Ils permettent de simplifier la configuration d'un projet
- ▶ Il en existe plus d'une trentaine comme par exemple le web-starter



@SpringBootApplication

- ▶ @SpringBootApplication vient remplacer @ComponentScan, @Configuration et @PropertySource
- ▶ Ajouter la dépendance suivante à votre projet dans votre pom.xml :

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-autoconfigure</artifactId>  
  <version>2.7.0</version>  
</dependency>
```



Spring-boot-starter-parent

- ▶ Ce starter fournit une configuration par défaut pour une application Spring et permet notamment de centraliser les versions entre les dépendances d'un projet Spring
- ▶



Spring-boot-starter

- ▶ Les starters sont des dépendances qui ajoutent de l'autoconfiguration à l'application basée sur **Spring Boot**. L'utilisation d'un **starter** permet d'indiquer que l'on veut ajouter une fonctionnalité à l'application et qu'on laisse au framework le soin de compléter notre configuration.
- ▶



Spring-boot-maven-plugin

- ▶ Ce plugin Maven fournit un support complet Spring Boot dans Apache Maven.
- ▶ Il vous permettra entre autres de packager et d'exécuter les fichiers jar ou war, lancer votre application Spring Boot, générer des builds d'information and démarrer votre application Spring Boot avant de lancer l'intégration des tests.
- ▶



Application Context

```
@SpringBootApplication
public class App
{
    public static void main( String[] args )
    {
        //ApplicationContext context = new AnnotationConfigApplicationContext(App.class);
        ApplicationContext context = SpringApplication.run(App.class);
        MovieController controller = context.getBean(MovieController.class);
        controller.addUsingConsole();
    }
}
```



@PropertySource

- ▶ Cette annotation Spring permet de fournir à l'environnement Spring un fichier de propriétés
- ▶ Cette annotation est utilisée avec l'annotation @Configuration
- ▶ Vous pouvez avoir plusieurs fichiers de propriétés dans un projet Spring

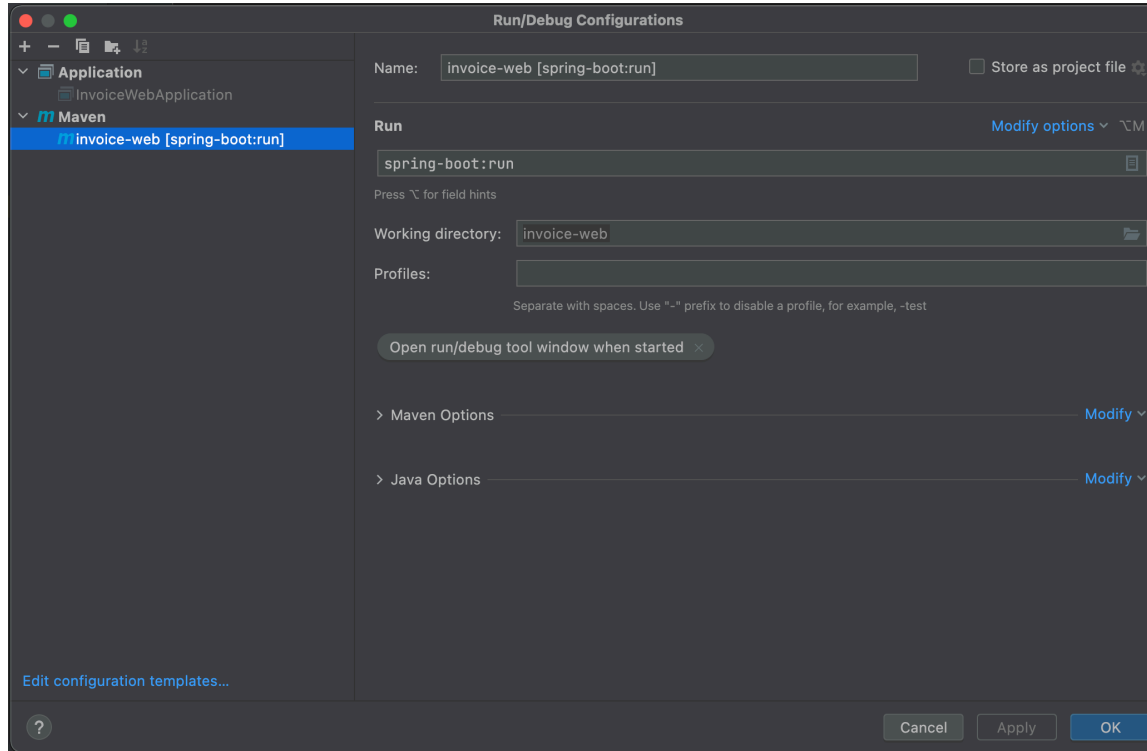


Fat archive

- ▶ Le fat archive est un jar avec ses dépendances.
- ▶ Ce jar ne contient donc pas seulement le programme Java développé mais également toutes ses dépendances incluses à l'intérieur
- ▶ Tout est donc inclus dans ce jar, aucun autre code Java ne sera nécessaire pour son déploiement et exécution.
- ▶



Configuration Maven - run configuration





5.

Les applications web avec Spring Boot



Spring Boot web starter

- ▶ Ce starter vous permettra de créer une application web avec Spring Boot et d'avoir un serveur Tomcat inclus pour lancer votre application
- ▶ Ce starter contient les dépendances web de Spring qui incluent spring-boot-starter-tomcat
- ▶ Spring-boot-starter-web contient spring-boot-starter-jackson

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```


Customiser la configuration Spring via application.properties

docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#appendix.application-properties.server

< Back to index

- 1. Core Properties
- 2. Cache Properties
- 3. Mail Properties
- 4. JSON Properties
- 5. Data Properties
- 6. Transaction Properties
- 7. Data Migration Properties
- 8. Integration Properties
- 9. Web Properties
- 10. Templating Properties
- 11. Server Properties**
- 12. Security Properties
- 13. RSocket Properties
- 14. Actuator Properties
- 15. Devtools Properties
- 16. Testing Properties

11. Server Properties

Name	Description	Default Value
server.address	Network address to which the server should bind.	
server.compression.enabled	Whether response compression is enabled.	false
server.compression.excluded-user-agents	Comma-separated list of user agents for which responses should not be compressed.	
server.compression.mime-types	Comma-separated list of MIME types that should be compressed.	text/html, text/x
server.compression.min-response-size	Minimum "Content-Length" value that is required for compression to be performed.	2KB
server.error.include-binding-errors	When to include "errors" attribute.	never
server.error.include-exception	Include the "exception" attribute.	false
server.error.include-message	When to include "message" attribute.	never
server.error.include-stacktrace	When to include the "trace" attribute.	never
server.error.path	Path of the error controller.	/error
server.error.whitelabel.enabled	Whether to enable the default error page displayed in browsers in case of a server error.	true
server.forward-headers-strategy	Strategy for handling X-Forwarded-* headers.	
server.http2.enabled	Whether to enable HTTP/2 support, if the current environment supports it.	false
server.jetty.accesslog.append	Append to log.	false
server.jetty.accesslog.custom-format	Custom log format, see org.eclipse.jetty.server.CustomRequestLog. If defined, overrides the "format" configuration.	



Modularité

- ▶ Grâce à Spring Boot et Spring, nous allons pouvoir séparer la partie web de la partie backend en créant une application web avec Spring Boot et une librairie tierce, un module coeur à parti, une sorte d'API avec Spring sans Spring Boot.
- ▶



6.

Spring MVC



ThymeLeaf

- ▶ Thymeleaf est un moteur de template, sous licence Apache 2.0, écrit en Java pouvant générer du XML/XHTML/HTML5. Il peut être utilisé dans un environnement web ou non web. Son but principal est d'être utilisé dans un environnement web pour la génération de vue pour les applications web basées sur le modèle MVC.

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-thymeleaf</artifactId>  
</dependency>
```



OGNL

- ▶ Object-Graph Navigation Language est un langage d'expression open-source pour Java qui, tout en utilisant des expressions plus simples que la gamme complète de celles prises en charge par le langage Java, permet d'obtenir et de définir des propriétés et d'exécuter des méthodes de classes Java.

- ▶ Le Spring Framework introduit son propre langage d'expression (nommé *SpEL* pour *Spring Expression Language*). Il est très similaire à l'EL Java EE.
- ▶ Pour la configuration d'un contexte d'application, ce langage permet l'évaluation d'expressions pour désigner le *bean* ou la valeur à injecter. Lorsqu'elle est écrite comme valeur de l'annotation `@Value`, une expression SpEL est délimitée par `#{ }`.

```
@Configuration
public class RemoteServiceConfiguration {

    @Value("${remote.service.url}")
    private URL url;

    @Value("${remote.service.connection.timeout : 1000}")
    private int connectionTimeout;

    public URL getUrl() {
        return url;
    }

    public int getConnectionTimeout() {
        return connectionTimeout;
    }
}
```



Ajout de dépendance vers notre API

- Maintenant que nous avons créer une librairie Spring et une application web Spring Boot, nous pouvons lier les deux en ajoutant une dépendance à notre application web.

```
<dependency>  
  <groupId>com.mycompany.bluraystore</groupId>  
  <artifactId>core</artifactId>  
  <version>1.0-SNAPSHOT</version>  
</dependency>
```



Model And View

- ▶ ModelAndView est un holder qui permet de unir le modèle et la vue dans un framework MVC
- ▶ L'union de ces deux classes rend possible pour un controller de retourner le modèle et la vue dans une seule et unique valeur de retour.
- ▶ La vue est résolue par un objet de type ViewResolver et le modèle est stocké dans une Map



Model

- ▶ Spring nous met à disposition une interface appelée Model pour travailler avec nos données.
- ▶ Cette interface définit un placeholder pour les attributs de notre model et est conçu pour ajouter des attributs au modèle.
- ▶ Elle est également utilisée pour transférer des données entre la vue et le controller d'application Spring MVC.



@RequestMapping

- ▶ Cette annotation est l'une des annotations les plus importantes puisqu'elle permet de mapper les requêtes HTTP vers les méthodes de nos controllers MVC et REST.
- ▶ Dans les applications Spring MVC, le DispatcherServlet (Le controller Front) est responsable du routing des requêtes HTTP entrantes vers les méthodes de nos controllers.



@GetMapping

- ▶ C'est l'annotation qui permet de mapper des requêtes HTTP en scénario GET vers les méthodes de nos controllers.
- ▶



@PostMapping

- ▶ C'est l'annotation qui permet de mapper des requêtes HTTP en scénario POST vers les méthodes de nos controllers.
- ▶

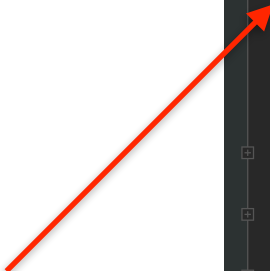
Backing Bean

- Les pages HTML côté client sont mappées avec des classes Java Bean côté backend, c'est que l'on appelle des 'Form Backing Bean Object'.

```
public class InvoiceForm {  
    private String number;  
    // si tu veux mettre un msg personnalisé  
    // @NotBlank(message = "Mon message personnalisé")  
    @NotBlank  
    private String customerInvoice;  
    @Size(min=10,max=13)  
    private String orderNumber;  
  
    public String getNumber() { return number; }  
  
    public void setNumber(String number) { this.number = number; }  
  
    public String getCustomerInvoice() { return customerInvoice; }  
  
    public void setCustomerInvoice(String customerInvoice) { this.customerInvoice = customerInvoice; }  
  
    public String getOrderNumber() { return orderNumber; }  
  
    public void setOrderNumber(String orderNumber) { this.orderNumber = orderNumber; }  
}
```

Annotations de validation

```
public class InvoiceForm {  
  
    private String number;  
    // si tu veux mettre un msg personnalisé  
    // @NotBlank(message = "Mon message personnalisé")  
    @NotBlank  
    private String customerInvoice;  
    @Size(min=10,max=13)  
    private String orderNumber;  
  
    public String getNumber() { return number; }  
  
    public void setNumber(String number) { this.number = number; }  
  
    public String getCustomerInvoice() { return customerInvoice; }  
  
    public void setCustomerInvoice(String customerInvoice) { this.customerInvoice = customerInvoice; }  
  
    public String getOrderNumber() { return orderNumber; }  
  
    public void setOrderNumber(String orderNumber) { this.orderNumber = orderNumber; }  
}
```

A red arrow originates from the left side of the slide and points directly to the `@NotBlank` annotation on the line above `customerInvoice` in the code block.



BindingResult

- ▶ Le BindingResult contient le résultat d'une validation et contient les erreurs qui auraient pu surgir. Il doit être utilisé juste après le model object qui est soumis à validation ou pour lequel Spring ne parvient pas à valider l'objet et renvoie des exceptions.



@Valid

- Cette annotation est clef dans la validation de Bean Java avec Spring. Elle permet de valider les graphs d'objets en appelant une seule fois le validator. Tous les champs qui devraient être vérifiés doivent avoir cette annotation.



7.

Développer une API Rest avec Spring



JAX-RS & Jersey ne sont pas nécessaires

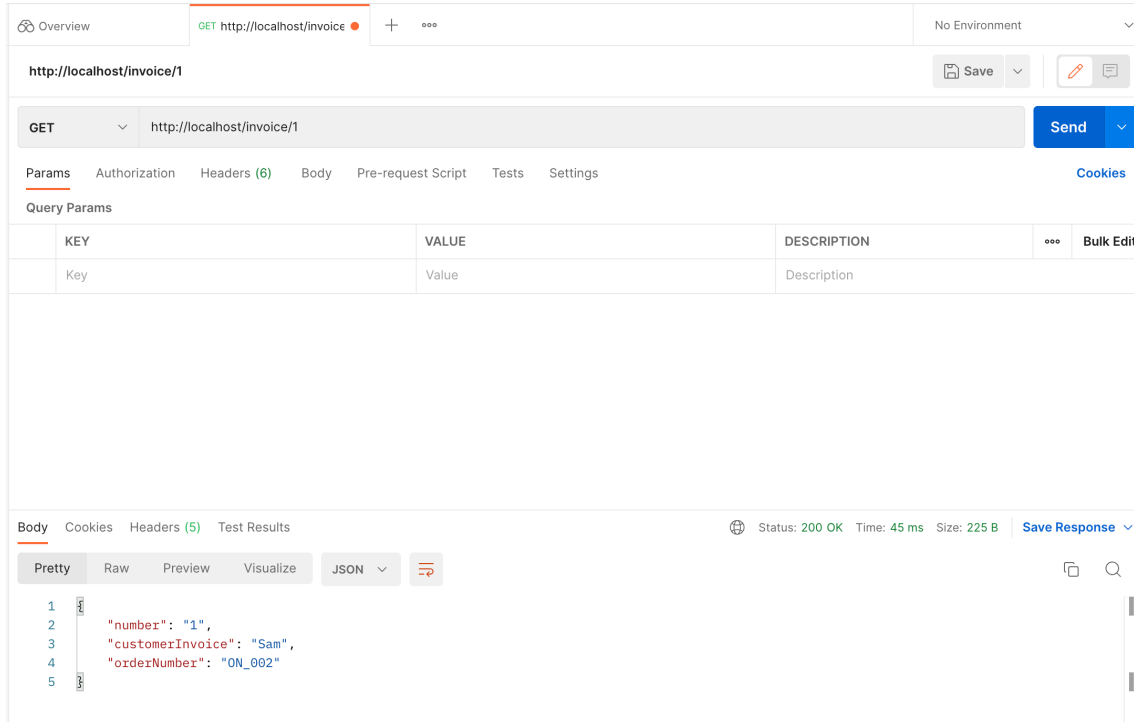
- ▶ En Java, pour créer des web services REST, on utilise ces spécifications.
- ▶ En Spring, on va utiliser la structure spring-mvc donc pas besoin d'installer autre chose



ResponseBody, RequestBody, RestController

- ▶ `@ResponseBody` permet de dire à un controller que l'objet retourné est automatiquement sérialisé en JSON et renvoyé à l'Objet `HttpResponse`.
- ▶ `@RequestBody` permet de récupérer le corps de la requête. On peut ensuite le retourner en String ou le désérialisé en POJO (Plain Old Java Object). Spring a un mécanisme pré-défini pour désérialisé des objets JSON et XML en POJO's, pour nous faciliter la tâche.
- ▶ `@RestController` nous permet de simplifier la création de web services REST. Il combine les annotations `@Controller` et `@ResponseBody` ce qui permet d'éviter d'annoter toutes les méthodes gérant les requêtes entrantes depuis le controller avec l'annotation `@ResponseBody`.

POSTMAN - appelez vos urls REST



The screenshot displays the Postman interface for a REST client. The top bar shows the 'Overview' tab, the request URL 'http://localhost/invoice', and the environment 'No Environment'. The main area shows a GET request to 'http://localhost/invoice/1'. Below the request, the 'Params' tab is active, showing a table for query parameters. The 'Body' tab is also visible, showing a JSON response.

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body

```
1 {
2   "number": "1",
3   "customerInvoice": "Sam",
4   "orderNumber": "ON_002"
5 }
```



Architecture REST

- ▶ @Controller
- ▶ @RequestMapping(« /invoice")
- ▶ @PostMapping("/create")
- ▶

```
fetch('../invoice')  
  .then(res => res.json())  
  .then(res => {...});
```



8.

Base données relationnelles - JDBC



DBeaver - Installation





Création de la BDD avec DBeaver

DBeaver 22.1.1 - INVOICE

SQL • Appliquer (commit) • Retour arrière (rollback) • Auto • localhost • < N/A > •

INVOICE x

Propriétés • Données • ER Diagram

localhost • Bases de données • INVOICE • Tables • INVOICE

Nom de la table: INVOICE

Moteur: InnoDB

Auto-Incrémentation: 4

Encodage: utf8mb3

Collation: utf8_general_ci

Description:

	Nom de la colonne	#	Type de donnée	Non Null	Auto-Incrémentation	Clef	Défaut	Extra	Expression	Commentaire
Contraintes	INVOICE_NUMBER	1	bigint	[v]	[v]	PRI		auto_increment		
Clefs étrangères	CUSTOMER_NAME	2	varchar(50)	[v]	[]					
Références	ORDER_NUMBER	3	varchar(13)	[]	[]					
Triggers										
Indexes										
Partitions										
Statistics										
DDL										
Virtual										

3 items

CET fr_FR

Sauvegarder ... • Revenir • Rafraîchir



Dépendances requises

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-jdbc</artifactId>  
</dependency>
```

```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <scope>runtime</scope>  
</dependency>
```



Connexion & Select

```
@SpringBootApplication
public class InvoiceWebApplication {

    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(InvoiceWebApplication.class, args);

        DataSource ds = context.getBean(DataSource.class);
        Connection conn = null;

        // byte short int long float double char boolean

        try {
            conn = ds.getConnection();
            System.out.println("success");
            ResultSet res = conn.createStatement().executeQuery("SELECT INVOICE_NUMBER, CUSTOMER_NAME FROM INVOICE");
            while(res.next()) {
                System.out.println(res.getLong("INVOICE_NUMBER") + " | " + res.getString("CUSTOMER_NAME"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if(conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```



JDBC template

```
@Autowired  
private JdbcTemplate jdbcTemplate;
```

```
@Override  
public Invoice createInvoice(Invoice invoice) {  
    // méthode générique pour modifier une donnée en bdd
```

```
    KeyHolder kh = new GeneratedKeyHolder();
```

```
    jdbcTemplate.update(connection -> {  
        PreparedStatement ps = connection.prepareStatement("INSERT INTO INVOICE (CUSTOMER NAME, ORDER NUMBER) VALUES  
(?,?)",  
            Statement.RETURN_GENERATED_KEYS);  
        ps.setString(1, invoice.getCustomerInvoice());  
        ps.setString(2, invoice.getOrderNumber());
```

```
        return ps;  
    }, kh);
```

```
    invoice.setNumber(kh.getKey().toString());
```

```
    return invoice;
```

```
}
```



8.

**Générer une
application avec
Spring Initializr**



Créer un projet avec Spring Initializr

start.spring.io

Mettre à jour

Project

☒ Maven Project ☐ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M4) ☐ 2.7.3 (SNAPSHOT) ☒ 2.7.2 ☐ 2.6.11 (SNAPSHOT) ☐ 2.6.10

Project Metadata

Group

com.mycompany.invoice

Artifact

invoice-web

Name

invoice-web

Description

Application de génération de facture

Package name

com.mycompany.invoice.invoice-web

Packaging

☒ Jar ☐ War

Java

☐ 18 ☒ 17 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... ⌘ + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

JDBC API SQL

Database Connectivity API that defines how a client may connect and query a database.

Thymeleaf TEMPLATE ENGINES

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

MySQL Driver SQL

MySQL JDBC and R2DBC driver.

GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

SHARE...

THE END.

**Avec tous nos remerciements et toutes
nos félicitations pour avoir suivi ce
cursus.**

Samih Habbani : s.habbani@coderbase.io