Développement d’application d’entreprise avec le framework Spring

Spring

- Introduction
- Concepts clés
  - Spring Container
  - Inversion of Control (IoC)
  - Dependency Injection (DI)
  - Architecture d’un projet
  - Beans et composants Spring

Spring

- Ressources REST, DTOs et JSON
- Services et entités
- Repositories, entités et JPA
- Gestion de la sécurité avec Spring

Introduction

- Qu’est-ce que Spring ?
  - Framework open-source Java qui permet de construire des applications de façon plus simple et plus rapide. De plus, les applications ainsi créées sont plus facilement maintenable et testable.
  - Adapté à de nombreux types de projets, mais est surtout très utilisé pour des applications d’entreprise (API ou MVC).
  - Fournit des modules distincts pour le support de fonctionnalités particulières :
    - Spring Core
    - Spring Data
    - Spring MVC
    - Spring Security
    - etc,…

Introduction

- Quels outils pour développer avec Spring ?
  - Les principaux IDE possèdent tous des plug-ins destinés à Spring, lorsqu'ils ne l'implémentent pas nativement.
    - Eclipse avec Spring Tools Suite (STS)
    - Netbeans
    - IntelliJ (version Ultimate)
  - Pour du développement web, un serveur d'application est indispensable :
    - Tomcat
    - Glassfish
    - …
    - On peut également utiliser Spring Boot, qui facilite la configuration, et dispose d'un serveur intégré.
  - Au final, l'IDE et le serveur utilisés importent peu.

Concepts clés
Spring Container

- Spring Container
  - Le container Spring est le noyau du framework. C'est lui qui permet de mettre en place les concepts d'inversion de contrôle et d'injection de dépendance, qui sont la base de la progammation avec Spring.
  - Il lie les classes Java avec les metadata qui lui sont fournies (en XML ou grâce à des annotations) et gère la création, la vie et la destruction des objets.
  - C'est lui également qui lie les classes entre elles lorsque c'est nécessaire.

Concepts clés
Spring Container

- Il existe deux types de container qui peuvent être utilisés, et qui sont basés sur deux interfaces distinctes :
  - org.springframework.beans.factory.BeanFactory
  - org.springframework.context.ApplicationContext

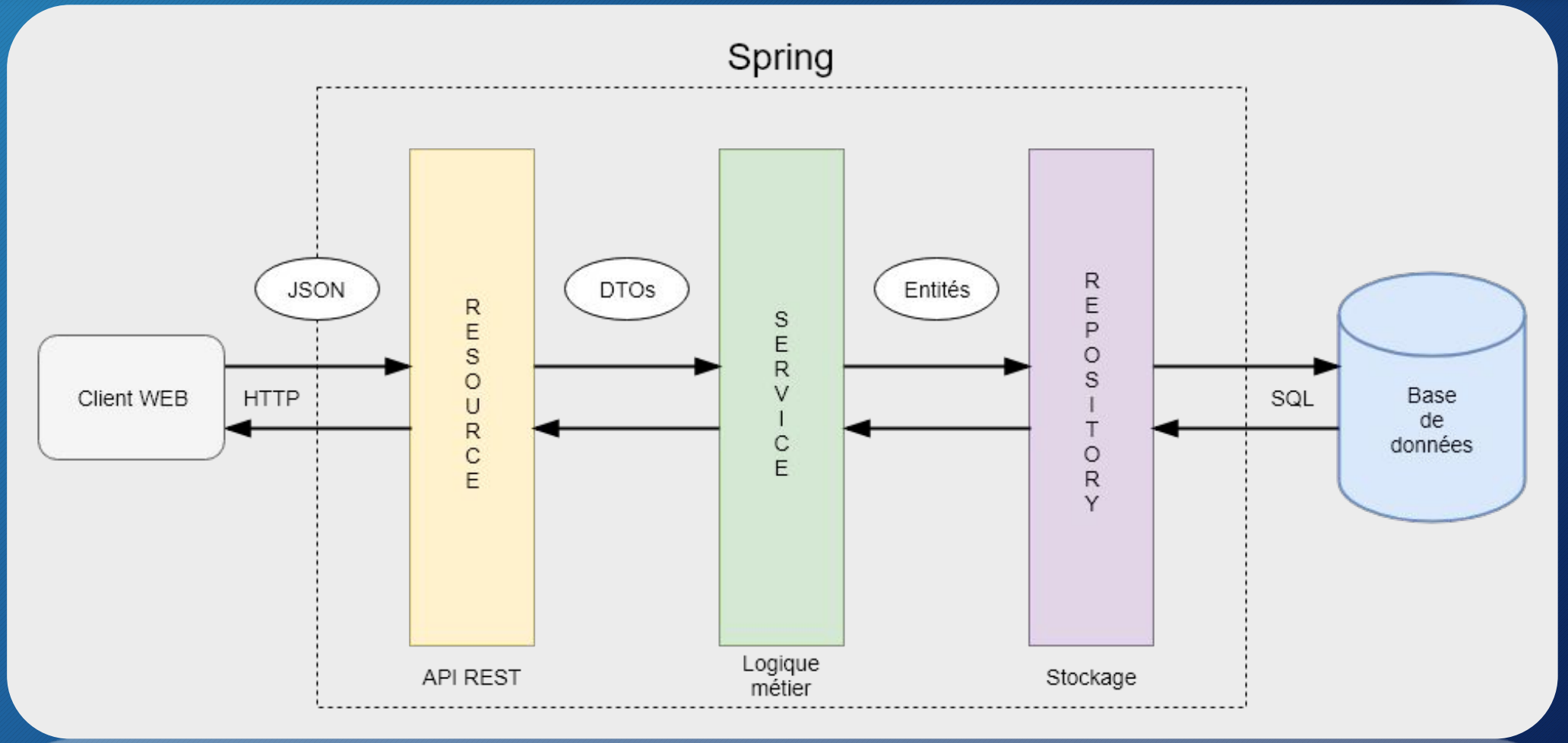
- C'est l'ApplicationContext, plus récente et plus complète, que nous utiliserons. Elle permet notamment, en plus de l'injection de dépendance, de lire des fichiers de propriétés ou encore de publier des évènements applicatifs auprès d'event listeners.

Concepts clés
IoC & DI

- Inversion of Control (IoC)
  - Le container Spring se charge de gérer le cycle de vie de certains de nos objets : création, vie, destruction. Ce n'est plus aux développeurs d'instancier tous les objets nécessaire au fonctionnement de l'application.

- Dependency Injection (DI)
  - L'injection de dépendance se produit lorsque le container Spring instancie une classe : le container va instancier toutes les dépendances de l'objet instancié et les lui fournir. Le code est plus facile à tester et à maintenir.

Concepts clés
Architecture d'un projet



Concepts clés
Beans et composants Spring

- Un Spring Bean est tout objet dont l'instanciation et la gestion revient au Spring Container.

- Ils peuvent être configurés grâce à du XML, ou via des annotations.

Concepts clés
Beans et composants Spring

- Composants :

Par annotation, dans une classe annotée @Component

- Les composants sont simplement des objets gérés par Spring. On peut ainsi créer des classes utilitaires qui pourront être injectées par le conteneur.

Concepts clés
Beans et composants Spring

- REST Controllers :

Par annotation, dans une classe annotée @RestController pour travailler en JSON

- Ils fonctionnent comme les composants, à part qu'on peut utiliser les annotations de Mapping afin de répondre à des requêtes HTTP.
  - @PostMapping("/resources") Pour créer une ressource
  - @GetMapping("/resources") Pour récupérer des ressources
  - @GetMapping("/resources/{id}") Pour récupérer une ressource
  - @PutMapping("/resources/{id}") Pour modifier une ressource
  - @DeleteMapping("/resources/{id}") Pour supprimer une ressource

Concepts clés
Beans et composants Spring

- Services :

Par annotation, dans une classe annotée @Service

- Les services sont les objets qui gèrent la logique métier de l'application et les différents traitements. Ils peuvent gérer la sécurité, la validation, et les intéractions entre les éléments.

Concepts clés
Beans et composants Spring

- Configuration :

Par annotation, dans une classe annotée @Configuration

- Les classes de configuration permettent de modifier le fonctionnement de l'application, comme par exemple la source de données utilisée ou la gestion de la sécurité.

Ressources REST, DTOs et JSON

- Les classes annotées @RestControllers permettent de répondre aux requêtes HTTP envoyées à l'application.
- Les requêtes peuvent être faites en POST, GET, PUT et DELETE, ce qui permet de définir le type de traitement qui doit être fait (ajout, lecture, modification et suppression).
- Chaque requête utilise en plus une URL propre à la ressource.
- Selon le type de méthode, on peut également envoyer des informations complémentaires, comme un body pour les méthodes POST et PUT.

Ressources REST, DTOs et JSON

- Les RestControllers convertissent automatiquement le JSON reçu et renvoyer en objets Java. Ces objets sont appelés « DTO », pour Data Transfer Object. Ce sont des classes Java classiques, avec champs privés, getters et setters. Leur particularité est qu'elles doivent obligatoirement définir un constructeur sans paramètres.

- L'annotation @RequestBody est utilisée devant un DTO en paramètre lorsque le corps de la requête (le body) doit être converti.

- Pour configurer la réponse à renvoyer, on utilise la classe ResponseEntity, qui permet de spécifier le statut HTTP à renvoyer, ainsi que le corps de la réponse s'il y en a un. La conversion du DTO en JSON se fait automatiquement.

Ressources REST, DTOs et JSON

- L’annotation @PathParam permet, devant un paramètre d’une méthode, de récupérer un des éléments variable de la route, par exemple, un identifiant.
  - Dans l’url, on notera cette partie entre accolades : /resources/{id}
  - Dans les paramètres, on notera @PathParam("id") String id

- Le même type de fonctionnement s’applique pour les paramètres de requête, avec @RequestParam("monParam"), mais dans ce cas le paramètre n’est pas inscrit dans l’url, et n’est pas requis. Ce mécanisme est surtout utilisé avec la méthode GET pour les filtres de recherche.

Services et entités

- La couche « Service » de l'application gère la logique métier. Cela comprend la sécurité, les traitements effectués ainsi que les appels à la couche « Repository ».

- Les classes annotées @Service fonctionnent comme les composants classiques et sont gérés par le conteneur Spring. Aucune différence fonctionnelle n'existe, l'annotation permet simplement de spécifier que ce composant fait partie de la couche « Service ».

- Les services suivent toutefois une convention particulière : on déclare toujours une interface, et au moins une implémentation. Pour l'injection de dépendances, c'est l'interface qui est utilisée. Cela permet au développeur de changer d'implémentation sans devoir modifier de code ailleurs dans l'application.

Services et entités

- Habituellement, les services travaillent avec des DTOs reçus en paramètres, qu'ils convertissent en « entités ». Les entités sont la représentation des objets tels qu'ils seront stockés en base de données.

- Les services manipulent les entités, afin d'implémenter les fonctionnalités nécessaires, y compris sauvegarder des modifications et récupérer des données depuis les « Repository ».

- En retour de méthode, ce sont également des DTOs qui sont utilisés.

Services et entités

- Pour faciliter la conversion des DTOs en Entités, des « Mappers » sont créés. Ce sont des classes utilitaires qui gèrent la conversion d'un type vers l'autre. Des outils tels que MapStruct permettent de faciliter leur développement.

Repositories, entités et JPA

- La couche « Repository » permet à l'application de stocker et de récupérer les données en interagissant avec une base de données.
- Les entités vont être configurées par des annotations afin de définir comment la base de données sera construite, et les repositories définiront les possibilités d'interactions avec cette base de données.
- Grâce à Spring Data JPA, on peut simplement déclarer une interface héritant de JpaRepository qui fournira des méthodes de base de communication avec la BD. Pour des cas plus complexes, les méthodes personnalisées ou l'annotation @Query peuvent être utiles.

Repositories, entités et JPA

- @Entity : définit une entité JPA, et donc une table en base de données
- @Table : permet de personnaliser la table (nom, index,…)

- @Id : définit la clé primaire de la table
- @GeneratedValue(strategy = GenerationType.IDENTITY) : auto-incrément

- @Column
  - name = "…« : permet de définir le nom de la colonne en base de données
  - nullable = false : permet de définir une contrainte de non-nullité