Enterprise application development with the Spring framework

Spring

- Introduction
- Concepts
  - Spring Container
  - Inversion of Control (IoC)
  - Dependency Injection (DI)
  - Spring Boot
  - Configuring a Spring application
  - Project architecture
  - Spring Beans and Components

Spring

- REST Resources, DTOs and JSON serialization
- Services and Domain classes
- Spring Data JPA, Entities and Repositories
- Spring Data Redis

Spring

- Spring Security
- Spring Integration with RabbitMQ
- Generate API documentation with Swagger
- Testing a Spring application with JUnit and Mockito

Introduction

- About Spring
  - Spring is an open-source Java framework meant to help developers build modern applications. Developers are able to work faster, better while creating easily maintainable and testable projects.
  - Can be used to build any kind of project : console apps, GUI apps, web apps,... It is used by lots of companies worldwide to implement their back-end logic.
  - Official and community modules can help you implement features and integrate with a lot of tools :

Spring Core, Spring Data, Spring MVC, Spring Security, Spring Integration, Spring Cloud,...
See https://start.spring.io

Introduction

- Tools
  - You can use almost any Java IDE to work with Spring, natively or using plug-ins.
    - Eclipse with Spring Tools Suite (STS)
    - IntelliJ (Ultimate version has first-class support for the framework)
    - Netbeans
  - You need a Java Development Kit, JDK 8 and JDK 11 are fine
  - If you’re doing web development, you must also have an application server :
    - Tomcat
    - Glassfish
    - …
    - Or just use Spring Boot and its embedded Tomcat server !

Concepts
Spring Container

- Spring Container
  - The Spring Container is the core of the framework. It allows developers to use the Inversion of Control pattern using Dependency Injection. These concepts are what has made the framework so popular.
  - The container manages the lifecycle of your objects : instanciation, retrieval, destruction,...

Concepts
Spring Container

- Two distinct containers exist :
  - org.springframework.beans.factory.BeanFactory
  - org.springframework.context.ApplicationContext

- The ApplicationContext interface is the most recent one, and the one used in this training. Its features are Dependency Injection, using properties file for application configuration, application events management, etc...

Concepts
IoC & DI

- Inversion of Control (IoC)
  - The Spring Container manages the lifecycle of objects in your application : creation, destruction. Developers don’t need to call the “new” operator for each and every object they want to use in their application.

- Dependency Injection (DI)
  - Dependency Injection is happening when the container is instantiating a class with dependencies : the Spring Container will actually resolve and instantiate all dependencies of the class. It makes the source code easier to maintain and to test.

Concepts
Spring Boot

- Spring Boot is the easiest way to use Spring. But just because it is easy doesn’t mean it’s wrong ! It’s also the best way.
- Spring Boot is an “opinionated view” of the Spring platform. It means that a basic configuration is in place so that you can start working right away. But if you need to, you can of course configure everything yourself using configuration files or custom classes.

Concepts
Configuring a Spring application

- If you need to use different configurations for diverse use cases, you can use profile-specific configuration files :

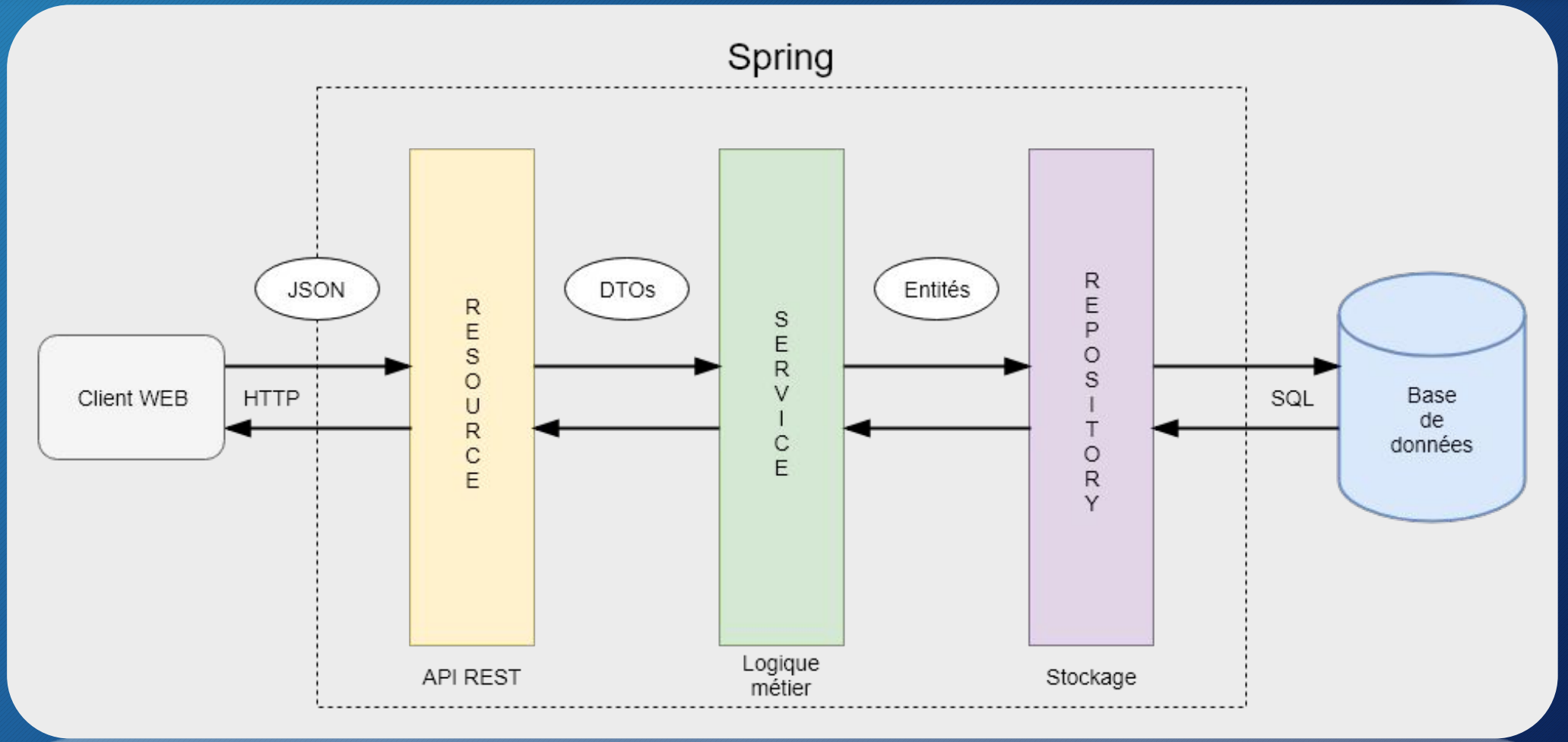
application-prod.yml will only be read if the “prod” profile is set.

You can use as many profiles as you need, all at the same time !

- If project configuration files are not an option (secret configuration for example), you can also use external files or environment variables.

See https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html

Concepts
Project architecture



Concepts
Spring Beans and Components

- Spring Beans are objects managed by the Spring Container.
- They can be configured with XML or with annotations.

Using annotations :

In a class annotated with @Configuration, define a public method annotated with @Bean and returning you custom class.

You can now inject it in another container-managed object !

Concepts
Spring Beans and Components

- Spring Components are just a shortcut to create Spring Beans.

Using annotations :

Add the @Component annotation to your custom class.

You can now inject it in another container-managed object !

Concepts
Spring Beans and Components

- Injecting your Beans

You can inject your beans in three ways :
- Constructor-based injection (highly recommended)
- Field-based injection with @Autowired
- Setter-based injection with @Autowired

REST Resources, DTOs and JSON serialization

- REST Controllers :

@RestController annotated class allow you to automatically parse and serialize JSON requests body.

- Use Mapping annotations to respond to HTTP requests.
  - @PostMapping("/resources") Create a resource
  - @GetMapping("/resources") Retrieve ressources
  - @GetMapping("/resources/{id}") Retrieve a specific resource
  - @PutMapping("/resources/{id}") Update a resource
  - @DeleteMapping("/resources/{id}") Delete a resource

REST Resources, DTOs and JSON serialization

- To get information from the request you can use :
  - @RequestBody gets the value of a parameter from the request body
  - @PathParam gets the value of a route parameter from the request URL
  - @RequestParam gets the value of a query parameter

Services and Entities

- The “Service” layer handles the business logic, including complex validation, security, features implementation and calls to the “Repository” layer.

- @Service annotated classes are simple Spring container-managed components. This annotation is used to express the fact that the component is part of the business layer.
- Services follow a specific convention : they are declared using an interface, and at least one implementation. When doing dependency injection, the interface name is used. It allows developers to switch between implementations at will (dev, test or prod profile for exemple).

Services and Entities

- Usually, services receive DTOs as parameter and response types and convert them to Entities to implement business logic. Entities are domain objects which can be stored in the database.

- Services use entities to implement the features, and retrieve and save the modifications using Repositories.

Services and Entities

- To make conversion between DTOs and Entities easier, developers can create simple Mappers classes, which are usually defined as Components.
- Usually Mappers implement a “toEntity” and a “toDto” method which can be used by the services.

Repositories, Entities and JPA

- The « Repository » layer is used to store and retrieve data. This is usually done using a relational database, but Spring can also help you work with NoSQL databases like Documents (MongoDB), Key-Value stores (Redis), and more !
- Entities are configured using JPA annotations. These annotations define the structure of the tables used. Repositories describe the way to interact with the database.
- Thanks to Spring Data JPA, creating a new interface extending JpaRepository is enough to be able to use basic CRUD features.

Repositories, Entities and JPA

- Spring Data JPA offers a nice way to query the database using method names.
- The @Query annotation can also be used above methods declared in your repositories to specify which query will be sent to the database. Queries can be written in JPQL, the Java persistence version of SQL, or in pure SQL.

Repositories, Entities and JPA

- @Entity : marks a class a an Entity to be stored in the database
- @Table : allows you to customize the table name, indexes,...

- @Id : defines the Primary Key of the table
- @GeneratedValue(strategy = GenerationType.IDENTITY) : uses auto increment feature of the database

- @Column
  - name = "..." : allows to specify the name of the table column
  - nullable = false : defines a NON NULL constraint on the column

Repositories, Entities and JPA

- @OneToOne, @ManyToOne, @OneToMany and @ManyToMany annotations can be used to describe relationships between entities.

Spring Data Redis

- Redis is a key-value store used mainly to store application configuration, user session data or cache data.
- Spring provides a Spring Data module to integrate with Redis easily, using all the Spring Data features : Repositories and RedisHash (Entities)
- Redis is a NoSQL data store and not a relational database. You should know how Redis handles data and queries if you want to use it !

Spring Security

- Spring Security is a dedicated module adding lots of security features to your application :
  - Form, Basic or custom authentication
  - Password Hashing
  - User identity management
  - User role management
  - API route security by-url
  - Service feature security
  - and more !

Spring Boot Security defines a default configuration that you can customize according to your needs.

Spring Security

Maven dependency :

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

Spring Security

Usually, the security configuration is implemented in a @Configuration annotated class.

Useful annotations :

- @EnableWebSecurity : enables the default security configuration.
- @EnableGlobalMethodSecurity(prePostEnabled = true) : allows the use of the @PreAuthorize annotation on your service method. Very useful to implement feature restrictions by user, by “role” or by “authority”.

Spring Security

- The security configuration class should extend WebSecurityConfigurerAdapter. The “configure(HttpSecurity http)” method can be overrided to customize the default configuration.

- Implementing a custom UserDetailsService and returning it from the @Bean annotated userDetailsService method allows you use any data store for you users and roles.

Integration with RabbitMQ

- RabbitMQ is a message-broker implementing the AMQP protocol. It is used to asynchronously send and receive messages between applications.
- Messaging allows the applications to be written in any language with an AMQP client available.
- Applications can be decoupled, which is very useful in distributed environment and when using microservices architecture.

Integration with RabbitMQ

- For simple use cases, the Spring AMQP module can be used.

https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-messaging.html#boot-features-amqp

- For complex use cases, the Spring Cloud Stream module can be used.

http://cloud.spring.io/spring-cloud-static/spring-cloud-stream/2.1.2.RELEASE/multi/multi_spring-cloud-stream.html

Generate API documentation with Swagger

- Swagger tools enables you to document your REST API.
- You can work with a generated documentation with an existing API, or you can write your documentation and generate a server stub
- Swagger is platform agnostic, and can help you generate clients in several languages including javascript, android, java, .NET,...

Generate API documentation with Swagger

- Dependencies :

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-bean-validators</artifactId>
    <version>2.9.2</version>
</dependency>
```

Generate API documentation with Swagger

- Add the @EnableSwagger2 annotation to a @Configuration class

- If you’re using Spring Security, you have to permit access to these URLs :
- /swagger-ressources/**
- /swagger-ui.html
- /v2/api-docs
- /webjars/**

Generate API documentation with Swagger

- Users can now visit the /swagger-ui.html address to read the generated documentation.

- They can also test every request to your application. No need to configure Postman to try it !

Testing a Spring application

- Automated tests are a must in every modern application

- Unit tests : used to test a small portion of your application (a service method, or a domain object). Quick to run, they should be executed before every “commit”.

- Integration tests : used to test integration between components or applications. They are usually run by Continuous Integration tools after every “commit”.

Testing a Spring application
JUnit

- By using the spring-boot-starter-test dependency, you can use JUnit features to test your application :

- Annotate your class with
  @RunWith(SpringRunner.class)
  @SpringBootTest

- Annotate your test methods with @Test

- You can also use @Before and @After to configure your test class.

Testing a Spring application
Mockito

- Mockito is a Mock utility which enables you to unit test components by “mocking” your dependencies.
- Dependency :

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>2.21.0</version>
</dependency>
```

Testing a Spring application
Mockito

- Use the Mockito.mock() method to mock any dependency in your application.

```
Mockito.mock(MyService.class);
```

- Use the Mockito.when() syntax to configure your mock.

```
Mockito.when(myService.getSomething(1)).thenReturn("Hello Mock !");
```