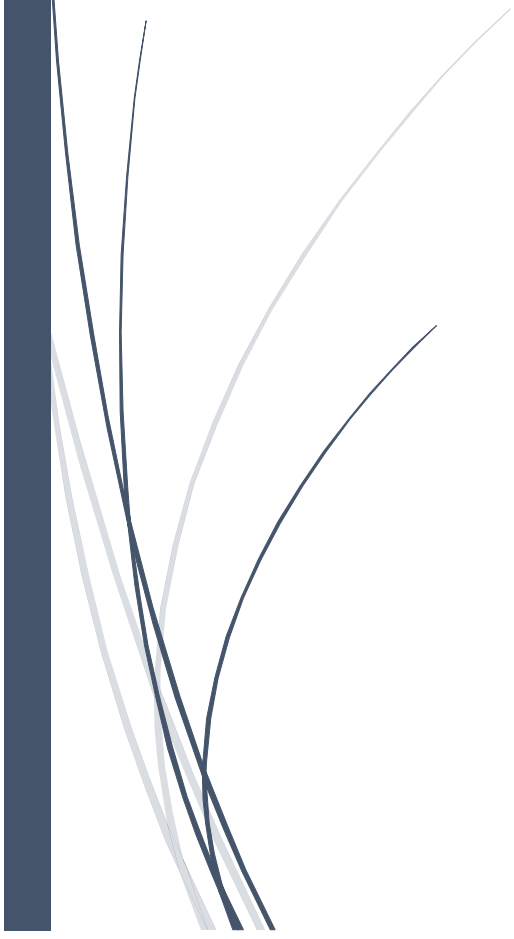
L’api JMS

TP : Implémentation d’une communication asynchrone avec JMS, Apache ActiveMQ, Spring Boot, Spring WebFlux et Thymeleaf



Architecture des composants d’entreprise

Table des matières

I. Objectif du TP

Développer une application réactive avec Spring Boot, Spring WebFlux, JMS et Apache ActiveMQ.

II. Prérequis

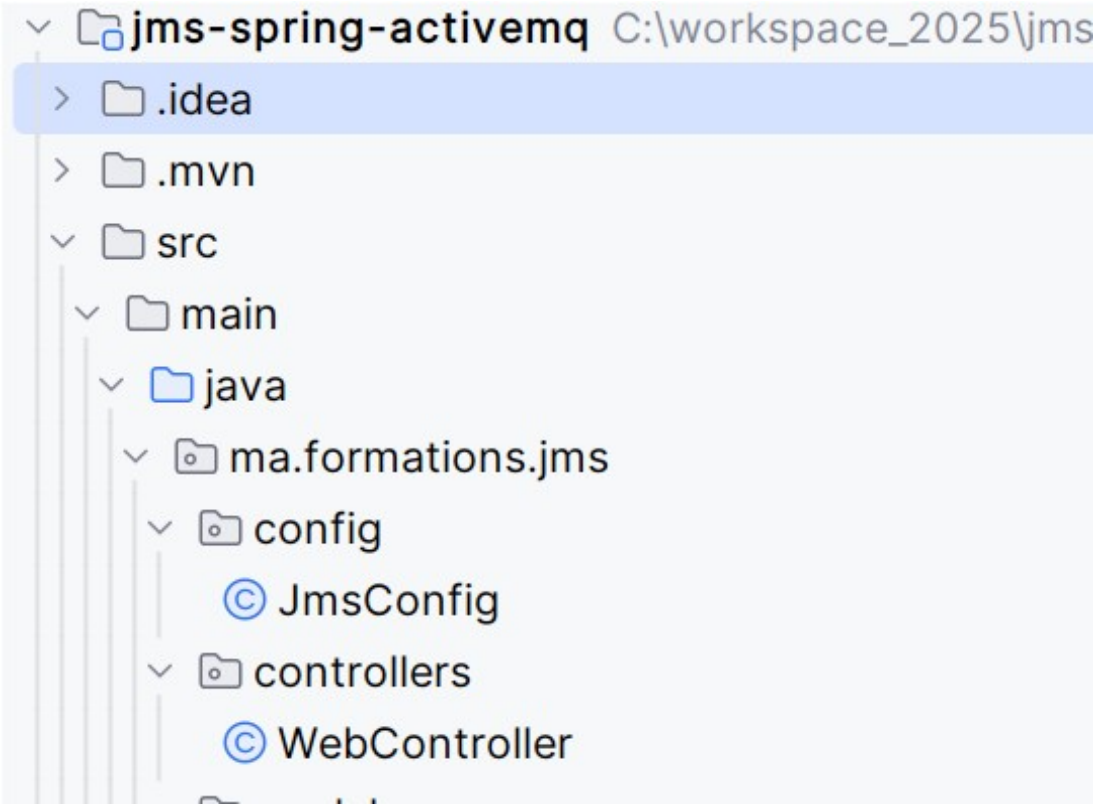
- Réaliser tout d'abord le premier atelier de JMS.
- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.

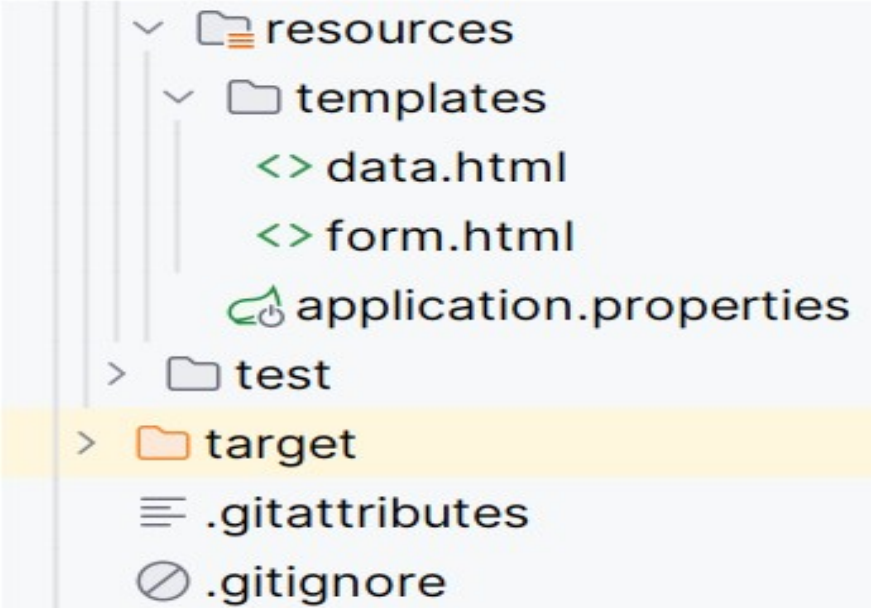
NB : *Ce TP a été réalisé avec IntelliJ IDEA 2023.2.3 (Ultimate Edition).*

I. Développement de l'application

a. L'arborescence du projet Maven

- Avec Intellig, créer un projet Spring Boot. Nommer le par exemple jms-spring-activemq.
- Créer l'arborescence suivante :





b. Le fichier pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.4.0</version>
    <relativePath/>
  </parent>
  <groupId>ma.formations.jms</groupId>
  <artifactId>jms-spring-activemq</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>jms-spring-activemq</name>
  <description>Reactive application using Spring Boot, Spring WebFlux, JMS and ActiveMQ</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-activemq</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>
    <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
      </dependency>
      <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>3.3.7</version>
      </dependency>
      <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
      </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
      </dependency>
      <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
      </dependency>
    </dependencies>
    <build>
      <plugins>
        <plugin>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-maven-plugin</artifactId>
          <configuration>
            <excludes>
              <exclude>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
              </exclude>
            </excludes>
          </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

- Le Framewrok jackson-databind permet de convertir les objets vers JSON et vice-versa.

c. Le fichier application.properties

```
spring.application.name=jms-spring-activemq
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
spring.thymeleaf.reactive.max-chunk-size=8192
```

- On suppose qu'Apache ActiveMQ est démarré dans le port 61616.
- Penseur à modifier éventuellement le compte utilisateur d'ActiveMQ.
- Au niveau de la clé spring.thymeleaf.reactive.max-chunk-size, on configure le nombre de chunk (morceaux) que seront transmis par Thymeleaf (ici c'est 8 KO=1024*8).

d. La classe JmsConfig

```
package ma.formations.jms.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jms.support.converter.MappingJackson2MessageConverter;
import org.springframework.jms.support.converter.MessageType;

@Configuration
public class JmsConfig {
   @Bean
   public MappingJackson2MessageConverter messageConverter() {
      MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
      converter.setTargetType(MessageType.TEXT);
      converter.setTypeIdPropertyName("_type");
      return converter;
   }
}
```

- Le bean MappingJackson2MessageConverter permettra de convertir l'objet vers JSON et vice versa.

e. La classe Employee

```
package ma.formations.jms.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;

@NoArgsConstructor
@AllArgsConstructor
@Data
public class Employee implements Serializable {
   private String name;
   private Double salaire;
   private String fonction;
}
```

f. La classe JmsServiceSender

```
package ma.formations.jms.service;

import ma.formations.jms.model.Employee;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Service;

@Service
public class JmsServiceSender {
   private final JmsTemplate jmsTemplate;
   public JmsServiceSender(JmsTemplate jmsTemplate) {
      this.jmsTemplate = jmsTemplate;
   }
   public void sendMessage(String destination, Employee employee) {
      jmsTemplate.convertAndSend(destination, employee);
   }
}
```

- Spring fourni la classe JmsTemplate qui facilite la création de connexion avec le broker JMS en optimisant le code (création de la factpry, création de la connexion, session, …).
- La méthode convertAndSend convertira l'objet vers JSON en utilisant le bean MappingJackson2MessageConverter avant de d'envoyer le message.

g. La classe JmsServiceReceiver

```
package ma.formations.jms.service;

import ma.formations.jms.model.Employee;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Service;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Sinks;


@Service
public class JmsServiceReceiver {
   private final Sinks.Many<Employee> messageSink;

   public JmsServiceReceiver() {
      this.messageSink = Sinks.many().multicast().onBackpressureBuffer();
   }
   @JmsListener(destination = "test-queue")
   public void onMessage(Employee employee) {
         messageSink.tryEmitNext(employee);
   }
   public Flux<Employee> getMessages() {
      return messageSink.asFlux();
   }
}
```

- @JmsListner permet de créer un Listner pour recevoir les messages envoyés par le Broker JMS.
- Sinks.many() : crée un type de **sink** qui peut émettre des données vers plusieurs abonnés (*subscribers*).
- multicast() : crée un **sink multicast**, où toutes les données émises par le sink sont envoyées **à** tous les abonnés **en même temps.**
- onBackpressureBuffer() : configure la gestion de la pression inverse (backpressure), qui se produit lorsque les abonnés ne consomment pas les données aussi vite qu'elles sont émises. Ici, les données sont mises en **tampon** (buffer) lorsque la consommation est plus lente que l'émission.

h. La classe WebController

```
package ma.formations.jms.controllers;

import lombok.AllArgsConstructor;
import ma.formations.jms.model.Employee;
import ma.formations.jms.service.JmsServiceReceiver;
import ma.formations.jms.service.JmsServiceSender;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.thymeleaf.spring6.context.webflux.IReactiveDataDriverContextVariable;
import org.thymeleaf.spring6.context.webflux.ReactiveDataDriverContextVariable;
@Controller
@RequestMapping("/")
@AllArgsConstructor
public class WebController {

   private JmsServiceSender sender;
   private JmsServiceReceiver receiver;
   @GetMapping
   public String welcome(Model m) {
      m.addAttribute("emp",new Employee());
      return "form";
   }

   @PostMapping(value="/send")
   public String sendMessage(Employee emp, Model m)  {
         sender.sendMessage("test-queue",emp);
         m.addAttribute("confirmation", "Message envoyé avec succès !");
      m.addAttribute("emp",new Employee());
         return "form";
   }

   @GetMapping("/messages")
   public String test(Model m) {
      IReactiveDataDriverContextVariable reactiveDataDrivenMode =
            new ReactiveDataDriverContextVariable(receiver.getMessages(), 1);
      m.addAttribute("employees", reactiveDataDrivenMode);
      return "data";
   }
}
```

i. La page form.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Test</title>
   <link rel="stylesheet" type="text/css" href="webjars/bootstrap/3.3.7/css/bootstrap.min.css"/>
</head>
<body>
<h1>Envoyer un message via JMS</h1>

<form action="#" th:action="@{/send}" th:object="${emp}" method="post">

   <label for="name">Name</label>
   <input type="text" th:field="${name}" id="name" placeholder="Name">

   <label for="fonction">Fonction</label>
   <input type="text" th:field="${fonction}" id="fonction" placeholder="Fonction">

   <label for="salaire">Salaire</label>
   <input type="text" th:field="${salaire}" id="salaire" placeholder="Salaire">

   <input type="submit" value="Envoyer" class="btn btn-primary">
</form>

<p th:if="${confirmation}" th:text="${confirmation}"></p>

</body>
</html>
```

j. La page data.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Test</title>
   <link rel="stylesheet" type="text/css" href="webjars/bootstrap/3.3.7/css/bootstrap.min.css"/>
</head>
<body>
<h1>Envoyer un message via JMS</h1>

<form action="#" th:action="@{/send}" th:object="${emp}" method="post">

   <label for="name">Name</label>
   <input type="text" th:field="*{name}" id="name" placeholder="Name">

   <label for="fonction">Fonction</label>
   <input type="text" th:field="*{fonction}" id="fonction" placeholder="Fonction">

   <label for="salaire">Salaire</label>
   <input type="text" th:field="*{salaire}" id="salaire" placeholder="Salaire">

   <input type="submit" value="Envoyer" class="btn btn-primary">
</form>

<p th:if="${confirmation}" th:text="${confirmation}"></p>

</body>
</html>
```

k. La classe MainApplication

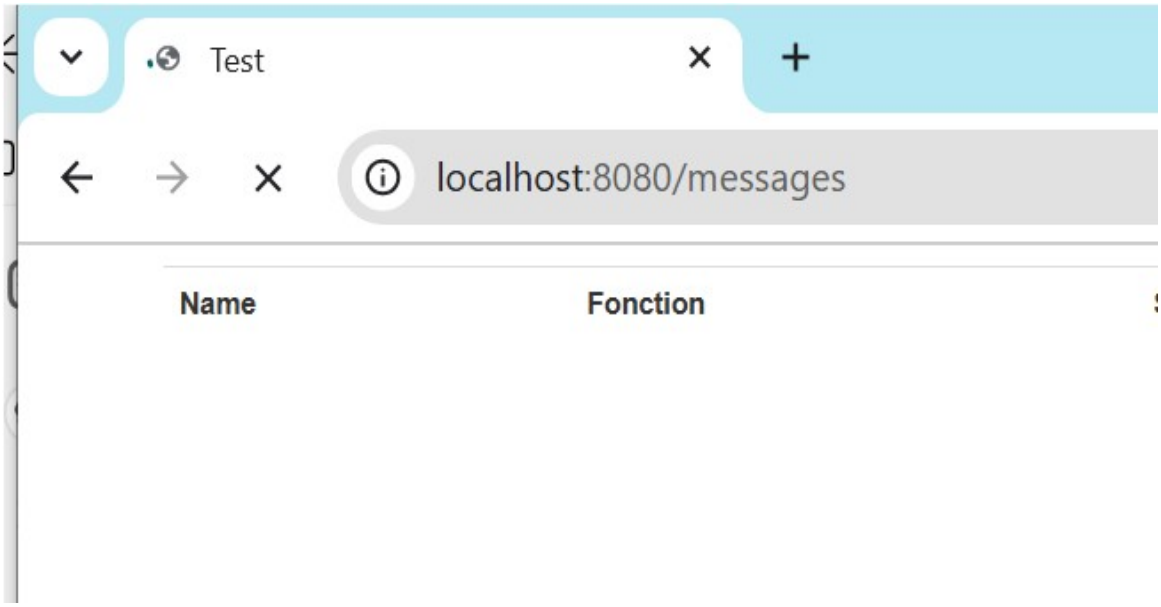
```
package ma.formations.jms;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MainApplication {
    public static void main(String[] args) {
        SpringApplication.run(MainApplication.class, args);
    }
}
```

I. Tester l'application

- Démarrer ActiveMQ ([ACTIVEMQ_PATH]/bin/activemq start).
- Lancer la méthode main de la classe MainApplication.
- Lancer d'URL : http://localhost:8080/messages



- Lancer ensuite l'url : http://localhost:8080.
- Ajouter des nouveaux employés et cliquer sur le bouton Envoyer.
- Vérifier que les employés sont affichés en mode Streaming automatiquement au niveau de la première fenêtre :