Labo Spring 1
- *Première application Web* -

Objectif

- Créer une première application Web en utilisant Spring Boot

Étape 1 : IntelliJ

Utilisez IntelliJ IDEA **Ultimate** (<> Community).

Si nécessaire, accès à la licence Ultimate via l'adresse mail Henallux (compte JetBrains).

Étape 2 : Création d'un nouveau projet

Pour rappel, ce qui identifie une classe en Java c'est le nom de la classe préfixé de son package. L'identifiant d'une classe ne doit contenir aucun espace.

- Veillez donc toujours à ce qu'un projet java (et donc ses packages) soit placé dans un répertoire dont le path (l'adresse complète) ne comprend aucun espace ni caractères spéciaux !

- Ne jamais utiliser en programmation Java de répertoire dont le nom contient un espace.

Spring Application Framework

Spring est un framework Open Source qui simplifie le développement d'application Java pour les entreprises.

Il supporte l'injection de dépendance et l'inversion de contrôle (voir plus loin).

Spring Boot

Spring Boot est une extension du framework Spring qui simplifie la configuration initiale d'une application Spring en proposant des configurations minimales.

pom.xml

Le fichier pom.xml est le fichier de configuration pour les applications Spring. Il contient entre autres les définitions des dépendances qui seront utilisées dans le projet.

Un squelette de projet a été déposé sur GitLab à l'adresse suivante :

https://gitlab.com/bastien.bodart.henallux/ig308-public

Dans IntelliJ, clonez ce projet à l'aide de son adresse publique

https://gitlab.com/bastien.bodart.henallux/ig308-public.git

Validez la confiance à ce projet si la question est posée par l'IDE .

Fichier pom.xml (extrait) :

```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>2.5.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.spring.henallux</groupId>
    <artifactId>firstSpringProject</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>firstSpringProject</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>17</java.version>
    </properties>
```

Étape 3 – Serveur Tomcat

Tomcat est le serveur d'applications Web sur lequel les servlets et pages jsp seront déployées. Le serveur Tomcat sera installé en local sur votre machine.

Pour ce faire, une dépendance a été ajoutée dans le pom.xml :

```xml
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Étape 4 : JSTL

JSTL

JavaServer Pages Standard Tag Library est une librairie de fonctions permettant entre autres de fournir du contenu dynamique dans les pages JSP.

Pour permettre l'utilisation de JSTL, une dépendance a été ajoutée dans le pom.xml :

```xml
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

Étape 5 : Page welcome.jsp

Page JSP

Les pages jsp représentent la partie Vue de l'application Web. JavaServer Pages est une technologie permettant de générer dynamiquement des pages Web. Ces pages contiennent des expressions statiques (html) et des éléments JSP permettant de construire le contenu dynamiquement.

Les pages web (couche vue) sont regroupées dans le répertoire webapp. Les pages jsp seront placées dans le répertoire jsp se trouvant dans le sous-répertoire WEB-INF de webapp.

Soit la structure du projet :

Adaptez le code de la page web *welcome.jsp* qui se trouve dans le répertoire jsp. Cette page doit contenir un message de bienvenue sur le site.

Étape 6 – ViewResolver

Inversion de contrôle et injection de dépendances

Le framework Spring permet l'inversion de contrôle (ex : le programmeur ne crée plus certains objets, c'est Spring qui gère le cycle de vie de ces objets) par injection de dépendance (création automatique par Spring d'objets appelés beans). Les informations nécessaires à Spring pour créer ces beans sont précisées soit dans des fichiers XML de configuration soit via annotations (ex : *@Bean*).

Classe de configuration

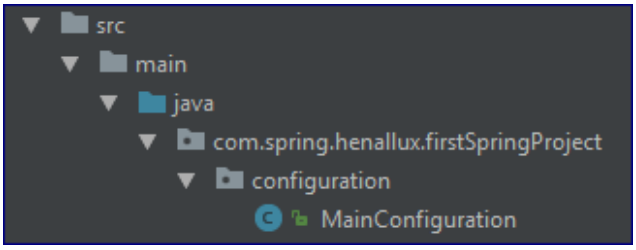
Les classes annotées *@Configuration* contiennent des méthodes de définition de beans. Ces méthodes sont annotées *@Bean*. Une méthode annotée *@Bean* est appelée par Spring pour créer et initialiser un objet (un bean). Le bean généré sera du type de retour de la méthode et, sauf mention contraire, portera le nom de la méthode.

ViewResolver

Le rôle du *ViewResolver* est de localiser les pages JSP à partir du répertoire où elles sont stockées ainsi que de leur suffixe jsp.

L'objet *ViewResolver* sera créé par injection de dépendance. Dans notre application, il va rajouter devant le nom de chaque page jsp retournée par un controller (voir ci-après) le préfixe "/WEB-INF/jsp/" (le répertoire contenant les pages jsp) et y ajouter le suffixe ".jsp".

Les classes de configuration sont placées dans le package *configuration*.



N.B. Le package *configuration* contient une classe *TilesConfiguration* dont le code est pour l'instant commenté. Cette classe sera utilisée dans le labo 2. Ne supprimez pas cette classe.

Adaptez le code de la classe *MainConfiguration* qui doit contenir un bean de type *ViewResolver*.

La classe doit être annotée *@Configuration* et implémenter *WebMvcConfigurer*.

La méthode créant un objet de type *ViewResolver* doit être annotée *@Bean* (injection de dépendance). Lors du déploiement de l'application, un objet bean de type *ViewResolver* portant le nom de la méthode sera créé automatiquement (il sera donc nommé *viewResolver*).

```java
package com.spring.henallux.firstSpringProject.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
public class MainConfiguration implements WebMvcConfigurer {

    @Bean
    public ViewResolver viewResolver () {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/jsp/");
        resolver.setSuffix(".jsp");
        return resolver;
    }
}
```

Étape 7 – Fichier application.yml

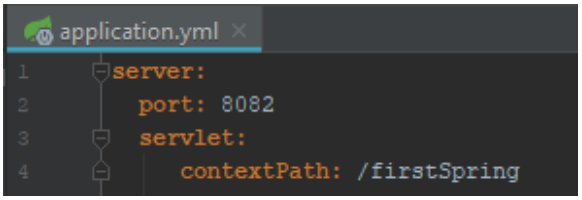
application.yml

Le fichier *application.yml* est un fichier de configuration de propriétés externes. Celui-ci permet de préciser les paramètres pour le déploiement de l'application (port d'écoute, url de la racine, ressource de type base de données…)

Adaptez le contenu du fichier *application.yml* qui se trouve dans *src/main/resources*.

Port : port d'écoute

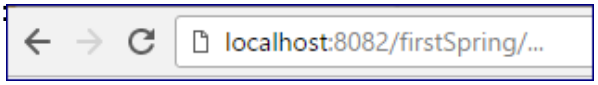
ContextPath : path/url de la racine du projet



Attention : Ce type de fichier est très sensible aux indentations et espaces : exactement 2 espaces avant chaque niveau d'indentation et aucun espace à la fin d'une ligne, ni de ligne blanche en fin de fichier !

Les informations de ce fichier seront utilisées lors de l'appel des pages dans un navigateur (*quand l'application sera déployée - voir plus loin*).

Si l'application est déployée localement :



Étape 8 – Welcome Controler

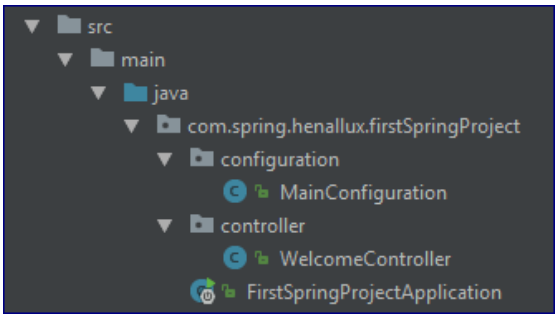
Controller

Les classes de type *controller* (annotées *@Controller*) contiennent les méthodes qui seront appelées par Spring en réponse aux requêtes http (*GET* ou *POST*). A une URL doit donc correspondre une méthode. Cette correspondance est établie via le mécanisme du *Request Mapping* (via *@RequestMapping* dans la déclaration des classes de type controller et dans la déclaration des méthodes de ces classes).

L'URL est constituée de /décomposée en :
l'URL de la racine du projet (cf. fichier *application.yml*)
+ valeur du *RequestMapping* de la déclaration de la classe controller
+ valeur du *RequestMapping* de la déclaration de la méthode.

Les annotations *GetMapping* et *PostMapping* peuvent être également utilisées.

Les classes controllers seront placées dans le package controller.



Créez le package *controller* au même niveau que le package *configuration*.

Créez la classe *WelcomeController* qui doit être annotée *@Controller*.

Ce controller contient une méthode de mapping GET qui retourne la page *welcome.jsp*. Précisez le path/url correspondant à l'appel de ce controller via l'annotation.

La valeur du *Mapping* sera utilisée dans l'URL à contacter.

```java
package com.spring.henallux.firstSpringProject.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/hello")
public class WelcomeController {

    @GetMapping
    public String home(Model model) {
        return "welcome";
    }
}
```

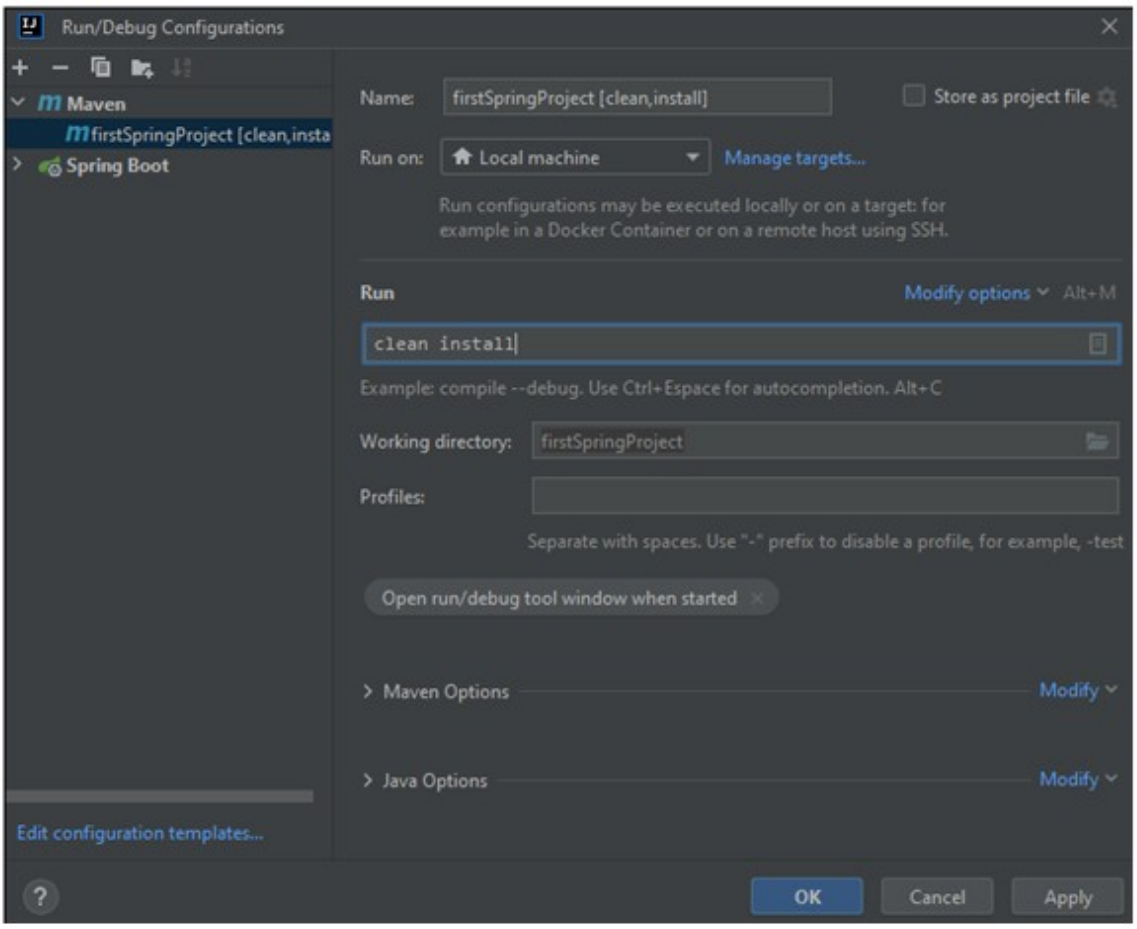
Etape 9 – Maven Build

Maven

Maven est un outil facilitant entre autres la construction (*build*) d'une application en utilisant les dépendances et autres informations de configuration spécifiées dans le fichier *pom.xml*.

Nous allons créer un script constitué des étapes *clean* et *install*. Il suffira ensuite de lancer ce script chaque fois que des modifications seront apportées (par exemple, lorsqu'une nouvelle dépendance aura été ajoutée dans le pom.xml).

Créez une "*run configuration*" pour Maven. Celle-ci permettra de facilement faire un clean du projet suivi d'une installation.



Etape 10 – Déploiement de l'application Web

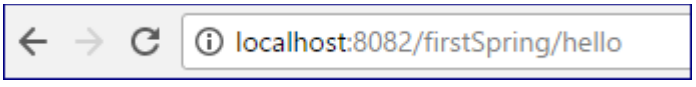
Build (clean + install) de l'application *(si nécessaire)*

Lancez le build Maven de votre application

Déploiement de l'application sur le serveur Tomcat

FirstSpringProjectApplication Run

Test de l'application web

Dans un navigateur : 

En cas de problème…

Si vous avez une erreur de type site inaccessible, vérifiez que l'url que vous introduisez dans le navigateur est bien formée et corresponde bien à la combinaison du *port* (*8082*) + du *contextPath* (*/firstSpring*) tels que définis dans le fichier *application.yml,* suivie de l'url définie dans le *@RequestMapping* de la classe *WelcomeController (/hello*). Vérifiez que les valeurs du *port*, du *contextPath* et du *@RequestMapping* sont correctement définies dans votre code.

Si vous avez une erreur de type *Whitelabel Error Page … (type=Not Found, satus=404)*, vérifiez la structure de vos répertoires et vérifiez enfin la syntaxe de la méthode *viewResolver* de la classe *MainConfiguration* (préfixe ="/WEB-INF/jsp/" et suffixe = ".jsp").

Pour aller plus loin

Etape 11 – Transfert d'une donnée du controller vers une page web

Objectif

- Transférer une donnée du controller vers la page jsp afin de l'afficher

Model Dictionary

Il est possible de transférer des données d'un controller vers une page jsp à travers un dictionnaire composé de paires clé-valeur. Ce dictionnaire est un objet de la classe *Model* qui peut être reçu en argument des méthodes dans les controllers ; il suffit d'ajouter une paire clé-valeur dans le dictionnaire.

Ce dictionnaire est directement accessible au sein ses pages jsp. Celles-ci peuvent récupérer les valeurs des entrées du dictionnaire à partir des clés via la syntaxe ${clé}

Imaginons que le *WelcomeController* veuille afficher un nom en particulier.

Pour ce faire, ajoutez un attribut au dictionnaire de type *Model*. Ajoutez-y une nouvelle entrée (paire clé-valeur) dont la clé est "name" et la valeur est une String correspondant au nom.

Récupérez la valeur de cet attribut du *Model* et affichez-la dans la page jsp, via l'instruction *${name}*.

```java
package com.spring.henallux.firstSpringProject.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/hello")
public class WelcomeController {

    @GetMapping
    public String home(Model model) {
        model.addAttribute( attributeName: "name",  attributeValue: "toto");
        return "welcome";
    }
}
```

```jsp
<%@ page pageEncoding="UTF-8"
         contentType="text/html; charset=UTF-8"%>
<html>
<head>
    <title> Welcome </title>
</head>
<body>
<div>
 Welcome ${name} !
</div>

</body>
</html>
```