



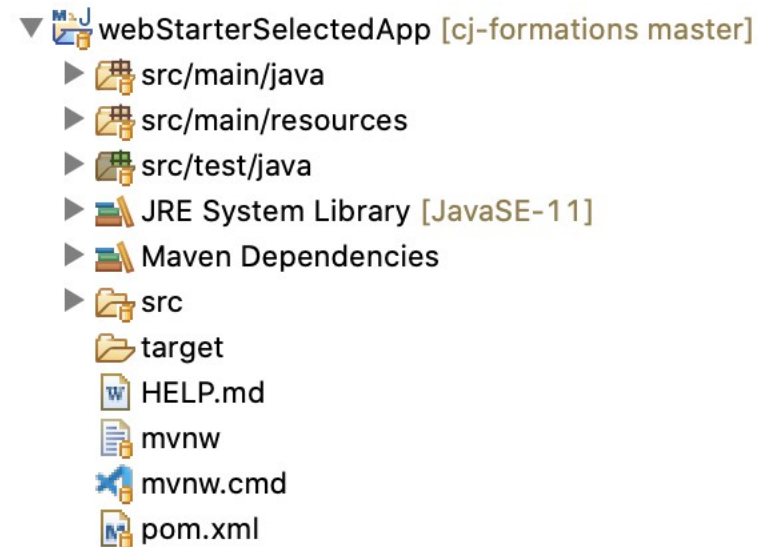
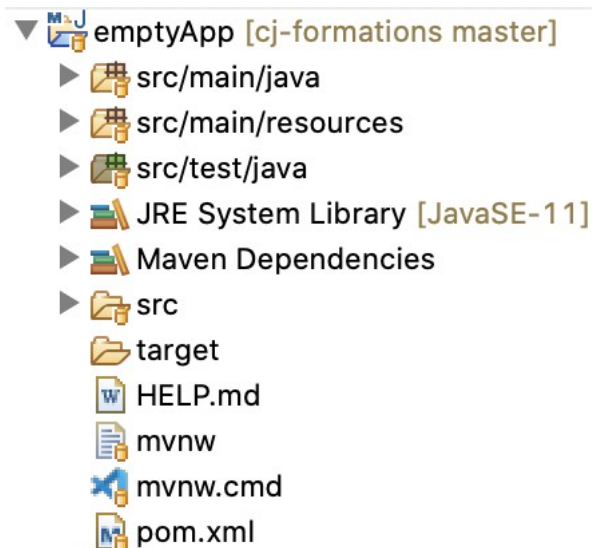
# **Spring Boot Structure d'un projet**

# La composition d'un projet Springboot

Nous avons généré deux projets différents :

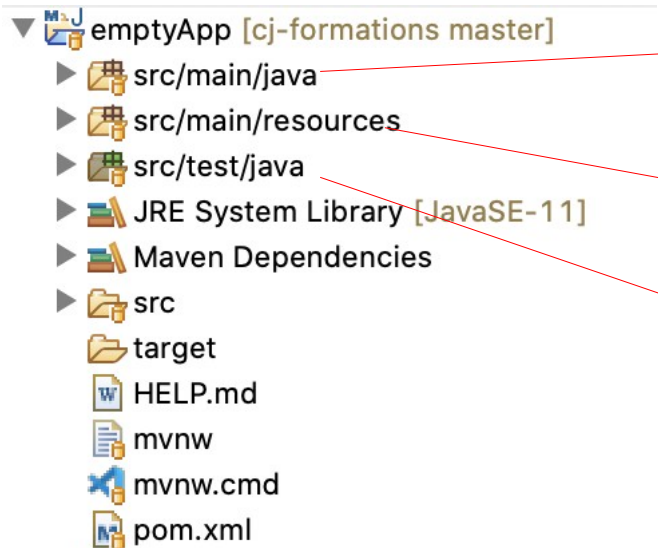
L'un sans  
spécifier des  
dépendances

L'autre en spécifiant  
une dépendance  
spring-boot-starter-web

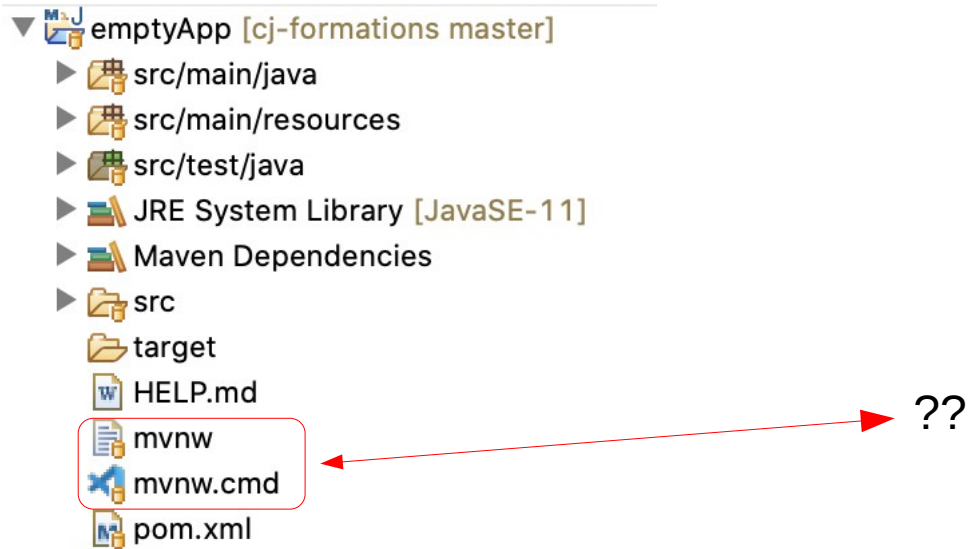


Les structures sont les mêmes

# Une strucutre maven standard

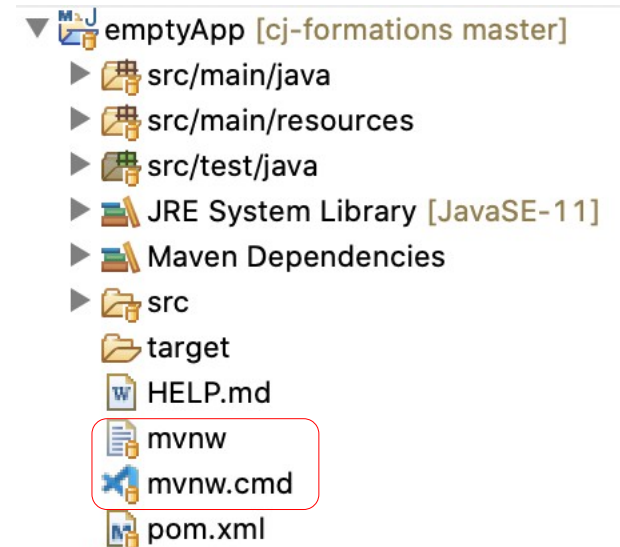
	Répertoire	Description
	src/main/java	Le code source Java de l'application que vous écrivez
	src/main/resources	Fichiers de properties , de configuration de l'application
	src/test/java	Les tests unitaires relatifs à votre code source

# Il existe des fichiers supplémentaires



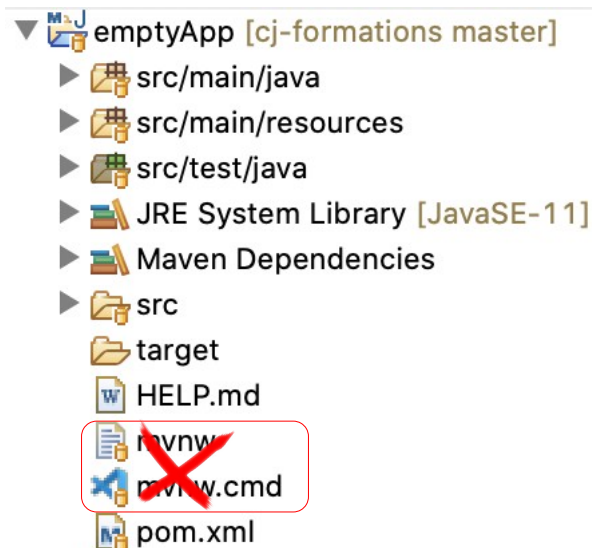
# Fichiers wrapper Maven

- Mvnw permet d'exécuter un projet maven
  - Sans avoir maven installé
  - Sans avoir besoin de vérifier les n° de version
  - Le téléchargement est automatisé pour une exécution sans installation ni configuration
- Deux fichiers sont fournis
  - mvnw.cmd pour Microsoft Windows → `> mvnw clean compile test`
  - mvnw.sh pour linux/Mac → `$ ./mvnw clean compile test`



# mvnw et mvnw.cmd non indispensables

- Si maven est déjà installé sur votre ordinateur , il n'est pas nécessaire de conserver ces deux fichiers wrappers de maven



```
> mvn clean compile test
```

```
$ ./mvn clean compile test
```

# pom.xml

- Ce fichier contient en premier lieu des informations que vous avez entré lors de la génération sur le site <https://start.spring.io/> (spring initializr)

```
<groupId>com.demo</groupId>
<artifactId>webStarterSelectedApp</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

- Il contient aussi une collection de dépendances, pour lesquelles on peut remarquer qu'elles s'appellent "starter".

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

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

# Starter

- Un starter est une collection de librairies
- Par exemple spring-boot-starter-web contient :  
spring-web, spring-webmvc, hibernate-validator,  
tomcat, json
- Cela évite au développeur de devoir gérer une  
liste de dépendances spécifiques et de devoir  
s'assurer de leur inter compatibilité



# Maven plugins

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

- En fin de pom.xml nous trouvons une section pour les plugin maven de springboot.
- Ici spring-boot-maven-plugin sert à :
  - à créer un fichier .jar `$ ./mvnw package`
  - à créer un fichier d'archive war
  - à exécuter l'application `$ ./mvnw spring-boot:run`



# Si maven est déjà sur votre machine

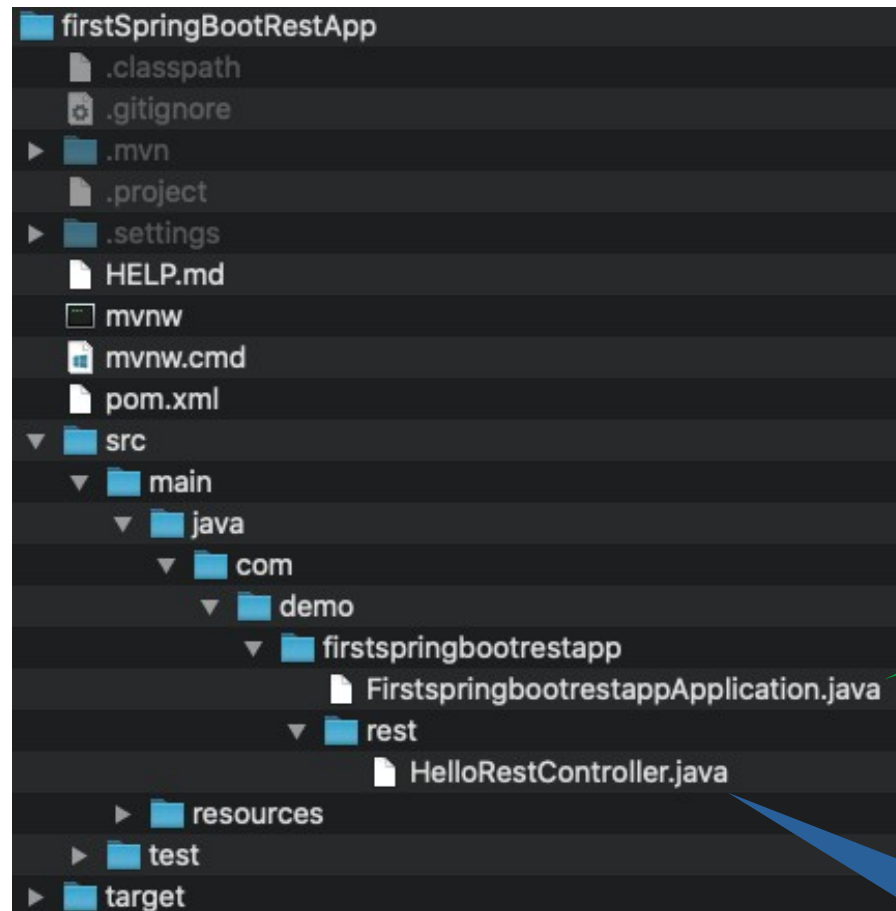
```
mvn package
```

et

```
mvn spring-boot:run
```

feront le même job à partir de votre environnement et de votre configuration locale

# La partie java du projet spring boot



La classe principale de l'application Spring Boot

Le contrôleur que nous avons créé

# SpringBoot Application

```
package com.demo.firstspringbootrestapp;

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

@SpringBootApplication
public class FirstspringbootrestappApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstspringbootrestappApplication.class, args);
    }

}
```

# SpringBoot Application

```
package com.demo.firstspringbootrestapp;

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

@SpringBootApplication
public class FirstspringbootrestappApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstspringbootrestappApplication.class, args);
    }

}
```

# SpringBoot Application

```
package com.demo.firstspringbootrestapp;

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

@SpringBootApplication
public class FirstspringbootrestappApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstspringbootrestappApplication.class, args);
    }

}
```

Activation  
de

Auto configuration  
Component scanning  
Additionnal configuration

}

=

@EnableAutoConfiguration  
+ @ComponentScan  
+ @Configuration

# @SpringBootApplication

Annotation	Description
@EnableAutoConfiguration	Activation les support d'auto-configuration de Spring Boot
@ComponentScan	Scan des composants depuis le package courant et récursivement dans les sous packages du package courant
@Configuration	Permet l'enregistrement de beans Spring supplémentaires (via avec l'annotation @Bean), et l'import de multiples classes de configuration.

# SpringApplication

```
package com.demo.firstspringbootrestapp;

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

@SpringBootApplication
public class FirstspringbootrestappApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstspringbootrestappApplication.class, args);
    }

}
```




# SpringApplication

```
package com.demo.firstspringbootrestapp;

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

@SpringBootApplication
public class FirstspringbootrestappApplication {

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



# SpringApplication

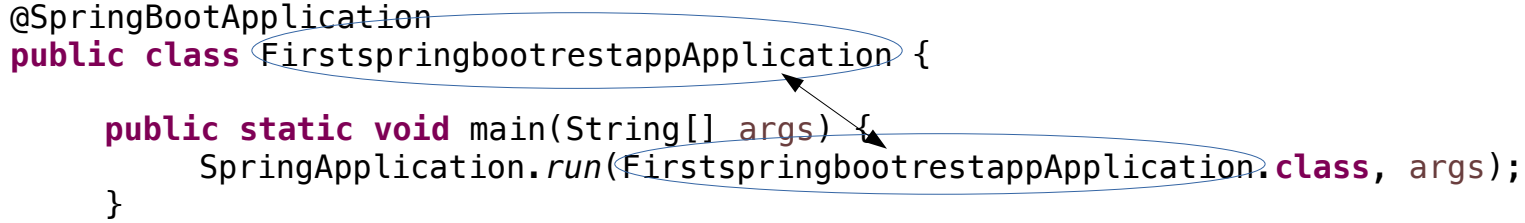
```
package com.demo.firstspringbootrestapp;

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

@SpringBootApplication
public class FirstspringbootrestappApplication {

    public static void main(String[] args) {
        SpringApplication.run(FirstspringbootrestappApplication.class, args);
    }

}
```

A diagram consisting of two blue ovals. The first oval encircles the text '@SpringBootApplication' and 'public class FirstspringbootrestappApplication {'. The second oval encircles the text 'SpringApplication.run(FirstspringbootrestappApplication.class, args);'. A black arrow points from the first oval to the second oval, indicating that the SpringBootApplication annotation is used to run the FirstspringbootrestappApplication class.

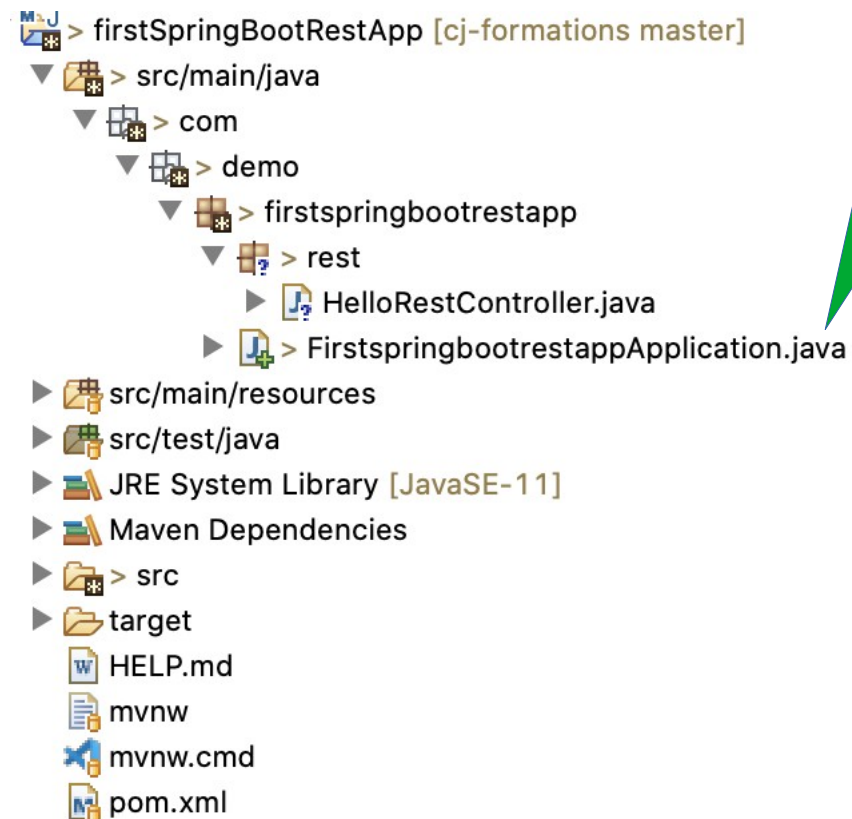
# Component Scanning

Il est recommandé de placer la classe principale `SpringBootApplication` au dessus de tous les autres packages de l'application

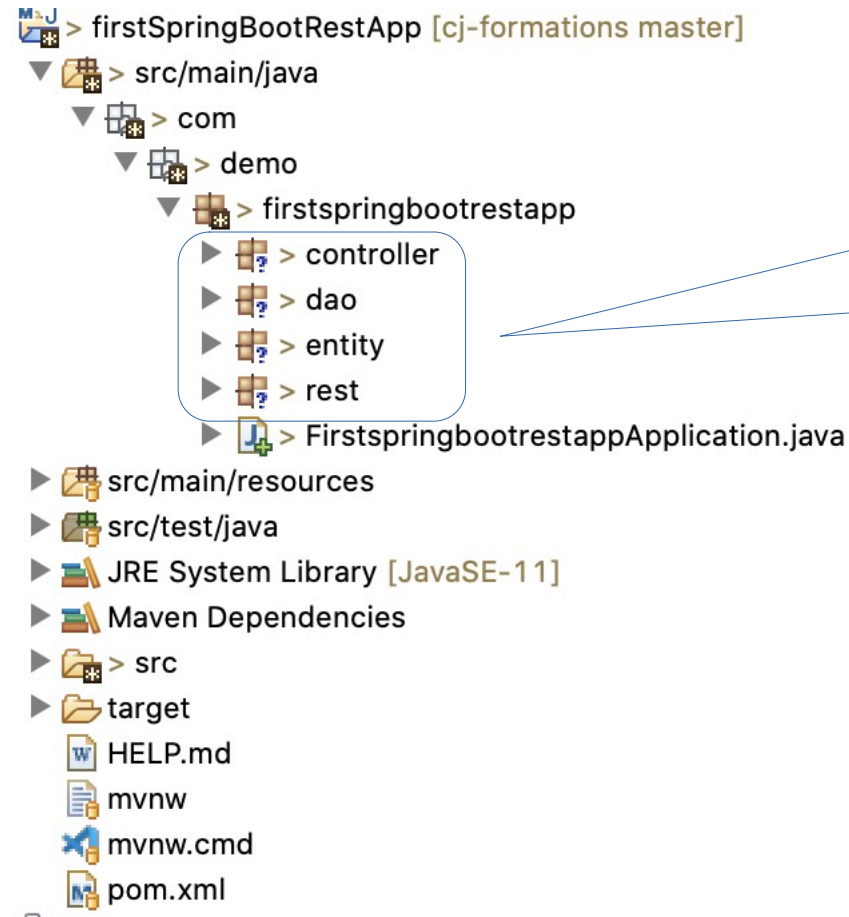
- Cela permet d'utiliser le package de la classe principale comme le package de base nécessaire au scanning des composants.
- Cela évite de devoir spécifier un package de base explicitement.

# Exemple

Scan automatique des composants dans les sous packages



# Sous package multiples



Inclure plusieurs sous -package, les composant à l'intérieur seront gérés par Spring

# Scanner des packages ailleurs

- On sait que les composants dans les packages situés "au dessous" de la classe principale seront chargés
- Comment scanner des structures de packages ailleurs dans l'application ? (sans lien dans l'arborescence des packages) ?

# Expliciter le scan d'un package

- @SpringBootApplication permet de faire cela

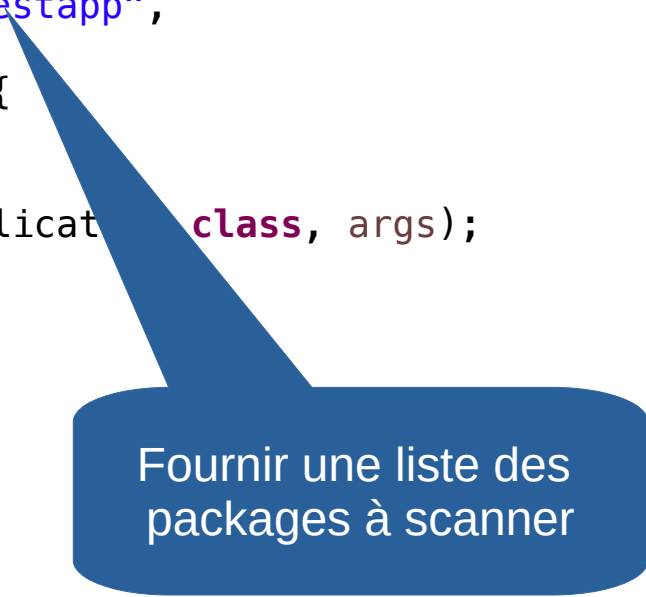
```
package com.demo.firstspringbootrestapp;

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

@SpringBootApplication(
    scanBasePackages = { "com.demo.firstspringbootrestapp",
                        "com.site", "org.domain.util" })
public class FirstspringbootrestappApplication {

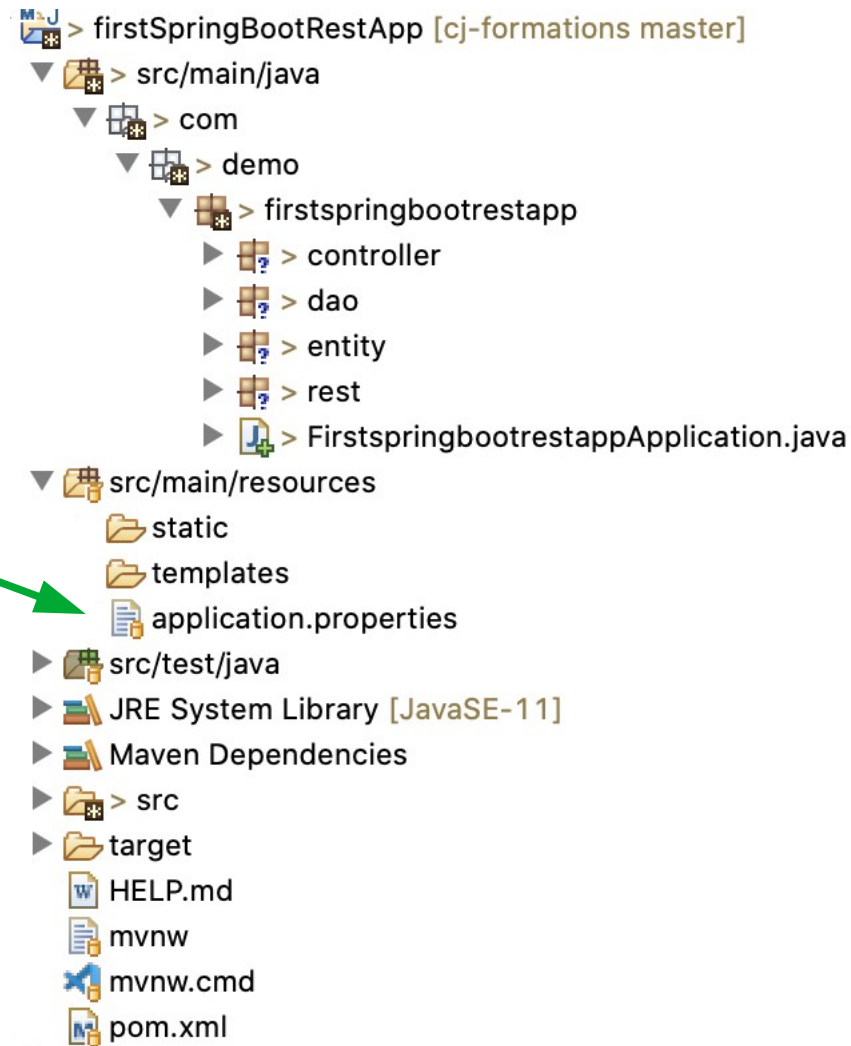
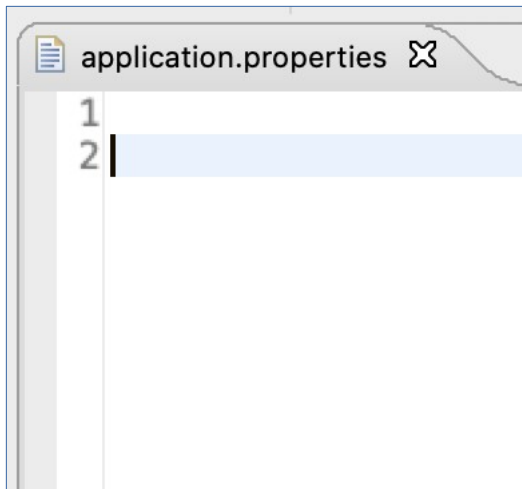
    public static void main(String[] args) {
        SpringApplication.run(FirstspringbootrestappApplication.class, args);
    }

}
```



Fournir une liste des packages à scanner

# application.properties





# Application.properties (suite)

- Par défaut spring boot va charger des propriétés depuis application.properties
- Sans que l'on ai besoin de configurer ce comportement





# Exemple :

application.properties

```
server.port=8081  
contact.email=john.connor@skynet.us  
contact.name=John connor  
entreprise.name=skynet
```

# Lire les propriétés

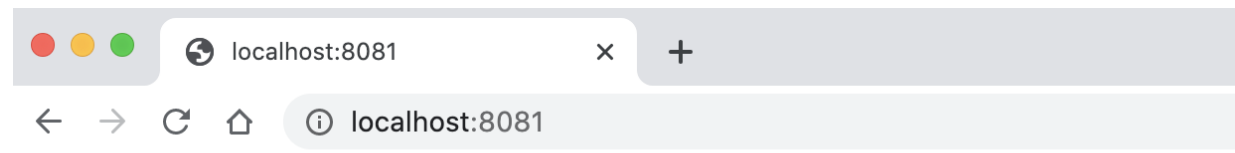
src/main/resources/application.properties

```
server.port=8081
contact.email=john.connor@skynet.us
contact.name=John connor
entreprise.name=skynet
```

```
@RestController
public class HelloRestController {

    @Value("${contact.name}")
    private String contactName;

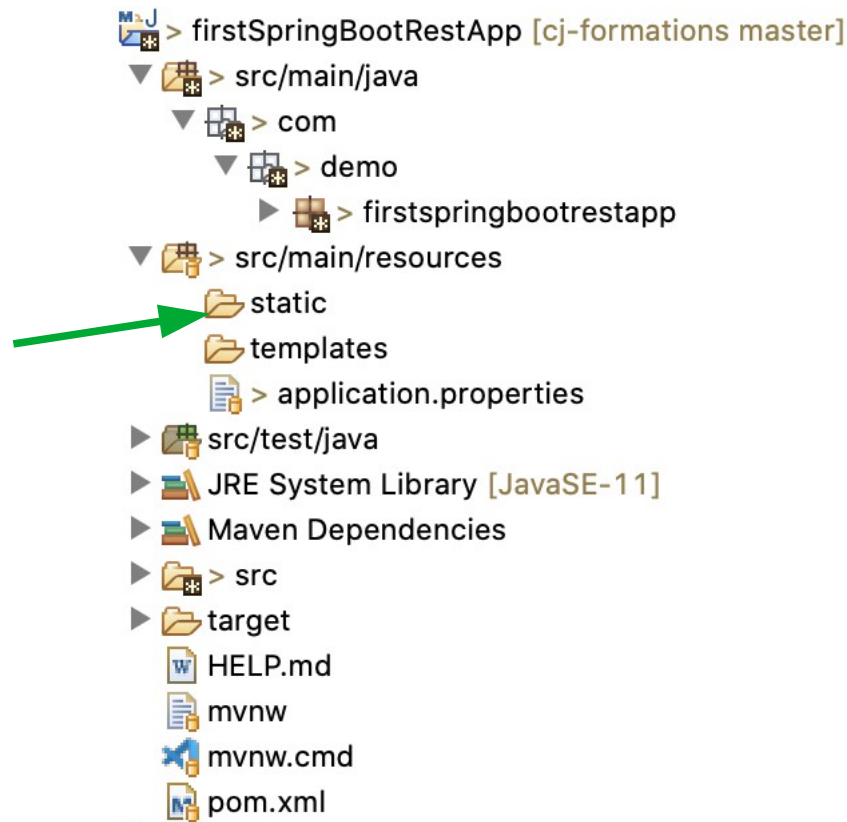
    @GetMapping("/")
    public String sayHello() {
        return "Hello " + contactName + " ! L'heure du serveur est " +
            LocalDateTime.now();
    }
}
```



Hello John connor ! L'heure du serveur est 2020-10-20T14:56:25.262596

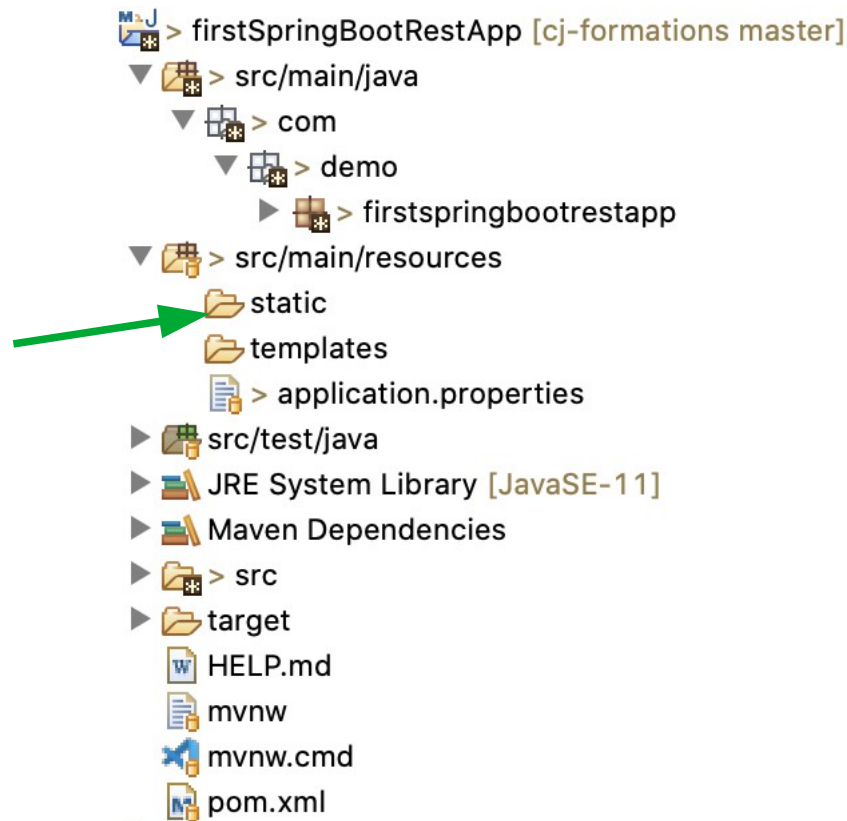
# Le contenu statique

- Il y a dans le src-main/resources un répertoire static



# Le contenu statique

- Il y a dans le src-main/resources un répertoire static



Attention : Warning !!  
N'utilisez pas le dossier src/main/webapp si votre application est packagée en jar



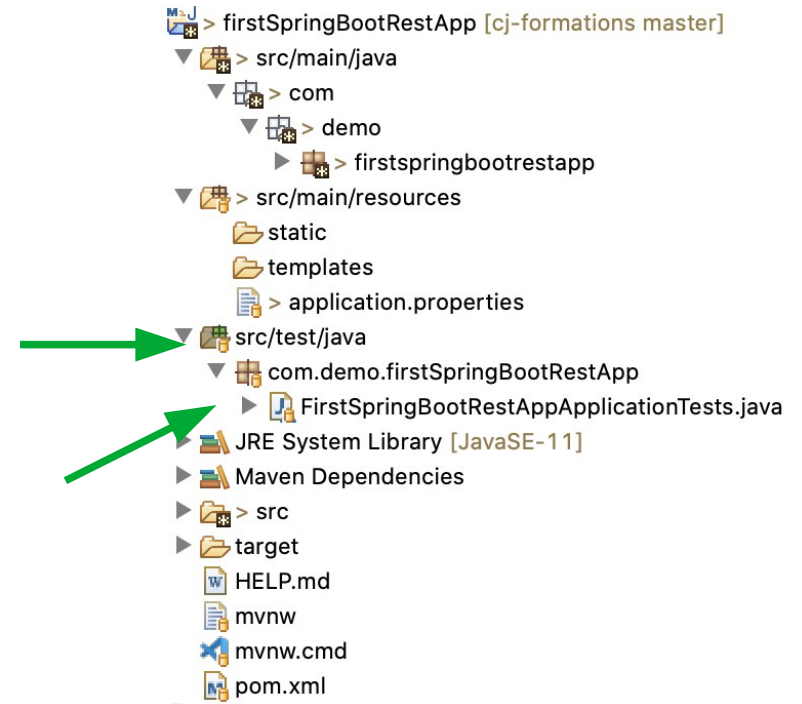
# templates

- Springboot autoconfigure les moteurs de template : FreeMarker, Thymleaf, Mustache ...
- SpringBoot charge les templates depuis `src/main/resources/templates`

# Tests Unitaires

Il existe un répertoire dédié aux tests unitaires

Il existe en particulier une classe créée par Spring Initializr :



```
package com.demo.firstSpringBootTestApp;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class FirstSpringBootTestAppApplicationTests {

    @Test
    void contextLoads() {
    }

}
```



# Tour d'un projet

- Nous avons fini ce premier aperçu rapide d'un projet Spring Boot
- J'espère qu'il vous donne envie d'en savoir plus
- Nous allons aborder plusieurs points clés de springBoot dans les prochains chapitres