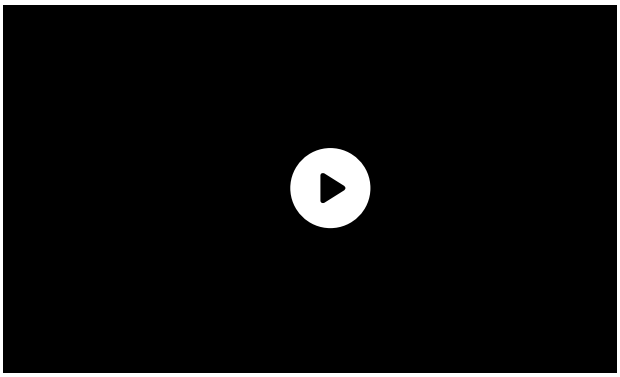
(/)

Guide to Internationalization in Spring Boot

FEATURED VIDEOS



Last updated: May 11, 2024

Written by: baeldung (https://www.baeldung.com/author/baeldung)

Reviewed by: Kevin Gilmore (https://www.baeldung.com/editor/kevin-author)

Spring Boot (https://www.baeldung.com/category/spring/spring-boot)

Get started with Spring Boot and with core Spring, through the *Learn Spring* course:

1. Overview

In this quick tutorial, we're going to take a look at how we can **add internationalization to a Spring Boot application**.

2. Maven Dependencies

For development, we need the following dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
    <version>1.5.2.RELEASE</version>
</dependency>
```

The latest version of spring-boot-starter-thymeleaf (https://mvnrepository.com/search?q=spring-boot-starter-thymeleaf)is available on Maven Central.

3. *LocaleResolver*

In order for our application to be able to determine which locale is currently in use, we need to add a *LocaleResolver* bean:

```
@Bean
public LocaleResolver localeResolver() {
    SessionLocaleResolver slr = new SessionLocaleResolver();
    slr.setDefaultLocale(Locale.US);
    return slr;
}
```

The *LocaleResolver* interface has implementations that determine the current locale based on the session, cookies, the *Accept-Language* header, or a fixed value.

In our example, we've used the session-based resolver *SessionLocaleResolver* and set a default locale with the value *US*.

4. *LocaleChangeInterceptor*

Next, we need to add an interceptor bean that will switch to a new locale based on the value of the *lang* parameter when present on the request:

```
@Bean
public LocaleChangeInterceptor localeChangeInterceptor() {
    LocaleChangeInterceptor lci = new LocaleChangeInterceptor();
    lci.setParamName("lang");
    return lci;
}
```

In order for this bean to take effect, we need to add it to the application's interceptor registry.

To achieve this, our *@Configuration* class has to implement the *WebMvcConfigurer* interface and override the *addInterceptors()* method:

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(localeChangeInterceptor());
}
```

5. Defining the Message Sources

By default, a Spring Boot application will look for message files containing internationalization keys and values in the *src/main/resources* folder.

Typically, the files for each locale will be named *messages_XX.properties*, where *XX* is the locale code. We can also define a fallback file *messages.properties*.

However, the fallback file should not be considered related to the default locale. They are two separate concepts.

The default locale is the locale to default to when the requested locale is unavailable, or null.

On the other hand, the fallback file is a place to look up properties when the locale translation fails.

If a key does not exist in a specific locale file, then the application will simply fall back to the fallback file.

(/)

The keys for the values that will be localized have to be the same in every file, with values appropriate to the language they correspond to.

For instance, let's define English properties in the fallback file *messages.properties*:

```
greeting=Hello! Welcome to our website!
lang.change=Change the language
lang.eng=English
lang.fr=French
```

Next, let's create a file called *messages_fr.properties* for the French language with the same keys:

```
greeting=Bonjour! Bienvenue sur notre site!
lang.change=Changez la langue
lang.eng=Anglais
lang.fr=Francais
```

6. Controller and HTML Page

Let's create a controller mapping that will return a simple HTML page called *international.html* that we want to see in two different languages:

```
@Controller                    (/)
public class PageController {

    @GetMapping("/international")
    public String getInternationalPage() {
        return "international";
    }
}
```

Since we're using Thymeleaf to display the HTML page, the locale-specific values will be accessed using the keys with the syntax *#{key}*:

```
<h1 th:text="#{greeting}"></h1>
```

The syntax for JSP files is a little different:

```
<h1><spring:message code="greeting" text="default"/></h1>
```

If we want to access the page with the two different locales, we have to add the parameter *lang* to the URL in the form: */international?lang=fr*

If no *lang* parameter is present on the URL, the application will use the default locale, which in our case is the *US* locale.

Let's add a drop-down to our HTML page with the two locales whose names are also localized in our properties files:

```
<span th:text="#{lang.change}">(x)</span>:
<select id="locales">
    <option value=""></option>
    <option value="en" th:text="#{lang.eng}"></option>
    <option value="fr" th:text="#{lang.fr}"></option>
</select>
```

Then we can add a jQuery script that will call the */international* URL with the respective *lang* parameter, depending on which drop-down option the user selects:

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js">
</script>
<script type="text/javascript">
$(document).ready(function() {
    $("#locales").change(function () {
        var selectedOption = $('#locales').val();
        if (selectedOption != ''){
            window.location.replace('international?lang=' +
selectedOption);
        }
    });
});
</script>
```

7. Running the Application

In order to initialize our application, we have to add a main class and annotate it with *@SpringBootApplication*:

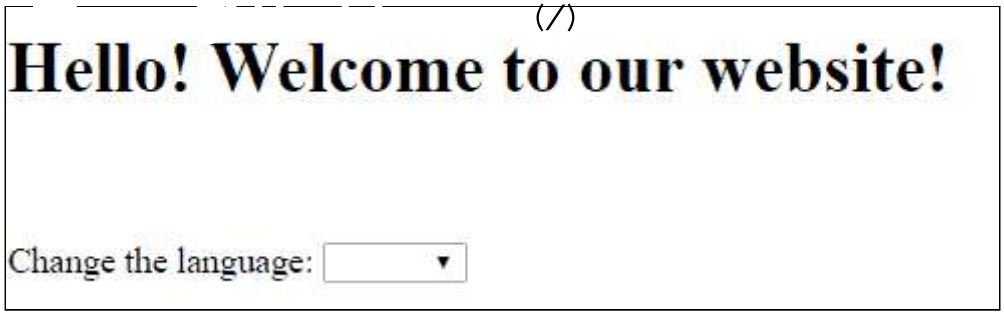
```
@SpringBootApplication
public class InternationalizationApp {

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

Depending on the selected locale, we'll view the page in either English or French when running the application.

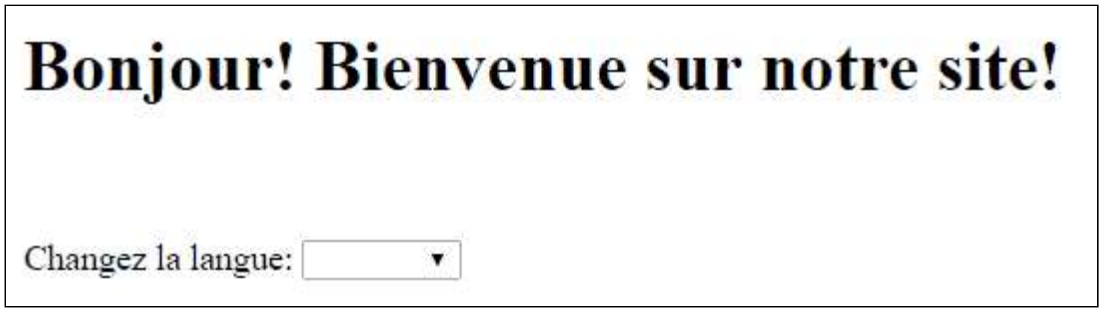
Let's see the English version:

(/)



(/wp-content/uploads/2017/03/piceng.png)

And now, let's see the French version:



(/wp-content/uploads/2017/03/picfr.png)

8. Conclusion

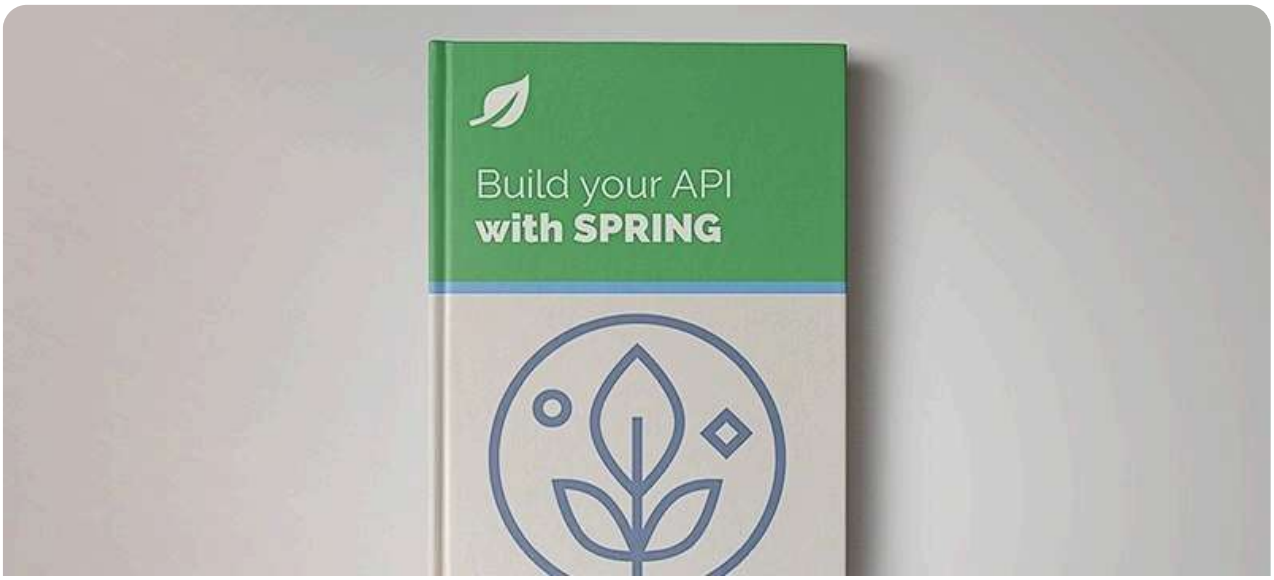
In this tutorial, we've shown how we can use the support for internationalization in a Spring Boot application.

The full source code for the example can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-boot-modules/spring-boot-mvc).

(/)

Get started with Spring Boot and with core Spring, through the *Learn Spring* course:

>> CHECK OUT THE COURSE (/ls-course-end)



Learning to build your API
with Spring?

Download the E-book (/rest-api-spring-guide)

(/)

COURSES

ALL COURSES (/COURSES/ALL-COURSES)
ALL BULK COURSES (/COURSES/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/COURSES/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)
JAVA ARRAY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/JAVA-ARRAY)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS/)
PARTNER WITH BAELDUNG (/PARTNERS/WORK-WITH-US)
EBOOKS (/LIBRARY/)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)

(/)