Spring MVC – Internationalization (i18n) and Localization (L10n)

Last Updated : 17 Dec, 2023

Spring MVC is a Model-View-Controller (MVC) framework that is used to build web applications in Java. The Spring MVC pattern has three parts:

- **Model:** The model contains the data that needs to be displayed on the view. A simple POJO class can be considered as a model.
- **View:** The view is used for rendering the UI Operations.
- **Controller:** The controller accepts user requests and passes them to the view for rendering.

Prerequisites of Topic

- Java Development Kit (JDK) 11 or higher
- Spring Boot 2.7.7 or higher
- IDE of your choice (e.g., Eclipse, IntelliJ IDEA, STS(Spring Tool Suite))

> ***Note:** Here I'm Using STS as a IDE.*

Spring MVC Internationalization (i18n)

Internationalization is the process of developing software so that it can be easily adapted to different languages and regions without any code changes.

Core Components of Spring MVC i18n:

- **LocaleResolver:** LocaleResolver is an interface that defines a strategy for resolving the current locale. It determines the locale to use from the HTTP request.
- **MessageSource:** MessageSource is an interface in Spring that provides a way to resolve messages for a specific locale. It supports properties, XML, and YAML formats.

- ViewResolvers: ViewResolvers determine how to render views based on the current locale. ContentNegotiatingViewResolver resolves the best matching view for a locale
- **LocaleChangeInterceptor:** LocaleChangeInterceptor provides a mechanism for dynamic locale switching. It intercepts locale change requests and updates the locale in the request/session through the LocaleResolver.
- **Resource bundles:** Properties files named messages_xx.properties containing localized text values for each locale.

> *Note: The **MessageSource** interface in Spring allows for internationalization support.*

Localization (L10n) with Spring MVC

Spring MVC application must be localized to work in many languages and regions. This allows building applications that can be seamlessly localized for global users without code changes.

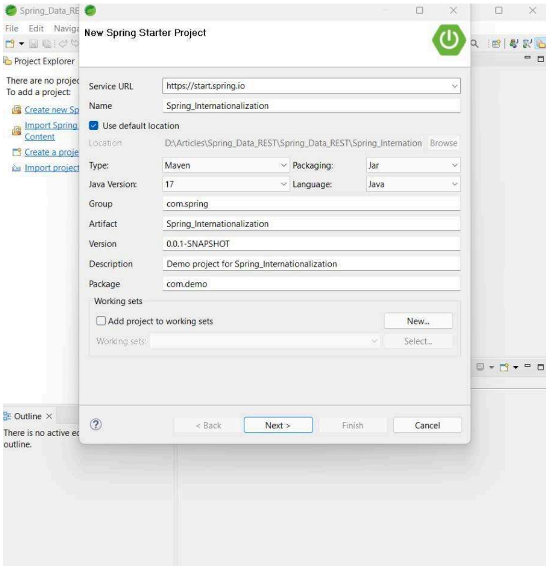
Core Components of Spring MVC L10n:

- **LocaleResolver:** LocaleResolver is an interface that determines the user's locale, which represents the user's language and country or region. Determines the locale to use from requests.
- **LocaleContext:** Makes the current locale available in controllers, and models. LocaleContextHolder is usually used.
- **MessageSource:** MessageSource is an interface that provides a way to resolve messages from resource bundles based on the current locale.
- **ViewResolvers:** ViewResolvers determines how to render views based on the current locale. ContentNegotiatingViewResolver resolves the best matching view for a locale.
- **LocaleChangeInterceptor:** LocaleChangeInterceptor is an interceptor that allows for changing the locale based on a request parameter or attribute.
- **Resource bundles:** Resource bundles are files that contains locale-specific key-value pairs, where keys represent *message codes* and values represent the *translated messages*.

Example of Internationalization and Localization

Step 1: Spring MVC Project setup

- Create a new Maven project in your preferred IDE (e.g., IntelliJ Eclipse or Spring Tool Suite) and add the following dependencies.
  - **Spring Web**
  - **Thymeleaf**

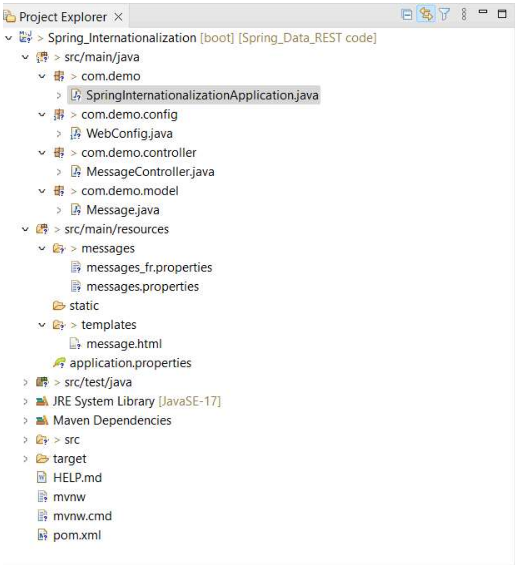


New Spring Starter Project

Service URL: https://start.spring.io
Name: Spring_Internationalization
☑ Use default location
Location: D:\Articles\Spring_Data_REST\Spring_Data_REST\Spring_Internation Browse
Type: Maven Packaging: Jar
Java Version: 17 Language: Java
Group: com.spring
Artifact: Spring_Internationalization
Version: 0.0.1-SNAPSHOT
Description: Demo project for Spring_Internationalization
Package: com.demo
Working sets
☐ Add project to working sets New...
Working sets: Select...

< Back Next > Finish Cancel

Project Structure:



```
Project Explorer
Spring_Internationalization [boot] [Spring_Data_REST code]
  src/main/java
    com.demo
      SpringInternationalizationApplication.java
    com.demo.config
      WebConfig.java
    com.demo.controller
      MessageController.java
    com.demo.model
      Message.java
  src/main/resources
    messages
      messages_fr.properties
      messages.properties
    static
    templates
      message.html
    application.properties
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Step 2: Create Message Controller: MessageController.Java

Java

```java
package com.demo.controller;

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

import com.demo.model.Message;
```

```
@Controller
public class MessageController {

     @GetMapping("/message")
    public String message(Model model) {
        Message message = new Message("data.message");
        model.addAttribute("message", message);
        return "message";
    }
}
```

Step 3: Create Message Model: Message.java

Java

```java
package com.demo.model;
public class Message {

    private String key;

    public Message(String key)
    {
      this.key = key;
    }

    public String getKey()
    {
      return key;
    }
}
```

Step 4: Create Views: message.html

- This file, located in **src/main/resources/templates**, contains a message.

HTML

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>GeeksforGeeks</title>
</head>
<body>
    <h1 th:text="#{${message.key}}" style="color: green;"></h1>
</body>
```

```
</html>
```

Step 5: Configure Internationalization: WebConfig.java

Java

```java
package com.demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.web.servlet.LocaleResolver;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.CookieLocaleResolver;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;

@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Bean public LocaleResolver localeResolver()
    {
        return new CookieLocaleResolver();
    }

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

    @Bean public ResourceBundleMessageSource messageSource()
    {
        ResourceBundleMessageSource messageSource
            = new ResourceBundleMessageSource();
        messageSource.setBasename("messages/messages");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
    }

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

Step 6: Messages Properties

- These properties files, located in **src/main/resources/messages**, contain localized messages for different languages.

1. **messages.properties**:

```
data.message=Hello and goodbye!
```

2. **messages_fr.properties:**

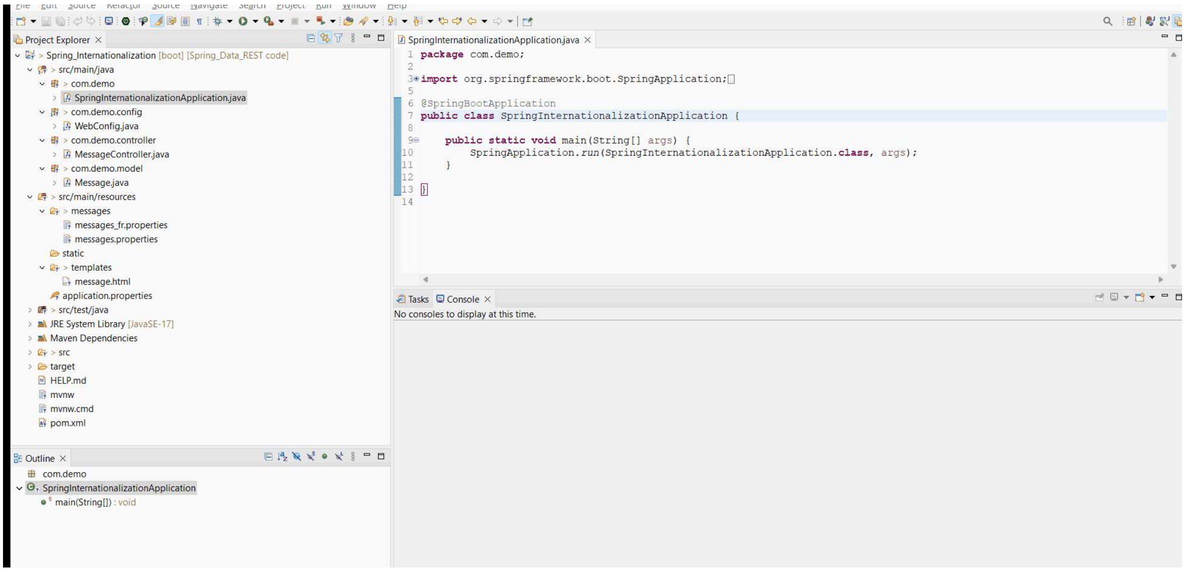
```
data.message=Bonjour et au revoir!
```

Step 7: Run the Application

- You can run your Spring Boot application from your IDE or by using the command-line tool provided by Spring Boot.

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

Output of an Application:

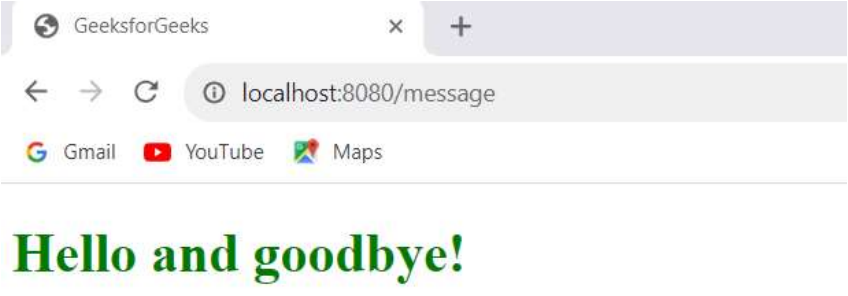


```
package com.demo;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class SpringInternationalizationApplication {

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

}
```

```
No consoles to display at this time.
```

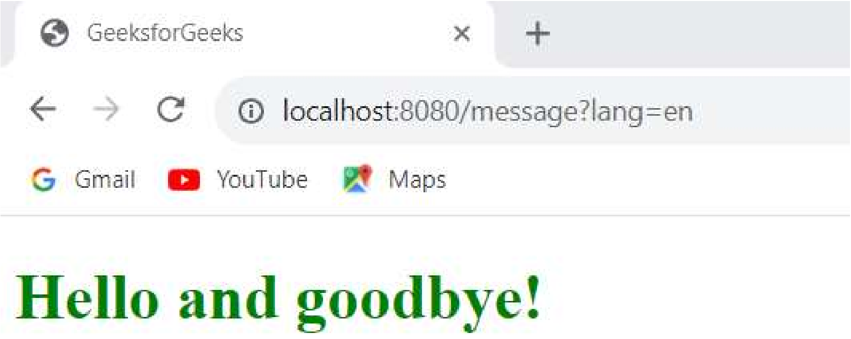
- Open your web browser and go to **http://localhost:8080/message**. You should see the default message in English.
- You can append **?lang=fr or ?lang=en** to the URL to switch between languages.

Output: http://localhost:8080/message



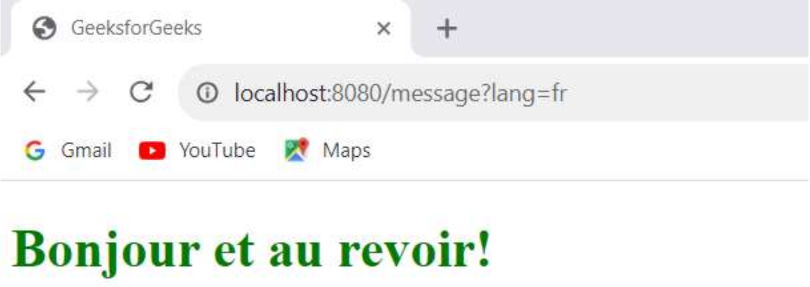
Hello and goodbye!

Output : http://localhost:8080/message?lang=en



Hello and goodbye!

Output: http://localhost:8080/message?lang=fr



Bonjour et au revoir!

P pran...

Next Article >

Explain about Internationalization and Localization

Similar Reads

Internationalization(I18N) in Java

Internationalization(I18N) is the process of designing web applications in such a way that which provides support for various countries, various languages,...

3 min read

Servlets - Internationalization(I18N)

Internationalization(I18N) is the process of designing web applications in such a way that it provides support for various countries, various languages, and...

4 min read

Spring - Resource Bundle Message Source (i18n)

A software is a multi purpose usage one. By using Message Source, it can be applicable to all languages. That concept is called i18n, that is according to t...

5 min read

Spring vs Spring Boot vs Spring MVC

Are you ready to dive into the exciting world of Java development? Whether you're a seasoned pro or just starting out, this article is your gateway to...

8 min read

Spring Boot - Internationalization

Before understanding Spring Boot-Internationalization, we must know what internalization in general means. Internationalization is an action or process o...

6 min read

View More Articles

Article Tags : Geeks Premier League 2023 Java-Spring Java-Spring-MVC Advance Java

+3 More

Practice Tags : Java

GeeksforGeeks
Sanchhaya Education Private Limited

A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

Company
About Us
Legal
In Media
Contact Us
Advertise with us
GFG Corporate Solution
Placement Training Program
GeeksforGeeks Community

Languages
Python
Java
C++
PHP
GoLang
SQL
R Language
Android Tutorial
Tutorials Archive

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
Top 100 DSA Interview Problems
DSA Roadmap by Sandeep Jain
All Cheat Sheets

Data Science & ML
Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
ML Maths
Data Visualisation Tutorial
Pandas Tutorial
NumPy Tutorial
NLP Tutorial
Deep Learning Tutorial

Web Technologies
HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
Bootstrap
Web Design

Python Tutorial
Python Programming Examples
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question
Django

Computer Science
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

DevOps
Git
Linux
AWS
Docker
Kubernetes
Azure

Software Development

Software Testing

GCP

DevOps Roadmap

System Design

High Level Design

Low Level Design

UML Diagrams

Interview Guide

Design Patterns

OOAD

System Design Bootcamp

Interview Questions

Inteview Preparation

Competitive Programming

Top DS or Algo for CP

Company-Wise Recruitment Process

Company-Wise Preparation

Aptitude Preparation

Puzzles

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

Commerce

World GK

GeeksforGeeks Videos

DSA

Python

Java

C++

Web Development

Data Science

CS Subjects