Ai name: ChatGPT 4o

Prompt:

Hi, How can I achieve internalization in java spring project (let's say we want to have polish and english)? What would be classes/files that I need?

Answer:

To achieve internationalization (i18n) in a Java Spring project, you would typically follow these steps to enable support for multiple languages such as Polish and English.

**Steps for Internalization in Java Spring:**

**1. Create Properties Files**

First, you'll need to create properties files for each language you want to support. For English and Polish, you can create the following files:

* messages.properties (for default language, typically English)
* messages\_pl.properties (for Polish)

These files will contain key-value pairs, where each key corresponds to a string used in your application, and the value is its translation.

**Example:**

* messages.properties (for English):
* greeting=Hello
* farewell=Goodbye
* messages\_pl.properties (for Polish):
* greeting=Cześć
* farewell=Do widzenia

Store these files in src/main/resources (or a suitable folder if you use a custom structure).

**2. Configure Spring's Message Source**

Spring provides a built-in mechanism to manage message resolution using MessageSource. You need to configure this in your application.properties or application.yml to specify how Spring should handle the message properties files.

In application.properties:

spring.messages.basename=messages

spring.messages.encoding=UTF-8

Here, basename=messages means that Spring will look for messages.properties and messages\_<locale>.properties files in the classpath.

**3. Create a MessageSource Bean in Configuration Class**

It's also a good practice to define a MessageSource bean in your configuration class so that Spring can inject the proper message source into controllers and other components.

**Example Configuration Class:**

import org.springframework.context.MessageSource;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.support.ResourceBundleMessageSource;

@Configuration

public class AppConfig {

@Bean

public MessageSource messageSource() {

ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();

messageSource.setBasename("messages"); // Points to messages.properties

messageSource.setDefaultEncoding("UTF-8");

return messageSource;

}

}

**4. Configure Locale Resolver**

You'll need to set up a LocaleResolver to determine the current locale of the user. You can use SessionLocaleResolver or AcceptHeaderLocaleResolver depending on how you want to manage the user's language preferences.

**Example using SessionLocaleResolver:**

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.i18n.SessionLocaleResolver;

import org.springframework.web.servlet.LocaleResolver;

import java.util.Locale;

@Configuration

public class LocaleConfig {

@Bean

public LocaleResolver localeResolver() {

SessionLocaleResolver resolver = new SessionLocaleResolver();

resolver.setDefaultLocale(Locale.ENGLISH); // Default language (English)

return resolver;

}

}

Alternatively, you can use AcceptHeaderLocaleResolver to change the language based on the Accept-Language header sent by the client.

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver;

import org.springframework.web.servlet.LocaleResolver;

@Configuration

public class LocaleConfig {

@Bean

public LocaleResolver localeResolver() {

return new AcceptHeaderLocaleResolver();

}

}

**5. Controller or Service to Change Locale**

You can create an endpoint in your controller to change the locale, typically by modifying the session or based on the request's headers.

**Example:**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.MessageSource;

import org.springframework.context.i18n.LocaleContextHolder;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.i18n.LocaleContextHolder;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import java.util.Locale;

@Controller

public class LocaleController {

@Autowired

private MessageSource messageSource;

@RequestMapping("/setLocale")

public String setLocale(String lang, RedirectAttributes redirectAttributes) {

Locale locale = new Locale(lang);

LocaleContextHolder.setLocale(locale);

redirectAttributes.addFlashAttribute("message",

messageSource.getMessage("greeting", null, LocaleContextHolder.getLocale()));

return "redirect:/";

}

}

In this example, the user can switch between languages (English, Polish) by calling /setLocale?lang=en or /setLocale?lang=pl.

**6. Use Messages in Views or Code**

In your application, you can now retrieve localized messages either in your code or views (e.g., Thymeleaf templates).

**In Java (Controller/Service):**

@Autowired

private MessageSource messageSource;

public String getGreeting() {

return messageSource.getMessage("greeting", null, LocaleContextHolder.getLocale());

}

**In Thymeleaf Template:**

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

The above code will display the localized greeting based on the user's locale (either in English or Polish).

**Summary of Required Classes/Files:**

* **messages.properties** and **messages\_pl.properties** for translations.
* **MessageSource bean** in configuration to load message files.
* **LocaleResolver** to handle locale management (e.g., SessionLocaleResolver or AcceptHeaderLocaleResolver).
* **Controller** to switch languages (optional but useful for user control).
* **Localized message retrieval** in Java code or templates.

This setup allows your Spring project to support multiple languages, such as English and Polish, based on the user's locale.