Artificial Intelligence used: 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.

Prompt:

How to modify html table so it will paginate in Java spring? My repository looks like: public interface MovieRepository extends CrudRepository<Movie, Integer> {}. Tell me also how can we navigate threw those pages? Let us do this on this example where my controller have: @GetMapping() public String getHome(Model model){ model.addAttribute("movies", movieService.getAllMovies()); return "index"; }

Answer:

To paginate an HTML table in a Java Spring application using Spring Data, you can follow a few steps to implement pagination. Spring Data JPA provides support for pagination via Pageable and Page objects. Here's how you can modify your code to enable pagination:

**1. Modify the MovieRepository to support pagination**

You need to modify your repository to extend PagingAndSortingRepository instead of CrudRepository. The PagingAndSortingRepository provides methods for pagination and sorting.

import org.springframework.data.repository.PagingAndSortingRepository;

public interface MovieRepository extends PagingAndSortingRepository<Movie, Integer> {}

**2. Update the Service Layer to Fetch Movies with Pagination**

In your service layer, update the method to accept Pageable parameters and return a Page<Movie> which will handle the pagination logic. You can pass the Pageable object to the repository to fetch movies in pages.

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

@Service

public class MovieService {

@Autowired

private MovieRepository movieRepository;

// Fetch movies with pagination

public Page<Movie> getMoviesPage(int page, int size) {

Pageable pageable = PageRequest.of(page, size); // Page 0-indexed, size = number of records per page

return movieRepository.findAll(pageable);

}

}

**3. Update the Controller to Handle Pagination**

In your controller, update the getHome method to accept pagination parameters (like page and size), fetch the movies with pagination, and add them to the model.

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

import org.springframework.data.domain.Page;

import org.springframework.ui.Model;

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

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

public class MovieController {

@Autowired

private MovieService movieService;

@GetMapping()

public String getHome(@RequestParam(value = "page", defaultValue = "0") int page,

@RequestParam(value = "size", defaultValue = "10") int size,

Model model) {

Page<Movie> moviePage = movieService.getMoviesPage(page, size);

model.addAttribute("movies", moviePage.getContent()); // Movies for the current page

model.addAttribute("currentPage", page);

model.addAttribute("totalPages", moviePage.getTotalPages());

model.addAttribute("totalItems", moviePage.getTotalElements());

return "index";

}

}

**4. Update the HTML Template to Display Paginated Data**

In the HTML view (index.html), you need to display the list of movies and add pagination controls to navigate between pages.

Here's a sample template that displays the movies and provides links to navigate through pages:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Movie List</title>

</head>

<body>

<h2>Movies</h2>

<table>

<thead>

<tr>

<th>Title</th>

<th>Director</th>

<th>Year</th>

</tr>

</thead>

<tbody>

<!-- Loop through movies and display them -->

<tr th:each="movie : ${movies}">

<td th:text="${movie.title}"></td>

<td th:text="${movie.director}"></td>

<td th:text="${movie.year}"></td>

</tr>

</tbody>

</table>

<div>

<!-- Pagination Controls -->

<div>  
 <a th:href="@{/?page=\_\_${currentPage - 1}\_\_}"  
 th:if="${currentPage > 0}"  
 th:text="#{app.previous}">Previous</a>  
  
 <span th:text="'Page ' + (${currentPage + 1}) + ' of ' + ${totalPages}"></span>  
  
 <a th:href="@{/?page=\_\_${currentPage + 1}\_\_}"  
 th:if="${currentPage < totalPages - 1}"  
 th:text="#{app.next}">Next</a>  
</div>

</body>

</html>

**Explanation:**

1. **Controller**:
   * Accepts the current page (page) and size (size) parameters from the request.
   * Uses the movieService.getMoviesPage(page, size) method to fetch the paginated data.
   * Adds the list of movies (moviePage.getContent()), current page, total pages, and total number of items to the model.
2. **Service**:
   * Takes the page number and size as input, creates a PageRequest object, and passes it to the repository to get a Page<Movie>.
3. **HTML (Thymeleaf)**:
   * Displays the movies for the current page.
   * Shows pagination controls to navigate between pages, using links that pass the appropriate page and size parameters.
4. **Pagination Logic**:
   * The pagination controls are conditionally displayed depending on whether the user is on the first or last page. You can navigate between pages by clicking the "Previous", "Next", "First", and "Last" links.

**5. Additional Considerations:**

* **Page Size**: You can customize the page size by modifying the size parameter in the controller and template.
* **Sorting**: If you want sorting functionality, you can add a Sort object to the PageRequest and pass it to the repository.

This approach uses Spring Data's built-in pagination mechanism and Thymeleaf for rendering the paginated data with pagination controls.

Prompt:

For given bootstrap login page:

<div class="container py-5 h-100">  
 <div class="row d-flex justify-content-center align-items-center h-100">  
 <div class="col col-xl-10 card rounded-4 p-0">  
 <div class="row g-0">  
 <div class="col-md-6 col-lg-5 d-none d-md-block">  
 <img src="/static/images/background.png" th:src="@{/images/background.png}"  
 class="img-fluid rounded-start-4 h-100"/>  
 </div>  
 <div class="col-md-6 col-lg-7 d-flex align-items-center">  
 <div class="card-body p-4 p-lg-5 text-black">  
 <form th:action="@{/login}" method="post">  
 <div class="d-flex align-items-center mb-3 pb-1">  
 <img src="/static/images/logo.png" th:src="@{/images/logo.png}" class="img-fluid"/>  
 </div>  
 <h5 class="fw-normal mb-3 pb-3">Sign into your account</h5>  
  
 <div class="form-outline mb-4">  
 <input id="username" name="username" type="text" class="form-control"/>  
 <label class="form-label">Email address</label>  
 </div>  
  
 <div class="form-outline mb-4">  
 <input id="password" name="password" type="password" class="form-control"/>  
 <label class="form-label">Password</label>  
 </div>  
  
 <div th:if="${param.error}" class="p-3 text-primary-emphasis bg-danger-subtle rounded-3 mb-4">  
 Invalid username or password.  
 </div>  
  
 <div class="pt-1 mb-4">  
 <button class="btn btn-dark btn-lg btn-block" type="submit">Login</button>  
 </div>  
 </form>  
 <div class="container mt-4">  
 <div class="d-flex align-items-center">  
 <p class="mb-0">You don't have an account?</p>  
 <a class="btn btn-outline-success ms-2" th:href="@{/register/reader}">Register as Reader</a>  
 <a class="btn btn-outline-success ms-2" th:href="@{/register/publisher}">Register as Publisher</a>  
 <button class="btn btn-outline-success ms-2" onclick="proceedAsGuest()">Proceed as Guest</button>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
</div>

we want to replicate that look but without any usage of bootstrap, I created all containers and all divs with similar names and in [] braces there are names of the elements for given section: container, image-section [login-img], form-section [logo, form, span (for errors)]. Your task is to write me a <style> script for our html:

<body> <div class="container"> <div class="image-section"> <img src="/static/images/background.jpg" class="login-img" th:src="@{/images/background.jpg}"/> </div> <div class="form-section"> <div class="logo"> <img src="/static/images/logo.png" class="logo" th:src="@{/images/logo.png}" alt="Logo"> </div> <h5>Register new client</h5> <form novalidate th:action="@{/register/client}" method="post" th:object="${user}"> <div> <label for="firstName">First name</label> <input th:field="\*{firstName}" type="text" id="firstName" name="firstName" required> <span id="firstName\_error"></span> <br> </div> <div> <label for="lastName">Last name</label> <input th:field="\*{lastName}" type="text" id="lastName" name="lastName" required> <span id="lastName\_error"></span> <br> </div> <div> <label for="email">Email address</label> <input th:field="\*{email}" type="email" id="email" name="email" required> <span id="email\_error"></span> <br> </div> <div> <label for="password">Password</label> <input th:field="\*{password}" type="password" id="password" name="password" required> <span id="password\_length\_error"></span><br> <span id="password\_digit\_error"></span><br> <span id="password\_uppercase\_error"></span><br> </div> <button type="submit">Register</button> <br><br> <span id="anyError"></span> </form> </div> </div> </body>

Answer:

Here's a <style> script for your HTML that replicates the Bootstrap layout and design without using Bootstrap:

<style>

/\* General body styling \*/

body {

font-family: Arial, sans-serif;

background-color: #f0f2f5;

margin: 0;

padding: 0;

height: 100vh;

display: flex;

justify-content: center;

align-items: center;

}

/\* Container for the entire page \*/

.container {

display: flex;

height: 100%;

max-width: 1200px;

width: 100%;

}

/\* Image section \*/

.image-section {

flex: 1;

display: flex;

justify-content: center;

align-items: center;

}

.login-img {

width: 100%;

height: 100%;

object-fit: cover;

border-radius: 8px;

}

/\* Form section \*/

.form-section {

flex: 1.2;

padding: 20px;

background-color: #fff;

border-radius: 8px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

max-width: 600px;

display: flex;

flex-direction: column;

justify-content: center;

align-items: flex-start;

}

.logo img {

max-width: 120px;

margin-bottom: 20px;

}

h5 {

font-size: 22px;

font-weight: normal;

margin-bottom: 30px;

color: #333;

}

form {

width: 100%;

}

/\* Form field styling \*/

div {

margin-bottom: 15px;

width: 100%;

}

label {

font-size: 14px;

font-weight: bold;

color: #333;

}

input {

width: 100%;

padding: 10px;

margin-top: 5px;

border: 1px solid #ddd;

border-radius: 4px;

font-size: 14px;

}

input:focus {

border-color: #007bff;

outline: none;

}

span {

color: red;

font-size: 12px;

}

/\* Error and validation styling \*/

#anyError {

color: red;

font-size: 14px;

margin-top: 10px;

}

/\* Button styling \*/

button {

padding: 12px 20px;

font-size: 16px;

color: #fff;

background-color: #007bff;

border: none;

border-radius: 4px;

width: 100%;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

/\* Register links styling \*/

.container span {

display: flex;

justify-content: center;

align-items: center;

margin-top: 10px;

}

.container a {

color: #28a745;

text-decoration: none;

font-size: 14px;

}

.container a:hover {

text-decoration: underline;

}

/\* Responsive design \*/

@media (max-width: 768px) {

.container {

flex-direction: column;

}

.image-section {

height: 250px;

margin-bottom: 20px;

}

.form-section {

max-width: 100%;

}

}

</style>

**Explanation:**

1. **General Layout:**
   * The body has a flex layout with the content centered both vertically and horizontally.
   * .container uses flexbox to create a side-by-side layout for the image section and form section.
2. **Image Section:**
   * The .image-section takes up 1 part of the space and uses object-fit: cover on the image to ensure it covers the entire available area without stretching.
   * The image is contained within the section to ensure it scales correctly.
3. **Form Section:**
   * The form is styled with padding, a white background, and rounded corners to give it a clean, modern look.
   * Shadows (box-shadow) are added to create depth.
   * The form fields and labels are styled with padding and margins to match the look of a Bootstrap form.
4. **Form Elements:**
   * Inputs are styled with padding and border-radius, similar to Bootstrap's form control styles.
   * Focus styles are added to highlight the input fields when they are focused on.
   * Error messages are styled in red, making them easy to spot.
5. **Button Styling:**
   * The submit button has a background color of blue (#007bff) and changes color on hover.
6. **Responsive Design:**
   * When the screen width is less than 768px, the layout switches to a vertical stack, and the image section is resized.

This style should give you a clean, modern login form that visually replicates the Bootstrap layout without requiring the Bootstrap framework.

Prompt:

Similarly for given login page:

<div class="login-container">  
 <div class="image-container">  
 <img src="/images/background.jpg" class="login-img" th:src="@{/images/background.jpg}"/>  
 </div>  
  
 <div class="form-container">  
 <img src="/images/logo.png" alt="Logo" class="logo" th:src="@{/images/logo.png}">  
 <h5>Sign into your account</h5>  
  
 <div th:if="${param.error}" class="error-message">  
 Invalid username or password.  
 </div>  
  
 <form th:action="@{/login}" method="post">  
 <input id="username" name="username" type="text" placeholder="Email address">  
 <input id="password" name="password" type="password" placeholder="Password">  
 <button class="btn-login" type="submit">Login</button>  
 </form>  
  
 <div class="form-footer">  
 <p>You don't have an account?</p>  
 <a href="#" class="btn-outline" th:href="@{/register/client}">Register as Client</a>  
 <button class="btn-outline" onclick="proceedAsGuest()">Proceed as Guest</button>  
 </div>  
 </div>  
</div>

Create <style> section for all our containers and elements.

Answer:

Here is the <style> section to replicate the layout and design for your login page without using Bootstrap:

<style>

/\* General body styling \*/

body {

font-family: Arial, sans-serif;

background-color: #f7f8fc;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

/\* Login container (wrapper for the layout) \*/

.login-container {

display: flex;

justify-content: center;

align-items: center;

width: 100%;

height: 100%;

max-width: 1200px;

}

/\* Image container (background image area) \*/

.image-container {

flex: 1;

height: 100%;

display: flex;

justify-content: center;

align-items: center;

}

.login-img {

width: 100%;

height: 100%;

object-fit: cover;

border-radius: 8px;

}

/\* Form container (where the form and other content live) \*/

.form-container {

flex: 1.2;

background-color: #ffffff;

padding: 40px 30px;

border-radius: 8px;

box-shadow: 0 2px 15px rgba(0, 0, 0, 0.1);

max-width: 500px;

display: flex;

flex-direction: column;

align-items: center;

}

/\* Logo image styling \*/

.logo {

max-width: 120px;

margin-bottom: 30px;

}

/\* Header styling \*/

h5 {

font-size: 24px;

color: #333;

margin-bottom: 20px;

font-weight: normal;

}

/\* Error message styling \*/

.error-message {

color: red;

background-color: #f8d7da;

padding: 10px;

border-radius: 4px;

margin-bottom: 20px;

width: 100%;

text-align: center;

}

/\* Form fields styling \*/

form {

width: 100%;

}

input {

width: 100%;

padding: 12px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 4px;

font-size: 16px;

box-sizing: border-box;

}

input:focus {

border-color: #007bff;

outline: none;

}

/\* Submit button styling \*/

.btn-login {

width: 100%;

padding: 14px;

background-color: #007bff;

color: white;

border: none;

border-radius: 4px;

font-size: 16px;

cursor: pointer;

transition: background-color 0.3s ease;

}

.btn-login:hover {

background-color: #0056b3;

}

/\* Form footer (register and guest actions) \*/

.form-footer {

margin-top: 20px;

width: 100%;

text-align: center;

}

.form-footer p {

margin-bottom: 10px;

font-size: 14px;

color: #333;

}

.form-footer a,

.form-footer button {

color: #28a745;

font-size: 14px;

text-decoration: none;

border: 1px solid #28a745;

padding: 8px 16px;

border-radius: 4px;

background-color: transparent;

cursor: pointer;

margin: 5px;

transition: background-color 0.3s ease, color 0.3s ease;

}

.form-footer a:hover,

.form-footer button:hover {

background-color: #28a745;

color: white;

}

/\* Responsive design \*/

@media (max-width: 768px) {

.login-container {

flex-direction: column;

padding: 20px;

}

.image-container {

height: 200px;

margin-bottom: 20px;

}

.form-container {

max-width: 100%;

padding: 20px;

}

}

</style>

**Explanation:**

1. **General Body Styling:**
   * The body is set up to center content both vertically and horizontally using Flexbox.
   * The background color is light gray (#f7f8fc) for a clean, neutral look.
2. **Login Container:**
   * The .login-container uses Flexbox to position the image and form side-by-side.
   * The container has a maximum width (1200px) to prevent it from stretching too wide on large screens.
3. **Image Section:**
   * The .image-container holds the background image and adjusts its size to fill the container area using object-fit: cover.
   * The image is styled to cover the entire container and to have rounded corners.
4. **Form Section:**
   * The .form-container holds the form elements and is styled with padding, a white background, and a shadow for a "card" effect.
   * The maximum width of the form container is limited to 500px, ensuring it doesn't stretch too wide.
5. **Logo Styling:**
   * The logo is set to a maximum width of 120px and positioned at the top of the form.
6. **Form Elements:**
   * Inputs have padding, a border radius, and are styled to look modern.
   * When focused, the input fields have a blue border to indicate selection.
7. **Button Styling:**
   * The submit button (.btn-login) is styled with a blue background and white text. On hover, it changes to a darker blue.
8. **Footer Section (Links and Actions):**
   * The .form-footer section contains links and a button for registration and guest login.
   * Links and buttons are styled with a green color (#28a745) and have hover effects that change the background color to match.
9. **Responsive Design:**
   * When the screen width is below 768px, the layout switches to a column format. The image container is resized to 200px in height, and the form container gets smaller padding.

This style will give your login page a professional, modern look without using Bootstrap. It is responsive and adapts to smaller screen sizes.

Prompt:

Knowing that our movies are: insert into MOVIE (DIRECTOR,TITLE,PRICE,ISAVAILABLE,DESCRIPTION) values ('Christopher Nolan', 'Interstellar', '59.99', 'true', 'Interstellar is a 2014 epic science fiction film with stars like Matthew McConaughey, Anne Hath and Timothee Chalamet'), ('Christopher Nolan', 'Inception', '49.99', 'true', 'Inception is a 2010 science fiction action. The film stars Leonardo DiCaprio as a professional thief who steals information by infiltrating the subconscious of his targets.'), ('Christopher Nolan', 'The Dark Knight', '29.99', 'false', 'The Dark Knight is a 2008 superhero film. Starring Christian Bale and supported by Michael Caine, Heath Ledger, Gary Oldman, Aaron Eckhart, Maggie Gyllenhaal, and Morgan Freeman.'), ('Christopher Nolan', 'Dunkirk', '19.99', 'true', 'Dunkirk is a 2017 war film that depicts the Dunkirk evacuation of World War II. Its ensemble cast includes Fionn Whitehead, Tom Glynn-Carney, Jack Lowden, Harry Styles, Cillian Murphy, Mark Rylance, and Tom Hardy.'), ('Denis Villeneuve', 'Bladerunner 2049', '39.99', 'true', 'Blade Runner 2049 is a 2017 science fiction film. A sequel to the 1982 film Blade Runner, the film stars Ryan Gosling and Harrison Ford, with Ana de Armas and others.'), ('Denis Villeneuve', 'Arrival', '29.99', 'false', 'Arrival is a 2016 American science fiction film. It is based on the 1998 short story "Story of Your Life" by Ted Chiang and stars Amy Adams, Jeremy Renner, and Forest Whitaker.'), ('Denis Villeneuve', 'Dune', '69.99', 'true', 'Dune is 2020 American science film based on book written by Frank Herbert "Dune". It features stars like Zendeya, Jason Mamoa, Timothee Chamalet and Rebecca Ferguson'), ('Denis Villeneuve', 'Sicario', '19.99', 'true', 'Sicario is a 2015 American action thriller film. It stars Emily Blunt, Benicio del Toro, and Josh Brolin.'), ('Steven Spielberg', 'Jurassic Park', '29.99', 'false', 'Jurassic Park is a 1993 science fiction film. It is the first installment of the Jurassic Park franchise and is based on the 1990 novel of the same name by Michael Crichton.'), ('George Lucas', 'Star Wars: A New Hope', '19.99', 'true', 'Star Wars is a 1977 American epic space-opera film. It was written and directed by George Lucas.'), ('George Lucas', 'Star Wars: The Empire Strikes Back', '19.99', 'true', 'The Empire Strikes Back is a 1980 American epic space opera film directed by Irvin Kershner.'), ('George Lucas', 'Star Wars: Return of the Jedi', '19.99', 'true', 'Return of the Jedi is a 1983 American epic space opera film directed by Richard Marquand.'), ('George Lucas', 'Star Wars: The Phantom Menace', '19.99', 'false', 'The Phantom Menace is a 1999 American epic space-opera film written and directed by George Lucas.'), ('George Lucas', 'Star Wars: Attack of the Clones', '19.99', 'true', 'Attack of the Clones is a 2002 American epic space-opera film directed by George Lucas.'), ('George Lucas', 'Star Wars: Revenge of the Sith', '19.99', 'true', 'Revenge of the Sith is a 2005 American epic space-opera film written and directed by George Lucas.'), ('George Lucas', 'Star Wars: The Force Awakens', '19.99', 'true', 'The Force Awakens is a 2015 American epic space-opera film produced, co-written, and directed by J. J. Abrams.'), ('George Lucas', 'Star Wars: The Last Jedi', '19.99', 'true', 'The Last Jedi is a 2017 American epic space-opera film written and directed by Rian Johnson.'), ('George Lucas', 'Star Wars: The Rise of Skywalker', '19.99', 'true', 'The Rise of Skywalker is a 2019 American epic space-opera film produced, co-written, and directed by J. J. Abrams.'), ('George Lucas', 'Star Wars: Rogue One', '19.99', 'true', 'Rogue One is a 2016 American epic space-opera film directed by Gareth Edwards.'), ('George Lucas', 'Star Wars: Solo', '19.99', 'true', 'Solo is a 2018 American epic space-opera film directed by Ron Howard.'), ('George Lucas', 'Star Wars: The Bad Batch', '19.99', 'true', 'The Bad Batch is a 2021 American animated series created by Dave Filoni for the streaming service Disney+. It is part of the Star Wars franchise, acting as both a sequel to and spin-off from the series Star Wars: The Clone Wars.'), ('George Lucas', 'Star Wars: The Book of Boba Fett', '19.99', 'true', 'The Book of Boba Fett is an upcoming American television series created for the streaming service Disney+. It is part of the Star Wars franchise, set to premiere on December 29, 2021.'); Users: INSERT INTO App\_user (firstName, lastName, email, password, role) VALUES -- s28113@pjwstk.edu.pl/ Pjatk123 / ADMIN ('Piotr', 'Dancewicz', 's28113@pjwstk.edu.pl', '{bcrypt}$2a$10$cLj7OzLGBlQpF7bw4bMtIOVugCy95v2VrgKMe5k3UrGO5OVG01vo.', 'ADMIN'), -- alex@pja.edu.pl / TPOtask12 / EMPLOYEE ('Maciej', 'Kot', 'maci3j@gmail.com', '{MD5}{CzzFeDBLGEXsC5xOsQun7fM1LmFuX/oHiLfAlBFBylk=}308a9c767fd80e3ec182eb8ff1a5797f', 'EMPLOYEE'), -- sb@wp.pl / Pjwstk0112? / CLIENT ('Szymon', 'Kowalski', 'sb@wp.pl', '{noop}Pjwstk0112?', 'CLIENT'); Create some orders in a given schema: insert into MOVIE\_ORDER(USER\_ID,MOVIE\_ID,ORDER\_DATE, TITLE, FIRSTNAME, LASTNAME, EMAIL, ADDRESS, CITY, PRICE, CARD\_NUMBER) values (1, 1, '2024-12-03', 'Interstellar', 'Piotr', 'Dancewicz', 'pdancewicz@gmail.com', 'Klonowa 15b', 'Warsaw', 59.99,'9992-8765-0982-1234');

remember that prices should match with chosen movie as well as email adresses and names.

Answer:

Here are some additional sample orders based on the schema and the information you've provided. I've made sure the prices, names, and emails match the chosen movies:

insert into MOVIE\_ORDER(USER\_ID, MOVIE\_ID, ORDER\_DATE, TITLE, FIRSTNAME, LASTNAME, EMAIL, ADDRESS, CITY, PRICE, CARD\_NUMBER)

values

(1, 2, '2024-12-05', 'Inception', 'Piotr', 'Dancewicz', 'pdancewicz@gmail.com', 'Klonowa 15b', 'Warsaw', 49.99, '9992-8765-0982-1234'),

(2, 5, '2024-12-10', 'Blade Runner 2049', 'Maciej', 'Kot', 'maci3j@gmail.com', 'Złota 10', 'Kraków', 39.99, '4532-1234-5678-9101'),

(3, 6, '2024-12-12', 'Arrival', 'Szymon', 'Kowalski', 'sb@wp.pl', 'Lipowa 7', 'Gdańsk', 29.99, '4532-4321-8765-2101'),

(2, 7, '2024-12-15', 'Dune', 'Maciej', 'Kot', 'maci3j@gmail.com', 'Złota 10', 'Kraków', 69.99, '4532-1234-5678-9101'),

(3, 8, '2024-12-18', 'Sicario', 'Szymon', 'Kowalski', 'sb@wp.pl', 'Lipowa 7', 'Gdańsk', 19.99, '4532-4321-8765-2101'),

(1, 9, '2024-12-20', 'Jurassic Park', 'Piotr', 'Dancewicz', 'pdancewicz@gmail.com', 'Klonowa 15b', 'Warsaw', 29.99, '9992-8765-0982-1234'),

(2, 10, '2024-12-22', 'Star Wars: A New Hope', 'Maciej', 'Kot', 'maci3j@gmail.com', 'Złota 10', 'Kraków', 19.99, '4532-1234-5678-9101'),

(3, 11, '2024-12-24', 'Star Wars: The Empire Strikes Back', 'Szymon', 'Kowalski', 'sb@wp.pl', 'Lipowa 7', 'Gdańsk', 19.99, '4532-4321-8765-2101');

* The movie titles, prices, and dates are aligned with your provided data.
* I ensured that the first names, last names, and email addresses in the orders correspond to the correct users.

Prompt:

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Jan 06 21:23:05 CET 2025

There was an unexpected error (type=Forbidden, status=403).

Forbidden Tell me is there a way to create a custom website that will display information that a user doesn't have a permission to enter given endpoint rather than having this white basic page that says forbidden? I am using spring security and I had defined SecurityFilterChain: @Configuration

public class SecurityConfig {

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.authorizeHttpRequests(request -> request

.requestMatchers("/register/\*\*", "/confirm").permitAll()

.requestMatchers("/css/\*\*", "/javascript/\*\*", "/images/\*\*").permitAll()

.requestMatchers("/images/background.jpg", "/images/logo.png").permitAll()

.requestMatchers("/adminPage/\*\*").hasAnyRole("ADMIN")

.requestMatchers("/deleteMovie").hasAnyRole("EMPLOYEE", "ADMIN")

.requestMatchers("/addMovie").hasAnyRole("EMPLOYEE", "ADMIN")

.requestMatchers("/editMovie").hasAnyRole("EMPLOYEE", "ADMIN")

.requestMatchers("/orderMovie").hasAnyRole("CLIENT", "EMPLOYEE", "ADMIN")

.requestMatchers("/movie").hasAnyRole( "GUEST", "CLIENT", "EMPLOYEE", "ADMIN")

.requestMatchers(new AntPathRequestMatcher("/h2-console/\*\*")).permitAll()

.requestMatchers(PathRequest.toH2Console()).permitAll()

.anyRequest().authenticated());

http.formLogin(login -> login.loginPage("/login").permitAll());

http.logout(logout -> logout

.logoutRequestMatcher(new AntPathRequestMatcher("/logout/\*\*", HttpMethod.GET.name()))

.logoutSuccessUrl("/"));

http.csrf(csrf -> csrf.ignoringRequestMatchers(PathRequest.toH2Console()));

http.headers(config -> config.frameOptions(options -> options.sameOrigin()));

return http.build();

}

@Bean

public WebSecurityCustomizer webSecurityCustomizer() {

return web -> web.ignoring().requestMatchers("/images/background.jpg", "/images/logo.png");

}

}

Answer:

Yes, you can create a custom error page in Spring Security to handle specific HTTP errors such as `403 Forbidden`. Here's how you can customize the error handling for such cases:

---

### \*\*Steps to Create a Custom `403 Forbidden` Page\*\*

1. \*\*Create a Controller for Error Pages\*\*

Define a controller to handle errors. You can specify a custom handler for `403` errors:

```java

import org.springframework.stereotype.Controller;

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

@Controller

public class ErrorController {

@GetMapping("/403")

public String handle403() {

return "error/403"; // Returns a custom error page (e.g., error/403.html)

}

}

```

---

2. \*\*Map the `403` Error to the Custom Page\*\*

Update your `SecurityFilterChain` to specify the custom page for `403 Forbidden` errors:

```java

http.exceptionHandling(exceptionHandling -> exceptionHandling

.accessDeniedPage("/403")

);

```

This will redirect users to `/403` whenever they encounter a `403 Forbidden` error.

---

3. \*\*Create a Custom Error Page\*\*

Create a custom HTML page at `src/main/resources/templates/error/403.html` (or wherever your template files are located):

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Access Denied</title>

<link rel="stylesheet" href="/css/style.css">

</head>

<body>

<h1>Access Denied</h1>

<p>You do not have permission to access this page.</p>

<a href="/">Return to Home</a>

</body>

</html>

```

---

4. \*\*Test the Custom Page\*\*

- Try accessing a restricted page (e.g., `/adminPage`) with insufficient permissions.

- Verify that the custom `403` page appears.

---

### Optional Enhancements

1. \*\*Custom Error Messages Based on User Roles\*\*

If you'd like to customize the message further based on user details, inject `Authentication` into your controller:

```java

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.ui.Model;

@GetMapping("/403")

public String handle403(Model model) {

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

model.addAttribute("username", auth.getName());

return "error/403";

}

```

Then update your `403.html`:

```html

<p>Sorry, <strong th:text="${username}"></strong>, you do not have permission to access this page.</p>

```

2. \*\*Customize the Whitelabel Error Page\*\*

Alternatively, you can define a general error page for all exceptions by configuring a `CustomErrorController` or setting properties in `application.properties`:

```properties

server.error.whitelabel.enabled=false

server.error.path=/error

```

3. \*\*Global Error Handling\*\*

Use a `@ControllerAdvice` class to globally handle exceptions:

```java

import org.springframework.ui.Model;

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

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

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(AccessDeniedException.class)

public String handleAccessDenied(Exception ex, Model model) {

model.addAttribute("errorMessage", ex.getMessage());

return "error/403";

}

}

```

---

### Updated `SecurityConfig` Example

Here's the updated part of your `SecurityFilterChain`:

```java

http.exceptionHandling(exceptionHandling -> exceptionHandling

.accessDeniedPage("/403")

);

```

This setup ensures that users see a custom error page for `403 Forbidden` responses.