

РОЗРОБКА SPRING ЗАСТОСУНКУ ІЗ РЕАЛІЗАЦІЮ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ

Мета

- 1 Поглиблення знань із принципів побудови та застосування Spring Security.
- 2 Набуття навиків створення Spring-застосунків, в яких реалізована автентифікація та авторизація із застосуванням можливостей Spring Security.
- 3 Набуття навиків визначення ролей та користувачів системи.
- 4 Набуття навиків створення системи авторизації засобами Spring Security.
- 5 Набуття навиків застосування можливостей, що надають класи, що відповідають за підтримку сесій під час роботи із Spring-застосунками.
- 6 Набуття навичків додавання тегів Spring Security на веб-сторінок, що створюються із застосуванням Thymeleaf.
- 7 Вдосконалення навиків розробки user friendly applications.

Завдання роботи

Додати до Spring-застосунку, створеного під час виконання ЛР№3, функціональність, що забезпечує авторизацію та автентифікацію із застосуванням Spring Security.

Застосунок має забезпечити:

- реєстрацію в системі із збереження логіну та паролів користувачів;
- вхід до системи виключно зареєстрованих користувачів;
- надання доступу до даних, а також до виконання CRUD-операцій, відповідно до рівня доступу, який визначений при реєстрації;
- відображення імені користувача, який увійшов до системи;
- обмеження часу знаходження зареєстрованого користувача у системі.

Предметна область

Проект розроблюється відповідно до варіанту 6, визначеного викладачем. Опис структури полів класу „OfficeWorker“ зображено на рис. 1.

Варіант №6 (HR-2)

Опис структури полів класу «OfficeWorker» (суттєвість 2)

Ім'я поля	Тип даних	Опис	Властивості поля
id	Long	Сурогатний ключ	PK
surname	String	Прізвище	Not null
name	String	Ім'я	Not null
pname	String	По-батькові	Not null
startWork	LocalDate	Дата початку роботи	Not null, «дд.мм.рррр»
endWork	LocalDate	Дата звільнення	«дд.мм.рррр», більша за дату початку не менш ніж 4 місяці
workerCod	String	Корпоративний код співробітника	4 цифри, можуть бути 0 на будь-якій позиції

Рисунок 1 – Структура полів класу „OfficeWorker“

Опис структури полів класу «FamilyState» (суттєвість 2) зображено на рис. 2.

Ім'я поля	Тип даних	Опис	Властивості поля
id	Long	Сурогатний ключ	PK
worker	OfficeWorker	Співробітник	FK до OfficeWorker. Not null. @OneToOne
famState	enum	Сімейний стан	Перелік значень: «Одружений», «Неодружений», «Цивільний шлюб», «Розлучений», «Вдовець»
numChild	int	Кількість дітей	0+
numLittleChild	int	Кількість дітей до 18 років	Не більше значення поля вище
isApartOwner	boolean	Чи є власником житла	true – є апартаменти у власності, false – немає (арендоване)

Рисунок 2 – Структура полів класу „FamilyState“

Хід роботи

1. Завдання 1. Реалізувати вбудовану автентифікацію

1.1 Налаштування проекту для роботи

Для налаштування проекту для роботи треба виконати наступні дії.

1 Так як ми будемо використовувати проект з минулої роботи, потрібність налаштування бази даних відпадає.

2 Додати потрібні нам залежності у pom.xml, див. рис 3.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

Рисунок 3 –Залежності для роботи з spring-security

1.2 Створення та налаштування структури проекту

Для створення та налаштування структури проекту необхідно виконати дії що описані нижче.

1 Перейти у редактор коду та додати пакет: securityconfig згідно рис. 4.

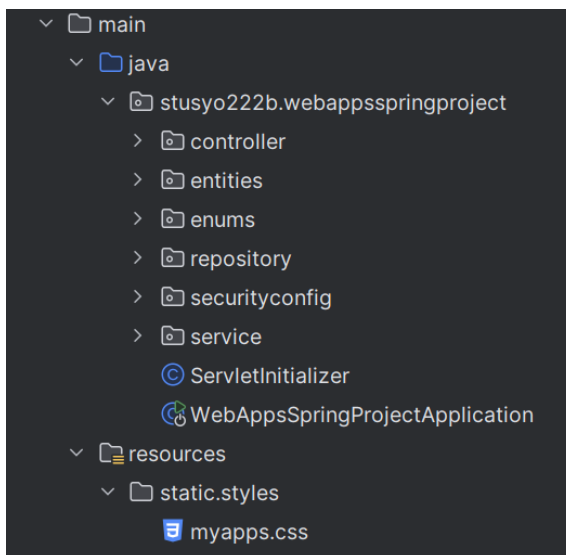


Рисунок 4 – Структура проекту

2 Створити клас CustomUserDetailsServiceConfig (Додаток А):

- налаштувати двох вбудованих користувачів: admin із паролем admin і роллю ADMIN, testuser із паролем testuser і роллю USER;
- реалізувати шифрування паролів через @Bean, що забезпечить безпечне зберігання даних.

3 Створити клас SpringSecurity. Налаштувати принципи авторизації:

- визначити доступ до ресурсів на основі ролі користувача;
- налаштувати форму входу, обробку помилок авторизації, а також функціонал виходу із системи.

1.3 Розробити інтерфейс для авторизації.

1 У папці resources створити папку auth та в ньому html-сторінки, див. рис. 5.

2 Реалізувати контролер AuthController (Додаток Б) для обробки запитів до форми авторизації:

- обробляє запити для відображення сторінок аутентифікації та стартової сторінки;
- забезпечує доступ до списку користувачів через метод users, доступний лише для адміністратора.

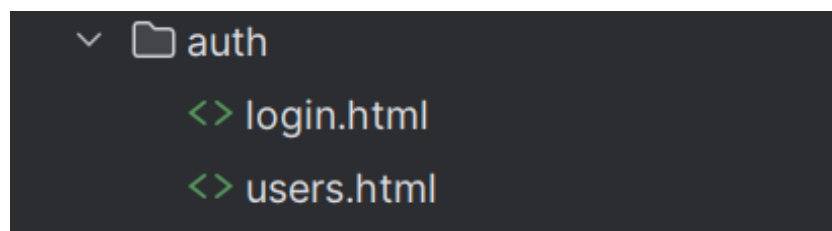
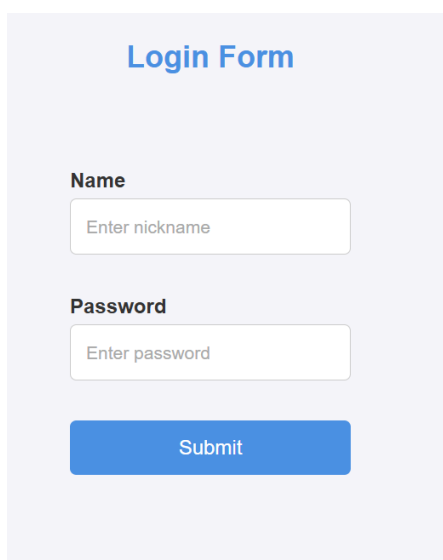


Рисунок 5 – Новостворені html-сторінки

3 В результаті запуску проекту із додатково введеними класами буде на початку представлена сторінка для авторизації (див. рис. 6). На рисунку 7 представлена сторінка у випадку введення помилкового користувача.



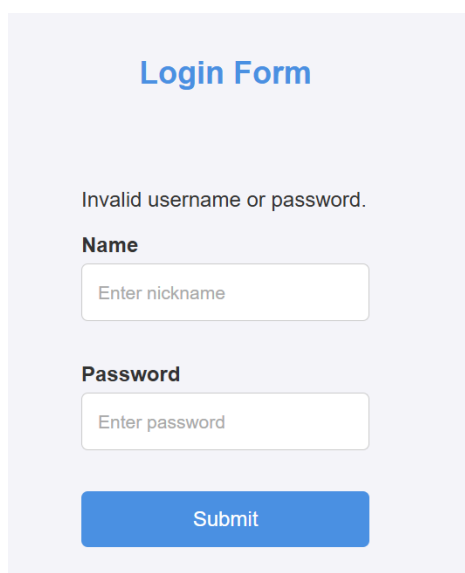
Login Form

Name

Password

Submit

Рисунок 6 – Сторінка для авторизації



Login Form

Invalid username or password.

Name

Password

Submit

Рисунок 7 – Сторінка у випадку введення помилкового користувача

2 Завдання 2

2.1 Відображення даних авторизації

Для відображення даних авторизації користувача, а також для зміни користувача, додати нові компоненти початкової сторінки застосунку WelcomeSpringPage.html (Додаток В).

1 Додати на неї код для відображення імені користувача та його ролі:

– `th:text="${#authentication.name}":` дозволяє отримати ім'я авторизованого користувача;

– `th:text="${#authentication.authorities}"`: відображає список ролей користувача.

2 Додати кнопку для завершення сеансу:

– кнопка LOGOUT: викликає URL `/logout`, що налаштовано у Spring Security для завершення сесії.

2.2 Управління параметрами сесії

Додати в конфігурацію Spring Security код для управління виходом із сесії, а також для управління параметрами сесії (Додаток Г):

– `server.servlet.session.timeout=60s`: задає тривалість сесії бездіяльності;
– `invalidSessionUrl("/login?sessionExpired=true")`: після завершення сесії перенаправляє користувача на сторінку входу;

– `maximumSessions(1)`: обмежує кількість активних сесій для одного користувача;

– `maxSessionsPreventsLogin(true)`: блокує новий вхід, якщо користувач уже має активну сесію.

Вигляд стартової сторінки після входу користувача «adminDB» із роллю адміністратора представлений на рис. 8.

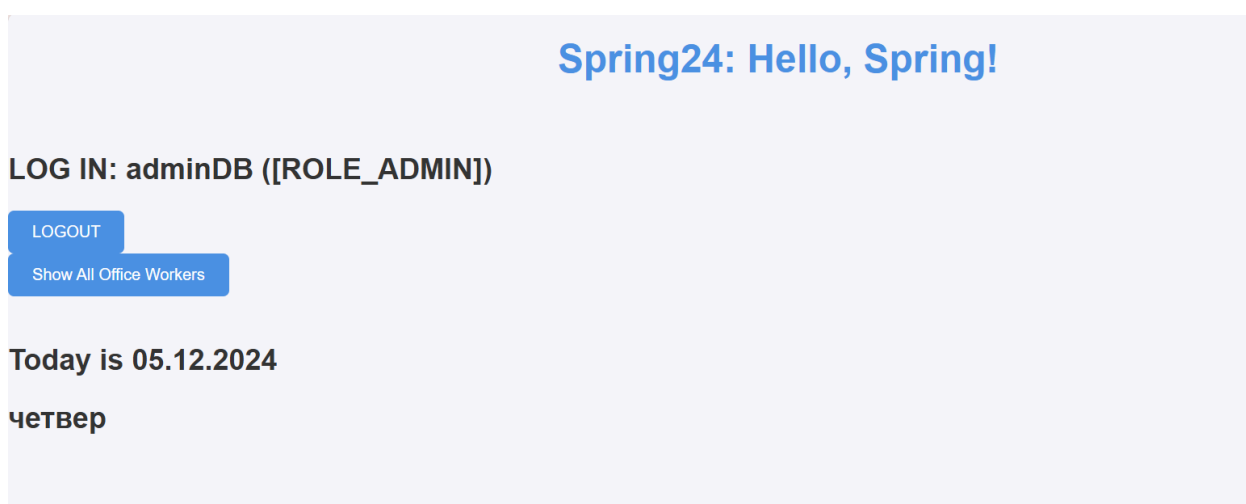
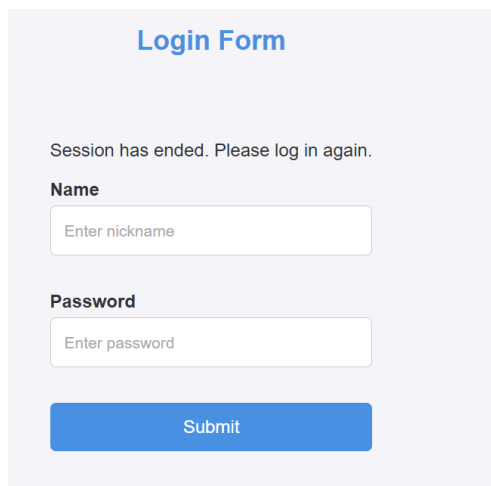


Рисунок 8 – Вигляд стартової сторінки після входу користувача «adminDB» із роллю адміністратора

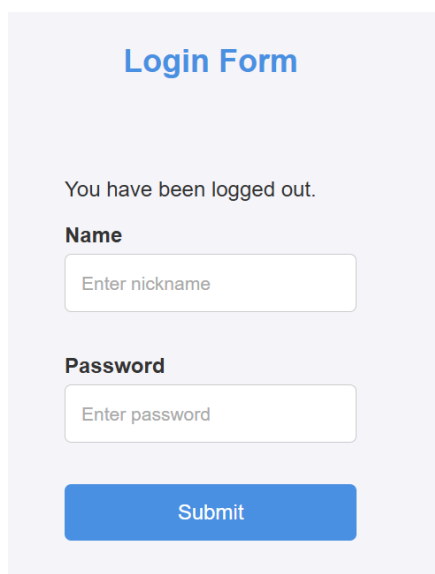
Якщо не виконувати у браузері 60 секунд (тобто встановлений інтервал для сесії), а потім оновити сторінку або виконати будь-яку операцію, то відбудеться перехід на сторінку входу в систему (див. рис. 9).



The image shows a login form titled "Login Form" in blue text. Below the title, a message states "Session has ended. Please log in again." in a small, grey font. The form contains two input fields: "Name" with a placeholder "Enter nickname" and "Password" with a placeholder "Enter password". Both fields have a light yellow background. At the bottom of the form is a blue button labeled "Submit". The entire form is set against a light purple background.

Рисунок 9 – Сторінка авторизації після завершення граничного часу на перебування в системі без дій

Якщо натиснути на кнопку примусового виходу (кнопка «LOGOUT» на сторінці з рис. 8), то здійснюється повернення на сторінку авторизації із відповідним повідомленням (рис. 10).



The image shows a login form titled "Login Form" in blue text. Below the title, a message states "You have been logged out." in a small, grey font. The form contains two input fields: "Name" with a placeholder "Enter nickname" and "Password" with a placeholder "Enter password". Both fields have a light yellow background. At the bottom of the form is a blue button labeled "Submit". The entire form is set against a light purple background.

Рисунок 10 – Стартова сторінка після виходу із сесії

3 Завдання 3. Налаштування авторизації

1 Налаштування авторизації передбачає визначення прав на виклик різних контекстів в залежності від ролі. Ця задача вирішена додаванням до класу `SpringSecurity` анотації `@EnableWebSecurity`.

2 Це передбачає, що контексти, які доступні лише для адміністратора означити анотацією `@PreAuthorize("hasRole('ADMIN')")`.

3 Код контролеру для роботи із екземплярами сутностей `OfficeWorker`, в якості прикладу представлений у додатку Д.

4 Елементи інтерфейсу веб-сторінок, які дозволяють перейти на дані контексти, приховати. Для цього можуть використати теги `SpringSecurity`, які є у складі тегів `Thymeleaf`, зокрема:

- `th:if="${#authorization.expression('hasRole('ADMIN')')}"`;
- `sec:authorize="isAuthenticated()"`.

5 Код сторінки для відображення екземплярів сутності із тегами для реалізації авторизації представлений у додатку Е. Вигляд сторінок із даними, порівняно до реалізації, отриманої раніше, в такому випадку не зміниться, якщо користувач увійде до системи із роллю "ADMIN", а у випадку входу із роллю "USER" сторінки будуть відображаються без кнопок для реалізації маніпулювання даними.

6 Вигляд сторінок із даними про `OfficeWorker`, `FFamilyState` при вході із роллю "USER" представлені на рис.11, 12.

Office Workers							
To START							
Filter by status:		Sort by:		Search by surname:			
All		Surname Start Date		Search...		Reset	
Surname	Name	Patronymic name	Date of start work	Date of end work	Status	WorkerCode	Additional Info
Shevchenko	Andriy	Mykhailovych	11.03.2021		Senior	0005	FAMILY STATE
Onoprienko	Mykyta	Oleksandrovych	20.10.2014		Lead	0001	FAMILY STATE
Stepanenko	Davyd	Olehovych	28.05.2023	14.11.2024	Intern	0953	FAMILY STATE
Borisenko	Volodymyr	Hryhorovych	05.10.2020	28.05.2023	Senior	3723	FAMILY STATE

Рисунок 11 – Сторінка `officeworkers.html` для представлення користувачу із роллю "USER"

The screenshot shows a web form titled 'FAMILY STATE' for a user named 'Shevchenko Andriy Mykhailovich'. The form is for 'OFFICE WORKERS' and contains the following fields:

- Family state:** A dropdown menu with 'Married' selected.
- Number of children:** A text input field containing the number '4'.
- Number of children under the age of 18:** A text input field containing the number '4'.
- Owns an apartment:** A checkbox that is checked.

Рисунок 12 – Сторінка ffamilyState_officeworker.html для представлення користувачу із роллю "USER"

4 Завдання 4. Реалізація автентифікації

Для реалізації автентифікації на основі зберігання відомостей про користувачів та їх ролей у базі даних додати до набору сутностей двох нових: Role та User, реалізувати функції CRUD над відповідними екземплярами цих сутностей, змінити файли конфігурування SpringSecurity: SpringSecurity.java та CustomUserDetailsServiceConfig.java.

4.1 Створення класів юзера та ролей

Для створення класу Юзера потрібно виконати кроки що наведені нижче.

- 1 Створити клас User, що буде описувати структуру юзера.
- 2 Додати у клас важливі імпорти, додати анотації для захисту на рівні БД, а також @ManyToOne @JoinTable з user_roles (див. Додаток К).
- 3 Створити клас Role, що буде описувати структуру ролей.
- 4 Додати у клас важливі імпорти, додати анотації, (див. Додаток Л).
- 5 Створити репозиторії до цих моделей: UserRepository, RoleRepository (див. Додаток М).
- 6 Реалізувати схему, яка передбачає:
 - кожен користувач має тільки одну роль ("ADMIN" або "USER");

– користувачі додаються до бази даних «вручну».

7 Така реалізація дозволяє скоротити час на виконання завдання, тому ще не вимагає витрат на створення інтерфейсу реєстрації користувачів.

8 Після створення набору класів, зазначених вище, запустити застосунок, який створить відповідні таблиці, що, в свою чергу, дозволить створити відповідні ролі та відповідних користувачів, аналогічних «вбудованим» користувачам, створених при імплементації задачі.

9 Зробити це шляхом створення класу-тесту. Варіант такого тесту TestESpringSecurity представлений у додатку Н. Цей тест дозволяє додати до бази користувачів такі дві записи:

– “adminDB” з паролем “adminAAA”, який буде мати роль «ROLE_ADMIN»;

– “testDBuser” з паролем “userAAA”, який буде мати роль «ROLE_USER».

4.2 Створення файлів для конфігурування Spring Security

1 Створити файли для конфігурування Spring Security для роботи із даними для автентифікації, що містяться у базі даних. Це класи CustomUserDetailsServiceConfigDB та SpringSecurityDB. Їх код представлений у додатках П та Р відповідно.

2 Частково код цих файлів співпадає із відповідними файлами налаштування для випадку використання «вбудованих» користувачів (CustomUserDetailsServiceConfig та SpringSecurity), але одночасна присутність цих файлів у проєкті призводить до конфліктів. Тому після створення конфігурації, яка дозволяє здійснювати автентифікацію на основі бази даних, попередні файл налаштувань перенести у архів securityconfig.7z.

3 Після запуску проєкту переконатися, що сторінки після входу із роллю ROLE_USER мають такий самий вигляд як й на рис.11, 12, але дані користувачів зберігаються у таблицях бази даних (рис. 13 та 14). Це означає, що

у майбутньому можна без зміни коду додавати нових користувачів із новими ролями після розробки відповідного інтерфейсу.

4 Також буде працювати перехід на введення паролю після закінчення часу таймауту.

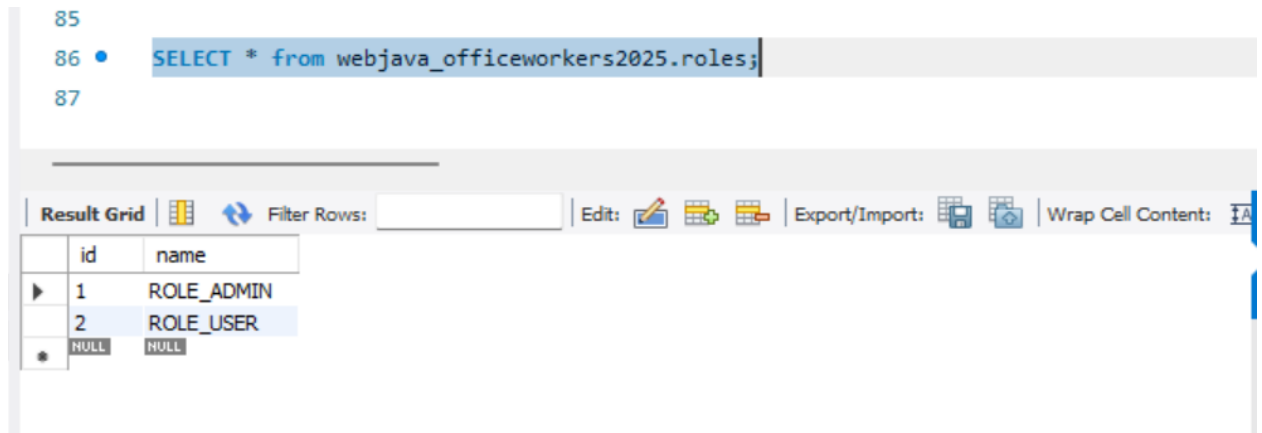


Рисунок 13 – Наповнення таблиці "roles"

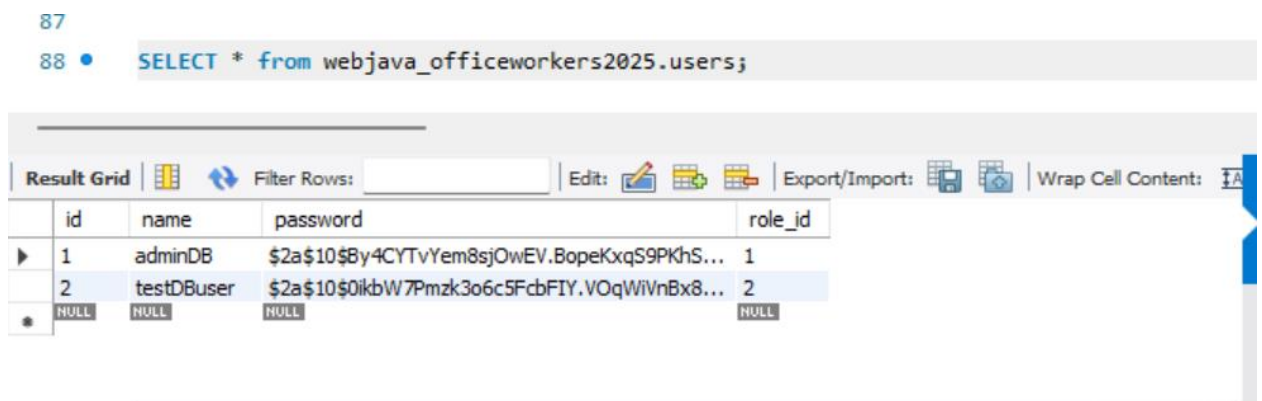


Рисунок 14 – Наповнення таблиці "users"

5 Завдання 5. Реалізація функціоналу відображення зареєстрованих користувачів лише для адміністраторів

Для її реалізації необхідно:

- створити сторінку для відображення користувачів (додаток С);
- додати метод обробки контексту для відображення користувачів (див. рис. 15);

– додати елементи управління для відображення переліку (у даному проєкті - кнопка USERS на сторінці officeworkers.html, рис. 16).

Вигляд сторінки із користувачами, що містяться в БД, представлений на рис. 17.

```
@GetMapping("/users")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String users(Model model) {
    List<User> users = userService.findAllUsers();
    model.addAttribute("users", users);
    return "auth/users";
}
```

Рисунок 15 – Код методу "User"

Office Workers

To START

Filter by status:
All

Sort by:
Surname Start Date

Search by surname:
Search... Reset

INSERT OFFICE WORKER

USERS

Surname	Name	Patronymic name	Date of start work	Date of end work	Status	WorkerCode	Additional Info	Actions
Shevchenko	Andriy	Mykhailovych	11.03.2021		Senior	0005	FAMILY STATE	UPDATE DELETE
Onoprienko	Mykyta	Oleksandrovych	20.10.2014		Lead	0001	FAMILY STATE	UPDATE DELETE
Stepanenko	Davyd	Olehovych	28.05.2023	14.11.2024	Intern	0953	FAMILY STATE	UPDATE DELETE

Рисунок 16 – Сторінка officeworkers.html із доданою кнопкою "USERS"

TO OFFICE WORKERS

List of Registered Users

Nickname	Role
adminDB	ROLE_ADMIN
testDBuser	ROLE_USER

Рисунок 17 – Сторінка users.html

Висновки

Проведення лабораторної роботи з розробки веб-застосунку з використанням Java, Spring Framework та Spring Security стало важливим кроком у професійній підготовці в галузі програмування. Виконання завдання продемонструвало необхідність глибокого розуміння принципів автентифікації та авторизації, які є основою безпеки сучасних веб-застосунків.

Знайомство зі Spring Security виявило значні переваги цієї технології. Зокрема, її модульна структура та готові до використання рішення дозволяють суттєво зменшити обсяг ручного кодування. Налаштування авторизації, реалізація політик доступу до ресурсів та інтеграція з різними джерелами даних стали простішими та ефективнішими. Використання конфігураційного підходу з анотаціями й API Spring спрощує налаштування безпеки, залишаючи більше часу для роботи над бізнес-логікою.

Особливо корисним було дослідження механізмів розмежування доступу до ресурсів залежно від ролей користувачів. Реалізація автоматичного таймауту сесії та обробки виходу з профілю дозволила підвищити зручність використання застосунку та забезпечити належний рівень безпеки. Крім того, завдяки простоті інтеграції Spring Security із інструментами керування користувачами розробка отримала універсальність, що дозволяє легко масштабувати рішення для різних типів застосунків.

Робота підкреслила значення інтеграції теоретичних знань з практичними навичками. Використання Spring Security, який підтримує сучасні стандарти безпеки (OAuth2, JWT, OpenID Connect), дозволяє будувати гнучкі, безпечні й масштабовані системи. Попри початкову складність вивчення технології, розуміння її основ значно розширює компетенції розробника, роблячи його готовим до викликів реальних проєктів.

Таким чином, результати лабораторної роботи засвідчили важливість освоєння Spring Security як одного з базових інструментів для розробки сучасних веб-застосунків. Вони також продемонстрували, що ці знання не

лише підвищують кваліфікацію, а й дають можливість створювати ефективні, безпечні й зручні рішення для користувачів.

ДОДАТОК А

Код класу CustomUserDetailsServiceConfig:

```
package stusyo222b.webappsspringproject.securityconfig;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;

@Configuration
public class CustomUserDetailsServiceConfig {

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails testUser = User.builder()
            .username("testuser")
            .password(passwordEncoder().encode("testuser"))
            .roles("USER")
            .build();

        UserDetails admin = User.builder()
            .username("admin")
            .password(passwordEncoder().encode("admin"))
            .roles("ADMIN")
            .build();

        return new InMemoryUserDetailsManager(testUser, admin);
    }

    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
}
```

ДОДАТОК Б

Код класу AuthController:

```

package stusyo222b.webappsspringproject.controller;

import stusyo222b.webappsspringproject.entities.User;
import stusyo222b.webappsspringproject.service.UserService;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

@Controller
public class AuthController {

    private UserService userService;

    public AuthController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/")
    public String showStartPage() {
        return "WelcomeSpringPage";
    }

    //----- LOGIN
    // handler method to handle login request
    @PostMapping("/login")
    public String login() {
        return "auth/login";
    }

    @GetMapping("/login")
    public String loginWithMessage(@RequestParam(value = "error", required = false) String error,
                                   @RequestParam(value = "logout", required = false) String logout,
                                   @RequestParam(value = "sessionExpired", required = false) String sessionExpired,
                                   Model model) {
        return "auth/login";
    }

    //----- USERS
    // handler method to handle list of users
    @GetMapping("/users")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String users(Model model) {
        List<User> users = userService.findAllUsers();
        model.addAttribute("users", users);
        return "auth/users";
    }
}

```


ДОДАТОК В

Код сторінки WelcomeSpringPage:

```
<!DOCTYPE html>
<html lang="en"
  xmlns:th="http://www.thymeleaf.org"
  xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity6"
>
<style>
  body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    color: #333;
    margin: 0;
    padding: 0;
  }
  h1 {
    text-align: center;
    color: #4a90e2;
    margin: 20px 0;
  }
  a {
    display: inline-block;
    margin: 10px 0;
    padding: 10px 20px;
    background-color: #4a90e2;
    color: white;
    text-decoration: none;
    border-radius: 5px;
    font-size: 16px;
  }
  a:hover {
    background-color: #357ab7;
  }
  .container {
    width: 90%;
    margin: 0 auto;
    padding: 20px;
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
  }
  .title {
    color: red;
    font-weight: bold;
    text-align: center;
    margin-bottom: 20px;
  }
  table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
  }
  th, td {
    padding: 10px;
    text-align: left;
    border: 1px solid #ddd;
  }
}
```

```

th {
  background-color: #4a90e2;
  color: white;
}
tr:nth-child(even) {
  background-color: #f9f9f9;
}
tr:hover {
  background-color: #f1f1f1;
}
input[type="text"], select {
  padding: 8px;
  width: 100%;
  margin-top: 5px;
  margin-bottom: 15px;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
button, input[type="submit"] {
  background-color: #4a90e2;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
button:hover, input[type="submit"]:hover {
  background-color: #357ab7;
}
form {
  display: inline;
}
.filters {
  display: flex;
  justify-content: space-between;
  flex-wrap: wrap;
  margin-bottom: 20px;
}
.filters div {
  flex: 1;
  margin: 10px;
}
@media (max-width: 768px) {
  .filters div {
    flex-basis: 100%;
  }
  table {
    font-size: 14px;
  }
}
</style>
<head>
  <meta charset="UTF-8">
  <title>Spring24 - Welcome!</title>
</head>
<body>
<div class="content">
  <div class="centered">
    <h1>Spring24: Hello, Spring!
    </h1>
    <br>

```

```

<!--      https://www.thymeleaf.org/doc/articles/springsecurity.html-->

<h2 sec:authorize="isAuthenticated()">LOG IN: <span sec:authentication="name"></span>
  (<span sec:authentication="principal.authorities"></span>)
</h2>
<!--      Button LOGOUT      -->
<!-- Important!!!! Mandatory method POST!!!! -->
<form th:action="@{/logout}" method="POST" sec:authorize="isAuthenticated()">
  <button accesskey="d"> LOGOUT </button>
</form>
<br/>
<form action="officeworkers" method="get">
  <button type="submit">Show All Office Workers</button>
</form>
<br><br>
<h2>Today is <span th:text="${#dates.format(#dates.createNow(),'dd.MM.yyyy')}"></span></h2>
<h2><span th:text="${#dates.format(#dates.createNow(),'EEEE')}"></span></h2>
</div>
</div>
</body>
</html>

```

ДОДАТОК Г

Код класу SpringSecurity:

```

package stusyo222b.webappsspringproject.securityconfig;

import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.session.HttpSessionEventPublisher;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SpringSecurity {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authorizeRequests -> authorizeRequests
                .requestMatchers("/login", "/session-expired").permitAll()
                .requestMatchers("/styles/**", "/js/**", "/images/**").permitAll()
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/login")
                .failureUrl("/login?error=true")
                .defaultSuccessUrl("/", true)
                .permitAll()
            )
            .logout(logout -> logout
                .logoutUrl("/logout")
                .logoutSuccessUrl("/login?logout=true")
                .invalidateHttpSession(true)
                .clearAuthentication(true)
                .deleteCookies("JSESSIONID")
                .permitAll()
            )
            .sessionManagement(session -> session
                .invalidSessionUrl("/login?sessionExpired=true")
                .maximumSessions(1)
                .expiredUrl("/login?sessionExpired=true")
            )
        ;
        return http.build();
    }

    @Bean
    public ServletContextInitializer initializer() {

```

```
return servletContext -> {  
    servletContext.getSessionCookieConfig().setMaxAge(60); // Timeout in seconds  
};  
}  
  
@Bean  
public HttpSessionEventPublisher httpSessionEventPublisher() {  
    return new HttpSessionEventPublisher();  
}  
}
```

ДОДАТОК Д

Код класу OfficeWorker:

```

package stusyo222b.webappsspringproject.entities;

import jakarta.persistence.*;
//import jakarta.validation.constraints.Pattern;
//import jakarta.validation.constraints.Size;
import lombok.*;
import org.hibernate.annotations.Check;
import org.springframework.format.annotation.DateTimeFormat;
import stusyo222b.webappsspringproject.enums.OfficeWorkerStatus;
import stusyo222b.webappsspringproject.enums.OfficeWorkerStatusConverter;

import java.time.LocalDate;

@Entity
@Table(name = "officeworkers")
@Check(constraints = "end_work >= start_work + INTERVAL 4 MONTH")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class OfficeWorker {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name="surname", length = 100,nullable = false)
    @Check(constraints = "REGEXP_LIKE(surname, '[A-Z][a-z]+' , 'c')= 1")
    private String surname;

    @Column(name="name", length = 100,nullable = false)
    @Check(constraints = "REGEXP_LIKE(name, '[A-Z][a-z]+' , 'c')= 1")
    private String name;

    @Column(name="pname", length = 100,nullable = false)
    @Check(constraints = "REGEXP_LIKE(pname, '[A-Z][a-z]+' , 'c')= 1")
    private String pname;

    @Column(name = "start_work", nullable = false)
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate startWork;

    @Column(name = "end_work")
    @DateTimeFormat(pattern = "dd.MM.yyyy")
    private LocalDate endWork;

    @Column(name="worker_cod", nullable=false, unique = true)
    //@Size(min = 4, max = 4, message = "Код працівника має складатися з 4 цифр.")
    //@Pattern(regexp = "[0-9]{4}$", message = "Код працівника повинен складатися з 4 цифр.")
    private String workerCod;

```

```
@Enumerated(EnumType.STRING)
@Column(name = "officeworker_status", nullable = false, length = 12)
@Convert(converter = OfficeWorkerStatusConverter.class)
private OfficeWorkerStatus officeWorkerStatus;

// Constructor for create instance for testing
public OfficeWorker(String surname) {
    this.id = null;
    this.surname = surname;
    this.name = name;
    this.pname = pname;
    this.startWork = startWork;
    this.endWork = endWork;
    this.workerCod = workerCod;
    this.officeWorkerStatus = OfficeWorkerStatus.middle;
}
}
```

ДОДАТОК Е

Код сторінки officeworkers.html:

```
<style>
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f9;
  color: #333;
  margin: 0;
  padding: 0;
}
h1 {
  text-align: center;
  color: #4a90e2;
  margin: 20px 0;
}
a {
  display: inline-block;
  margin: 10px 0;
  padding: 10px 20px;
  background-color: #4a90e2;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  font-size: 16px;
}
a:hover {
  background-color: #357ab7;
}
.container {
  width: 90%;
  margin: 0 auto;
  padding: 20px;
  background-color: white;
  border-radius: 8px;
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
}
.title {
  color: red;
  font-weight: bold;
  text-align: center;
  margin-bottom: 20px;
}
table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
}
th, td {
  padding: 10px;
  text-align: left;
  border: 1px solid #ddd;
}
th {
  background-color: #4a90e2;
  color: white;
}
tr:nth-child(even) {
```



```

        background-color: #f9f9f9;
    }
    tr:hover {
        background-color: #f1f1f1;
    }
    input[type="text"], select {
        padding: 8px;
        width: 100%;
        margin-top: 5px;
        margin-bottom: 15px;
        border: 1px solid #ccc;
        border-radius: 4px;
        box-sizing: border-box;
    }
    button, input[type="submit"] {
        background-color: #4a90e2;
        color: white;
        padding: 10px 20px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
    button:hover, input[type="submit"]:hover {
        background-color: #357ab7;
    }
    form {
        display: inline;
    }
    .filters {
        display: flex;
        justify-content: space-between;
        flex-wrap: wrap;
        margin-bottom: 20px;
    }
    .filters div {
        flex: 1;
        margin: 10px;
    }
    @media (max-width: 768px) {
        .filters div {
            flex-basis: 100%;
        }
        table {
            font-size: 14px;
        }
    }
</style>

```

ДОДАТОК К

Код класу User:

```
package stusyo222b.webappsspringproject.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import org.hibernate.annotations.Check;

import java.util.ArrayList;
import java.util.List;

//https://www.javaguides.net/2018/10/user-registration-module-using-springboot-springmvc-springsecurity-
//hibernate5-thymeleaf-mysql.html
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name="users")
public class User
{
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    //nickname!!!!
    @Column(nullable=false)
    @Check(constraints = "REGEXP_LIKE(name, " +
        "'^[a-zA-Z](\\\\\\\\.\\\\\\\\-)?$'[a-zA-Z\\\\\\\\d]{1,3}$','c') = 1")
    private String name;

    //8-20 symbols
    @Column(nullable=false)
    private String password;

    @ManyToOne
    @JoinColumn(name = "role_id")
    private Role role;
}
```

ДОДАТОК Л

Код класу Role:

```
package stusyo222b.webappsspringproject.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name="roles")
public class Role
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable=false, unique=true)
    private String name;
}
```

ДОДАТОК М

Код інтерфейсу RoleRepository

```
package stusyo222b.webappsspringproject.repository;

import stusyo222b.webappsspringproject.entities.Role;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface RoleRepository extends JpaRepository<Role, Long> {
    Role findByName(String name);
}
```

Код інтерфейсу UserRepository

```
package stusyo222b.webappsspringproject.repository;

import stusyo222b.webappsspringproject.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    User findByName(String username);
}
```

ДОДАТОК Н

Код тестового класу testESpringSecurity:

```
package stusyo222b.webappsspringproject;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.ApplicationContext;
import org.springframework.security.crypto.password.PasswordEncoder;
import stusyo222b.webappsspringproject.entities.Role;
import stusyo222b.webappsspringproject.entities.User;
import stusyo222b.webappsspringproject.service.RoleService;
import stusyo222b.webappsspringproject.service.UserService;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotEquals;

@SpringBootTest
class TestESpringSecurity {

    @Autowired
    RoleService roleService;

    @Autowired
    UserService userService;

    @Autowired
    ApplicationContext context;

    @Test
    void contextLoads() {
    }

    @Test
    void insertUsersRolesTest(){
        //===== add roles if its not on db
        Role adminR = roleService.findByName("ROLE_ADMIN");
        if (adminR == null) {
            adminR = new Role(1L, "ROLE_ADMIN");
            roleService.save(adminR);
            adminR = roleService.findByName("ROLE_ADMIN");
        }

        Role userR = roleService.findByName("ROLE_USER");
        if (userR == null) {
            userR = new Role(2L, "ROLE_USER");
            roleService.save(userR);
            userR = roleService.findByName("ROLE_USER");
        }

        assertEquals(2, roleService.findAll().size());

        PasswordEncoder passwordEncoder = context.getBean(PasswordEncoder.class);

        User userAdmin = userService.findByName("adminDB");
        if (userAdmin == null) {
```

```
        userAdmin = new User();
        userAdmin.setName("adminDB");
        userAdmin.setPassword(passwordEncoder.encode("adminAAA"));
        userAdmin.setRole(adminR);
        userService.saveUser(userAdmin);
        userAdmin = userService.findByName("adminDB");
    }

    assertEquals(userAdmin.getRole(), null);

    User user = userService.findByName("testDBuser");
    if (user == null) {
        user = new User();
        user.setName("testDBuser");
        user.setPassword(passwordEncoder.encode("userAAA"));
        user.setRole(userR);
        userService.saveUser(user);
        user = userService.findByName("testDBuser");
    }
    assertEquals(user.getRole(), null);
}
}
```

ДОДАТОК II

Код класы CustomUserDetailServiceConfigDB:

```
package stusyo222b.webappsspringproject.securityconfig;

import org.springframework.context.annotation.Bean;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import stusyo222b.webappsspringproject.entities.User;
import stusyo222b.webappsspringproject.service.UserService;

import java.util.ArrayList;
import java.util.List;

@Service
public class CustomUserDetailsServiceConfigDB implements UserDetailsService {

    private final UserService userService;

    public CustomUserDetailsServiceConfigDB(UserService userService) {
        this.userService = userService;
    }

    // This @Bean is already declared in the configuration file,
    // created for use by "built-in" users (CustomUserDetailServiceConfig class)
    // It must be commented out in one of the places:
    // either here or in CustomUserDetailServiceConfig

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userService.findByName(username);

        if (user == null) {
            throw new UsernameNotFoundException("Invalid username or password.");
        } else {
            List<GrantedAuthority> listAuthorities = new ArrayList<GrantedAuthority>();
            listAuthorities.add(new SimpleGrantedAuthority(user.getRole().getName()));
            return new org.springframework.security.core.userdetails.User(username, user.getPassword(),
                true, true, true, true, listAuthorities);
        }
    }
}
```

ДОДАТОК Р

Код класу SpringSecurityDB:

```
package stusyo222b.webappsspringproject.securityconfig;

import org.springframework.boot.web.servlet.ServletContextInitializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.session.HttpSessionEventPublisher;

@Configuration
@EnableWebSecurity
public class SpringSecurityDB {

    private final CustomUserDetailsServiceConfigDB userDetailsService;

    public SpringSecurityDB(CustomUserDetailsServiceConfigDB userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public AuthenticationManager authManager(AuthenticationConfiguration authConfig) throws Exception {
        return authConfig.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChainDB(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(authorizeRequests -> authorizeRequests
                .requestMatchers("/login", "/session-expired").permitAll()
                .requestMatchers("/styles/**", "/js/**", "/images/**").permitAll()
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/login")
                .failureUrl("/login?error=true")
                .defaultSuccessUrl("/", true)
                .permitAll()
            )
            .logout(logout -> logout
                .logoutUrl("/logout")
                .logoutSuccessUrl("/login?logout=true")
                .invalidateHttpSession(true)
                .clearAuthentication(true)
                .deleteCookies("JSESSIONID")
                .permitAll()
            )
            .sessionManagement(session -> session
                .invalidSessionUrl("/login?sessionExpired=true")
                .maximumSessions(1)
            )
    }
}
```



```
        .expiredUrl("/login?sessionExpired=true")
    )
    ;
    return http.build();
}

@Bean
public ServletContextInitializer initializer() {
    return servletContext -> {
        servletContext.getSessionCookieConfig().setMaxAge(60); // Timeout in seconds
    };
}

@Bean
public HttpSessionEventPublisher httpSessionEventPublisher() {
    return new HttpSessionEventPublisher();
}
}
```

ДОДАТОК С

Код сторінки users.html:

```
<!DOCTYPE html>
<html lang="en"
  xmlns:th="http://www.thymeleaf.org"
>
<style>
  body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    color: #333;
    margin: 0;
    padding: 0;
  }
  h1 {
    text-align: center;
    color: #4a90e2;
    margin: 20px 0;
  }
  a {
    display: inline-block;
    margin: 10px 0;
    padding: 10px 20px;
    background-color: #4a90e2;
    color: white;
    text-decoration: none;
    border-radius: 5px;
    font-size: 16px;
  }
  a:hover {
    background-color: #357ab7;
  }
  .container {
    width: 90%;
    margin: 0 auto;
    padding: 20px;
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
  }
  .title {
    color: red;
    font-weight: bold;
    text-align: center;
    margin-bottom: 20px;
  }
  table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
  }
  th, td {
    padding: 10px;
    text-align: left;
    border: 1px solid #ddd;
  }
}
```

```

th {
  background-color: #4a90e2;
  color: white;
}
tr:nth-child(even) {
  background-color: #f9f9f9;
}
tr:hover {
  background-color: #f1f1f1;
}
input[type="text"], select {
  padding: 8px;
  width: 100%;
  margin-top: 5px;
  margin-bottom: 15px;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}
button, input[type="submit"] {
  background-color: #4a90e2;
  color: white;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
button:hover, input[type="submit"]:hover {
  background-color: #357ab7;
}
form {
  display: inline;
}
.filters {
  display: flex;
  justify-content: space-between;
  flex-wrap: wrap;
  margin-bottom: 20px;
}
.filters div {
  flex: 1;
  margin: 10px;
}
@media (max-width: 768px) {
  .filters div {
    flex-basis: 100%;
  }
  table {
    font-size: 14px;
  }
}
</style>
<head>
  <meta charset="UTF-8">
  <title>Spring24 - Users</title>
</head>
<body>
<div class="content">
  <div>
    <a th:href="@{/officeworkers}">TO OFFICE WORKERS</a>
    <h2>List of Registered Users</h2>
  </div>
  <table>

```

```
<thead>
  <tr>
    <th>Nickname</th>
    <th>Role</th>
  </tr>
</thead>
<tbody>
  <tr th:each="user : ${users}">
    <td th:text="${user.name}"></td>
    <td th:text="${user.role.name}"></td>
  </tr>
</tbody>
</table>
</div>
</body>
</html>
```