

Лабораторная работа №3. Spring Security

Spring Security — это мощный и гибкий фреймворк для обеспечения безопасности в приложениях на основе Spring. Он предоставляет комплексное решение для аутентификации, авторизации, защиты от атак и управления сессиями пользователей.

Основные возможности Spring Security:

1) Аутентификация. Spring Security обеспечивает механизм аутентификации пользователей, который позволяет проверять их идентичность.

2) Авторизация. Фреймворк позволяет определить права доступа для различных ролей пользователей и контролировать доступ к защищенным ресурсам на основе этих прав.

3) Защита от атак. Spring Security предоставляет защиту от распространенных уязвимостей, таких как атаки CSRF (межсайтовая подделка запроса), атаки инъекции, утечки информации и другие. Включает в себя механизмы для предотвращения атак перебора паролей, фильтрацию и валидацию входных данных и другие средства защиты.

4) Управление сессиями. Spring Security позволяет управлять сессиями пользователей. Поддерживаются различные стратегии управления сессиями, такие как фиксированная продолжительность сессии, сессии, основанные на токенах, и другие.

Защита веб-приложения с помощью Spring Security осуществляется путем конфигурации различных слоев безопасности, включая аутентификацию, авторизацию и другие меры защиты. Вот основные шаги, необходимые для защиты веб-приложения с помощью Spring Security:

1) Настройка правил аутентификации. Определите, как пользователи будут аутентифицироваться в приложении. Это может быть базовая аутентификация, форма входа, аутентификация через социальные сети, аутентификация с использованием токенов и т. д. Настройте аутентификацию в соответствии с требованиями вашего приложения.

2) Настройка правил авторизации. Определите, какие ресурсы и функциональность доступны для различных ролей пользователей. Это может включать в себя ограничение доступа к определенным URL-адресам, методам контроллеров, страницам или действиям на основе ролей пользователя.

3) Настройка защиты от атак. Включите защиту от распространенных уязвимостей, таких как атаки CSRF, инъекции, перебора паролей и другие. Настройте фильтры и механизмы защиты в соответствии с рекомендациями Spring Security.

4) Настройка управления сессиями. Определите, как управлять сессиями пользователей, включая создание, управление и завершение сессий. Настройте параметры сессий и стратегии управления сессиями в соответствии с требованиями вашего приложения.

Цель работы: Получить практические навыки работы с Spring Framework.

Пример выполнения работы.

Добавим систему авторизации в приложение из работы №4. Для этого необходимо модифицировать файл с зависимостями, добавив в него зависимость `spring-boot-starter-security`. Полный код `pom.xml` находится в приложении 1.

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

Далее нам необходимо создать класс `SecurityConfig` с настройками конфигурации безопасности:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    @Bean
    public InMemoryUserDetailsManager userDetailsService() {
        UserDetails user = User.withDefaultPasswordEncoder()
            .username("user")
            .password("password")
            .roles("USER")
            .build();
        UserDetails moderator = User.withDefaultPasswordEncoder()
            .username("moderator")
            .password("password")
            .roles("MODERATOR")
            .build();
        return new InMemoryUserDetailsManager(user, moderator);
    }
}
```

Аннотации `@Configuration`, `@EnableWebSecurity` и `@EnableGlobalMethodSecurity(prePostEnabled = true)`: Эти аннотации объявляют класс `SecurityConfig` как конфигурационный для Spring, активируют конфигурацию веб-безопасности и глобальную методическую безопасность с использованием аннотаций `@PreAuthorize`, `@PostAuthorize`, `@Secured` и `@RolesAllowed`.

Метод `userDetailsService()`: Этот метод создает менеджер пользователей в памяти для аутентификации. Он создает двух пользователей:

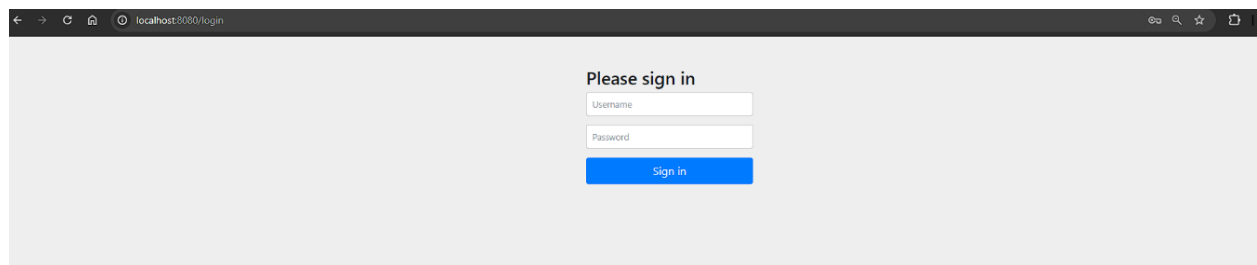
Пользователь "user": С именем "user" и паролем "password" и ролью "USER".

Пользователь "moderator": С именем "moderator", паролем "password" и ролью "MODERATOR".

Каждый пользователь создается с использованием статического метода `User.withDefaultPasswordEncoder()`, который использует стандартное кодирование паролей для сохранения паролей в зашифрованном виде.

Метод `InMemoryUserDetailsManager` будет использоваться Spring Security для аутентификации и авторизации пользователей в приложении.

Попробуем воспользоваться приложением и введем в адресной строке любого браузера `http://localhost:8080/`

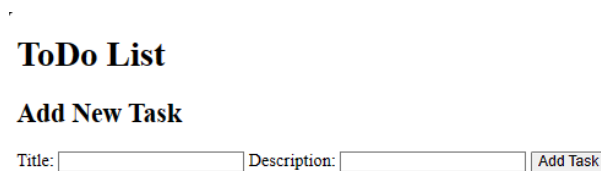


Локальный сервер сразу перенаправляет нас на `http://localhost:8080/login`

Введем логин и пароль указанные в классе с конфигурацией:



Попадаем на страницу с списком задач:



Мы добавили две роли, но на данный момент они имеют одинаковые возможности. Сделаем так, чтобы только MODERATOR мог менять статус задач и удалять записи. Для этого нам нужно модифицировать класс TaskController:

```
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.util.ArrayList;
import java.util.List;
@Controller
public class TaskController {
    private List<Task> taskList = new ArrayList<>();
    @GetMapping("/")
    public String index(Model model) {
        model.addAttribute("tasks", taskList);
        return "index";
    }
    @PostMapping("/addTask")
    public String addTask(@ModelAttribute Task task) {
        taskList.add(task);
        return "redirect:/";
    }
    @PreAuthorize("hasRole('MODERATOR')")
    @GetMapping("/deleteTask/{id}")
    public String deleteTask(@PathVariable Long id) {
        taskList.removeIf(task -> task.getId().equals(id));
        return "redirect:/";
    }
    @PreAuthorize("hasRole('MODERATOR')")
    @PostMapping("/updateTask/{id}")
    public String updateTask(@PathVariable Long id, @RequestParam boolean
        completed) {
        for (Task task : taskList) {
            if (task.getId().equals(id)) {
                task.setCompleted(completed);
                break;
            }
        }
    }
}
```

```

    }
    }
    return "redirect:/";
}
}

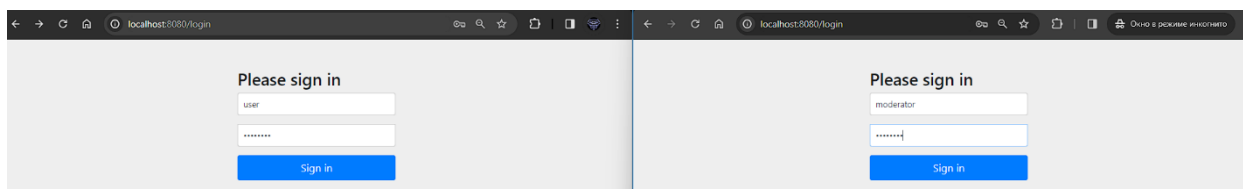
```

Отличие заключается в том, что мы импортировали библиотеку `org.springframework.security.access.prepost.PreAuthorize`;

И добавили аннотацию `@PreAuthorize("hasRole('MODERATOR')")` для определения правил доступа к методам контроллера на основе предварительной аутентификации и авторизации. В данном случае, `hasRole('MODERATOR')` проверяет, имеет ли текущий аутентифицированный пользователь роль "MODERATOR".

Подход `@PreAuthorize` позволяет определять правила доступа к методам на уровне кода. В примере, который вы предоставили, методы `deleteTask` и `updateTask` будут доступны только пользователям с ролью "MODERATOR". Если пользователь не имеет этой роли, при попытке доступа к этим методам будет сгенерировано исключение.

Запускаем приложение. Авторизуемся в одном браузере как USER, в другом(или во вкладке в режиме инкогнито) как MODERATOR:



Добавляем за пользователя несколько задач:

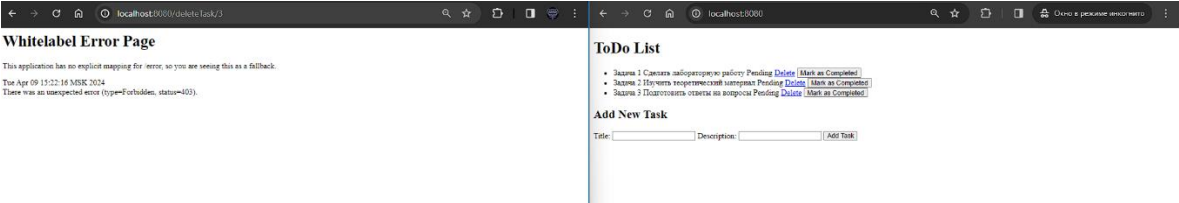
ToDo List

- Задача 1 Сделать лабораторную работу Pending [Delete](#) [Mark as Completed](#)
- Задача 2 Изучить теоретический материал Pending [Delete](#) [Mark as Completed](#)
- Задача 3 Подготовить ответы на вопросы Pending [Delete](#) [Mark as Completed](#)

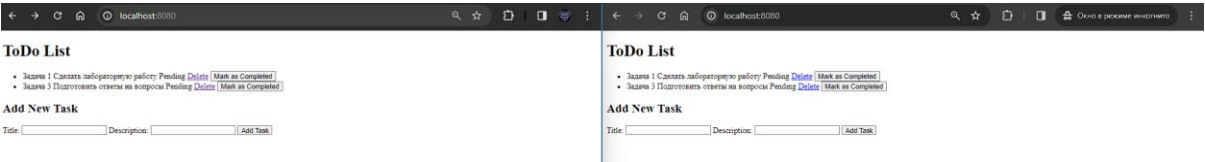
Add New Task

Title: Description: [Add Task](#)

Потом пробуем их удалить за пользователя:



Получаем ошибку. Удаляем запись за модератора:



Обновляем за пользователя страницу и убеждаемся в том, что всё работает.

Задание:

1. Изучить теоретический материал;
2. Написать приложение следуя примеру;
3. Выполнить дополнительное задание;
4. Сделать отчет.

Дополнительное задание.

Добавьте еще одну или две роли, которые будут использовать разработанный вами функционал из прошлой лабораторной работы.

Приложение 1

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.2.3</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>21</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

