Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра «Программной инженерии»
кафедра

Практическая работа 6

Spring Security
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата инициалы, фамилия

Студент ЗКИ21-16б _____ Л. Д. Чумаченко
номер группы подпись, дата инициалы, фамилия

Красноярск 2025

СОДЕРЖАНИЕ

1 Задание

Изменить практическую работу №5, добавив следующий функционал:

Добавить простейшую страницу регистрации. Пользователь вводит свои логин и пароль и данная информация вносится в базу данных, пользователю присваивается роль пользователя (*User*) приложения.

Добавить простейшую форму аутентификации. Форма создается программно, а не автоматически генерируется *Spring*.

В приложении должен быть предусмотрен пользователь — администратор (*Admin*) с ролью отличной, от *User*.

Разграничить уровни доступа к страницам приложения. Пользователь (*User*) имеет доступ только к страницам просмотра всех записей и запросов. Администратор (*Admin*) имеет возможность добавлять, редактировать и удалять записи.

Информация о пользователях и их ролях должна храниться в базе данных. Способ хранения — на усмотрение студента.

Предусмотреть возможность выхода из приложения (*logout*).

Продемонстрировать умение настраивать безопасность на уровне представлений. Для этого реализуется приветствие пользователя после его входа и отображение элемента на основе его роли.

2 Описание варианта задания

Вариант 14. Сущность данного варианта – парфюмерия.

3 Ход выполнения

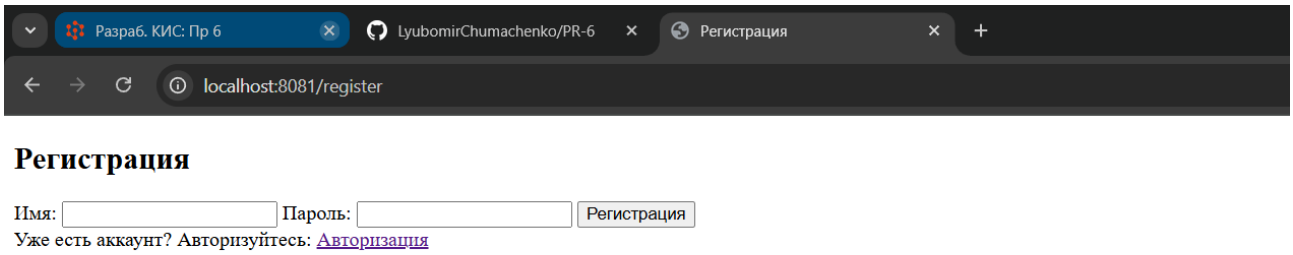
Создаём дополнительные страницы для Входа и Регистрации

Листинг 1 – Шаблон страницы - Регистрация

```
<!DOCTYPE html>
<html>
<head>
    <title>Регистрация</title>
```

```
</head>
<body>
    <h2>Регистрация</h2>
    <form action="/register" method="post">
        <label>Имя:</label>
        <input type="text" name="username" required>
        <label>Пароль:</label>
        <input type="password" name="password" required>
        <button type="submit">Регистрация</button>
    </form>
    <div>Уже есть аккаунт? Авторизуйтесь: <a href="/login">Авторизация</a></div>
    <div style="color:red;">
        <p th:if="${error}" th:text="${error}"></p>
    </div>
</body>
</html>
```

Результат выполнения:

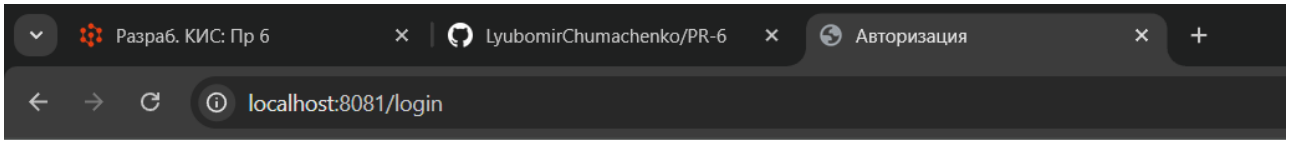


Листинг 2 – Шаблон страницы - Авторизация

```
<!DOCTYPE html>
<html>
<head>
    <title>Авторизация</title>
</head>
<body>
    <h2>Авторизация</h2>
    <form action="/login" method="post">
        <label>Имя:</label>
        <input type="text" name="username" required>
        <label>Пароль:</label>
        <input type="password" name="password" required>
        <button type="submit">Login</button>
    </form>
```

```
    <div>Нет аккаунта? Зарегистрируйтесь: <a
href="/register">Регистрация</a></div>
</body>
</html>
```

Результат выполнения:



Авторизация

Имя: Пароль: Login

Нет аккаунта? Зарегистрируйтесь: Регистрация

Листинг 3 – Класс AdminConfig, который автоматически настраивает роли в системе при запуске приложения

```
@Configuration
public class AdminConfig {

    private final RoleRepository roleRepository;
    private final UserService userService;
    private final PasswordEncoder passwordEncoder;
    private final String adminPassword;

    @Autowired
    public AdminConfig(RoleRepository roleRepository, UserService userService,
PasswordEncoder passwordEncoder,
                       @Value("${app.admin.password}") String adminPassword) {
        this.roleRepository = roleRepository;
        this.userService = userService;
        this.passwordEncoder = passwordEncoder;
        this.adminPassword = adminPassword;
    }

    @PostConstruct
    public void init() {
        // Создаем роли с префиксом ROLE_ для корректной работы с Spring
Security
        Role userRole = roleRepository.findByName("ROLE_USER")
                .orElseGet(() -> roleRepository.save(new Role("ROLE_USER")));

        Role adminRole = roleRepository.findByName("ROLE_ADMIN")
                .orElseGet(() -> roleRepository.save(new Role("ROLE_ADMIN")));

        // Проверяем существование администратора и обновляем его при
необходимости
        Optional<User> existingAdmin = userService.findUserByUsername("admin");
```

```
        if (existingAdmin.isPresent()) {
            User admin = existingAdmin.get();

            // Проверяем, имеет ли администратор роль ADMIN
            boolean hasAdminRole = admin.getRoles().stream()
                .anyMatch(role -> role.getName().equals("ROLE_ADMIN"));

            if (!hasAdminRole) {
                admin.addRole(adminRole);
            }

            // Проверяем, совпадает ли текущий пароль с установленным
            if (!passwordEncoder.matches(adminPassword, admin.getPassword())) {
                admin.setPassword(passwordEncoder.encode(adminPassword));
            }

            userService.saveUser(admin);
        } else {
            // Создаем нового администратора, если он не существует
            User admin = new User();
            admin.setUsername("admin");
            admin.setPassword(passwordEncoder.encode(adminPassword));
            admin.addRole(adminRole);
            // Опционально можно добавить роль USER администратору
            admin.addRole(userRole);
            userService.saveUser(admin);
        }
    }
}
```

Листинг 4 – Класс SecurityConfig, который настраивает правила авторизации и аутентификации в веб-приложении

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private final CustomUserDetailsService customUserDetailsService;

    public SecurityConfig(CustomUserDetailsService customUserDetailsService) {
        this.customUserDetailsService = customUserDetailsService;
    }

    @Bean
    public AuthenticationSuccessHandler appAuthenticationSuccessHandler() {
        return new AppAuthenticationSuccessHandler();
    }
```

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

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .authorizeHttpRequests((authorize) -> authorize

                .antMatchers("/login*", "/register*").permitAll()
                // Добавляем правила для администраторских URL
                .antMatchers("/parfume/new", "/parfume/search").hasRole("ADMIN")
                // URL доступные всем авторизованным пользователям
                .antMatchers("/parfume").authenticated()
                .anyRequest().authenticated()


            )
            .logout((logout) -> logout
                .logoutUrl("/logout")
                .logoutSuccessUrl("/login?logout")
                .invalidateHttpSession(true)
                .deleteCookies("JSESSIONID")
            )
            .formLogin((formLogin) -> formLogin
                .loginPage("/login")
                .defaultSuccessUrl("/", true) // Перенаправление на главную
страницу после успешного входа
                .permitAll()
            )
            .csrf((csrf) -> csrf.disable())
            .userDetailsService(customUserDetailsService);

        return http.build();
    }
}
```

Листинг 5 – Класс-сущность User. Он позволяет:

- Хранить информацию о пользователях в базе данных
- Связывать пользователей с их ролями для реализации разграничения доступа
- Интегрироваться с механизмами безопасности Spring Security через интерфейс UserDetails

```
@Entity
@Table(name = "users")
public class User implements UserDetails {
```

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

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "role_id")
    )
    private Set<Role> roles = new HashSet<>();

    public User() {}

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }


    public Long getId() {
        return id;
    }


    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Set<GrantedAuthority> authorities = new HashSet<>();
        for (Role role : roles) {
            authorities.add(new SimpleGrantedAuthority("ROLE_" +
role.getName()));
        }
        return authorities;
    }

    @Override
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    @Override
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
```

```
        this.password = password;
    }

    public Set<Role> getRoles() {
        return roles;
    }

    public void setRoles(Set<Role> roles) {
        this.roles = roles;
    }

    public void addRole(Role role) {
        this.roles.add(role);
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

Листинг 6 – Класс-сущность Role. Он работает вместе с классом User и используется в ранее рассмотренных классах AdminConfig (для инициализации ролей) и SecurityConfig (для проверки прав доступа)

```
@Entity
@Table(name = "roles")
public class Role {

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

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

    @ManyToMany(mappedBy = "roles")
    private Set<User> users = new HashSet<>();
```

```
    public Role() {}

    public Role(String name) {
        this.name = name;
    }

    public Long getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Set<User> getUsers() {
        return users;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }
}
```

Листинг 7 – Интерфейсы UserRepository и RoleRepository:

1. **Обеспечивают доступ к базе данных** для сущностей User и Role без написания SQL-запросов, используя возможности Spring Data JPA.
2. **Реализуют паттерн "Репозиторий"**, изолируя бизнес-логику от деталей хранения данных.
3. **Обслуживают систему безопасности**: UserRepository находит пользователей для аутентификации, а RoleRepository обеспечивает доступ к ролям для авторизации.
4. **Предоставляют готовые методы CRUD** и возможность создавать специализированные запросы (например, findByUsername).

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

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
}
```

Листинг 7 – Представление элементов интерфейса для роли «ADMIN» и роли «USER» на главной странице

```
<!DOCTYPE html>
<html lang="ru" >
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Главная страница</title>
</head>
<body>
<h1>Магазин парфюмерии</h1>
<h1 sec:authorize="hasRole('ADMIN')">Ваша роль: Администратор</h1>
<h1 sec:authorize="hasRole('USER')">Ваша роль: Обычный пользователь</h1>
    <ul>
```

```
        <li sec:authorize="hasRole('ADMIN')"> <a href="/parfume/new">1. Добавить
парфюм</a></li>
        <li ><a href="/parfume">2. Просмотр всего парфюма</a></li>
        <li sec:authorize="hasRole('ADMIN')"><a href="/parfume/search"> 3. Поиск
парфюма по типу</a></li>
        <li><a href="/logout">4. Выход</a></li>
    </ul>
</body>
</html>
```

4 Результат выполнения работы

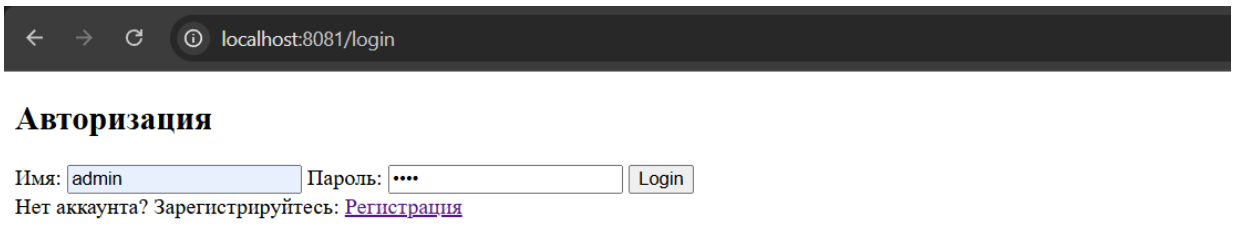
- Созданы страницы Авторизации и Регистрации
- Определены классы для работы с ролями пользователей и интерфейсы для хранения их в базе данных
- Реализовано визуальное представление (показ или скрытие элементов страницы) в зависимости от роли пользователя
- Также реализована возможность выхода из приложения и возможность запуска приложения из командной строки через jar файл

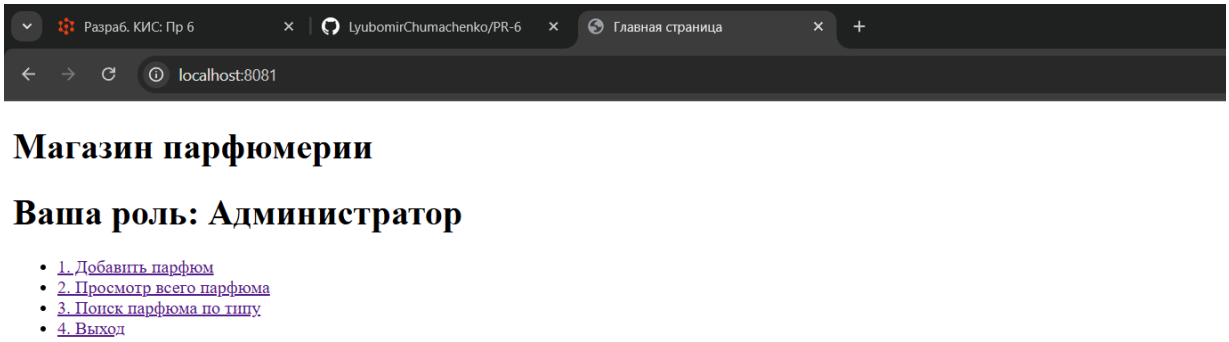
Визуальная демонстрация работы программы:

-запуск программы через консоль

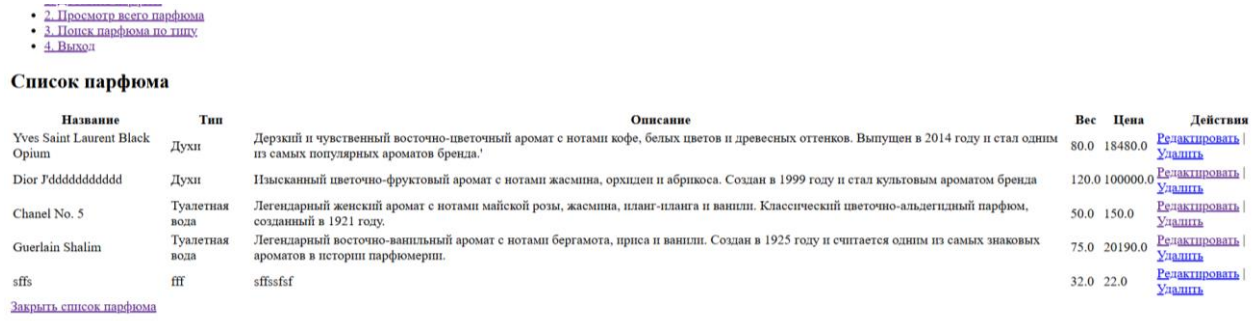
```
1. Добавить парфюм
2. Показать все позиции
3. Редактировать по ID
4. Удалить по ID
5. Искать по типу
6. Выход
http://localhost:8081/index.html
Данные для входа в аккаунт Администратора:
Логин: admin
Пароль: 0890
```

-переходим по ссылке и входим через пользователя с ролью ADMIN

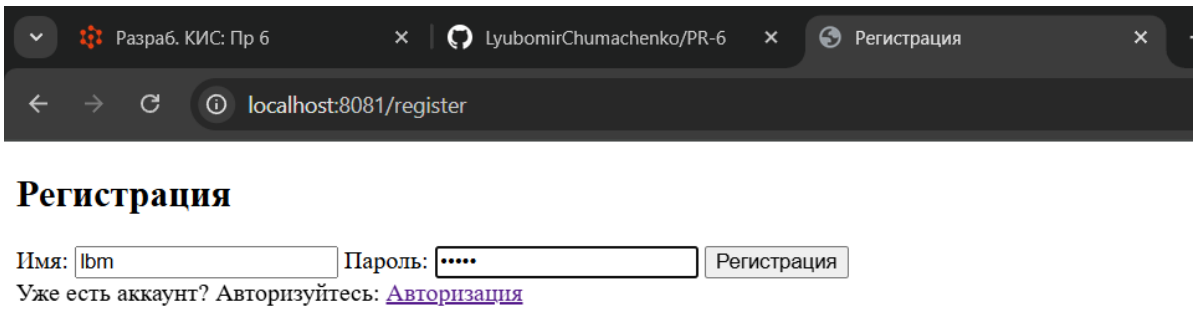


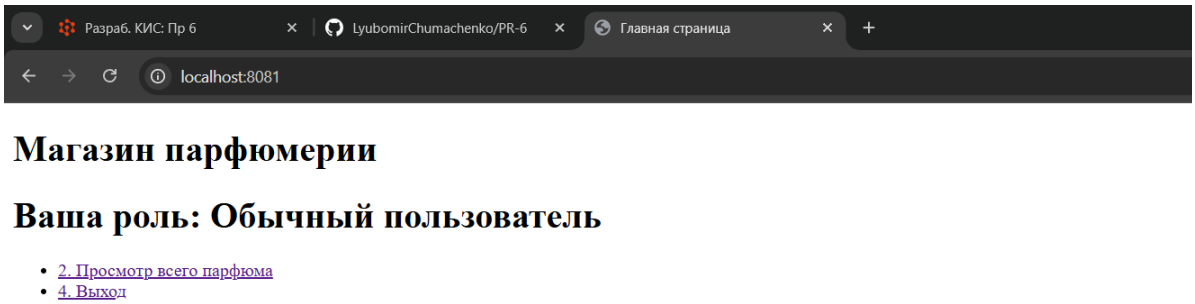


-с текущей ролью нам доступны все пункты меню, а также возможность удалять и редактировать записи о парфюме из БД



-теперь выберем 6 пункт меню и выйдем из приложения, создадим нового пользователя с ролью USER





-сейчас мы авторизовались с ролью USER поэтому нам доступны только 2 пункта меню, а во втором пункте меню мы теперь не можем редактировать и удалять записи о парфюме из БД