УДК 004.056

В. Л. Поліщук
І. В. Богач

РЕАЛІЗАЦІЯ АВТОРИЗАЦІЇ HTTP-ЗАПИТІВ ДО REST-ЕНДПОЙНТІВ ЗА ДОПОМОГОЮ SPRING SECURITY DSL

Вінницький національний технічний університет

Анотація

Розроблено конфігураційний модуль авторизації HTTP-запитів для REST API з використанням Spring Security DSL. Досліджено особливості побудови ланцюга фільтрів безпеки, реалізації безстанової аутентифікації за допомогою JWT та застосування комбінованого підходу на основі ролей і привілеїв. Також розглянуто механізми маршрутизації доступу через MvcRequestMatcher та обробку винятків при несанкціонованому доступі. Проведене модульне тестування підтвердило коректність авторизації запитів і стабільну роботу всієї конфігурації безпеки.

Ключові слова: Spring Security DSL, JWT, SecurityFilterChain, безстановна аутентифікація, MvcRequestMatcher, ролі, привілеї, REST API.

Abstract

A configuration module for authorizing HTTP requests to a REST API was developed using the Spring Security DSL. The specifics of constructing the security filter chain, implementing stateless authentication with JWT, and applying a combined role-and-privilege-based approach were investigated. Mechanisms for routing access via MvcRequestMatcher and handling exceptions for unauthorized access were also examined. Module testing confirmed the correctness of request authorization and the stable operation of the entire security configuration.

Keywords: Spring Security DSL, JWT, SecurityFilterChain, stateless authentication, MvcRequestMatcher, roles, privileges, REST API.

Вступ

У сучасних веб-застосунках безпечна організація доступу до ресурсів залишається ключовим елементом довіри користувачів та стійкості бізнес-процесів. Надійні механізми автентифікації та авторизації покликані запобігати несанкціонованому доступу й забезпечувати коректне розмежування прав між різними групами користувачів. Використання спеціалізованих інструментів, таких як Spring Security DSL у поєднанні з токен-орієнтованими підходами (наприклад, JWT), дозволяє створювати гнучкі та масштабовані рішення для захисту REST-інтерфейсів і внутрішніх компонентів застосунку, що є особливо актуальним в умовах швидкого розвитку веб-технологій та збільшення кіберзагроз.

У даній роботі досліджується підхід реалізації конфігурації безпеки, зосереджений на визначенні правил доступу до ендпойнтів за допомогою Spring Security DSL. Особливу увагу приділено використанню MvcRequestMatcher, поєднанню ролей та привілеїв, а також побудові ланцюгу фільтрів зі безстановими сесіями та обробкою помилок.

Результати дослідження

У результаті дослідження було реалізовано конфігураційний клас *SecurityConfig* для авторизації REST API. Розроблений модуль довів гнучкість розмежування доступу до ресурсів та доцільність застосування Spring Security для забезпечення більш високого рівня безпеки застосунку.

Клас забезпечує централізовану побудову ланцюгу фільтрів, що включає відключення CSRF-захисту для безстанового режиму, налаштування політики сесій як STATELESS та інтеграцію кастомних компонентів — JWT-фільтра, провайдера аутентифікації, обробника прострочених токенів і механізму логауту.

Рис. 1. Фрагмент налаштування компонентів авторизації у Spring Security

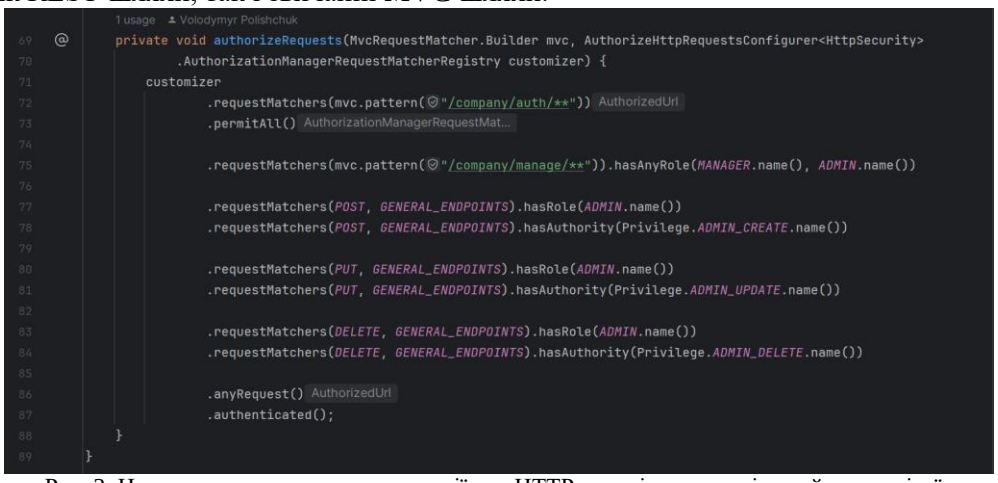
Клас *SecurityConfig* позначено анотаціями *@Configuration* та *@EnableWebSecurity*, що перетворюють його на джерело Spring компонентів безпеки та застосовує модуль захисту HTTP-запитів[1]. Конструктор класу отримує чотири ключові компоненти: провайдер аутентифікації *AuthenticationProvider*, фільтр для перевірки JWT у запитах *JwtAuthenticationFilter*, обробник протермінованих токенів *ExpiredTokenHandler* та обробник виходу з системи *LogoutHandler*.

Першим кроком у методі *secure(HttpSecurity http, MvcRequestMatcher.Builder mvc)* є відключення CSRF-захисту, оскільки застосунок працює в режимі Stateless і не використовує cookies для автентифікації[2]. Далі за допомогою лямбда-функції конфігурується розподіл прав доступу: виклики *customizer.requestMatchers(mvc.pattern(...)).permitAll()* дозволяють відкритий доступ до шляху /company/auth/**, що використовується для реєстрації та автентифікації користувачів.

Рис. 2. Налаштування автентифікації, авторизації та обробки винятків

Далі застосовується патерн розмежування доступу відповідно до ролей та привілеїв. За висхідною складністю, спочатку визначено, що виклики до /company/manage/** доступні лише користувачам з ролями MANAGER або ADMIN. Потім за допомогою перевизначення HTTP-методів POST, PUT і DELETE накладаються додаткові вимоги: для кожного набору загальних ресурсів *GENERAL_ENDPOINTS* операції створення (POST), оновлення (PUT) та видалення (DELETE) можуть виконувати лише користувачі з роллю ADMIN, причому на рівні привілеїв додатково перевіряється наявність відповідного *Privilege.ADMIN_CREATE*, *Privilege.ADMIN_UPDATE* або *Privilege.ADMIN_DELETE*. Останній виклик *.anyRequest().authenticated()* гарантує, що всі інші запити потребують просто аутентифікації, незалежно від ролей [3].

Побудова запитів для перевірки відповідності URI здійснюється через компонент *MvcRequestMatcher.Builder*, що дозволяє використовувати переваги Spring MVC (такі, як інтерполяцію патернів, врахування контекстного шляху) у фільтрах безпеки. Це дає змогу єдиним інтерфейсом описувати як REST-шляхи, так і звичайні MVC-шляхи.



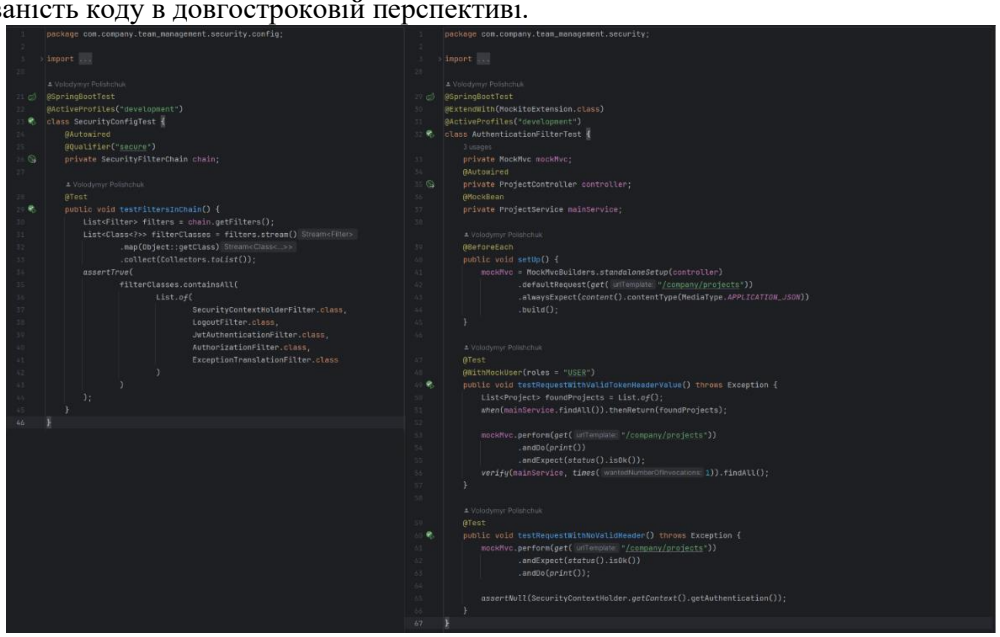
```
1 usage  Volodymyr Polishchuk
private void authorizeRequests(MvcRequestMatcher.Builder mvc, AuthorizeHttpRequestsConfigurer<HttpSecurity>
        .AuthorizationManagerRequestMatcherRegistry customizer) {
    customizer
            .requestMatchers(mvc.pattern("/company/auth/**")) AuthorizedUrl
            .permitAll() AuthorizationManagerRequestMat...

            .requestMatchers(mvc.pattern("/company/manage/**")).hasAnyRole(MANAGER.name(), ADMIN.name())

            .requestMatchers(POST, GENERAL_ENDPOINTS).hasRole(ADMIN.name())
            .requestMatchers(POST, GENERAL_ENDPOINTS).hasAuthority(Privilege.ADMIN_CREATE.name())

            .requestMatchers(PUT, GENERAL_ENDPOINTS).hasRole(ADMIN.name())
            .requestMatchers(PUT, GENERAL_ENDPOINTS).hasAuthority(Privilege.ADMIN_UPDATE.name())

            .requestMatchers(DELETE, GENERAL_ENDPOINTS).hasRole(ADMIN.name())
            .requestMatchers(DELETE, GENERAL_ENDPOINTS).hasAuthority(Privilege.ADMIN_DELETE.name())

            .anyRequest() AuthorizedUrl
            .authenticated();
    }
}
```

Рис. 3. Налаштування правил авторизації для HTTP-запитів на основі ролей та привілеїв

У розділі сесійної політики (sessionManagement) встановлено *SessionCreationPolicy.STATELESS*, що підкреслює безсесійний характер аутентифікації: кожен HTTP-запит містить у заголовках JWT, який перевіряється фільтром *JwtAuthenticationFilter* перед обробкою контролером. Сам фільтр додається перед стандартним *UsernamePasswordAuthenticationFilter*, щоб перехоплювати та валідовувати токен ще до ініціалізації аутентифікації Spring Security.

Обробка невдалих спроб доступу реалізована через *.exceptionHandling(config -> config.authenticationEntryPoint(expiredTokenHandler))*. У разі виконання запиту із простроченим токеном або без нього, *ExpiredTokenHandler* формує відповідь з необхідним HTTP-статусом (зазвичай 401) та повідомленням про помилку[4].

Нарешті, налаштування виходу з системи визначає URL-ендпойнт /company/logout, обробника виходу *logoutHandler* та очищення контексту безпеки за допомогою *SecurityContextHolder.clearContext()*, що забезпечує надійне видалення всіх залишкових даних авторизації. Таким чином, конструкція *SecurityFilterChain* у *SecurityConfig* демонструє сучасний патерн реалізації авторизації HTTP-запитів у Spring Boot: відключення стану сесій, перевірка JWT-токенів на ранніх етапах, гнучка маршрутизація прав доступу за допомогою Spring Security DSL та чітка обробка помилок і виходу з системи. Подібний підхід гарантує високу безпеку, масштабованість і підтримуваність коду в довгостроковій перспективі.



```
package com.company.team_management.security.config;

import ...

Volodymyr Polishchuk
@SpringBootTest
@ActiveProfiles("development")
class SecurityConfigTest {
    @Autowired
    @Qualifier("secure")
    private SecurityFilterChain chain;

    Volodymyr Polishchuk
    @Test
    public void testFiltersInChain() {
        List<Filter> filters = chain.getFilters();
        List<Class<?>> filterClasses = filters.stream() Stream<Filter>
                .map(Object::getClass) Stream<Class<...>>
                .collect(Collectors.toList());
        assertTrue(
                filterClasses.containsAll(
                        List.of(
                                SecurityContextHolderFilter.class,
                                LogoutFilter.class,
                                JwtAuthenticationFilter.class,
                                AuthorizationFilter.class,
                                ExceptionTranslationFilter.class
                        )
                )
        );
    }
}
```

```
package com.company.team_management.security;

import ...

Volodymyr Polishchuk
@SpringBootTest
@ExtendWith(MockitoExtension.class)
@ActiveProfiles("development")
class AuthenticationFilterTest {
    3 usages
    private MockMvc mockMvc;
    @Autowired
    private ProjectController controller;
    @MockBean
    private ProjectService mainService;

    Volodymyr Polishchuk
    @BeforeEach
    public void setUp() {
        mockMvc = MockMvcBuilders.standaloneSetup(controller)
                .defaultRequest(get( urlTemplate: "/company/projects"))
                .alwaysExpect(content().contentType(MediaType.APPLICATION_JSON))
                .build();
    }

    Volodymyr Polishchuk
    @Test
    @WithMockUser(roles = "USER")
    public void testRequestWithValidTokenHeaderValue() throws Exception {
        List<Project> foundProjects = List.of();
        when(mainService.findAll()).thenReturn(foundProjects);

        mockMvc.perform(get( urlTemplate: "/company/projects"))
                .andDo(print())
                .andExpect(status().isOk());
        verify(mainService, times( wantedNumberOfInvocations: 1)).findAll();
    }

    Volodymyr Polishchuk
    @Test
    public void testRequestWithNoValidHeader() throws Exception {
        mockMvc.perform(get( urlTemplate: "/company/projects"))
                .andExpect(status().isOk())
                .andDo(print());

        assertNull(SecurityContextHolder.getContext().getAuthentication());
    }
}
```

Рис. 4. Тестування модуля безпеки

Розроблений модуль було протестовано для відповідності вимогам безпеки. Було реалізовано модульні тести за допомогою JUnit5, що дозволили перевірити правильність обробки HTTP-запитів та розмежування доступу до ресурсів. Додатково проведено тестування правильності конфігурації ланцюгу фільтрів безпеки, що забезпечило перевірку окремих компонентів та загальну коректність роботи модулю безпеки.

Висновки

У результаті дослідження було реалізовано ефективний модуль авторизації HTTP-методів для REST API на основі Spring Security. Використання Spring Security DSL, дозволило гнучко розширити стандартні механізми безпеки, забезпечуючи гранульоване керування доступом на основі ролей та привілеїв до різних ресурсів додатку. Такий підхід є обґрунтованим і ефективним, оскільки гарантує належний рівень безпеки, забезпечує можливість масштабування та дозволяє гнучко керувати доступом у сучасних веб-додатках.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Spring Security Reference Documentation – Spring MVC Integration [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-security/reference/servlet/integrations/mvc.html (дата звернення: 26.05.2025).
2. Spring Security Architecture. Spring Documentation [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-security/reference/servlet/architecture.html#servlet-securityfilterchain (дата звернення: 26.05.2025).
3. Spring Security Reference Documentation – Authorize HttpServletRequests [Електронний ресурс] – Режим доступу: https://docs.spring.io/spring-security/reference/servlet/authorization/authorize-http-requests.html (дата звернення: 26.05.2025).
4. Dikanski A., Steinegger R., Abeck S. Identification and Implementation of Authentication and Authorization Patterns in the Spring Security Framework [Електронний ресурс] // ResearchGate. – 2012. – Режим доступу: https://www.researchgate.net/publication/289782166_Identification_and_implementation_of_authentication_and_authorization_patterns_in_the_spring_security_framework (дата звернення: 26.05.2025).

Поліщук Володимир Леонідович – студент групи 1КН-21Б, кафедра комп'ютерних наук, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: volodimirpolishchuck@gmail.com

Богач Ілона Віталіївна – к.т.н., доцент кафедри автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних технологій та автоматизації, Вінницький національний технічний університет, м.Вінниця, e-mail: ilona.bogach@gmail.com

Polishchuk Volodymyr Leonidovich – student of 1KN-21b group, Department of Computer Science, Faculty of Intelligent Information Technology and Automation, Vinnytsia National Technical University, Vinnytsia, e-mail: volodimirpolishchuck@gmail.com

Bogach Ilona Vitaliivna – Associate Professor of Automation and Intelligent Information Technologies Department, Faculty of Computer Systems and Automatics, Vinnytsia National Technical University, Vinnytsia, e-mail: ilona.bogach@gmail.com