

РЕАЛІЗАЦІЯ МЕХАНІЗМУ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ REST API НА ОСНОВІ JWT TOKEN ТА SPRING SECURITY

Вінницький національний технічний університет

Анотація

Розроблено модуль автентифікації та авторизації для REST API з використанням JWT Token та Spring Security. Досліджено JWT як засіб автентифікації, його структуру та принцип генерації особистого токена для користувача. Розглянуто Spring Security, зокрема фільтри безпеки та Lambda DSL для керування доступом до ресурсів. Також проведено тестування з використанням JUnit5 та Postman.

Ключові слова: JWT, автентифікація, авторизація, REST API, DSL.

Abstract

Developed an authentication and authorization module for a REST API using JWT Token and Spring Security. JWT was researched as an authentication mechanism, including its structure and the process of generating a personal token for a user. Spring Security was examined, particularly security filters and the Lambda DSL for managing access to resources. Additionally, testing was conducted using JUnit 5 and Postman

Keywords: JWT, authentication, authorization, REST API, DSL.

Вступ

У сучасних інформаційних системах безпека відіграє критично важливу роль, особливо в контексті веб-додатків, що взаємодіють із користувачами через мережу. Незахищеність REST API може призвести до несанкціонованого доступу, витоку конфіденційних даних та інших загроз, що ставлять під сумнів надійність програмного забезпечення.

Сучасні підходи до забезпечення безпеки веб-додатків включають автентифікацію на основі сесій, OAuth 2.0, OpenID Connect, механізми двофакторної автентифікації, використання токенів тощо. Кожен із цих методів має свої переваги та недоліки, тому в реальних умовах доцільно використовувати їх у поєднанні, а також зважати на конкретні вимоги до застосунків для підвищення рівня захисту.

У даній роботі досліджується підхід, що поєднує JWT (JSON Web Token) та Spring Security для автентифікації та авторизації REST API. Така комбінація може забезпечити зручний механізм управління доступом без необхідності зберігання сесій на сервері, що сприятиме масштабованості та продуктивності системи. У дослідженні проаналізовано можливості цього підходу та реалізовано модуль на основі цього підходу.

Результати дослідження

У результаті дослідження було реалізовано механізм автентифікації та авторизації для REST API. Розроблений модуль довів доцільність поєднання JWT та Spring Security з метою забезпечення більш високого рівня безпеки.

Веб-токен JSON (JWT) є широко розповсюдженим і перевіреним засобом безпечного обміну інформацією у вигляді об'єктів JSON між суб'єктами. Токени цього типу вирізняються компактністю, що робить JWT гарним вибором для використання в середовищах HTML та HTTP, сумісністю з URL-адресами, підтримкою цифрового підпису за допомогою HMAC, що забезпечує розширені функції безпеки, а отже, є надійним варіантом для автентифікації в сучасних веб-додатках без сесії[1].

Для початку взаємодії користувача з ресурсами веб-додатку, користувач повинен отримати власний JWT токен для подальшої автентифікації. Для цього здійснюється запит на відповідний ендпоінт, який генерує JWT токен для поточного користувача на основі тіла HTTP POST запиту. Тіло містить ім'я користувача, електронну пошту, пароль та роль. На основі цих даних здійснюється збереження користувача в базу даних та генерація особистого токена.

JWT складається з трьох частин: заголовка (header), корисного навантаження (payload) та підпису (signature). Заголовок містить інформацію про тип токена та алгоритм підпису, наприклад, {"alg": "HS256", "typ": "JWT"}. Корисне навантаження включає набір тверджень (claims) про суб'єкт токена, такі як {"sub": "1234567890", "name": "John Doe", "admin": true}. Твердження — це частини інформації, що заявляються про суб'єкта токена. Вони представлені у вигляді пар "ключ-значення". Специфікація JWT визначає сім зарезервованих тверджень, які не є обов'язковими, але рекомендуються для забезпечення сумісності зі сторонніми додатками[2]: iss (issuer — видавець токена), sub (subject — суб'єкт токена), aud (audience — аудиторія), exp (expiration time — час закінчення дії), nbf (not before time — час, раніше якого токен не є дійсним), iat (issued at time — час видачі) та jti (JWT ID — унікальний ідентифікатор токена). Підпис у JWT створюється шляхом використання секретного ключа та алгоритму підпису (наприклад, HMAC-SHA256). Спочатку секретний ключ декодується з формату Base64 у байтовий масив, після чого з нього створюється ключ HMAC. Далі цей ключ використовується для криптографічного підпису токена, що гарантує його цілісність та автентичність.

В результаті три частини кодуються за допомогою Base64url та об'єднуються крапками, утворюючи повний JWT, наприклад:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c.

Генерація охоплює такі кроки:

1) Встановлення імені поточного користувача як subject (sub) токена:

Твердження sub (subject) використовується для ідентифікації основного суб'єкта токена, тобто користувача, для якого цей токен було видано. Зазвичай у цьому полі зберігається унікальний ідентифікатор користувача, такий як ім'я користувача.

2) Встановлення дати створення та дати закінчення терміну дії токена:

Твердження iat (issued at) вказує час, коли токен було створено, і зазвичай представлений у форматі UNIX-часу. Твердження exp (expiration time) визначає момент, після якого токен вважається недійсним.

3) Підписання токена на основі секретного ключа з використанням алгоритму HS256:

Після визначення заголовка та корисного навантаження токена, вони кодуються за допомогою Base64url та об'єднуються через крапку. Отриманий рядок підписується з використанням алгоритму HMAC-SHA256 (HS256) та секретного ключа. Алгоритм HS256 використовує спільний секретний ключ як для створення, так і для перевірки підпису.

4) Встановлення додаткових тверджень:

Окрім стандартних, можна додавати власні кастомні твердження для передачі специфічної інформації, наприклад, ролі користувача або його прав доступу.

В результаті запиту користувач отримує відповідь у форматі JSON, що містить згенерований особистий токен та токен для оновлення.

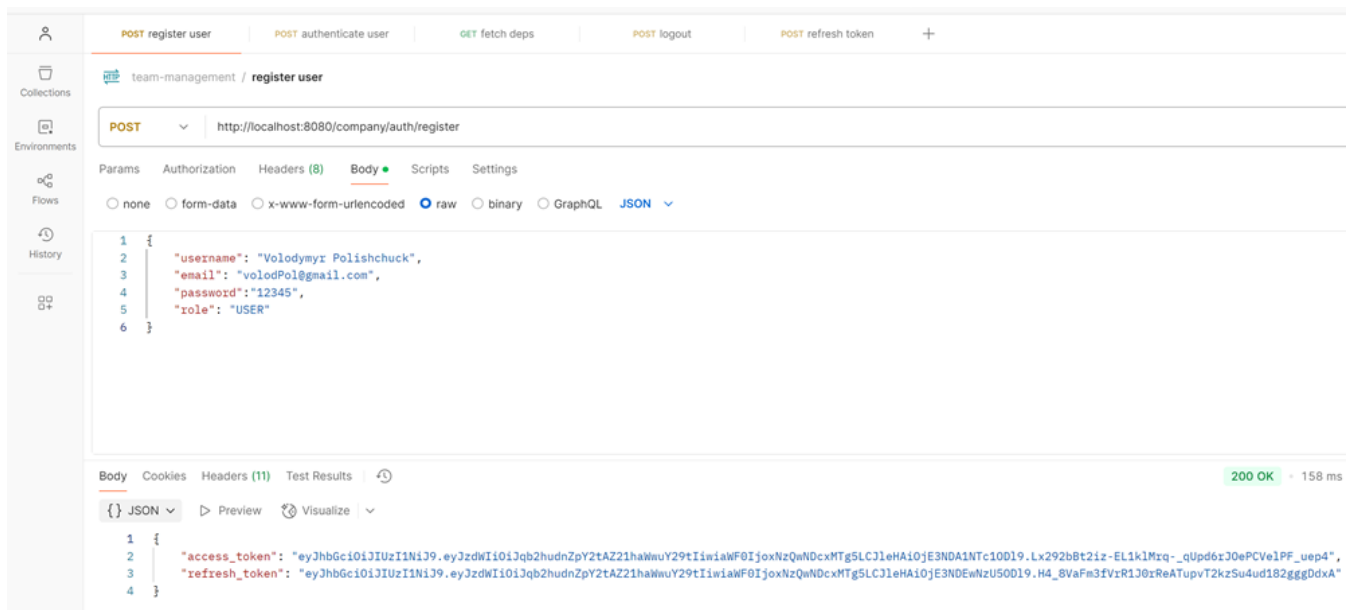


Рис. 1. Результат виконання запиту на реєстрацію користувача

Тепер користувач може використовувати токен для доступу до ресурсів веб-додатку, додаючи до кожного HTTP-запиту заголовок "Authorization" зі значенням "Bearer < особистий токен >".

Після проходження етапу автентифікації, кожен такий запит буде підлягати авторизації. Для цього застосовується Spring Security DSL. Конфігурація розпочинається з налаштування SecurityFilterChain API, що дозволяє розширити ланцюг фільтрів безпеки, які можна використати для наступних задач: автентифікація, авторизація, захист від експлойтів та інші[3]. Одним із таких розроблених фільтрів JwtAuthenticationFilter. Він спочатку перехоплює запит, перевіряє наявність токена в заголовках, а потім витягує з нього ім'я користувача та підтверджує автентифікацію за умови, що токен дійсний і не відкликаний. Якщо токен протермінований, відбувається позначення його як недійсного у базі даних, після чого запит проходить далі по ланцюгу фільтрів.

Також реалізовано інтерфейс AuthenticationEntryPoint для обробки запитів із протермінованими або відсутніми JWT-токенами. Якщо клієнт надсилає запит із недійсним токеном, цей обробник повертає відповідь із повідомленням про необхідність оновлення токена у форматі JSON, встановлюючи статус 401 Unauthorized HTTP-відповіді.

Далі конфігурується розмежування прав доступу до ендпоінтів на основі Lambda DSL, вказуючи запити яких шляхів дозволені для всіх, а які потребують зазначених ролей та привілеїв.

```
1 usage  ▲ Volodymyr Polishchuk
69  @ private void authorizeRequests(MvcRequestMatcher.Builder mvc, AuthorizeHttpRequestConfigurer<HttpSecurity>
70      .AuthorizationManagerRequestMatcherRegistry customizer) {
71      customizer
72          .requestMatchers(mvc.pattern("@"/company/auth/**")) .authorizedUrl
73          .permitAll() AuthorizationManagerRequestMat...
74
75          .requestMatchers(mvc.pattern("@"/company/manage/**")).hasAnyRole(MANAGER.name(), ADMIN.name())
76
77          .requestMatchers(POST, GENERAL_ENDPOINTS).hasRole(ADMIN.name())
78          .requestMatchers(POST, GENERAL_ENDPOINTS).hasAuthority(Privilege.ADMIN_CREATE.name())
79
80          .requestMatchers(PUT, GENERAL_ENDPOINTS).hasRole(ADMIN.name())
81          .requestMatchers(PUT, GENERAL_ENDPOINTS).hasAuthority(Privilege.ADMIN_UPDATE.name())
82
83          .requestMatchers(DELETE, GENERAL_ENDPOINTS).hasRole(ADMIN.name())
84          .requestMatchers(DELETE, GENERAL_ENDPOINTS).hasAuthority(Privilege.ADMIN_DELETE.name())
85
86          .anyRequest() .authorizedUrl
87          .authenticated();
88      }
89  }
```

Рисунок 2 – Фрагмент коду для розмежування прав доступу до ресурсів

Розроблений модуль було протестовано для перевірки коректності роботи та відповідності вимогам безпеки. Тестування виконувалося за допомогою Postman, що дозволило перевірити правильність обробки HTTP-запитів, видачі JWT-токенів та контролю доступу до ресурсів. Додатково проведено модульне тестування за допомогою JUnit5, що забезпечило перевірку окремих компонентів, зокрема сервісів генерації та валідації токенів. Інтеграційні тести дозволили оцінити взаємодію між фільтрами ланцюга безпеки в межах контексту веб-додатку, підтвердивши коректність реалізованого механізму автентифікації та авторизації.

Висновки

У результаті дослідження було реалізовано ефективний механізм автентифікації та авторизації для REST API на основі JWT-токенів і Spring Security. Використання кастомного фільтра JwtAuthenticationFilter, інтегрованого в SecurityFilterChain, дозволяє гнучко розширювати стандартні механізми безпеки, забезпечуючи перевірку автентичності користувачів на основі їхніх токенів. Крім того, налаштоване розмежування доступу до ендпоінтів за ролями та привілеями дозволяє вибірково контролювати рівень доступу до ресурсів. Такий підхід є доцільним і виправданим, оскільки забезпечує високий рівень безпеки, масштабованість та гнучкість управління доступом у сучасних веб-додатках.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Shingala K. JSON Web Token (JWT) Based Client Authentication in Message Queuing Telemetry Transport (MQTT) [Електронний ресурс] – Режим доступу: <https://arxiv.org/abs/1903.02895> (дата звернення: 10.03.2025).
2. JSON Web Tokens. Auth0 [Електронний ресурс] – Режим доступу: <https://auth0.com/docs/secure/tokens/json-web-tokens> (дата звернення: 10.03.2025).
3. Spring Security Architecture. Spring Documentation [Електронний ресурс] – Режим доступу: <https://docs.spring.io/spring-security/reference/servlet/architecture.html#servlet-securityfilterchain> (дата звернення: 10.03.2025).

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, Faculty of Computer Systems and Automatics, Vinnytsia National Technical University, Vinnytsia, e-mail: ilona.bogach@gmail.com