Advanced Security Mechanisms in the Spring Framework: JWT, OAuth, LDAP and Keycloak

Nikola Dimitrijević[1,*], Nemanja Zdravković[1], Milena Bogdanović[1] and Aleksandar Mesterovic[2]

[1]*Faculty of Information Technology, Belgrade Metropolitan University, Tadeuša Košćuška 63, 11000 Belgrade, Serbia*
[2]*Department of Security Studies and Criminology, Faculty of Art, Macquarie University, Sydney, Australia*

Abstract
The security of software applications is a critical concern in modern software development, especially with the prevalence of distributed systems and microservices. The Spring Framework stands out as a premier Java ecosystem development platform that offers an extensive range of options for implementing robust security mechanisms. This paper will shift its focus to explore advanced approaches to securing enterprise environments using the Spring Framework; specifically discussing topics such as JSON Web Token (JWT), OAuth 2.0, Lightweight Directory Access Protocol (LDAP) and Keycloak-based solutions.

The use of JWT is pivotal for the secure communication of information between disparate parties, particularly in the context of stateless authentication inherent to micro-service architectures. OAuth 2.0 serves as a standard for authorization that permits users access to shared resources while safeguarding sensitive user credentials from being exposed unnecessarily. LDAP finds practical applicability by facilitating centralized management and governance over identities and privileged accesses, chiefly advantageous when dealing with complex organizational structures at scale. As an open-source platform solution specifically tailored towards identity recognition and managed authorizations, Keycloak offers integration opportunities within Spring applications ecosystem where it introduces support services catering to commonly accepted protocols such as OpenID Connect or SAML; providing sound solutions essential in ensuring well-regulated confidential interactions akin during situations demanding trusted validations occasioned by both internal needs or external supply chain partners alike.

In this, paper, we investigate the manner in which advanced technologies can be suitably employed within the Spring Framework for creating secure and scalable applications. The analysis delves into each of these mechanisms, outlining their advantages and challenges along with integration considerations when complex business scenarios arise. Ultimately, this exploration is intended to enhance comprehension surrounding progressive security measures applicable to the Spring environment thereby equipping developers with improved capacity for constructing more resilient application solutions.

Keywords
Spring framework, Security awareness, JWT, OAuth, LDAP, Keycloak

1. Introduction

The Spring Framework has become a fundamental component in the development of contemporary Java-based applications. This is particularly attributed to its extensive infrastructure support for application building [1]. A core feature within this framework is Spring Security; an influential and personalized authentication and access control system that plays a critical role in safeguarding applications against prevalent security threats.

The Spring Framework, which was first introduced in 2003, brought about a significant transformation to Java development by introducing an Inversion of Control (IoC) container that is lightweight and simplified the management of application components. This groundbreaking concept has evolved over time with the inclusion of various modules designed to cater to different aspects of enterprise application development. Notably among these arrangements is the Spring Security module that plays an important role in securing applications through its provision of comprehensive security services tailored for Java EE-based enterprise software applications [2].

According to [3, 4] 44.1% of respondents use the free AdoptOpenJDK distribution in production. However, Oracle still has a significant presence, with 28% for their OpenJDK build and 23% for the commercial Oracle JDK.

The JSON Web Token (JWT) represents a widely adopted and established medium of securely exchanging information as JSON objects among entities. These tokens stand out for their compactness, compatibility with URLs, digital signature support resulting in enhanced security features, therefore constituting an ideal option in stateless authentication contexts within contemporary web applications [5]. When merged into Spring Security System Architecture , JWTs provide reliable and uninterrupted mechanisms compatible with the overall design of secure non-session-based functionalities instructured developments derived from spring programming methodology.

BISEC'23: 14th International Conference on Business Information Security, November 24, 2023, Niš, Serbia
*Corresponding author.
nikola.dimitrijevic@metropolitan.ac.rs (N. Dimitrijević); nemanja.zdravkovic@metropolitan.ac.rs (N. Zdravković); milenaBogdanovic@metropolitan.ac.rs (M. Bogdanović); aleksandar.mesterovic@students.mq.edu.au (A. Mesterovic)
0000-0002-6595-9277 (N. Dimitrijević); 0000-0002-0707-5174 (N. Zdravković); 0000-0003-0316-4484 (M. Bogdanović)

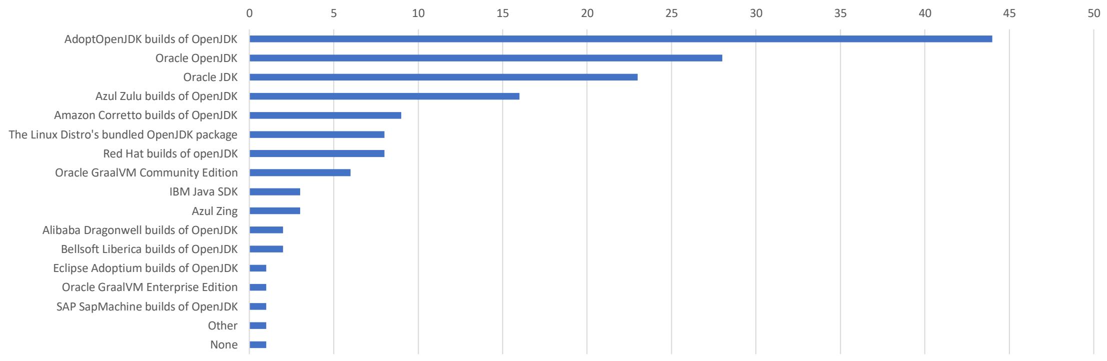


Figure 1: JDKs in production.

The OAuth 2.0 framework serves as a means of authorization that allows applications to acquire restricted access to user accounts on an HTTP service. This process involves the delegation of user authentication tasks to the hosting service, as described by Hardt in 2012. In relation to Spring Security, OAuth 2.0 presents a formidable technique for safeguarding RESTful services and APIs through outsourcing user authentication functions towards an external authorization server.

The Lightweight Directory Access Protocol (LDAP) is a commonly utilized protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. Within Spring Security, LDAP assumes a pivotal role in managing user identities and access control - particularly within extensive enterprise environments as flagged by Rouse's research findings in 2005.

Keycloak is an open-source solution for Identity and Access Management that caters to contemporary applications and services. It harbors a vast array of features including Single-Sign On (SSO), identity brokering, as well as social login capabilities. Keycloak effectively integrates with Spring Security platforms allowing developers seamless access to diverse authentication mechanisms alongside authorization protocols which enhance the security parameters over their application environment [6].

The incorporation of sophisticated security mechanisms, namely JWT, OAuth, LDAP and Keycloak into the Spring Framework via Spring Security epitomizes a noteworthy progression towards creating secure Java applications. This amalgamation not only streamlines the implementation process for intricate security requisites but also guarantees that these applications are resilient against an extensive gamut of adversarial incursions.

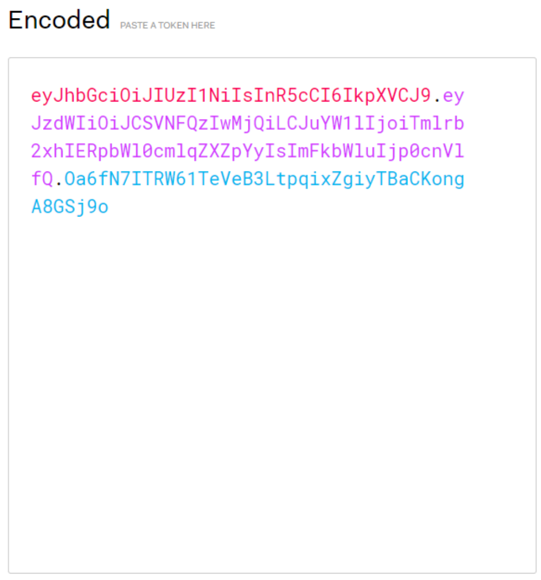
2. JWT and Its Implementation in Spring Framework

The use of JWT has garnered considerable significance in contemporary web security practices as it provides a concise and autonomous approach for transferring information between participants via a JSON object that facilitates high-level confidentiality. JWTs are designed to enable signing mechanisms, which can be achieved by employing either secret key cryptography utilizing the HMAC algorithm or public-private encryption with RSA or ECDSA algorithms, thereby assuring data integrity during transmission [7]. With such authentication protocols in place that do not rely on session state storage, JWT serves aptly suited scenarios like RESTful APIs.

A JWT generally comprises of three components: a header, a payload and a signature. The header typically encompasses two parts that comprise the kind of token - which is JWT - and the algorithm for signing being utilized. The payload entails claims regarding an entity (usually the user) alongside supplementary data. Finally, to guarantee that no changes have been made after issuance, we use signatures in order to ensure authenticity over time lapse periods.

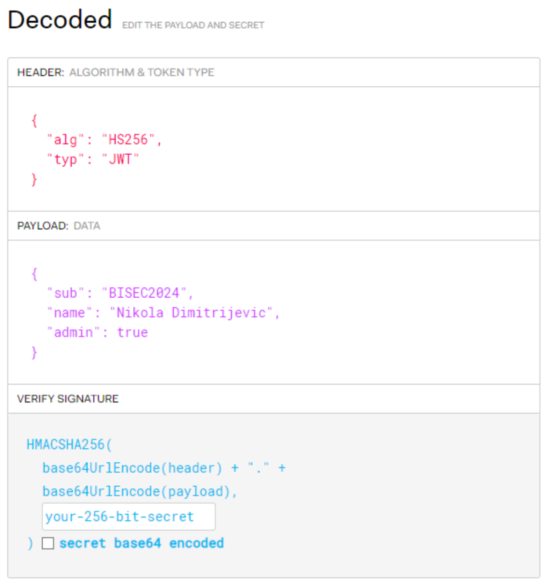
Spring Security offers comprehensive backing to JWT. The incorporation of JWT within Spring Security facilitates developers with an opportunity to address user authentication and authorization in a non-persistent approach, thereby proving significantly advantageous for RESTful applications. With the help of the Spring Security framework, validation procedures for JWTs are made accessible; ensuring that they possess proper formation whilst verifying their signature as well as claims' validity [8].

When incorporating JWT into a Spring application, developers commonly rely on established libraries such as `spring-security-oauth2` or `spring-security-jwt`. These libraries contain the



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiJCSVNFQzIwMjQiLCJuYW1lIjoiTmlrb
2xhIERpbWl0cmlqZXZpYyIsImFkbWluIjp0cnVl
fQ.Oa6fN7ITRW61TeVeB3LtpqixZgiyTBaCKong
A8GSj9o
```

Figure 2: JSON Web Token Structure - Encoded.



```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "BISEC2024",
  "name": "Nikola Dimitrijevic",
  "admin": true
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Figure 3: JSON Web Token Structure - Decoded.

essential resources required to efficiently generate, analyze and authenticate JWTs. The implementation process entails configuring a `JwtTokenStore` and `JwtAccessTokenConverter` while providing an optional `TokenEnhancer` for supplementing additional information within the JWT. Furthermore, it is imperative that developers configure an authentication manager in addition to outlining security restrictions placed upon endpoints utilized by said application instance.

The JWT protocol is especially advantageous in situations where it is essential to establish the authenticity of a user and their requisite authorizations for accessing designated resources. It serves as an added advantage within microservices architecture, wherein secure inter-service communication becomes imperative. To optimally utilize JWT with Spring framework, established guidelines comprise deployment of HTTPS to safeguard token interception threats, setting realistic expiration timeframes for tokens and judicious management pertaining information contained in payload sections so that sensitive data may not get exposed inadvertently.

The incorporation of JSON into Spring Security provides a dependable and efficient approach to managing authentication and authorization in an immutable fashion. Its versatility combined with its user-friendliness render it an optimal alternative for safeguarding applications based on the Spring framework, specifically those structured around micro-services as well as RESTful services.

3. OAuth 2.0

OAuth 2.0 is an authorization framework that grants third-party applications limited access to an HTTP service, whether through representation of a resource owner or autonomous acquisition of access privileges. Its distinction from authentication renders it indispensable in situations wherein user data must be requested from other services without compromising their respective credentials [9]. OAuth 2.0 introduces several roles:

- Resource Owner: The user who authorizes an application to access their account.
- Resource Server: Hosts the protected user data.
- Client: The application requesting access to the user's account.
- Authorization Server: Validates the identity of the resource owner and issues access tokens.

OAuth 2.0 specifies four primary grant types, catering to different application types:

- Authorization Code Grant: Ideal for clients that can securely store client secrets.
- Implicit Grant: Designed for clients that are unable to securely store client secrets.
- Resource Owner Password Credentials Grant: Suitable for highly trusted clients.
- Client Credentials Grant: Used for applications accessing their own resources.

Spring Security's OAuth 2.0 support simplifies the implementation of these grant types:

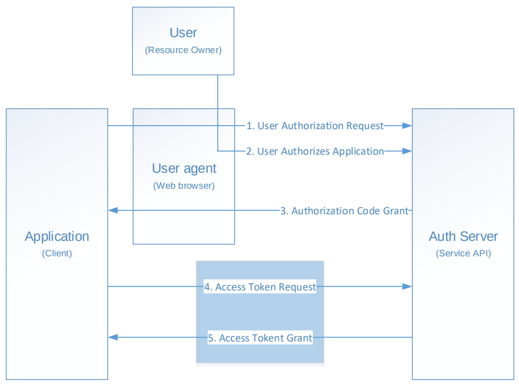


Figure 4: OAuth five-way handshake.

- Configuration: Utilize `EnableAuthorizationServer` and `EnableResourceServer` annotations to set up the authorization and resource servers.
- Client Details Service: Configure client details, including `client_id`, `client_secret`, and `scopes`.
- Token Management: Implement token store and token services to manage token generation, expiration, and refresh.
- Security Configuration: Define security constraints for different endpoints, specifying which are protected and which are publicly accessible.

Spring Security OAuth 2.0 also supports advanced features like:

- Custom Token Enhancers: To add additional information to the OAuth tokens.
- Approval Handlers: To manage user approvals for token grants.
- Redirection and User Information Endpoints: To handle user redirection after authentication and to provide user information to clients.

Key best practices include:

- Securing Client Secrets: Store client secrets securely and never expose them in client-side code.
- Validating Redirect URIs: Ensure that all redirect URIs are pre-registered and validated to prevent unauthorized redirection.
- Token Security: Use HTTPS for all communications involving tokens and credentials. Implement token revocation and rotation strategies.

The utilization of OAuth 2.0 within Spring Security presents a sturdy architecture for establishing secure authorization protocols in applications. Through the strategic employment of Spring's configuration and customization capabilities, developers possess the ability to tailor OAuth 2.0 implementation to address diverse application requirements while ensuring optimal functionality and security measures are upheld.

4. LDAP

The Lightweight Directory Access Protocol (LDAP) is a prominently utilized protocol designed for accessing and sustaining the functionality of dispersed directory information services on an Internet Protocol (IP) network. LDAP serves various purposes, including but not limited to email lookup, authentication processes as well as organization of company data. It has emerged particularly advantageous in facilitating user information management alongside enabling seamless authentication and authorization capabilities within vast enterprise environments [10].

In the sphere of Spring Security, LDAP functions as a fundamental source for both user data and authentication. With its extensive support for LDAP, Spring Security effectively facilitates seamless integration with already-existing LDAP servers. Consequently, this synergy confers upon applications the ability to validate users whilst retrieving pertinent user role information that has been preserved in an independent directory within an LDAP database.

Implementing LDAP authentication in a Spring application typically involves several steps:

- Dependency Management: Include Spring LDAP and Spring Security LDAP dependencies in your project.
- LDAP Context Source Configuration: Configure an LdapContextSource to specify the URL and base suffix of the LDAP server.
- LDAP Authentication Provider: Set up an `LdapAuthenticationProvider` to handle authentication requests. This involves specifying a user search base, user search filter, and optionally a group search base and group search filter.
- User Details Mapping: Map LDAP attributes to user details in Spring Security. This can be done using `DefaultLdapAuthoritiesPopulator` for role retrieval and `PersonContextMapper` for user information mapping.
- Security Configuration: Define security constraints in the Spring Security configuration, specifying which endpoints are protected and which are publicly accessible.

Advanced LDAP configurations in Spring can include:

- Custom User Details Service: Implementing a custom user details service for more complex user information retrieval.
- Password Policies: Configuring password policies and handling password exceptions.
- LDAP Templates: Using LdapTemplate for more complex LDAP operations beyond authentication.

When implementing LDAP in Spring, it's important to follow best practices:

- Secure Communication: Use LDAPS (LDAP over SSL) for secure communication with the LDAP server.
- Password Handling: Ensure that passwords are not logged or stored in an insecure manner.
- Injection Protection: Guard against LDAP injection attacks by validating and sanitizing input.

The incorporation of LDAP into Spring Security presents a highly effective approach to managing user authentication and authorization across enterprise applications. Through the advantageous utilization of Spring's inherent support for LDAP, software developers can establish seamless connectivity with LDAP directories while concurrently fortifying security and scalability within their respective application frameworks.

5. Keycloak

Keycloak is a state-of-the-art solution for Identity and Access Management, developed by Red Hat as an open-source software. Its primary objective lies in streamlining the integration of standard protocols such as OpenID Connect and SAML 2.0 into authentication processes while facilitating authorization procedures. In addition to centralized management console capabilities concerning user identities, Keycloak enables features that ensure SSO, two-factor authentication, and social login functionalities are supported efficiently. These advanced security provisions make it particularly suited for safeguarding modern applications' integrity within diverse service environments where tailored identity management solutions are highly valued [11].

In the context of Spring Security, Keycloak presents itself as a viable choice for an authentication and authorization server. As such, it affords Spring applications the option to delegate their user authentication and authorization protocols directly to Keycloak—a dynamic that subsequently streamlines security management efforts. This integration furthermore empowers said applications with access to advanced features exclusive to Keycloak; examples include SSO, token-based authentication measures, in addition to user federation capabilities.

Implementing Keycloak in a Spring application typically involves several steps:

- Dependency Management: Include the Keycloak Spring Boot adapter dependency in your project.
- Keycloak Server Setup: Set up and configure a Keycloak server, defining realms, clients, roles, and users.
- Spring Boot Application Configuration: Configure the Spring Boot application to use Keycloak for authentication and authorization. This involves setting up Keycloak properties in the application.properties or application.yml file.
- Security Configuration: Configure Spring Security to use Keycloak's adapter for authentication. This includes defining security constraints and specifying protected resources in the application.
- User and Role Management: Utilize Keycloak's administration console to manage users and roles, which can be mapped to Spring Security authorities.

Keycloak's integration with Spring allows for advanced customizations, such as:

- Custom User Attributes: Adding and managing custom user attributes in Keycloak.
- Identity Brokering: Configuring Keycloak to act as an identity broker between different identity providers.
- Theme Customization: Customizing the look and feel of login pages and emails.

When integrating Keycloak with Spring, it's important to follow best practices:

- Secure Communication: Ensure that all communications between the Spring application and Keycloak server are secured using HTTPS.
- Client Secrets: Securely manage and store client secrets used for communication with Keycloak.
- Token Validation: Implement proper token validation in the Spring application to prevent unauthorized access.

Keycloak's integration into Spring Security offers a powerful and flexible solution for managing authentication and authorization in applications. By leveraging Keycloak, developers can enhance the security of their Spring applications, taking advantage of features like SSO, token-based authentication, and user federation.

6. Literature overview

JWTs have now become a critical component for ensuring web security in contemporary times. In the context of this, a scholarly research titled "Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History" published in 2022 underlines the vital significance of incorporating user behavior history while utilizing JWT to optimize overall application security. It is noteworthy that Spring Security endorses such an approach via providing robust support for implementing stateless authentication and authorization features using JWT [12].

Furthermore, it is highlighted in a study in 2017 that the significance of JWTs extends across various sectors. The research exhibits the versatility of JWT usage in multiple contexts such as smart home environments, thereby accentuating its efficacy specifically with regard to Spring-based applications [13].

The utilization of OAuth 2.0 in Spring is indispensable for ensuring sound authorization measures [14]. The paper scrutinizes the intricacies and methods pertinent to microservices architecture encompassing OAuth 2.0 as a core part thereof. This approach coincides with the aid provided by Spring Security's advanced support for OAuth 2.0 protocols aimed at streamlining diverse grant types within applications built on this platform.

The well-established function of LDAP in the management of user authentication and authorization can be further enhanced through its integration with Spring Security by taking into account the principles expounded upon in [15]. The paper's elucidation on context-aware authorization within IoT and blockchain domains is highly informative for LDAP implementation within complex enterprise environments operating under Spring.

The integration of Keycloak with Spring Security provides a potent means to manage the authentication and authorization process. A recent study [16] serves as an illustrative example of how combining Keycloak and Spring Security can effectively secure APIs within a microservice-based structure. This study highlights the efficacy of utilizing Keycloak alongside Spring Security for ensuring resolute application security mechanisms.

Finally, the research paper entitled "Exploring the Utilization of JWT in MQTT" published on arXiv in 2019 delves into the versatile application of JWT within MQTT, a lightweight communication protocol. This study emphasizes that JWT can be extended to various protocols and applications, including those developed with Spring Framework [5].

7. Conclusion

The Spring Framework encompasses the integration of JWT, OAuth 2.0, LDAP and Keycloak for a multi-layered approach to security, with each component possessing its own advantages and drawbacks. In particular, JWT boasts stateless functionality as well as scalability suitability which renders it fitting for contemporary web applications; however meticulous monitoring of token security is critical in order to prevent any potential vulnerability or theft risk. OAuth 2.0 serves as an extensive yet adaptable authorization framework suitable across diverse application types such IoT implementations; nevertheless complexity may present challenges during implementation while strict adherence to best practice guidelines must be maintained continuously throughout operation.

LDAP excels at managing user identities within vast operational environments through centralized authentication mechanisms but setting up can pose significant logistical hurdles especially when confronted by rapidly changing data sets needing constant adjustments compared to alternate solutions available. Finally integrating Keycloak into microservice architectures enables simpler handling of comprehensive identity access management features significantly simplifying administration needs albeit simultaneously placing additional demands on server configuration requirements possibly introducing performance reduction issues without careful optimization attention being given determining effective trade-offs relative required specific infrastructure capability constraints.

The cumulative package delivered via incorporation all these methods launched efficiently using Spring affords robust overall system protection ensuring mitigation maximization against detrimental vulnerabilities arisen from optimal deployment following exhaustive comprehension fundamental principles defining reliable secure ecosystem operations governance broadly applicable many industry type verticals benefiting handsomely therefrom upon successful implementation completion achieving strategic business objectives intending businesses reaping profitable outcomes thereof gaining competitive advantage over peers not leveraging innovative approaches towards future-proofing their information technology systems accordingly

Acknowledgment

This paper was supported in part by the Blockchain Technology Laboratory at Belgrade Metropolitan University, Belgrade, Serbia and in part by the Ministry of Education, Science and Technological Development, Republic of Serbia ref. no. 451-03-47/2023-01/200029.

References

[1] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack, et al., The spring framework-reference documentation, interface 21 (2004) 27.

[2] C. Walls, Spring in action, 4th edition, Manning Publications, 2013.

[3] Snyk, JVM Ecosystem Report 2021, https://snyk.io/reports/jvm-ecosystem-report-2021/, 2022.
[4] Ł. Wyciślik, Ł. Latusik, A. M. Kamińska, A comparative assessment of jvm frameworks to develop microservices, Applied Sciences 13 (2023) 1343.
[5] K. Shingala, JSON web token (JWT) based client authentication in message queuing telemetry transport (MQTT), arXiv preprint arXiv:1903.02895 (2019).
[6] S. Thorgersen, P. I. Silva, Keycloak-identity and access management for modern applications: harness the power of Keycloak, OpenID Connect, and OAuth 2.0 protocols to secure applications, Packt Publishing Ltd, 2021.
[7] M. Jones, J. Bradley, N. Sakimura, RFC 7519: JSON Web Token (JWT), 2015.
[8] M. Knutson, R. Winch, P. Mularien, Spring Security: Secure your web applications, RESTful services, and microservice architectures, Packt Publishing Ltd, 2017.
[9] D. Hardt, RFC 6749: The OAuth 2.0 authorization framework, 2012.
[10] M. Rouse, Ldap (lightweight directory access protocol), Enterprise Mobile Computing news and information (2019).
[11] R. Hat, Keycloak–open source identity and access management, 2021.
[12] A. Bucko, K. Vishi, B. Krasniqi, B. Rexha, Enhancing jwt authentication and authorization in web applications based on user behavior history, Computers 12 (2023).
[13] N. Hong, M. Kim, M.-S. Jun, J. Kang, A study on a jwt-based user authentication and api assessment scheme using imei in a smart home environment, Sustainability 9 (2017).
[14] M. G. de Almeida, E. D. Canedo, Authentication and authorization in microservices architecture: A systematic literature review, Applied Sciences 12 (2022).
[15] T. Sylla, L. Mendiboure, M. A. Chalouf, F. Krief, Blockchain-based context-aware authorization management as a service in iot, Sensors 21 (2021) 7656.
[16] A. Chatterjee, A. Prinz, Applying spring security framework with keycloak-based oauth2 to protect microservice architecture apis: A case study, Sensors 22 (2022) 1703.