

Research Article



# Design and Implementation of Spring Security Framework Using OAuth 2.0 for Spring Boot Applications

Shameer Mohammed,<sup>1,\*</sup>  Ali Ibrahim Mohammed Lassakeur<sup>1</sup> and P. Rutravigneshwaran<sup>2</sup> 

<sup>1</sup> Department of Computing and Information Sciences, University of Technology and Applied Sciences Sur, Ash Sharqiyah Sur, PC-411, Sultanate of Oman

<sup>2</sup> Department of Computer Science, Karpagam Academy of Higher Education, Coimbatore, Tamil Nadu, 641021, India

\*Corresponding Author

Shameer Mohammed

✉ [shameer.mohammed@utas.edu.om](mailto:shameer.mohammed@utas.edu.om)

Received: 06 August 2026

Revised: 20 September 2026

Accepted: 22 September 2026

Published Online: 29 September 2026

## Citation

S. Mohammed, A. I. M. Lassakeur, P. Rutravigneshwaran, Design and implementation of spring security framework using OAuth 2.0 for spring boot applications, *Journal of Cybersecurity and Intelligent Systems*, 2026, 1(1), 26505, <https://doi.org/10.64189/cis.26505>.

## Open Access

This article is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/), which permits the non-commercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as appropriate credit to the original author(s) and the source is given by providing a link to the Creative Commons License and changes need to be indicated if there are any. The images or other third-party material in this article are included in the article's Creative Commons License, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons License and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this License, visit: <https://creativecommons.org/licenses/by-nc/4.0/>

© The Author(s) 2026

## Abstract

In modern web applications, ensuring secure authentication and authorization has become a critical requirement for protecting sensitive data and maintaining user trust. This research focuses on implementing an enhanced security model using encrypted OAuth 2.0 and JSON Web Tokens (JWT) within Spring Boot applications. The primary objective is to strengthen authentication and authorization mechanisms by integrating encryption techniques to secure tokens and communication channels between clients and servers. This research emphasizes encrypting and decrypting OAuth 2.0 tokens and JWTs to enhance data confidentiality and effectively manage role-based access control (RBAC). It presents the design and implementation of a secure system by enhancing the standard OAuth 2.0 framework with comprehensive JWT encryption. Building on the Spring Boot framework, the system leverages Spring Security to fortify the OAuth 2.0 flow. This approach ensures that confidential user data, claims, and, most importantly, user roles are fully protected within the token payload, preventing exposure even in the event of a token breach. The intended approach was rigorously evaluated through security tests, including unit tests and manual penetration testing, to identify potential vulnerabilities. The results demonstrate that integrating encrypted JWTs with the OAuth 2.0 protocol in Spring Boot applications significantly enhances security by protecting token contents and securing the RBAC mechanism that effectively mitigates risks associated with token interception, replay attacks, and privilege escalation, which confirms the viability and effectiveness of this enhanced security model for modern, secure application development.

**Keywords:** Spring boot; Spring security; OAuth 2.0; JSON Web Token; Encryption; Role-Based Access Control; Authentication; Authorization; API security.

## 1. Introduction

OAuth 2.0 and JSON Web Tokens (JWT) have emerged as popular solutions for authentication and authorization in modern web applications because of their scalability, flexibility and support for distributed architectures.<sup>[1-3]</sup> Traditional token-based security approaches have been widely used, but they focus mainly on ensuring the integrity of tokens using electronic signatures.<sup>[4,5]</sup> Sensitive information included in JWT payloads may remain exposed in the case of token interception. This may cause security problems such as unauthorized access, replay attacks, the disclosure of user claims and privilege escalation, especially for applications using role-based access control (RBAC) to make authorization decisions.<sup>[4,6,7]</sup> While several approaches, such as OAuth 2.0, JWT, Spring Security, Keycloak, and multi-factor authentication, have been studied to improve application security, little focus has been given to protecting the confidentiality of token contents and embedded role data by encrypting the token.<sup>[8-10]</sup> Thus, a lightweight and secure framework with OAuth 2.0, encrypted JWTs and RBAC in the spring boot ecosystem is needed. In this work, an enhanced authentication and authorization framework based on encrypted OAuth 2.0 and JWT mechanisms using spring booting and spring security is proposed and implemented. The proposed approach guarantees the contents of tokens through encryption, preserves user claims and role information and improves RBAC enforcement while supporting stateless authentication architectures. A proof-of-concept application was developed that included an authorization server, resource server, encrypted token management and secure access control mechanisms.<sup>[8]</sup> The framework uses BCrypt hashing for password protection and ensures the secure validation, refreshment, and revocation of tokens. The use of communication based on HTTPS further improves the overall security of the system.<sup>[11]</sup> The main goal of this work is to enhance data confidentiality, integrity and access control by integrating enhanced token protection with the standard workflow of OAuth 2.0. The effectiveness of the proposed system is evaluated through functional, security and penetration testing scenarios on authentication, authorization, token expiration, token revocation and RBAC validation. The results show that the proposed framework provides an additional layer of security against token interception, the disclosure of sensitive claims, replay attacks and privilege escalation. The proposed solution offers a practical and secure way to handle modern Spring Boot applications that require strong authentication and authorization mechanisms.<sup>[9]</sup>

## 2. Literature survey

Several studies and technical papers have examined the development of secure authentication and authorization

mechanisms using OAuth 2.0 and JSON Web Tokens (JWT) within the context of modern web applications. This review highlights the most relevant research focusing on web security, microservice integration, and the role of spring booting and spring security in implementing robust access-control systems.

Nishat and Muzaffar *et al.* emphasize the need for stronger authentication and authorization in web applications through the combined use of OAuth 2.0, JWT, and Multi-Factor Authentication (MFA).<sup>[12]</sup> Their work describes OAuth 2.0 as a scalable authorization framework that allows secure access delegation without exposing user credentials, whereas JWT provides a stateless mechanism for identity verification and claim transfer. They highlight that although OAuth 2.0 and JWT are powerful, integrating MFA provides an additional safeguard against credential theft and replay attacks, forming a multilayered defense for modern web applications.

Rahim *et al.* explored the use of Keycloak and OAuth 2.0 for secure authentication and authorization in a microservice-based angular and spring boot application.<sup>[13]</sup> This study illustrates how distributed architectures increase the complexity of security management, particularly when multiple microservices must independently verify and authorize users. The implementation of Keycloak as an OAuth 2.0 Authorization Server offered centralized identity management, role-based access control, and single-sign-on (SSO) functionality, demonstrating how microservices can securely communicate via tokens issued and verified through a unified identity provider. Manne *et al.* presented a detailed analysis of the spring security framework for securing Java-based web applications.<sup>[14]</sup> This paper discussed Spring Security's support for authentication mechanisms such as form-based login, LDAP, and OAuth 2.0/OIDC, along with token-based authentication via JWT. The research highlights the framework's modular architecture, comprising AuthenticationManager, AccessDecisionManager, and SecurityContext, which allows granular access control at both the HTTP and method levels. Moreover, Spring Security's ability to mitigate OWASP's top 10 vulnerabilities (e.g., CSRF and XSS) through built-in protection mechanisms was identified as a major advantage. Chaudhari *et al.* focused on secure API gateways for financial lending platforms using the Java Spring ecosystem.<sup>[15]</sup> Their paper reviewed the role of the Spring Cloud Gateway in enforcing authentication, authorization, rate-limiting, and payload inspection within microservice architectures. It underscores the importance of token-based security (OAuth 2.0 and JWT) in financial systems that handle sensitive user data and are subject to compliance standards such as the GDPR and PCI-DSS. The authors showed that centralized API

gateways can mitigate risks by validating and decrypting tokens before routing requests to backend microservices, ensuring end-to-end data confidentiality and integrity.

Katta explores the integration of Spring Security with Amazon Web Services (AWS) to address security and compliance challenges in financial services.<sup>[7]</sup> This study emphasizes the importance of a multilayered security architecture that combines OAuth 2.0, JWT-based authentication, RBAC, ABAC, and encryption strategies using AWS Key Management Service (KMS). The research highlights how financial institutions can leverage these technologies to meet regulatory requirements such as the GDPR, PCI DSS, and SOX while maintaining scalability and operational efficiency. This work underscores the relevance of combining application-level security frameworks with cloud-native services to achieve defense-in depth. Zahir and Esenalieva provide a comparative analysis of authentication methods in spring security, including basic authentication, JWT, and OAuth 2.0.<sup>[16]</sup> They detail the implementation steps, advantages, and security implications of each method. The authors note that while Basic Authentication is simple and stateless, it is insecure without HTTPS; JWT offers scalability and statelessness but requires careful key management; and OAuth 2.0 is praised for its delegation capabilities and integration with third-party identity providers. This study reinforces the importance of token encryption and secure role embedding to prevent token theft and privilege escalation. Dimitrijević *et al.* investigated advanced security mechanisms such as JWT, OAuth 2.0, LDAP, and Keycloak within the spring ecosystem.<sup>[17]</sup> The authors discussed how JWT enables stateless authentication in microservices, whereas OAuth 2.0 facilitates secure authorization across distributed systems. The integration of LDAP and Keycloak simplifies identity management and supports SSO. The authors emphasize enforcing HTTPS, encrypting tokens, and securely managing keys to mitigate interception and replay attacks. Joshi examines the use of Java and Spring Boot in the development of scalable and secure payment systems.<sup>[18]</sup> Matcha and Munish *et al.* realized the basic for robust token management, authorization restrictions, and common threat protection in any OAuth-based solution.<sup>[6]</sup> Neelan *et al.* investigated the various OAuth 2.0 grants and secure authorization practices on online, mobile, and API-based platforms. Grasping how to handle security tokens, grant selection and authorization requirements.<sup>[1]</sup> Ollila *et al.* studied OAuth 2.0 authorization for enterprise integration platforms. For support and access management, there should be standard frameworks of authorization obliged for secure integration of remote services.<sup>[2]</sup> According to Johnson *et al.* the most critical features of web application cryptographic solutions were encryption, security of keys, and transport-layer security of sensitive data. We

have higher level of token-based content protection than the signature-based ones.<sup>[4]</sup> Katta *et al.* encryption, access control, and cloud-native security services improve organizational compliance, scalability, and sensitive application data protection.<sup>[7]</sup>

This study highlights the role of the microservice architecture in achieving fault isolation and maintainability and emphasizes the importance of encryption, tokenization, and PCI DSS compliance. The author concludes that Spring Boot's embedded security features and Java's encryption libraries provide a strong foundation for secure financial applications, aligning with this project's goal of implementing encrypted JWTs and RBAC. Building upon these foundations, the current study (2025) enhances the OAuth 2.0 and JWT security models by introducing encryption of tokens within a Spring Boot application. Previous research has focused primarily on signature-based token integrity; however, this project addresses the limitations of plain payload exposure. By encrypting JWTs and implementing decryption mechanisms at the resource server, sensitive claims (including user roles for the RBAC) remain confidential even if tokens are intercepted. This approach also mitigates vulnerabilities such as replay attacks and privilege escalation, as noted in earlier works. The reviewed literature consistently emphasizes the need for encrypted tokens, secure communication channels, and role-based access control in modern authentication systems.<sup>[19]</sup> The combination of OAuth 2.0 and encrypted JWTs within the Spring Boot framework emerges as a robust solution for protecting sensitive data and enforcing authorization policies. These insights directly inform the design and implementation of the recommended approach, which enhances security through token encryption, RBAC integration, and defense-in-depth principles.

## 2.1 Research gaps

Existing OAuth 2.0 and JWT solutions primarily ensure token integrity through signatures but provide limited protection for token confidentiality and embedded RBAC information. This creates a need for a lightweight Spring Boot framework that combines encrypted JWTs with secure role-based access control. The proposed system addresses this gap by protecting both the token content and the user role through encryption.

## 2.2 Comparative analysis of existing works

According to Table 1, secure authorization frameworks, cryptographic protection, and OAuth-based authentication have been the hub of contemporary indexed publications and standards-based endeavors. Stimulating token secrecy and access control helps encrypted OAuth 2.0 and JWT frameworks.

**Table 1:** Comparative analysis of existing studies identified through the Systematic Literature Review (SLR)

| Author(s) & Year                                  | Authentication Method                    | Performance (Fast)                                                     | Security (Strong)                                                                  | Encryption/Decryption (Better)                                              |
|---------------------------------------------------|------------------------------------------|------------------------------------------------------------------------|------------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| Nishat & Muzaffar (2024) <sup>[12]</sup>          | OAuth 2.0 + JWT + MFA                    | Moderate: additional verification steps slightly affect response time  | Very Strong: multifactor verification greatly enhances account protection          | Basic: uses signed JWTs without full encryption                             |
| Rahim (2021) <sup>[13]</sup>                      | OAuth 2.0 + Keycloak (SSO)               | High: token reuse and centralized management improve speed             | Strong: centralized access control reduces credential exposure                     | Basic: JWTs signed but not encrypted by default                             |
| Manne (2023) <sup>[14]</sup>                      | Spring Security + OAuth 2.0 + JWT        | High: optimized Spring integration ensures efficient performance       | Strong: robust CSRF and XSS protection via Spring filters                          | Moderate: limited encryption integration through Spring Security            |
| Chaudhari <i>et al.</i> (2024) <sup>[15]</sup>    | OAuth 2.0 + JWT via Spring Cloud Gateway | High: centralized gateway routing improves response handling           | Very Strong: enforces authentication, rate-limiting, and compliance (GDPR/PCI-DSS) | Moderate: token inspection, minimal encryption support                      |
| Katta (2025) <sup>[7]</sup>                       | OAuth 2.0 + JWT + RBAC + ABAC + AWS KMS  | High: scalable cloud-native architecture maintains performance         | Very Strong: AWS KMS and layered security ensure regulatory compliance             | Excellent: encryption and decryption managed by AWS Key Management Service  |
| Zahir & Esenalieva (2025) <sup>[16]</sup>         | Basic Auth, JWT, OAuth 2.0 (comparative) | Moderate–High: depends on method; JWT and OAuth faster than Basic Auth | Strong: OAuth 2.0 most secure; Basic Auth weakest without HTTPS                    | Moderate: recommends token encryption for JWT and OAuth                     |
| Dimitrijević <i>et al.</i> (2023) <sup>[17]</sup> | JWT + OAuth 2.0 + LDAP + Keycloak        | High: stateless JWT improves microservice scalability                  | Very Strong: SSO and LDAP integration strengthen identity control                  | Better: recommends enforced encryption and secure key management            |
| Joshi (2021) <sup>[18]</sup>                      | Spring Boot + JWT + Encryption           | High: lightweight microservices optimize processing                    | Strong: encryption and tokenization protect payment data                           | Better: uses Java cryptography and token encryption                         |
| Hardt <i>et al.</i> (2023) <sup>[3]</sup>         | OAuth 2.1 Authorization Framework        | High-reduced complexity                                                | Very Strong tokens                                                                 | Strong encryption                                                           |
| Lodderstedt <i>et al.</i> (2025) <sup>[11]</sup>  | OAuth 2.0 Security Best Practice         | High                                                                   | Very Strong                                                                        | Strong                                                                      |
| Spring Security Team (2025) <sup>[8]</sup>        | OAuth 2.0 Resource Server with JWT       | High performance                                                       | Strong security exhibited                                                          | Better encryption technique                                                 |
| OpenID Foundation (2024) <sup>[10]</sup>          | OpenID Connect with OAuth 2.0            | High                                                                   | Very Strong                                                                        | Strong                                                                      |
| Proposed System                                   | Encrypted OAuth 2.0 + JWT + RBAC         | High: optimized token validation and decryption maintain speed         | Very Strong: encrypted tokens and RBAC prevent privilege escalation                | Excellent: full token encryption/decryption ensures maximum confidentiality |

### 2.3 Comparative discussion

Most existing approaches rely on signed JWTs that protect integrity but not confidentiality. In contrast, the proposed framework combines OAuth 2.0, encrypted JWTs, and RBAC in Spring Boot to safeguard both token contents and role information against unauthorized disclosure and privilege escalation.

**Novelty:** The proposed work integrates OAuth 2.0, JWT security, and RBAC enforcement into a unified spring boot framework with token lifecycle management and enhanced protection of role information.

**Replay-Attack Prevention:** Replay attacks are mitigated through token expiration, refresh-token validation, token revocation, and server-side verification mechanisms that reject expired or reused credentials.

## 3. Methodology

### 3.1 System architecture

The anticipated approach architecture enhances the standard OAuth 2.0 and JWT models by integrating encryption and decryption mechanisms for secure token management and role-based access control (RBAC). The

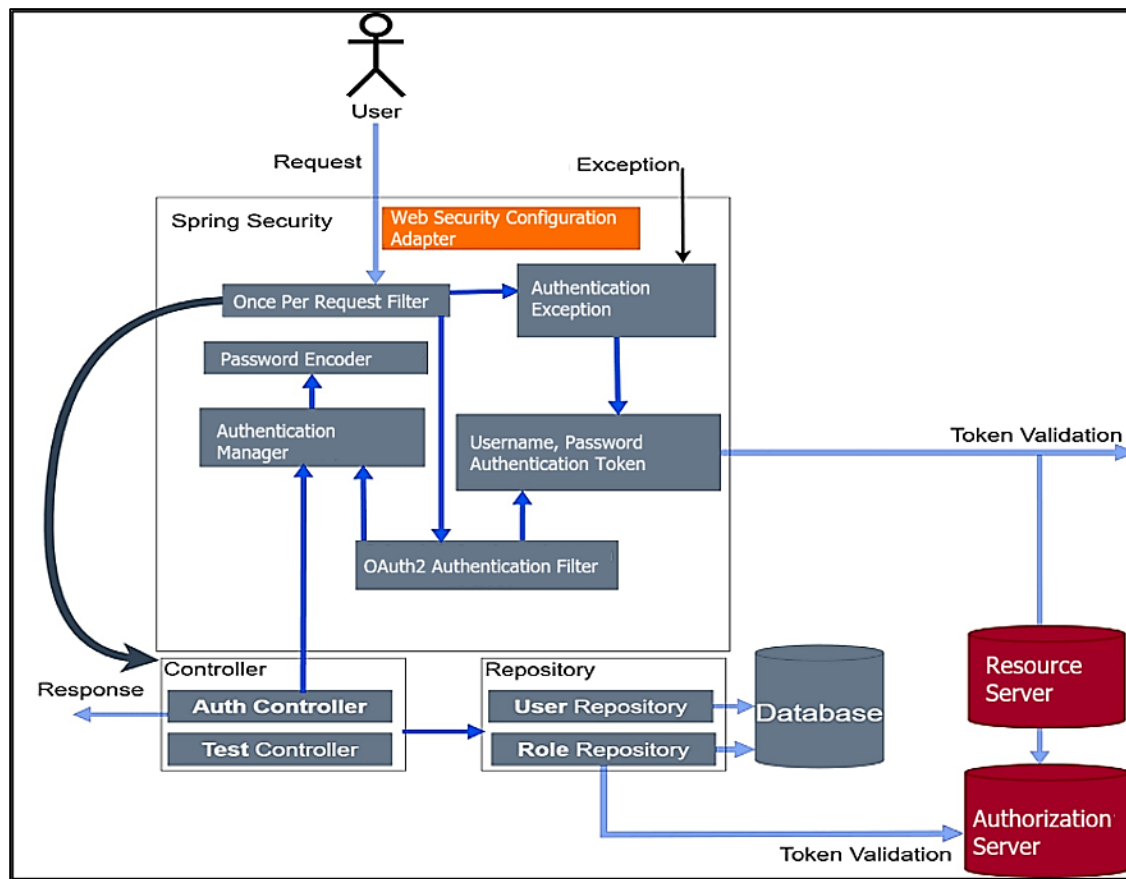


Fig. 1: Flow of proposed architecture.

system is built on the Spring Boot framework and uses Spring Security to handle authentication, authorization, and token encryption seamlessly.[3,5,8]

#### Keywords:

Encrypted OAuth 2.0 Tokens: User passwords are securely saved using BCrypt hashing, while JWT privacy is ensured through encryption tools such as RSA. Token integrity is further defended through cryptographic signing and validation. The access and refresh tokens generated by the authorization server are encrypted using algorithms such as BCrypt, ensuring confidentiality even if the token is intercepted.

Encrypted JWT Payloads: JWTs contain user claims and roles, which are encrypted before issuance and decrypted by the resource server during validation.

Role-based Access Control (RBAC): User roles and permissions are embedded securely within encrypted tokens, preventing unauthorized access or privilege escalation.

Secure Communication Channels: All communication between clients, servers, and microservices is secured via HTTPS to prevent eavesdropping.

Integration with Spring Security: Spring Security filters manage token validation, access control, and exception handling, ensuring consistent enforcement across all endpoints.

Stateless Authentication: The system remains scalable and stateless and is suitable for distributed environments

such as microservices or cloud applications.

#### Architectural workflow:

Fig. 1 depicts the OAuth2 and Spring Security authentication architecture for secure user authentication, token validation, and resource access with the below steps:

- Step1: A user sends an authentication request to the authorization server.
- Step2: The authorization server authenticates the credentials and generates an encrypted JWT access token.
- Step3: The token is returned to the client application.
- Step4: User enters the encrypted token in the authorization header of later requests to the resource server.
- Step5: The resource server decrypts and validates the token, checking claims and roles before granting access to protected APIs.
- Step6: All actions and validations are performed over secure HTTPS connections.

### 3.2 System design

#### 3.2.1 Use case diagram

The system supports the following primary actors and use cases:

User: Register, Login, Access Protected Resources, Logout, Refresh Token

Authorization Server: Authenticate Users, Issue

Encrypted Tokens, Validate Tokens

Resource Server: Decrypt Tokens, Validate Roles, Grant/Deny Access

### 3.2.2 Activity diagram

In the activity flow, a user registration is done with BCrypt password encryption, then the user logs in with credential validation, then Token generation and encryption take place, then token validation and decryption on the API request is done, then RBAC enforcement based on decrypted roles is carried out and then token refreshment and revocation handling happens.

### 3.2.3 Sequence diagrams

**Authenticate the user sequence:** In Fig. 2, the client sends credentials to the Authorization Server, which then validates them against the User\_Info database of Table 2 before generating an encrypted JWT access token and refresh tokens to returned to the client.

**Login sequence:** In Fig. 3, the user submits their username and password, which the Spring Security Authentication Manager validates. Upon successful authentication, the system generates an encrypted JWT with embedded roles and sends the token to the client for subsequent requests.

**Refresh the token sequence:** In Fig. 4, client sends an expired access token with a valid refresh token; server validates the refresh token from the Refresh\_Token database of Table 3. If valid and not revoked, a new encrypted access token is issued; an old refresh token may be rotated for security.<sup>[3,5,8,9]</sup>

### 3.3 Database design

As shown in Table 2, the User\_Info table stores essential user details required for authentication, authorization, and role management within the system.

As presented in Table 3, the Refresh\_Token table defines the roles and permissions that support the RBAC

mechanism adopted in the system.

### 3.4 Functional requirements

The proposed framework uses encrypted JWTs and spring security for implementing a secure OAuth 2.0 based authentication and authorization method. User identities are verified against saved credentials and then encrypted access and refresh tokens are issued. Authorization is enforced via Role-Based Access Control (RBAC) and roles and permissions are securely encoded into JWT claims. RSA based token encryption, signature verification and custom token validation filters are used to ensure that tokens are genuine, authentic and secret. We use refresh token rotation, automatic expiration handling, and token revocation methods to ensure token lifetime management. Proposed API access solution is secure and scalable User credentials are hashed with Bcrypt; all protected resources are only available with the confirmed JWTs.

### 3.5 Nonfunctional requirements

Non-functional security is managed by HTTPS, encrypted JWTs and Spring Security mechanisms. It provides high performance with stateless token validation. Robustness with secure token lifecycle management. System scalability for growing user bases and microservices deployment. Modular architecture easy to maintain Easy to use, clear API responses and informative error messages.

### 3.6 Implementation requirements

Table 4 shows the software required for this system to be developed.

Table 5 represents the hardware requirements for this system to be implemented.

## 4. Experiments and results

### 4.1 Prototype testing scenarios

The proposed application was tested through the

**Table 2:** User information table

| Column        | Data Type    | Constraints                 | Description                       |
|---------------|--------------|-----------------------------|-----------------------------------|
| Id            | BIGINT       | PRIMARY KEY, AUTO_INCREMENT | Unique identifier for the user    |
| user_name     | VARCHAR(255) | NOT NULL                    | The display name of the user      |
| email_id      | VARCHAR(100) | NOT NULL                    | The user's email, used for login  |
| Password      | VARCHAR(225) | NOT NULL                    | BCrypt-hashed password            |
| mobile_number | VARCHAR(20)  |                             | User's mobile number              |
| Roles         | VARCHAR(255) | NOT NULL                    | ROLE_USER/ROLE_ADMIN/ROLE_MANAGER |

**Table 3:** Information about refresh token

| Column        | Data Type | Constraints                 | Description                               |
|---------------|-----------|-----------------------------|-------------------------------------------|
| Id            | BIGINT    | PRIMARY KEY, AUTO_INCREMENT | Unique identifier for the token record    |
| refresh_token | TEXT      | NOT NULL                    | The encrypted refresh token string        |
| revoked       | BOOLEAN   | NOT NULL, DEFAULT = FALSE   | True if token revoked                     |
| user_id       | BIGINT    | FOREIGN KEY (USER_INFO.id)  | Reference to the user who owns this token |

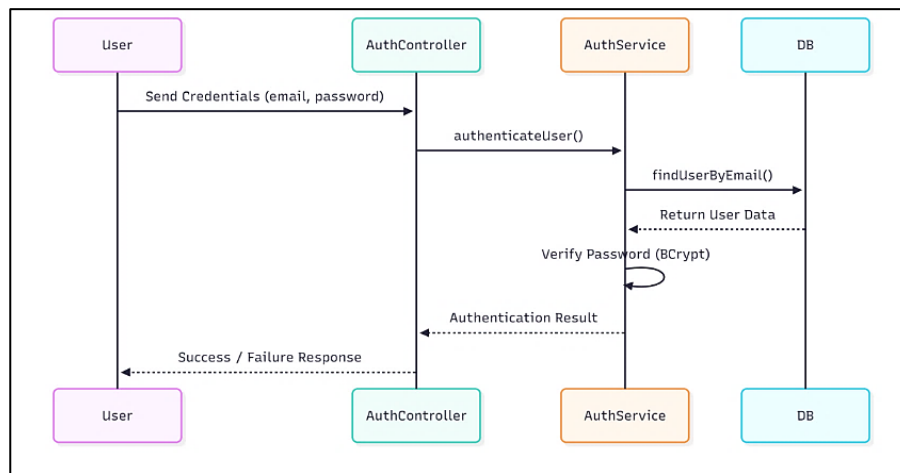


Fig. 2: User authentication sequence.

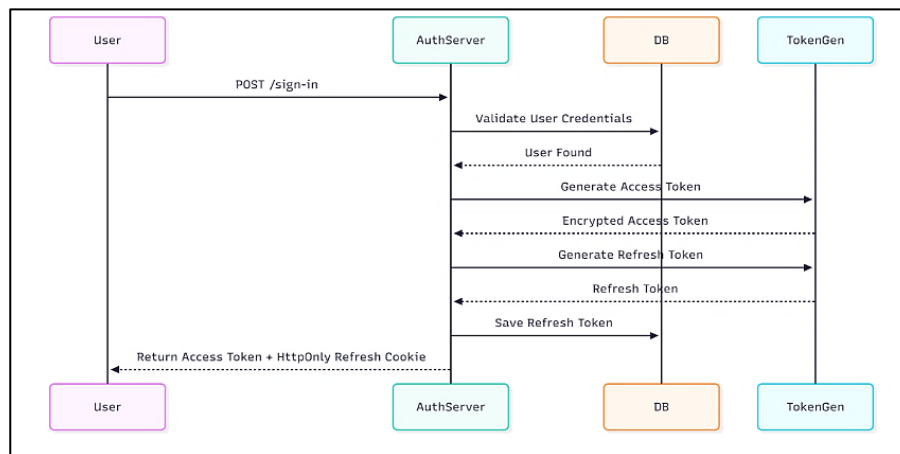


Fig. 3: Login sequence flow.

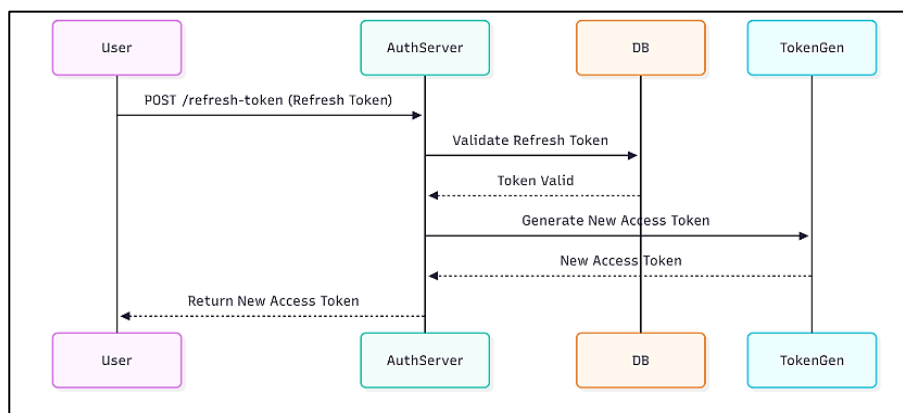


Fig. 4: Refresh token sequence.

Table 4: List of software's used

| Component            | Tools                                                    |
|----------------------|----------------------------------------------------------|
| IDE                  | Spring Tools Suite (STS)                                 |
| Frontend             | POSTMAN                                                  |
| Backend              | MySQL/H2                                                 |
| Application Server   | Apache Tomcat (embedded in Spring Boot)                  |
| Framework            | Spring Boot and Spring Security                          |
| Programming Language | Java (Spring Boot)                                       |
| Scripting Language   | JavaScript                                               |
| Middleware           | RESTful APIs for communication between client and server |
| Documentation        | Microsoft Word for project documentation                 |

**Table 5:** Hardware details

| Component        | Minimum                     | Recommended                                 |
|------------------|-----------------------------|---------------------------------------------|
| Operating System | Windows 10                  | Windows 11                                  |
| RAM              | 8 GB                        | 16 GB or higher                             |
| Processor        | Intel Core i5 or equivalent | Intel Core i7 or higher                     |
| Hard Disk Space  | 1TB free space              | 2 TB or more for development and deployment |

following scenarios to validate the security model:

*Scenario 1:* User creation and access token generation scenario a new user was created in the system with assigned roles. Upon successful registration, the system generated an encrypted JWT access token. Then the token was verified to contain encrypted payload with embedded role information.

*Scenario 2:* Role-based access control enforcement scenario; Admin Access Test: A user with ROLE\_USER attempted to access and administer only the endpoint. The system correctly denied access with an authorization error, confirming RBAC enforcement. For performing a manager access test; a user with ROLE\_MANAGER attempted to access manager-authorized endpoints. The system granted access successfully, validating role-based permissions.

*Scenario 3:* In database verification, the User\_Info Table 2 was inspected to confirm the Passwords stored as BCrypt hashes (not plaintext). Role assignments correctly persisted. The Refresh\_Token table was verified to ensure the encrypted refresh tokens stored with proper user associations and Revocation flags works correctly

*Scenario 4:* For authentication flow testing scenario users are successfully authenticated with valid credentials using Username/Password Sign-In module. Upon login, encrypted JWT access tokens were generated and returned with the help of Access token retrieval functionality. And clients will use the access token to successfully access protected resources with the help of Authenticated API access module. After 1 minute (configured expiration), the token was rejected as expired, demonstrating proper TTL enforcement is performed using Token expiration test functionality.

*Scenario 5:* Refreshing the token mechanism is tested in this scenario when the access token expired, the client submitted the refresh token to the refresh endpoint where the system validated the refresh token against the database, and a new encrypted access token was issued successfully. The old refresh token was optionally rotated for enhanced security.

*Scenario 6:* Logout and token revocation is tested with the user-initiated logout through the logout endpoint then the refresh token was marked as revoked in the database. Subsequent attempts to use the revoked refresh token were rejected then access tokens were invalidated via server-side validation mechanisms.

## 4.2 Security testing results

The system underwent rigorous security testing as described in the previous section. Fig. 5 exhibits few sample test cases after performing unit testing where Individual components (encryption/decryption modules, token filters, and RBAC validators) were tested in isolation. Then simulated attacks includes token interception attempts (mitigated by encryption), replay attack attempts (mitigated by token expiration and validation), privilege escalation attempts (mitigated by encrypted RBAC enforcement) and CSRF attack prevention (verified through Spring Security configurations).

## 4.3 Results summary

The results demonstrate that integrating encrypted JWT with the OAuth 2.0 protocol in Spring Boot apps greatly improves security, providing confidentiality and integrity of the token and role data in the face of no access or change. Sensitive data is also encrypted when sent. Server-level decryption and validation is performed to avoid privilege escalation. Token expiration and refresh techniques may also be used to mitigate the risk of replay attacks by limiting the validity time of compromised tokens.

## 4.4 Performance evaluation and metrics

The current work focuses primarily on the design, implementation, and security validation of the proposed encrypted OAuth 2.0 and JWT framework. Quantitative metrics such as latency, throughput, and computational overhead were not evaluated in the present study and are acknowledged as limitations. A dedicated subsection discussing this limitation has been added to the Results section, and comprehensive performance benchmarking has been included in the Future Recommendations section.

## 4.5 Limitations

The suggested OAuth2-based authentication system enhances security by token-based authorization and role-based access control but has limitations. For large distributed systems with heavy user loads testing may have to go beyond the framework. The study is focused on the authentication and authorization, not zero-day and sophisticated insider threats. Deep performance analysis and advanced threat detection.

## 5. Future recommendations

This research provides a robust foundation for secure authentication and authorization using encrypted OAuth 2.0 and JWTs within the spring boot ecosystem. The enhancements are recommended for future development can include the inclusion of multi-factor authentication (MFA), the use of better encryption and key management

|   | Test Objective        | Test case                                                  | Actual Result                                                                                                                | Expected Result                                                                                                      |
|---|-----------------------|------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| 1 | To test Admin login   | Enter valid Admin email and password                       | 200 ok successfully                                                                                                          | The system authenticate the user successfully and grant access to the user role resources                            |
| 2 | To verify User login  | Enter invalid User email and password                      | {<br>"timestamp": "2026-05-09T16:49:17.041+00:00",<br>"status": 401,<br>"error": "Unauthorized",<br>"path": "/sign-in"}<br>} | The system should reject the login request, return 401 Unauthorized, and deny access without generating a token      |
| 3 | To test Admin message | After login as admin, access the admin message endpoint    | Admin::admin@admin.com has this message:something                                                                            | The system should allow the admin to access the endpoint and return the admin-specific message successfully          |
| 4 | To test Sign Up       | Fill in username, email, password, mobile number, and role | {<br>"access_token": ".....",<br>"access_token_expiry": 300,<br>"token_type": "Bearer",<br>"user_name": "Ali"}<br>}          | The system should register the new user successfully, store the user data correctly, and return a valid access token |

**Fig. 5:** Testing scenarios snapshot.

systems, the extension of the framework to microservices and cloud-native settings, and the implementation of Attribute-Based Access Control (ABAC). Moreover, automated security testing, thorough audit logging, OAuth 2.1/OIDC compliance and large-scale performance evaluation can further improve the security, scalability and practical usability of the suggested solution.

## 6. Conclusion

The anticipated logic has been carefully studied and analyzed with the help of the required workflow diagrams to understand all major processes. The implementation was planned to use suitable tools and techniques to address the vulnerabilities inherent in traditional and unencrypted token-based systems. This system achieved its primary objective of strengthening authentication and authorization mechanisms by integrating encrypted OAuth 2.0 and JSON Web Tokens (JWT) within a spring boot environment. The system ensures higher data confidentiality and integrity than standard implementations through complete encryption of JWT payloads containing user claims and roles, secure role-based access control with roles embedded in encrypted tokens, statelessness suitable for distributed and microservice architectures, integration with spring security for comprehensive endpoint protection, token lifecycle management, including expiration, refresh, and revocation. Security tests (unit tests, manual penetration testing) confirmed the system's resilience to common threats such as token interception, replay attacks, and privilege escalation. This research confirms the feasibility and efficiency of this improved security model for the

construction of modern secure applications. The limitation of this study is the lack of quantitative performance benchmarking and future work will address latency, throughput, scalability and computational overhead introduced by encrypted JWT processing. By combining OAuth 2.0, encrypted JWTs, RBAC and Spring Security, result a robust, scalable, maintainable method to protect sensitive data by enforcing authorization policies in modern web applications.

## CRedit Author Contribution Statement

**Shameer Mohammed:** Conceptualization, Formal analysis, Methodology, Project administration, Software, Supervision, Validation, Writing - Original draft, Writing - Review & editing, **Ali Ibrahim Mohammed Lassakeur:** Data curation, Investigation, Software, Validation, Visualization. **P. Rutravigneshwaran:** Formal analysis, Methodology, Resources, Writing - Review & editing, Supervision.

## Funding Declaration

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

## Data Availability Statement

No data were generated or analyzed during the current study. Therefore, data sharing is not applicable to this article.

## Conflict of Interest

There is no conflict of interest.

## Artificial Intelligence (AI) Use Disclosure

The authors declare that artificial intelligence (AI)-assisted tools were used only for language refinement, grammar improvement, and manuscript structuring purposes during the preparation of this work. All technical content, experimental implementation, results, and interpretations were independently developed and verified by the authors.

## Supporting Information

Not applicable

## References

- [1] A. Neelan, OAuth 2.0 Evolution, Grant types and best practices for secure authorization in web, mobile, and API platforms, *International Journal of Leading Research Publication*, 2022, **3**, 1-12.
- [2] J. Ollila, O. Galkin, V. Leppänen, J. Papalitsas, Authorization and OAuth 2.0 in frends integration platform, Master of Science (Tech) Thesis, University of Turku, 2024, <https://www.utupub.fi/server/api/core/bitstreams/e2831052-62bd-4098-96bc-2b8f709e7670/content>.
- [3] D. Hardt, A. Parecki, T. Lodderstedt, The OAuth 2.1 authorization framework, Internet-Draft, 2023, <https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/>, Accessed 19 July 2026.
- [4] E. Johnson, A review of cryptographic protocols for securing web applications, *American Journal of Cryptography and Network Security*, 2024, **5**, 6-11, doi: 10.71465/ajcnss1082.
- [5] F. C. Ramdani, A. Rahmatulloh, R. N. Shofa, Implementation of JSON web token on authentication with HMAC SHA-256 algorithm, *Sistemasi: Jurnal Sistem Informasi*, 2023, **12**, 194-205, doi: 10.32520/stmsi.v12i1.2450.
- [6] S. Matcha, M. Kumar, Enhancing software security with oauth 2.0: Implementation strategies and vulnerability mitigation, *Journal of Emerging Technologies and Innovative Research*, 2025, **12**, e886-e902.
- [7] V. K. Katta, Cloud-enabled financial services: building secure and compliant solutions with AWS and Spring Security, *Journal of Computer Science and Technology Studies*, 2025, **7**, 675-681, doi: 10.32996/jcsts.2025.7.4.80.
- [8] Spring Security Reference Guide, 2025, <https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html>, Accessed 24 July 2026.
- [9] R. K. Sharma, Security-first API design with OAuth 2.0 in spring-based microservices, *International Journal for Research Trends and Innovation*, 2025, **10**, d268-d272, doi: 10.56975/ijrti.v10i5.206098.
- [10] OpenID Foundation, 2024, <https://openid.net/developers/how-connect-works/>, Accessed 24 July 2026.
- [11] T. Lodderstedt, J. Bradley, A. Labunets, D. Fett, Best current practice for OAuth 2.0 security, 2025, doi: 10.17487/rfc9700.
- [12] A. Nishat, J. Muzaffar, Securing web applications with OAuth 2.0, JWT, and multi-factor authentication, *Euro Vantage journals of Artificial intelligence*, 2024, **1**, 36-43, doi: 10.65923/1mpb7r55.
- [13] A. Rahim, Conception of authentication and authorization of users and microservices in an angular, Spring Boot application using Keycloak and OAuth2. 0, PhD diss., Hamburg University of Applied Sciences, 2021, <https://reposit.haw-hamburg.de/handle/20.500.12738/15303>
- [14] T. A. K. Manne, Enhancing Web security in Java applications: A deep dive into Spring Security framework, *ESP Journal of Engineering & Technology Advancements*, 2023, **3**, 179-185.
- [15] B. Chaudhari, S. C. G. Verma, S. R. Somu, A review of secure API gateways with Java Spring for financial lending platforms, *International Journal of Current Science*, 2024, **14**, 315-326, doi: 10.56975/ijcsp.v14i4.303090.
- [16] R. M. Zahir, G. Esenalieva, authentication approaches in Spring Security, Preprints, 2025, doi: 10.20944/preprints202501.0899.v1
- [17] N. Dimitrijević, N. Zdravković, M. Bogdanović, A. Mesterovic, Advanced security mechanisms in the Spring framework: JWT, OAuth, LDAP and Keycloak, in: Proceedings of the 14th International Conference on Business Information Security (BISEC 2023), 2024, 3676, 64-70.
- [18] P. K. Joshi, From concept to production: crafting payment solutions with Java and Spring Boot, *International Journal of Science and Research*, 2021, **10**, 1590-1596, doi: 10.21275/SR211101102708.
- [19] L. Fernando, T. Albert, Authenticating and authorizing users with JWT and tokenization, U.S. Patent Application 15/942,266, filed October 3, 2019.

**Publisher Note:** The views, statements, and data in all publications solely belong to the authors and contributors. GR Scholastic is not responsible for any injury resulting from the ideas, methods, or products mentioned. GR Scholastic remains neutral regarding jurisdictional claims in published maps and institutional affiliations.