International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-13 Issue-5, January 2025

An In-Depth Approach to Strengthening Security in Open-Access Libraries Utilizing JSON Web Tokens (JWT)

Ayodeji Ismail Moshood, Zoe Jeffrey

Abstract: ***In response to growing security concerns in software development, this study introduces an open-access library designed to enhance authentication systems using JSON Web Tokens (JWT). This research addresses critical challenges in integrating secure authentication mechanisms by developing a new, scalable, user-friendly library focused on security and ease of implementation. The library incorporates JWT rotation, Redis integration, and customizable validation to ensure robust, adaptable security for developers. Utilizing an agile, Extreme Programming (XP) methodology, the library was iteratively tested and optimized based on real-world developer feedback. The result of the new library shows improved usability, flexibility, and token management efficiency, demonstrating the effectiveness in supporting secure authentication practices compared to the state-of-the-art libraries. The new library is offering a practical, open-source solution to strengthen authentication systems in modern web applications, advancing the accessibility of secure, reliable software development tools.***

Keywords**:** ***Authentication, JSON Web Token, JWT, Token-based Authentication, Open-Access Library.***

I. INTRODUCTION

Authentication plays a critical role in modern software applications, ensuring that users can only access authorized resources. The use of JSON Web Tokens (JWT) for authentication has gained popularity due to its stateless nature and ability to securely carry user information [1]. JWT is currently one of the most widely used token-based authentication mechanism in global applications.

Despite the widespread acceptance of JWT, implementing secure and user-friendly authentication mechanisms remains a challenge for developers. The need for an open-source, user-friendly security measure library for authentication systems has emerged to streamline the integration of robust authentication practices into various software applications.

This paper presents a new open-source, user friendly, powerful Node.js - based secured library for handling user authentication and token management through JWTs.

Manuscript received on 31 October 2024 | First Revised Manuscript received on 15 November 2024 | Second Revised Manuscript received on 02 December 2024 | Manuscript Accepted on 15 January 2025 | Manuscript published on 30 January 2025.

*****Correspondence Author(s)

Ayodeji Ismail Moshood*, Department of Physics, Engineering and Computer Science and University of Hertfordshire, Hatfield, UK. Email: ayo.i.moshood@gmail.com, ORCID ID: 0009-0002-2724-8121

Zoe Jeffrey, Department of Physics, Engineering and Computer Science and University of Hertfordshire, Hatfield, UK. Email: z.j.jeffrey@herts.ac.uk, ORCID ID: 0000-0002-1755-1218

II. RELATED WORK

Previous research on JWT-based authentication reveals several challenges and solutions. For example, Bucko et al. (2023) explored the use of two-step authentication based on user behavior but faced penalties for mobile users with changing IP addresses [2]. Varalakshmi et al. (2022) proposed a dynamic secret key system for JWTs but struggled with the scalability of token storage [3]. Ahmed and Mahmood (2019) suggested using client and server-side timestamps for improved security, though their solution required frequent database lookups, which hampered performance [4]. Haekal and Eliyani (2020) demonstrated the implementation of JWTs in restful web services, focusing on token-based authentication challenges within service-oriented architectures [5]. The popular authentication libraries Passport JS and OAuth 2.0 [6], provides software developers with a secure system. However, the library is complex with high level of configuration requirements for customization [7].

While these approaches offer partial solutions, they often sacrifice usability or scalability. This project builds on these findings by focusing on creating a scalable, flexible JWT solution that integrates seamlessly into existing systems, while ensuring security and efficiency.

III. METHODOLOGY

This project follows an agile-based Extreme Programming (XP) development process. The Methodology involves iterative development with regular feedback cycles to ensure that the library meets both security and usability requirements [8].

Analysis: Challenges in existing JWT implementations includes complexity in usability, customization limitations due to predefined structures, and session-based defaults that require manual token validation, making them less intuitive for modern token-based authentication systems.

Design of the JWTAuthLib: The architecture of the library focuses on flexibility, modularity, and ease of integration. JWTs are used to manage user authentication, with Redis as the token storage system. The system allows for secure token rotation and blacklisting, ensuring that compromised tokens can be revoked immediately.

Key Design Features Include:

- *JWT Rotation:* Tokens are periodically refreshed to limit the lifespan of any potentially compromised tokens.
- *Redis Integration:* Redis is used for in-memory token

storage, providing fast access and the ability to blacklist tokens. Redis is an open-source, in-memory data structure store used as a database, cache, and message broker known for its high performance and scalability [7].

- *Non-opinionated Flexibility:* Developers can customize validation logic without being locked into a specific implementation pattern.

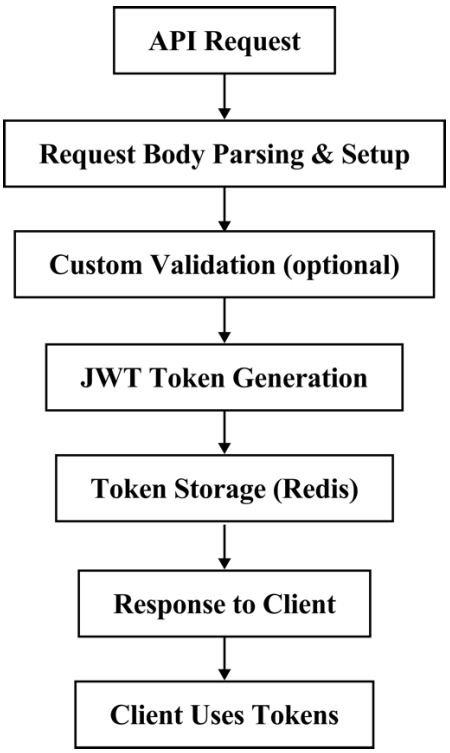
Implementation: The library was built in Node.js using TypeScript with Redis, leveraging an industry-standard library, as the session storage system. The implementation includes secure token handling, automated token rotation, and robust documentation to ensure ease of integration.

Testing: The library underwent several rounds of testing:

- *Non-opinionated Flexibility:* Developers can customize validation logic without being locked into a specific implementation pattern.
- Integration Tests: Validated successful and failed login attempts, token generation, and token refresh workflows.
- Unit Tests: Focused on specific components of the system, such as token validation and Redis-based token management.
- End-to-end Tests: Simulated real-world usage to ensure that the library could handle high traffic and large volumes of token refresh requests without performance degradation.

Library Architecture Design

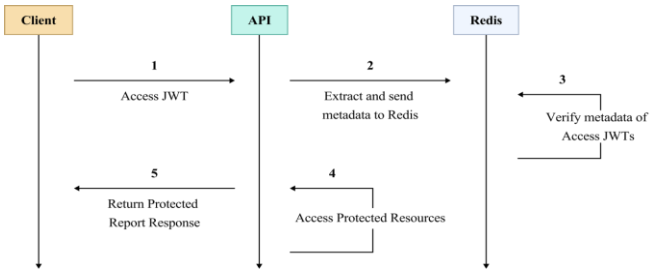
The five major components of the authentication system are Register, Access and Refresh Token, Login, Logout.



[Fig.1: Authenticated Request Flow]

A. Protected Route "Authenticated Request"

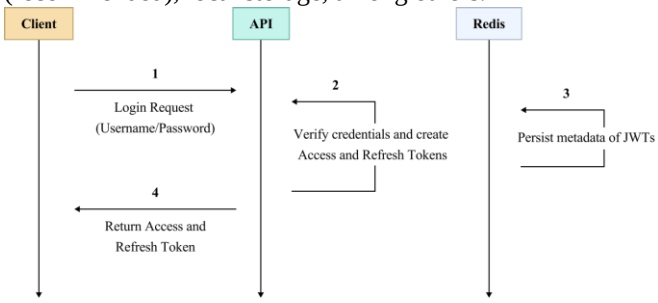
Valid Access JWT is a prerequisite for initiating an Authenticated Request. The provided Access JWT in the request undergoes a validation process. If the validation is successful, the Access JWT allows access to protected resources within the API server. Conversely, if the validation fails, the caller (client) receives an unauthorized response. The architectural layout of an authenticated request event is shown in Figure 2 below.



[Fig.2: Protected Route "Authenticated Request" Architecture Design]

B. Login

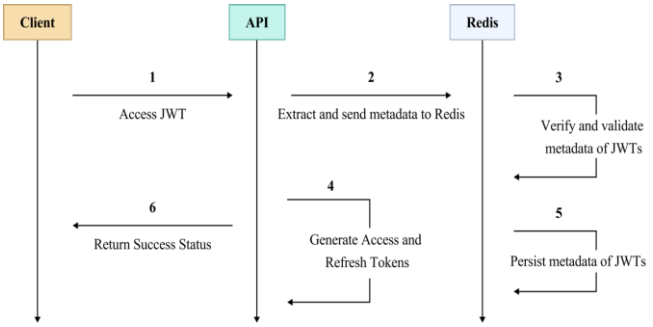
Prior to initiating an authenticated request within the application, the user is required to log in by providing their username and password. The login architecture is depicted in Figure 3 below. Post credential verification, two JWTs Access and Refresh JWTs are generated. Each JWT incorporates a Unique Identifier (UUID) as a fundamental component. The metadata of the JWTs comprises this UUID and the user's authentication ID, constituting the request. This metadata (UUID and userId) is then stored in cache storage (Redis). Upon successful generation, the Access and Refresh JWTs are provided to the requester, who has the flexibility to opt for token storage methods such as an HTTP-only cookie (recommended), local storage, among others.



[Fig.3: Login Route "Authenticated Request" Architecture Design]

C. Access & Refresh JWTs

The intended design is for the Access JWT to have a brief validity period, typically ranging from 5 to 15 minutes. Conversely, the Refresh JWT is structured to possess an extended validity span, usually falling within 1 to 7 days. Whenever the Access JWT nears expiration, just prior to an authenticated request, the Refresh JWTs come into play, automatically initiating the creation of a fresh set of Access and Refresh JWTs on the client side [9]. Axios Interceptors in a JavaScript client application serve as a prime example of this automatic trigger [10]. This project is specifically geared towards presenting an API (Application Programming Interface) endpoint, showcasing a manual trigger for refreshing the Access and Refresh JWTs [11]. The architecture for generating a new set of Access and Refresh JWTs is shown in Figure 4 below [12].



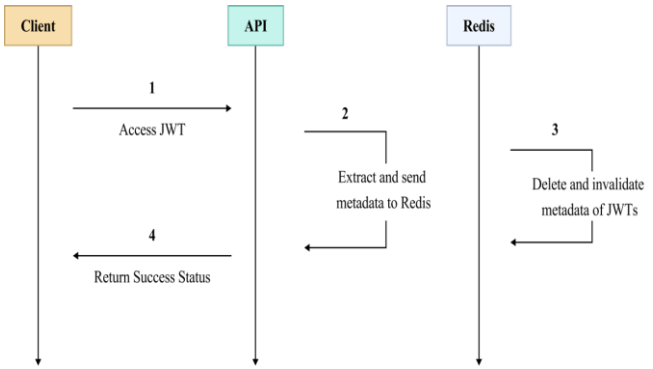
[Fig.4: Access and Refresh JWTs Architecture Design]

D. Logout

Following a successful login, users invalidate their JWT tokens by themselves, which means they'll either need to manually logout or depending on the use case of the application a user is logged in to [13]. For example, A financial technology application, sets a limited time for JWT to be invalidated if user is idle on the application.

One limitation of using JWTs is the lack of a mechanism for users to invalidate a JWT as needed. This project tackles this issue as one of its objectives [14].

As illustrated in the logout architecture depicted in Figure 5 below, a user has the ability to invalidate a JWT at their discretion by removing the metadata, used to generate the respective JWT, from the Redis cache. Following a logout event, both the Access and Refresh JWTs become invalidated and unusable for initiating any authenticated requests on behalf of the user.



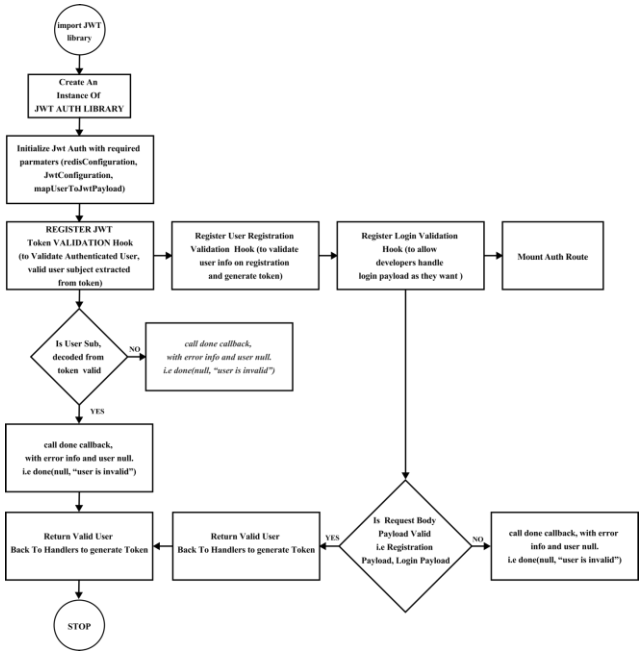
[Fig.5: Logout Architecture Design]

E. Caching Mechanism using Redis

Redis is an open-source in-memory data store that can serve as an excellent caching mechanism (Redis, 2022) [6]. A good feature of Redis is its document expiration feature, which enables us to specify the time that each metadata will expire, which is not possible if it were a traditional database that is used. Redis stores the metadata (UUID and userId) of the Access and Refresh JWTs. While the UUID is the key, the userId is the value.

There are two scenarios where metadata is stored in Redis: Login and Refresh JWTs operations. Each time an authenticated request is performed, the JWT metadata stored as part of the JWT claim is extracted and validated if it tallies with the record in Redis.

During a logout operation, the metadata is deleted from Redis, automatically invalidating the JWT.



[Fig.6: The Library Architecture User Flowchart]

The above figure demonstrates the flow chart of the different Aspects of the authentication system, from when a user logs in to when they make an authenticated request

IV. RESULTS AND DISCUSSION

The authentication library was subjected to a series of rigorous tests to ensure that it met both the security standards and the functional requirements. The following types of tests were conducted:

- *Integration Testing*

Successful Login Test: The library was tested to ensure that it correctly generates access and refresh tokens upon successful login attempts. Testing scenarios included:

- Valid credentials returning a 200 OK status and providing JWTs for further authenticated requests.
- Invalid credentials returning a 401 Unauthorized status.
- Token expiration and refresh functionality working seamlessly through automated refresh mechanisms.
- Failure Login Test: Invalid credentials or malformed requests consistently returned appropriate error messages and status codes, with no sensitive information exposed.

- *JWT Refresh Mechanism*

The refresh mechanism was evaluated under various conditions. The refresh token was successfully rotated before the expiration of the access token, minimizing the need for frequent user re-authentication.

When a refresh token was invalidated (either by user logout or token expiration), the library responded by requiring re-authentication, preventing unauthorized access.

- *Redis Caching:*

Redis was used to store token metadata, improving the efficiency of token validation. Redis automatically handles token expiration, and the system performed well under high traffic without causing performance bottlenecks. In addition, the integration of Redis improved the response time when managing blacklisted tokens

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

and allowed seamless token invalidation during logout events.

A. Security Features

A key priority of the JWTAuthLib project was to ensure security without sacrificing usability. The following features were implemented and evaluated:

- **JWT Rotation:** The rotation of JWT tokens ensured that even if a token was compromised, its lifespan was limited to a short period, reducing the potential for malicious use.
- **Blacklisting Tokens:** The library includes a token blacklisting feature that invalidates tokens upon user logout or suspicious activity. This feature was effective in ensuring that no revoked tokens could be reused.
- **Token Storage:** The use of Redis for token storage enabled secure and efficient handling of both access and refresh tokens. Redis' in-memory data store with expiration functionality ensured that tokens were stored temporarily, preventing excessive memory usage while securing token data.

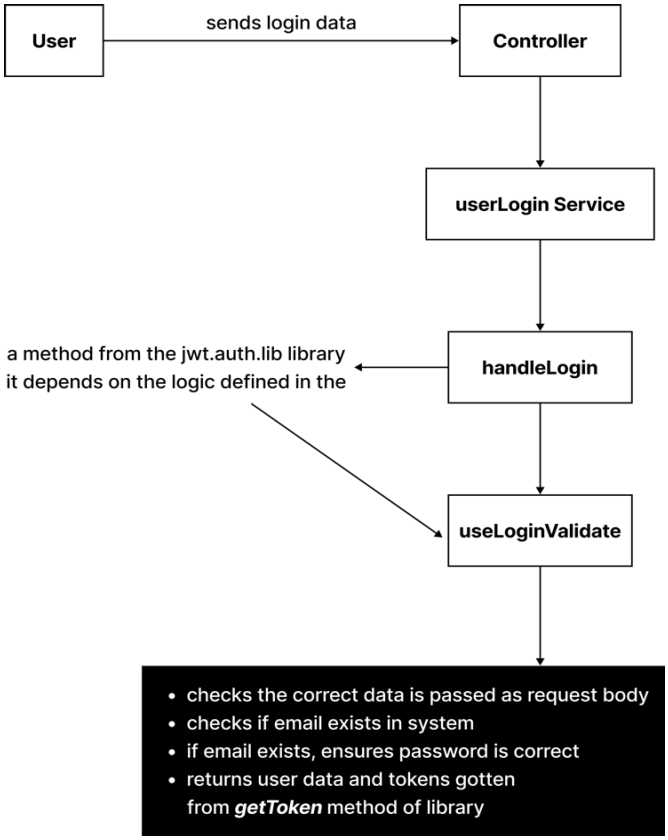
B. User Feedback

Developers reported positive feedback on the ease of integrating the library into their applications, highlighting its flexibility, customizability, and robust documentation. Some suggestions for future improvements included dynamic token expiration based on user activity.

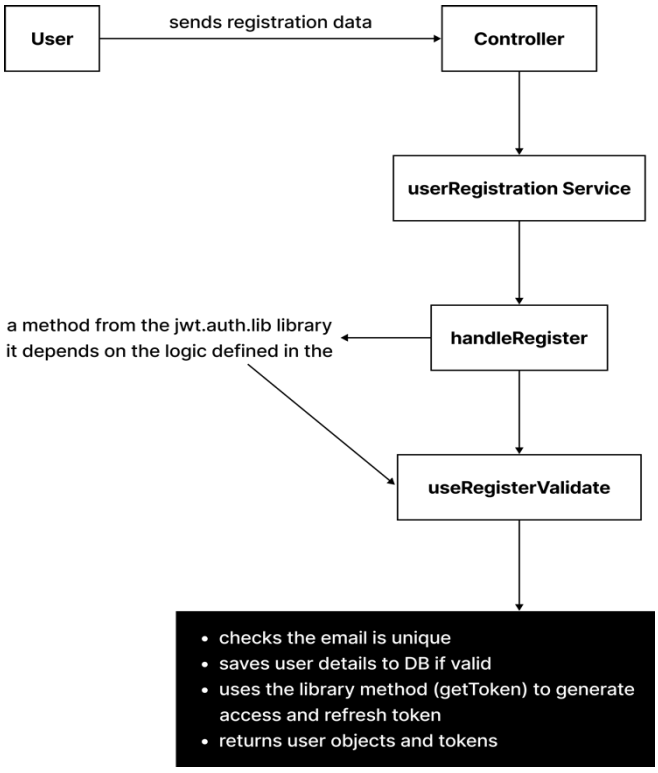
C. Example Application and Ease of Integration

Scenario: A Web Application with Secure Authentication

- User Registration: A new user submits their details (e.g., email, password) to the registration route /auth/register. The function useRegisterValidate() is used to validate the input (e.g., checking if the email is unique). After validation, the user's data is stored in the database, and jwt-auth-lib generates an access and refresh token.
- User Login: When a user logs in via /auth/login, useLoginValidate() then checks if the email and password are correct. Upon success, the library generates new JWTs (access and refresh tokens).
- Token Management: Redis is used to store refresh tokens securely, allowing efficient token validation and rotation when the access token expires. The access token is short-lived and used for securing APIs, while the refresh token provides long-term authentication.
- Guarding Routes: Secure routes like /dashboard are protected using the middleware authenticateJwt(), ensuring only authenticated users can access them.
- Token Refresh: When the access token expires, the user can use the refresh token via /auth/refresh to get a new access token without logging in again.



[Fig.7: Example Library Application: User Sends Login Details for Validation]



[Fig.8: Example Library Application: User Sends Registration Details for Validation]

D. Comparison with Other Solutions

Compared to alternatives like Passport.js and OAuth2, JWTAuthLib offers a lightweight, flexible solution for token-based authentication. Unlike Passport.js, it does not enforce strict authentication strategies, and it is easier to implement than OAuth2 in simple applications and refreshes without slowing down the application. The library has been published as an open-source through NPM and is available freely to software developers to download on the following link ***https://www.npmjs.com/package/jwt-auth-lib*** Currently it has over 1200 users.

V. FUTURE WORK

Future works includes implementing enhanced built-in validation to provide more robust default checks, such as email formats, password strength, and unique user identifiers, to reduce the need for manual work. Secondly, token revocation improvements to allow for more granular control, such as invalidating tokens upon password changes or user deactivation. Expanding storage options beyond Redis, such as in-memory or database storage, to offer more flexibility. Improve performance to be optimized for large-scale applications by improving asynchronous token handling, leveraging better async/await processes. Additional enhancement would involve integrating front-end components to demonstrate how authentication systems work. This would include designing intuitive UI elements for login functionality using email, password inputs, and keystrokes for accessibility. Such integration would provide a user-friendly experience while ensuring secure authentication, promoting inclusivity for users with diverse needs. These front-end improvements will further align the authentication system with modern usability and accessibility standards, enhancing overall user interaction.

VI. CONCLUSION

JWTAuthLib effectively addresses the challenges of building secure and user-friendly authentication systems using JWTs. Its flexibility, combined with robust security features such as token rotation and Redis-based token management, making it an ideal solution for developers.

Future work includes expanding the library's capabilities, such as adding multi-factor authentication and OAuth2 support, to further enhance its usability.

ACKNOWLEDGMENT

I would like to express my gratitude to my supervisor, Dr. Zoe Jeffrey, for her invaluable support throughout the development of this project. Her guidance and feedback were crucial to the success of this work.

DECLARATION STATEMENT

After aggregating input from all authors, I must verify the accuracy of the following information as the article's author.

- **Conflicts of Interest/ Competing Interests:** Based on my understanding, this article has no conflicts of interest.
- **Funding Support:** This article has not been sponsored or funded by any organization or agency. The independence of this research is a crucial factor in affirming its impartiality, as it has been conducted without any external sway.
- **Ethical Approval and Consent to Participate:** The data provided in this article is exempt from the requirement for ethical approval or participant consent.
- **Data Access Statement and Material Availability:** All data and materials supporting the findings of this study are available upon request. The JWTAuthLib library is open-source and publicly accessible for download via Node Package Manager (NPM) and GitHub, where the source code, documentation, and issue tracking are maintained.
- **Authors Contributions:** The authorship of this article is contributed equally to all participating individuals.

REFERENCES

1. M. Jones, B. Campbell, C. Mortimore *"JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants,"* RFC 7523, May 2015. DOI: https://doi.org/10.17487/RFC7523
2. A. Bucko, K. Vishi, B. Krasniqi and B. Rexha *"Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History"* Computers, vol. 12, no. 4, pp. 1-15, 2023. DOI: https://doi.org/10.3390/computers12040078
3. P. Varalakshmi, G. Bhuvaneswari, V. S. Praveena, D. Thomas, and S. Kannan, *"Improvising JSON Web Token Authentication in SDN,"* 2022 *International Conference on Communication, Computing and Internet of Things (IC3IoT)*, 2022, pp. 1-8. DOI: https://doi.org/10.1109/IC3IOT53935.2022.9767873.
4. S. Ahmed, and Q. Mahmood *"An authentication-based scheme for application using JSON Web token"*, 2019 *22nd International Multitopic Conference (INMIC)*. pp. 11-15. DOI: https://doi.org/10.1109/INMIC48123.2019.9022766
5. M. Haekal, and Eliyani *"Token-based authentication using JSON Web Token on SIKASIR RESTful Web Service,"* 2016 *International Conference on Informatics and Computing (ICIC)*, 2016, pp. 175-179, DOI: https://doi.org/10.1109/IAC.2016.7905711
6. D. Hardt, The OAuth 2.0 Authorization Framework, RFC 6749, Oct. 2012. DOI: https://doi.org/10.17487/RFC6749
7. M. Karlsson, *"Analysis of the use of the Redis in the distributed order processing system in the restaurant network,"* Redis Labs, 2022. DOI: *https://doi.org/10.15587/2706-5448.2021.238460*
8. C. J. Stettina, J. Garbajosa, and P. Kruchten, *"Agile Processes in Software Engineering and Extreme Programming: Proceedings of the 24th International Conference, XP 2023, Amsterdam, The Netherlands,"* Springer, 2023. DOI: https://doi.org/10.1007/978-3-031-33976-9.
9. S. Dalimunthe, E. H. Putra, M. A. F. Ridha "*Restful API Security Using JSON Web Token (JWT) With HMAC-Sha512 Algorithm in Session Management"* 2023. DOI: https://dx.doi.org/10.25299/itjrd.2023.12029
10. Reddy, P. A., & Reddy, P. H. chandan. (2020). User Authentication and Password Protection using an Algorithm ACR. In International Journal of Innovative Technology and Exploring Engineering (Vol. 9, Issue 4, pp. 3212–3215). Doi: https://doi.org/10.35940/ijitee.c8869.029420
11. Mahindrakar, P., & Pujeri, Dr. U. (2020). Insights of JSON Web Token. In International Journal of Recent Technology and Engineering (IJRTE) (Vol. 8, Issue 6, pp. 1707–1710). Doi: https://doi.org/10.35940/ijrte.f7689.038620
12. Mahindrakar, P., & Pujeri, U. (2020). Security Implications for Json web Token Used in MERN Stack for Developing E Commerce Web Application. In International Journal of Engineering and Advanced Technology (Vol. 10, Issue 1, pp. 39–45). Doi: https://doi.org/10.35940/ijeat.a1663.1010120
13. Kumar, Dr. A., Bhatia, Dr. A., Mishra, Dr. A., & Gupta, T. (2024). A Model Apporach for Identity and Access Management (IAM) System in the Cloud. In International Journal of Soft Computing and Engineering (Vol. 13, Issue 6, pp. 28–36). Doi: https://doi.org/10.35940/ijsce.d3645.13060124
14. Dungarani, R., & Gujjar, Dr. S. N. (2024). Intrusion Detection System to Secure a Network using ACNN Model and Machine Learning. In International Journal of Innovative Science and Modern

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

Engineering (Vol. 12, Issue 6, pp. 1–5). Doi: https://doi.org/10.35940/ijisme.g1319.12060624

AUTHORS PROFILE

Ayodeji Ismail Moshood is an MSc Software Engineering graduate from the University of Hertfordshire. His research focuses on open-source software development, security systems, and authentication mechanisms. Ayodeji has experience in developing software solutions for various industries and peris passionate about improving user experience and security in digital platforms.

Zoe Jeffrey, PhD is a principal lecturer at the University of Hertfordshire. In the School of Physics, Engineering and Computer Science. Her research interest is in the area of software and hardware SoC customization and optimization, signal processing algorithms and AI and machine learning. Her recent work includes software development, embedding AI for data matrix decoding and the use of barcodes in biosample management.