IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.15437

Role-Based User Authentication, Authorization and Secure Content Delivery System

Mrs. V. Krishna Vijaya[1], V. Balaram[2], P. Srikanth[3], Y. Bharath[4], Sk. Roshan Zameer[5]

Assistant Professor, Department of Information Technology,

KKR & KSR Institute of Technology and Sciences, Guntur, Andhra Pradesh, India[1]

BTech Student, Department of Information Technology,

KKR & KSR Institute of Technology and Sciences, Guntur, Andhra Pradesh, India[2-5]

Abstract: When web systems change, keeping data private and controlling who uses what becomes more pressing. Most security tools handle just simple identification tasks but struggle with fine-level permissions or strong protection for digital content, especially across complex stacks. Because of these shortcomings, this work brings a new stateless security design aimed at large- scale content distribution and user authorization tied to roles. Technically, the solution leverages a Spring Boot backend to enforce API-level security via Spring Security, while the frontend utilizes Angular to manage modular routing and responsive design. Bridging these layers, JSON Web Tokens (JWT) are employed to facilitate stateless, encrypted communication, eliminating the reliance on server-side sessions. Apart from old-style fixed designs, this setup keeps the user interface separate from the backend API - making access rules match no matter where you interact. Because of that split, what comes out is something ready for real-world use: built to grow, using MySQL to lock down data securely, while following today's top-tier security guidelines for businesses.

I. INTRODUCTION

The way cloud and web systems are growing fast has made it very important to have strong security rules to protect our digital information. Nowadays systems handle a lot of data like company secrets and personal financial information which makes them a big target for bad people who want to misuse this data. So it is very important to have security checks in place to make sure only the right people can access this data. The cloud and web systems need to have security rules to keep our digital assets safe. This is why making sure people are who they say they are and giving them access to what they need is a crucial part of building modern software systems for the cloud and web.

In the past people used to check if someone is really who they say they are by using sessions on the server. This way of doing things worked, but it had some big problems. It made it hard for systems to handle a lot of users at the same time and it made things slower in big networks. This was because the server had to keep track of a lot of information which was a lot of work. Also it was not very good at controlling what people could and could not do so sometimes people could do things they were not supposed to do and get access to information they should not have. Because of these problems a new way of keeping things secure was developed. This new way does not use sessions on the server and instead uses something called JSON Web Tokens or JWT for short. Now JSON Web Tokens are the way to keep things secure when making new applications. JSON Web Tokens are used to make sure that only the right people can get to the information [2].

This study is trying to solve some problems. It does this by creating a security system. This system uses Spring Boot for the backend and Angular for the frontend. It also uses a MySQL repository to store information. The key part of this system is a way of keeping peoples login information safe. This way is called a stateless JWT mechanism. It helps keep peoples credentials safe without using a lot of space on the server. The system also has something called Role-Based Access Control. This means that the system decides what each user can do based on their role. For example administrators can do things than standard users.

The security system is designed to keep everything safe and make sure that users, like administrators and standard users can only do what they are supposed to do. This approach ensures a cohesive security posture by synchronizing frontend route guards with backend API validation, resulting in a robust, scalable solution for enterprise deployment

II. LITERATURE REVIEW

The number of cloud-distributed applications is growing really fast. This means we need to rethink how we control who gets to use these applications. In the past big systems used something called server-side state management to keep

DOI: 10.17148/IJARCCE.2026.15437

track of who the users were. This way of doing things does not work very well for modern systems. Most experts agree that using authentication is a problem because it slows things down and makes it hard to add more power to the system when a lot of people are using it at the same time. Cloud-distributed applications are getting more and more popular and this is causing issues with the old way of controlling access.

The industry has moved towards stateless authentication protocols because of these scalability constraints. JSON Web Tokens or JWT have become the standard for this. They let authorization claims be signed and sent without storing anything on the server. Research shows that using JWT architectures really cuts down on server overhead and makes it easy to integrate with client platforms like Single Page Applications and mobile interfaces [2]. This is good for JSON Web Tokens because they make things easier. JSON Web Tokens are used a lot now.

When we talk about the server using Spring Boot and Spring Security together is a way to keep things secure. The server can check tokens and make sure people have the right permissions before anything else happens. This is done by using something called filter chains. It is very important to separate how we check who someone is from how we store information about their session. This way our security rules always work the same no matter which server is handling the request. Spring Boot and Spring Security make this process pretty straightforward [3].

When we talk about security these days we need to think about the big picture, including the part that people interact with. Tools like Angular give us tools to help with this. For example they have something called HTTP Interceptors that automatically add security tokens to requests. They also have Route Guards that stop people from navigating to parts of the website they should not be able to see. The frontend, which is what people see and interact with cannot be the only thing that keeps us safe. It is very important for making sure people can use the website easily and it helps prevent people from getting to parts of the website that they are not supposed to see, like protected views. Security is important. The frontend plays a big role in security so security and the frontend go hand in hand.

Security is not just about logging in. It needs to be really strong. This is where Role-Based Access Control comes in. Role-Based Access Control is important. It makes sure that people can only do what they are allowed to do. For example an Admin can do things than a User. Role-Based Access Control works by only giving people the access they need. This is called the principle of least privilege [1].

When it comes to storing data we need to make sure it is safe. This is where cryptographic hashing comes in. Cryptographic hashing is a must. We need to use it to store credentials. If we use MySQL to store credentials we should use algorithms like BCrypt. This helps to prevent data breaches. Data breaches are a problem in old systems. Role- Based Access Control and cryptographic hashing are important for security. We need to use them to keep our data safe.

While individual components—JWT, Spring Security, and Angular—are well-documented, there remains a scarcity of comprehensive studies that detail the end-to-end integration of these technologies into a unified, production-grade architecture. This project aims to bridge that gap by implementing a cohesive Full-Stack Security System, demonstrating a seam- less pipeline from the Angular frontend to the secure MySQL repository.

III. METHODOLOGY

Starting off, the setup splits duties across separate layers. One piece handles what users see. Another manages behind-the-scenes logic. These parts link together carefully. Security rules wrap around each stage. Access checks happen at every turn. Data storage sits apart but connects when needed. The flow moves step by step. Each section talks only to its neighbors. Protection stays built in throughout.

User interface runs on Angular, handling logins, personalized views based on roles, plus safe communication with backend services. Built with Spring Boot, the middle section manages web requests through REST, checks who can access what using tokens, applies security rules dynamically. Under- neath it all, MySQL keeps passwords hidden securely, holds account details, role assignments, and allowed actions in tables designed for quick lookups.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.15437

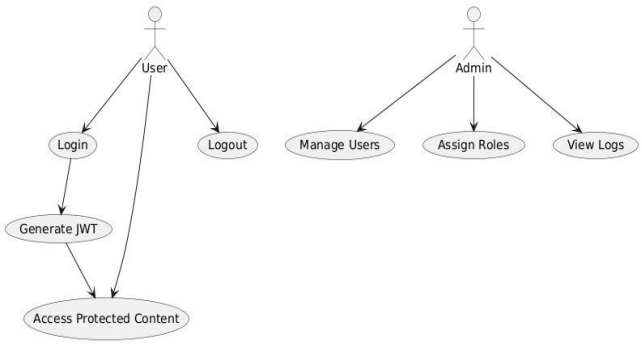

Fig. 1: User registration and role assignment workflow

A. User Registration and Role Assignment

Someone signs up with a name that stands alone, protected by scrambled letters only they should know. Built into the sys- tem, every person gets tagged - maybe just once, maybe many times - with labels like ROLE_USER or ROLE_ ADMIN. Access shifts based on those tags; some doors open, others stay shut. When keys go digital, they twist through BCrypt, making stolen copies useless to intruders watching nearby. Guessing won't help. Precomputed shortcuts fail too - all thanks to layered number games running behind each login.

B. Authentication Flow

Login begins when someone types their details into the web interface. If those match what is stored in the database, thanks to Spring handling the check, a special key gets created on the spot. This key holds who you are, your access level, plus how long it lasts before timing out. Sent back to the browser, that key then travels with every future request needing permission. Each call carries it quietly inside a standard header labeled Authorization Bearer followed by the actual code. Only once verified does the system allow passage forward.

C. Authorization and Access Control

Holding entry tight happens through Spring Security's gate- keeping layers. When a request shows up, a JWT checker

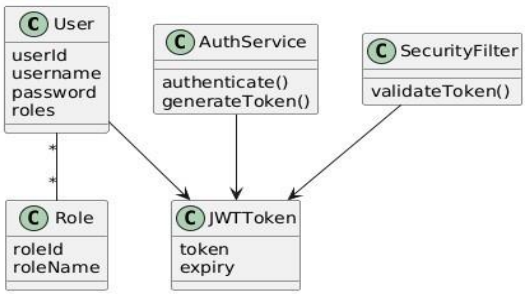

Fig. 2: Authentication flow using JWT tokens

steps in first. Validation kicks off - token gets examined, then unpacked. Out come the user roles, lined up against what each endpoint demands. Permission shifts depending on how those roles match preset controls. Only when credentials align does hidden material slip through. What slips out stays protected, seen just by rightful eyes. Security setup with stateless ses- sions JWT checks role limits encrypted connections and error control.

IV. EVALUATION METRICS

One way they checked the system involved how well it recognized real users. Checking if only the right people reached certain parts came next. How much slower things got during checks also mattered. Another point looked at how hard it was for outsiders to sneak in. People ran tests by hand, trying different roles to see what would happen.

V. RESULTS

Every test showed the system worked without issues. Only correct credentials granted entry every single time. Access stayed blocked for anyone lacking proper permissions. No extra load built up on servers thanks to token-based sessions.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026

DOI: 10.17148/IJARCCE.2026.15437

Components from frontend to backend linked cleanly through encrypted channels. Locked routes stayed off-limits whenever tokens failed verification. Security held firm even when bad or outdated keys arrived at endpoints.

VI. DISCUSSION

Tests show JWT plus role controls work well for today's online apps. Without stored sessions, handling more users becomes easier, yet permissions stay clear and manageable through assigned roles. Trouble pops up when tokens need early cancellation or safe saving on user devices. Short expiry times, fresh token updates, and better local protection help reduce those risks. While old server-session methods tie logic together tightly, this method spreads tasks apart neatly while growing without strain.

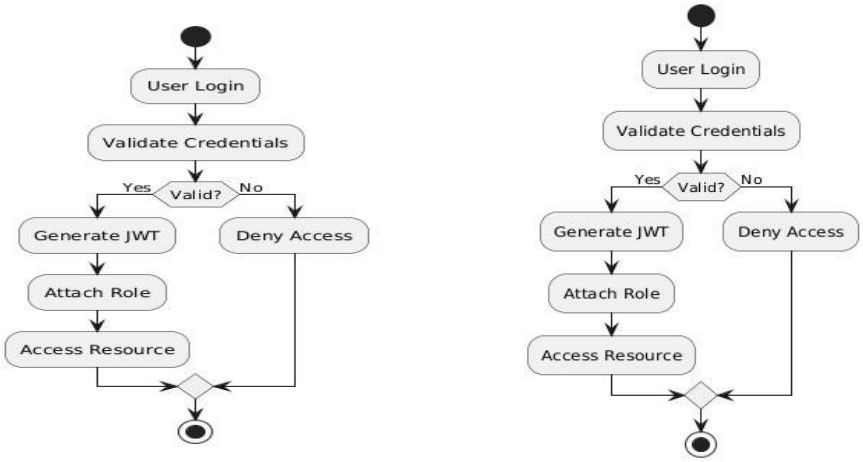

Fig. 3: Authorization and access control mechanism

VII. CONCLUSION

Security takes shape through roles here, built on Angular paired with Spring Boot and its JWT support, backed by MySQL. Tokens guard entry points, each one tied to permis- sions that shift based on user function within the setup. Access flows only where allowed, thanks to fine-grained rules rooted in who someone is inside the system. Performance doesn't stall even as demands grow - structure stays light but sturdy. Enterprise settings fit well into this mold, especially when tight control matters alongside clean separation of front and back layers. Integration runs quietly beneath, linking pieces without noise or excess weight.

VIII. FUTURE SCOPE

Down the road, new updates could bring support for OAuth 2.0 alongside OpenID Connect. Security might get a boost through multi-factor authentication. Instead of letting tokens expire on their own, systems may handle renewal and removal automatically. Running across separate microservices is another possibility. Built-in protections tailored for cloud environments might also be added later.

REFERENCES

[1]. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.
[2]. M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF RFC 7519, May 2015.
[3]. B. Spring, "Spring Security Reference Documentation," Pivotal Software, 2023.
[4]. Google, "Angular Security Guide," Official Angular Documentation, 2023.
[5]. Oracle, "MySQL Reference Manual," Oracle Corporation, 2022.
[6]. A. Shamir, "Cryptographic Hashing for Secure Password Storage," *Communications of the ACM*, vol. 58, no. 9, pp. 46–55, 2015.
[7]. S. Hardt, "OAuth 2.0 Authorization Framework," IETF RFC 6749, Oct. 2012.
[8]. D. Ferraiolo, D. Kuhn, and R. Chandramouli, "Role-Based Access Control," *Artech House*, 2003.
[9]. A. Gupta and R. Singh, "Secure RESTful APIs Using Spring Boot and JWT," *International Journal of Computer Applications*, vol. 176, no. 12, pp. 15–22, 2020.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 4, April 2026
DOI: 10.17148/IJARCCE.2026.15437

[10]. M. Verma and S. Agarwal, "Stateless Authentication Mechanisms for Web Applications," *IJCSIT*, vol. 11, no. 4, pp. 221–227, 2020.
[11]. N. Gruschka and L. Lo Iacono, "Secure Token-Based Authentication in Web Applications," *IEEE Security & Privacy*, vol. 8, no. 2, pp. 38–44, 2010.
[12]. S. Reddy and K. Kumar, "Design of Secure Full-Stack Web Applica- tions," *International Journal of Engineering Research and Technology*, vol. 9, no. 6, pp. 310–315, 2020.
[13]. T. Wilson and J. Brown, "Security Challenges in Single Page Applica- tions," *Journal of Web Engineering*, vol. 19, no. 2, pp. 145–162, 2021.
[14]. A. Jain and P. Malhotra, "Enterprise Authentication Systems Using JWT," *International Journal of Network Security*, vol. 23, no. 4, pp. 610–617, 2021.
[15]. R. Kaur and H. Singh, "Role-Based Authorization in Distributed Sys- tems," *IEEE Access*, vol. 9, pp. 145233–145241, 2021.
[16]. OWASP, "OWASP Top 10 Web Application Security Risks," OWASP Foundation, 2023.
[17]. M. Al-Shabi, "Evaluation of Modern Web Security Frameworks," *Jour- nal of Software Engineering*, vol. 15, no. 6, pp. 210–221, 2022.
[18]. R. Das and P. Chatterjee, "Secure Content Delivery in Cloud-Based Systems," *International Journal of Cloud Security*, vol. 5, no. 1, pp. 33–41, 2023.

