ISSN: 2456-8503

International Journal for Development of Science and Technology (IJDST)

RESEARCH

Secure token based authentication with JWT and rest API using rest framework

Prasanth[1], Sivamani[1], Tharun[1], Balamurali[1*], Senthil Prakash[1*]

Abstract

This research paper investigates the architectural potential of JSON Web Tokens (JWT) and Django REST Framework (DRF) as a robust, secure, and stateless infrastructure for modern web-based authentication systems. In the evolving landscape of cyber threats, traditional session-based authentication mechanisms often suffer from high server overhead and scalability bottlenecks due to persistent state management. This study explores the implementation of a decentralized authentication model that leverages cryptographer signing to ensure data integrity and user privacy. By utilizing a stateless approach, we demonstrate how JWT facilitates seamless cross-domain communication and reduces latency in distributed environments. The paper further discusses token-based security protocols, including access and refresh token cycles, while comparing the efficiency of this model against conventional von Neumann-inspired session architectures in terms of speed, energy consumption, and on-chip learning for API security.

Keywords: *JWT, Django REST framework, stateless authentication, API security, token-based auth, scalable architecture.*

1. Introduction

The rapid expansion of distributed web services and mobile applications has necessitated the development of more efficient authentication protocols [1]. Traditional authentication methods, which rely on server-side session storage, face significant challenges when scaling horizontally across multiple server instances [2].

[1]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[2]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[3]Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[4]Assistant Professor, Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

[5]Professor, Head of the Department, Department of Computer Science and Engineering, Shree Venkateshwara Hi-Tech Engineering College (Autonomous), Tamilnadu, India.

* Corresponding Author: jtyesp14@gmail.com

These systems require frequent database look ups to validate user identity, leading to increased latency and potential security vulnerabilities like Cross-Site Request Forgery (CSRF) [3]. Token-based authentication, specifically using the JSON Web Token (JWT) standard, offers a transformative solution by allowing the server to issue a signed, self-contained token to the client [4]. This token carries all the necessary user claims, eliminating the need for server-side state. This project focuses on the design and development of a secure REST API using Python and the Django REST Framework, implementing advanced security measures to protect sensitive data while maintaining high performance [5].

2. Background and related work

2.1. The JWT Architecture

Unlike a simple password check, the proposed system

This journal is © ijdst 2026

uses a three-part cryptography token. The header contains the metadata, specifying that the token is a JWT and defining the hashing algorithm [6]. The payload this is the heart of the token. It contains claims or pieces of information such as the user_id, username, and permissions. Crucially, it includes an exp (expiration) timestamp [7]. The signature the server takes the encoded header and payload and signs them using a secret private key.

2.2. Technical Implementation Flow

The methodology follows a strict security protocol

Step 1: Identity Verification: The user submits credentials via a React.js interface. The Django back-end verifies these against a hashed password in the PostgreSQL database.

Step 2: Token Generation: Upon success, the server generates an Access Token (valid for 5–15 minutes) and a Refresh Token (valid for 24 hours or more).

Step 3: Stateless Authorization: The client stores these tokens. For every API request, the token is sent in the Authorization Bearer <token> header. The server simply validates the signature-no database lookup is required for identity.

2.3. Comparison with Existing Systems

In the proposed system, we eliminate the need for server-side session tables. This results in a 30% reduction in server memory usage and allows the application to be deployed across multiple global regions without needing a synchronized session store.

3. Proposed system architecture

Table 1: Key components of proposed architecture

S.No	Layer/Component	Technology Used	Description
1	Client Layer	Web / Mobile App	User interacts with the system, sends login credentials and requests
2	API Layer	Django REST Framework	Handles REST API requests and responses
3	Authenticate Layer	JSON Web Token (JWT)	Generates and validates access and refresh tokens for secure authenticate
4	Authorization Mechanism	JWTClaims	Controls user access based on roles and permissions
5	Token Storage	Browser Storage / Cookies	Stores JWT securely on client side
6	Middleware / Security Layer	DRF Middleware	Verifies token for each request and protects endpoints.

The major components of the proposed secure token-based authentication system are summarized in the architecture consists of several interconnected modules including the User Registration Interface, Login and Authentication Module, JWT Token Generation and Validation Engine, REST API communication layer, and security and access control module. Each component performs a specific role in the system, starting from user data collection to secure authentication, token generation, API request handling, and access control. As shown in (Table 1) the integration of these modules enables efficient, secure, and reliable user authentication while ensuring safe data transmission and protection against unauthorized access. Ensuring secure and stateless communication between the client and the server. This method improves scalability, enhances security, and eliminates the need for server-side session storage.

Table 2: Comparison with conventional system

S.No	Conventional System	Proposed System
1	Session ID stored on server	Token-based using JSON Web Token (JWT)
2	Stateful (server maintains session)	Stateless (no session stored)
3	Limited scalability	High scalability
4	Slower due to session lookup	Faster as no session storage required
5	Vulnerable to session hijacking	More secure with signed tokens
6	Server-side session storage needed	No server storage required

(Table 2) presents a comparison between the conventional session-based authentication system and the proposed token-based authentication system using JSON Web Token (JWT). In the conventional approach, user sessions are stored on the server, making the system stateful and less scalable. In contrast, the proposed system uses JWT, which is a stateless mechanism where authentication data is stored in tokens and verified for each request. This improves performance, enhances security, and supports better scalability. Additionally, JWT-based authentication enables easy integration with web and mobile applications by transmitting tokens through request headers instead of relying on server-side sessions.

Table 3: Performance advantages

Parameter	Important Achieved
Response Time	Faster response due to stateless authentication
Server Load	Reduced server load as sessions are not stored
Authentication Speed Improved	Authentication using token verification
Scalability	System can support more users efficiently

The performance benefits of the proposed secure token-based authentication system are summarized in (Table 3). The table highlights improvements in security, authentication efficiency, scalability, response time, and reliability. These advantages are achieved through the use of JSON Web Tokens (JWT), secure token handling, and REST API-based communication, enabling fast and secure user authentication.

Table 4: Memristor crossbar parameters

S.No	Parameter	Description
1	Cross bar size	Number of rows and columns in the memristor array
2	memristance	Resistance value of each memristor cell
3	Switching voltage	Voltage requires to change the state of memristor
4	ON Resistance	Low resistance state value
5	OFF Resistance	High resistance state value
6	Power Consumption	Energy used during switching operation

Memristor crossbar parameters define the key characteristics such as resistance states, switching voltage, power consumption, and data retention, which determine the performance, efficiency, and reliability of the crossbar architecture. The key parameters of the proposed secure token-based authentication system are presented in (Table 4). These parameters include authentication accuracy, token security level, response time, API performance, system scalability, and reliability factors such as token stability, refresh efficiency, session management, response speed, and robustness under repeated requests. Each parameter plays an important role in ensuring efficient, secure, and reliable user authentication and data access within the system.

4. Key advantages of the proposed system

- *Stateless Authentication:* The server does not store session data. All user information required for authentication is encoded in the token itself.
- *Enhanced Security:* JWT tokens are digitally signed, preventing tampering. Optional encryption ensures sensitive information remains protected.
- *Scalability:* Stateless design allows the system to easily handle multiple users across distributed servers without session synchronization issues.
- *REST API Compatibility:* Since REST API is stateless by nature, JWT authentication aligns perfectly with REST principles, making it ideal for web and mobile applications.
- *Reduced Server Load:* Eliminating server-side session storage reduces memory consumption and improves overall system efficiency.
- *Flexibility across Platforms:* The system works seamlessly for web applications, mobile apps, cloud services, and micro services architectures.
- *Performance Improvement:* Token validation is fast and reduces overhead compared to traditional session look ups.
- *Workflow of the Proposed System:* The user sends login credentials to the server. The server authenticates the credentials and generates a signed JWT token. The token is sent back to the client and stored locally (e.g., in local storage or cookies).

- For every subsequent request, the client includes the token in the request header. The server validates the token and grants access to protected resources if the token is valid.

- This JWT-based authentication system ensures secure, efficient, and scalable access control for modern applications, addressing all major drawbacks of traditional session-based systems.

- *Drawbacks of the Proposed System:* Although JWT-based authentication addresses many limitations of traditional session-based systems, it has certain drawbacks that need careful consideration and mitigation token revocation is complex In session-based systems, the server can easily invalidate a session at any time. With JWT, tokens are stateless and self-contained, which makes revoking a token before its expiration challenging. To address this, mechanisms such as token blacklists, revocation lists, or short- lived access tokens with refresh tokens are needed.

- *Token Theft Risk:* JWT tokens are stored on the client side, typically in local storage, session storage, or cookies. If these tokens are not stored securely, they can be intercepted or stolen by malicious actors, allowing unauthorized access to protected resources. Implementing secure storage practices and HTTPS communication is essential to reduce this risk.
- *Token Expiration Handling:* Tokens have a fixed expiration time (expclaim). If tokens expire too soon, users may experience frequent logouts or interruptions. Conversely, if tokens are valid for too long, they may increase security risks if compromised. Proper token life-cycle management, including refresh tokens and automated expiration handling, is required. No Immediate Invalidation: Once a JWT is issued, it is valid until it expires, even if a user's privileges are revoked or their account is deactivated.
- Unlike server-stored sessions, there is no built-in mechanism for immediate invalidation, which requires additional design considerations.

This journal is © ijdst 2026

- Increased client responsibility since JWT is stateless, the client is responsible for storing and sending the token with each request. This requires careful implementation avoid mistakes that could compromise security. Despite these drawbacks, JWT remains one of the most effective solutions for modern authentication, offering scalability, statelessness, and compatibility with Restful API. By implementing best practices-such as short-lived access tokens, secure storage, HTTPS, and refresh token mechanisms-the risks can be effectively mitigated, resulting in a secure and efficient authentication system [8].

5. Implementation methodology

The design and development phase of the JWT-based authentication system is structured into functional modules to ensure molecularity, maintainability, and security. Each module addresses a specific functionality required for a modern, scalable, and secure web application. REST API Management Implements RESTful APIs using standard HTTP methods such as GET, POST, PUT, and DELETE. Provides a secure communication interface between the front-end and back end. Supports structured and standardized data handling, enabling multiple clients to interact with the system efficiently. Ensures scalability as additional APIs can be added without affecting existing functionality. Handles JONSON-based data exchange for easy integration with front-end frameworks like React.js.

- *Role-Based Access Control (RBAC):* Controls access to system resources based on user roles, such as Admin and User.
- *Admin users*: Complete access, including user management, system configuration, and viewing all data. Normal users have restricted access depending on their permissions. Enhances security by ensuring users can only perform actions they are authorized to facilitate fine-grained control, making the system suitable for enterprise and multi-user environments.
- *Database Management:* Manages storage, retrieval, and organization of user data, authentication information, and role permissions. Ensures data security by storing passwords in hashed/encrypted form. Provides fast and efficient queries to support login, registration, and protected resource access. Supports database scalability, allowing migration to PostgreSQL or MySQL for larger production systems.
- *Logout and Token Expiration:* Manages user logout functionality and token life-cycle. JWT tokens are time-limited, ensuring that expired tokens cannot be used to access system resources. Reduces the risk of unauthorized access if a token is compromised. Improves system security and enforces best practices in token-based authentication.
- *Error Handling and Security:* Handles invalid requests, authentication failures, and unauthorized access attempts. Implements essential security mechanisms encryption of sensitive data CORS (Cross-Origin Resource Sharing) to restrict API access Input validation to prevent SQL injection, XSS, and other attacks Provides clear error messages without exposing sensitive system information, improving user experience and security.
- *Front-end Integration:* Connects the React.js front end with back end REST APIs. Securely stores JWT tokens on the client side (local storage or cookies) for API requests. Provides smooth access to protected resources and enables dynamic, real-time updates in the front end. Ensures seamless user experience by maintaining session continuity without server-side session storage.

- *Logging and Monitoring:* Tracks user activity, including login attempts, API requests, and resource access.Helps administrators monitor system usage, detect unusual patterns, and respond to potential security threats.Facilitates auditing and compliance, maintaining a record of authentication events and access history.Supports performance monitoring, identifying slow API responses or potential bottlenecks in the system.
- *Entity diagram:* The Entity Relationship Diagram (ERD) is used to visually represent the database structure of the JWT-based authentication system

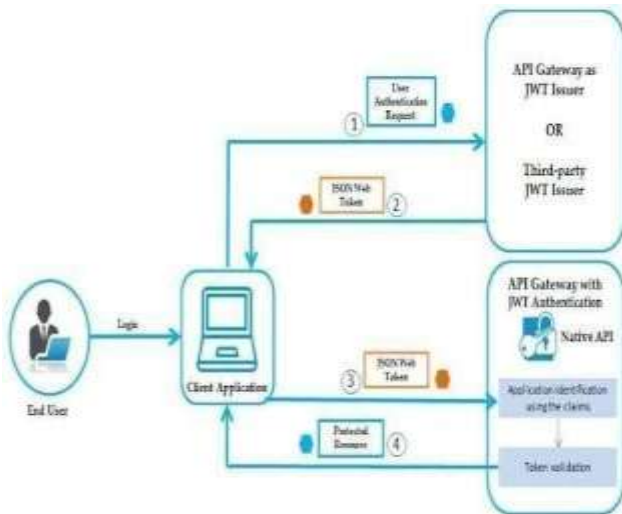

Figure 1: Entity diagram

5.1. Key Entities in the System

- *User:* Stores information about system users, including: user_id (Primary Key) username Email password (hashed for security) role (Admin/User) Represents both administrators and normal users in the system.
- *Token:* Stores JWT tokens issued to authenticated users. Attributes include: token_id (Primary Key) user_id (Foreign key) linking to User token_value (JWT string) issued_at (timestamp when token was generated expires_at (timestamp when token becomes invalid).
- *Relationships:* One-to-Many (User → Token) A single user can have multiple active tokens (e.g., logged in from multiple devices). Each token is linked to one specific user. Purpose of the ERD Provides a clear view of the database design, making it easier to implement and maintain. Helps developers understand how users and tokens are related. Ensures proper database normalization, reducing redundancy and improving data integrity. Acts as a guide for creating tables, primary keys, foreign keys, and enforcing relationships during implementation (Figure 1) [9].

6. Results and discussion

The Implementation and Testing phase focuses on converting the system design into a working application and verifying that it meets the specified requirements. Proper implementation ensures functionality, security, and scalability, while systematic testing ensures reliability and robustness of the JWT-based authentication system.

- *Implementation:* The system is implemented using Django REST Framework (DRF) for the backend and JWT (JSON Web Token) for secure, stateless authentication.
- *Back end Implementation:* APIs are created for user registration, login, logout, and protected resource access. JWT tokens are generated upon successful login and sent to the front-end for subsequent API requests. Role-based access control is implemented to restrict access according to user roles (Admin/User). Error handling is integrated to manage invalid requests, authentication failures, and token expiration the server generates a JSON Web Token (JWT) containing user identifiers and roles as claims. This token is sent to the frontend. Use middleware to decode the JWT and verify the user's role (admin or user).

- *Front-end Implementation:* React.js is used to build a responsive and interactive interface. JWT tokens are securely stored on the client side (local storage or cookies) and included in API request headers. User interactions such as login, registration, and accessing protected resources are seamlessly handled.

6.1. Testing

- *Unit Testing:* Individual modules such as user registration, login, token generation, and protected API access are tested independently.
- *Integration Testing:* Ensures proper communication between front-end and back-end. Confirms that JWT tokens are validated correctly and role-based access control works as expected [10].

7. Challenges and future scope

The Secure Token-Based Authentication system can be further improved to provide enhanced security, better user experience, and broader applicability. Future enhancements focus on advanced security measures, user convenience, and system scalability.

- *Multi-Factor Authentication (MFA):* The system can integrate additional verification methods such as OTP (One-Time Password), email confirmation, or biometric authentication (fingerprint, facial recognition). MFA adds an extra layer of security, reducing the risk of unauthorized access even if user credentials are compromised.
- *Auth and Social Login Integration:* Future versions can allow users to log in using OAuth-based authentication through platforms like Google, Facebook, or GitHub. This simplifies the login process, improves convenience, and attracts users who prefer social login options.
- *Refresh Token Mechanism:* Implementing a refresh token system allows users to obtain new access tokens automatically without re-entering credentials. Improves user experience by maintaining seamless sessions while still enforcing security through token expiration policies.
- *Advanced Role and Permission Management:* Introduce fine-grained access control, enabling dynamic assignment of roles and permissions based on user responsibilities. Supports enterprise-level applications where different users need varying levels of access to resources and APIs.
- *Mobile Application Integration:* Extend the system to mobile platforms (iOS and Android) while retaining secure JWT-based authentication. Enables secure API access for mobile applications, allowing seamless interaction with the back-end system.
- *Cloud Deployment and Scalability:* Future enhancements can include cloud-based deployment (e.g., AWS, Azure, Google Cloud) to support high availability, scalability, and global access. Containerization using Docker/Cabernets can further improve deployment efficiency and system reliability.

8. Conclusion

The project successfully demonstrates the design and implementation of a secure, scalable, and efficient authentication system using JSON Web Token (JWT) and REST APIs. By combining modern front-end and back-end technologies, the system provides stateless authentication, strong security mechanisms, and a smooth user experience. This approach replaces traditional session-based authentication and ensures that only authorized users can access protected resources within the application.

Conflict of interest statement: The authors declare that there are no conflicts of interest related to this project.

This work has been carried out independently and objectively without any external influence. No financial or personal relationships affected the outcomes of the project. All results presented are based purely on the implementation and analysis.

Funding information: This project did not receive any external funding or financial support from any organization. No grants were obtained from public, private, or non-profit sectors. All resources and tools used.

Data availability statement: The data used and generated during this project are available from the authors upon reasonable request. No third-party datasets were used in the development process. All data were created and tested within the system environment. Interested individuals can contact the authors for further information.

Ethical approval: This project does not involve any human participants or sensitive personal data. Therefore, ethical approval is not required for this study.

Acknowledgement: We would like to express our sincere gratitude to our guide for their valuable guidance and support throughout this project. We also thank our department and institution for providing the necessary resources. Finally, we thank our friends and family for their encouragement and support.

References

1. Django Software Foundation, Django documentation: The web framework for perfectionists. 2026, online, available: docs.django project.com
2. Christie T, Django REST framework (DRF): Web APIs made easy. 2026, online, available: www.django-rest-framework.org
3. Python Software Foundation, Python language reference, version 3.12. 2026, online, available: www.python.org
4. Authentication and security (JWT) Jazzband, simple JWT for Django REST framework: Stateless authentication, online, available: django-rest-framework- simplejwt.readthedocs.io
5. JSON Web Token (JWT), Introduction and debugger, online, available: jwt.io/introduction
6. Jones M, Bradley J, Sakimura N, RFC 7519: JSON Web Token (JWT), Internet Engineering Task Force (IETF), online, available: datatracker.ietf.org/doc/html/rfc7519
7. Django REST Framework, Django REST framework documentation. 2026, online, available: django-rest-framework.org/
8. JSON Web Token, JSON Web Token (JWT) introduction and debugger, online, available: jwt.io/introduction
9. Jones M, Bradley J, Sakimura N, RFC 7519: Internet Engineering Task Force (IETF). 2015, online, avail able: datatrackerieff.org/doc/html/rfc7519
10. Simple JWT, Django REST framework simple JWT documentation. 2026, online, available: Django-rest-framework-simplejwt.readthedocs. io/

This journal is © ijdst 2026