International Science Journal of Engineering & Agriculture
2024; 3(6): 117-129
https://isg-journal.com/isjea
doi: 10.46299/j.isjea.20240306.12
ISSN: 2720-6319

INTERNATIONAL SCIENCE GROUP

Secure authentication in e-government 2.0: a comparative analysis of traditional session-based and modern jwt-based authentication

Ilgar Shikhverdiyev
Baku Turkish Anatolian High School, Baku, Azerbaijan
ORCID 0009-0005-7062-506X

Elchin Babayev
Laboratory of nanostructured metal-polymer catalysts, Institute of Catalysis and Inorganic Chemistry, Baku, Azerbaijan
ORCID 0009-0004-8602-9951

Ceyhun Rahimli
Azerbaijan Technical University, Baku, Azerbaijan
ORCID 0009-0000-4779-5021

Nargiz Rahimli
Laboratory of nanostructured metal-polymer catalysts, Institute of Catalysis and Inorganic Chemistry, Baku, Azerbaijan
ORCID 0000-0003-4097-7302

Hacar Aslanova
Laboratory of nanostructured metal-polymer catalysts, Institute of Catalysis and Inorganic Chemistry, Baku, Azerbaijan
ORCID 0000-0002-5579-1802

To cite this article:
Shikhverdiyev Ilgar, Babayev Elchin, Rahimli Ceyhun, Rahimli Nargiz, Aslanova Hacar. Secure authentication in e-government 2.0: a comparative analysis of traditional session-based and modern jwt-based authentication. International Science Journal of Engineering & Agriculture. Vol. 3, No. 6, 2024, pp. 117-129. doi: 10.46299/j.isjea.20240306.12.
Received: 10 21, 2024; **Accepted**: 11 20, 2024; **Published**: 12 01, 2024

Abstract: In the era of e-Government 2.0, the security of web applications is paramount, particularly in terms of user authentication. This article provides a comprehensive examination of two primary authentication methods: session-based authentication and JSON Web Token (JWT)-based authentication. It begins by discussing the foundational aspects of secure authentication, emphasizing its importance in e-Government platforms. The article then delves into the mechanics of session-based authentication, highlighting its reliance on server-side session management and the associated challenges. In contrast, JWT-based authentication is explored in depth, showcasing its stateless nature, structure, and the advantages of using access and refresh tokens in theory and also in practice. Through a detailed code example in Express.js, the article demonstrates the implementation of JWT-based authentication in a web application. The analysis concludes by summarizing the benefits of JWT, including enhanced security, scalability, and improved user experience, making it a suitable choice for modern e-Government applications.

Keywords: Information society public services, Website, e-Government 2.0, interactive services authentication.

1. Introduction

In the digital age, the shift towards e-Government 2.0 represents a transformative evolution in how governments interact with citizens, businesses, and other entities. This new paradigm leverages advanced web technologies to offer a wide array of public services online, aiming to enhance accessibility, efficiency, and transparency. However, with the increasing reliance on web-based platforms for critical governmental functions comes an escalating need for robust security measures, particularly in the realm of authentication [8,13]. Secure authentication is the linchpin of any secure web application, serving as the gateway to sensitive information and services. In the context of e-Government 2.0, where web applications handle vast amounts of personal data, financial transactions, and confidential communications, the stakes are exceedingly high. Ensuring that only authorized individuals can access these services is paramount to maintaining public trust and safeguarding national security [2,7].

The importance of secure authentication in e-Government 2.0 can be illustrated through several key aspects:

Protection of Sensitive Data: Government websites often store and process sensitive information, including personal identification details, tax records, social security numbers, and health information. Robust authentication mechanisms prevent unauthorized access to this data, mitigating risks of identity theft, fraud, and privacy breaches.

Maintaining Public Trust: Citizens must have confidence that their interactions with government services are secure. Any security breach, particularly one involving authentication, can severely undermine public trust in digital government initiatives. Secure authentication practices help build and maintain this trust, which is crucial for the widespread adoption of e-Government services.

Preventing Fraud and Misuse: Effective authentication prevents unauthorized individuals from exploiting government services for fraudulent purposes, such as falsely claiming benefits, manipulating records, or accessing restricted information. By ensuring that only legitimate users can access services, governments can reduce the incidence of such fraudulent activities.

Compliance and Regulatory Requirements: Governments are often subject to stringent regulatory frameworks that mandate specific security standards for handling and protecting data. Implementing secure authentication mechanisms is a critical component of complying with these regulations and avoiding legal and financial penalties [10,11,18,].

As e-Government 2.0 continues to evolve, the challenge of securing web applications grows increasingly complex. This evolving complexity of securing web applications necessitates adopting advanced, scalable authentication mechanisms suitable for high-demand, distributed government systems.

2. Object and Subject of the Study

The object of this study is the authentication mechanisms utilized in e-Government 2.0 systems, focusing specifically on how these mechanisms affect the security, efficiency, and user experience of digital government platforms. The subject of the study is a comparative analysis between traditional session-based authentication and modern JSON Web Token (JWT)-based authentication models, with a particular emphasis on their implications for security and scalability in government applications. This comparison centers on evaluating each model’s resilience to critical security threats, such as identity theft, cross-site request forgery (CSRF), cross-site scripting (XSS), and session hijacking, while also assessing their compatibility with high-volume, distributed architectures. Special attention is given to each model's potential to manage token invalidation, mitigate server load, and minimize public trust risks, which are paramount to the integrity of modern e-Government systems.

3. Purpose and Objectives of the Study

The purpose of this study is to systematically analyze and evaluate the security, scalability, and feasibility of session-based and JWT-based authentication models within the framework of e-Government 2.0, with the aim of identifying the most effective approach for secure and reliable user authentication in government applications.

The specific objectives of the study are as follows:

1. To assess the security risks associated with session-based and JWT-based authentication in e-Government applications, focusing on potential attack vectors and vulnerabilities.
2. To compare the two authentication models regarding their effectiveness in protecting sensitive data, ensuring the integrity of transactions, and maintaining public trust.
3. To provide practical implementation insights through Node.js more specifically Express.js code snippets, showcasing real-world examples of how each model is implemented and highlighting security measures for preventing common security issues.
4. To evaluate the scalability and performance of each model in high-demand, distributed environments typical of e-Government services.
5. To suggest recommendations for selecting and optimizing authentication mechanisms based on security needs, compliance requirements, and operational constraints in government contexts.

By addressing these objectives, this study aims to offer a clear, evidence-based foundation for the adoption of robust authentication mechanisms in e-Government platforms, enhancing both user trust and operational security.

4. Literature Analysis

In recent years, the demand for robust and scalable authentication systems has intensified across various digital platforms, from commercial applications to e-Government services. Literature on this topic highlights several critical areas of concern, particularly surrounding user information security, vulnerabilities in web-based systems, and the emerging need for efficient, low-overhead authentication methods. This section synthesizes findings from recent studies to analyze key challenges and developments in session-based and JWT-based authentication, especially as they relate to e-Government 2.0 platforms.

4.1. JWT-Based Authentication for Reduced Server Overhead and Enhanced Security

JWT (JSON Web Token) is increasingly recognized for its role in modernizing authentication practices, particularly by reducing server dependency and enhancing scalability. According to the first study, JWT is instrumental in minimizing server overhead by lowering the frequency of database calls, managing token storage, and enabling automatic token refresh mechanisms. Furthermore, the study emphasizes the importance of secure token invalidation techniques, especially during logout or potential data breaches, underscoring JWT's adaptability in environments where reduced server load and enhanced control over token security are priorities. This finding is particularly relevant in e-Government contexts, where the ability to handle high user volumes without significant infrastructure strain is essential for smooth and scalable service delivery.

4.2. Security Challenges in e-Government Systems and the Impact of Cyber-Attacks

As the use of e-Government systems expands globally, so do the associated risks, with cyber-attacks posing a notable threat to the integrity of these platforms. The second study provides a comprehensive review of cyber-attacks impacting e-Government systems, suggesting that increased digital interconnectedness also amplifies system vulnerabilities. Notably, the study finds that cyber-attacks can undermine public trust, making security a top priority for digital governance frameworks.

The review, covering a fifteen-year period, highlights the evolving landscape of cybersecurity threats and the necessity for more rigorous protection mechanisms in e-Government systems, where trust and reliability are essential for citizen engagement and system effectiveness.

4.3. CSRF Mitigation Strategies in JavaScript Frameworks

JavaScript frameworks are at the forefront of web application development, playing a pivotal role in shaping application security practices. The third study analyzes several prominent server-side JavaScript frameworks—Express.js, Koa.js, Hapi.js, Sails.js, and Meteor.js—specifically examining how they address Cross-Site Request Forgery (CSRF) vulnerabilities. By conducting a thorough analysis using static security tools, the study evaluates the extent of CSRF mitigation across applications built with these frameworks. Results indicate that security measures vary significantly among frameworks, influencing developers' ability to produce secure applications. The study's recommendations suggest that framework developers can help prevent security vulnerabilities like CSRF through more standardized and comprehensive mitigation strategies. These insights highlight the importance of both framework design and developer practices in achieving secure authentication systems, especially for government applications where security and user data protection are paramount.

4.4. Critical Web Application Security Threats: XSS and Defensive Mechanisms

Cross-Site Scripting (XSS) vulnerabilities present a significant security risk to web applications, particularly those handling sensitive user information. The fourth study provides an extensive review of XSS as a critical and pervasive security threat, exploring both the mechanisms by which XSS attacks are executed and the defensive measures available to mitigate these risks. This study emphasizes the danger posed by XSS in high-value applications—such as e-commerce, healthcare, and banking—which are frequently targeted by malicious actors. Defensive mechanisms, such as input validation, content security policies, and secure coding practices, are recommended to protect applications from unauthorized data access. Given the sensitive nature of information in e-Government applications, the findings from this study underscore the necessity of adopting proactive security measures to prevent data breaches and enhance user trust in digital governance systems.

4.5. The Role of HTTPS in Secure Authentication

HTTPS plays a fundamental role in maintaining secure communication channels, particularly when dealing with sensitive user information like login credentials. This protocol ensures that data transmitted between client and server is encrypted, mitigating risks of interception or tampering. However, HTTPS is a stateless protocol, meaning that each request is treated independently, with no memory of previous interactions. While this design choice enhances scalability and resource management by avoiding server load associated with maintaining session state, it poses challenges for applications that rely on stateful interactions, such as user sessions. To manage these limitations, secure authentication mechanisms, like JWT and session cookies, often supplement HTTPS by handling stateful aspects independently. The reviewed literature emphasizes HTTPS as a foundational layer for secure interactions, complemented by advanced authentication strategies to ensure both security and scalability in complex e-Government systems [9,15].

Synthesis of Findings

The reviewed literature collectively emphasizes that as web-based applications evolve, so too must the approaches to authentication and security. JWT-based authentication is emerging as a viable alternative to traditional session-based models, particularly in contexts requiring scalability and reduced server load. However, the inherent risks associated with token storage and invalidation remain an active area of research. Additionally, cyber-attacks present a substantial threat to e-

Government systems, where maintaining public trust is crucial. The literature suggests that while advancements in security practices—such as CSRF and XSS prevention—are being integrated into modern frameworks and application design, there remains a pressing need for ongoing research and the adoption of adaptive, secure authentication methodologies in the realm of e-Government.

5. Research Methods

To provide a comprehensive understanding of secure authentication in practice, this study incorporates hands-on implementations of both session-based and JWT-based authentication in Node.js. Through practical code examples, each authentication model is examined in a real-world context, demonstrating configurations for secure session handling, token management, and protection against security threats common in e-Government applications.

Session-Based Authentication.

Session-based authentication addresses the stateless nature of HTTP by providing a mechanism to maintain state across multiple requests. When a user logs in to a web application, the server creates a session and associates it with the user. This session is then used to track the user's authenticated state across subsequent requests.

To understand how Session-based authentication is implemented, we explored its usage on the server side of the application using JavaScript, specifically with the Express framework based on Node.js.

(Implementation details are explained in the code as comments)

```
const User = require('../model/User');
const session = require('express-session');
const MongoStore = require('connect-mongo');
const bcrypt = require('bcrypt'); // For secure password comparison

// Initialize session middleware for managing user sessions
// - `secret`: A string used to sign the session ID cookie for security purposes.
// - `resave`: Ensures that sessions are only saved if modified, reducing unnecessary database writes.
// - `saveUninitialized`: Prevents uninitialized sessions (sessions with no data) from being stored.
// - `store`: Specifies MongoDB as the storage option, which maintains sessions after server restarts.
// - `cookie`: Configures cookie properties like expiration time, security, and accessibility settings.
const sessionMiddleware = session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    store: MongoStore.create({
        mongoUrl: process.env.MONGO_URI,
        collectionName: 'sessions' // MongoDB collection where sessions are stored
    }),
    cookie: {
        maxAge: 1000 * 60 * 60 * 24, // Session expires after 1 day (in milliseconds)
        httpOnly: true, // Ensures cookie is accessible only by the web server
        secure: process.env.NODE_ENV === 'production', // Enables HTTPS-only cookies in production
        sameSite: 'strict' // Prevents CSRF by restricting cookie use to same-origin requests
    }
```

```
});

// Login handler function for authenticating users with sessions
// - Receives username and password from request body and validates them against the database.
// - If authentication is successful, creates a session storing user data, accessible on future requests.
const handleLogin = async (req, res) => {
    const { user, pwd } = req.body; // Extracts username and password from request body
    if (!user || !pwd) {
        return res.status(400).json({ 'message': 'Username and password are required.' }); // Validation
error
    }

    // Finds user in the database based on username
    const foundUser = await User.findOne({ username: user }).exec();
    if (!foundUser) {
        return res.sendStatus(401); // Unauthorized response if user not found
    }

    // Securely compares submitted password with stored hashed password using bcrypt
    const match = await bcrypt.compare(pwd, foundUser.password);
    if (match) {
        // Creates a session if login is successful, storing user details like ID, username, and roles
        req.session.user = {
            id: foundUser._id,
            username: foundUser.username,
            roles: foundUser.roles // Stores user roles for authorization checks
        };

        res.json({ message: 'Login successful!' }); // Confirms successful login
    } else {
        res.sendStatus(401); // Unauthorized if password doesn't match
    }
};

// Middleware function to check if the user is authenticated
// - Ensures only authenticated users can access protected routes by checking session data.
// - Proceeds to the next middleware/route if session exists, otherwise returns Unauthorized.
const isAuthenticated = (req, res, next) => {
    if (req.session?.user) {
        next(); // User is authenticated, proceed to the next middleware
    } else {
        res.sendStatus(401); // Unauthorized if no session exists
    }
};

// Logout handler to destroy user session and clear session cookie
// - Ensures session data is removed from MongoDB and session cookie is cleared from the client.
const handleLogout = (req, res) => {
    req.session.destroy(err => {
        if (err) {
```

```
            return res.status(500).json({ 'message': 'Failed to log out.' }); // Server error on logout
failure
        }
        res.clearCookie('connect.sid'); // Clears the session cookie from the client side
        res.json({ message: 'Logout successful!' }); // Confirms successful logout
    });
};

// Exporting functions and middleware for use in other parts of the application
module.exports = { sessionMiddleware, handleLogin, isAuthenticated, handleLogout };
```

In this code snippet, the session ID cookie should be marked as HttpOnly and Secure to prevent access via JavaScript and ensure it is only sent over HTTPS, respectively.(httpOnly: true, // Not accessible by JavaScript)

JWT-based authentication code implementation

Now, let's take a look at an example of implementing login with JWT authentication using the Express framework.

```
// Importing required modules
const User = require('../model/User');
const jwt = require('jsonwebtoken');

// Asynchronous function to handle user login
const handleLogin = async (req, res) => {
    // Extract cookies from the request, if any
    const cookies = req.cookies;

    // Extract username and password from request body
    const { user, pwd } = req.body;
 // Validation: If either field is missing, return HTTP 400
    if (!user || !pwd) return res.status(400).json({ 'message': 'Username and password are required.'
});



    // Check if a user with the provided username exists in the database
    const foundUser = await User.findOne({ username: user }).exec();

    // If user not found, return HTTP 401 Unauthorized
    if (!foundUser) return res.sendStatus(401);

    // Direct comparison of the plaintext password with the stored password (for demo purposes
only; hashing should be used in production)
    const match = pwd === foundUser.password;
    if (match) {
        // If passwords match, retrieve user roles, filtering out any falsy values
        const roles = Object.values(foundUser.roles).filter(Boolean);

        // JWT Creation: Access Token
        const accessToken = jwt.sign(
            {
```

```
            "UserInfo": {
                "username": foundUser.username, // Embeds username and roles in the token
                "roles": roles
            }
        },
        // Secret key for signing the token
        process.env.ACCESS_TOKEN_SECRET,
         // Short-lived token for enhanced security (expires in 60 seconds)
        { expiresIn: '60s' }
    );

    // JWT Creation: Refresh Token
    const newRefreshToken = jwt.sign(
        // Only the username is embedded in the refresh token for a lighter payload
        { "username": foundUser.username },
        process.env.REFRESH_TOKEN_SECRET, // Secret key for signing the refresh token
        { expiresIn: '1d' } // Refresh token valid for 1 day
    );

    // If there is an existing refresh token in cookies, filter it out from the user's stored tokens
    let newRefreshTokenArray =
        !cookies?.jwt
            ? foundUser.refreshToken // If no cookie token is found, retain all refresh tokens
            : foundUser.refreshToken.filter(rt => rt !== cookies.jwt); // Remove existing token to
avoid duplication

    // If a refresh token cookie exists, clear it and check if it was stored in the database (indicating
reuse)
    if (cookies?.jwt) {
        const refreshToken = cookies.jwt;
        const foundToken = await User.findOne({ refreshToken }).exec();

        // Token reuse detection: If the cookie token is absent in the database, clear all refresh
tokens
        if (!foundToken) {
            newRefreshTokenArray = []; // If token reuse detected, empty the refresh token array
        }

        // Clear the existing refresh token cookie
        res.clearCookie('jwt', { httpOnly: true, sameSite: 'Strict' });
    }

    // Save the new refresh token in the user's token array in the database
    foundUser.refreshToken = [...newRefreshTokenArray, newRefreshToken];
    const result = await foundUser.save();

    // Send the new refresh token in a secure, HTTP-only cookie to the client
    res.cookie('jwt', newRefreshToken, { httpOnly: true, secure: true, sameSite: 'Strict', maxAge:
48 * 60 * 60 * 1000 });

    // Send the access token back in the JSON response, allowing client to use it for authorization
```

```
      res.json({ accessToken });

    } else {
      // If passwords do not match, respond with HTTP 401 Unauthorized
      res.sendStatus(401);
    }
}

// Export the handleLogin function for use in routing or controllers
module.exports = { handleLogin };
```

Also in this code snippet, the specific attributes (httpOnly, secure, sameSite, and maxAge) ensure that the refresh token is stored and transmitted securely, mitigating risks such as XSS and ensuring compatibility with cross-site requests [4].

The code implementations in both cases demonstrate the practical deployment of each approach, setting a foundation for evaluating real-world efficacy in secure data handling and user session management. This dual approach aims to offer a nuanced understanding of how traditional session-based mechanisms compare with JWT in terms of performance, security, and user management—a critical assessment for advancing secure authentication practices in e-government platforms.

In the following section, the advantages and disadvantages of each method will be analyzed, focusing on scalability, vulnerability to specific threats, user experience and their alignment with evolving requirements of e-government 2.0 environments.

6. Research results

This study has evaluated two primary authentication methods—session-based and JWT-based authentication—within the context of e-Government 2.0, with particular emphasis on security, scalability, control, and user experience. The following results summarize the key distinctions and considerations that emerged from implementing and analyzing each approach.

6.1 Session-Based Authentication

Implementation Simplicity and Control:

Session-based authentication is known for its straightforward implementation, particularly within traditional server-side frameworks. This simplicity makes it a preferred choice for developers, as it requires minimal configuration and leverages established methods within the framework for session handling.

Additionally, this approach offers a significant degree of control to the server. Since session data is stored on the server side, administrators retain full authority over session lifecycle management. This includes the ability to invalidate sessions selectively, monitor session activity, and apply security policies in real time. These control features are particularly beneficial in environments where security compliance or dynamic access control is critical.

While session-based authentication is simple and straightforward to implement, it has inherent limitations, especially in terms of scalability and security.

Scalability Issues and user experience

Storing sessions on the server can become problematic in distributed systems or high-traffic applications which can impact performance and user experience in these scenarios. Since it requires session data synchronization across multiple servers. This can lead to several challenges:

Load Balancing**:** In a load-balanced environment, where multiple servers handle requests, ensuring that a user's session data is available to whichever server handles their request is crucial.

Session Stickiness**:** One common workaround is implementing session stickiness (also known as "sticky sessions" or "session affinity"), where a load balancer routes a user's requests to the same server. This approach, however, has significant drawbacks:

It reduces the effectiveness of load balancing and fails to handle server failures gracefully, as sessions are tied to specific servers.

Centralized Session Storage: Another workaround is using a centralized session storage solution, such as a database or a distributed cache like Redis or Memcached, to store session data. But this introduces additional complexity in managing and maintaining the session store. It also creates a single point of failure and potential performance bottleneck, which can impact the overall scalability and reliability of the application [5].

Vulnerability to CSRF

Cross-Site Request Forgery (CSRF) attacks exploit the fact that the browser automatically includes cookies in requests. Proper CSRF mitigation strategies must be implemented to protect against such attacks. One effective workaround is to use anti-CSRF tokens. These tokens are unique to each session and must be included in every form submission or state-changing request. But implementing and managing anti-CSRF tokens adds to the development complexity. Developers must ensure that tokens are correctly generated, included in forms, and validated on the server-side. Mistakes in this process can lead to vulnerabilities or functional issues. As another workaround, setting the SameSite attribute on cookies can mitigate CSRF risks by restricting how cookies are sent with cross-site requests. While this reduces CSRF risks, it may affect legitimate cross-site interactions, such as third-party integrations, and requires thorough testing to ensure it does not break existing functionality [14].

6.2 JWT-Based Authentication

Security:

JWT-based authentication provides a stateless mechanism that eliminates the need for server-side session storage, thus reducing the risk of certain attacks, such as session hijacking and server-side session fixation. This stateless nature also simplifies server management, as JWT tokens are self-contained and include all necessary claims and permissions, allowing for easy token validation without reliance on stored session data. Additionally, the common use of access and refresh tokens further enhances security by limiting the lifespan of access tokens and reducing the risk of unauthorized access. Access tokens are typically short-lived, meaning they expire quickly, while refresh tokens—stored securely client-side—can be used to obtain new access tokens without requiring the user to reauthenticate, thus minimizing exposure to potential attacks like Cross-Site Scripting (XSS) through the use of HTTP-only cookies for refresh token storage. [3,12].

Scalability and User Experience:

JWT-based authentication is inherently scalable due to its stateless design, making it ideal for distributed and high-traffic applications such as those found in e-Government 2.0 platforms, where scalability and security are of utmost importance [1]. Since tokens are self-contained and do not require centralized session storage, JWT-based systems allow for easy horizontal scaling across multiple servers without the need for session synchronization or the risk of centralized bottlenecks. This server-agnostic architecture enables any server in the network to validate tokens and process requests seamlessly, supporting a smoother user experience even as demand scales.

Limited Control over Token Invalidation:

Unlike traditional session-based authentication, where sessions can be invalidated server-side, JWTs are stateless and do not rely on server storage, making immediate token revocation challenging. If a token is compromised, there is no built-in mechanism to invalidate it until it expires. Although this can be managed with short-lived access tokens and centralized revocation lists, implementing and maintaining these solutions can be complex, especially in high-scale systems.

Security Risks of Long-Lived Refresh Tokens:

The use of refresh tokens, while beneficial for maintaining continuous sessions, also introduces risks. If a refresh token is compromised, an attacker can request new access tokens, potentially maintaining access over a long period. To mitigate this, refresh tokens must be stored securely, often in HTTP-only cookies, and applications may need to implement additional protections, such as token rotation and IP/device tracking, to identify and revoke suspicious activity. These extra security measures add complexity to the implementation and increase the burden on development teams.

Summary of Comparative Analysis

In summary, both session-based and JWT-based authentication methods have distinct strengths and limitations. Session-based authentication offers high control and ease of implementation but faces scalability and security challenges in distributed environments, requiring additional configuration and management efforts. In contrast, JWT-based authentication is inherently more scalable and better suited to high-traffic, distributed applications typical of e-Government 2.0, though it introduces token invalidation complexities and additional security management requirements for refresh tokens. [17]

7. Prospects for Further Development of Research

As e-Government 2.0 platforms evolve to accommodate growing security demands, higher user volumes, and more complex distributed architectures, further research is essential to address the limitations and potential enhancements of session-based and JWT-based authentication models. Several promising areas for continued investigation include:

1. **Advanced Token Revocation Mechanisms**:

Given the challenges associated with JWT token invalidation, developing enhanced revocation methods, such as real-time revocation checks and hybrid token schemes, could mitigate risks associated with compromised tokens. Solutions involving decentralized revocation databases or blockchain-based revocation systems may offer viable avenues for overcoming the limitations of current token-based authentication in large-scale e-Government environments.

2. **Adaptive Security Models**:

Traditional and token-based authentication methods could benefit from adaptive security techniques that adjust to user behavior and environmental factors in real-time. By implementing machine learning and AI-driven models that continuously monitor and respond to suspicious activities, future research can explore the potential for smarter, behavior-based authentication systems that dynamically adjust token expiration or session lifetime based on risk levels.

3. **Hybrid Authentication Architectures**:

Integrating the benefits of both session and token-based methods into a hybrid model could address the scalability limitations of session-based systems while maintaining some degree of server-side control over user sessions. This research direction may investigate adaptive hybrid approaches, where session-based and token-based mechanisms are used selectively based on user type, access level, or activity.

4. **Improved Anti-CSRF Techniques and Secure Token Storage**:

Further studies could focus on enhancing CSRF protection strategies within session-based systems to better mitigate risks without impacting legitimate cross-site interactions. Similarly, research into more secure client-side storage options for JWT refresh tokens—such as hardware-

based or cryptographic storage solutions—can help secure long-lived tokens, addressing one of the critical vulnerabilities in JWT-based authentication.

5. **Scalability and Performance Optimization in High-Traffic Applications**:

As demand grows for high-performance, low-latency authentication in government platforms, research on performance optimizations for both session and token-based systems is warranted. This might include exploring distributed caching, advanced load balancing algorithms, or efficient token verification mechanisms that reduce the processing overhead associated with handling large numbers of users across distributed networks.

Expanding on these areas could lead to significant improvements in authentication security, scalability, and user experience for e-Government applications, ensuring they remain resilient against evolving cyber threats and adaptable to new technologies.

8.Conclusion

This study has compared session-based and JWT-based authentication within the context of e-Government 2.0, focusing on security, scalability, and control. Both methods demonstrate unique strengths and weaknesses that must be evaluated carefully when selecting an appropriate authentication mechanism for government applications.

Session-based authentication offers a high degree of control and ease of implementation, making it a reliable choice for applications requiring strict server-side control and real-time session management. However, its limitations in scalability and susceptibility to CSRF attacks pose significant challenges, particularly in distributed and high-traffic environments. Ensuring secure and efficient session handling across multiple servers demands additional solutions, such as centralized storage, which may introduce complexity and performance constraints.

JWT-based authentication, by contrast, is stateless and highly scalable, aligning well with the demands of modern e-Government platforms. Its architecture facilitates horizontal scaling, enhances user experience, and minimizes server load by eliminating the need for centralized session storage. However, the challenges of token invalidation and the security risks associated with long-lived refresh tokens require careful management and advanced security measures, particularly in high-security environments.

In conclusion, selecting an authentication model for e-Government systems necessitates a balanced consideration of both security and performance requirements. Conventional session-based authentication methods, while effective in many scenarios, may not be sufficient to address the unique demands of modern, large-scale, and distributed government systems. This is where advanced techniques, such as JSON Web Token (JWT) based authentication, come into play, offering enhanced security, scalability, and flexibility [6,16]. Future research and development efforts focused on hybrid authentication models, advanced token revocation techniques, and adaptive security measures could yield novel solutions that leverage the strengths of both methods, contributing to the evolution of secure, scalable authentication in e-Government 2.0.

References:

1) Bucko, A., Vishi, K., Krasniqi, B., & Rexha, B. (2023). Enhancing JWT authentication and authorization in web applications based on user behavior history. *Computers*, 12(4), 78. https://doi.org/10.3390/computers12040078

2) Conklin, A., & White, G. B. (2006). e-Government and cyber security: The role of cyber security exercises. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)* (pp. 1–7). Kauai, HI, USA. https://doi.org/10.1109/HICSS.2006.133

3) Melesse, A. (2023). Enhancing REST API access control using multiple factor authentication with refresh token. *University of Texas at Dallas Electronic Theses and Dissertations*. https://hdl.handle.net/10735.1/10036

4) Akanksha, & Chaturvedi, A. (2022). Comparison of different authentication techniques and steps to implement robust JWT authentication. In *2022 7th International Conference on Communication and Electronics Systems (ICCES)* (pp. 1–5). Coimbatore, India. https://doi.org/10.1109/ICCES54183.2022.9835796

5) Braun, B., Kucher, S., Johns, M., & Posegga, J. (2012). A user-level authentication scheme to mitigate web session-based vulnerabilities. In *Information Security Practice and Experience (ISPEC 2012)* (pp. 12–24). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-32287-7_2

6) Medjahed, B., Rezgui, A., Bouguettaya, A., & Ouzzani, M. (2003). Infrastructure for e-government web services. *IEEE Internet Computing*, 7(1), 58–65. https://doi.org/10.1109/MIC.2003.1167340

7) Tolbert, C. J., & Mossberger, K. (2006). The effects of e-government on trust and confidence in government. *Public Administration Review*, 66(3), 354–369. https://doi.org/10.1111/j.1540-6210.2006.00594.x

8) Fan, J., & Yang, W. (2015). Study on e-gov services quality: The integration of online and offline services. *Journal of Industrial Engineering and Management*, 8(3), 693–718. https://doi.org/10.3926/jiem

9) Dolnák, I., & Litvik, J. (2017). Introduction to HTTP security headers and implementation of HTTP strict transport security (HSTS) header for HTTPS enforcing. In *2017 15th International Conference on Emerging eLearning Technologies and Applications (ICETA)* (pp. 1–6). Stary Smokovec, Slovakia. https://doi.org/10.1109/ICETA.2017.8102478

10) Shah, I. A., Habeeb, R. A., Rajper, S., & Laraib, A. (2022). The influence of cybersecurity attacks on e-governance. In *Cybersecurity Measures for E-Government Frameworks* (pp. 77–95). IGI Global. https://doi.org/10.4018/978-1-7998-9624-1.ch005

11) Zhao, J. J., & Zhao, S. Y. (2010). Opportunities and threats: A security assessment of state e-government websites. *Government Information Quarterly*, 27(1), 49–56. https://doi.org/10.1016/j.giq.2009.07.004

12) Kubovy, J., Huber, C., Jäger, M., & Küng, J. (2016). A secure token-based communication for authentication and authorization servers. In *Advances in Service-Oriented and Cloud Computing (ESOCC 2016)* (pp. 237–250). Springer, Cham. https://doi.org/10.1007/978-3-319-33313-7_19

13) Melitski, J., Holzer, M., Kim, S., Kim, C., & Rho, S. (2005). Digital government worldwide: A e-government assessment of municipal web sites. *International Journal of Electronic Government Research (IJEGR)*, 1(1), 1–18. https://doi.org/10.4018/jegr.2005010101

14) Peguero, K., & Cheng, X. (2021). CSRF protection in JavaScript frameworks and the security of JavaScript applications. *Human-Centric Computing and Information Sciences*, 1(2), 100035. https://doi.org/10.1016/j.hcc.2021.100035

15) Zolotukhin, M., Hämäläinen, T., Kokkonen, T., & Siltanen, J. (2014). Analysis of HTTP requests for anomaly detection of web attacks. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing* (pp. 1–7). Dalian, China. https://doi.org/10.1109/DASC.2014.79

16) Choejey, P., Fung, C. C., Wong, K. W., Murray, D., & Xie, H. (2015). Cybersecurity practices for e-government: An assessment in Bhutan. In *The 10th International Conference on e-Business (iNCEB2015)* (pp. 1–5). Bangkok, Thailand.

17) Ahmed, S., & Mahmood, Q. (2019). An authentication based scheme for applications using JSON web token. In *2019 22nd International Multitopic Conference (INMIC)* (pp. 1–5). Islamabad, Pakistan. https://doi.org/10.1109/INMIC48123.2019.9022766

18) Singh, S., Kumar, V., Paliwal, M., Verma, P., & Rajak, B. (2022). A citizen-centric approach to understand the effectiveness of e-government web portals: Empirical evidence from India. *Information Polity*, 27, 539–555. https://doi.org/10.3233/IP-220001