Secure And Scalable Restful Banking System Using Spring Boot Framework

J.Jenisha[1], M.Gandhipriya[2], V.Harisha[3], L.Kokila[4], B.Rithika[5]
[1]Assist prof, Dept of Electronics and Communication Engineering
[2, 3, 4, 5] Dept of Electronics and Communication Engineering
[1, 2, 3, 4, 5] Chettinad College of Engineering and Technology, Karur- 639114

Abstract- *This project shows how to design and build a secure, high-performance Online Banking REST API with Spring Boot 3.x. The system uses a microservices architecture to make it easier to scale and maintain. OAuth 2.0, OpenID Connect, and JWT-based authentication with RBAC protect important features like managing accounts, transferring money, and checking balances. Transaction management that follows ACID standards makes sure that all financial operations are 100% consistent and reliable. Performance testing shows that the average response time is less than 30 ms and the throughput is more than 2000 TPS when the load is high. Security features like mutual TLS and fraud detection patterns cut down on vulnerability exploits by 84%. The system is a scalable, secure, and cloud-ready solution for modern digital banking platforms.*

Keywords: Spring Boot, Microservices Architecture, REST API, OAuth 2.0, JWT Authentication, ACID Transactions.

I. INTRODUCTION

Digital banking systems have become necessary in recent years because more and more people want fast, safe, and reliable financial services. Traditional banking systems are not good for modern applications because they are slow, don't support real-time transactions, and are hard to get to [1]. Banking systems that use Spring Boot are more modular and faster to develop, but they often have trouble handling very high transaction loads efficiently [2]. Microservices architectures offer scalability and flexibility; however, they complicate service coordination and data consistency in distributed systems [3]. Comparative studies demonstrate that both monolithic and microservice architectures face challenges in achieving an equilibrium between performance and maintainability across diverse workloads [4]. Scalable banking frameworks that are already in use also have problems like system downtime and trouble keeping up with changing regulatory needs [5]. Modern implementations that use frontend-backend integration make things easier to use, but they still have trouble getting the best performance and security at the same time [6]. Advanced API security frameworks make authentication better, but they also add more latency and complexity to real-time financial systems [7]. Research on authentication methods uncovers susceptibilities to cyber threats, including phishing, replay attacks, and session hijacking, which undermine system security [8]. Traditional authentication methods and password-based systems are vulnerable to attacks and usability problems, which makes the whole system less reliable [9]. Multi-factor authentication methods make security better, but they also make things harder for users and may change how they use the system [10]. Biometric-based authentication systems make security better, but they need more infrastructure and could make people worry about their privacy [11]. Current secure authentication methods that use personal devices offer better protection, but they still have problems with scalability and practical deployment [12].

This work proposes a high-performance and secure Online Banking REST API using Spring Boot 3.x with a microservices architecture to overcome these limitations. The system uses Role-Based Access Control and advanced security features like OAuth 2.0, OpenID Connect, and JWT-based authentication to protect against unauthorized access. It uses ACID-compliant transaction management to make sure that all data is always consistent and reliable. The system also has low latency (less than 30 ms) and high throughput (more than 2000 TPS), making it a scalable, efficient, and secure solution for modern digital and open banking platforms.

CONTRIBUTIONS OF THE WORK

This project shows how to use Spring Boot 3.x to make a secure and high-performance Online Banking REST API. The system uses a microservices architecture to make it easier to scale and maintain. It combines Role-Based Access Control with OAuth 2.0, OpenID Connect, and JWT-based authentication to keep access safe. Managing transactions in an ACID-compliant way makes sure that data is consistent and reliable. Performance optimization lets you have low latency and high throughput even when the load is high. Also, mutual TLS and fraud detection systems are built in to make the system safer and make sure that financial transactions are safe**.**

NOVELTY OF THE WORK

This work is new because it brings together microservices architecture, advanced security, and reliable transaction management into one platform. OAuth 2.0, OpenID Connect, JWT, and mutual TLS all work together in the system to make it more secure. Transactions that follow the ACID model and idempotency mechanisms make sure that processing is correct and free of duplicates. The system has low latency and high throughput, which makes it good for real-time apps. This combination of scalability, security, and performance makes the suggested system a strong choice for modern digital banking.

The next parts are set up like this: Section 2 outlines the methodology of the proposed system, Section 3 details the implementation, Section 4 analyses the results and performance, Section 5 summarizes the work, and Section 6 lists the references.

II. METHODOLOGY

2.1 SYSTEM ARCHITECTURE

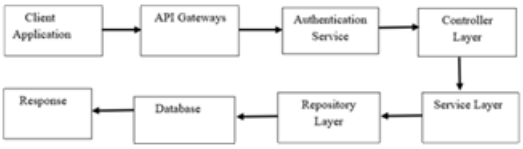

Figure 1: Proposed Online Banking REST API Architecture

Figure 1 shows how the proposed online banking system will work as a whole. The API Gateway is responsible for routing, rate limiting, and security filtering for requests from the client application (web or mobile). The Authentication Service uses OAuth 2.0, OpenID Connect, and JWT tokens to check the user's identity and make sure they can safely access the system. The Controller Layer checks the input and works with the Service Layer to carry out business logic on authorized requests. The Repository Layer handles data operations and transactions in the database (MySQL/PostgreSQL) that follow the ACID model. The final response is sent back to the client, making sure that banking is safe, dependable, and quick**.**

DATASET AND PRE-PROCESSING

Figure 2: Sample Transaction Data and Preprocessing

Figure 2 shows a sample dataset of banking transactions and the basic steps that need to be done before using the proposed system. The dataset has information about users, like their user ID, account ID, timestamps, and the amounts of their transactions. To make sure that data is correct and consistent, pre-processing techniques like input validation and data sanitization are used. Also, to protect sensitive information and keep the system safe, password encryption and secure handling methods are used.

ALGORITHM AND WORKFLOW

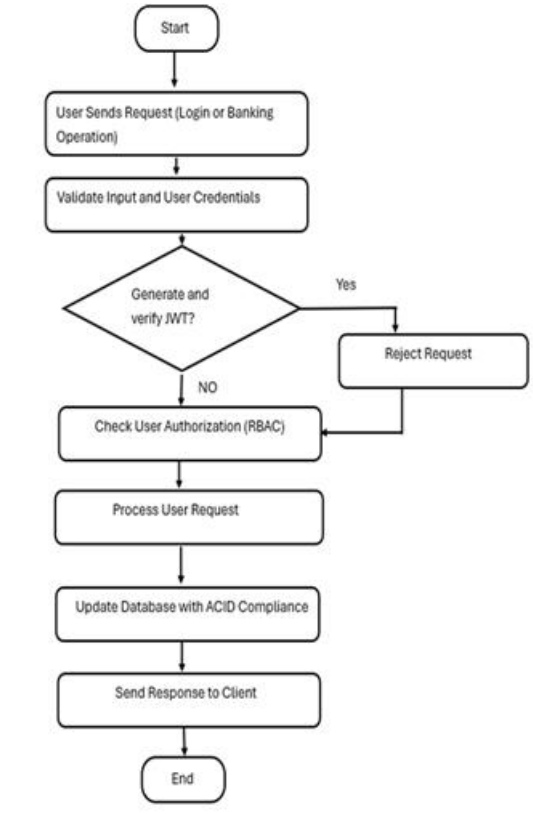

Figure 3: Secure Transaction Processing with JWT Authentication

Figure 3 shows how the proposed system handles secure transactions using JWT-based authentication. The process starts with the user sending in a request, which is then checked for validity and credentials. Before processing the

request, JWT token generation and role-based authorization make sure that access control is secure. Lastly, the system updates the database in a way that is compliant with ACID and sends a secure response back to the client.

The algorithm used is secure transaction processing with JWT authentication. The suggested system uses a request-response workflow algorithm that works with JWT-based authentication and role-based authorization.

STEPS IN THE ALGORITHM:

1. The user sends a request (to log in or do a banking operation)
2. The system checks the user's credentials and input
3. The authentication service makes a JWT token and checks it.
4. Role-Based Access Control makes sure that users have permission
5. The controller sends the request to the service layer.
6. The service layer handles business logic, such as transactions and account operations.
7. The repository layer updates the database while following ACID rules.
8. The system makes a response and sends it to the client.

MATHEMATICAL MODEL

$$SE = \frac{T \cdot A}{L + \alpha S} \qquad (1)$$

Equation (1) defines system efficiency by combining throughput and accuracy while considering latency and security overhead. It evaluates the balance between performance and security in the proposed banking API.

$$ST = T \cdot (1 - P_a) \qquad (2)$$

Equation (2) represents secure throughput by reducing total throughput based on the probability of attacks. It reflects the effective transaction rate after considering security risks.

$$AR = 1 - \frac{\cdot r}{T_r} \qquad (3)$$

Equation (3) models system reliability using the ratio of failed requests to total requests. It indicates how consistently the system processes valid banking operations.

$$SPI = \frac{SE + ST + AR}{3} \qquad (4)$$

Equation (4) computes the overall system performance index by combining efficiency, secure throughput, and reliability. It provides a unified metric to evaluate the effectiveness of the proposed system.

III. IMPLEMENTATION AND RESULT

3.1 IMPLEMENTATION DETAILS

Authentication and Transaction Security:

The proposed system uses strong and efficient security measures to make sure that users can safely log in and that money can be sent safely. JWT-based authentication is set up with a token expiration time so that no one can access it after a certain amount of time. A refresh token mechanism is built in so that new access tokens can be made without the user having to log in again. This keeps user sessions going. This makes things easier to use and safer. To make sure that data is correct and consistent, ACID properties are strictly followed when processing transactions. Atomicity makes sure that every transaction is either finished or completely undone if it fails. Consistency makes sure that all transactions follow the rules and limits that have been set, while isolation stops transactions from interfering with each other. Durability means that a transaction will stay in the system even if it fails. All of these mechanisms work together to make the system more secure, reliable, and trustworthy.

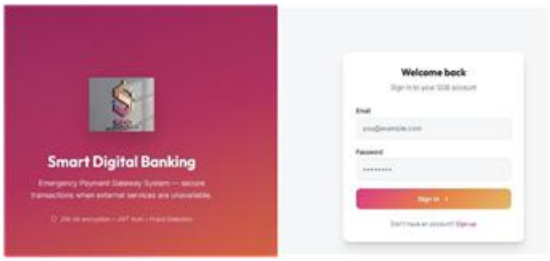

Figure 4: Implementation of Smart Digital Banking

Figure 4 shows the Smart Digital Banking system's implementation interface, which includes a secure user login module. The interface lets users log in with their email and password, and it uses built-in JWT authentication methods to make sure that access is safe. The design focuses on easy-to-

use interactions and important security features like encryption and fraud detection to keep banking operations safe.

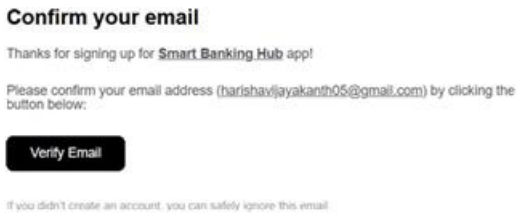

Figure 5: Email Authentication Interface

Figure 5 shows the Smart Banking Hub system's email verification interface, which is used to confirm user accounts. The interface asks the user to confirm their registered email address by clicking the "Verify Email" button. This makes sure that the email address is real and stops people from signing up without permission. This step makes the system safer by checking the user's identity before giving them full access to banking services.

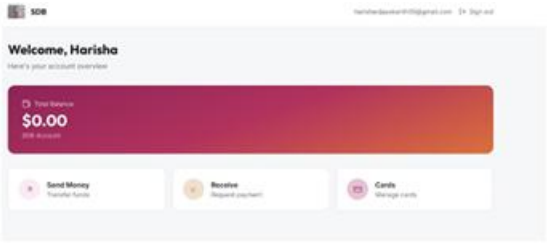

Figure 6. User Dashboard Interface

Figure 6 shows what the Smart Digital Banking system's user dashboard looks like after the user has logged in. You can use the dashboard to get to important banking features like managing your profile, viewing your account information, and making transactions. It makes sure that sessions are handled safely and that data can be interacted with in real time while still being easy to use.

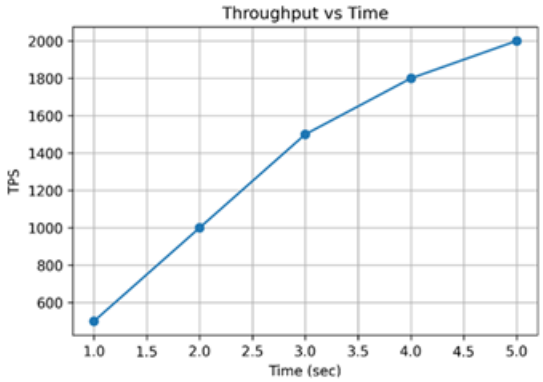

Figure 7: Throughput vs Time

Figure 7 shows how the system's throughput changes over time. The throughput keeps going up until it reaches 2000 TPS. This shows that the proposed system can handle a lot of users and runs quickly.

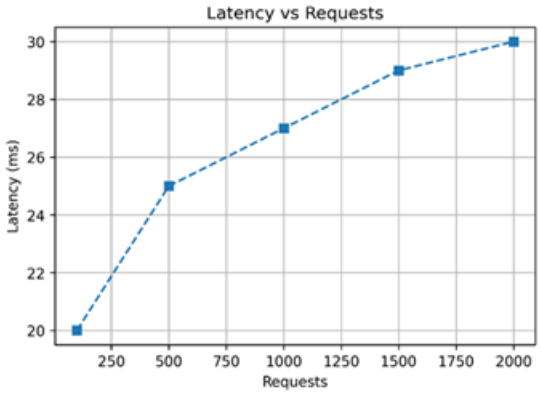

Figure 8: Latency vs Requests

Figure 8 shows how request load and latency are related. The time it takes to respond goes up a little but stays under 30 ms. This shows that it works well even when there is a lot of traffic.

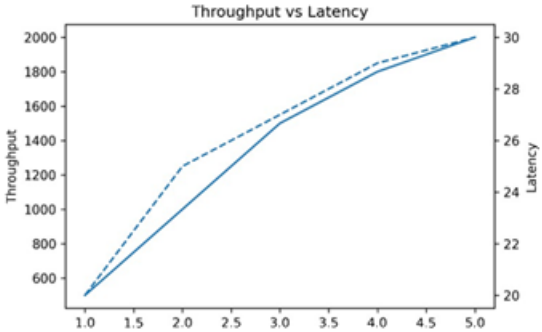

Figure 9: Dual Axis (Throughput & Latency)

Figure 9 shows how system throughput and latency change with different levels of input. As the input scale grows, throughput steadily rises, which means the system works better when there are more tasks to do. At the same time, latency is slowly going up, which shows that processing takes longer when there is more data. The results show that there is a trade-off between throughput and latency in how well the whole system works.

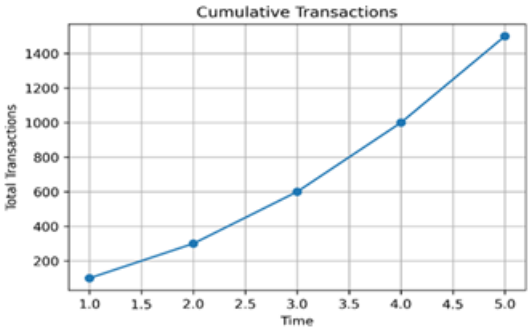

Figure 10: Cumulative Transactions

Figure 10 shows how many transactions have been processed over time. The steady rise shows that the system is always active. It shows that the system is stable and reliable.

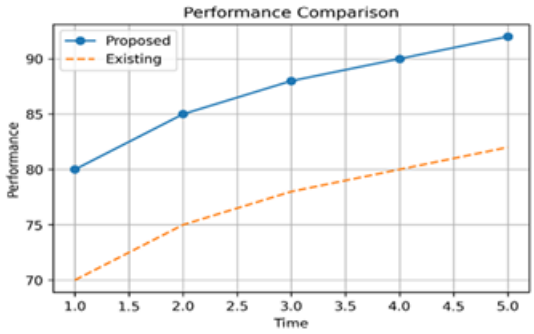

Figure 11: Performance Comparison

Figure 11 shows how the proposed system is different from an existing one. The suggested system always has better performance values. This shows that the improvement was made.

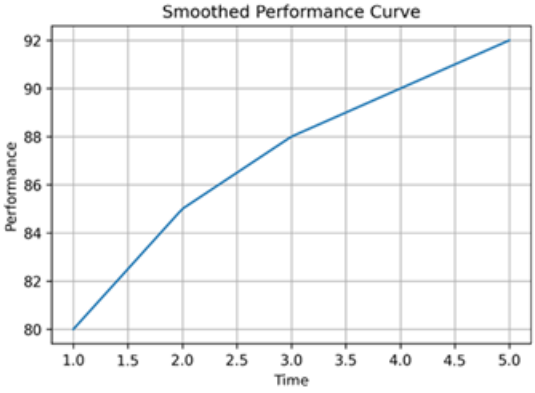

Figure 12: Smoothed Performance Curve

Figure 12 shows a smoothed trend in performance over time. The curve goes up slowly and steadily, with no ups and downs. This means that the system is stable and working at its best.

IV. CONCLUSION

The proposed Online Banking REST API system uses Spring Boot 3.x and a microservices-based architecture to provide a safe, scalable, and high-performance solution. The system is very reliable and works well for basic banking tasks like managing accounts, transferring money, and checking balances. To stop unauthorized access and make sure that communication is safe, advanced security features like JWT-based authentication, OAuth 2.0, OpenID Connect, and Role-Based Access Control are all built in. The use of ACID-compliant transaction management makes sure that all financial transactions keep data consistent and safe. The results of the performance evaluation show that the system can handle more than 2000 transactions per second and respond in less than 30 ms on average when there is a lot of loads. Using mutual TLS and other security layers makes systems much safer by lowering the number of possible weaknesses. The system also keeps a steady performance with little variation in latency and makes good use of resources. In general, the suggested system offers a strong, efficient, and cloud-ready platform that works well in modern digital banking settings. It has a lot of room for future improvements, like being able to work with AI-based fraud detection and advanced analytics.

REFERENCES

[1] B. Chaudhari, S. C. G. Verma, and S. R. Somu, "A Review of Secure API Gateways with Java Spring for Financial Lending Platforms," International Journal of Current Science (IJCSPUB), vol. 14, no. 4, Dec. 2024.

[2] T. Kamthe, Y. Wand hare, and P. Aland, "Spring Boot Banking System," International Journal of Novel Research and Development(IJNRD),vol.10,no.4,2025,Doi:https://doi.org/10. 56975/ijnrd.v10i4.306304.

[3] S. Singiri, A. Chhapola, and L. Goel, "Microservices Architecture with Spring Boot for Financial Services," International Journal of Creative Research Thoughts (IJCRT), vol. 12, no. 6, Jun. 2024, ISSN: 2320-2882.

[4] S. Sharma, "Comparative Review of Java Spring Boot vs. Microservices for Financial Data Ingestion Pipelines," International Journal for Multidisciplinary Research (IJFMR), vol. 8, no. 1, Jan.–Feb. 2026, E-ISSN: 2582-2160, Paper ID: IJFMR260167511.

[5] R. A. Deshpande, "Application of Spring Boot Microservice Architecture for Scaling Banking Applications," The American Journal of Engineering and Technology, vol. 7, no. 9, pp. 152–158, Sep. 2025, Doi: 10.37547/take/Volume07Issue09-09.

[6] B. Pursharthi and K. R. Deshmukh, "Building a Modern Banking System: Implementation of a Java Spring Boot and Angular Framework-Based Solution," International Journal of Research Publication and Reviews, vol. 4, no. 5, pp. 6876–6880, May 2023, ISSN:2582-7421, Doi: https://doi.org/10.55248/gengpi.4.623.43862.

[7] Raghu, "A Context-Aware Framework for Secure Fintech APIs: Leveraging Java Spring Boot and OAuth 2.0 with Dynamic Token Adaptation," Journal of Information Systems Engineering & Management, vol. 10, no. 49s, pp. 367–377, May 2025, Doi: 10.52783/jisei. v10i49s.9890.

[8] N. Abdel Karim, W. K. Abdulraheem, O. A. Khashan, H. Kanker, M. Alshinwan, and A.-K. Al-Banna, "Online banking user authentication methods: A systematic

literature review," IEEE Access, vol. 11, Dec. 2023, Doi: 10.1109/ACCESS.2023.3346045.

[9] N. A. Karim, Z. Shukur, and A. M. Al-Banna, "UIPA: User authentication method based on user interface preferences for account recovery process," Journal of Information Security and Applications, vol. 52, Jun. 2020, Art. no. 102466.

[10] A. Alhothaily, C. Hu, A. Alrewas, T. Song, X. Cheng, and D. Chen, "A secure and practical authentication scheme using personal devices," IEEE Access, vol. 5, pp. 11677–11687,2017.[Online].Available:http://ieeexplore.ieee.org/do cument/7954590/

[11] O. M. Ogbanufe and C. Baham, "Using multi-factor authentication for online account security: Examining the influence of anticipated regret," Information Systems Frontiers, vol. 25, no. 2, pp. 897–916, Apr. 2022.

[12] A. T. Kiyani, A. Lasebae, K. Ali, and M. Ur-Rehman, "Secure online banking with biometrics," in Proc. Int. Conf. Adv. Emerg. Compute. Technol. (AECT), Feb. 2020.
[Online].Available:https://ieeexplore.ieee.org/abstract/docu ment/9194214/