ISSN: 2456-8503

International Journal for Development of Science and Technology (IJDST)

Secure online banking REST API

RESEARCH

Mathan Kumar[1], Nivash[1], Sathya Sri[1], Deepan[1], Senthil Prakash[1*]*

Abstract

Online banking has become an indispensable part of the modern financial ecosystem, necessitating systems that are not only highly functional but also rigorously secure and scalable. This paper presents a robust backend solution designed using Java Full Stack technologies, specifically the Spring Boot Framework, to replace traditional, monolithic banking software with a lightweight, decoupled RESTful architecture. The proposed system addresses critical security vulnerabilities inherent in legacy architectures by implementing Stateless Authentication using JSON Web Tokens (JWT) and Spring Security, effectively mitigating risks associated with Cross-Site Request Forgery (CSRF) and session-based attacks. Data integrity is mathematically guaranteed through ACID-compliant transaction management using the @Transactional architecture, ensuring that fund transfers are processed reliably without data loss during network failures. Furthermore, the application utilizes Spring Data JPA for efficient interaction with a MySQL database, providing built-in protection against SQL injection. This research demonstrates a production-ready approach to building secure financial applications, prioritizing modularity, maintainability, and advanced cryptographic data protection.

Keywords: *REST API, spring boot, JSON Web Token (JWT), stateless authentication, acid transactions, and cybersecurity.*

1. Introduction

The financial sector has undergone a massive paradigm shift in recent years, transitioning from traditional brick-and-mortar banking to highly accessible digital-first solutions. Online banking systems have become the backbone of the modern economy.

However, financial data remains a primary target for sophisticated cyberattacks, necessitating robust, secure, and highly scalable software architectures. Traditional monolithic systems rely heavily on stateful server-side session management (such as HTTP cookies), which significantly increases the server load during peak transaction hours and exposes the system to Cross-Site Request Forgery (CSRF) and session hijacking attacks. To address these modern challenges, this paper proposes a comprehensive backend solution titled Secure Online Banking REST API [1]. Built using Java Full Stack technologies and the Spring Framework ecosystem, the core of the application is a Representational State Transfer (REST) API that serves as a headless, secure interface between the underlying database and any potential frontend clients.

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

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

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

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

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

**Corresponding Author: jtysp14@gmail.com*

This journal is © ijdst 2026

The architecture leverages JSON Web Tokens (JWT) to ensure every API request is stateless and strictly authenticated, while passwords are mathematically salted and encrypted using the BCrypt hashing algorithm to protect sensitive credentials from database breaches [2].

2. Background and related work

2.1. Evolution of Spring Boot in Enterprise Architectures

Modern enterprise software development has shifted towards frameworks that reduce boilerplate code. The Java Spring Boot Framework employs a Convention over Configuration approach, making it highly superior for building scalable, multithreaded backend systems compared to lighter, less structured frameworks. For a banking system requiring high reliability and concurrency, the Spring ecosystem provides an industry-standard foundation capable of handling heavy transaction loads efficiently [3].

2.2. Stateless Security Models and JSON Web Tokens (JWT)

Traditional session-based authentication consumes heavy server memory and is highly susceptible to CSRF. Stateless security models utilizing JSON Web Tokens (JWT) resolve this by securely exchanging interoperable data between the client and server without storing session files locally. By placing the security state within a cryptographically signed token stored on the client side, the API remains lightweight and resistant to traditional session-based attacks [4].

2.3. Cryptographic Hashing and Transactional Integrity

Protecting sensitive data at rest is critical. Algorithms utilizing key stretching and salting, specifically BCrypt, provide a mathematically superior defense against brute-force and rainbow table attacks compared to older algorithms like MD5. BCrypt's adaptive hashing mechanism deliberately slows down the encryption process to thwart automated cracking tools. Furthermore, maintaining the ACID (Atomicity, Consistency, Isolation, Durability) properties in banking operations ensures that multi-step financial operations are executed flawlessly without data corruption or partial updates [5].

3. Proposed system architecture

The proposed system completely abandons the monolithic approach in favor of a Service-Oriented Architecture (SOA). It acts as a headless backend, communicating exclusively via lightweight JSON payloads. The implementation is structured around the Controller-Service-Repository Architecture, ensuring a strict Separation of Concerns [6].

Table 1: Key components of the proposed architecture

Component	Function	Implementation	Benefit
Authentication Module	Manages registration, login, token generation	Spring Security & JWT	Statelessprevents CSRF.
Account Module	Creates accounts and retrieves balances.	Custom Java Services	Prevents unauthoriz ed balance inquiries.
Transaction Module	Processes deposits, withdrawals, and transfers.	Spring @Transac-tional	Guarantees ACID properties; prevents overdrafts.
Data Persistence	Manages database schemas and CRUD operations	Spring Data JPA / Hibernate	Eliminates SQL Injection via parameterized queries.

3.1. Authentication Module

This is the system's security checkpoint, handling user registration and logins using Spring Security and JWT. By issuing stateless tokens instead of relying on server memory, it ensures the system is fast and protected against session hijacking and CSRF attacks [7].

3.2. Account Module

Acting as the ledger manager, this module uses Custom Java Services to open accounts and check balances. It ensures strict privacy by verifying the user's token ID against the account owner, preventing unauthorized access to other users' information.

3.3. Transaction Module

This is the financial engine that handles deposits, withdrawals, and transfers using Spring @Transactional. It strictly checks balances to prevent overdrafts and enforces database locks (ACID properties) so that if a transfer fails midway, the entire transaction is safely rolled back to prevent lost funds.

3.4. Data Persistence

This component manages how the application saves data to the MySQL database using Spring Data JPA / Hibernate. It acts as a secure translator, converting Java code into safe "parameterized queries" that completely neutralize the threat of SQL injection attacks (Table 1).

Table 2: Comparison with conventional systems

Feature	Proposed REST API Architecture	Legacy Monolithic system
Architecture Style	Service-Oriented Architecture (SOA)	Tightly Coupled Monolith
Authentication	Stateless (JSON Web Tokens)	Stateful (Server-Side Cookies)
Data Payload	Lightweight JSON	Heavy HTML/JSP Rendering
Scalability	High (Stateless Nodes)	Low (Memory Exhaustion)
SQL Protection	Automated Parameterized Queries (QPA)	Often vulnerable raw SQL queries

3.5. Architecture Style

The proposed system uses a Service-Oriented Architecture (SOA), meaning the backend (server) and frontend (user interface) are completely separate. A Monolithic System bundles everything together, making it harder to update or maintain without breaking the whole application.

3.6. Authentication

The new system is Stateless using JWT. The server gives the user a secure token and forgets them, saving server memory. Legacy systems are Stateful, meaning the server has to actively remember every logged-in user using cookies, which consumes a massive amount of RAM.

3.7. Data Payload

The proposed API sends data back and forth using Lightweight JSON (simple text data), making it incredibly fast. Legacy systems force the server to build and send Heavy HTML/JSP files for every single page load

3.8. Scalability

Because the new system is stateless and lightweight, its Scalability is High-you can easily add more servers to handle millions of users. Legacy systems have Low Scalability because their memory gets exhausted quickly during peak traffic hours.

3.9. SQL Protection

The proposed system uses Spring Data JPA to automatically sanitize database queries, making it immune to SQL Injection attacks. Legacy systems often rely on Raw SQL queries, which hackers can easily exploit to steal data (Table 2).

Table 3: Performance and security advantages

Parameter	Improvement Achieved
CSRF Protection	100% Neutralized due to absence of cookies
Data Security	Crypt adaptive hashing secures passwords at rest
Transaction Reliability	Atomic Rollbacks prevent financial discrepancies
Platform Independence	Seamless integration with React, iOS, Android

3.10. CSRF Protection

By replacing automatic browser cookies with JSON Web Tokens (JWT), the system prevents hackers from authorize fake transactions, making Cross-Site Request Forgery (CSRF) attacks impossible.

3.11. Data Security

Passwords stored in the database are protected using Crypt. This algorithm adds random data salt and intentionally slows down the encryption process, making it extremely difficult for hackers to crack stolen passwords via brute force.

3.12. Transaction Reliability

Fund transfers use atomic rollbacks to treat multi-step transactions as a single, unbreakable unit. If an error or server crash happens mid-transfer, the system automatically reverses the entire process to ensure no money is ever lost or duplicated.

3.13. Platform Independence

The new headless REST API communicates using universal JSON data instead of building specific HTML web pages. This allows the exact same backend code to seamlessly power websites, iOS apps, and android apps without needing to be rewritten (Table 3).

4. Implementation methodology

The system implementation utilizes a layered Java approach. The base project structure was generated using Spring Initializer, pulling dependencies for Spring Web, Spring Data JPA, Spring Security, and the MySQL Connector [8].

- *Routing:* The application's entry points were implemented using the @RestController annotation to bridge HTTP clients (like Postman) with the business logic. Data Transfer Objects (DTOs) were utilized with @RequestBody to automatically deserialize incoming JSON payloads.
- *Business Logic Persistence:* The TransactionService class executes the complex logic required for fund transfers. The @Transactional annotation was strategically applied to the transfer funds method to guarantee that deduction from the source and addition to the target occur as a single, atomic database transaction. Data access was handled by extending the JPA repository interface, providing out-of- the-box CRUD functionality.
- *Security:* A custom JWT Authentication filter intercepts every incoming HTTP request, extracts the JWT from the authorization header, and cryptographically verifies its signature. The Crypt Password Encoder bean ensures all plain-text passwords are automatically salted and hashed before database persistence [9].

5. Results and discussion

The system was rigorously tested using Unit Testing (JUnit 5, Mockito) and Integration Testing via Postman to simulate HTTP requests. In functional testing, POST requests to /api/auth/login successfully returned signed JWT strings upon providing valid credentials. For transaction testing, attempts to withdraw funds exceeding the account balance correctly resulted in an HTTP 400 Bad Request Insufficient Funds,

proving the efficacy of the validation logic. Valid fund transfers successfully deducted and added exact amounts across accounts seamlessly. In security and vulnerability testing, attempts to access protected endpoints without a JWT resulted in the expected 401 Unauthorized response. Furthermore, malicious SQL strings injected into the login payload were successfully neutralized, as the Hibernate ORM utilizes parameterized queries by default, proving the system's resilience against SQL injection [10][11].

6. Challenges and future scope

Despite the robustness of the stateless architecture, token management introduces complexity. Unlike server-side sessions that can be instantly destroyed, JWTs are valid until expiration, requiring additional architectural overhead (such as database blacklists) for immediate revocation. Furthermore, frontend clients must securely store the tokens to prevent Cross-Site Scripting (XSS) theft. Future iterations of this system will focus on Microservices Architecture migration, decoupling the monolith into independent, deployable services (Auth, Account, Transaction) managed by an API Gateway. Advanced security enhancements will include upgrading to an OAuth2 authorization framework and integrating Multi-Factor Authentication (MFA). Finally, integrating Artificial Intelligence (AI) and Machine Learning models to analyze real-time transaction payloads will enable automated detection of behavioral anomalies and fraud.

7. Conclusion

The Secure Online Banking REST API project successfully addresses the severe limitations of legacy banking systems by providing a lightweight, high-performance alternative built on a security-first mindset by shifting away from vulnerable, stateful session management and implementing a stateless authentication architecture using JSON Web Tokens (JWT) and Spring Security, the system effectively neutralizes common attack vectors such as CSRF. The utilization of Spring Data JPA and MySQL ensures reliable data persistence, while ACID-compliant transactions guarantee that financial ledgers remain completely accurate during simulated system failures. This architecture lays a formidable foundation for building enterprise-grade financial technology, proving that rigorous security engineering and efficient software architecture can seamlessly coexist.

Conflict of interest statement: The authors declare that there is no conflict of interest regarding the publication of this research paper.

Funding information: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Ethical approval statement: This study involves the development and evaluation of a software prototype and does not involve human participants or sensitive personal data.

Data availability statement: The data used in this study were generated during the testing phases of the project architecture and are available upon request.

Acknowledgement: The authors sincerely thank the Department of Computer Science and Engineering at Shree Venkateshwara Hi-Tech Engineering College, Gobi, for providing the academic support, guidance, and a conducive environment for the completion of this project and study.

References

1. Baeldung E, REST API with spring boot and security architecture, Baeldung Web Tutorials. 2024, 45-52
2. Bauer C, King G, Java persistence with hibernate and spring data, Manning Publications. 2015, 112-140
3. Bloch J, Effective java: Best practices for software engineering, Addison-Wesley Professional, 3rd Edition. 2018, 88-105

4. De Ryck P, Web security for developers: Real threats, practical defense, No Starch Press. 2022, 110-135
5. Gutierrez F, Pro spring boot 2: An authoritative guide to building microservices, Apress. 2021, 300-325
6. Hoffman J JSON Web Tokens (JWT) in action, Manning Publications. 2017, 40-65
7. Kleppmann M, Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems, O'Reilly Media. 2017, 220-250
8. Leonard A, Spring security in action: Defending REST APIs, Manning Publications. 2020, 210-235
9. Long J, Cloud native java: Designing resilient systems with spring boot, spring cloud, and cloud foundry, O'Reilly Media. 2021, 150-175
10. Schildt H, Java: The complete reference, McGraw-Hill Education, 12th Edition. 2021, 500-550
11. Walls C, Spring boot in action: Convention over Configuration, Manning Publications. 2018, 20-35