International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Blockchain Based Banking System

Design and Development of a Secure Blockchain-Based Banking System Using Smart Contracts

Amit Yadav[1]

Department of Computer Science, Patkar-Varde College, University of Mumbai

---

Abstract— Traditional banking systems rely on centralized databases and application servers to manage customer accounts and financial transactions, creating challenges related to data integrity, transaction transparency, security, and auditability. This paper presents the design and development of a secure Blockchain-Based Banking System that integrates blockchain technology and smart contracts with conventional web application technologies. The proposed system uses a React-based frontend, a Node.js and Express.js backend, MongoDB for customer information, authentication, Know Your Customer (KYC) data, and application-level transaction records, and Solidity smart contracts to manage financial account states and banking operations. The system supports role-based access for administrators, cashiers, and customers, enabling customer account management, KYC processing, deposits, withdrawals, fund transfers, and transaction history tracking. Web3.js facilitates communication between the backend and the blockchain network, allowing the application to retrieve authoritative account balances and execute smart-contract-based transactions. Blockchain transaction hashes and block numbers are associated with application records to support transaction traceability and verification. A transaction status management mechanism distinguishes Pending, Completed, and Failed operations and supports reconciliation when blockchain execution succeeds but the corresponding database update encounters an error. Sensitive customer and authentication information is maintained off-chain in MongoDB rather than being stored directly on the blockchain. The system is developed and tested using Ganache as a local blockchain environment, with Ethereum Sepolia considered for production-oriented testing and deployment. The proposed system demonstrates a hybrid architecture that combines conventional database management with blockchain-based financial state management to improve transaction integrity, traceability, and accountability in a banking application. The effectiveness of the implementation can be evaluated through functional testing, role-based access-control testing, transaction verification, and database–blockchain consistency checks.

Keywords— Blockchain, Banking System, Smart Contracts, Ethereum, Solidity, Web3.js, MongoDB, Transaction Integrity, Role-Based Access Control, Transaction Verification.

1. INTRODUCTION

The increasing adoption of digital technologies has transformed the way financial services are delivered, enabling customers to access accounts and perform financial transactions through web-based banking applications. Traditional banking systems commonly depend on centralized databases and application servers to maintain customer information, account balances, and transaction records. Although this architecture supports efficient data processing and centralized administration, it creates challenges related to data integrity, transaction transparency, unauthorized modifications, and the verification of financial records. Maintaining the accuracy and consistency of transaction information is particularly important because banking operations require reliable records of deposits, withdrawals, and fund transfers.

Blockchain technology offers an alternative approach to maintaining financial records through a distributed ledger in which transactions are recorded and validated according to the rules of the underlying network. Its cryptographic mechanisms and linked transaction history make unauthorized alteration of confirmed records more difficult to conceal. Furthermore, smart contracts allow predefined business rules to be executed on a blockchain, providing a programmable mechanism for managing account states and processing financial operations. Ethereum-compatible blockchain platforms support smart contracts written in Solidity, making it possible to integrate blockchain functionality into web-based banking applications.

Despite these advantages, incorporating blockchain into a banking application does not automatically guarantee complete security or eliminate every operational challenge. Smart contracts may contain programming vulnerabilities, blockchain transactions may fail or experience confirmation delays, and application records can become inconsistent if a blockchain operation succeeds but the corresponding database update fails. Moreover, storing all customer information directly on a public blockchain can create privacy concerns because blockchain records are designed to be persistent and publicly verifiable on public networks. Therefore, an effective blockchain-based banking system must carefully separate sensitive application data from financial state and provide a reliable mechanism for handling transaction outcomes.

To address these challenges, this paper presents the design and development of a Blockchain-Based Banking System that combines conventional web technologies with Solidity smart contracts. The proposed system uses a React-based frontend and a Node.js and Express.js backend, with MongoDB responsible for customer information, authentication data, Know Your Customer (KYC) records, and application-level transaction details. The blockchain smart contract maintains the authoritative financial account state, including account balances and account status, and supports banking operations such as deposits, withdrawals, and fund transfers. Web3.js

provides communication between the backend and the blockchain network, enabling the application to retrieve account balances and submit smart-contract transactions.

The system also incorporates role-based access control for administrators, cashiers, and customers. These roles provide access to functions according to their responsibilities, including customer account management, KYC processing, account information retrieval, and permitted banking operations. For transaction tracking, the application associates MongoDB transaction records with blockchain information such as transaction hashes and block numbers. A transaction status management mechanism distinguishes Pending, Completed, and Failed operations. If a blockchain transaction succeeds but the corresponding MongoDB update fails, the transaction can remain Pending until reconciliation is performed, rather than being incorrectly reported as a failed financial operation.

The main contributions of this work are summarized below:

- **Blockchain-based financial state management:** The system uses Solidity smart contracts to maintain authoritative account balances and process supported banking operations on an Ethereum-compatible blockchain.
- **Hybrid database and blockchain architecture:** MongoDB manages customer, authentication, KYC, and application-level transaction information, while the blockchain maintains financial account state and blockchain transaction records. This separation keeps sensitive customer information off-chain.
- **Role-based banking operations:** The application provides separate access for administrators, cashiers, and customers, supporting controlled access to customer management, KYC, and banking functions.
- **Transaction tracking and verification:** Transaction records are associated with blockchain hashes and block numbers, allowing users and authorized personnel to inspect the corresponding blockchain transaction information.
- **Transaction synchronization and error handling:** The system uses Pending, Completed, and Failed transaction statuses to represent processing outcomes and supports reconciliation when the blockchain and MongoDB updates do not complete together.
- **Practical implementation and evaluation:** The system integrates React, Node.js, Express.js, MongoDB, Solidity, and Web3.js. Ganache provides a local blockchain environment for development and testing, while Ethereum Sepolia can be used for production-oriented testing and deployment evaluation.

2. RELATED WORK

2.1 Blockchain-Based Financial Systems

Blockchain technology has emerged as an alternative approach to maintaining digital transaction records without relying exclusively on a centralized database. Nakamoto [1] introduced Bitcoin as a peer-to-peer electronic cash system that uses a distributed ledger and cryptographic mechanisms to record transactions. This work established an important foundation for blockchain-based financial applications by demonstrating how transaction verification and record integrity could be supported through a decentralized network.

Subsequent developments extended blockchain technology beyond cryptocurrency transactions. Buterin [2] introduced Ethereum as a platform supporting programmable smart contracts and decentralized applications. Smart contracts enable predefined rules to be executed on a blockchain, making them suitable for applications that require consistent processing of digital transactions. In a banking application, this capability can be applied to account management, balance updates, and financial operations such as deposits, withdrawals, and fund transfers.

Research has also examined the potential applications and limitations of blockchain technology across different domains. Yli-Huumo et al. [3] reviewed blockchain research and identified areas of opportunity as well as challenges related to scalability, performance, and practical adoption. Casino et al. [4] presented a systematic review of blockchain-based applications, highlighting the technology's potential while identifying unresolved issues affecting real-world implementation. These studies provide a foundation for investigating how blockchain can be integrated with conventional web technologies to improve the traceability and integrity of financial transaction records.

However, adopting blockchain alone does not address every requirement of a banking system. Customer information, authentication credentials, and Know Your Customer (KYC) records require appropriate privacy and access controls. Smart contracts may also contain vulnerabilities, and transactions must be handled carefully when blockchain execution and application-database updates do not complete together. Therefore, a practical blockchain-based banking application requires a suitable division of responsibilities between on-chain financial state management and off-chain application data storage.

Building on these concepts, the proposed Blockchain-Based Banking System combines Solidity smart contracts with a conventional web application and MongoDB. The blockchain maintains the authoritative financial account state and supports banking operations, while MongoDB stores customer information, authentication data, KYC records, and application-level transaction details. Web3.js connects the backend to the blockchain, and blockchain transaction hashes and block numbers support transaction tracking and verification. This hybrid architecture aims to combine blockchain-based financial record integrity with the flexibility of conventional database management.

2.2 Evaluation of Blockchain-Based Financial Systems

Several studies have examined the applications, capabilities, and limitations of blockchain technology in financial and transaction-management systems. Yli-Huumo et al. [3] reviewed existing blockchain research and identified opportunities as well as challenges involving scalability, performance, security, and practical implementation. Their review highlights the importance of evaluating blockchain applications beyond their basic functionality, particularly when they are intended to support financial operations.

Casino et al. [4] conducted a systematic review of blockchain-based applications across multiple domains. Their study examined the potential of distributed ledger technology to improve transparency, traceability, and the reliability of digital records. These characteristics are relevant to banking applications, where accurate account balances and verifiable transaction histories are essential. However, the practical benefits of blockchain depend on the underlying architecture, implementation quality, and operational requirements of the application.

Research on smart-contract security has also emphasized the importance of evaluating the correctness and reliability of blockchain-based applications. Atzei et al. [5] surveyed attacks against Ethereum smart contracts and described vulnerabilities that may arise from incorrect contract design and implementation. This indicates that using a blockchain does not automatically make a banking application secure. Smart-contract logic, access restrictions, input validation, and transaction-handling mechanisms must also be examined.

For the proposed Blockchain-Based Banking System, evaluation focuses on functional correctness, role-based access control, financial transaction processing, blockchain transaction verification, and consistency between MongoDB records and blockchain state. The system supports account management, KYC processing, deposits, withdrawals, and fund transfers through a combination of conventional web technologies and Solidity smart contracts. Testing these functions helps determine whether the implementation behaves as expected and whether transaction outcomes can be traced to their corresponding blockchain records. The evaluation also considers error scenarios, including blockchain transaction failures and database-update failures after successful blockchain execution..

2.3 Blockchain-Based Transaction Verification and Smart-Contract Validation

Transaction verification is an essential component of a banking system because every financial operation must be processed according to defined rules and recorded accurately. In conventional applications, transaction validation commonly involves checking user permissions, account status, transaction amounts, and available balances before an operation is approved. In a blockchain-based system, these checks can be combined with smart-contract execution to maintain the financial account state and provide a verifiable record of transaction activity.

Smart contracts provide programmable rules for processing blockchain transactions. In the proposed system, Solidity smart contracts manage customer account states and support operations such as deposits, withdrawals, and fund transfers. Before a transaction is executed, the backend validates the request and retrieves the authoritative account balance from the blockchain. The smart contract then processes the requested operation according to its defined conditions. This approach helps prevent application-level database changes from directly modifying the financial balance maintained by the smart contract.

However, successful smart-contract execution and accurate application-level recordkeeping are separate concerns. A blockchain transaction may succeed even if the corresponding MongoDB update fails. Similarly, a blockchain transaction may be rejected or fail before the intended state change is completed. Therefore, the system must distinguish between blockchain execution outcomes and the status of the corresponding application record. Blockchain transaction hashes and block numbers provide additional information for tracing and verifying recorded operations.

The proposed Blockchain-Based Banking System combines backend validation, Solidity smart-contract rules, and blockchain transaction metadata to support transaction verification. MongoDB maintains application-level transaction information, while the blockchain remains the authoritative source for financial account state. Transaction records are initially assigned a Pending status, updated to Completed after successful blockchain execution, or marked Failed when blockchain execution fails. If blockchain execution succeeds but the database update encounters an error, the record remains Pending for reconciliation. This combined approach supports transaction traceability and helps identify inconsistencies between application records and blockchain state.

2.4 Existing Blockchain-Based Banking Systems and Security Tools

Several blockchain platforms and development tools are available for building, testing, and evaluating smart-contract-based applications. Ethereum provides a programmable blockchain environment in which smart contracts can enforce predefined rules and maintain application state. Solidity is commonly used to develop these contracts, while development frameworks such as Truffle support contract compilation, deployment, and testing. Ganache provides a local blockchain environment that allows developers to simulate transactions and evaluate contract behavior during development.

Smart-contract security analysis tools also contribute to the evaluation of blockchain applications. For example, Oyente is a symbolic-execution tool designed to identify potential

vulnerabilities in Ethereum smart contracts, while Mythril performs security analysis of EVM-compatible smart contracts to detect issues such as unsafe execution patterns and access-control weaknesses. Although these tools can help identify certain classes of vulnerabilities, their findings do not by themselves establish that a complete banking application is secure. Application authentication, role-based authorization, database consistency, and transaction-processing logic must also be evaluated.

In addition to contract-level analysis, blockchain applications require mechanisms for tracking and verifying transaction execution. Ethereum records transaction information on-chain, including transaction hashes and block-related metadata, which can be used to inspect and verify blockchain operations. Web3.js provides an interface through which JavaScript applications can communicate with Ethereum-compatible networks and interact with deployed smart contracts. These capabilities support the integration of blockchain operations with conventional web application components.

The proposed Blockchain-Based Banking System combines Solidity smart contracts with a React frontend, a Node.js and Express.js backend, MongoDB, and Web3.js. The smart contract maintains financial account state and processes operations such as deposits, withdrawals, and fund transfers. MongoDB stores customer profiles, authentication-related information, KYC records, and application-level transaction metadata, while blockchain state remains authoritative for financial balances and contract-executed operations. Transaction hashes and block numbers are associated with application records to support traceability and verification.

A further consideration is consistency between blockchain execution and off-chain database records. A blockchain transaction may succeed even if the corresponding database update fails, while a rejected blockchain operation must not be recorded as a completed financial transaction. The proposed system therefore distinguishes between Pending, Completed, and Failed transaction statuses and retains successful blockchain operations with unsuccessful database updates as Pending for subsequent reconciliation. This approach helps prevent application-level status records from incorrectly representing the outcome of a blockchain operation.

Overall, existing blockchain platforms and security tools provide important capabilities for smart-contract development, transaction inspection, and vulnerability analysis. However, evaluating a blockchain-based banking application requires more than confirming that its smart contracts can execute successfully. The evaluation must also consider role-based access control, account-operation correctness, transaction traceability, failure handling, and consistency between on-chain financial state and off-chain application records. The proposed system addresses these concerns through a hybrid architecture and a transaction-status management process, with functional testing and security evaluation needed to assess its effectiveness.

2.5 Research Gap

Existing blockchain platforms and security tools provide capabilities for smart-contract development, transaction verification, and vulnerability detection. However, these tools do not independently address the complete requirements of a banking application, particularly the integration of financial transaction processing, role-based access control, and off-chain database management.

The proposed Blockchain-Based Banking System aims to address this gap by integrating Solidity smart contracts with a React frontend, a Node.js and Express.js backend, MongoDB, and Web3.js. It combines blockchain-based account balance management with conventional banking features, including customer account management, KYC processing, deposits, withdrawals, and fund transfers.

A further challenge is maintaining consistency between blockchain transactions and application-level database records. The proposed system addresses this through transaction status management, blockchain transaction hashes, block numbers, and a reconciliation process for cases where blockchain execution succeeds but the corresponding database update fails.

Table 1 summarizes how the proposed system differs from existing blockchain development and security tools in terms of banking functionality, transaction traceability, access control, and on-chain/off-chain consistency management. The system's effectiveness in addressing these requirements must be established through functional and security testing.

Approach	Primary Focus	Transaction Tracking	Role-Based Access Control	Database–Blockchain Consistency
Ethereum Smart Contracts	Automated transaction execution	Yes	Contract-dependent	Not automatic
Truffle	Smart-contract development and testing	Supports testing and deployment	Not a complete solution	Not provided directly
Ganache	Local blockchain simulation	Yes, on the local network	Not a complete solution	Not provided directly
Mythril	Smart-contract vulnerability detection	Not its primary focus	Detects certain contract-level issues	Not its primary focus
Proposed BBBS	Integrated blockchain-based banking	Transaction hashes and block numbers	Administrator, cashier, and customer roles	Transaction statuses and reconciliation

3. PROPOSED FRAMEWORK

This section presents the proposed Blockchain-Based Banking System and its main components. Section 4 explains the system workflow and transaction-processing stages, while Section 5 describes the implementation technologies and development environment.

3.1 Overview

The proposed framework integrates blockchain technology with a conventional web application to support secure banking operations. The system uses a React frontend for user interaction, a Node.js and Express.js backend for request processing and access control, MongoDB for customer and application records, and Solidity smart contracts for managing financial account balances and transactions. Web3.js enables communication between the backend and the blockchain network.

When a user initiates an operation such as a deposit, withdrawal, or fund transfer, the backend validates the request and executes the corresponding smart-contract function. The blockchain maintains the authoritative financial account state, while MongoDB stores application-level transaction details and associated blockchain metadata. Transaction statuses and reconciliation mechanisms help manage failures between blockchain execution and database updates, supporting transaction traceability and consistency across the application.

3.2 Two Operating Modes

The proposed Blockchain-Based Banking System supports different user roles and banking operations through a unified application. In the customer mode, users can access their accounts, view balances and transaction histories, and initiate supported banking operations. In the administrator mode, authorized users can manage customer accounts, review KYC information, and oversee banking activities. Cashiers can perform authorized account and transaction operations according to their assigned permissions.

Both modes use the same backend services and blockchain interaction mechanism. The backend validates user authentication, role permissions, and transaction requests before invoking the appropriate Solidity smart-contract functions. MongoDB maintains customer information and application-level records, while the blockchain stores the authoritative financial account state and transaction outcomes. This shared architecture provides consistent transaction processing and traceability across the different user roles.

3.3 Framework Components

The main components of the proposed Blockchain-Based Banking System are:

- User Interface: Developed using React, Vite, and Tailwind CSS, it provides interfaces for administrators, cashiers, and customers to manage accounts and access authorized banking operations.
- Backend and Authentication: Node.js and Express.js handle API requests, validate transactions, and enforce role-based access control. JWT authentication using HTTP-only cookies helps protect user sessions.
- Database Management: MongoDB stores customer profiles, authentication-related information, KYC records, and application-level transaction details. It does not serve as the authoritative source for financial account balances.
- Smart Contracts and Blockchain: Solidity smart contracts manage account states, balances, deposits, withdrawals, and fund transfers. Ganache provides the local blockchain environment for development and testing.
- Blockchain Integration: Web3.js connects the backend to deployed smart contracts, enabling transaction execution and retrieval of blockchain data, including transaction hashes and block numbers.
- Transaction Tracking and Reconciliation: Each transaction is tracked using Pending, Completed, or Failed status. Successful blockchain operations are recorded with their blockchain metadata, while cases involving failed database updates are retained for reconciliation to help maintain consistency between on-chain and off-chain records.

3.4 Transaction Status Classification

Each banking transaction is assigned a status based on the outcome of its processing, as summarized in Table 2. The proposed system distinguishes between Pending, Completed, and Failed transactions to represent the relationship between blockchain execution and application-level record updates. These statuses help administrators and authorized users understand the current processing state of a transaction.

Transaction Status	**Meaning**
Pending	The transaction has been initiated, but its final processing status has not yet been confirmed or the corresponding database update requires reconciliation.
Completed	The blockchain operation has succeeded, and the application record has been updated with the relevant transaction details.
Failed	The blockchain operation failed or was rejected, and the intended financial state change was not completed.

4. METHODOLOGY

4.1 Problem Formulation

Consider a blockchain-based banking system that manages customer accounts and financial transactions using smart contracts and a conventional web application. A user initiates an operation, such as account creation, deposit, withdrawal, or fund transfer, through the application. The backend validates the request and invokes the corresponding smart-contract function, while MongoDB maintains customer information and application-level transaction records. The primary objective is to process banking operations according to predefined rules while maintaining accurate financial account states and traceable transaction records.

A conventional database-based approach alone may not provide independently verifiable transaction records, while a blockchain-only approach does not address all application requirements, such as customer profile management, authentication, and KYC processing. Therefore, the proposed system combines blockchain-based financial state management with off-chain data storage and role-based access control.

The methodology focuses on four main tasks: validating user requests and permissions, executing financial operations through smart contracts, recording blockchain transaction details in MongoDB, and handling failures between blockchain execution and database updates. Transaction outcomes are classified as Pending, Completed, or Failed, with reconciliation used when blockchain execution succeeds but the corresponding database update encounters an error. This approach provides a structured basis for evaluating transaction correctness, traceability, access control, and consistency between on-chain and off-chain records.

4.2 Transaction Request Validation

The first step in the proposed system is to validate the banking request submitted by the user. The backend checks the user's authentication status, assigned role, required input fields, account details, and transaction amount before invoking the relevant smart-contract function. These checks help ensure that each operation is performed only by an authorized user and satisfies the application's predefined validation rules.

Validating a transaction requires more than checking whether the submitted data is correctly formatted. For example, a withdrawal must verify the account's available blockchain balance, while a fund transfer must validate the sender's and receiver's accounts and ensure that the requested amount satisfies the contract's conditions. Separating these checks from transaction execution helps prevent invalid requests from reaching the smart contract.

After validation, the backend communicates with the deployed Solidity smart contract through Web3.js. The contract processes the operation according to its defined rules, and the application records the transaction outcome and relevant blockchain metadata in MongoDB. This process supports controlled transaction execution, traceability, and consistent handling of successful and unsuccessful banking operations.

4.3 Blockchain Interaction and Transaction Validation

After a transaction request is validated, the backend retrieves the relevant account information from the blockchain through Web3.js. The smart contract maintains the authoritative account balances and state, allowing the system to verify whether the requested operation satisfies the contract's predefined conditions. For example, a withdrawal or fund transfer must satisfy the required balance and account-status checks before the financial state is updated.

A validation stage is applied before executing the smart-contract operation. If the request contains invalid account information, an unauthorized action, or an amount that violates the contract's rules, the operation is rejected. Valid requests are submitted to the deployed smart contract, which processes the transaction according to its defined logic. This approach helps ensure that financial operations are governed by explicit rules rather than relying solely on application-level database records.

After execution, the system records the transaction status and available blockchain metadata, including the transaction hash and block number, in MongoDB. If blockchain execution fails, the transaction is marked Failed; if execution succeeds, the application record is updated to Completed. When the blockchain operation succeeds but the database update fails, the record remains Pending for reconciliation. This process supports transaction verification and helps maintain consistency between blockchain state and application-level records.

4.4 Multi-Stage Transaction Validation

The proposed system validates banking operations using multiple checks before and during smart-contract execution. The first stage verifies the user's authentication, role permissions, account details, and transaction parameters. The second stage relies on the Solidity smart contract to enforce financial rules, such as checking account status and ensuring sufficient balance for withdrawals and fund transfers.

These validation stages complement each other because application-level checks and smart-contract rules serve different purposes. The backend controls access to banking functions and validates incoming requests, while the smart contract enforces the conditions governing changes to blockchain-based account balances. A request that fails validation or violates a contract rule must not be treated as a successfully completed transaction.

After the operation is processed, the system evaluates the blockchain execution result and updates the corresponding MongoDB transaction record. Successful operations are marked Completed, failed blockchain operations are marked Failed, and transactions requiring database reconciliation remain Pending. Blockchain transaction hashes and block numbers are retained when available to support verification and traceability. Combining these checks helps reduce inconsistencies and provides a clearer record of transaction outcomes.

4.4 Multi-Stage Transaction Validation

The proposed system validates banking operations using multiple checks before and during smart-contract execution. The first stage verifies the user's authentication, role permissions, account details, and transaction parameters. The second stage relies on the Solidity smart contract to enforce financial rules, such as checking account status and ensuring sufficient balance for withdrawals and fund transfers.

These validation stages complement each other because application-level checks and smart-contract rules serve different purposes. The backend controls access to banking functions and validates incoming requests, while the smart contract enforces the conditions governing changes to blockchain-based account balances. A request that fails validation or violates a contract rule must not be treated as a successfully completed transaction.

After the operation is processed, the system evaluates the blockchain execution result and updates the corresponding MongoDB transaction record. Successful operations are marked Completed, failed blockchain operations are marked Failed, and transactions requiring database reconciliation remain Pending. Blockchain transaction hashes and block numbers are retained when available to support verification and traceability. Combining these checks helps reduce inconsistencies and provides a clearer record of transaction outcomes.

4.6 Transaction Reliability and Status Tracking

The proposed system tracks the outcome of each banking transaction using its execution status and available blockchain metadata. Rather than relying on a numerical confidence score, the system uses the blockchain execution result, transaction hash, block number, and database update status to determine whether an operation has completed successfully or requires further investigation.

A transaction is marked Completed when the smart-contract operation succeeds and the corresponding MongoDB record is updated successfully. It is marked Failed when blockchain execution fails or the operation is rejected. If the blockchain operation succeeds but the database update fails, the transaction remains Pending until reconciliation is performed. This distinction prevents a database synchronization problem from being incorrectly interpreted as a failed financial operation.

5. SYSTEM IMPLEMENTATION

5.1 Architecture

The proposed Blockchain-Based Banking System is implemented as a full-stack web application that integrates a frontend interface, backend services, a database, and blockchain-based smart contracts. The React frontend provides interfaces for administrators, cashiers, and customers to manage accounts, process banking operations, review KYC information, and access transaction histories. The backend coordinates authentication, authorization, transaction validation, database operations, and communication with the blockchain network.

The Node.js and Express.js backend handles customer requests and invokes the appropriate smart-contract functions through Web3.js. MongoDB stores customer profiles, authentication-related information, KYC records, and application-level transaction details, while the smart contract maintains the authoritative financial account state. Transaction hashes, block numbers, and processing statuses are stored alongside application records to support transaction tracking and verification.

Component	Technology
Frontend	React.js, Vite, Tailwind CSS, React Icons
Backend	Node.js, Express.js
Primary database	MongoDB, Mongoose
Smart contracts	Solidity
Blockchain development and testing	Ganache, Truffle
Blockchain communication	Web3.js
Authentication	JWT, HTTP-only cookies, cookie-parser
API communication	Axios
Email notifications	Nodemailer, SMTP

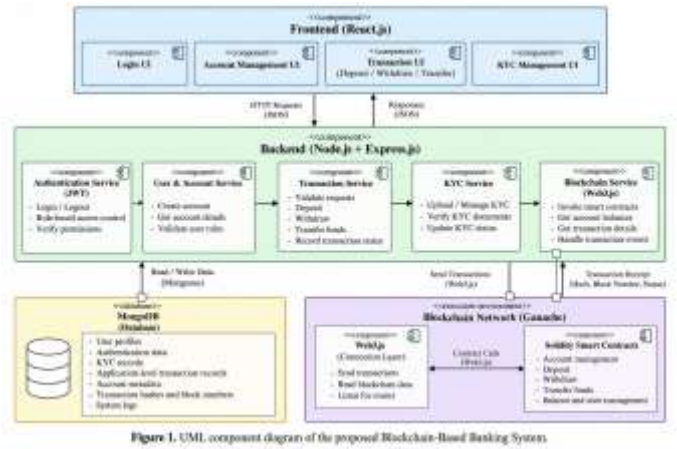


Figure 1. Proposed architecture of the Blockchain-Based Banking System, showing the React frontend, Node.js and Express.js backend, MongoDB database, and Solidity smart contracts deployed on the blockchain.

The system uses JWT authentication with HTTP-only cookies and role-based authorization to restrict access to banking functions. During local development, Ganache provides the blockchain environment and Truffle supports smart-contract compilation, deployment, and testing. The architecture separates customer and application data from blockchain-based financial state, while transaction-status management and reconciliation address potential inconsistencies when blockchain execution and database updates do not complete together.

5.2 Technology Stack

The proposed Blockchain-Based Banking System uses open-source technologies to support development, testing, and maintenance while minimizing software costs. The frontend is developed using React.js, Vite, Tailwind CSS, and React Icons, while Node.js and Express.js handle backend processing and API requests. MongoDB, managed through Mongoose, stores customer profiles, authentication-related information, KYC records, and application-level transaction data.

Solidity is used to implement smart contracts that manage account states, balances, deposits, withdrawals, and fund transfers. Web3.js connects the backend to deployed smart contracts, while Ganache provides a local blockchain environment for development and testing. Truffle supports smart-contract compilation, deployment, and testing, and Axios facilitates communication between the frontend and backend.

For authentication and supporting services, the system uses JWT with HTTP-only cookies and cookie-parser to manage authenticated sessions and role-based access. Nodemailer with SMTP supports email notifications. The selected technologies provide a modular architecture that separates application data management from blockchain-based financial state.

The current implementation uses Ganache for local blockchain testing. Ethereum Sepolia, Alchemy, MetaMask, and MongoDB Atlas are considered for production-oriented testing and deployment. These planned technologies are distinguished from the components currently used in the local development environment.

5.3 Procedural Flow

The overall processing flow of the proposed Blockchain-Based Banking System consists of three main workflows: user authentication and account management, banking transaction processing, and transaction verification and recordkeeping. Although each workflow begins with a different user request, they share common stages involving backend validation, database interaction where required, and communication with the blockchain through Web3.js.

In the account management workflow, the system authenticates the user, checks role-based permissions, and processes account creation or account information requests. Customer profiles, KYC information, and related application records are maintained in MongoDB, while account states managed by the smart contract are accessed through the blockchain service.

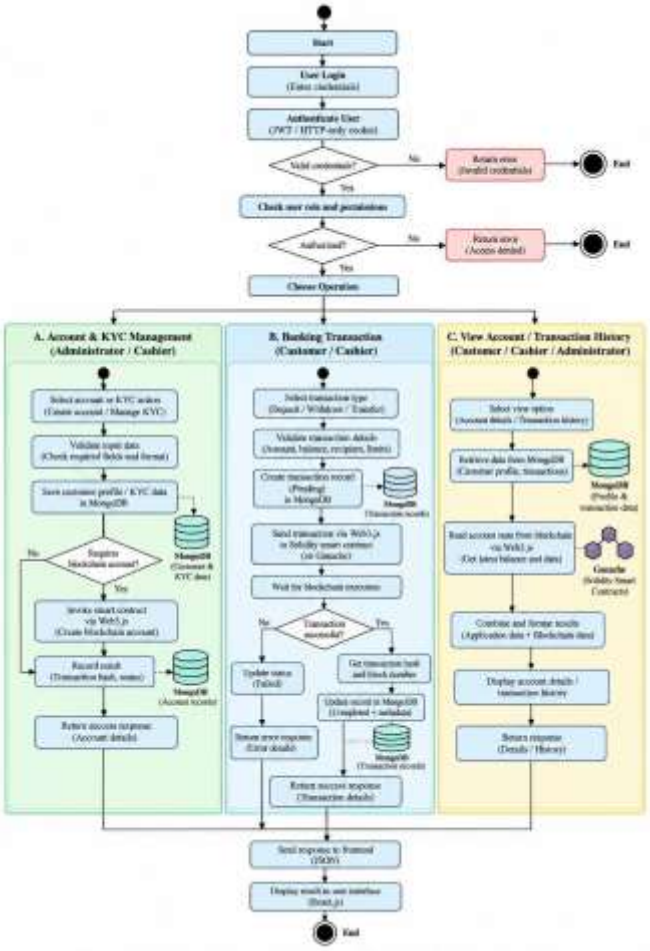


Figure 2. Procedural flow of account management, banking transaction processing, and transaction record verification in the proposed Blockchain-Based Banking System.

6. EVALUATION METHODOLOGY

The proposed Blockchain-Based Banking System is evaluated through functional and comparative testing to assess transaction correctness, access control, blockchain integration, and consistency between blockchain state and MongoDB records. Different scenarios are tested, including successful transactions, invalid requests, unauthorized operations, insufficient-balance withdrawals, and failures during database updates. This evaluation helps identify whether the system processes banking operations according to the defined rules and handles errors appropriately.

6.1 Metrics

The following metrics are used to evaluate the system:

- Transaction success rate: Measures the proportion of valid transactions that complete successfully.
- Transaction failure rate: Measures the proportion of attempted transactions that fail or are rejected under the tested conditions.
- Access-control accuracy: Evaluates whether users can perform permitted operations and whether unauthorized requests are correctly rejected.
- Average transaction latency: Measures the average time required to process an operation, including backend validation and blockchain execution.
- Data consistency rate: Measures how accurately MongoDB transaction records correspond to blockchain execution outcomes and account state.
- Reconciliation success rate: Measures the proportion of transactions requiring reconciliation that are successfully matched with their blockchain results and updated application records.

6.2 Test Dataset Construction

The evaluation uses a custom test dataset designed to assess the functionality and reliability of the proposed Blockchain-Based Banking System. The dataset consists of test scenarios covering account creation, customer and KYC management, deposits, withdrawals, fund transfers, account balance queries, and transaction history retrieval. These scenarios are designed to represent normal banking operations as well as invalid requests and exceptional conditions.

Each test case is assigned an expected outcome based on the system's business rules and smart-contract logic. For example, a withdrawal with sufficient balance should succeed, while a withdrawal exceeding the available balance should be rejected. Additional test cases cover unauthorized access, invalid account details, unsuccessful blockchain transactions, and database update failures following successful blockchain execution.

The test results are compared with the expected outcomes to evaluate transaction correctness, access-control enforcement, transaction-status accuracy, and consistency between MongoDB records and blockchain state. Reconciliation scenarios are also included to determine whether incomplete application records can be corrected using blockchain transaction information. The resulting dataset provides a structured basis for evaluating the implemented system, with all reported results derived from actual test executions.

6.3 Transaction Consistency Evaluation

The evaluation examines whether the transaction status recorded in MongoDB accurately reflects the outcome of the corresponding blockchain operation. Successful blockchain transactions should be associated with Completed records, while rejected or failed blockchain operations should be recorded as Failed. Transactions for which blockchain execution succeeds but the database update fails should remain Pending until reconciliation is completed.

The evaluation also examines whether transaction hashes, block numbers, account balances, and transaction records can be used to identify inconsistencies between the application and the blockchain. Test cases involving successful operations, rejected requests, and database or network failures are used to assess the system's ability to handle different transaction outcomes.

The results are used to determine the accuracy of transaction status management and the effectiveness of the reconciliation process. This evaluation helps establish whether the proposed system maintains reliable application records while treating the blockchain as the authoritative source of financial account state.

7. LIMITATIONS AND FUTURE WORK

7.1 Limitations

Although the proposed Blockchain-Based Banking System integrates smart contracts with conventional web technologies, it has several limitations. The system currently uses Ganache for local blockchain development and testing, so its behavior may differ from that of a public blockchain network in terms of transaction latency, network fees, and network conditions. In addition, the correctness of financial operations depends on the smart-contract implementation, backend validation, and appropriate access-control configuration.

Another limitation is the dependence on blockchain and database communication. Network interruptions, failed transactions, or unsuccessful MongoDB updates can cause transaction records to remain pending until reconciliation is completed. The system also maintains customer information and KYC records off-chain, which requires appropriate authentication, authorization, and data-protection measures to safeguard sensitive information.

The current implementation is primarily a research prototype and has not yet established its suitability for large-scale production banking. Further testing is required to evaluate concurrent transactions, scalability, security vulnerabilities, and recovery from system failures. Future work will focus on testing with Ethereum Sepolia, improving reconciliation and error handling, strengthening smart-contract security, and evaluating performance under different workloads.

7.2 Future Work

Several improvements can be explored in future versions of the proposed Blockchain-Based Banking System. First, the system can be deployed and tested on the Ethereum Sepolia test network to evaluate smart-contract execution, transaction latency, network fees, and behavior under realistic network conditions. Integration with Alchemy, MetaMask, and MongoDB Atlas can also be explored as part of a more deployment-oriented architecture.

Second, the system can be enhanced by strengthening smart-contract security through automated vulnerability analysis, comprehensive contract testing, and additional access-control checks. Improvements to transaction reconciliation, error handling, and recovery mechanisms could further help manage situations in which blockchain execution and database updates do not complete together.

Finally, future work can focus on evaluating scalability under concurrent transactions, supporting larger numbers of users, and measuring performance under different workloads. Additional testing can examine transaction correctness, data consistency, and system availability during network interruptions or database failures. These improvements would provide a stronger basis for assessing the suitability of the proposed system for more demanding banking application scenarios.

8. CONCLUSION

This paper presented the design and development of a Blockchain-Based Banking System that integrates blockchain technology and smart contracts with conventional web application technologies. The proposed system combines a React frontend, a Node.js and Express.js backend, MongoDB for customer and application records, and Solidity smart contracts for managing financial account balances and banking operations. It supports account management, KYC processing, deposits, withdrawals, fund transfers, and transaction history tracking through role-based access for administrators, cashiers, and customers.

The system uses Web3.js to communicate with deployed smart contracts and associates blockchain transaction hashes and block numbers with application-level records to support transaction traceability and verification. A transaction-status management mechanism distinguishes Pending, Completed, and Failed operations and supports reconciliation when blockchain execution succeeds but the corresponding database update fails. By maintaining customer information off-chain and financial account state on the blockchain, the proposed architecture combines conventional data management with blockchain-based transaction processing.

The current implementation is developed and tested using Ganache as a local blockchain environment. Further evaluation is required to assess transaction correctness, access-control enforcement, database–blockchain consistency, processing latency, and failure recovery. As the system is still a research prototype, no claims of improved security, scalability, or performance are made without completed experimental results. Future work will focus on broader testing, smart-contract security analysis, and evaluation using Ethereum Sepolia to investigate the system's behavior under more realistic blockchain network conditions.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to **Ms. Ahinsa Gaikwad** for her valuable guidance, support, and encouragement throughout the development of the Blockchain-Based Banking System. The authors also thank the faculty members and their institution for providing the necessary resources and academic support to complete this research work. Finally, they extend their appreciation to everyone who contributed directly or indirectly to the successful completion of this project.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[2] V. Buterin, “A Next-Generation Smart Contract and Decentralized Application Platform,” 2014.

[3] G. Wood, “Ethereum: A Secure Decentralised Generalised Transaction Ledger,” Ethereum Yellow Paper, 2014.

[4] F. Casino, T. K. Dasaklis, and C. Patsakis, “A Systematic Literature Review of Blockchain-Based Applications: Current Status, Classification and Open Issues,” Telematics and Informatics, vol. 36, pp. 55–81, 2019.

[5] ConsenSys, “Mythril: Security Analysis Tool for EVM Bytecode.”

[6] Ethereum Foundation, “Web3.js Documentation.”

[7] Truffle Suite, “Truffle Documentation.”

[8] Ethereum Foundation, “Ethereum Developer Documentation.”