Internship in Information Technology Division at City Bank PLC

by

Md Mainul Hossain Chisty
24241084

An internship report submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
October 2025

Declaration

It is hereby declared that

1. The report submitted is my own original work while completing degree at Brac University.
2. The report does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The report does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. I have acknowledged all main sources of help.

Student's Full Name & Signature:

_____

Md Mainul Hossain Chisty

24241084

Approval

The internship report titled "Internship in Information Technology Division at City Bank PLC" submitted by

1. Md Mainul Hossain Chisty (24241084)

of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science in October 10, 2025.

Examining Committee:

Supervisor:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
BRAC University

Thesis Coordinator:
(Member)

Dr. Md. Golam Rabiul Alam

Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Sadia Hamid Kazi

Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

This internship experience, in the Information Technology Division of the City Bank PLC, has been a crucial link between theory and practical knowledge in the banking industry IT practices. The program offered a platform to use theoretical knowledge on practical issues and also develop vital skills of technical documentation, problem-solving approaches through teamwork, and understanding organization processes. One of the key highlights during this internship was the building of a prototype of a Core Banking System (CBS), that modeled the essential banking processes, including, but not limited to account management and fund transfers, as well as loan processes. This project provided an informative experience on system design, database management, and the incorporation of secure authentication controls, per the operational needs of financial organization at present day. This experience contributed to both my academic and professional growth. This report displays the experiences and challenges of the industry also portrays the development process, challenges, and way of overcoming those challenges of Core Banking System project in detail.

Keywords: City Bank PLC, Information Technology, Problem Solving, Real-World Projects, and Technical Documentation.

Acknowledgement

All praises to almighty Allah for allowing me to complete my internship without any major drawbacks. I would also like to express my gratitude towards Dr. Md. Golam Rabiul Alam Sir for supervising and guiding me in my internship journey. I am grateful to my parents for always supporting me, without their help this would not have been possible. Lastly, thanks to everyone who have helped, guided and motivated me during my journey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

City Bank PLC is one of the oldest premier private commercial banks in Bangladesh, incorporated in 1983. Known for its innovative banking solutions and loyal commitment to its customer satisfaction, it is playing a vital role in the economic and financial sector in Bangladesh. Their goal is to provide best-in-class economic services to the customer and business to create economic growth. Their missions are offering a wide array of products and services that differentiate and excite all customer segments, being the "employer of choice" by offering an environment where people excel and leaders are created, continuously challenging processes and platforms to enhance effectiveness and efficiency, and promoting innovation and automation with a view to guaranteeing and enhancing excellence in service. With their slogan "making sense of money," their vision is to become a financial supermarket with a winning culture offering an enjoyable experience.

The City Bank's goal is to be a business hub with an upbeat attitude and engaging activities to do. Offering a broad variety of innovative and distinctive services and products to all kinds of clients is their aim. They are always looking for ways to make their processes and systems more efficient and effective. In order to provide top-notch services, they also encourage innovation and automation. In anything they do, they pledge to uphold good governance, respect the community, and remain completely compliant. Results are important to them, and they take accountability, act bravely and respectfully, remain involved and inspired, and always prioritize customer satisfaction.

The bank offers services like personal banking, business banking, priority banking, and corporate and institutional services. Their IT division is composed of several teams like IT Service Management, IT Governance, Card and ADC System, Business Enterprise Application, Enterprise Architecture, Information Security, and DFS Tech, all of which support both customer-facing and internal digital banking platforms. As part of the Enterprise Architecture team, my assigned role was documentation. This allowed me to understand how developers and other IT teams collaborate while adhering to strict security guidelines set by Bangladesh Bank.

1.2 Rational of the Study or Motivation

This internship was motivated by the need to bridge the gap between academic knowledge and practical industry experience. While pursuing a Computer Science degree, I chose the internship path to gain hands-on exposure to the technological demands of the banking sector. With financial institutions like City Bank evolving rapidly through digital transformation, my goal was to understand how technical skills, particularly documentation and system analysis, integrate into the banking industry. The relevance of this experience lies in its focus on technical documentation, which is an underexplored but critical component of IT operations in high-security environments like banking.

1.3 Problem Statement

Despite the growing role of IT in the banking sector, there exists a gap in effective documentation practices that balance technical detail, usability, and security compliance. During my internship at City Bank PLC, I primarily worked on technical documentation for internal banking systems. However, as a practical extension of my internship learning, I also started building a Core Banking System (CBS) using the PERN stack (PostgreSQL, Express.js, React.js, Node.js).

The project aims to replicate fundamental banking operations such as customer account creation, transaction handling, loan requests, and admin-employee workflows, inspired by the operational structure I observed at City Bank. This CBS project is a direct response to the need for modular, scalable, and secure digital banking systems, particularly for institutions in developing regions where digital transformation is still in progress.

Through this project, I hope to address the challenges of integrating multiple user roles, maintaining transactional integrity, and ensuring clarity in user interfaces — all within the framework of modern web technologies. This product reflects both the technical skills developed during the internship and the industry gaps I identified.

1.4 Objective

The objective of this report is to document my internship experience at City Bank PLC with an emphasis on the technical documentation process and how it supports banking IT systems.

Specific objectives include:

- To provide an in-depth explanation of the business environment of City Bank PLC.
- To analyze the technological solutions utilized by the bank and their integration into banking activities, with a specific focus on platforms like Citytouch.
- To document the technical writing tasks carried out by me and their contribution to IT operations.
- To evaluate the competencies I developed, such as communication, collaboration, and documentation skills.

- To show how the internship served as a bridge between academic concepts and real-world applications.

1.5 Methodology in Brief

This study is based on a combination of primary and secondary data sources.

- **Primary Data:** Personal observations, daily tasks during the internship, and discussions with team members including developers and mentors.
- **Secondary Data:** Official documents and reports from City Bank, relevant publications, and industry literature regarding banking IT systems and documentation practices.

1.6 Scopes and Challenges

The study is limited by the security policies of the bank, which restrict intern access to source code and sensitive systems. Thus, while the scope covers the documentation process and internal workflows, it does not include in-depth development or deployment details. Another challenge was ensuring clarity in documentation while complying with internal data privacy protocols. However, this allowed a better understanding of how non-code roles significantly contribute to secure and scalable systems.

1.7 Team Overview

I was assigned to the Enterprise Architecture team under the Information Technology Division. My primary responsibility was documentation. Though I was not involved in core development due to compliance restrictions, I collaborated closely with developers, analysts, and other team members to gather information about systems and draft user manuals, procedural guides, and system overviews. The team maintained a cooperative environment that encouraged knowledge sharing and feedback-based revisions.

1.8 Key Learnings and Insights

Throughout the internship, I learned that technical documentation is not merely a support task—it is essential for system usability, regulatory compliance, and inter-team communication. I gained experience in structuring documents for different audiences, incorporating feedback loops, and adapting to organizational security policies. I also improved my professional communication, learned to coordinate with technical teams, and gained insights into the structure and operation of enterprise-level banking systems. In addition to documentation, I set a personal goal to build a core banking system using the PERN stack to demonstrate the skills and knowledge acquired during the internship.

Chapter 2

Literature Review

2.1 Preliminaries

The purpose of this chapter is to explore the existing literature and theoretical framework related to my internship experience at City Bank PLC. This review will focus on key areas including the role of IT in the banking sector, the importance of technical documentation, digital transformation in financial institutions, previous practices and new emerging technology, and their adaptation in the banking IT sector.

As a part of my internship responsibility, I was assigned the task of technical documentation and user manual creation. Technical documentation is an important part of the banking IT sector because it allows smooth communication between users, developers and complex financial systems. Existing research and literature suggest that user manuals are very crucial in guiding the end users to learn about intricate banking software while meeting the strict regulatory requirements [5], [8]. However, despite this recognition, there is limited literature specifically tailored to the financial sector's unique challenges, such as regulatory compliance, data confidentiality and evolving cybersecurity threats, which often constrains how much detail can be shared in documentation.

2.2 Review of Existing Research

2.2.1 Technical Documentation: Lifecycle and practices

Their evolution has also posed challenges such as the simplification of complex workflows, maintaining the content's value and achieving a suitable position between detail and simplicity [14], [16], [27]. This reflects my own experience, where I had to ensure clarity for end users while deliberately omitting sensitive configurations to comply with internal security policies. Such trade-offs underscore a persistent tension between usability and information security, a topic underexplored in current literature.

Similarly, API documentation is needed to allow developers to integrate bank systems with other third-party services but must be precise, very secure and interoperable across platforms [2], [17]. While existing studies emphasize precision, platform independence and robust security, they often overlook the day-to-day maintenance challenges within high-security environments like banks. As I observed at City Bank, changes in one system

often had ripple effects across APIs, requiring vigilant versioning and cross-team collaboration elements frequently simplified in academic discussions.

In order to improve readability and maintainability, best practices for both types of documentation recommend the use of easy-to-understand language, systematic structure, graphical assistance, code examples, and versioning to enhance usability and maintainability [10], [13]. Yet, a notable limitation in the literature is the insufficient focus on documentation lifecycle management, especially in rapidly changing DevOps environments. Apart from this, more recent innovations like DevDocOps promote continuous and automatic documentation upgrading, while AI tools help in identifying outdated content and improving content quality [9], [16], [24].

These innovations show promise, but empirical evidence on their adoption in developing economies or regulated sectors like banking is sparse. Thus, more context-specific studies are needed to assess their effectiveness in real-world settings. These innovations show that technical documentation is a dynamic aspect that has become more important in terms of enabling innovation, compliance, and user experience in the financial technology industry. To reduce the danger of data breaches, I was told to provide system overviews simply while eliminating sensitive technical information, in accordance with industry standards. This illustrates the fine balance between security and clarity in professional documentation.

2.2.2 Modernization of Banking Technology Stacks

The banking sector is undergoing a technological transformation, driven by the need to improve efficiency, customer experience, and security. The traditional bank infrastructure has usually relied on robust technologies such as Java, with frameworks like Spring Boot for backend, Angular for frontend, and Hibernate for database management to ensure scalability and maintainability [26]. Java and PHP were the languages of choice for City Bank. They are still using Spring Boot, ASP.NET, and Laravel for many of their applications.

However, these legacy systems often introduce rigidity and slower adaptation to emerging technologies, posing a critical limitation in scalability and modernization. These systems are further reinforced with integrated security measures and customer relationship management (CRM) features to ensure industry standards and compliance.

Recent trends have witnessed a shift towards newer web development stacks such as the MERN stack, which stands for MongoDB, Express.js, React.js, and Node.js, that offer dynamic user interfaces, backend performance, and seamless data flow [12]. However, while such stacks promise agility, they also bring new challenges related to security hardening, backward compatibility, and team retraining, issues that literature often glosses over.

At the same time, the integration of AI and machine learning is enabling personalized services, automation, and better decision-making [21], while blockchain and big data technologies are promoting transaction security, risk management, and regulatory compliance [23]. However, the adoption of such technologies requires significant infrastructure, technical skills, and trust, resources that are not equally available across institutions, especially in developing countries. Fintech innovations continuously reshape the landscape by developing unique business models and digital solutions, compelling traditional institutions to

evolve [11].

Technologies like cloud computing and decentralized fintech are making banking operations more resilient and scalable [23]. Yet cloud adoption also brings concerns over data residency, third-party dependencies, and vendor lock-in issues that remain insufficiently addressed in mainstream discussions. As much as such technologies have evolved, banks still need to balance innovation with their conventional role as financial intermediaries in a way that the adoption of emerging technologies improves their primary mandate [11].

2.2.3 Core Banking Systems and Digital Transformation in Bangladesh

The growth of core banking systems (CBS) in Bangladesh has been highly driven by the growing application of information and communication technology, which has been imperative to enhance competitiveness, operational efficiency, and regulatory compliance [4]. The majority of banks have built or upgraded their CBS to raise the level of service delivery as well as to keep up with technological advancements [3]. However, full-scale digital transformation remains hampered by stakeholder misalignment, data migration issues, and the limited flexibility of existing software.

These findings resonate with what I observed during my internship, where CBS enhancements were often delayed by technical constraints and coordination challenges across departments. Perfect implementation is also hampered by management concerns, including coordinating stakeholder expectations and staff roles, as well as technological constraints in software flexibility and data migration [4]. Further, demanding the competence and trustworthiness of software vendors remains a fundamental concern [4]. Additionally, the dependency on foreign vendors and legacy codebases adds to the complexity, raising concerns about sustainability and long-term control, areas that require further academic exploration.

By providing specialized solutions for enabling dependable and secure banking services, local software companies are essential in overcoming these obstacles [15]. However, they are limited in their ability to facilitate strategic decision-making because of several shortcomings, particularly in the integration of business information with CBS [3]. These issues need to be addressed in order to make way for sustainable core banking system development in Bangladesh.

2.3 Summary of Key Findings

Overall, while recent studies highlight the central role of IT in banking innovation and hint at the growing significance of technical documentation, they are also prone to being short on clarifying the intricate problems of institutions in regulated and scarcity-based environments. My internship experience at City Bank PLC confirmed most of these deficiencies, varying from the inability to balance document security and transparency to limitations in older infrastructure and coordination problems across teams. Though new developments like AI-boosted documentation and newer stacks like MERN have much promise, their successful implementation requires further representative studies that reflect actual complexity.

Thus, additional research specifying the operational dynamics of banks in emerging markets, especially those faced with legacy systems, regulatory demands, and digital transformation simultaneously, is required to make progress in theory as well as in real-world environments.

Chapter 3

Requirements, Impacts and Constraints

3.1 Final Specifications and Requirements

The core banking system developed in this project has a set of technical resources and methodological requirements that guided its design and implementation. Technically, the frontend of the system was developed using React.js with Vite, and it provides a speedy, modular, and reactive user interface. Tailwind CSS, extended by DaisyUI, was used to get a responsive and user-friendly interface for different roles, including customers, employees, and administrators. The backend was built using Node.js with the help of Express.js, thus allowing the creation of RESTful API endpoints to handle all server-side operations, including account creation, transaction processing, fund transfer, loan management, and audit logging. The database was built using PostgreSQL, hosted on the Neon cloud platform, and structured to store and handle sensitive banking information like user credentials and transaction history. The authentication system was built using the database and backend itself with the help of JSON Web Token (JWT), and third-party services were not used. To keep the user login information safe, bcryptjs was used for hashing user passwords. To add to that, a role-based access control system is also used to protect users' information from unauthorized access by other users. Libraries and modules like Emailjs, Jspdf, etc., are used for implementing crucial features like email notifications and statement download options. This system was deployed in Render, which provides cloud hosting to not only the backend but also the frontend as well and ensures accessibility and reliability.

From the resources point of view, the project required hardware and software resources. It required minimal hardware resources to be developed on a personal computer with 8 gigabytes of RAM and 256 gigabytes of SSD, which was sufficient to hold development servers, databases, and frontend applications running concurrently. Software requirements included a code editor such as VS Code, backend runtime through Node.js, PostgreSQL client tools for data handling, version control through Git, and a current web browser for testing and deployment. The system was designed to utilize freely available software and open-source liberties where possible, keeping project costs low while still being functional. The requirements for data involved creating dummy data for employees, customers, loans, and transactions to test the entire system. Methodologically, the project followed a modular development path instead of the traditional model-view-controller architecture, which allows flexibility in positioning features and facilitating iterative testing and integration. Overall, interweaving technical, resource, and methodological specifications formed the basis for the system so that it would be both academically productive and practically useful

as well as scalable and flexible for future upgrading.

3.2 Societal Impact

The core banking system that has been developed under this project has significant implications for society, particularly in terms of electronic financial management. The most significant influence on society is the facilitation of secure, efficient, and reliable banking operations for the users, which reduces the dependency on traditional banking activities. By automating banking procedures, CBS improves key performance indicators (KPIs) like profitability and total deposits. In addition, banks can provide services anywhere and at any time, greatly boosting the client convenience and operational effectiveness [6]. With account opening, transactions, fund transfers, and loan management automated, the system eliminates the possibility of human errors and facilitates higher accuracy and reliability in financial operations. This ensures higher confidence and transparency among banks and customers. Secondly, the audit logging feature of the system allows thorough monitoring of user behavior that not only improves accountability but also prevents fraud and unauthorized access and otherwise enhances citizen confidence in digital banking products further. From a financial inclusion perspective, the system can provide easier banking access, particularly in places with fewer physical banking facilities. According to Dunbar and Treku, technological innovation like CBS can reduce the number of unbanked individuals by improving access to banking services. Increased awareness of digital banking solutions correlates with a decrease in unbanked populations, particularly among vulnerable groups [18]. By availing an online platform, the system allows its users to manage their finances remotely, reducing the need for trips to physical bank branches and, in the process, saving time and resources. The system also promotes financial literacy, as customers are provided with complete details of their transactions, balance, and loan terms, enabling them to comprehend and manage their finances better.

With regard to legal and safety issues, the system is supported by role-based access control and a safe method of authentication that meets the expectation of societal privacy and protection of data in online banking. While the existing system is largely a prototype in educational settings and far from fully in line with regulatory security requirements, the incorporation of security features like hashed passwords, JWT authentication based, Parameterized database queries, and HTTPS deployment reflects some attention to legal requirements and professional integrity in software development. Culturally, the system has been made flexible to be able to accommodate local banking culture to make it sustainable and relevant in countries like Bangladesh, where individuals are adopting digital banking at a very rapid pace. By doing away with differences between traditional banking and the new digital system, the CBS is capable of bringing modernization to society and accelerating financial transactions. In addition, transparency, accessibility, and simplicity of the system are welcome to users' trust and confidence, particularly where societies are not conducive to a departure from conventional banking conduct. In addition, transparency, accessibility, and simplicity of the system are beneficial to trust and confidence of users, particularly where societies are resistant to a departure from conventional banking habits. In summary, the social implication of the CBS project extends beyond functionality; it promotes social development by enhancing the access to finance, transparency of operations, promoting user trust, and enabling cautious financial behavior in the modern digital economy.

3.3 Environmental Impact

The core banking system assists in environmental sustainability primarily through the digitalization of the banking processes, which effectively reduces the utilization of paper-based documentation and transactions. The traditional banking processes are usually characterized by extensive use of hard copy forms, printed documents, and receipts, all of which contribute to environmental degradation in the form of deforestation, wastage of paper, and the energy consumed on the production of paper. With its all-digital system, the CBS minimizes the consumption of paper, thereby decreasing the ecological footprint of banking operations. By reducing the use of paper and energy, digital banking drastically lowers the carbon emission. This change is especially noticeable in Asia, where it has been demonstrated that digital banking methods lessen the carbon footprint of conventional banking activities [22]. Furthermore, the system's cloud deployment on the Render and Neon platforms allows for efficient utilization of computing resources. Cloud hosting, unlike conventional servers, typically has optimized power usage, scalability, and virtualization methods that reduce the overall ecological footprint. The approach enables multiple users to access banking services without each having to have personal physical infrastructure, decreasing energy consumption and environmental costs associated with hardware maintenance.

The system indirectly conserves carbon emissions by reducing the need for customers to physically travel to bank branches to perform transactions, transfer money, or verify accounts. These decreases in travel result in reduced consumption of fuel and greenhouse gas emissions. Although the project does not directly manage energy efficiency in devices used by the users, its architecture encourages remote operation, which is aligned with global tendencies of reducing environmental impact through digitalization. Further, the green habits of the system are continued through its modular and reusable software components, which are easily updated and maintained without having to overhaul entire systems and mitigate resource usage involved in software. Even though the prototype is small-sized and largely academical, the principles employed in its development demonstrate how digital banking solutions can guarantee environmental sustainability. With reduced paper use, optimized server utilization, and remote access, the CBS represents a virtual solution that is both operationally effective and green, which serves as proof of green technology integration in financial services.

3.4 Ethical Issues

The core banking system has several ethical concerns that need to be addressed while designing, developing, and implementing the system. One of the significant ethical concerns is the security and confidentiality of sensitive financial information. Because the system manages user credentials, account balances, transaction records, and loan information, it is now the responsibility of the developer to protect the data from storage to transfer. The ethical considerations in this project were as follows:hashing of the password using bcryptjs, secure authentication using JWT and encrypted communication using HTTPS. However, being a prototype, it does not necessarily have to match the security standards of production, and an example of ethical responsibility in the work of the professional software developer involves ensuring it is not abused or misused. Next is the transparency ethical policy in which the practice is shown by the introduction of audit logs in which administra-

tors can monitor the actions of users. This fosters responsibility and prevents fraud, which is in line with professional requirements in developing financial software. Professional responsibility also obligates the developer to make the system provide the users with the assurance that the system is not deceiving them about what it can or cannot do or what is safe and what is not. Moral issues aim not to present information in a misleading way, to reflect precise transactions, and to secure user inviolability and predictability in the use of the system.

Also, ethical considerations are passed on to the social impact of the existence of an online banking system. It should consider the needs of users who are not technologically challenged or digitally handicapped and should not discriminate against or exclude these users in any way. The CBS is to be open and friendly, yet in an educational institution as a demonstration project, its operation has focused to a greater degree on these legal requirements on data privacy, financial management, and cybersecurity despite the fact that a prototype can never achieve full compliance. The promise of ethical software development is to always think about the risks, mitigate vulnerability, and report system boundaries voluntarily. In all, the CBS project is attentive to ethical issues in bank software, and it places the highest priority on privacy, security, transparency, responsibility, and wide inclusiveness. By addressing these challenges, the project adheres to ethical principles in software engineering and professional responsibility, providing a sound foundation for future development and implementation in the real world.

3.5 Project Management Plan

The project plan for the core banking system was carefully defined to allow smooth implementation, timely completion, and optimal utilization of resources while managing the constraint of an academic prototype developed simultaneously with an Internship. While my total internship period was 2 semesters, I spent the first semester in the industry and the second semester working on this project and writing the report. A three-month overall timeline was broken down into milestones or sprints to systematically track the development and fix issues that came up. The first phase involved requirements gathering, analysis, and system design, including database schema creation, role-based access definitions, and overall process flow of the system. This laid a solid foundation for the development phase and organized the work. The second phase was development of the system. For easier work management, the system was divided into requirement-based splits, which means one requirement feature was built and tested before starting the next one. For each requirement, first the frontend was built with input fields, tables, and other necessary components for a user-friendly system. After that, the backend logic was implemented for the feature to work, and finally, the frontend and backend were connected through APIs.

Budget planning was a crucial part of the project. As a student developer building an academic prototype, I made sure that all the software, hardware, and hosting platforms used for this system were free and open-sourced. As the only developer of the system, the sprint planning and work distribution had to be efficient. An iterative and incremental approach was taken for the development, where a feature was built and tested before moving to a new one. Git was used to maintain the version control of the system, as any minor change to the code was made against the code repository. The process of test, code, and refine was a cyclical process that made sure that every module was in place as ex-

pected before it was implemented into the system. The project management plan added a formal schedule, resource management, quality, and risk management to create a comprehensive guide that managed to control the successful implementation and construction of the CBS. This is a methodological method that was able to not only lead to the creation of an operational academic prototype but also provided a working implementation of project management in practice that is comparable to commercial software development, with a focus on planning, implementation, monitoring, and responsiveness.

3.6 Risk Management

Risk Management became a crucial factor associated with the construction of the Core Banking System as it assisted in raising the potential risks and how they might be minimized so that they would produce fewer adverse effects. The users presented the project with various categories of risk such as the technical risk, security risk, operational risk and the adaptation risk. The technical risks included flaws in backend API, database issues, and frontend integration issues which would expose the system. To mitigate on these risks, a lot of test was done at various levels to ensure that all the tests that were performed were unit tests on the backends, integration tests on the frontend-backends and manual, functional tests on the end-to-end tests like account creation, transactional tests and loan processing tests. Git version control reduced technical risks through rollback in case of code error or faulty merges. Security risks were high with the privacy of financial data. The system used bcryptjs for hashing of passwords, jwt for authentication, and HTTPS for secure communication. Although the transaction data should have been encrypted, they are not encrypted because of debugging ease. Audit logging was implemented to record user activity to be able to establish accountability and catch any potential security violations. While these added security, complete production-grade protections were not practical given the nature of the project being academic, highlighting the importance of planning and risk awareness. Operational risks were also kept in consideration, like server downtime, deployment failure, and incorrectly setting environment variables. These were managed by utilizing reliable cloud hosting platforms like Render and Neon and testing deployment processing in a staging environment. Time pressure from balancing the project with report writing presented a schedule risk, which was redacted by the official project plan with clearly stated milestones, weekly goals, and buffer time for debugging and refinement. User adaptation risks focused on the risk that end users who were not competent in the digital banking system would have difficulty utilizing the system. This risk was mitigated by the development of simple-to-use, intuitive interfaces, providing role-based dashboards, and providing clear and helpful error messages and navigation feedback. The project also looked to future scalability risks, such as increased user load or database growth, by designing modular backend architecture and normalizing database schemes that would be scalable with minimal working. Money risks were maintained minimal, with the project depending largely on free material and software; any unintentional spending was, however, catered for under a small budget to avoid overruns.

Monitoring and mitigating were continuous processes throughout the project lifecycle. Each risk class had corresponding preventative measures and corrective actions, ensuring that prospective issues were dealt with preemptively and not retroactively. Technical issues were tracked and addressed using iterative testing, debugging, and logging, and functional issues were tracked using system logs and cloud monitoring tools. Security

monitoring consisted of verification of the authentication mechanism and data integrity checks. With a clear risk management strategy, the project not only achieved successful development and implementation of the CBS but also set the standard in software development best practices in risk anticipation, assessment, and mitigation. The formal process minimized interruption, optimized performance, and reinforced the professional and ethical responsibility to provide a stable, secure, and functional banking system.

3.7 Economic Analysis

The core banking system economic analysis takes both the cost of creating and implementing it and the benefits that may occur from its deployment into consideration. On the cost side, the project was started as a low-cost academic pilot project utilizing free or open-source resources and cloud infrastructure in order to reduce expenses. Cloud hosting by Render and PostgreSQL database hosting by Neon enabled robust deployment without significant infrastructure expenditure. The use of open-source libraries such as React, Tailwind CSS, Express.js, bcryptjs, and jwt also enabled the functionality of complete capabilities to be executed without the need for licensing fees. Hardware costs were limited to the developer's own machine, which was adequate for development server, database, and frontend interface execution. Overall, the financial outlay was minimal, demonstrating that effective banking prototypes can be developed and deployed utilizing limited funding.

The benefits of the system extend beyond development at minimal cost, to significant economic worth through efficiency, scalability, and productivity. The CBS achieves efficiency in operations through automation of banking activities, eliminating manual processing, paper-based transactions, and physical infrastructure, leading to cost savings for institutions adopting similar systems. Automated account management, transaction, and loan processing eliminate human errors, improving operations efficacy and dependability. The audit logging feature also provides greater accountability and reduces the losses from fraud or ignorance. To the customer, the system provides cost and time savings with remote access to banking services, thus avoiding travel and reducing dependency on physical branch locations. Even though the prototype is small in size, the model demonstrates the way digital banking solutions can deliver major long-term economic benefits through process automation, the removal of administrative overhead, and improvement in overall service quality.

The cost-benefit analysis also considers scalability and production deployment. While the academic prototype is largely a proof of concept, the fact that it is modular and deployable to the cloud suggests that scaling to accommodate bigger user bases would be possible with more investment. Such benefits as enhanced transaction accuracy, user satisfaction, and operational transparency would only improve with the system scale. Further, by providing a model for secure online banking, the project transfers insight and practices that can reduce the cost of subsequent development for similar systems. In total, the CBS has a favorable economic impact through the union of minimal initial investment and great operating and productivity benefits. It provides a model for low-cost, scalable, and sustainable digital banking solutions with both economic sense and technological efficiency that are institutional and societal economic objectives.

Chapter 4

Proposed Methodology

Developing a core banking system is a strategic, deliberative process that needs to be meticulously carried out in a way to ensure the final product is sound, efficient and scalable to handle the complex needs of today's banks. The banking system should capable to managing a wide variety of functions, ranging from transaction processing to security and compliance, while maintaining high performance with additional warranty and usability. Thus, it is critical to select the appropriate development methodology and technological stack. This chapter outlines the methodology adopted for the design and development of the proposed core banking system, reflecting the choices made to enable the system to be flexible, modular and in a state to adapt to emerging technological innovation and future customer needs.

4.1 Design Process or Methodology Overview

The core banking system is an important tool of any banking institution that facilitates daily banking operations such as transaction processing, account management, and reporting. While building this core banking system, the key aspects that were considered were flexibility and scalability. To maintain modularity and simplicity of the system and keep it scalable to add new features in the future as transactions and the number of users increase, the MERN stack, which includes Mongodb, Express.js, React.js, and Node.js, was used with PostgreSQL as the database instead of mongoDB. React.js and tailwind css was used to build the user interface, or frontend, and Express.js was used for the server, known as the backend as shown in figure 4.1.
This process was built following the agile approach. The agile process is a modern iterative process of software development that puts the needs of the client first and produces high-quality products quickly. Agile processes involve repeated cycles (iterations) that allow for incremental improvements and adjustments based on customer feedback [7]. Following this development process, the system was divided into smaller manageable iterations to make sure each component is heavily tested before starting the next iteration. Core features like role-based access control and account management were given priority to achieve administrative needs. This design was structured to support future improvement as the system grows in complexity. Agile methodologies, such as the Agile Unified Process (AUP), allow for adjustments in project scope and requirements as they evolve [1].
Selection of the modular system architecture was intentional because it enables the developer to upgrade or modify individual components without hampering other functions. A modular web content architecture allows for independent content modules that can be

maintained and updated without affecting the core application, improving overall system robustness [20]. This approach is especially effective for the efficient management of banking systems, where different operations, like account management, transaction processing, etc., need to be functional both independently and cooperatively at the same time.

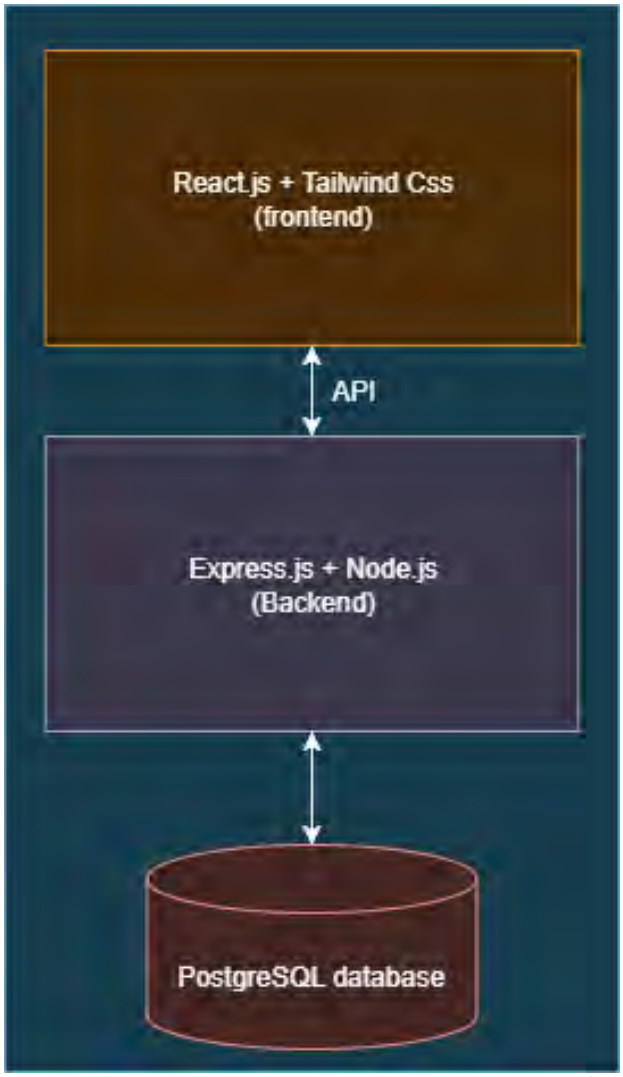

Figure 4.1: Design of the system and it's parts

4.2 Preliminary Design or Design (Model) Specification

At the beginning stage of the project design, different web development architectures were considered to achieve the goal of flexibility and scalability to create an efficient and secure banking system. The three main architectures shown in the figure 4.2 that were considered during that phase are

Monolithic Architecture: A monolithic architecture is a traditional development method where the complete application is built as a single unit. Monolithic systems are easier to develop and deploy, making them ideal for small- to medium-sized applications with limited complexity [25]. This would have made the development process easier because the project was an academic prototype, but considering the scalability factor, this option was ruled out. Because, As applications grow, scaling a monolithic architecture can become

cumbersome, often requiring the entire application to be redeployed for minor changes [25]. Also, as the system evolves, it is harder to make changes in the tightly packed codebase [19]. Finally, A failure in one part of the application can lead to the entire system going down, posing significant risks for critical applications [20].

Modular Monolithic Architecture: A modular monolithic architecture is a design process where the best of both worlds can meet. In this architecture, complex systems are broken down into smaller components so that they can be built, used, and tested independently and be interchangeable while keeping them in a single codebase. It lets the developer create different components separately and call them from a single point of entry. This architecture was chosen because modules can be reused across different applications, reducing redundancy and development time. It also helps in error handling because isolating a component and debugging it doesn't affect other components.

Micro-Service Architecture: Although it has its similarities with modular architecture, micro-service architecture is built with each service or component as an independent and loosely coupled module that can function autonomously. Although this architecture provides greater flexibility and stability, keeping the factor of easier management and maintenance in mind, this architecture was also not chosen for the project.

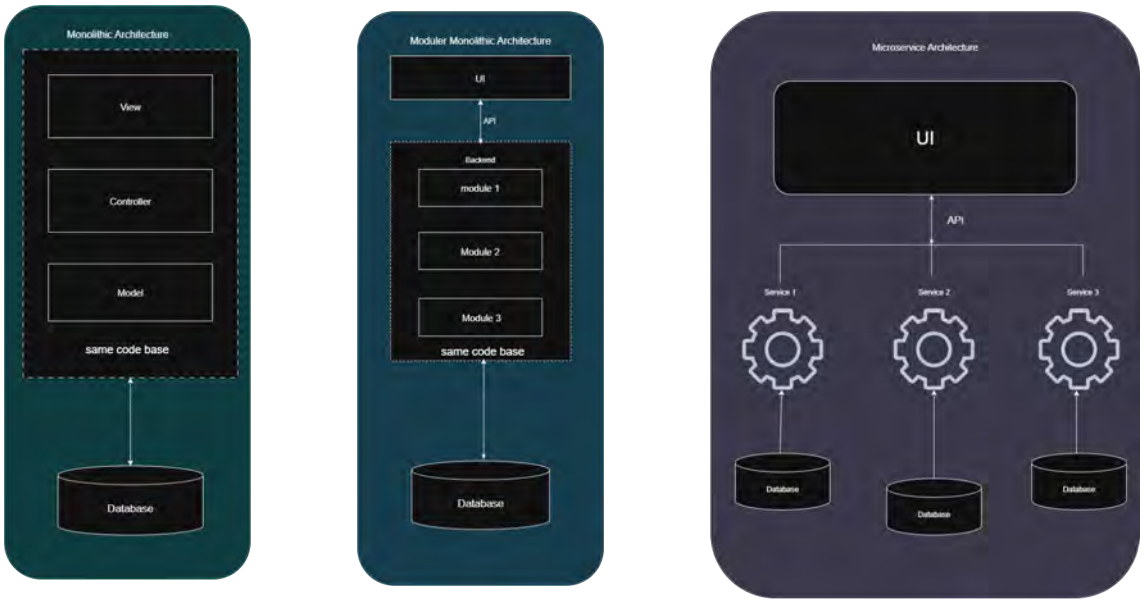

Figure 4.2: Architecture Comparison

From the database standpoint, two options were considered: SQL and NOSQL. PostgreSQL is a highly valued open-source relational database management system known for durability, scalability, and compliance with SQL standards. On the other hand, MongoDB is a leading NOSQL database that has significantly influenced modern data management, unlike traditional relational database systems, mongoDB uses a document-oriented structure, which helps in dynamic queries and the use of JSON like documents. While both offer different advantages, PostgreSQL was chosen for its ACID properties, which refer to Atomicity, Consistency, Isolation, and Durability. These properties are especially important for a banking application because the security of the data is highly valued. MongoDB with NOSQL allows dynamic queries and flexibility through its unstructured database sys-

tem but also compromises the ACID properties for performance and availability.

4.3 Implementation of Selected Design

The final design of the system was chosen, focusing on the modularity and scalability factor for a reliable and efficient banking system. Using the PERN stack for the full-stack application. PERN stack stands for PostgreSQL, Express.js, React.js, and Node.js, where MongoDB was replaced by PostgreSQL, making it different from the regular MERN stack. After the technological stack and architecture were selected, then comes the important part of requirement analysis and coding. As this was built as an academic prototype to showcase the understanding of the banking system and workflow the requirements were very simple to begin with. The application was divided into two main parts: Frontend, the client side and backend, the server side.

4.3.1 Frontend

The frontend was built using React.js. React is an open-source Javascript library to build user interfaces. Vite, a build tool and development server, was used as well. Vite is designed to simplify and accelerate the development process of frameworks like Vue.js and React. To style the UI components, Tailwind CSS was used. It is a utility-first CSS framework that has predefined classes, which allows for highly customizable and responsive user interfaces.

4.3.2 Backend

The backend was created with Express.js, a simple and adaptable Node.js framework that is intended for creating APIs and web applications. For this application the MVC architecture was not used because in the MVC architecture the view works as the user interface, the model works as the database table, and the controller links these two together. But in this application, data is passed through APIs. The frontend and backend for the project were built in parallel for every feature.

4.3.3 Feature Selection and Implementation

Basic banking features were prioritized on logical dependencies, ensuring that each feature supported or built upon the previous one.

Authentication & Authorization

First authentication was implemented. For this, first a table in the database was created named user, which basically has all the information about the user. After the table was created, then comes the frontend part, the register page. This page was built with the input fields needed for the user. The frontend sends the data through the API using the POST method to the database and then into the database. In the login page, the user credentials are entered; it again takes the data to the backend, where user data is pulled from the database using the credentials provided and checked. If found, the authentication is complete, and if not found, the authentication fails. The frontend gets a JWT (JSON Web Token) as confirmation of authentication. This application uses role-based access control.

So, with authentication, authorization happens as well. There are 3 roles in the application: customer, employee, and admin. Role-based access control ensures that users can only access the features permitted for their role. The process can be seen in the diagram 4.3 below.

Database Schema Example:
User: id (pk), email, password, role (customer, employee, admin), created_at, name, phone_number, address.

API Endpoints:

- POST /api/auth/register: Register new user
- POST /api/auth/login: Authenticate and Return JWT

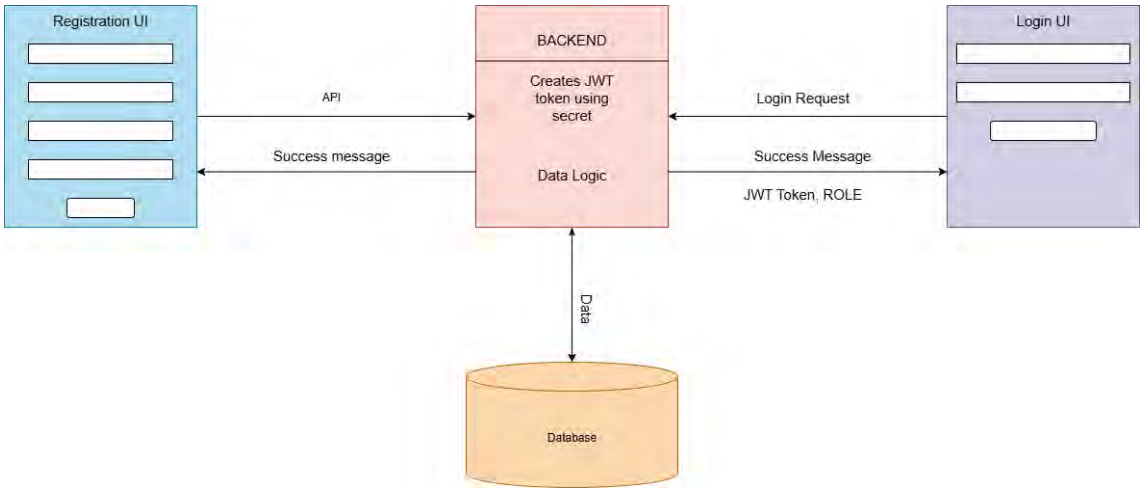

Figure 4.3: Authentication Process

Account Management

The initial plan was to create customer accounts from the application, and employee and admin accounts will be created manually from the backend. That's why initially there was only one register page for the customer only. The plan was changed when the role-based access was introduced in the system. Now customer accounts are created by the employee and employee accounts are created by the admin. Admin accounts still need to be created manually from the backend. The database was changed accordingly by creating 3 more tables for customer, employee, and admin, all referencing the user table. Now when a customer account is created, the usual information is entered with an extra input that is the initial deposit, which is 0.0 by default. The data goes to the backend through api, user data gets stored in the user table, a random account number is generated and assigned to the customer, and the initial deposit gets stored in the customer table. After a customer account is created, an email goes to the customer's provided email address containing the account number and a randomly generated password, which they can change after the first time logging in from their account. Employee accounts are created by admin in the same way customer accounts are created, but for an employee account, employee ID and department also need to be entered. The backend then stores the data into their appropriate tables.

Database Schema: customer: id(fk), account_number, balance
employee: id(fk), employee_id, department

API Endpoints:

- POST: /api/account: Creates customer accounts
- POST: /api/create-employee: Creates employee accounts.

Role-Based Access Control

Role-based access is maintained using both the frontend and backend. The backend authentication middleware checks the jwt token; if the token is valid, it authorizes the user, and if the token is invalid, the user access is rejected. Every route in the backend that has a role-based access checks the user's role from the api request by fetching the user information from the database. If the role is authorized for that route, it is given access and performs the operation. The frontend uses React Router to protect routes from unauthorized users. React router is a react library that is used for navigation and routing within the react application. When the login is successful, the backend sends the token and user role information as a response that is saved in the local storage of the browser the application is running on. When a user tries to enter a route, react router checks the user's role and gives them access if they are allowed in the route.

Table 4.1: Comparison of Functionalities by User Role

Customer	**Employee**	**Admin**
Has their own dashboard	Has their own dashboard	Has their own dashboard
Transaction of their own account	Make transactions on behalf of the customer	Can approve loan request
Request for fund transfer	Make fund transfer on behalf of the customer	Can create employee account
Check transactions	Approve fund transfer request	Can delete customer and employee accounts
Change password	Open customer account	Check logs of all activity
Check loan status and pay EMI	Delete customer account	
Download account statement		

Transactions & Fund Transfer

Transaction is one of the most basic features of the core banking system. In this project, there are 2 main types of transactions. General transactions like withdrawal, deposit, and fund transfer. A table in the database is created named transactions, which has the necessary fields needed for transactions. Because transactions can be initiated by both the

customer and the employee, a user ID in the table was added to keep a record of the user making the transaction. Customer and employee land on the same page for the transaction, where role-based access is set for the customer to directly choose between withdrawal and deposit, set the amount, and initiate the transaction. On the other hand, employees need to enter the account number as well to do the transaction. The api request goes to the backend and, based on the selected operations logic, adds or deducts money from the account, changes the balance in the customer table, and stores the transaction record in the transactions table.
Fund transfer is the kind of transaction when you can send money to another account. The fund transfer is implemented differently for customers and employees. Customers can not directly transfer funds but can request a transfer. When an employee accepts the request, the fund transfer is successfully completed. An employee can do a transfer from the employee dashboard. For fund transfer from the customer side, only the receiver's account number and amount are needed for the request. The request goes to the backend and stores the information in the database. All the fund transfers marked pending can be seen from the employee dashboard. If the employee approves the transfers, the receiver's account gets credited, the approver's information gets stored in the database, and the transfer is marked approved. Employee-initiated fund transfers are directly approved, so they just credit the receiver's account and store the information in the database.

Database Schema:
transactions: id (pk), user_id, transaction_type (withdrawal, deposit), amount, created_at, account_number.

fund_transfers: id (pk), sender_id, receiver_account_number, amount, status, requested_at, requested_by, approved_at, sender_account_number

API Endpoints:

- POST: /api/transaction: making transactions (both employee and customer)
- POST: /api/fund-transfer: making fund transfer (both customer and employee)
- GET: /api/fund-transfer/pending: fetching pending transfer requests
- PUT: /api/fund-transfer/approve/$transferId: approving pending transfer request.

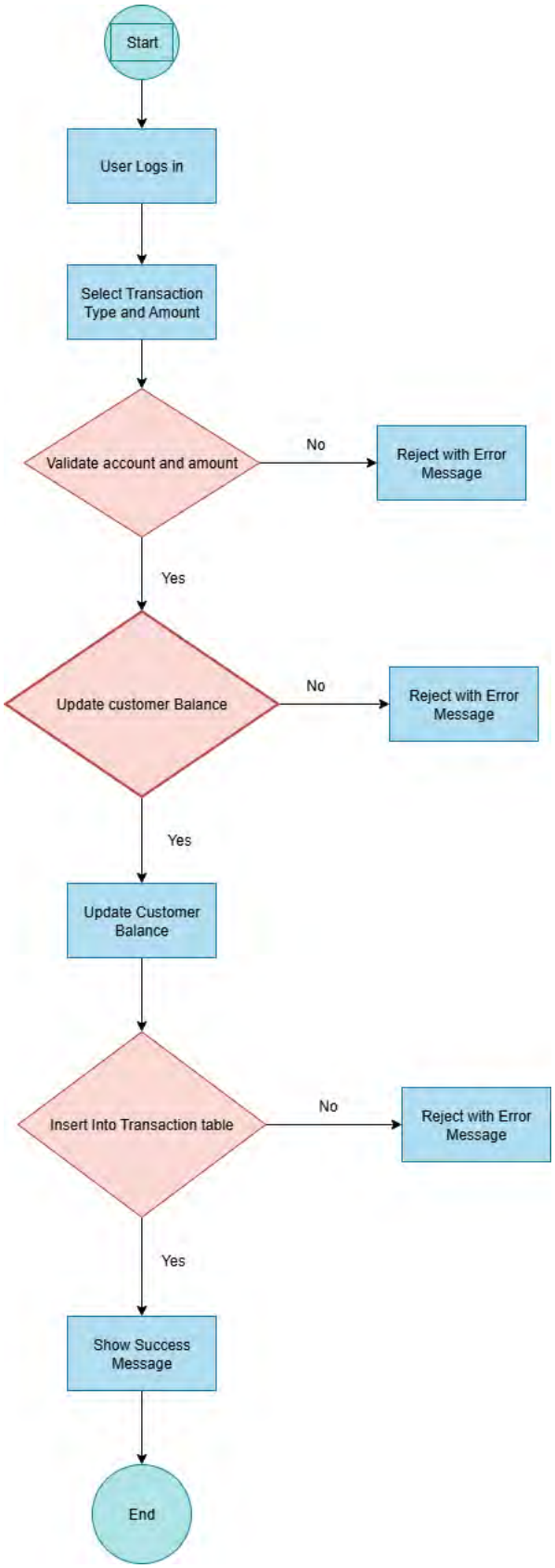

Figure 4.4: Flow Chart for Transaction Process

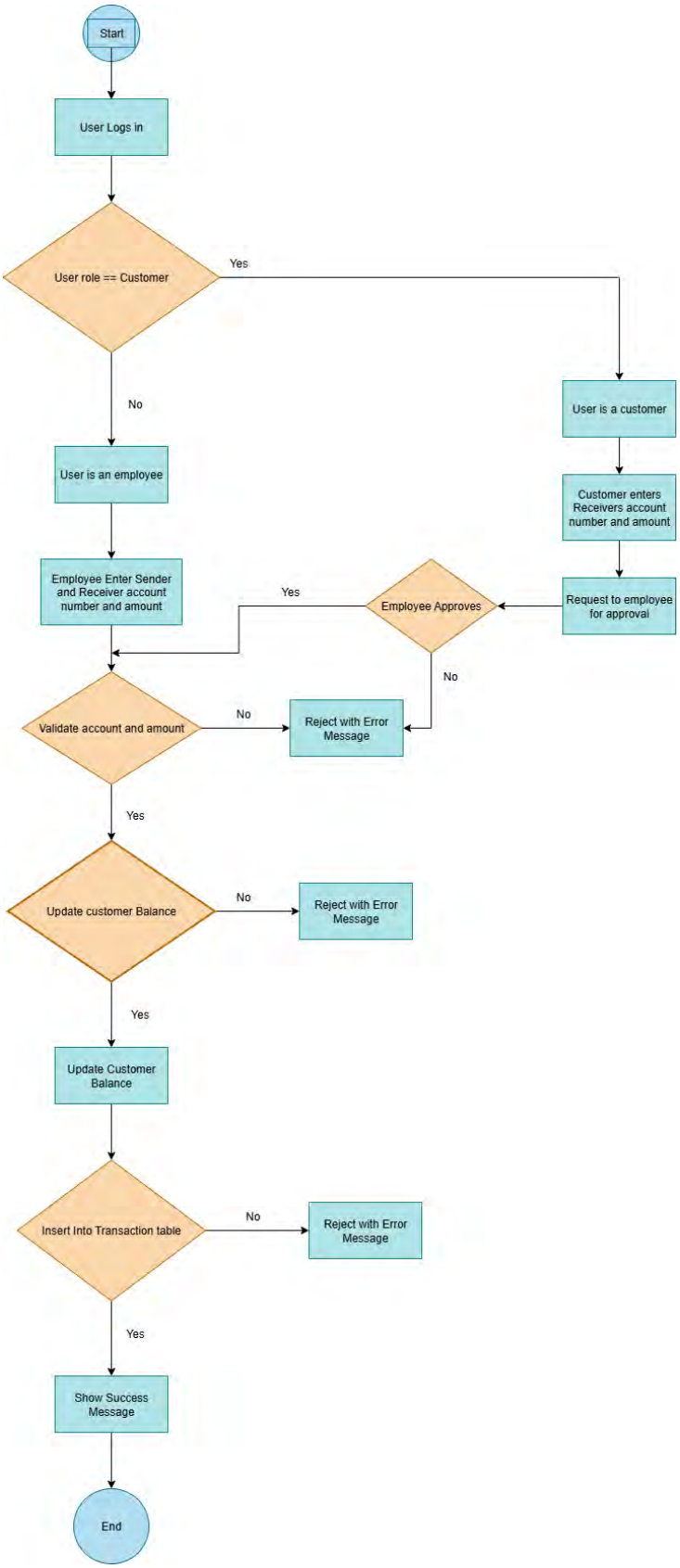

Figure 4.5: Flow Chart for Fund Transfer

Loan Management

Loan management is an important component of the core banking system. Loan requests for customers are initiated by employees. Customers can check loan request status from their dashboard. Only employees can request a loan on behalf of the customer. To request a loan, the employee needs to enter customer details, interest rate, loan duration, and payment delay penalty rate and submit the loan request. The request goes to the backend through the API and gets stored in the loan table marked as pending. The pending loan requests are fetched from the database and shown in the admin dashboard. Admin can either approve or reject the loan. If the loan request is approved, the customer receives an email, and the loan payment gets the data of the customer's pending EMI. The formula of the EMI is

$$\text{EMI} = \frac{P \times R \times (1+R)^N}{(1+R)^N - 1}$$

Where:

- P = Principal loan amount
- R = Monthly interest rate (annual interest rate divided by 12 and converted to decimal)
- N = Number of monthly installments (loan tenure in months)

Customers can check their dashboard and see their payment records. They can directly pay from the dashboard, and the customer's EMI will be paid.

Database Schema:
loans: id(pk), account_number, requested_by, approved_by, amount, term_months, interest_rate, monthly_due, penalty_due, status, requested_at, approved_at, completed_at, purpose.

loan_payments: id(pk), loan_id, due_date, paid_at, amount_due, amount_paid, penalty, status.

API Endpoints:

- POST: /api/loan-request: making loan request
- GET: /api/loan-request/pending: fetching loan requests
- PATCH: /api/loan-approval/approve/$loanId: approving loan request
- PATCH: /api/loan-approval/reject/$loanId: rejecting loan request

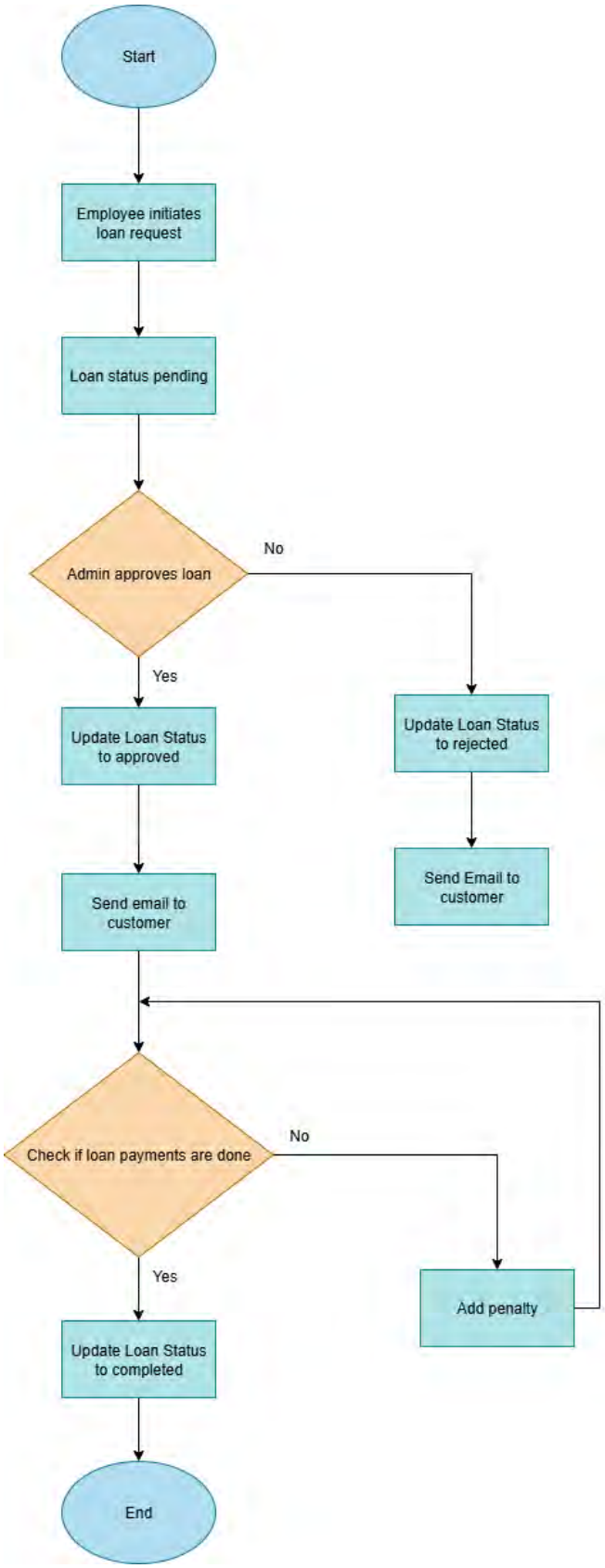

Figure 4.6: Flow chart for Loan Management

Audit Log

Audit logging generally means keeping records of events and activities of an application. In banking systems it is especially important to keep records of activities because a lot of sensitive data and asset security is at stake here. To keep track of activities and audit-logger function is created in the backend server. Every time a successful operation happens, this function is called. This function saves a record of that event in the audit_logs table in the database. This audit log data can be seen from only the admin's dashboard.

Database Schema:
audit_logs: id (pk), action, target_type, metadata, timestamp, user_id, target_id

API Endpoints:

- GET: /api/audit-logs: fetching the audit log data.

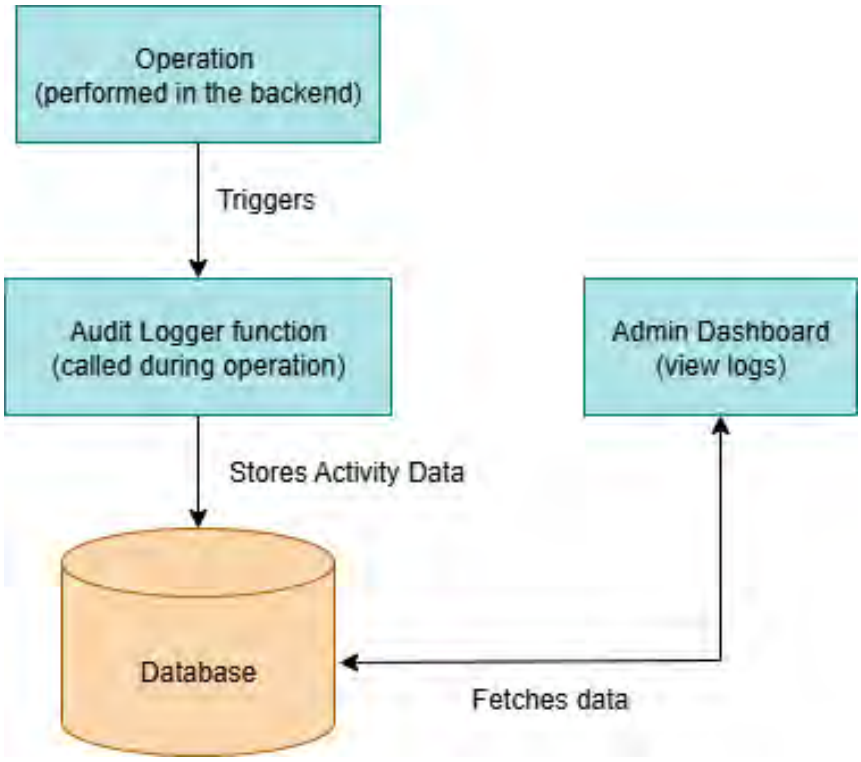

Figure 4.7: Data Flow Diagram for Audit Logger Function

Database Overview

The core banking system database schema is the foundation for the secure management of user identities, transactions, fund transfers and loan services. The schema is a real representation of a three-role banking structure with customers, employees and admins, as well as enabling efficient tracking of financial activities and audit logs for auditing.

The system begins with the user table, which is the central table holding common user information such as name, email, password, role (customer, employee, admin), phone number, address and creation timestamp. All users are identified by a unique auto-incremented id,

which is also used as a foreign key in corresponding role-specific tables. The role column uses a CHECK constraint to define legal user types, ensuring data integrity. To implement role-based data, the database has three linked tables.

Customer's table: Stores customer-specific fields such as a unique account number and balance. One customer has one-to-one relationship with a record in the users table based on the id.

Employee's table: Stores employee-specific fields such as an individual employee id and his or her department.

Admin's table: Tracks administrative users using a unique admin id and a description of their role.

The transaction table records customer and employee deposits or withdrawals. It is joined to the user table by user id and optionally to the customer table by account number. Transaction type is restricted to 'deposit' and 'withdraw' so that when finances are examined, there is no room for confusion.

To facilitate fund transfers between peers, money is moved from one account to another within the fund transfer table. It contains foreign keys referencing the sender id and both the sender's account number and the receiver's account numbers. It contains a status field (pending, approved, and rejected) that facilitates workflow for checking and approval by employees. Columns like 'approved by' and 'approved' at track who approved the transfer.

For credit services the loans table allows employees to request loans on the customer's behalf. Admins may reject or approve them. Each loan has fields such as amount, term duration, interest rate, monthly due, and penalty rate for late payments. Status tracking (pending, approved, rejected, completed) allows tracking throughout the life of the loan. A new purpose column was introduced to provide meaning to the loan request. Finally, loan_payments manages the installment-by-installment repayments. Each record is tied to a loan and indicated the due date, amount due, amount paid, penalty assessed, and payment status (due, paid, or past due)

The audit_logs table stores system events such as transaction or loan updates. This log uses a flexible schema with a type JSONB metadata column, hence allowing it to hold dynamic information without a predefined schema. Each log holds the user who caused the action, the resource type and id impacted, and a time stamp.

This modular and normalized database schema is scalable, easy to maintain, and highly dependable in processing diversified banking operations, ranging from user management to financial transactions, fund transfers, and loan servicing. It is full-stack application compatible and fulfills the essential demands of a rudimentary but functional CBS prototype.

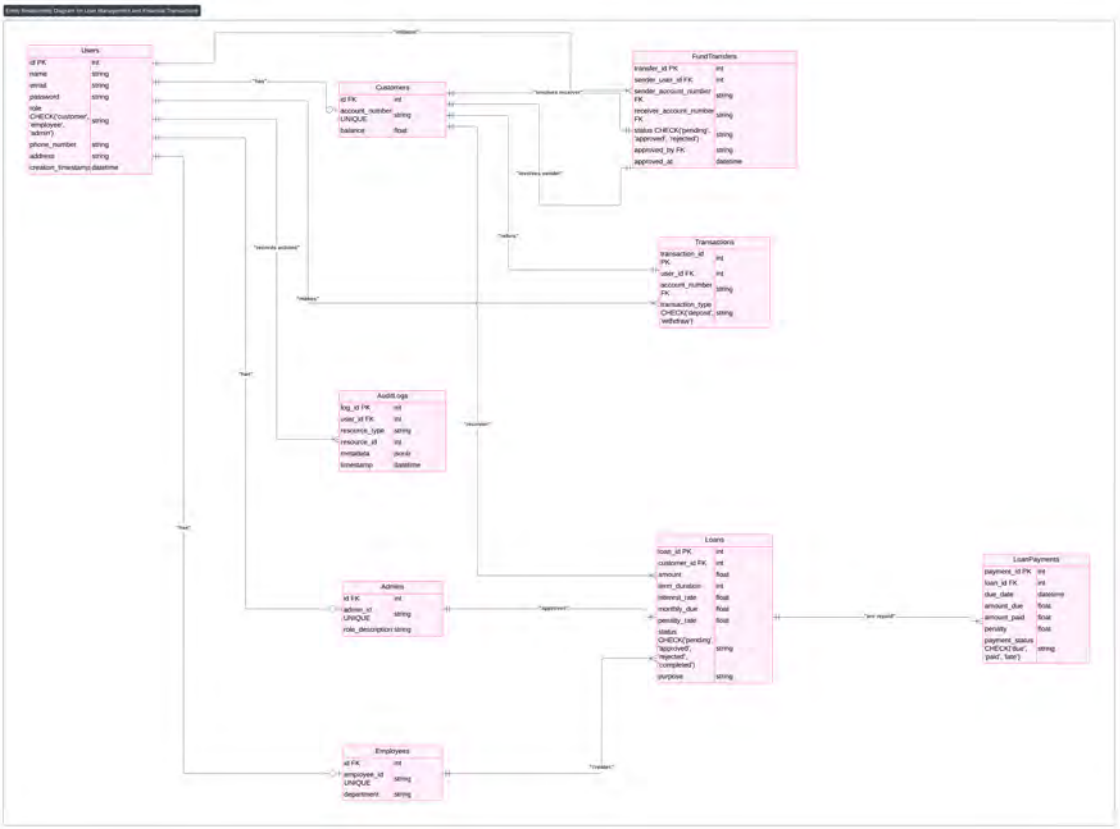

Figure 4.8: Database ER Diagram

Chapter 5

Result Analysis

5.1 Performance Evaluation

5.1 Performance Evaluation The system performance test of the Core Banking System (CBS) involved a detailed verification of the system's response, reliability, efficiency, and accuracy. The test accounted for the system's ability to fulfill its intended functions for different stakeholders like the employees, customers, and administrators. Accuracy was a central metric because the system needed to accurately report all transactions, maintain account balances, and maintain records of loans without error. During testing, different scenarios were put in place, including deposits, withdrawals, fund transfers, loan requests, and repayments. All the operations were verified to ensure that database records had the correct information. The test ensured that the system consistently kept the financial data with integrity, no mismatched balances, and no incorrect updates encountered. Efficiency was quantified by looking into the response time of API endpoints within typical usage situations. The backend APIs built using Express.js reacted in between 2 and 5 seconds for common operations like transaction processing and account inquiry. Response time confirmed that the system was capable of processing normal banking activities with no perceived delay, with users being able to experience a smooth and interactive experience. Reliability was verified using repeated operations for different user roles and system stability. The CBS handled role-specific operations and data retrieval requests without crashing, a sign of high reliability for the system. Also, the system's ability to maintain session integrity and role-based access during testing was an expression of its capability in real usage.

Testing strategies included both manual and automated testing approaches. Automated unit tests ensured backend operations to check isolated operations such as account opening, fund transfer, and loan processing worked as intended in isolation. Manual integration testing included the recreation of behaviors that duplicated real-life situations using the frontend dashboards to ensure that user interactions caused the correct backend processes to be invoked and that the output displayed was accurate. Audit logs were verified during performance testing to ensure all the key actions the user took were being correctly recorded for traceability and accountability. Through the combination of these assessment criteria, accuracy, efficiency, dependability, and test procedures, the performance test gave a general overview of the strength areas and improvement areas of the system. Generally, the CBS proved to function effectively in its aimed functionalities, with it providing an efficient, secure, and precise platform for online banking operations, as well as simplifying

the field of scalability and security upgrades for implementation at the production level.

5.2 Analysis of Design Solutions

The CBS design solutions were analyzed in terms of accuracy, efficiency, feasibility, cost-effectiveness, and performance as a whole. The backend APIs were modularized and scalable to simplify integration and maintenance. Each module, such as account management, transactions, loans, and audit logging, was constructed as a standalone functional unit so that it can be tested and optimized separately afterwards. This module-based design simplified maintenance and lowered the risk of error propagation in development. Frontend dashboards were constructed using React.js and role-based dynamic rendering that only added functionality available to customers, employees, and administrators depending on their role. User experience and system security were improved by preventing unauthorized access. There was improved efficiency via better database queries that were optimized and minimized API calls, thereby decreasing latency and response time. Front-end and back-end design integration enabled easy transmission of information such that fund transfers, history of transactions, and servicing of loans were made easy and rapid.

The viability of the design options was evaluated in light of resources, schedule, and deployment needs of the project. The use of open-source software and free cloud hosting helped in the viability of deploying the system at little to no financial expense while maintaining functional integrity. Moreover, the employment of a relational database like PostgreSQL enabled organized storage and retrieval of bank data, providing a reliable platform for sophisticated operations like multi-step transactions and logging auditing. Cost-effectiveness was ensured by careful use of provided utilities, such as open-source authentication libraries, password hashing, and UI components. Emphasized design advantages were modularity, role-based access control, audit logging, and a responsive, clean UI. Weak points were also pointed out, namely in security and scalability areas. While bcrypt hashing and JWT authentication added negligible security, superior protection against XSS, SQL injection, or encryption of transactional data was not part of the system in order to restrict its preparedness for real-world deployment. Additionally, EMI calculations for loans were manually done rather than automatically, something that will need automation in future versions. Overall, the analysis confirmed that the design solutions adequately solved the academic needs of the project in presenting an effective, reliable, and functioning digital banking system with room for improvement in security, automation, and scalability.

5.3 Final Design Adjustments

Based on the performance and design analysis, some final changes and improvements are done with the system. First of all, when the system was built, it accounted for a single branch, but a financial institution like a bank should have multiple branches to reach the people everywhere. That's why the branch system was integrated into the project. Although the branch has a significant effect on the project, the changes were mainly done in the database and backend. Initially, a table named 'branches' was created, and a branch column was added to employee and customer tables to categorize them using their branch. In order to assign the branch to any customer or employee from the frontend, it was then

integrated into the forms for creating employee accounts and customer accounts. Branch table has attributes like 'id,' which refers to the branch id and also the primary key, branch name, address, and created_at. The branch ID is used in other tables as a foreign key to keep records. Secondly, some rules based on the user's branch are put in place, like fund-transfer requests can only be accepted by the employees of the same branch, and loan requests have to be made by employees of the same branch. This change completed the project and gave the system a real-world feel.

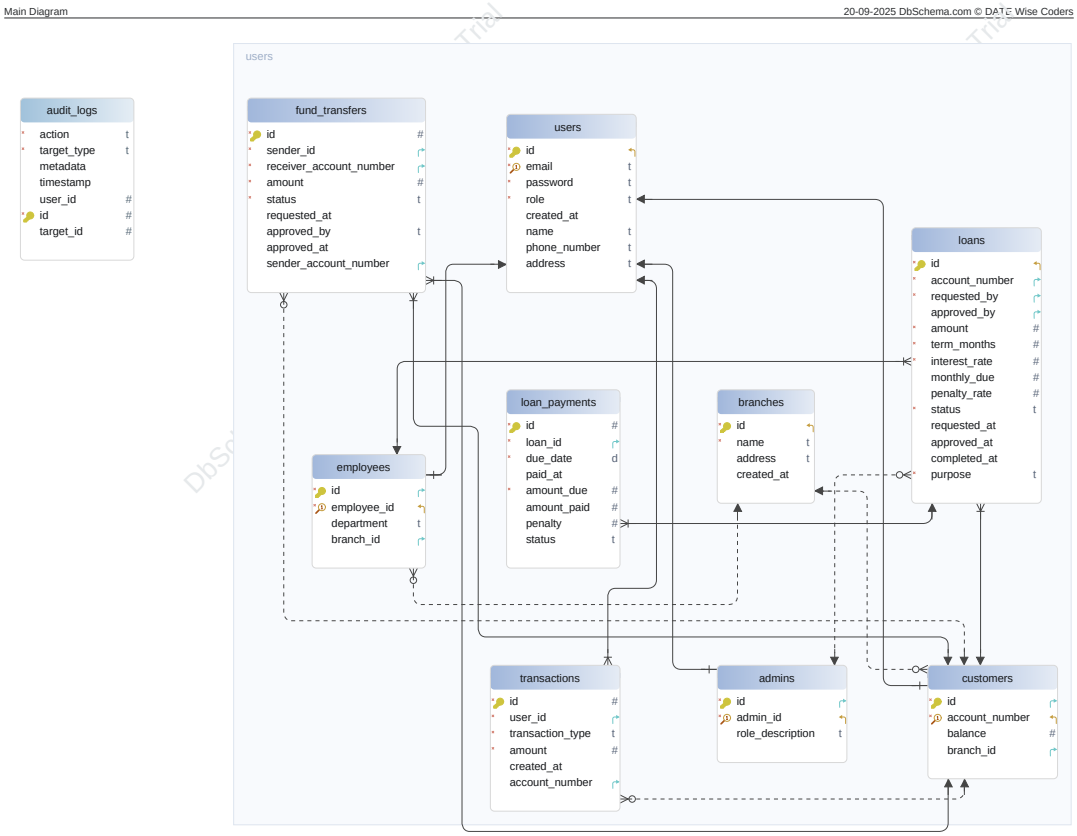

Figure 5.1: Database ER Diagram

Another mentionable change has been done to the FAQ section of the customer. The customer can read answers for frequently asked questions, and if they are still having problems, they can mail about their problem directly from the system. To send mail directly from the user dashboard, EmailJS was used. Some other changes in the UI are done to make the system more usable and accessible. Finally, the system was deployed and tested to check for potential issues. Overall, the final changes in UI and adding branch to the system completed the project and it is set for demonstration and real-world application.

5.4 Statistical Analysis

The statistical evaluation of the Core Banking System (CBS) was conducted in order to determine its accuracy, efficiency, and reliability under different modes of utilization. Since this project is a more academic prototype than a big commercial setup, the statistical evalu-

ation was done based on performance measures obtained through testing, simulated transaction behavior, and role-based user activity. To estimate reliability, standard banking operations, such as deposits, withdrawals, fund transfers, and loan requests, were experimented upon repeatedly. Recording accuracy was up to 100% in all the simulated transactions after it was ready for deployment, with each operation registered correctly within the database without conflict. This suggests the system is statistically reliable at the size for which it was tested. Efficiency was tested by comparing API response times for critical operations. Account inquiry response was averaged at 2 seconds, while fund transfer and loan operations were averaged at 2-3 seconds. These results may vary depending on the device configuration and available internet speed. These results are comparable to tolerable web-based banking application response times, which show the capability of the system to accommodate normal workloads without noticeable lag

Statistical evaluation of the CBS prototype exhibited a minor failure rate below 3-5%, attributed primarily to timeouts during deployment across the free-tier hosting environment, which was addressed by environment configuration and API call optimization. Role-based access control was experimented with user activity analysis, customer transactions, employee account management, and admin loan approval, and in all cases, permission enforcement was illustrated to be efficient. This validates the prototype's correctness and stability for academic use but recognizes scalability and error handling as the areas to be optimized to achieve production level, with its potential to be a stable system in managed environments.

5.5 Comparisons and Relationships

To see the effectiveness of the Core Banking System (CBS) in reality, one has to compete on the basis of functionality and performance in comparison to online banking systems currently being applied, including those that are provided by big shareholders of this industry, such as Bkash, Nagad, and banks such as City Bank and Dutch Bangla Bank. Relative to these effective frameworks, the CBS is robust in its thin, academically biased design, specifically in rapid prototyping, modular design, and core feature integration. As an example, CBS offers high-end features like fund transfer, account processing, audit logging, and handling of loans with caution in the direction of ease and transparency, with millisecond-range response times of operations like balance validation that are competitive with the performance at minimal enterprise-grade infrastructure cost that is offered by most web-based services. Its customers, employees, and admins have a multi-role access hierarchy system that is a reflection of the real hierarchies of individuals in applications such as these renowned banks and the log audit that is tamper proof. Paperwork and branch visits are also needed by banks when opening accounts, conducting transactions, and obtaining bank statements. However, this project attempted to reduce paper consumption and bank visits by ensuring the use of papers was held down to a minimum and operations were done remotely.

But comparing the CBS with similar internet banking systems reveals a number of inadequacies to the system that indicate areas for improvement. Business systems will typically have advanced security features implemented, such as end-to-end encryption, biometric authentication, and AI-based fraud detection, which the CBS does not have and hence may be more vulnerable to risk in riskier environments. Amenities such as timely automated

loan repayment, multi-currency, real-time global wires, and third-party service integration (for example, APIs for investment portals or budget management tools) are par for the course in organizations such as HSBC's digital banking or Stripe's financial services but not in the CBS's prototype design. Scalability is the other key differentiator: existing platforms employ microservices, cloud load balancing, and database-crafted tuning to handle millions of simultaneous users, whereas the CBS leverages a simpler architecture for smaller-scale installations. Even with all these shortcomings, the CBS proves its value as a building block system by reconnecting theory in academia to practical usability. This evaluation emphasizes that while the CBS is not able to match the power of established online products, it offers a functional, active alternative for studying or minor-scale use, with significant room to evolve into more direct competition within the digital bank space.

5.6 Discussions

The result of the Core Banking System project reflects both the achievement and flaws of the implemented design. On the positive side, the system achieved its primary objectives of having an operational banking model that supports multiple roles, like customers, employees, and admins, while enabling core activities such as account opening, transactions, fund transfers, loan processing, and audit logging. The system performance evaluation and the statistical analysis were high and showed the high accuracy and reliability of the system, thereby ensuring the efficiency of the design decisions, including the modular backend APIs and the role-based access control. Deployment to a cloud-based environment also ensured system feasibility in the real-world-like environment with secure remote access. These findings illustrate the project's contribution to academic learning and its potential applicability in the development of digital finance platforms for small institutions or training environments. The implication of these findings is significant: it shows that complex financial systems are possible using open-source software and with limited resources, and they become accessible to students, small businesses, and low-budget organizations.

But to challenge the CBS against these internet banking offerings demonstrates that there are some shortcomings in the system that lead toward areas in which improvement can take place. Business systems will typically include advanced security integrated into them, such as end-to-end encryption, biometric verification, and AI-based fraud prevention, which the CBS is still lacking, hence potentially being more vulnerable in riskier environments. Functions such as automated scheduling of loan repayments, multi-currency, real-time worldwide wire transfer, and third-party integration of services (e.g., APIs to budgeting software or investment sites). However, the shortcomings discovered in the evaluation process also indicate to what degree things may be refined and perfected. The system is stable when it comes to learning test conditions but not scalable due to the reliance on free-tier hosting and not having sophisticated load-balancing systems. Security, while controlled through password hashing, JWT authentication, and HTTPS, is not ready for production since the system does not have protections against attacks such as XSS attacks and data encryption vulnerabilities. These lapses indicate the need for production-grade improvements in the later versions. Manual processing of some processes, like loan repayment tables and EMI calculations, is another limitation to the degree of system automation. Despite such limitations, CBS is highly viable as a research platform, particularly with sophisticated features like predictive analytics for fraud detection, automated loan management, and multi-currency transactions incorporated into it. Lastly, the presentation proves

that the CBS fulfilled academic objectives and effectively developed the concept of digital banking systems. This approach also represents a constant area of innovation, bridging the gap between higher education designs and real financial implementations, emphasizing security, scalability, and automation in business applications.

Chapter 6

Conclusion

6.1 Summary of Findings

It has been a difficult but enlightening research and practical project to design and implement the Core Banking System (CBS). The project began with a vision to develop a secure, role-based banking system that would be able to serve several parties, customers, employees, and administrators, in a given and extensible platform. Through a series of stages in requirement analysis, architectural design, implementation, and testing, the system arrived at a level of functionality that can effectively demonstrate how current-day bank services can be redesigned leveraging contemporary web technologies and relational database systems. The outcome of this project can be separated into a series of ruling themes. Firstly, the system demonstrates that PostgreSQL, together with an Express.js backend and React.js frontend, can constitute a suitable platform for financial applications requiring flexibility as well. The database schema created for the CBS could handle customers, employees, administrators, transactions, loans, and audit trails relationally and consistently. This allowed the system to achieve main functionalities such as customer account registration, secure login with role-based authentication, deposits and withdrawals, funds transfer, loan application and approval, and extensive audit logging. JWT authentication provided strict access control along with role sensitivity so that no unauthorized users could view features beyond their capabilities. Secondly, the project stressed the importance of modular development in banking systems. Each role (customer, employee, admin) had its own dashboard and feature set, which allowed separation of concerns and better maintainability. This modularity made testing, debugging, and additional development simpler. It also makes the system scalable, because new features could be introduced without disrupting the ones already there.

Another finding was the importance of incorporating security features in the project in the initial stages. Although the academic prototype did not have thorough incorporation of newer security features such as encryption of data in transactions, multi-factor authentication, or anti-XSS filters, the project also brought out the need for such protection in a production-quality financial application. Similarly, the computation of EMI for loan repayment through manual code was functional but brought out the need for automated, accurate, and scalable financial calculations in real-world banking systems. From a deployment point of view, the project proved that cloud platforms such as Render (for backend) and Neon (for PostgreSQL) can be used to host academic prototypes at low effort and cost. In spite of some latency problems, successful deployment proved the feasibility of cloud

banking systems, particularly for those entities that cannot afford to retain on-premise infrastructure due to lack of finances.

In summary, the findings emphasize that while the CBS project is an academic prototype, it has huge potential to grow into a production-level banking platform with security enhancements, optimization of performance, and compliance with financial regulation. The system has been able to demonstrate not only the technical feasibility of implementing core banking transactions using modern full-stack tools but also the wisdom of coding software that achieves a balance between usability, security, and maintenance.

6.2 Contributions to the Field

The Core Banking System (CBS) designed in this project contributes to both computer science academic studies and practical banking technology in several ways. It is an academic contribution through presenting a case study to computer science students and researchers on how theoretical abstractions like relational database design, authentication means, and modular system structure can be achieved in the real world. By showing the proof of concept for requirements being converted into working modules, i.e., dashboards, fund transfer, and audit logs, the project bridges the gap between abstract academia and practice.

Technologically, the CBS project is valuable in that it shows how modern full-stack web development frameworks can be employed to simulate required banking features. Previous academic prototypes have been guilty of either simplistically oversimplifying or overly relying on legacy technologies. This project demonstrates that React, Express, and PostgreSQL, when well combined together, can serve as an effective, dependable, and scalable foundation for financial applications. It also demonstrates how relational databases hosted on the cloud, like Neon, can be an alternative to traditional on-premise banking databases so that one can more easily deploy and scale them.

One of the standout features of this project is role-based authentication and dashboards. Instead of developing a single user interface for all, the CBS ensures that admins, employees, and customers only access features relevant to their position. Not only is this more secure but also reflects actual banking procedures, where responsibilities are allocated between parties clearly. This can be utilized as a template for future academic projects or even startups intending to develop finance management systems.

Another contribution is the audit logging system that records significant action throughout the platform. While audit logging is usually an afterthought in student projects, its inclusion here reflects professional responsibility and industry best practices. Audit trails play a critical role in compliance, accountability, and fraud detection in bank systems, and their inclusion in the project ensures students understand their value beyond functionality.

Second, the project contributes to knowledge about limits in academic prototypes. By clearly announcing where the system falls short, i.e., in such areas as encryption, calculation of EMI on an automated basis, and financial standards compliance, the project not only specifies what is done but also sets a foundation upon which other work can proceed. This reflexive contribution is necessary, since it educates both the academic and industrial

practitioner communities about the challenges of producing secure, production-level systems.

Finally, this project furthers the overall societal goal of financial inclusion. By showing that a cheap, modular, and scalable banking system can be built using open-source technologies, the CBS project implies the possibility of building low-cost banking platforms for poor or underserved communities or small financial institutions. In that sense, the project ventures outside of the academic community and might serve as a possible spur to yet more innovations in community banking and fintech.

6.3 Recommendations for Future Work

Although the CBS project succeeded in achieving its purpose and was an academic demonstration of conceptual validity, it can be greatly improved to be prepared as a production-level bank solution. It is possible to identify several recommendations on the future work.

First of all, the security architecture of the system will have to be extended. Production level systems need more protection although JWT-based authentication was successfully performed. These consist of end-to-end encryption of sensitive information, the implementation of HTTPS and the use of the SSL/TLS algorithm, the implementation of the multi-factor authentication (MFA), more secure password hashing algorithms with salting. The system should also provide protection against cross-site scripting (XSS) attacks and distributed denial-of-service (DDoS) attacks. In the absence of these, the system would not be secure enough to be deployed.

Second, this loan repayment module must be more advanced. Presently, the calculations of EMI is done manually, which is not befitting a bank system with a large number of customers and loan tenures. Automatic EMI calculation, incorporation of amortization schedule and dynamic penalty calculation of late payment should be taken up in a future version. Moreover, the interest rate simulations can be incorporated in the system, whereby the employees and the customers will be able to observe the long-term repayment effects.

Third, 3rd party payment gateways or financial APIs would be included in the transaction mechanism. As an example, it would be possible to integrate it with other systems like Bkash or Nagad so that funds would be transferred between banks and not only within the CBS prototype in real-time. This would lead the project to a complete new stage of open academic system to the one, which can interface the entire financial universe.

Fourth, the system must be optimized in terms of scalability and performance. Although the prototype can be done with Render and Neon, scale-out deployment would need load balancing, caching and indexing of the database. Also, it is possible to switch to custom cloud infrastructure like AWS or GCP to enhance reliability and minimize the latency. Stress testing should also be carried out with artificial loads, to guarantee that the system can be scaled by loading the system heavily.

The other suggestion is to incorporate elements of compliance. The banking systems in the real world must be regulated in line with the Data security regulations like GDPR, payment regulation like PCI-DSS, and risk control regulation like Basel III. Likewise, banking reg-

ulations of banks and banking systems in Bangladesh Bank. These compliance elements would not only make the system more realistic, but would also equip the system with the real-life financial use.

Usability wise, the customer interface can be improved with assistance provided by means of chatbots and data visualization dashboards wherein the customers can view spending patterns, loan payment schedules, and savings patterns. The admin dashboard also can be upgraded with more high-end reporting, AI-supported fraud detection, and predictive financial decision-making analytics.

Lastly, the system can consider using the more recent technologies, like blockchain, to make transactions transparent, machine learning, and natural language processing to provide automated customer service or chatbots. The functionalities will make the CBS more innovative and fintech future-proof. Overall, as far as the CBS project has provided a solid groundwork, the future of the project is in expanding security, scaling, compliance and integration with newer technologies to a new level. By mapping out these paths, the project will be able to expand to a point where it will be an academic demonstration of an idea and then become a functioning banking system that will enable financial institutions to provide secure, stable, and inclusive banking services.

Bibliography

[1] C. Edeki, "Agile unified process," *International Journal of Computer Science and Mobile Applications (IJCSMA)*, vol. 1, no. 3, pp. 13–17, 2013. [Online]. Available: http://www.ijcsma.com/publications/september2013/V1I304.pdf

[2] W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," in *Proceedings of the Software Engineering Conference (SE)*, 2014, p. 29. [Online]. Available: http://dblp.uni-trier.de/db/conf/se/se2014.html#MaalejR14

[3] S. D. Fernando, *Core banking systems and business intelligence for effective strategic decision making*, 2015. [Online]. Available: http://repository.kln.ac.lk/jspui/handle/123456789/10588

[4] M. A. Rahman and X. Qi, "Core banking software (cbs) implementation challenges of e-banking: An exploratory study on bangladeshi banks," *Social Science Research Network*, 2016. DOI: 10.2139/SSRN.2884269 [Online]. Available: https://doi.org/10.2139/SSRN.2884269

[5] C. J. Satish and M. Anand, "Software documentation management issues and practices: A survey," *Indian Journal of Science and Technology*, vol. 9, no. 20, pp. 1–7, 2016. DOI: 10.17485/IJST/2016/V9I20/86869 [Online]. Available: https://doi.org/10.17485/IJST/2016/V9I20/86869

[6] R. Gadge, "The prior and after adoption of cbs (core banking solution) system and its influence on kpis (key performance indicators) in urban co-operative banks: An empirical study," in *2017 International Conference on Data Management, Analytics and Innovation (ICDMAI)*, 2017, pp. 102–110. DOI: 10.1109/ICDMAI.2017.8073493

[7] R. Shah, "A literature review on agile model methodology in software development," *International Journal of Advance Research and Innovative Ideas in Education*, vol. 3, no. 6, pp. 293–297, 2017. [Online]. Available: https://ijariie.com/AdminUploadPdf/A_Literature_Review_on_Agile_Model_Methodology_in_software_Development_ijariie6941.pdf

[8] E. Aghajani et al., "Software documentation issues unveiled," in *International Conference on Software Engineering*, 2019, pp. 1199–1210. DOI: 10.1109/ICSE.2019.00122 [Online]. Available: https://doi.org/10.1109/ICSE.2019.00122

[9] G. Rong, Z. Jin, H. Zhang, Y. Zhang, W. Ye, and D. Shao, "Devdocops: Enabling continuous documentation in alignment with devops," *Software - Practice and Experience*, vol. 50, no. 3, pp. 210–226, 2020. DOI: 10.1002/SPE.2770 [Online]. Available: https://doi.org/10.1002/SPE.2770

[10] J. Y. Khan, M. T. I. Khondaker, G. Uddin, and A. Iqbal, *Automatic detection of five api documentation smells: Practitioners' perspectives*, 2021. arXiv: arXiv:2102.08486 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2102.08486

[11] A. Skeja and S. Xhaferi, *Fintech as the accelerator of the future of banking systems*, 2021. DOI: 10.29322/ijsrp.11.05.2021.p11329 [Online]. Available: https://doi.org/10.29322/ijsrp.11.05.2021.p11329

[12] *Mern stack unveiled: A research study on the technology's architecture and benefits*, 2023. DOI: 10.52783/tjjpt.v44.i1.2215 [Online]. Available: https://doi.org/10.52783/tjjpt.v44.i1.2215

[13] S. Nadi, "Evaluating software documentation quality," in *IEEE Working Conference on Mining Software Repositories*, 2023, pp. 67–78. DOI: 10.1109/MSR59073.2023.00023 [Online]. Available: https://doi.org/10.1109/MSR59073.2023.00023

[14] M. Nassif and M. P. Robillard, *A field study of developer documentation format*, 2023. DOI: 10.1145/3544549.3585767 [Online]. Available: https://doi.org/10.1145/3544549.3585767

[15] A. Simmons, "Banking software services: Current status, challenges, impact and prospects," in 2023, pp. 123–133. DOI: 10.1007/978-981-19-9304-6_13 [Online]. Available: https://doi.org/10.1007/978-981-19-9304-6_13

[16] W. S. Tan, M. Wagner, and C. Treude, "Detecting outdated code element references in software repository documentation," *Empirical Software Engineering*, vol. 29, no. 1, 2023. DOI: 10.1007/s10664-023-10397-6 [Online]. Available: https://doi.org/10.1007/s10664-023-10397-6

[17] A. G. Adeleke, T. O. Sanyaolu, C. P. Efunniyi, L. A. Akwawa, and C. F. Azubuko, "Api integration in fintech: Challenges and best practices," *Finance & Accounting Research Journal*, vol. 6, no. 8, pp. 1531–1554, 2024. DOI: 10.51594/farj.v6i8.1506

[18] K. Dunbar and D. Treku, "Examining the impact of a central bank digital currency on the access to banking," *International Review of Financial Analysis*, 2024. DOI: 10.1016/j.irfa.2024.103220

[19] M. Felisberto, *The trade-offs between monolithic vs. distributed architectures*, https://doi.org/10.48550/arxiv.2405.03619, Preprint on arXiv, 2024. DOI: 10.48550/arxiv.2405.03619

[20] K. Hmue, M. P. Phyu, and A. M. M. Paing, "Microservices vs monolith: A comparative analysis and problem-solving approach in web development area," in *2024 IEEE International Conference on Advanced Information Technology (ICAIT)*, 2024, pp. 1–5. DOI: 10.1109/icait65209.2024.10754922

[21] K. Kumar, V. Srinidhi, B. S. Shamanth, and A. K. Tyagi, "Modern smart banking with cutting-edge technologies," in *Advances in Business Information Systems and Analytics Book Series*, 2024, pp. 73–94. DOI: 10.4018/979-8-3693-4187-2.ch004 [Online]. Available: https://doi.org/10.4018/979-8-3693-4187-2.ch004

[22] B. Kuosuwan, A. Risman, E. Dudukalov, and E. Kozlova, "Digital banking and environmental impact: How fintech supports carbon footprint reduction," *BIO Web of Conferences*, vol. 145, p. 05 017, 2024. DOI: 10.1051/bioconf/202414505017

[23] P. Mukherjee, "Using technological advancements - changing the banking sector's face value in world market," *Indian Scientific Journal Of Research In Engineering And Management*, vol. 08, no. 12, pp. 1–6, 2024. DOI: 10.55041/ijsrem39539 [Online]. Available: https://doi.org/10.55041/ijsrem39539

[24] S. R. G. d. Oliveira and D. Lucrédio, "Guidelines for data engineering documentation in a devdocops approach," 2024, pp. 31–40. DOI: 10.5753/sbcars.2024.3834 [Online]. Available: https://doi.org/10.5753/sbcars.2024.3834

[25] A. A. Padvekar and V. Gupta, "Comparative analysis of monolithic vs. distributed architecture," *International Journal of Advanced Research in Science, Communication and Technology*, pp. 433–442, 2024. DOI: 10.48175/ijarsct-18946

[26] M. Ramaraju, M. V. Varshini, K. Pragathi, L. Suprathika, and V. Sanjay, "Building a banking management system with full stack java," *International Journal of Advanced Research in Science, Communication and Technology*, 2024. DOI: 10.48175/ijarsct-18168 [Online]. Available: https://doi.org/10.48175/ijarsct-18168

[27] G. Bondel, A. Cerit, and F. Matthes, "Challenges of api documentation from a provider perspective and best practices for examples in public web api documentation," in *International Conference on Enterprise Information Systems*, pp. 268–279. DOI: 10.5220/0011089700003179 [Online]. Available: https://doi.org/10.5220/0011089700003179