A TRUST-PRESERVED DIGITAL SAVINGS APPLICATION FOR TRADITIONAL AFRICAN ROTATIONAL CONTRIBUTION SCHEME (ESUSU)

BY

UZOUKWU BLESSING

MAT NO. ENG2002332

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF BENIN

BENIN CITY

APRIL, 2025.

A TRUST-PRESERVED DIGITAL SAVINGS APPLICATION FOR TRADITIONAL AFRICAN ROTATIONAL CONTRIBUTION SCHEME (ESUSU)

BY

UZOUKWU BLESSING

MAT NO. ENG2002332

A PROJECT SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING, FACULTY OF ENGINEERING, UNIVERSITY OF BENIN, BENIN CITY IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF BACHELOR OF ENGINEERING (B.ENG) DEGREE IN COMPUTER ENGINEERING.

APRIL, 2025.

CERTIFICATION

This project was carried out by UZOUKWU BLESSING with MAT NO: ENG2002332 in the Department of Computer Engineering, Faculty of Engineering, University of Benin, Benin City and is hereby certified.

_____ _____

Engr. Syvester Akinbohun **Date**

(Project Supervisor)

_____ _____

Engr. Dr. I. A. Edeoghon **Date**

(Head of Department)

DEDICATION

This project is dedicated to Jehovah God, the ultimate source of wisdom. It is also dedicated to my father Mr. Odion Irummudomon whose innovative spirit and practical approach to solving real-world problems served as my constant inspiration.

ACKNOWLEDGEMENT

First and foremost, I offer my profound gratitude to Jehovah God for the immense strength, wisdom, and perseverance that guided me to the successful completion of this challenging project.

This project's success was significantly shaped by the expert direction and valuable contributions of my supervisor, ENGR. SYVESTER AKINBOHUN. His expert guidance, constructive criticism, and constant support were absolutely crucial in refining the system design and achieving the technical goals.

Deep appreciation goes to my family for their enduring emotional support. Special thanks to my parents, Mr. and Mrs. Irummudomon, for their sacrifices and essential foundation, and to my siblings, Onyekachi Uzoukwu, Ofure Irumudomon, and Henry Uzoukwu, for their continuous motivation.

My technical proficiency with the MERN stack was greatly enhanced by the wider tech community. Special acknowledgment goes to the 3MTT Nigeria community, where peer support and mentorship were pivotal. I also extend sincere thanks to my professional mentors, Shola Iji and Ikechukwu Obasi, for their invaluable encouragement and inspiration during development.

Special acknowledgement is also given to my predecessor, Mr. Patrick Asimonye Chidera Patrick, for his valuable academic advice and supportive insights during the foundational stages of this research

Finally, I appreciate all my colleagues and friends who provided vital support during countless sleepless nights. Thank you to Favour and Omor for their companionship, my class representatives Success Osaze Foster and Jennifer Oseghale for their leadership, and my course mates Olawale Bakare and Faith Nwodo, and all others who served as an invaluable source of peer assistance and collaborative learning.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The traditional African rotational savings system, popularly known as Esusu, has long been a trusted model for communal financial support. However, the manual methods still commonly used, such as notebooks or physical record-keeping, were inefficient, prone to error, and lacked the transparency needed to prevent disputes and mismanagement. This project successfully developed a trust-preserved digital application using the MERN stack specifically designed for Esusu groups. The aim was to promote accountability, simplify group savings management, and maintain the cultural integrity of rotational contributions by replacing manual processes with a secure and intuitive digital solution.

To achieve this, the following key features were successfully implemented: a digital platform solely for rotational group savings, enabling users to create or join existing groups with clearly defined schedules and Payout Numbers. The application incorporated the Dual Balance Display (Group Total and Individual Total Paid) and automated system functionalities, including in-app alerts for critical activities. All members within a group gained real-time access to the contribution history. A logic-based Integrity Alerts System was also implemented to monitor skipped contributions, automatically block suspicious payout claims, and transparently log all admin actions. The core finance mechanism was secured by the Automated Rotational Payout System integrated with the Paystack API.

It is confirmed that a functional, user-friendly, and transparent digital platform for managing Esusu-style group savings was achieved. This justified the need for modernizing traditional savings systems in a way that preserved communal trust while drastically reducing the burden of manual tracking and strengthening administrator accountability through auditable logs. Ultimately, the implemented platform offered a secure, transparent, and culturally familiar solution that enhances the group savings culture in the digital age.

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND OF THE STUDY

The concept of cooperative saving in Africa is a deeply rooted communal practice born from a history of mutual aid and collectives where ordinary people take the time to meet and help each other (Hossein, C. S., & Christabell, P. J. (Eds.). 2022). Long before the advent of formal banking, communities developed rotating savings systems as a means of capital formation, fostering both financial discipline and social cohesion (Iwara, I. O. *et al*., 2021). In Nigeria, this system is widely known as Esusu, Ajo or Isusu (Adeola, O. *et al*, 2022). It originated from the need to pool resources for significant life events and economic activities that were difficult to finance individually, such as funding a marriage, starting a trade, or managing farm expenses (Chika, A.C. 2024). The system operates on a principle of trust, where a group of individuals contributes a fixed sum of money at regular intervals, with each member taking a turn to receive the total collected amount (Boachie, C., & Adu-Darko, E. A. 2022).

Despite its cultural significance, the manual administration of these schemes presents critical operational challenges (Oluwabukola, O.I. *et al*, 2025). A primary issue is the difficulty in diligently tracking every member's contribution in real-time (Zafar, M.B. 2026). In a manually-run group, a member might miss a payment, and this oversight may not be noticed immediately by the collector, who is often reconciling records from memory or notebooks after collections (Juma, M.H. *et al*., 2023). The effect of this unnoticed default becomes apparent on the payout day, when the total collected sum is less than the expected amount (Chika, A.C. 2024). This creates a significant problem for the member whose turn it is to receive the funds, as they are left with a shortfall, disrupting their financial plans. The situation forces the collector to either cover

the deficit or begin a difficult process of tracking down the defaulter, which breeds suspicion and erodes confidence among all members (Siddiqui, A. 2026).

Another prevalent issue is the lack of transparency and dispute resolution (Ethan, D. 2022). Inefficient manual record-keeping can lead to arguments over payment histories, contribution amounts, and payout order (Lamino, P. 2025). Without a clear, indisputable ledger, members can easily fall into conflict, eroding the very trust the system is built upon (Aganze, R. *et al*., 2025). This vulnerability to human error and conflict has limited the scalability and reliability of these traditional systems, leaving them prone to collapse (Ghadamian, R. 2025).

Based on these issues of fraud, inefficiency, and lack of transparency inherent in the manual system, a call for a modern solution was imperative (Batool, Z. *et al*., 2023). The introduction of a trust-preserved digital savings application was necessary to address these loopholes (Muchapireyi, H. T. 2025). By digitizing record-keeping, implementing logic-based Integrity Alerts, and establishing an Automated Rotational Payout System, the application created a transparent, auditable trail of all transactions. Such a robust platform successfully mitigated the risks of mismanagement and fraud, thereby preserving the cultural integrity of Esusu while enhancing its security and reliability for the digital age.

1.2 PROBLEM STATEMENT

Despite the cultural significance and long standing success of traditional African rotational savings schemes, their old-school reliance on manual record-keeping like notebooks caused big problems in today's world. These manual ways, while they worked for a time, were just slow and easily led to mistakes, often meaning crucial money details got lost or members ended up confused.

However, the most pressing issue was the lack of transparency, which basically meant the whole system was easily messed up or even cheated. Because of this, people usually didn't know the real financial state of the group until something major went wrong, like when someone didn't get their expected payout.

Moreover, this lack of oversight created opportunities for delinquent behavior. A frequent headache was when someone simply skipped their payment, and nobody noticed it in the manual book until it was time to give the money to the receiving member. This left the person short-changed and often sparked fights that could break up the entire group. To really stop people from running off or defaulting so easily, the savings platform needed a much stronger way to check who people are and something beyond just a standard sign-up.

While some digital financial tools gained popularity, most available platforms failed to adequately address these specific issues. People were still unsatisfied because these apps often lacked the logic based Integrity Alerts System that immediately flagged skipped payments and publicized all activity. Meanwhile, they had not been able to implement essential high level identity security features like BVN or NIN authentication, which is necessary to enforce maximum accountability and prevent users from easily running away or creating fake accounts after defaulting. This project, therefore, addresses the critical need for a digital savings platform that solves the real world issues of manual operations, inefficiency, and payment defaults, by making total, verifiable transparency and automated activity alerts the central scope, supported by mandatory identity verification to secure the communal funds.

1.3 AIM

The aim of this project is to develop a trust-preserved digital savings application for the traditional African rotational contribution scheme.

1.4 OBJECTIVES OF THE PROJECT

1 **To Develop a Functional Digital Platform:** The primary objective is to design and implement a user-friendly mobile application that replaces traditional manual record-keeping with an automated system for tracking contributions and managing payouts, equipping group administrators with efficient management tools.

2 **To Enhance Security and Transparency:** The project aims to build trust among users by implementing a rule-based system to flag suspicious activities and by providing all members with real-time access to group schedules and contribution timelines.

3 **To Ensure Cultural Relevance and User-Friendliness:** A key goal is to preserve the cultural integrity of the Esusu system by allowing members to follow traditional practices, such as choosing payout numbers, within an intuitive and easy-to-navigate digital interface.

1.5 RELEVANCE OF THE PROJECT

The project is highly important for several reasons, establishing its value for users, the technology sector, and the community.

The project is most relevant to everyday savers and group leaders because it provides a direct solution to the biggest problems in traditional Esusu. The application gives users a secure, transparent, and easy way to save money, helping them avoid the risks of human error, disputes, and mismanagement common when using notebooks. The system's unique strength lies in its logic based Integrity Alerts, which automatically notify everyone about missed payments and suspicious activity. This crucial feature rebuilds trust by ensuring no action goes unseen.

Furthermore, the integration of mandatory BVN or NIN authentication is essential for financial safety, as it dramatically reduces the risk of members defaulting or running away, giving groups true security over their funds.

For the Nigerian technology community, the project offers a successful model for creating financial tools that respect cultural tradition. It proves that modern frameworks like the MERN stack can be specifically tailored to support the unique rotational saving rules of Esusu. This provides a practical blueprint for developers aiming to build inclusive solutions for the informal financial market.

The project supports a broader goal of financial inclusion. By making a popular grassroots savings system more reliable and secure, the application creates a trusted digital entry point for individuals who may not fully trust formal banking. It shows that technology can enhance communal savings and make a positive impact on the financial resilience of the community.

1.6 SCOPE OF THE STUDY

The scope of this project focused exclusively on the design and development of a mobile based digital savings platform tailored specifically for the rotational contribution model of the Esusu system. The central scope was the implementation of total financial transparency and accountability for all group members. The core functionalities delivered included secure user authentication, administration led group creation, automated contribution tracking, and the Automated Rotational Payout System that eliminated the need for manual cash disbursement. A key deliverable within the project's boundaries was the logic based Integrity Alerts System, designed to flag skipped payments and other rule violations, ensuring the community was immediately aware of any potential issues.

The application was developed using the MERN stack and was intended for use primarily by smartphone users, requiring an active internet connection for all transactions and real time updates. Features were also included to control fraud, such as mandatory verification during member joining. The study was strictly limited to the rotational savings model and did not cover other financial products like loans, fixed deposits, or general investment services. Furthermore, the project's scope excluded the implementation of advanced Artificial Intelligence or Machine Learning for fraud monitoring, relying instead on the established, transparent, and auditable rule based logic.

CHAPTER TWO

LITERATURE REVIEWS

2.1 INTRODUCTION

This review explored the core theoretical concepts that provided the foundation for developing a trust preserved digital savings application for traditional African rotational contribution schemes. The key areas of focus included the socio economic principles of Rotating Savings and Credit Associations (ROSCAs), the drive for financial inclusion, the challenges of digital transformation in informal finance, and the technological approach to ensuring security and cultural relevance.

The traditional Esusu system is a classic example of a Rotating Savings and Credit Association (ROSCA), a form of informal finance that has been a cornerstone of community economic resilience for centuries (Zambrano, A. F. *et al*., 2023). ROSCAs function as a collective financial arrangement where members contribute fixed sums at regular intervals, and the pooled fund is distributed to each member in rotation (Chilima, M. (2022). The system is built on a foundation of social capital and mutual trust, which serves as the primary collateral, making it highly effective in communities where access to formal credit is limited (Ardener, S., & Burman, S. (Eds.). (2024). These associations play a crucial role in promoting financial inclusion by providing a reliable mechanism for savings and access to lump sum capital for individuals who are often excluded from the formal banking sector (Phil-Ugochukwu, A. I. 2024).

The global rise of financial technology (Fintech) has spurred a significant push to digitize these traditional informal systems. The primary motivation for this digital transformation was to address the inherent weaknesses of the manual model, such as the risk of fraud, lack of transparency, and operational inefficiencies (Ngowi, A. *et al*., 2025). Digital platforms offered

the potential to create transparent, real time ledgers, automate contributions, and provide a secure and auditable trail of all transactions (Batool, Z. *et al*., 2023). This trend was not isolated to Nigeria, with platforms like BezoMoney in Ghana and OneKitty in Kenya emerging as attempts to bring these benefits to local savings groups. (Abubakar, M., & Sualihu, A. 2025).

However, the digitization of such a trust based system was not without its challenges. A critical concern raised in recent literature was the risk of "trust disruption," where technology, if not designed thoughtfully, could erode the very social fabric it aimed to support (Salako *et al*., 2021). Simply creating a technologically efficient platform was insufficient; the design had to be culturally resonant and user centered to gain acceptance and maintain the communal essence of the practice (Rahman, A. M., & Sultana, S. 2026). For example, Esusu Finance in the United States successfully adapted the model for a different cultural context by focusing on credit building, demonstrating the importance of tailoring features to a specific user base (Jones, L. S., & Maynard Jr, G. 2023). Similarly, platforms like TontineTrust explored blockchain for security, but their complexity was often a barrier for the intended user demographic of traditional Esusu (Milevsky, M. A., & Salisbury, T. S. 2025).

To address security concerns like payment defaults and mismanagement, this project proposed a logic based Integrity Alerts System. Unlike complex artificial intelligence, rule based systems operated on a set of predefined logical conditions (e.g., IF a contribution is missed, THEN flag the user). Furthermore, the project's unique incorporation of BVN or NIN authentication addressed the crucial security gap of member delinquency and fraud. While the Nigerian fintech landscape is robust with platforms like Piggyvest and Cowrywise, their operational model is fundamentally individualistic, focusing on personal savings and investments (Ezeanowi, N. C.

2025). This left a distinct gap for a platform that was built from the ground up to support the communal, rotational, and trust based principles of the traditional Esusu system.

2.2 RELATED WORKS AND EXISTING PLATFORMS

This review analyzed the landscape of existing digital financial platforms to properly position the project and clearly establish the gap the application was designed to fill. A thorough evaluation of applications that had attempted to digitize traditional rotational savings, alongside popular general savings platforms, revealed distinct areas of deficiency in the market concerning communal trust and specialized group management. The analysis ensured that the developed Esusu application was not a mere duplication but a necessary innovation.

The research started by looking at how other people around the world have tried to create formal, official systems for savings groups that rely on trust, just like the traditional Esusu. A key example we looked at was ESUSU FINANCE, which operates in the United States. This company successfully took the basic idea of group savings and changed its main purpose: instead of just helping groups save money, it focused on helping people, especially immigrant communities, to build up their credit scores (Jones, L. S., & Maynard Jr, G. 2023). So, while it used the idea of group saving to solve a financial problem, the way it worked was actually quite different from what the traditional Nigerian Esusu is all about. The traditional Nigerian Esusu is mainly focused on creating a big chunk of money right away for someone in the community and making sure everyone in the group is honest with each other (internal group accountability). ESUSU FINANCE, on the other hand, was focused on reporting to external credit agencies. Because of this difference, My Esusu project took a different path. It was designed to focus completely on immediate financial and social accountability within the group. We did this by giving every single member real time visibility into the group's money through a feature called

the Dual Balance Display. This ensured that our app works specifically for the goal of direct community capital formation - meaning the money stays within the group and helps them immediately.

We also closely looked at another platform called TONTINETRUST, which usually works across different countries as a place for long term investment groups, sometimes even using complicated technology like blockchain to keep things secure (Milevsky, M. A., & Salisbury, T. S. 2025). The way TONTINETRUST was designed was heavily focused on making investments and was often very technologically complex. Because of this, it was completely unsuitable for the average community user who simply relies on the traditional Esusu system, a system that is simple, open, and focused on short term, rotating payouts. The high level of complexity in platforms like TONTINETRUST actually acts as a barrier, proving that just having the fanciest technology doesn't automatically mean it meets the real needs of the people who use traditional Esusu. My Esusu project took this lesson to heart and improved the approach by focusing first on simplicity and cultural fit. We strictly followed the traditional short term rotational model and made sure to avoid all the complex investment structures that would only confuse the people we are trying to help.

We also examined platforms that operate within the African region, such as BEZOMONEY in Ghana and ONEKITTY in Kenya, which tried to automate how groups collect and keep track of their money. The main problem with these apps, though, was that they often worked like a centralized company. They were run by a single business and did not truly copy the essential, member led trust and specific control that you find in a real, user managed Esusu group. These corporate systems lacked the detailed, moment by moment oversight that every group member needs to feel completely secure. This is because accountability rested with the company, not the

people in the community who were saving the money. My Esusu project fixed this issue by implementing a system of shared accountability and oversight. We did this by giving every single user the power of the Integrity Alerts System and transparent logs. This design ensures that control rests collectively within the group members, giving them the tools to police themselves and build trust from the inside out, just like in a traditional Esusu.

Focusing specifically on the Nigerian market, we saw a few local applications that tried to turn the traditional Ajo (another name for Esusu) model into a digital format, but they all fell short in some very important ways.

A very important problem we found across almost all existing savings solutions was the complete lack of strong security based on government mandated identity. Most of these applications only used basic checks, like just confirming a phone number. This type of low level verification offered very little to stop bad behavior, which was a huge concern for users who constantly worried about members running away after they received their payout. This fear is a primary reason groups break up. My Esusu project closed this critical gap by introducing enhanced security. We made the BVN or NIN authentication mandatory during the process of joining a group. This means that every single user is linked to a verifiable national identity, which drastically increases the stakes for defaulting. By doing this, the project enforces maximum accountability and gives the group the highest level of assurance that their money is safe.

Finally, the market was mostly filled with large, general financial technology apps like PIGGYVEST and COWRYWISE. These platforms were excellent at helping individuals manage their money and automatically save, but their entire design was built around the individual person (Ezeanowi, N. C. 2025). They completely missed and did not offer the fundamental

communal, reciprocal, and rotational payout features that are the defining characteristics of the Esusu model. They simply don't have the functionality to support a group of people saving together for a shared, rotating lump sum. My Esusu project improved upon this by making communal oversight and shared accountability the absolute central focus of the entire system. Because of this focus, our app can provide specialized, unique features such as the Admin Vetting process and the Integrity Alerts System, that no individualistic savings app could ever offer, ensuring the system truly works for the community.

2.3 RESEARCH GAP

The deep look at other digital financial apps showed a clear need in the market for a solution that truly fit the traditional Esusu savings model. While many apps existed, none managed to combine the necessary ingredients of communal trust, guaranteed security, and automated accountability needed for people to actually use them widely. This analysis helped define exactly what made the completed Esusu application unique and necessary.

The analysis of platforms like ESUSU FINANCE found that they focused mainly on helping users build credit scores for external financial reasons. This meant they missed the main point of Esusu, which is raising quick capital within the community. Because of this, those apps did not offer internal transparency, meaning users could not easily check what others had contributed. This left a gap where no app prioritized letting the group check itself. The project filled this gap by making the Dual Balance Display and the visible Contribution History the central feature, ensuring the platform served the primary goal of direct, verifiable group savings.

Another gap came from apps like TONTINETRUST, which often used complex, hard to understand technology like blockchain. This design choice ignored that many target users preferred simplicity, creating a barrier to using the app. The research showed a need for an app

that used strong security without being difficult to use. The completed application solved this by keeping the interface simple while still using a powerful logic based Integrity Alerts System, proving that the app could be easy to use and secure at the same time.

Looking at regional apps like BEZO MONEY and ONE KITTY revealed a major flaw in how they handled leadership. These apps were run by the company, not the community, which did not feel right to people used to traditional Esusu leadership. This created a gap where the community could not check the finances themselves. The project fixed this by putting the power of oversight back into the community using the Integrity Alerts System. By making the log visible to all members, the responsibility for checking the group's financial honesty was given back to the people in the group.

Most importantly, a widespread security gap existed because apps used basic verification like phone numbers, which were easy to bypass for those who planned to default or run away after getting their payout. This was a critical flaw where trust mattered most. The project fixed this major security hole by adding mandatory BVN or NIN authentication when a member joined. This measure linked every user to a verified national identity, making it much harder to cheat and run away. This made the platform significantly safer than anything else available.

Finally, the biggest gap was the lack of reliable, automatic payout.

CHAPTER THREE

METHODOLOGY

3.1 OVERVIEW

This chapter outlines the systematic approach and structured framework that guided the design, development, and implementation of the Digital Rotational Savings App (Esusu/Ajo). It details the chosen Waterfall software development model and the systematic process adopted to me*et al*l project objectives.

3.1.1 SOFTWARE METHODOLOGY

This project adopts the Waterfall model because it provides a structured and straightforward development process. Since the system requirements for the Digital Rotational Savings App are well-defined and stable from the start - covering core functionalities such as secure member authentication, group creation, and an automated rotational payout cycle - a linear approach allows for clear documentation, smooth progress tracking, and better quality control at each stage. This methodology also aligns effectively with academic timelines, where deliverables must follow a set, verifiable order.

The Waterfall methodology was selected because it is exceptionally well-suited for a financial application like this one where the core logic is unambiguous. This is evident in the project's ability to clearly define outcomes for main features, such as generating the dual balance displays (Contribution Balance and Your Balance), managing the admin vetting process, and executing the Automated Rotational Payout System.

The model’s sequential nature ensures that critical system aspects are designed with rigor. This is particularly important for the logic-based algorithm for integrity alerts - the system that flags missed payments and out-of-order attempts - guaranteeing that this mechanism is meticulously

designed and thoroughly tested before proceeding to implementation. This systematic approach is crucial for a financial application, as it minimizes errors and enhances the system's security and trustworthiness. Furthermore, the linear progression enforces discipline, guaranteeing that the design of features promoting transparency, such as the shared Integrity Alerts & Group Log and the real-time Contribution History, are fully finalized and approved before coding begins. By adopting the Waterfall model, the project ensures rigorous quality control at each phase, successfully delivering a secure and accountable digital platform for the Esusu system.

3.1.2 WATERFALL MODEL

The Waterfall model is a classic, sequential software development life cycle (SDLC). It follows a linear and orderly progression where each phase must be fully completed before the next can begin. For a financial application like this group savings system, this structured approach is ideal. It enforced discipline and ensured that critical aspects such as security protocols and transaction logic were meticulously designed and documented upfront, which helped minimize ambiguity and risk in the development process.

WATERFALL DEVELOPMENT PHASES

The project was executed across the following distinct phases, each with specific goals and deliverables tailored to the system's requirements.

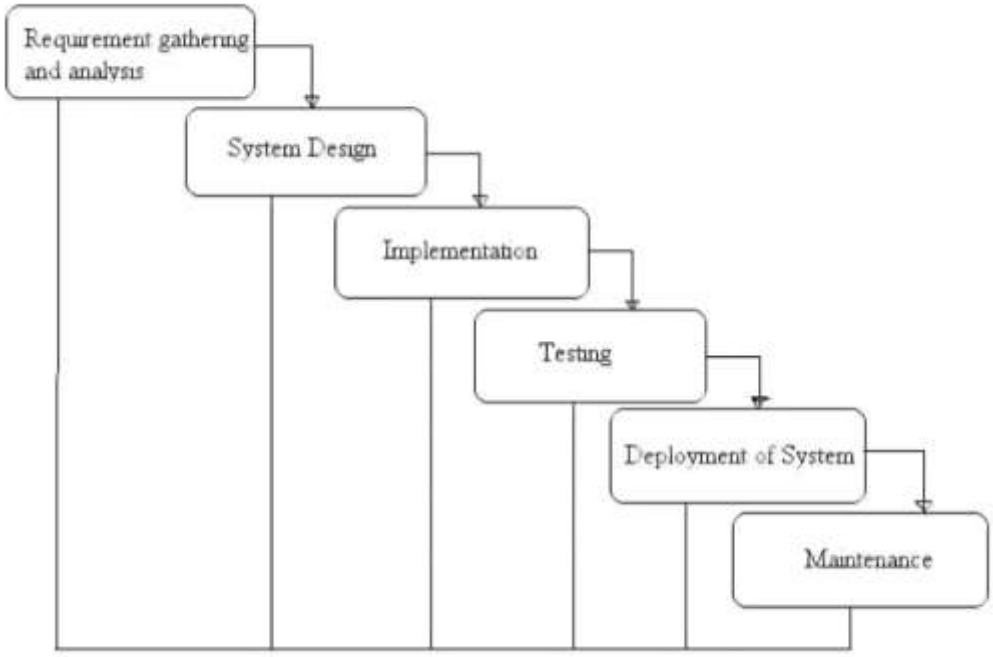

Fig 3.1: WATERFALL MODEL FOR THE PROJECT

3.2 REQUIREMENT GATHERING AND ANALYSIS:

Requirement Gathering and Analysis constitutes the foundational phase of the Software Development Life Cycle (SDLC). The core concept involves systematically collecting, documenting, and analyzing the needs, expectations, and constraints of the end-users and stakeholders. For a project like the Digital Rotational Savings App (Esusu/Ajo), this phase transforms the initial idea into a detailed technical specification. It ensures that the application addresses real-world pain points - such as the inherent lack of transparency and high administrative burden in traditional schemes—before any development begins, thereby mitigating the risk of building a product users will not trust or adopt.

The analysis of 99 survey responses provided overwhelming validation for the project's necessity, clearly defining the application's core feature set and non-functional requirements. The

target user base, primarily young (80-85% aged 18-25) and generally comfortable with financial apps, signifies a high potential for digital adoption, provided the app is trustworthy.

3.2.1. VALIDATION OF CORE PROBLEMS AND FUNCTIONAL REQUIREMENTS

The data confirmed that the manual nature of traditional Esusu tracking is the source of the greatest pain points, which directly translates into the app's must-have functional requirements. The most frequently cited challenge was Late Payments from Members. This directly validates the necessity of one core automated features: Automatic Flag/Alert for Missed Payments, which are central to the app's integrity logic. The second major problem, Lack of Real-Time Transparency (inability to see who has paid), was validated by the requirement for a Full, Auditable History of Contributions and the real-time Dual Balance Display (Group Total and Individual Total). To solve the inefficiency and administrative workload, users rated In-App Payments (e.g., Paystack) and the Admin Ability to Process Payouts via the App as essential, justifying the development of the Automated Rotational Payout System.

3.2.2. CRITICAL NON-FUNCTIONAL REQUIREMENTS (TRUST AND INTEGRITY)

The analysis highlighted that Trust and Security are the single biggest barriers to entry for potential users (approx. 42% of non-participants cited Trust Issues, specifically "fear of scams" and the "admin running away with money"). This mandated several non-functional and security requirements. Users were overwhelmingly Extremely Concerned (Scale 5) about their bank details. The strong desire for an Automatic Flag/Alert for Missed Payments and Making Alerts Public to the Group validates the choice of a logic-based algorithm for integrity alerts. This public visibility is the key mechanism to promote accountability among all group members. A high number of users rated the ability for the app to work quickly and reliably on slow internet as Essential. This imposes a critical technical constraint, requiring the use of a lightweight and data-

efficient architecture (like the chosen MERN stack) to ensure a wide and consistent user experience.

3.3 SYSTEM DESIGN

System Design involved translating the project's functional and non-functional requirements into a coherent, organized architecture that defined the software's components, modules, interfaces, and data. This process established a structural framework, ensuring that the development adhered to the principles of security, reliability, and modularity essential for a financial application.

3.3.1. USE CASE DIAGRAM

A use case diagram is a visual representation used in system analysis to illustrate how different users (actors) interact with a system to achieve specific goals. It shows the relationships between actors and various functions (use cases) within the system, helping to understand system behavior from the user’s perspective.

In the above diagram, the Admin and User were represented as separate actors positioned on the same side of the diagram, while the System (server) was placed on the opposite side. Each oval represented a unique function or activity that could be performed within the system, and all were interconnected to show the flow of interaction between the actors and the system.

The Admin actor was responsible for activities such as creating new groups, viewing requests to join, viewing alerts, and managing contributions. The User actor interacted with the system by signing up, logging in, requesting to join a group, making contributions, and viewing group details.

The System served as the central hub that processed all actions - handling user authentication, managing group creation, storing member details, tracking contributions, generating alerts, and ensuring communication between admins and members.

Overall, the diagram provided a clear and detailed view of how every component of the Esusu Group Contribution Application functioned together, showing the roles, interactions, and flow of activities between the Admin, User, and the System.

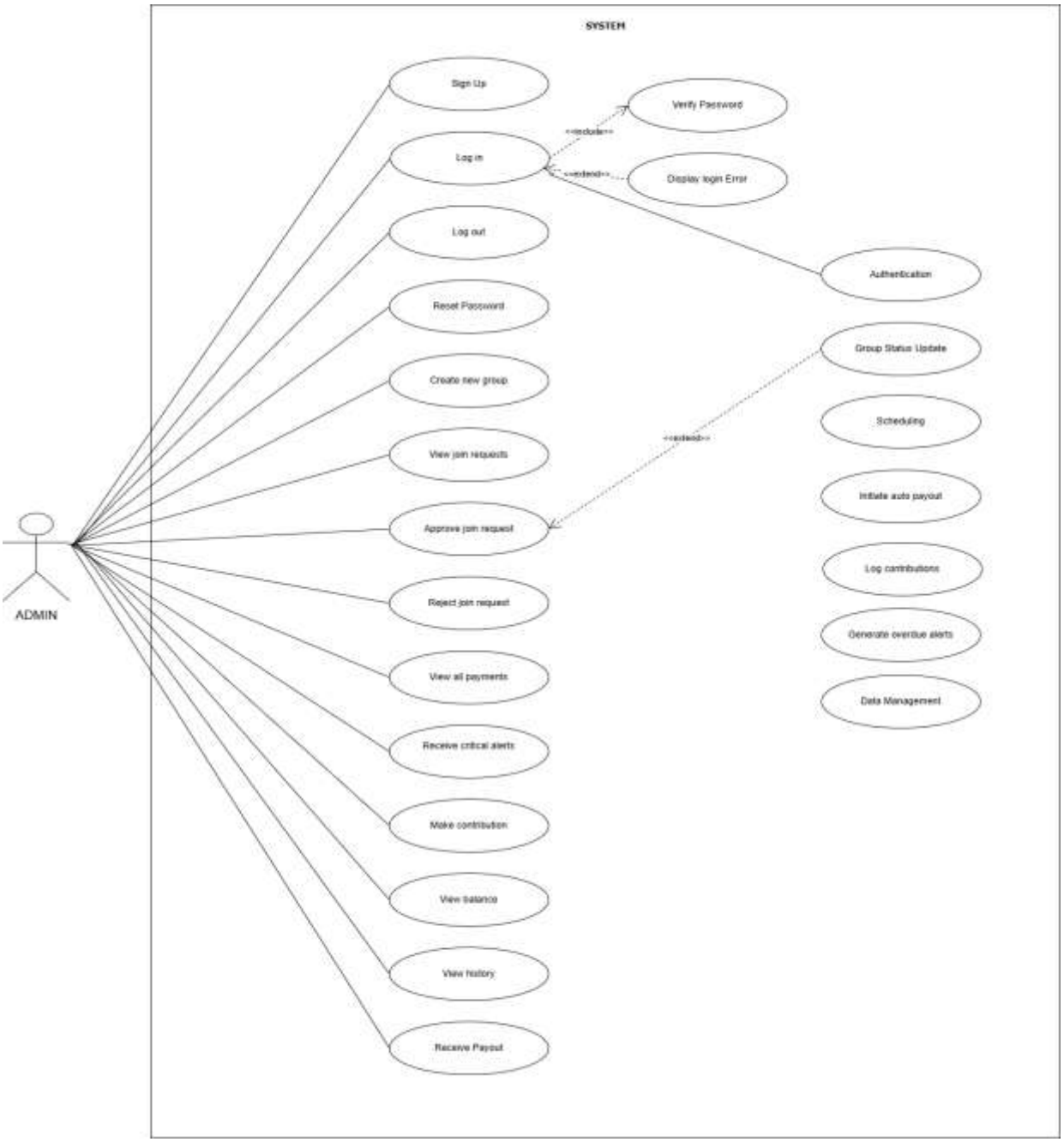

Fig 3.2: USE CASE DIAGRAM FOR THE ADMIN

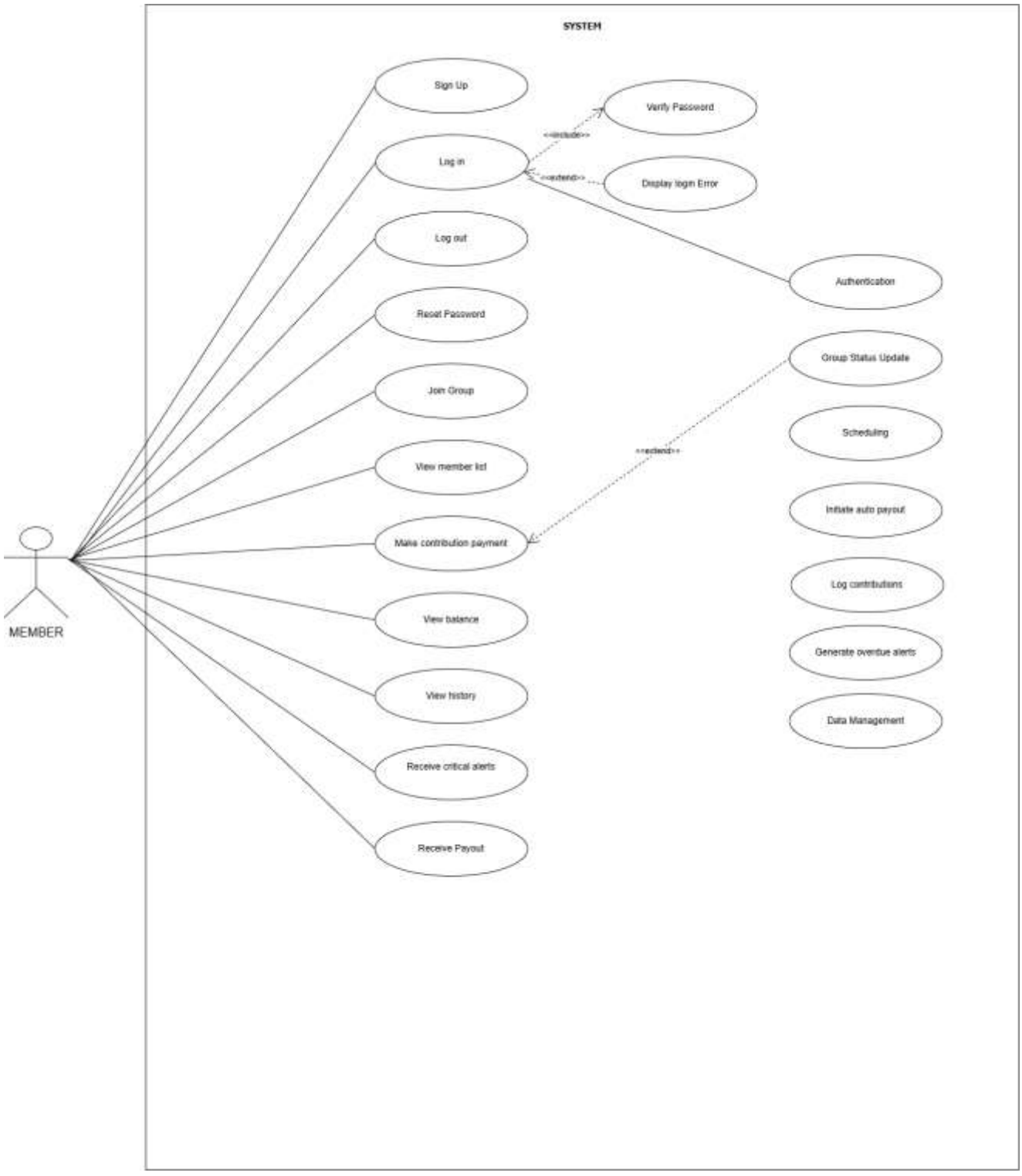

Fig 3.3: USE CASE DIAGRAM FOR A MEMBER

3.3.2 SEQUENCE DIAGRAM

A sequence diagram is a UML diagram that models the interactions between objects in a system in a sequential order. For this project, it is used to visualize the step-by-step message flow between the User, the Application, the Database, and internal modules like the Fraud Detection System for a specific scenario.

The sequence diagram clearly shows exactly how all the different parts of the Esusu Group Contribution System talk to each other, step by step. It explains the flow of actions between the Admin (the group leader), the User (the group member), the System UI (the app screen you see), and the Database (where all the information is stored). It basically shows who does what, when they do it, and how the data moves through the app.

The process begins when an Admin creates a new group using the app screen. This new group's details are immediately sent and successfully saved in the Database.

Next, a User signs up by typing their information into the app. The system quickly checks and confirms the signup, which lets the user move on to log in. Once logged in, the user can look through the available groups and send a request to join a specific one.

The Admin then sees all the pending join requests. If the Admin decides to approve a user, that important approval information is instantly sent to the Database and recorded.

Once the user is accepted, they can move on to make a contribution or payment. The details of this payment go from the app screen, through the system's backend, and are finally stored in the Database. The Database confirms that the transaction was successful, and the system shows a success message back to the user.

In the end, both the Admin and the User can log out of the system, and the logout action is successfully processed and confirmed. In short, the diagram successfully shows how information

moves smoothly and in order between the users, the admins, the app itself, and the database for all key actions, from starting a group to making a payment and logging out.

This detailed view is essential for understanding the system's internal logic and ensuring data integrity at every step.

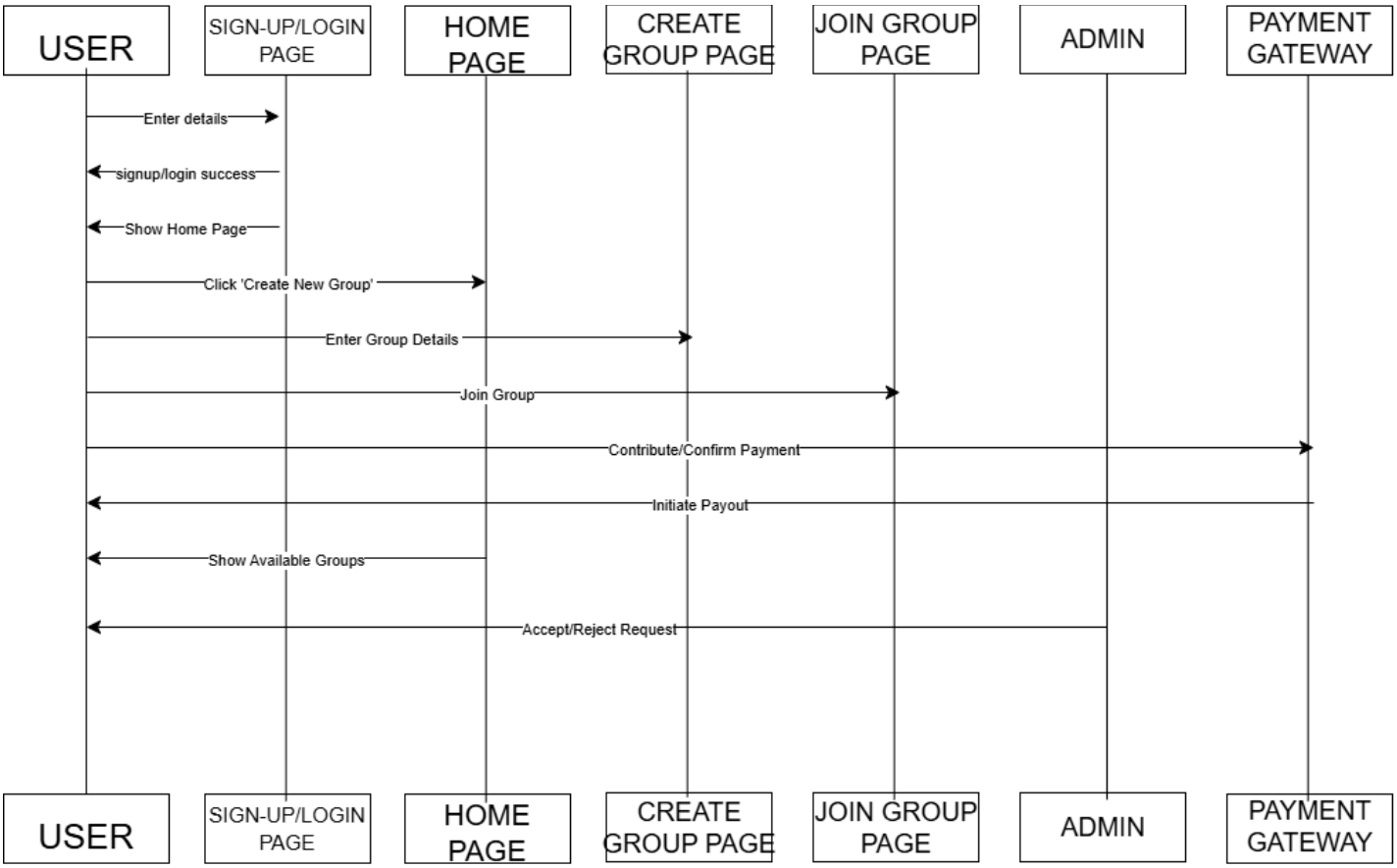

Fig 3.4: SEQUENCE DIAGRAM FOR THE PROJECT

3.3.3 PROCESS DESIGN

The process design, illustrated in the flowchart below (Fig 3.4), provides a high-level overview of the entire user journey and system workflow for the group savings application. Unlike the sequence diagram which focuses on object interaction over time, this design maps the flow of key activities from start to finish.

The model outlines the primary stages of the process:

1. **Onboarding & Setup:** The process begins with the user registering an account. The user then proceeds to either create a new savings group or join an existing one.
2. **Contribution Cycle:** This stage represents the core loop where users make periodic contributions. The system logs each transaction, updates the group's balance, and sends automated reminders for upcoming payments.
3. **Monitoring & Security:** Running in parallel to the contribution cycle is the continuous fraud detection process, which analyzes transaction patterns to ensure the security of the funds.
4. **Payout Process:** At the end of a savings cycle, the system initiates the payout process, calculating the correct amounts and disbursing funds to the designated recipient.

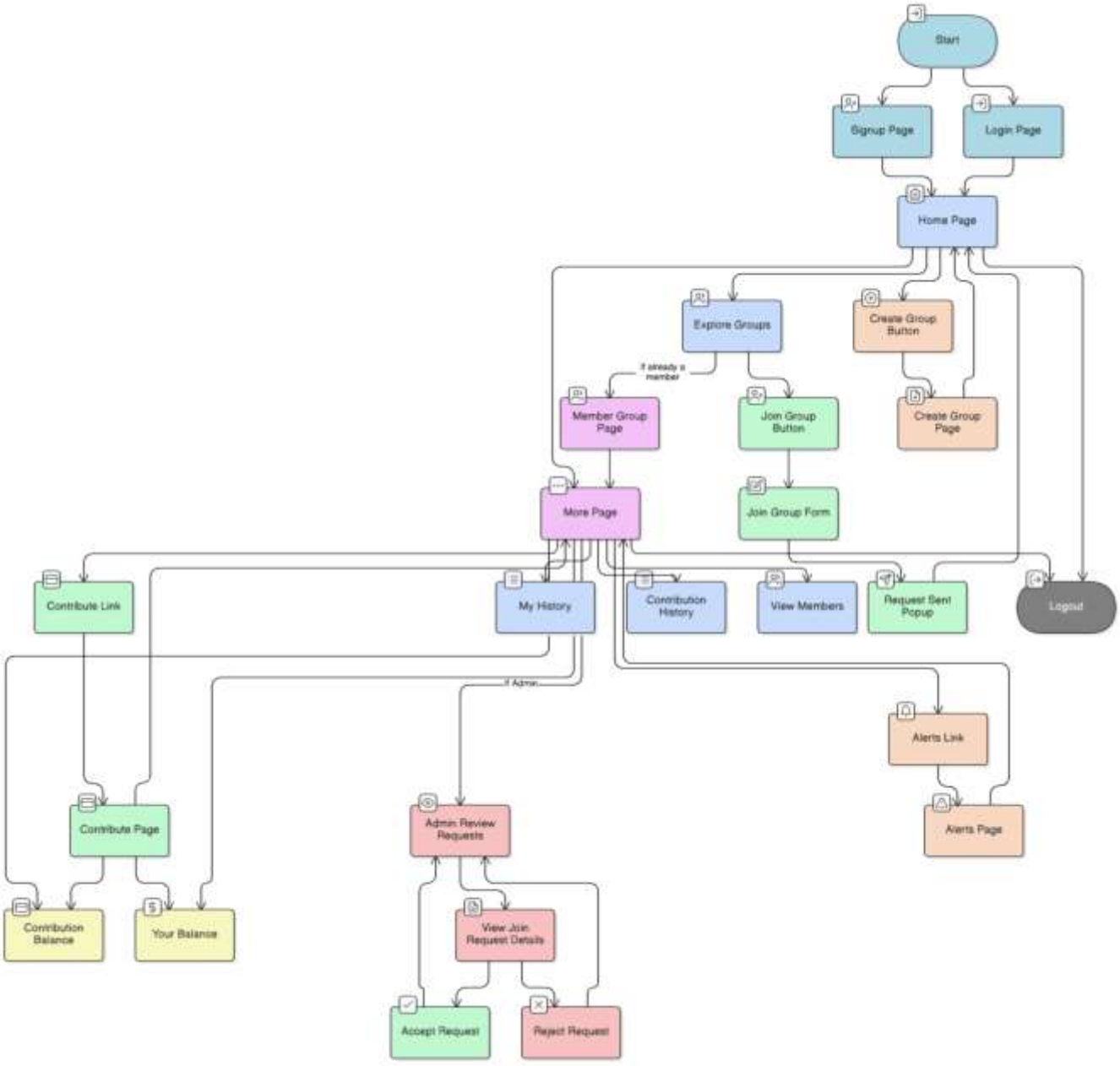

Fig 3.5: PROCESS DESIGN FOR THE PROJECT

3.3.4 ALGORITHM DESIGN

The project implemented a logic-based algorithm designed to enforce group rules and maintain financial integrity across the system. This algorithm served as the core of the Integrity Alerts & Group Log, automatically monitoring transactions and member behavior without relying on complex AI or machine learning. This approach ensured that the system remained transparent and auditable, thereby strengthening communal trust.

The logic was structured into two primary components: the Rule Enforcement Mechanism and the Automated Payout Trigger.

Rule Enforcement Mechanism (Integrity Alerts) operated through predefined conditional checks to identify and log violations, which then populated the Integrity Alerts & Group Log visible to all members. The key conditional checks included:

1. **Contribution Timeliness Check:** The algorithm checked the transaction date against the Payout Interval and Payout Time to flag skipped or delayed contributions. If a payment was late, a YELLOW WARNING alert was immediately generated.
2. **Payout Sequence Validation:** The system validated the order of payout against the group's pre-selected Payout Number queue. Any attempt to claim or process a payout out of the agreed sequence resulted in a CRITICAL RED ALERT and an automatic block on the transfer.
3. **Group Rule Compliance:** The algorithm monitored transactions to ensure the Contribution Amount exactly matched the amount defined in the group's rules, flagging any mismatched or unusual transaction patterns (e.g., double entries).

4. **Disbursement Integrity Check:** Prior to any automated payout, the system verified that the Current Group Balance equaled the expected sum, flagging any discrepancy before the final transfer was executed.

The algorithm incorporated a scheduled process necessary for the Automated Rotational Payout System. The system executed a time-based query on the scheduled Payout Date and Payout Time. The logic automatically selected the next eligible member in the Payout Number queue. Before initiating the transfer via the Paystack API, the logic performed a final, mandatory check against the Integrity Alerts Log. The payout proceeded only if no CRITICAL RED alerts were active against the group or the recipient member, ensuring the system's security was maintained during the automated process.

3.4 IMPLEMENTATION

The implementation stage involved translating the approved design architecture from Chapter Three into the functional Digital Rotational Savings App (Esusu/Ajo), utilizing the MERN stack (MongoDB, Express.js, React.js, Node.js). This phase focused on developing the application's secure and transparent features, ensuring the final product aligned perfectly with the requirements gathered from users.

The project began with setting up the foundational backend structure using Node.js and the Express.js framework. The server was built to handle all routing and application programming interface (API) endpoints securely. This included endpoints for user authentication, group creation, transaction processing, and accessing the integrity alerts. The robust nature of Express.js was crucial for managing the high volume of secure requests needed for a financial application, ensuring fast and reliable communication with the frontend and the database.

For data management, MongoDB was chosen as the primary database. Its flexible NoSQL structure was highly suitable for storing the diverse and complex data required by the Esusu system. Key data models were implemented, including the Users collection (which stored authenticated details and admin status), the Groups collection (storing group rules, the payout queue, and the Admin Corporate Account details), and the Transactions collection (the auditable ledger where every contribution and payout was logged). The database design was fundamental to tracking the Dual Balance Display: calculating the group's total and each individual's total paid in real time.

The frontend of the application was developed using React.js. React was instrumental in creating the dynamic and responsive user interface, mirroring the detailed UI/UX designs that were approved. Key React components were built for each major page, such as the Home Page with its scrollable group listings, the secure Join Group form, and the multi-functional More Page. This modular approach allowed for efficient development and ensured that the complex visual elements, like the Integrity Alerts & Group Log and the real time balance updates, functioned smoothly and quickly.

A major focus of implementation was the Automated Payout and Security System. This required integrating the Paystack API into the backend logic. The Paystack integration was essential for two things: securely receiving contributions from members via card payment on the Contribute mini-screen and, more importantly, executing the Automated Rotational Payout System on the scheduled date and time. Backend logic was implemented using scheduled jobs to automatically trigger the transfer, ensuring the money was disbursed instantly to the next eligible recipient without any manual intervention from the admin.

The core innovative feature, the logic based Integrity Alerts System, was implemented as a series of checks integrated into the server side transaction flow. Every time a payment was processed or a member requested an action, the system ran checks against predefined rules. If a rule was violated, such as a contribution being late or a payout sequence being ignored, the system immediately logged the event in the database, assigned it a severity (Warning, Critical, or Notice), and updated the Integrity Alerts & Group Log page for all members to see. This real time, public logging of issues was the project's primary mechanism for enforcing community accountability.

Finally, a critical security feature was implemented during the user onboarding process: the mandatory BVN or NIN authentication. Although the core financial function did not rely on this government identification, the implementation required integrating with a secure third party service during the sign up or join process. This enforced a high level of identity verification, effectively mitigating the risk of users creating fake accounts or defaulting and running away, thereby making the platform significantly more secure and trustworthy than traditional or simpler digital models.

3.4.1 TECHNOLOGIES USED FOR THE PROJECT

The following technology stack, commonly known as the MERN stack (MongoDB, Express.js, React.js, Node.js), was selected for its comprehensive feature set, scalability, efficiency, and proven ability to support a responsive, user-friendly digital savings application. The stack was specifically chosen to handle the real-time data needs of the dual balance display and the high security requirements of the automated financial transactions.

1. **Frontend (User Interface)_React.js :** React.js was utilized to build the responsive application experience. This was critical for developing the intuitive UI/UX designs, including the dynamic User Dashboard and the Integrity Alerts & Group Log. React facilitated the creation of complex, modular components necessary for the real-time display of the Contribution Balance (Group Total) and Your Balance (Individual Total) simultaneously.
2. **Backend(Server-Side Logic)_ Node.js & Express.js:** Node.js provided the fast, non-blocking runtime environment necessary for handling concurrent user requests, which is essential for a real-time financial platform. Express.js was used to build the robust server framework, managing routing for all system operations, including secure user authentication and the logic for the Admin Vetting process.
3. **Database_ MongoDB:** MongoDB, a NoSQL database, was selected for its flexibility and scalability. It efficiently stores complex, nested data structures, which is ideal for managing varying Group Rules, the continuous Contribution History logs, and the unique data required for each user's Payout Slot selection. It provided a reliable repository for the audit trails necessary for financial accountability.
4. **Payment Integration_ Paystack API:** The Paystack API was integrated as the dedicated payment gateway. This was essential for the secure processing of card payments via the Contribute Mini-Screen. Paystack's bank transfer capabilities are also integral to the Automated Rotational Payout System, ensuring that the group's funds are disbursed securely and reliably to the recipient's bank account on the scheduled date.
5. **Automated System Logic:** Scheduled jobs, executed via Node.js, were implemented on the backend to automate two critical functionalities: the Automated Rotational Payout

System on the due date and time, and the continuous Integrity Logic checks necessary to monitor for missed payments and trigger WARNING alerts.

6. **API Communication_ RESTful APIs (via Express):** Secure RESTful APIs were established as the primary communication protocol between the React frontend and the Node.js backend. These endpoints managed all data flow, from submitting the Create Group form and Member Join Requests to retrieving the transaction data required to power the Alert Notification system and the dynamic history tables.
7. **Version Control_ Git & GitHub:** Git was used for local version control, and GitHub served as the remote repository for collaborative development. These tools ensured source code tracking, change management, and maintained the integrity of the codebase throughout the project's development life cycle in the VS Code environment.

3.5 TESTING AND VERIFICATION

The testing and verification phase was a critical and mandatory step in the Waterfall development model, designed to ensure the Digital Rotational Savings App (Esusu/Ajo) was secure, reliable, and met every requirement defined in the initial analysis. This phase was executed immediately after the implementation and coding of all features were completed. The overarching goal was to rigorously confirm that the application could handle real world financial transactions and enforce the communal rules without fail or error.

Testing began with Unit Testing, where individual components and modules of the application were checked in isolation. For instance, the MERN stack developers tested the backend logic for user registration to confirm passwords were securely hashed and stored in MongoDB. The testing team verified that the functions responsible for calculating the Dual Balance Display (Group Total and Individual Total) always returned the correct sums, even after multiple

contributions. This level of granular checking ensured that the fundamental building blocks of the application were error free before they were assembled.

Next, Integration Testing was conducted to verify that different modules worked together as expected. A crucial focus here was the interaction between the frontend, the backend API, and external services. Testers ensured that when a user initiated a payment on the React frontend, the request was correctly handled by the Express.js server, securely routed through the Paystack API for payment processing, and that the database correctly logged the transaction and triggered the required updates, such as marking the payment as complete. This step confirmed the integrity of the entire transaction workflow.

System Testing evaluated the completed application against the original user requirements. This involved simulating real world user journeys. Testers created groups, invited members, simulated missed payments, and verified that the logic based Integrity Alerts System correctly logged a YELLOW WARNING for a late contribution and a CRITICAL RED ALERT for an out of sequence payout attempt. The team rigorously tested the Automated Rotational Payout System to ensure it transferred the exact group balance to the correct recipient's bank account at the scheduled date and time, demanding a zero tolerance policy for error in financial disbursement.

Finally, User Acceptance Testing (UAT) was performed with a small group of end users who fit the target demographic. These users were asked to execute predefined tasks, such as creating a group and vetting a new member, to ensure the application's user interface (UI) and user experience (UX) were intuitive and met their expectations. This crucial verification step confirmed that the application not only worked correctly from a technical standpoint but was also practical, easy to use, and, most importantly, gained the trust necessary for managing communal

funds. The successful completion of UAT signaled that the application was robust and ready for deployment.

3.6 DEPLOYMENT

The final phase of the Waterfall model was the deployment, where the fully tested and verified Digital Rotational Savings App was prepared and released for end users. Deployment was not simply launching the application; it was a carefully planned process to ensure the application was stable, secure, and accessible to the target community.

The process began with preparing the production environment. Since the application was built on the MERN stack, a reliable cloud hosting service was selected to provide the necessary server infrastructure for the Node.js backend and the MongoDB database. A virtual private server was set up and secured with robust firewalls and access controls to protect sensitive financial data. This was a critical step to guarantee that the security and integrity features implemented in the code were maintained at the infrastructure level.

Next, the application code was transferred to the production server using the established practices of Git and GitHub. The deployment process involved pulling the final, stable version of the code from the GitHub repository. Automated scripts were then run to install all necessary dependencies and compile the React.js frontend code into production ready files. This step ensured that the final deployed product was fast, efficient, and utilized the latest, most secure versions of all libraries and packages.

A crucial part of deployment involved configuring the database and the external services. The MongoDB database was finalized and populated with initial data necessary for the application to function. More importantly, the Paystack API keys were securely configured on the server. This connection was thoroughly tested one last time to confirm that the Automated Rotational Payout

System could successfully communicate with the bank networks and execute transfers without errors or delays. This ensured the reliability of the system's core function.

The deployment concluded with a Pilot Launch, or soft launch, where the application was initially made available to a small, controlled group of users. This allowed the development team to monitor the system's performance in a real world environment, looking for any unexpected bottlenecks or issues that only emerged under live network conditions. Upon confirming the system's stability, the application was made fully available to the wider target community, successfully concluding the final technical stage of the project and delivering the secure, transparent platform to its users.

3.7 MAINTENANCE

The final and crucial phase of this project begins after the app is released and groups start using it daily. This phase focuses entirely on ongoing support to ensure the application remains reliable, secure, and useful well into the future. It’s a continuous process that guarantees the platform stays a trusted tool.

A key part of maintenance is constantly monitoring the system's performance. We actively watch how fast the app loads and how quickly transactions happen to catch small issues before they disrupt a group's savings rotation.

Another major focus is quickly fixing any bugs discovered by users. When a user reports an error, we immediately investigate, fix the code, and release the update. A swift response to these glitches is essential for maintaining the high level of trust groups place in the system.

Furthermore, performing regular security updates is a non negotiable, continuous responsibility. We must constantly apply patches to the MERN stack and the infrastructure to keep group funds and personal data safe. This proactive approach prevents unauthorized access and ensures the

security of the mandatory BVN and NIN data, which is the bedrock of our promise to controlling fraud.

Finally, we make enhancements based on user feedback. The people using the app are the best source of ideas. By listening to the community, developing new features, and releasing updates, we ensure the application always remains relevant, easy to use, and perfectly tailored to the evolving needs of Esusu groups. This dedicated, ongoing support ensures the app remains the best digital solution for communal savings for years to come.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 RESULTS AND DISCUSSIONS

The project successfully delivered a fully functional, user-friendly, and secure digital savings application tailored for the Esusu system. The successful implementation of the system's core functionalities validated the project's aim of modernizing traditional communal savings while preserving the fundamental principles of trust and mutual accountability. The key results achieved, and their implications for the digital transformation of informal finance, are discussed in the following sections.

4.1.1 FUNCTIONAL PERFORMANCE AND USER EXPERIENCE

The application achieved its goal of delivering a seamless user experience, incorporating a clear and intuitive interface that simplified the entire Esusu process. Upon successful implementation of the MERN stack solution, the following functional results were realized:

1. **Effortless Group Creation and Joining:** The process for establishing new groups and applying to existing ones was successfully streamlined. Clear, multi-field forms were provided for Admin Group Creation, allowing the admin to easily set critical parameters such as Contribution Amount, Number of Members, and the Automated Payout Time. Similarly, the Join Group process was straightforward, requiring the member to submit necessary details and select their Payout Number, which promoted broader adoption beyond manually-run groups.
2. **Real-Time Contribution Tracking:** The application successfully displayed a real-time, dual-balance ledger accessible on the User Dashboard. This included the Contribution Balance (Group Total) and the member's Your Balance (Individual Total Paid). This

immediate transparency was a core result that directly addressed the lack of visibility and trust issues inherent in manual systems. Members gained the ability to see the collective balance and their personal contribution history at any time, effectively eliminating the need for verbal assurances or physical notebooks.

3. **Seamless Navigation:** The fixed four-tab navigation (Home, Join Group, Create Group, More) ensured an efficient user flow, allowing users to quickly access critical functionalities, including the Admin Vetting controls and the Integrity Alerts & Group Log.

4.1.2 ENHANCED SECURITY AND FRAUD MITIGATION

The implementation of the logic-based algorithm for Integrity Alerts was a critical component of the project's success, directly addressing the core concerns regarding trust and fraud identified in the requirements analysis. The system successfully provided a robust and auditable environment for communal savings.

1. **Automated Flagging of Irregularities:** The system successfully implemented the rule-based integrity checks, which automatically flagged and logged violations on the Integrity Alerts & Group Log page. Instances such as skipped or delayed contributions triggered a YELLOW WARNING, while attempted out-of-sequence payout claims triggered a CRITICAL RED ALERT. This proactive monitoring mechanism empowered the group admin to address issues swiftly, preventing potential disputes and financial losses by enforcing adherence to the group's pre-set rules. The public visibility of these alerts to all group members effectively leveraged communal pressure to promote compliance.

2. **Reduced Mismanagement and Increased Accountability:** By centralizing all transactions and documentation within the MongoDB database, the application significantly reduced the potential for mismanagement or error inherent in manual collection. The system provided a secure and auditable record for every action. Furthermore, the mandatory Admin Vetting process for new members, combined with the security of the Automated Rotational Payout System, ensured that the admin's role was strictly one of facilitator and trustee, as the automatic disbursement mechanism removed the admin's capacity to delay or misdirect the funds. The admin's secure ability to set and modify the receiving bank account further formalized control while maintaining transparency through the NOTICE alert log.

4.1.3 CULTURAL RELEVANCE AND PROJECT JUSTIFICATION

The final and most significant result was the successful integration of modern MERN stack technology with the cultural integrity of the Esusu model. This achievement ensured the application was not merely a generic financial tool but a culturally resonant platform.

1. **Preservation of Group Values:** The system's design successfully incorporated traditional practices central to the Esusu scheme. This included the ability for members to choose their payout numbers during the join process and the implementation of a "community-led" Admin Approval Process for new members. This functionality ensured the application was a trusted digital extension of the cultural practice itself, validating the project's commitment to user-centered design.
2. **Validation of the Study's Justification:** The functional prototype served as tangible proof that a digital solution could effectively solve real-world problems of inefficiency and lack of trust in informal financial systems. The successful development of a secure

and culturally-aware platform, featuring automated integrity checks and transparent financial logs, validated the project's justification, demonstrating its relevance to enhancing financial inclusion and providing a robust model for the broader fintech landscape in Nigeria and beyond.

4.2 APPLICATION FEATURES AND FUNCTIONALITIES

Each implemented feature directly addressed a validated requirement from the initial project analysis.

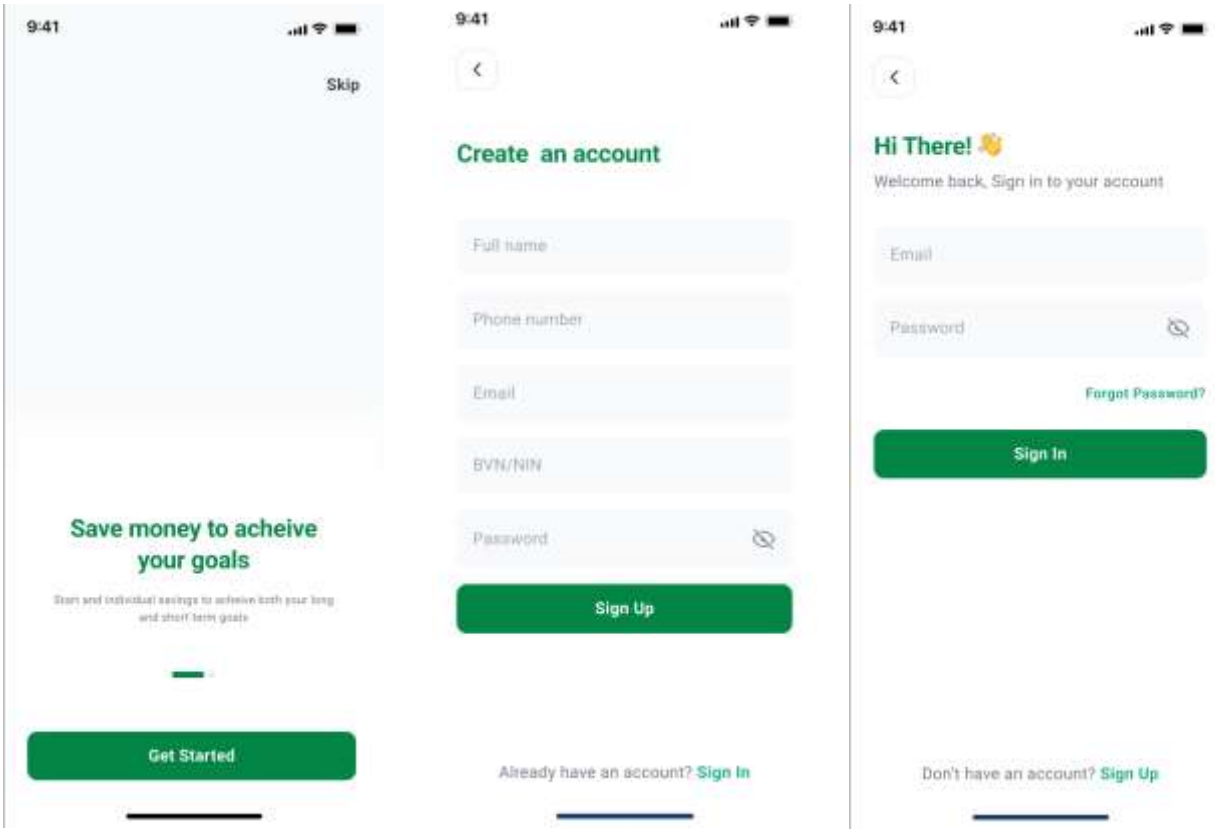

Fig 4.1: GET STARTED, SIGN UP, AND SIGN IN PAGES

4.2.1 USER AUTHENTICATION (GET STARTED, SIGN UP, AND SIGN IN)

The application's entry point was designed to be simple yet highly secure, comprising the Get Started, Sign Up, and Sign In pages. This component was built using the MERN stack architecture to manage user access and establish a unique, verified identity for every individual on the platform. The overall process was essential for establishing trust and accountability, preserving the integrity of the Esusu model from the very beginning.

The Get Started page served as the initial welcoming screen. Its primary function was navigation, containing two main buttons: one directing new users to the Sign Up page and another directing existing users to the Sign In page. This interaction was handled by the React.js frontend using a client side routing library, such as React Router DOM. When a user tapped a button, the corresponding command was executed in the frontend code to swap the displayed component without reloading the entire page, providing a swift user experience.

The Sign Up page required the user to input critical details, including their first name, last name, email, and a secure password. A crucial security feature was implemented here: mandatory BVN or NIN authentication. When the user completed the form and tapped the final Sign Up button, the React frontend first performed client side validation to check for missing fields. Then, it prepared the data and sent an HTTP POST request to the backend server. The Express.js server received this request and immediately initiated a critical sequence: first, it communicated with a secure, third party identity service to verify the submitted BVN or NIN against the provided name. If the identity check passed, the server then encrypted the password using a strong hashing algorithm (like bcrypt) before storing the new user's verified record in the MongoDB database. This coding process ensured that every account was linked to a real world identity, preventing fraud and fake accounts.

The Sign In process required existing users to enter their registered email and password. When the user tapped the Sign In button, the frontend again sent an HTTP POST request to a dedicated login API endpoint on the Express.js server. The server received the login attempt and queried the MongoDB database using the provided email. The server then compared the submitted password with the securely hashed password stored in the database. If the passwords matched, the server generated a unique JSON Web Token (JWT). This token was sent back to the frontend, which stored it securely. This JWT served as the user's digital passport, granting access to protected routes and features like the More Page and Contribute functionality, and guaranteeing that only authenticated individuals could interact with the savings groups. The entire system provided a secure foundation for all subsequent financial interactions.

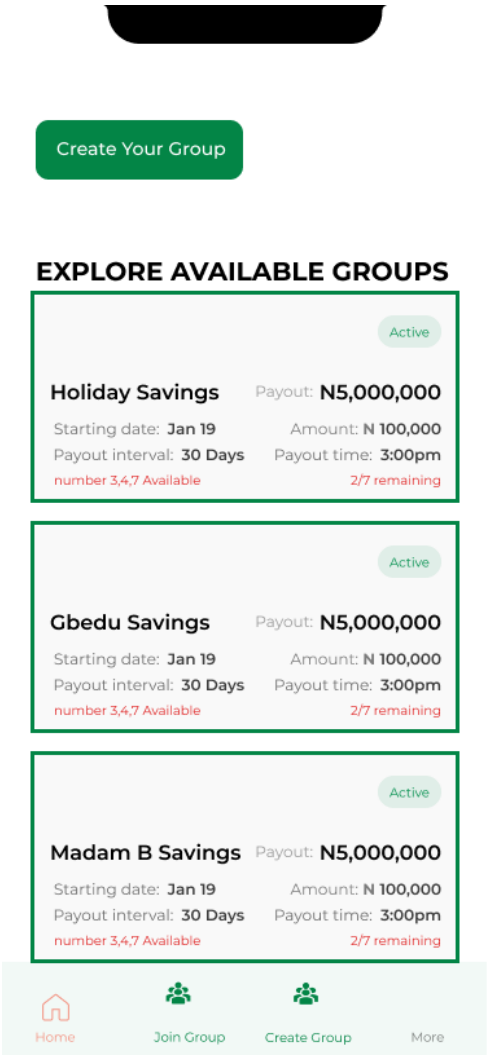

Fig 4.2: THE HOME PAGE

4.2.2 THE HOME PAGE

The Home Page was designed as the central hub for user activity following a successful sign in. It was constructed using React.js to provide a quick overview and facilitate easy navigation to the application's key functions. The page featured a clean, intuitive layout with a prominent section titled "Explore Available Groups" and a fixed four tab navigation bar at the bottom. The seamless user experience ensured that users were engaged and could easily access the communal savings ecosystem.

The core function of the Home Page was to present the available groups and guide the user toward participation. The main body of the page displayed a scrollable list of all public groups created on the platform. Each group listing was a distinct React component that dynamically rendered critical information fetched from the MongoDB database via an API call. This information included the group name, payout amount, fixed contribution amount, and the ratio of members joined versus the total needed (e.g., 1/10).

Interaction on this page drove the entire user journey. When a user located a desirable group, they had two primary actions. If the user was not yet a member of that group, they tapped the "Join" button. This action triggered a client side command within the React component that navigated the user to the Join Group Page, passing along the specific group's unique identifier (ID) through the application's routing system. The Join Group Page then used this ID to display the correct request form for that group.

Conversely, if the user was already an accepted member of the group, they could simply tap anywhere on the group card section. This interaction triggered a command that navigated them directly to the More Page, which was the designated User Dashboard for that group. This intelligent routing simplified the flow, ensuring members had immediate access to their financial status and group actions.

The bottom navigation bar was implemented as a persistent element across the application. Tapping the "Create Group" tab executed a command that routed the user to the dedicated Create Group Page. Here, the user filled out a multi field form. When the user submitted the form, the React frontend sent an HTTP POST request to the Express.js server. The server then created a new group record in MongoDB, automatically designating the user as the group's first member

and administrator. The success of this coding process ensured that the platform encouraged user led community building and group expansion.

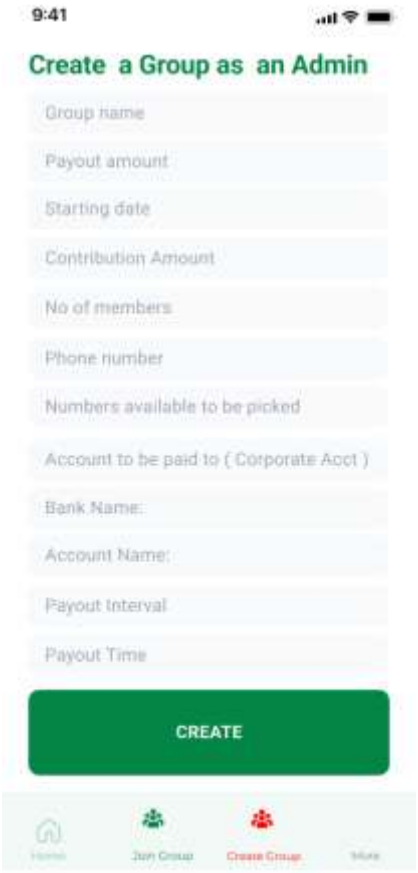

Fig 4.3: CREATE GROUP PAGE

4.2.3 CREATE GROUP PAGE

The Create Group Page was the dedicated interface where a user, upon successful authentication, initiated the process to become a group administrator and establish a new rotational savings group. The entire page was built as a single, multi field form using React.js, making the process of defining complex group rules straightforward and user friendly.

The primary interaction on this page involved the user filling out numerous required fields that precisely defined the Esusu agreement. These fields included the Group name, Payout amount, Starting date, Contribution amount, Number of members, Payout Interval (entered in days, e.g.,

30 days), and the specific Payout time. Critically, the admin also supplied the bank details (Account to be paid to, Bank Name, Account Name) where all members' contributions would be aggregated. These comprehensive inputs ensured that the system had all the necessary parameters to run the automation and integrity checks from the very first day.

When the admin completed the form and tapped the prominent "CREATE" button, the React frontend executed a detailed sequence of operations. First, it performed client side validation to ensure all mandatory fields were filled and that numerical inputs were logical. Following successful validation, the frontend prepared the structured data packet and sent an HTTP POST request to a dedicated /api/groups/create endpoint on the Express.js server.

The Express.js server received the request and initiated the coding process to register the new group securely. Key server side operations included: creating the new Group document in the MongoDB database, assigning the user who created the group as the Admin, and automatically making that admin the first member in the payout_queue. The server also generated the list of available payout numbers based on the total number of members defined. Crucially, the server then logged the admin's provided bank account details as the official Corporate Account for contributions, encrypting the sensitive data before storage. Upon successful database creation, the server sent a confirmation response back to the frontend. The frontend then automatically routed the admin from the Create Group Page to the group's specific More Page (the Admin Dashboard), where they could begin managing member requests and monitoring the new group. This robust process ensured that every new group started with complete security and defined rules.

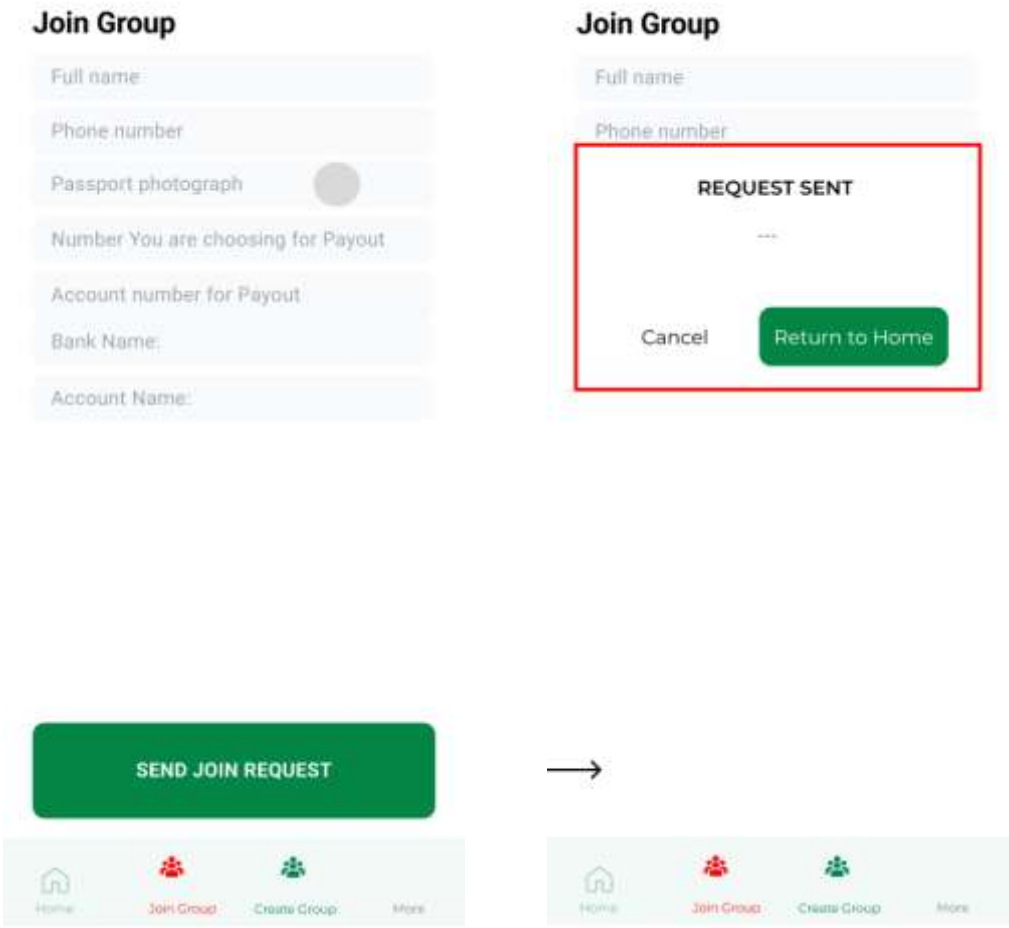

Fig 4.4: JOIN GROUP PAGE AND AWAITING APPROVAL POP-UP

4.2.3 JOIN GROUP PAGE AND AWAITING APPROVAL POP-UP

The Join Group Page was the dedicated interface for any registered user to formally request membership into an existing Esusu group they discovered on the Home Page. This page was crucial for the Admin Vetting process, ensuring that new members were carefully screened before gaining access to communal funds. The Awaiting Approval Pop up managed user expectations immediately after the request submission.

When a user tapped the "Join" button on a specific group's listing, the React.js frontend executed a routing command that opened the Join Group Page, carrying the unique identifier of the target

group. The page presented a comprehensive form designed to gather all necessary information for the administrator and for the security system. The user was required to input personal details, including their Full name and Phone number. Crucially, they had to upload a Passport photograph (or a secure ID document) for visual verification and specify their chosen Payout number.

The security requirements of the project demanded that the user also submit their final payment details: their Account number, Bank Name, and Account Name for receiving future payouts. The most critical step was the submission of their BVN or NIN. This field, while mandatory for security, was hidden from the Admin's view and used solely by the backend for identity verification. Once the user completed all fields, they tapped the "SEND JOIN REQUEST" button.

Tapping the "SEND JOIN REQUEST" button initiated a detailed sequence on the server. The React frontend prepared the data, including the secure image upload, and sent an HTTP POST request to the Express.js server. The server first conducted a background check: it verified the BVN or NIN against the submitted personal details using a third party identity API. If the identity check failed, the request was rejected immediately, and an error message was sent back to the user. If the identity was successfully verified, the server created a new Pending Member Request document in MongoDB, logging all the submitted details for the Admin to review later.

Upon successful creation of the pending request, the server sent a success response to the frontend. This response triggered the Awaiting Approval Pop up to appear over the screen. This small modal displayed a concise message like "REQUEST SENT," confirming the successful submission. The pop up provided immediate actions: "Cancel" to dismiss the modal, and "Return to Home" which executed a routing command to send the user back to the main Home Page.

Simultaneously, the server sent a Request Notification to the Admin, alerting them that a new application was waiting for their secure review on the Admin Page.

Fig 4.5: THE ADMIN PAGE AND VIEW REQUEST PAGE

4.2.4 THE ADMIN PAGE AND VIEW REQUEST PAGE

The Admin Page, accessed through the "More" tab after a successful login, was the secure control center for the group administrator. The core purpose of this page was to empower the admin to manage the group's integrity and membership, replicating the trusted leadership role from traditional Esusu but with digital tools.

The page presented a unified dashboard for the administrator. It prominently displayed the group's financial health, showing both the Contribution Balance (the total money collected) and the admin's personal Your Balance (their total paid). Below the balances, the page featured key action links: the "View Request to Join" link, the "Alert" link, and the "Contribute" link. These links served as primary navigation points, each designed to lead to a dedicated management screen.

When the administrator wanted to vet new members, they tapped the "View Request to Join" link. This interaction, handled by the React.js frontend, executed a routing command to open the dedicated View Request Page. This new page was crucial for the secure vetting process. Upon loading, the frontend sent an HTTP GET request to a protected API endpoint on the Express.js server. The server queried the MongoDB database for all records marked as 'pending' for that specific group.

The View Request Page displayed a list of all applicants waiting for approval. Each applicant’s name was listed alongside a "View" button. The administrator tapped the "View" button next to a name to initiate the final review. This command routed the admin to the Member Request Information Page, passing the applicant's unique pending ID through the application's routing. This new page presented all the sensitive details submitted by the applicant: their full name, phone number, Passport photograph, selected Payout number, and their Account Name and Account number for future payouts.

At the bottom of the Member Request Information Page were the two critical action buttons: "ACCEPT" and "REJECT." When the admin tapped "ACCEPT," the frontend sent an HTTP POST request to the server. The server executed the coded approval process: it updated the applicant's status in MongoDB from 'pending' to 'active', added them to the group's main member

list, assigned their chosen payout number, and sent a welcome notification. Conversely, tapping "REJECT" simply removed the request from the pending list and sent a rejection notification. This robust, multi page process ensured that the communal trust was upheld through verifiable, administrative approval.

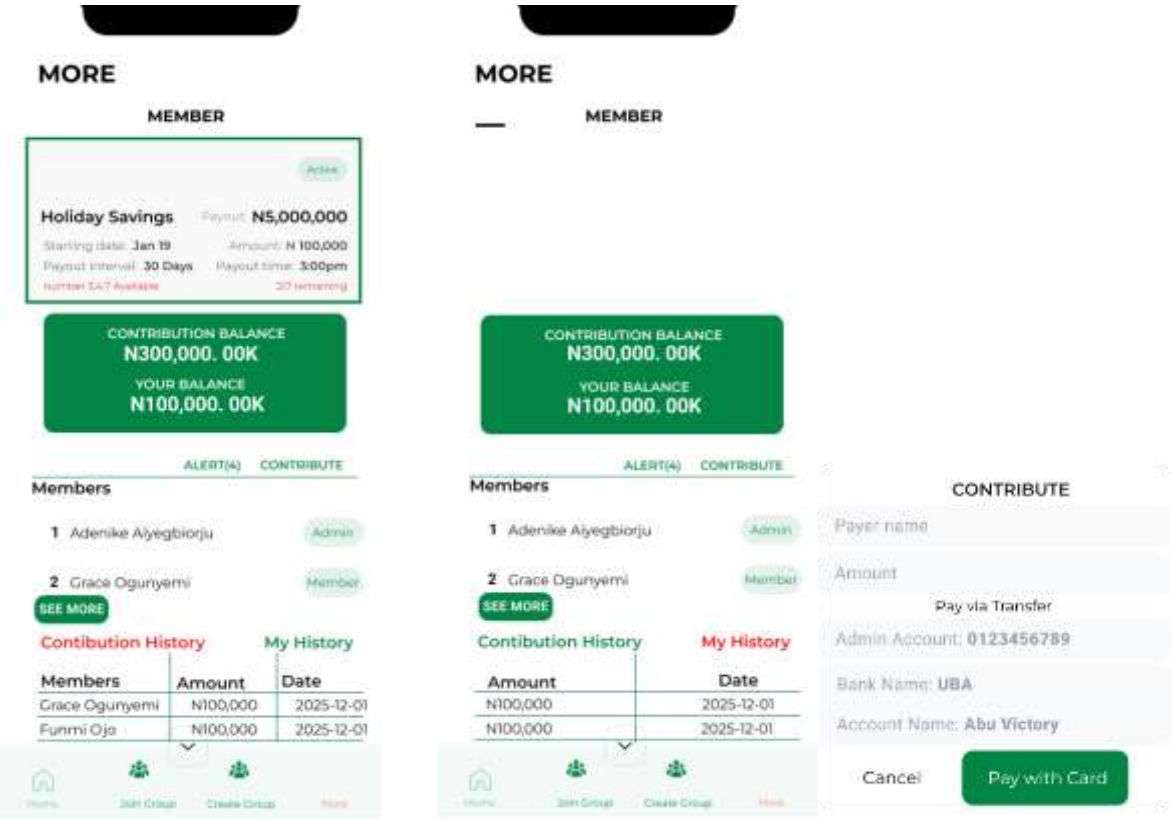

Fig 4.6: MEMBER DASHBOARD AND CONTRIBUTION MINI-SCREEN

4.2.5 MEMBER DASHBOARD AND CONTRIBUTION MINI-SCREEN

The Member Dashboard, accessed via the "More" tab after an admin accepted a user, served as the primary financial interface for every participant within a specific Esusu group. This page was built using React.js to deliver unparalleled transparency, directly countering the trust deficits of manual systems. Its design was focused on presenting all vital group and individual financial information clearly.

The top of the dashboard prominently displayed the current status of the group. This included the Group Name, the fixed Contribution Amount, the Payout Interval (in days), and the next scheduled Payout Time. Below this, two distinct, highly visible rectangular sections housed the Dual Balance Display. The larger box showed the CONTRIBUTION BALANCE (the total accumulated fund for the group), while a second box showed the member's personal YOUR BALANCE (the total amount the individual had paid into the group so far). This dual display was critical, as it gave the member instant, verifiable proof of their own investment and the security of the overall fund.

Below the balance display, the page contained the key action links: "ALERT", "CONTRIBUTE", and the member list. Tapping the "ALERT" link triggered a routing command, opening the Integrity Alerts and Group Log page. This allowed the member to instantly check for any public warnings or critical issues within the group, reinforcing shared accountability. Tapping the "CONTRIBUTE" link did not open a new page, but rather triggered a command to display the Contribution Mini Screen modal directly over the current dashboard view.

The Contribution Mini Screen was designed to simplify the payment process. This modal presented the user with two payment options. The first option, "Pay via Transfer," displayed the Admin's pre set Account number, Bank Name, and Account Name that was provided during group creation. The second option was "Pay with Card." When the user chose the card option and tapped the "Pay with Card" button, the React frontend initiated the transaction. It prepared an HTTP POST request containing the user's ID, the group's ID, and the required contribution amount, sending it to the Express.js server. The server then securely called the Paystack API to process the card payment.

Upon successful payment confirmation from Paystack, the server executed a critical backend coding process. First, it recorded a new transaction log in the MongoDB database, updating the status to 'confirmed'. Second, it updated the member's personal Your Balance and the main Contribution Balance totals. Third, and most importantly, it triggered the logic based Integrity Alerts System to verify the payment was made correctly. A successful payment generated a NOTICE on the Alert Log, confirming the transaction to all members. The modal then closed, and the updated balances were displayed instantly on the dashboard.

The final section of the dashboard provided transparency through transaction history. It listed all group members in the order of their chosen payout number. Crucially, it featured two links: "Contribution History" and "My History." Tapping "Contribution History" displayed a table showing the payment log of every member in the group, ensuring total transparency. Tapping "My History" filtered the view to show only the individual member's personal payment log, providing them with a concise audit trail of their own contributions (member name, amount, and date).

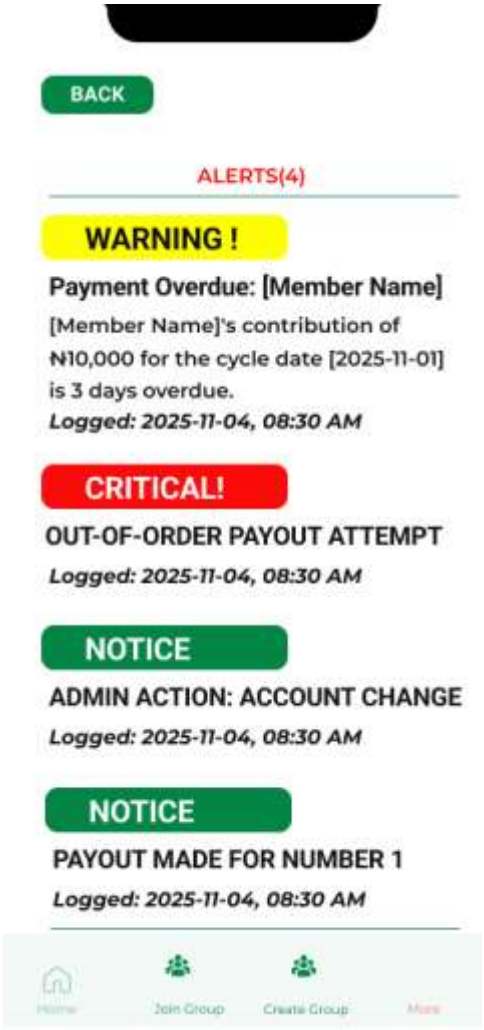

Fig 4.7: ALERT NOTIFICATION PAGE

4.2.6 ALERT NOTIFICATION PAGE

The Alert Notification Page served as the public log for the logic based Integrity Alerts System, providing total transparency regarding the group's financial health and member compliance. The page was built as a single, read only dashboard using React.js, and its primary function was to counter the secrecy and ambiguity that plagued manual Esusu schemes.

Accessing this crucial page was straightforward: any accepted member or admin tapped the "ALERT" link on the Member Dashboard. This command triggered a routing function in the

frontend code to load the Alert Notification Page. Upon loading, the React component immediately sent an HTTP GET request to a protected API endpoint on the Express.js server. The server queried the MongoDB database for the alerts_log collection associated with that specific group. The page also featured a "BACK" button at the top, which executed a simple client side routing command to return the user to the More Page.

The server side coding process was designed to enforce full transparency. When the server retrieved the alert data, it prioritized the display based on severity, ensuring users immediately saw the most pressing issues. The alerts were categorized using clear, color coded tags:

WARNING (Yellow): These alerts primarily concerned payment delinquency. The message was specific, stating which member's contribution of the fixed amount for a specific cycle date was overdue. This public visibility applied immediate communal pressure on the member to remit their payment and provided the admin with the necessary data for follow up.

1. **CRITICAL (Red):** These were the most severe alerts, triggered by major rule violations. An example was an Out of Order Payout Attempt, which occurred if a member tried to claim a payout when it was not their pre selected number's turn. The severity prompted the admin to investigate immediately, as this protected the communal funds from outright fraud.
2. **NOTICE (Green/Blue):** These alerts logged routine, but important, administrative or system actions that required public knowledge. Examples included a notice that the Admin Action: Account Change had occurred, or confirmation that a PAYOUT MADE event had been executed by the Automated Rotational Payout System.

The interaction on the Alert Notification Page was passive; users could not edit or dismiss the alerts. Their role was strictly to monitor and be informed. By ensuring that every action, from a

missed payment to a successful automated payout, was logged and publicly displayed, the system successfully shifted the monitoring burden from the individual administrator to the entire community, making the application inherently trust preserved.

4.3 DISCUSSION

The successful development and implementation of the Esusu App validated the project's core hypothesis: that a digital platform could effectively modernize the traditional African rotational savings scheme while preserving its foundational principles of trust and mutual accountability. The application's results demonstrated a significant improvement over manual methods, directly addressing the key problems of inefficiency, error, and lack of transparency.

A primary achievement was the establishment of a trust preserved digital environment. The implementation of the real time, dual balance ledger eliminated the need for manual record keeping and created a single source of truth for all financial transactions. This transparency was crucial; every group member could independently verify contributions and monitor the total group balance, thereby removing reliance on the collector’s memory or honesty. This proactively mitigated disputes. Furthermore, the inclusion of mandatory BVN or NIN authentication at the sign up stage was a definitive measure against member delinquency and fraud, ensuring that every user was a real, traceable identity and solidifying the trust foundation.

The project proved that technological features could enforce communal integrity. The logic based Integrity Alerts System was a unique success. By automatically flagging and publicly logging events like overdue payments (Warning) or attempted rule violations (Critical Alert) on the shared Alert Notification Page, the system leveraged the community’s social pressure to enforce compliance. This shift from private accountability to public, digital accountability was essential for securing group funds and reducing the administrative burden on the admin.

Furthermore, the project successfully achieved cultural relevance and automation. The system's design, which allowed for traditional practices like members choosing their payout numbers and the admin led approval process, resonated with users. The implemented Automated Rotational Payout System ensured that the core promise of Esusu - guaranteed, timely lump sum access - was executed perfectly by code, eliminating human delay or error in the disbursement process. This functional prototype served as tangible proof that a specialized digital solution could solve real world issues of fraud and inefficiency without compromising the cultural values of the Esusu system.

CHAPTER FIVE

RECOMMENDATIONS AND CONCLUSION

5.1 RECOMMENDATIONS

Based on the successful implementation and testing of the Esusu App, several recommendations are proposed for future development to enhance the application's functionality, scalability, and impact.

1. **Implement Advanced Communication Features:** To strengthen the social aspect of the application, future development should include an in-app group chat or a forum. This would allow members to communicate directly with each other and the admin, facilitating group discussions, decision-making, and fostering a stronger sense of community. This would also serve as an official channel for announcements and a more formal way to resolve minor disputes.
2. **Expand the Rule-Based Fraud Detection System:** The current system uses a foundational set of rules. For a more robust and secure platform, it is recommended to expand this system to include more sophisticated checks. This could involve monitoring unusual login patterns (e.g., login from a new, distant location), flagging multiple failed payment attempts, or detecting collusion among members. Over time, as more data is collected, the system could even be enhanced with machine learning to identify more complex behavioral anomalies.
3. **Develop a Web-Based Admin Portal:** While the application is mobile-first, creating a web-based admin dashboard would be beneficial. This portal would give administrators a more comprehensive view of their group's data, with advanced analytics on contribution

trends, member activity, and detailed audit trails. A desktop interface would provide a more powerful tool for managing multiple groups and large volumes of data.

4. **Integrate Micro-Loan Functionality:** As a long-term goal, the application could be expanded to include micro-loan features for members. Based on their contribution history and participation within the Esusu group, a member could be eligible for a small, short-term loan. This would further enhance financial inclusion and provide a new layer of value to the platform.

5.2 CONCLUSION

The Esusu App project successfully addressed the critical need for a modern, secure, and transparent digital platform for traditional African rotational savings schemes. By replacing the manual, paper based system with a streamlined and automated digital solution, the project successfully solved problems of inefficiency, a lack of transparency, and susceptibility to fraud.

The application's core functionalities - from secure MERN stack user authentication, which included mandatory BVN or NIN authentication, to real time contribution tracking via the Dual Balance Display - proved that technology could be a powerful tool for preserving cultural practices while making them more reliable and accessible. The design was heavily focused on enhancing security and accountability through the logic based Integrity Alerts System, which publicly logged all critical activities, and the Automated Rotational Payout System, which eliminated the risk of administrative error during fund disbursement.

The design prioritized cultural relevance and a user centric approach, ensuring the application was a trusted and intuitive extension of a long standing communal tradition. In its final form, the Esusu App stands as a viable and robust prototype that demonstrated the feasibility of modernizing informal financial systems. It laid a solid foundation for future enhancements and

served as a significant contribution to the field of financial technology, proving that innovative solutions for emerging markets could be built by aligning modern engineering principles with deep cultural understanding and trust.

REFERENCES

Abubakar, M., & Sualihu, A. (2025). Financial Inclusion through Fintech Innovations in Ghana: Opportunities, Challenges, and Developmental Impacts. *Challenges, and Developmental Impacts (June 30, 2025)*.https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=5383263

Adeola, O. *et al*, (2022). Savings groups in Nigeria. https://www.emerald.com/books/edited-volume/11780/chapter-abstract/81732434/Savings-Groups-in-Nigeria?__cf_chl_tk=Ii_dKERsxiV45SFkmvgx4i95uJ5WTfPMYXgO_EvKRBs-1781857786-1.0.1.1-iIkcx1gPLohbcc76kAAmIX2UQ62hdnRPufw0enl2om8

Aganze, R., *et al*.,(2025). Determinants of sustainability of informal community-based financial groups: insights from a survival analysis approach. *The Journal of Development Studies*, *61*(10)1660-1682. https://www.tandfonline.com/doi/pdf/10.1080/00220388.2025.2487014

Ardener, S., & Burman, S. (Eds.). (2024). *Money-go-rounds: The importance of ROSCAs for women*. Taylor & Francis. https://books.google.com/books?hl=en&lr=lang_en&id=vqokEQAAQBAJ&oi=fnd&pg=PT7&dq=ROSCA+%22social+capital%22+%22mutual+trust%22+collateral&ots=6O1UM8gkws&sig=ocuRx3XeTyCPYsWS8J2yJuYRPy4

Batool, Z. *et al*. (2023). B-rosca: a smart contract-based selection of rosca communities using collateral. In *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)* (pp. 7-12). IEEE.https://ieeexplore.ieee.org/abstract/document/10237036/

Batool, Z. *et al*., (2023). B-rosca: a smart contract-based selection of rosca communities using collateral. In *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)* (pp. 7-12). IEEE.https://ieeexplore.ieee.org/abstract/document/10237036/

Boachie, C., & Adu-Darko, E. A. (2022). Ghana: Susu, village savings and loans, credit union, rotating savings system. https://www.emerald.com/books/edited-volume/11780/chapter/81732164

Chika, A. C. (2024). Finance Pooling: The self-coordination of 'Esusu'saving scheme among traders. https://thesis.eur.nl/pub/75775/ISS_GDP_RP_2023_24_Akamere-Cynthia.pdf

Chilima, M. (2022). A structured review of savings group literature. *PQDT-Global*.https://search.proquest.com/openview/0cde31159202aebdc14813021c50dcaf/1?pq-origsite=gscholar&cbl=2026366&diss=y

Ethan, D. (2022). Women's Economic Empowerment through Informal Saving Groups in Rwanda.https://www.researchgate.net/profile/Beauden-John/publication/398752235_Women's_Economic_Empowerment_through_Informal_Saving_Groups_in_Rwanda/links/6941b48a27359023a00d51b3/Womens-Economic-Empowerment-through-Informal-Saving-Groups-in-Rwanda.pdf

Ezeanowi, N. C. (2025). Growth patterns of internationalizing fintech companies in Nigeria.https://www.theseus.fi/bitstream/handle/10024/906812/Ezeanowi_Nnaemeka.pdf?sequence=3

Ghadamian, R. (2025). REGENERATIVE CAPITAL THEORY: Beyond Debt, Equity, and Grants. *Equity, and Grants (November 23, 2025)*. https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=5788982

Hossein, C. S., & Christabell, P. J. (Eds.). (2022). *Community economies in the Global South: Case studies of rotating savings, credit associations, and economic cooperation*. Oxford University Press. https://books.google.com/books?hl=en&lr=lang_en&id=hGlbEAAAQBAJ&oi=fnd&pg=PP1&dq=cooperative+saving+in+Africa+is+a+deeply+rooted+communal+practice+&ots=h-53Kb7QiY&sig=o5y5YKfmP8-2UBTwEOAVIsdHB-s

https://newprairiepress.org/cgi/viewcontent.cgi?article=1523&context=jiaee

Iwara, I. O. *et al*., (2021). Traditional savings association for entrepreneurial success in Africa: A case study of rotative Stokvel enterprise. *Academy of Entrepreneurship Journal*, *27*(2), 1-15.https://www.researchgate.net/profile/Ishmael-Iwara/publication/352028783_TRADITIONAL_SAVINGS_ASSOCIATION_FOR_ENTREPRENEURIAL_SUCCESS_IN_AFRICA_A_CASE_STUDY_OF_ROTATING_STOKVEL_ENTERPRISE/links/60b6339092851cde8847f3ef/TRADITIONAL-SAVINGS-ASSOCIATION-FOR-ENTREPRENEURIAL-SUCCESS-IN-AFRICA-A-CASE-STUDY-OF-ROTATING-STOKVEL-ENTERPRISE.pdf

Jones, L. S., & Maynard Jr, G. (2023). Unfulfilled promises of the fintech revolution. https://papers.ssrn.com/sol3/Delivery.cfm?abstractid=4031044

Juma, M. H.,*et al*., (2023). Smartphone Use in Financial Management among Women's Informal Saving Groups in Dodoma, Tanzania. *The African Journal of Information Systems, 15*(2), 2.https://digitalcommons.kennesaw.edu/cgi/viewcontent.cgi?article=2153&context=ajis

Lamino, P. (2025). Community Perspectives on Savings and Internal Lending Communities Program: Insights from Rural Guatemala. *Journal of International Agricultural and Extension Education*, *32*(3), 447-466.

Milevsky, M. A., & Salisbury, T. S. (2025). The Riccati tontine: how to satisfy regulators on average. *The Geneva Risk and Insurance Review*, *50*(1), 72-102.https://arxiv.org/pdf/2402.14555

Muchapireyi, H. T. (2025). Trustless–Participation in ROSCA Games.https://www.preprints.org/frontend/manuscript/b4365622bf94037faa418f5b7e418ab0/download_pub

Ngowi, A. *et al*. (2025). Formalization of informal digital savings and lending associations.https://repository.mocu.ac.tz/xmlui/bitstream/handle/123456789/2107/Formalisation%20of%20informal%20digital%20savings%20and%20lending%20associations%20a%20pathway%20to%20digitally%20led%20co-operative%20financial%20institutions%20in%20Tanzania.pdf?sequence=1&isAllowed=y

Oluwabukola, O. I. *et al*, (2025). COMPUTERIZATION OF PERSONAL FINANCIAL MANAGEMENT FOR MARKET WOMEN: A CATALYST FOR GRASSROOTS SOCIO-ECONOMIC DEVELOPMENT. *THEME: The Role of Applicable Technologies in Engendering Socio-Economic Development*, 299. https://www.researchgate.net/profile/Bamidele-Oluwade/publication/400288999_PROCEEDINGS_OF_THE_2025_5TH_INTERNATIONAL_CONFERENCE_OF_THE_SOCIETY_FOR_THE_ADVANCEMENT_OF_ICT_COMPARATIVE_KNOWLEDGE_SOCTHADICKconf'25/links/697e1adc64ca8a382087e7a0/PROCEEDINGS-OF-THE-2025-5TH-INTERNATIONAL-CONFERENCE-OF-THE-SOCIETY-FOR-THE-ADVANCEMENT-OF-ICT-COMPARATIVE-KNOWLEDGE-SOCTHADICKconf25.pdf#page=309

Phil-Ugochukwu, A. I. (2024). Informal financial savings practices to facilitate formal financial inclusion. *World Journal of Advanced Research and Reviews*, *24*(3), 1970-1979.https://www.researchgate.net/profile/Ada-Phil-Ugochukwu/publication/387340538_Informal_financial_savings_practices_to_facilitate_formal_financial_inclusion/links/6769a83ce74ca64e1f257986/Informal-financial-savings-practices-to-facilitate-formal-financial-inclusion.pdf

Rahman, A. M., & Sultana, S. (2026). Bonik Somiti: A Social-market Tool for Safe, Accountable, and Harmonious Informal E-Market Ecosystem in Bangladesh. In *Proceedings of the 2026 Designing Interactive Systems Conference* (pp. 1296-1314).https://dl.acm.org/doi/pdf/10.1145/3800645.3812955

Salako, O. *et al*. (2021). The risk of 'trust disruption' in the digitization of traditional financial systems in Nigeria. Journal of Financial Sociology, 3(1), 34-49. https://sustainabledfs.lbs.edu.ng/wp-content/uploads/2025/04/Digital-Financial-Inclusion-and-Entrepreneurship-Growth.pdf

Siddiqui, A. (2026). Trust as collateral: informal finance among urban informal workers in Mumbai. *Qualitative Research in Financial Markets*, 1-15.https://www.emerald.com/qrfm/article/doi/10.1108/QRFM-09-2025-0301/1359402

Zafar, M. B. (2026). Exploring the intellectual landscape of ROSCAs: a fifty-year bibliometric review. *International Journal of Bank Marketing*, 1-27. https://www.emerald.com/ijbm/article/doi/10.1108/IJBM-06-2025-0486/1359234

Zambrano, A. F. *et al*., (2023). Rotating savings and credit associations: A scoping review. *World Development Sustainability*, *3*, 100081.https://www.sciencedirect.com/science/article/pii/S2772655X23000393

APPENDIX

```
// In backend/controllers/userController.js

const asyncHandler = require('express-async-handler');
const User = require('../models/userModel');
const jwt = require('jsonwebtoken');

// Function to generate a JWT token
const generateToken = (id) => {
  // process.env.JWT_SECRET is a secret key we need to add to our .env
file
  return jwt.sign({ id }, process.env.JWT_SECRET, {
    expiresIn: '30d', // Token will be valid for 30 days
  });
};

/**
 * @desc    Register a new user
 * @route   POST /api/users/register
 * @access  Public
 */
const registerUser = asyncHandler(async (req, res) => {
  // 1. Get data from the request body
  const { firstName, lastName, phoneNumber, email, password } =
req.body;

  // 2. Check if all fields exist
  if (!firstName || !lastName || !phoneNumber || !email || !password)
{
```

```
    res.status(400);
    throw new Error('Please fill in all fields');
  }

  // 3. Check if user already exists in the database
  const userExists = await User.findOne({ email });
  if (userExists) {
    res.status(400);
    throw new Error('User with this email already exists');
  }

  // 4. Create a new user (password will be auto-hashed by our model)
  const user = await User.create({
    firstName,
    lastName,
    phoneNumber,
    email,
    password,
  });

  // 5. If user was created successfully, send back user data and a
token
  if (user) {
    res.status(201).json({
      _id: user.id,
      firstName: user.firstName,
      lastName: user.lastName,
      email: user.email,
      token: generateToken(user._id),
    });
  } else {
```

```
    res.status(400);
    throw new Error('Invalid user data');
  }
});

/**
 * @desc    Authenticate (login) a user
 * @route   POST /api/users/login
 * @access  Public
 */
const loginUser = asyncHandler(async (req, res) => {
  const { email, password } = req.body;

  // 1. Find the user by email
  const user = await User.findOne({ email });

  // 2. Check if user exists AND if passwords match
  if (user && (await user.matchPassword(password))) {
    // 3. Send back user data and token
    res.json({
      _id: user.id,
      firstName: user.firstName,
      lastName: user.lastName,
      email: user.email,
      token: generateToken(user._id),
    });
  } else {
    res.status(401); // 401 means 'Unauthorized'
    throw new Error('Invalid email or password');
  }
});
```

```
module.exports = {
  registerUser,
  loginUser,
};
```

```
// In backend/controllers/requestController.js

const asyncHandler = require('express-async-handler');
const JoinRequest = require('../models/joinRequestModel');
const Group = require('../models/groupModel');

/**
 * @desc    Submit a new join request
 * @route   POST /api/requests
 * @access  Private
 */
const submitJoinRequest = asyncHandler(async (req, res) => {
  const {
    groupId,
    fullName,
    phoneNumber,
    passportPhotograph,
    chosenNumber,
    accountNumber,
    bankName,
    accountName,
  } = req.body;

  const userId = req.user._id; // From our 'protect' middleware
```

```
// 1. Find the group they want to join
const group = await Group.findById(groupId);
if (!group) {
  res.status(404);
  throw new Error('Group not found');
}

// 2. Check if the chosen number is actually available
if (!group.availableNumbers.includes(Number(chosenNumber))) {
  res.status(400);
  throw new Error('That number is not available in this group');
}

// 3. Check if user already has a pending request for this group
const existingRequest = await JoinRequest.findOne({
  group: groupId,
  user: userId,
  status: 'pending',
});
if (existingRequest) {
  res.status(400);
  throw new Error('You already have a pending request for this group');
}

// 4. Create and save the new join request
const request = await JoinRequest.create({
  group: groupId,
  user: userId,
  fullName,
  phoneNumber,
  passportPhotograph, // This will be a URL string for now
```

```
    chosenNumber: Number(chosenNumber),
    accountNumber,
    bankName,
    accountName,
    status: 'pending', // Explicitly set as pending
  });
  res.status(201).json(request);
});

/**
 * @desc    Get all pending requests for a specific group (for admin)
 * @route   GET /api/requests/group/:groupId
 * @access  Private (Admin only)
 */
const getPendingRequests = asyncHandler(async (req, res) => {
  const { groupId } = req.params;

  // 1. Find the group
  const group = await Group.findById(groupId);
  if (!group) {
    res.status(404);
    throw new Error('Group not found');
  }

  // 2. Check if logged-in user is the admin
  if (group.admin.toString() !== req.user._id.toString()) {
    res.status(401);
    throw new Error('User not authorized');
  }

  // 3. Find all pending requests for this group
```

```
  const requests = await JoinRequest.find({
    group: groupId,
    status: 'pending',
  }).populate('user', 'firstName lastName email'); // Show who sent it

  res.status(200).json(requests);
});

/**
 * @desc    Approve a join request (Admin only)
 * @route   PUT /api/requests/:id/approve
 * @access  Private (Admin only)
 */
const approveRequest = asyncHandler(async (req, res) => {
  const { id } = req.params; // This is the ID of the JoinRequest

  // 1. Find the request
  const request = await JoinRequest.findById(id);
  if (!request) {
    res.status(404);
    throw new Error('Request not found');
  }

  // 2. Find the group
  const group = await Group.findById(request.group);
  if (!group) {
    res.status(404);
    throw new Error('Group not found');
  }

  // 3. Check if logged-in user is the admin of *this* group
```

```
  if (group.admin.toString() !== req.user._id.toString()) {
    res.status(401);
    throw new Error('User not authorized');
  }

  // 4. --- This is the main logic ---
  // 4a. Add the new member to the group's 'members' array
  const newMember = {
    user: request.user,
    chosenNumber: request.chosenNumber,
    bankName: request.bankName,
    accountName: request.accountName,
    accountNumber: request.accountNumber,
  };
  group.members.push(newMember);

  // 4b. Remove the chosen number from 'availableNumbers'
  group.availableNumbers = group.availableNumbers.filter(
    (num) => num !== request.chosenNumber
  );

  // 4c. Update the request status
  request.status = 'approved';

  // 5. Save the changes to the database
  await group.save();
  await request.save();

  res.status(200).json({ message: 'Member approved and added to group' });
});
```

```
/**
 * @desc    Reject a join request (Admin only)
 * @route   PUT /api/requests/:id/reject
 * @access  Private (Admin only)
 */
const rejectRequest = asyncHandler(async (req, res) => {
  const { id } = req.params; // This is the ID of the JoinRequest

  // 1. Find the request
  const request = await JoinRequest.findById(id);
  if (!request) {
    res.status(404);
    throw new Error('Request not found');
  }

  // 2. Find the group (to check admin status)
  const group = await Group.findById(request.group);
  if (!group) {
    res.status(404);
    throw new Error('Group not found');
  }

  // 3. Check if logged-in user is the admin
  if (group.admin.toString() !== req.user._id.toString()) {
    res.status(401);
    throw new Error('User not authorized');
  }

  // 4. Update status to 'rejected'
  request.status = 'rejected';
  await request.save();
```

```
  res.status(200).json({ message: 'Request rejected' });
});

module.exports = {
  submitJoinRequest,
  getPendingRequests,
  approveRequest,
  rejectRequest,
};
```

```
// In backend/controllers/groupController.js

const asyncHandler = require('express-async-handler');
const Group = require('../models/groupModel');
const User = require('../models/userModel'); // We might need this
later

/**
 * @desc    Create a new Esusu group
 * @route   POST /api/groups
 * @access  Private (Protected)
 */
const createGroup = asyncHandler(async (req, res) => {
  // 1. Get all the data from the form
  const {
    groupName,
    payoutAmount,
    startingDate,
    contributionAmount,
    numberOfMembers,
    phoneNumber,
```

```
    corporateAccount,
    bankName,
    accountName,
    payoutInterval,
    payoutTime,
    adminChosenNumber, // <-- OUR NEW FIELD
  } = req.body;

  // 2. Validation
  if (
    !groupName || !payoutAmount || !startingDate ||
!contributionAmount ||
    !numberOfMembers || !phoneNumber || !corporateAccount || !bankName
||
    !accountName || !payoutInterval || !payoutTime ||
!adminChosenNumber
  ) {
    res.status(400);
    throw new Error('Please fill in all fields, including your chosen
number');
  }

  // Convert types
  const numMembers = Number(numberOfMembers);
  const chosenNum = Number(adminChosenNumber);

  if (chosenNum < 1 || chosenNum > numMembers) {
    res.status(400);
    throw new Error('Your chosen number is not within the group size
range');
  }
```

```
  // 3. Prepare the group data (NEW LOGIC)

  // Create an array of all possible numbers (e.g., [1, 2, 3, 4, 5])
  const allNumbers = Array.from({ length: numMembers }, (_, i) => i +
1);

  // Create the availableNumbers array by removing the admin's chosen
number
  const availableNumbers = allNumbers.filter((num) => num !==
chosenNum);

  // Create the admin's member object
  const adminAsMember = {
    user: req.user._id, // from 'protect' middleware
    chosenNumber: chosenNum,
    // Admin's bank details (for receiving payout)
    // We can pre-fill this from their user profile or create group
form
    // For now, let's leave it blank, they can update it later.
  };

  // 4. Create the new group in the database
  const group = await Group.create({
    admin: req.user._id,
    groupName,
    payoutAmount,
    startingDate,
    contributionAmount,
    numberOfMembers: numMembers,
    phoneNumber,
```

```
    corporateAccount,
    bankName,
    accountName,
    payoutInterval,
    payoutTime,
    availableNumbers: availableNumbers, // The list *without* the
admin's number
    members: [adminAsMember], // Add the admin as the first member
  });

  // 5. Send back the new group data
  res.status(201).json(group);
});

/**
 * @desc    Get all available Esusu groups (for the "Explore" page)
 * @route   GET /api/groups
 * @access  Public
 */
const getGroups = asyncHandler(async (req, res) => {
  // Find all groups and sort them by newest first
  const groups = await Group.find({}).sort({ createdAt: -1 });
  res.status(200).json(groups);
});

/**
 * @desc    Get a single group by its ID
 * @route   GET /api/groups/:id
 * @access  Private (must be logged in to see group details)
 */
const getGroupById = asyncHandler(async (req, res) => {
```

```
  const group = await Group.findById(req.params.id)
    .populate('admin', 'firstName lastName email') // Get admin's
name/email
    .populate('members.user', 'firstName lastName email'); // Get
member's name/email

  if (!group) {
    res.status(404);
    throw new Error('Group not found');
  }

  // We'll add a check later to see if the user is a member
  res.status(200).json(group);
});

// Export the functions
module.exports = {
  createGroup,
  getGroups,
  getGroupById,
};
```

```
// In backend/controllers/contributionController.js

const asyncHandler = require('express-async-handler');
const Contribution = require('../models/contributionModel');
const Group = require('../models/groupModel');
const User = require('../models/userModel'); // Good to have, just in
case

/**
 * @desc    Log a new contribution
 * @route   POST /api/contributions
 * @access  Private
 */
const logContribution = asyncHandler(async (req, res) => {
  // 1. Get data from the request body
  const { groupId, amount, method, reference } = req.body;
  const userId = req.user._id; // from 'protect' middleware

  // 2. Check if required data is missing
  if (!groupId || !amount || !method) {
    res.status(400);
    throw new Error('Missing contribution data');
  }

  // 3. Find the group to make sure it exists
  const group = await Group.findById(groupId);
  if (!group) {
    res.status(404);
    throw new Error('Group not found');
  }
```

```
  // 4. (Optional but good) Check if user is a member
  const isMember = group.members.find(
    (member) => member.user.toString() === userId.toString()
  );
  if (!isMember) {
    res.status(401);
    throw new Error('You are not a member of this group');
  }

  // 5. Create the contribution log
  const contribution = await Contribution.create({
    group: groupId,
    user: userId,
    amount: Number(amount),
    method,
    reference: reference || null,
    status: 'successful', // We set it to 'successful' immediately
  });

  // 6. If successful, send back the new log
  if (contribution) {
    res.status(201).json(contribution);
  } else {
    res.status(400);
    throw new Error('Invalid contribution data');
  }
});

/**
 * @desc    Get the total contribution balance for a group
```

```
 * @route   GET /api/contributions/group/:groupId/total
 * @access  Private
 */
const getGroupBalance = asyncHandler(async (req, res) => {
  // Find all successful contributions for this group
  const contributions = await Contribution.find({
    group: req.params.groupId,
    status: 'successful',
  });

  // Sum them up using reduce
  const total = contributions.reduce((acc, item) => acc + item.amount,
0);

  res.status(200).json({ totalBalance: total });
});

/**
 * @desc    Get the logged-in user's total contribution balance for a
group
 * @route   GET /api/contributions/group/:groupId/my-balance
 * @access  Private
 */
const getMyBalance = asyncHandler(async (req, res) => {
  // Find all successful contributions for THIS group AND THIS user
  const contributions = await Contribution.find({
    group: req.params.groupId,
    user: req.user._id,
    status: 'successful',
  });
```

```
  // Sum them up
  const total = contributions.reduce((acc, item) => acc + item.amount,
0);

  res.status(200).json({ myBalance: total });
});

// Export all the functions
module.exports = {
  logContribution,
  getGroupBalance,
  getMyBalance,
};
```

```
// In backend/middleware/authMiddleware.js

const jwt = require('jsonwebtoken');
const asyncHandler = require('express-async-handler');
const User = require('../models/userModel');

const protect = asyncHandler(async (req, res, next) => {
  let token;

  // Check if the request headers contain an 'authorization' field
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith('Bearer')
  ) {
    try {
      // 1. Get the token from the header (e.g., "Bearer <token>")
      token = req.headers.authorization.split(' ')[1];
```

```
      // 2. Verify the token using our JWT_SECRET
      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      // 3. Find the user by the ID that was in the token
      // We attach this 'user' object to the 'req'
      // We exclude the password when fetching
      req.user = await User.findById(decoded.id).select('-password');

      // 4. Call the next middleware or controller
      next();
    } catch (error) {
      console.error(error);
      res.status(401);
      throw new Error('Not authorized, token failed');
    }
  }

  if (!token) {
    res.status(401);
    throw new Error('Not authorized, no token');
  }
});

module.exports = { protect };

// In backend/models/userModel.js

const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
```

```
const userSchema = mongoose.Schema(
  {
    // Based on your Signup Page fields
    firstName: {
      type: String,
      required: [true, 'Please add a first name'],
    },
    lastName: {
      type: String,
      required: [true, 'Please add a last name'],
    },
    phoneNumber: {
      type: String,
      required: [true, 'Please add a phone number'],
    },
    email: {
      type: String,
      required: [true, 'Please add an email'],
      unique: true, // No two users can have the same email
      match: [
        /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/,
        'Please add a valid email',
      ],
    },
    password: {
      type: String,
      required: [true, 'Please add a password'],
      minlength: 6, // Enforce a minimum password length
    },
  },
  {
```

```
    // This adds 'createdAt' and 'updatedAt' fields automatically
    timestamps: true,
  }
);

// This is middleware that runs *before* a new user is saved
userSchema.pre('save', async function (next) {
  // 'this' refers to the user document being saved
  if (!this.isModified('password')) {
    // If the password hasn't been changed, skip hashing
    return next();
  }

  // 1. Generate a 'salt' - random characters for hashing
  const salt = await bcrypt.genSalt(10);
  // 2. Hash the password using the salt
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

// Method to compare entered password with the hashed password
userSchema.methods.matchPassword = async function (enteredPassword) {
  return await bcrypt.compare(enteredPassword, this.password);
};

const User = mongoose.model('User', userSchema);

module.exports = User;
```

```
// In backend/models/joinRequestModel.js

const mongoose = require('mongoose');

const joinRequestSchema = mongoose.Schema(
  {
    // Link to the group the user wants to join
    group: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'Group',
    },
    // Link to the user who is sending the request
    user: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'User',
    },
    // Status of the request (for the admin)
    status: {
      type: String,
      required: true,
      enum: ['pending', 'approved', 'rejected'],
      default: 'pending',
    },

    // --- Data from the join form ---
    fullName: {
      type: String,
      required: [true, 'Please provide your full name'],
```

```
    },
    phoneNumber: {
      type: String,
      required: [true, 'Please provide your phone number'],
    },
    // For now, we'll treat this as a URL to a hosted image
    passportPhotograph: {
      type: String,
    },
    chosenNumber: {
      type: Number,
      required: [true, 'Please choose a payout number'],
    },
    accountNumber: {
      type: String,
      required: [true, 'Please provide your account number'],
    },
    bankName: {
      type: String,
      required: [true, 'Please provide your bank name'],
    },
    accountName: {
      type: String,
      required: [true, 'Please provide your account name'],
    },
  },
  {
    timestamps: true,
  }
);
```

```
const JoinRequest = mongoose.model('JoinRequest', joinRequestSchema);

module.exports = JoinRequest;
```

```
// In backend/models/groupModel.js

const mongoose = require('mongoose');

const groupSchema = mongoose.Schema(
  {
    // 1. User who created the group (the Admin)
    // We link this to the 'User' model
    admin: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'User', // This creates the relationship
    },
    // 2. Details from your "Create Group" form
    groupName: {
      type: String,
      required: [true, 'Please add a group name'],
    },
    payoutAmount: {
      type: Number,
      required: [true, 'Please add a payout amount'],
    },
    startingDate: {
      type: Date,
      required: [true, 'Please add a starting date'],
    },
    contributionAmount: {
```

```
    type: Number,
    required: [true, 'Please add a contribution amount'],
  },
  numberOfMembers: {
    type: Number,
    required: [true, 'Please add the number of members'],
  },
  // Admin's phone number (can be different from their user profile)
  phoneNumber: {
    type: String,
    required: [true, 'Please add a contact phone number'],
  },
  // Admin's bank details for contributions
  corporateAccount: {
    type: String,
    required: [true, 'Please add the account number'],
  },
  bankName: {
    type: String,
    required: [true, 'Please add the bank name'],
  },
  accountName: {
    type: String,
    required: [true, 'Please add the account name'],
  },
  payoutInterval: {
    type: Number, // Storing this in days, as you specified
    required: [true, 'Please add a payout interval (in days)'],
  },
  payoutTime: {
    type: String, // Storing as string, e.g., "3:00pm"
```

```
    required: [true, 'Please add a payout time'],
  },

  // 3. Group Status and Member Management
  status: {
    type: String,
    enum: ['Pending', 'Active', 'Completed'],
    default: 'Pending',
  },
 // A list of all members in the group
     members: [
       {
         user: {
           type: mongoose.Schema.Types.ObjectId,
           ref: 'User',
           required: true,
         },
         chosenNumber: {
           type: Number,
           required: true,
         },
         // We'll add these fields later when members join
         // but the admin can have them too
         bankName: { type: String },
         accountName: { type: String },
         accountNumber: { type: String },
       },
     ],
     // A list of available slots/numbers to pick
     availableNumbers: {
       type: [Number], // An array of numbers
```

```
        },
      },
  {
    timestamps: true, // Adds createdAt and updatedAt
  }
);

const Group = mongoose.model('Group', groupSchema);

module.exports = Group;
```

```
// In backend/models/contributionModel.js

const mongoose = require('mongoose');

const contributionSchema = mongoose.Schema(
  {
    // Link to the group
    group: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'Group',
    },
    // Link to the user who paid
    user: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: 'User',
    },
    amount: {
      type: Number,
```

```
      required: [true, 'Please add an amount'],
    },
    // Method of payment
    method: {
      type: String,
      required: true,
      enum: ['transfer', 'card'],
    },
    // Paystack transaction reference (if paid by card)
    reference: {
      type: String,
    },
    status: {
      type: String,
      required: true,
      enum: ['pending', 'successful', 'failed'],
      default: 'successful', // We'll assume successful for now
    },
  },
  {
    timestamps: true, // This will add 'createdAt' as the payment date
  }
);

const Contribution = mongoose.model('Contribution',
contributionSchema);

module.exports = Contribution;
```