DEV OPPS

SRS DOCUMENT

DEMO 1

STOCKFELLOW

BRIGHT BYTE ENTERPRISES

TABLE OF CONTENTS

INTRODUCTION

Stokvels have been a vital part of South African communities for generations, providing a trusted mechanism for collective savings and financial support. These groups foster a sense of community and mutual aid, enabling individuals to achieve goals that might otherwise be out of reach. However, traditional stokvels face significant challenges: trust issues due to lack of transparency, manual record-keeping prone to errors, and delays in payments that erode confidence. StockFellow aims to modernize this cultural practice by delivering a secure, automated, and inclusive digital solution that preserves its communal essence while overcoming these obstacles.

PROJECT DESCRIPTION

StockFellow is a modern financial solution designed to streamline and secure traditional stokvel savings groups. By leveraging automation, robust security, and open-source technologies, the platform addresses the core challenges of trust, efficiency, and compliance while introducing innovative features to enhance user experience.

Key Features

· **Secure User Registration & Verification:** Biometric authentication, ID verification, and credit scoring ensure safe onboarding, even for users without formal credit histories.

· **Automated Contributions & Payouts:** Scheduled debit orders and automated fund distribution eliminate delays, with bank API integration for seamless transactions.

· **Intelligent Group & Tier Management:** Users are assigned to affordability-based tiers using credit scoring, with automated promotions based on contribution consistency.

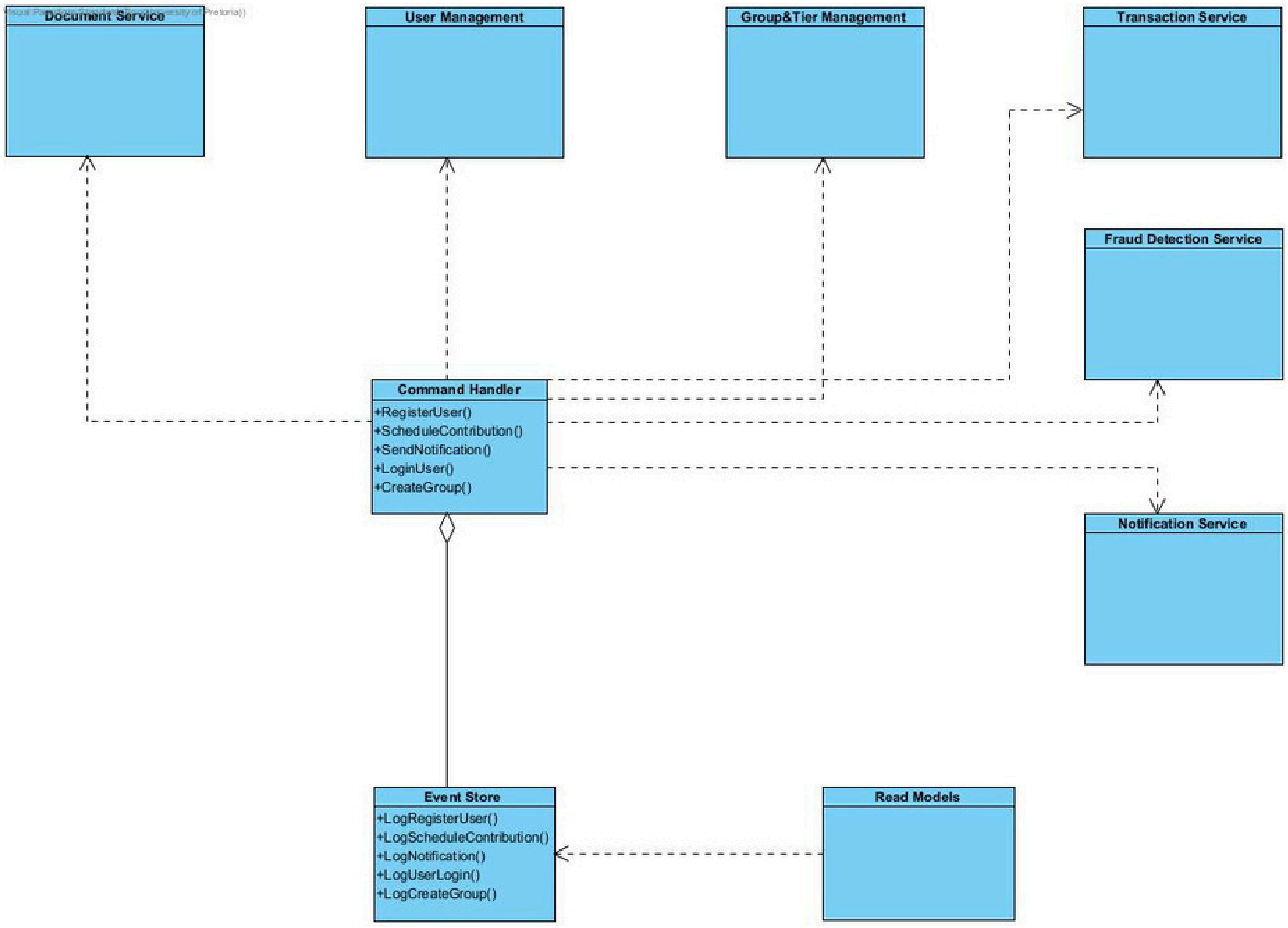
· **Fraud Detection & Security:** AI-driven fraud detection, multi-factor authentication (MFA), and real-time monitoring safeguard user funds and data.

· **Real-Time Notifications & Reporting:** SMS, email, and in-app alerts keep users informed about payments, tier changes, and group activities.

· **Compliance & Data Protection:** Adherence to POPIA, GDPR, and AML regulations, with encryption and audit logging for transparency.

· **Potential Enhancements:** AI-based risk assessment for emergency loans, blockchain for transparent transaction records, and USSD support for offline access.

DOMAIN MODEL



USER STORIES

As a Stokvel Group Member...

- I can register for an account using my email and phone number, with ID verification, face recognition and verified documents for security.
- I can log into my account using my email address and password with multi-factor authentication for secure access.
- I can create or join a stokvel group suited to my financial tier, which is based on my creditworthiness and/or reliability when using the app.
- I can verify my bank details to set up automated contributions via scheduled debit orders.

- I can view my group contribution history, current balance, and upcoming payout dates.
- I can receive real-time notifications for contributions, payouts, tier promotions and security alerts.
- I can request withdrawals of my stokvel savings.
- I can view my earned badges and rank based on my financial contributions.
- I can view the tasks to be done to advance to the next financial tier.

As a Stokvel Group Administrator...

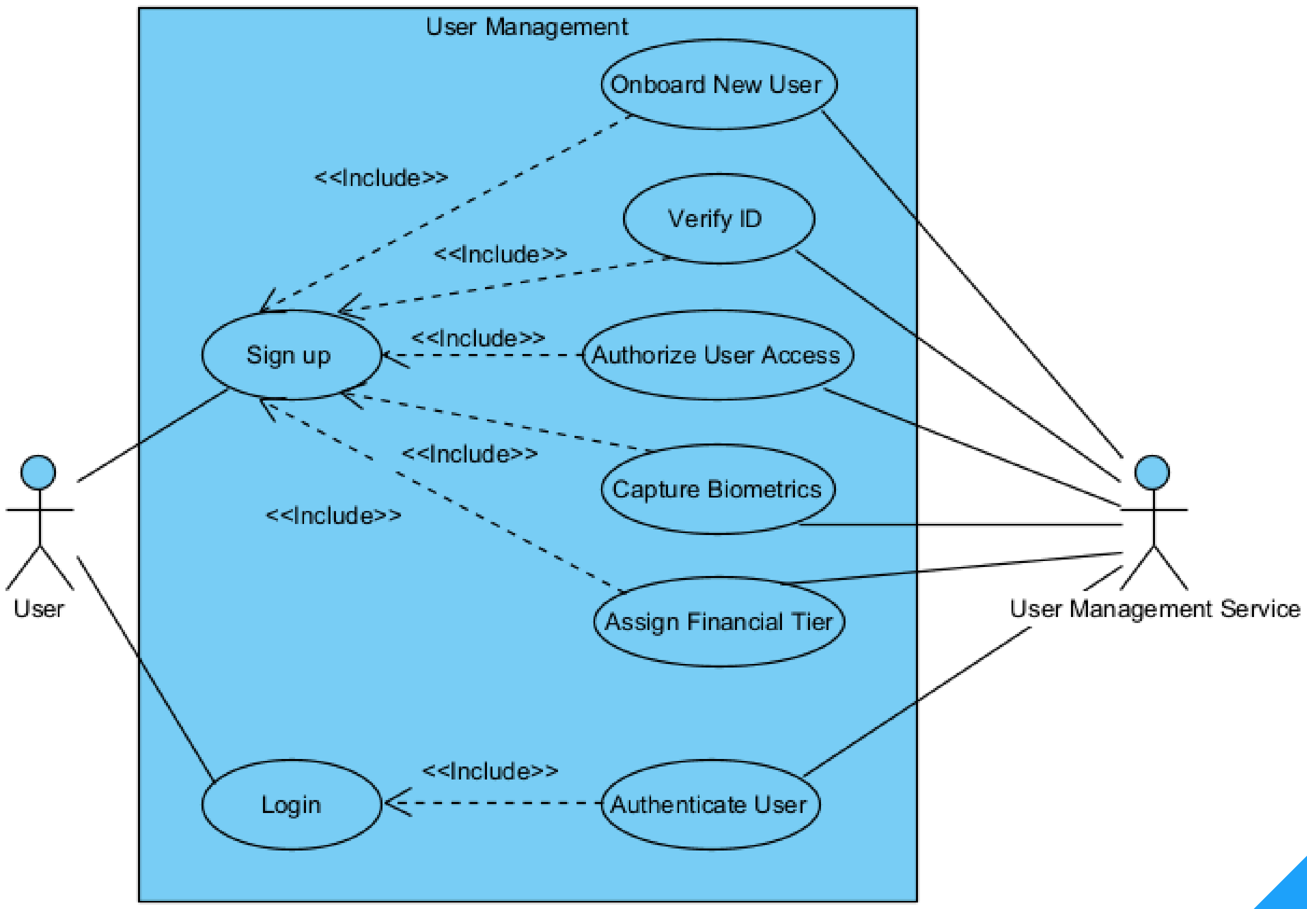
- I can do everything a stokvel group member can do.
- I can manage my owned stokvel groups, setting rules for contributors and payouts.
- I can approve or reject new member applications for my stokvel groups.

As a System Administrator...

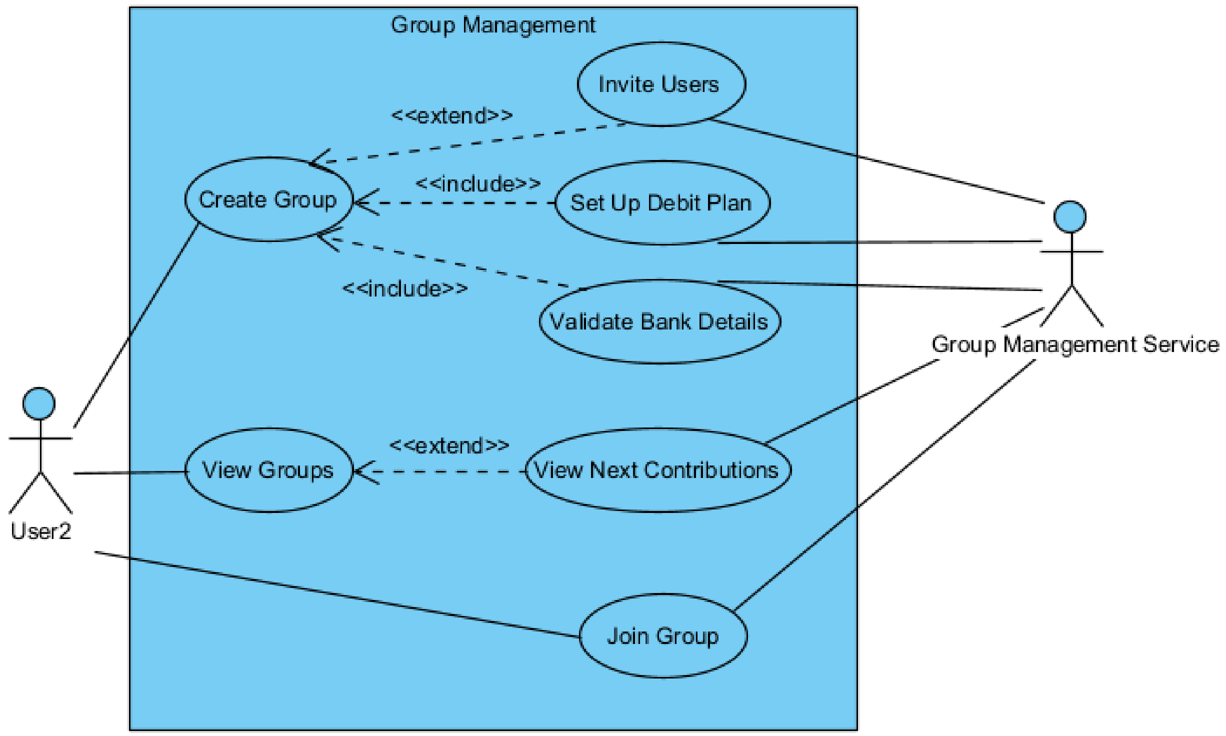
- I can view and manage users, including their roles and permissions.
- I can monitor the system for fraudulent activities to protect users.

USE CASE DIAGRAMS

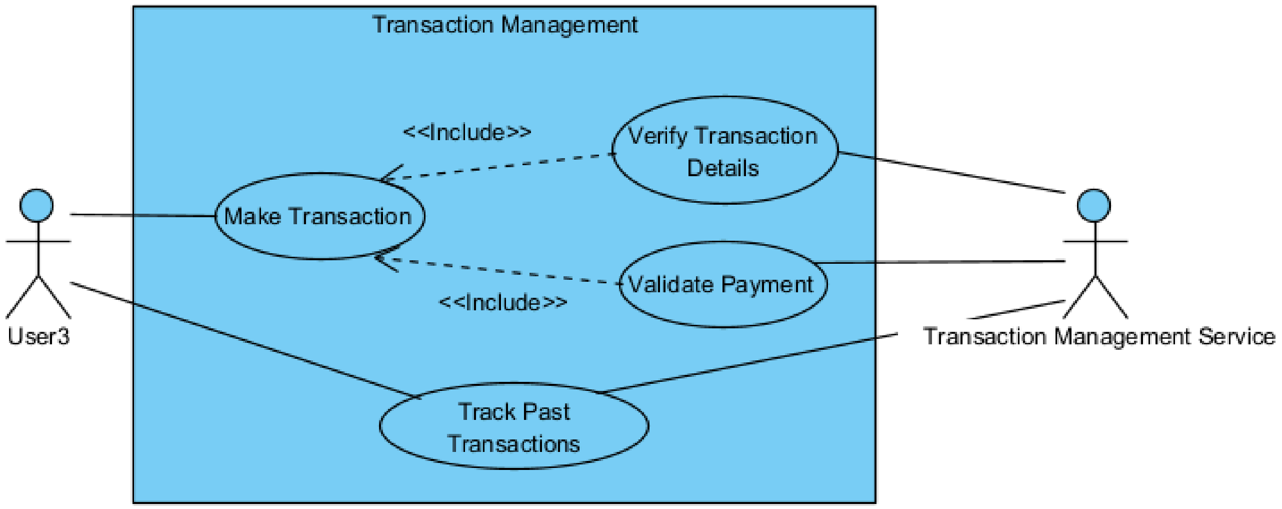
User Management



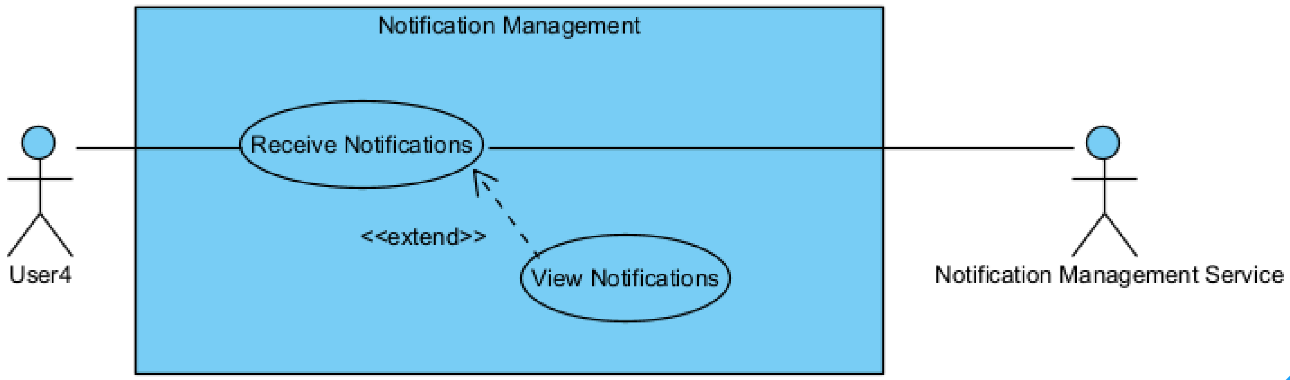
Group Management



Transaction Management



Notification Management



FUNCTIONAL REQUIREMENTS

R1: Set up a new account

- R1.1: Register a new user
  - **R1.1.1: Enter personal details**
  - The system shall prompt the user to input their full name, email address, and phone number during registration.
  - **R1.1.2: Verify email address**
  - The system shall send a verification link to the provided email address and require the user to confirm it before proceeding.
  - **R1.1.3: Set up a secure password**
  - The system shall enforce password complexity rules (e.g., minimum length, special characters) and securely store the hashed password.
- R1.2: Verify user identity
  - **R1.2.1: Upload ID documents**
  - The system shall allow users to upload scanned copies or photos of government-issued ID documents for verification.
  - **R1.2.2: Perform biometric authentication**
  - The system shall use facial recognition technology to match the user’s live image with the photo on their ID document.
  - **R1.2.3: Confirm verification status**
  - The system shall display the verification status (e.g., pending, approved, rejected) and notify the user upon completion.
- R1.3: Perform credit scoring
  - **R1.3.1: Collect financial information**
  - The system shall request additional financial details (e.g., income, employment status) to assess creditworthiness.
  - **R1.3.2: Run credit scoring algorithm**
  - The system shall use a predefined algorithm to calculate a credit score based on the provided financial data.
  - **R1.3.3: Assign user to a financial tier**
  - The system shall categorize the user into a specific tier (e.g., Tier 1, Tier 2) based on their credit score.

R2: Manage a group/stokvel

- R2.1: Create a new stokvel group
  - **R2.1.1: Define group name and description**
  - The system shall allow the group creator to input a unique name and a brief description for the stokvel group.
  - **R2.1.2: Set contribution amounts and schedules**
  - The system shall enable the creator to specify the contribution amount and frequency (e.g., monthly, bi-weekly).
  - **R2.1.3: Establish payout rules**
  - The system shall allow the creator to define payout rules, such as the order of payouts and conditions for fund distribution.
- R2.2: Join an existing group
  - **R2.2.1: Search for available groups**
  - The system shall provide a search function for users to find groups based on criteria like name, location, or contribution amount.
  - **R2.2.2: Submit a join request**
  - The system shall allow users to send a request to join a group, which may require approval from the group administrator.
  - **R2.2.3: Await approval from group administrator**
  - The system shall notify the user once the group administrator approves or rejects their join request.
- R2.3: Manage group membership
  - **R2.3.1: Add new members**
  - The system shall enable group administrators to invite or approve new members to join the group.
  - **R2.3.2: Remove existing members**
  - The system shall allow group administrators to remove members who violate group rules or fail to contribute.
  - **R2.3.3: Update member roles**
  - The system shall permit group administrators to assign or change roles (e.g., promote a member to co-administrator).

R3: Manage member contributions

- R3.1: Set up debit orders
  - **R3.1.1: Link bank account**
  - The system shall guide users through linking their bank account via a secure API for automated payments

  - **R3.1.2: Configure debit order amount and frequency**
  - The system shall allow users to set the exact amount and schedule (e.g., monthly on the 1st) for their contributions.
  - **R3.1.3: Confirm debit order setup**
  - The system shall send a confirmation notification once the debit order is successfully configured.
- R3.2: Track contributions
  - **R3.2.1: Record each contribution transaction**
  - The system shall log every successful contribution with details like amount, date, and transaction ID.
  - **R3.2.2: Display contribution history to the user**
  - The system shall provide a dashboard where users can view their past contributions in chronological order.
  - **R3.2.3: Calculate total contributions made**
  - The system shall compute and display the cumulative amount a user has contributed to their group.
- R3.3: Handle missed contributions
  - **R3.3.1: Detect missed contributions**
  - The system shall identify when a scheduled contribution fails to process (e.g., due to insufficient funds).
  - **R3.3.2: Send notification to the user**
  - The system shall alert the user via their preferred notification channel about the missed contribution.
  - **R3.3.3: Apply penalties as per group rules**
  - The system shall automatically apply any penalties (e.g., fees or tier demotion) defined by the group.

R4: Process member payouts

- R4.1: Schedule payouts
  - **R4.1.1: Determine payout dates based on group rules**
  - The system shall calculate payout dates according to the group's predefined schedule (e.g., every month, every 3 months).
  - **R4.1.2: Identify the next payout recipient**
  - The system shall select the next eligible member for a payout based on the group's rotation or rules.

  - **R4.1.3: Prepare payout transaction**
  - The system shall queue the payout transaction for processing on the scheduled date.
- R4.2: Distribute funds
  - **R4.2.1: Verify recipient's bank account details**
  - The system shall confirm that the recipient's bank account is linked and valid before initiating the transfer.
  - **R4.2.2: Initiate fund transfer**
  - The system shall use a secure API to transfer the payout amount from the group's pooled funds to the recipient's account.
  - **R4.2.3: Confirm successful transfer**
  - The system shall verify the transaction's success and update the payout status accordingly.
- R4.3: Record payout transactions
  - **R4.3.1: Log payout details**
  - The system shall record the payout amount, recipient, date, and transaction reference for auditing.
  - **R4.3.2: Update group's financial records**
  - The system shall adjust the group's total funds and individual member balances after each payout.
  - **R4.3.3: Make transaction history available for audit**
  - The system shall store payout records in a secure, accessible format for transparency and compliance.

R5: Manage User Tiers

- R5.1: Assign users to tiers
  - **R5.1.1: Evaluate user's credit score**
  - The system shall use the credit scoring algorithm to determine the user's financial reliability.
  - **R5.1.2: Determine appropriate tier based on score**
  - The system shall map the credit score to a predefined tier (e.g., Tier 1 for high scores, Tier 3 for low).

    - **R5.1.3: Notify user of their tier assignment**
    - The system shall inform the user of their assigned tier and explain the associated benefits or restrictions.
- R5.2: Monitor contribution behavior
    - **R5.2.1: Track consistency of contributions**
    - The system shall monitor whether users meet their contribution obligations on time.
    - **R5.2.2: Calculate reliability score**
    - The system shall compute a reliability score based on the user's contribution history (e.g., on-time payments).
    - **R5.2.3: Check against promotion criteria**
    - The system shall compare the reliability score against thresholds for tier promotion.
- R5.3: Promote users to higher tiers
    - **R5.3.1: Identify users eligible for promotion**
    - The system shall automatically detect users who meet the criteria for a higher tier.
    - **R5.3.2: Update user's tier status**
    - The system shall upgrade the user's tier in the database and reflect the change in their profile.
    - **R5.3.3: Notify user of tier promotion**
    - The system shall send a congratulatory notification to the user about their tier upgrade.

R6: Provide security and fraud detection services

- R6.1: Implement multi-factor authentication (MFA)
    - **R6.1.1: Enable MFA for user accounts**
    - The system shall allow users to set up MFA using methods like SMS codes or authenticator apps.
    - **R6.1.2: Require MFA for login**
    - The system shall enforce MFA during the login process for all users.
    - **R6.1.3: Require MFA for fund transfers**
    - The system shall mandate MFA confirmation before processing any fund transfers or payouts.

- R6.2: Detect fraudulent activities
    - **R6.2.1: Analyze transaction patterns**
    - The system shall use AI algorithms to identify unusual patterns in contributions or payouts.
    - **R6.2.2: Flag suspicious activities**
    - The system shall mark transactions or user behaviors that deviate from normal patterns for review.
    - **R6.2.3: Alert system administrators**
    - The system shall notify administrators immediately when potential fraud is detected.
- R6.3: Monitor system in real-time
    - **R6.3.1: Set up real-time monitoring tools**
    - The system shall integrate monitoring software to track system performance and security metrics.
    - **R6.3.2: Track user activities**
    - The system shall log all user actions, especially those involving financial transactions.
    - **R6.3.3: Generate alerts for anomalies**
    - The system shall trigger alerts for any abnormal activities, such as multiple failed login attempts.

R7: Send notifications and reporting

- R7.1: Send real-time notifications
    - **R7.1.1: Configure notification preferences**
    - The system shall allow users to select which events trigger notifications (e.g., contributions, payouts).
    - **R7.1.2: Send notifications via SMS, email, or in-app**
    - The system shall deliver notifications through the user’s preferred channel(s).
    - **R7.1.3: Ensure timely delivery of notifications**
    - The system shall prioritize notification delivery to ensure users receive updates promptly.

- R7.2: Generate reports
    - **R7.2.1: Collect data for reporting**
    - The system shall aggregate data on contributions, payouts, and group activities for reporting purposes.
    - **R7.2.2: Create customizable report templates**
    - The system shall provide templates for common reports (e.g., monthly contribution summaries) that users can personalize.
    - **R7.2.3: Allow users to download reports**
    - The system shall enable users to export reports in formats like PDF or CSV for offline access.
- R7.3: Customize notification preferences
    - **R7.3.1: Provide options for notification types**
    - The system shall let users choose which types of events they want to be notified about (e.g., only payouts).
    - **R7.3.2: Allow selection of notification channels**
    - The system shall offer multiple channels (SMS, email, in-app) and let users select their preferred ones.
    - **R7.3.3: Save user preferences**
    - The system shall store each user’s notification settings and apply them consistently across the platform.

ARCHITECTURAL REQUIREMENTS

QUALITY REQUIREMENTS

QR1: Performance

- **QR1.1:** The system shall respond to user actions (e.g., registration, login, contribution setup) within 2 seconds under normal load (100 concurrent users).
- **QR1.2:** API endpoints (e.g., user registration, payment processing) shall have a maximum latency of 500ms for 95% of requests under peak load.
- **QR1.3:** Real-time notifications (in-app) shall be delivered within 5 seconds of a payment event trigger, using ActiveMQ for event-driven processing.

Ensures a responsive experience for users in rural areas with limited connectivity, aligning with the proposal's accessibility constraints.

QR2: Reliability

- **QR2.1**: The system shall achieve 99.9% uptime, excluding scheduled maintenance, to support critical financial transactions (mocked locally for Demo 1 using free-tier services like AWS Free Tier).
- **QR2.2**: The system shall handle transaction failures gracefully, with automatic retries (exponential backoff, up to 3 attempts) for failed debit orders and logged errors for audit trails.
- **QR2.3**: The system shall maintain data consistency across microservices (e.g., user and payment services), ensuring no duplicate transactions via idempotency keys.

Builds trust in automated financial operations, critical for stokvel members relying on timely contributions and payouts.

QR3: Scalability

- **QR3.1**: The system shall support up to 1,000 concurrent users during peak transaction times (e.g., month-end) without performance degradation, using a microservices architecture and load balancing.
- **QR3.2**: The database (PostgreSQL) shall handle 10,000 transactions per hour with optimized queries, leveraging ElasticSearch for analytics and Redis for caching frequent queries (e.g., tier status).
- **QR3.3**: The system shall auto-scale services (e.g., payment processing) to handle a 50% traffic spike within 5 minutes, mocked using local containerized setups for Demo 1.

Prepares the system for growth, as stokvels may scale to thousands of users, per the proposal's scalability constraint.

QR4: Security

- **QR4.1**: The system shall encrypt all personal and financial data (e.g., SA IDs, bank details) at rest (AES-256) and in transit (TLS 1.3), complying with POPIA and GDPR.
- **QR4.2**: The system shall implement multi-factor authentication (MFA) via Keycloak, requiring email or SMS verification for login and sensitive actions (e.g., debit order setup).

- **QR4.3**: The system shall detect and flag at least 90% of anomalous transactions (e.g., irregular payment patterns) using basic AI-driven fraud detection (TensorFlow, mocked).
- **QR4.4:** The system shall log all access attempts and transactions in PostgreSQL, retaining audit trails for 12 months to meet AML compliance.

Protects user data and prevents fraud, addressing the proposal's emphasis on security and regulatory compliance.

QR5: Maintainability

- **QR5.1**: The system shall achieve at least 70% code coverage through automated unit tests (Jest for frontend, JUnit for backend), tracked via Codecov.
- **QR5.2:** The system shall use a modular microservices architecture, allowing independent updates to services (e.g., user management, payments) with minimal downtime (less than 5 minutes per update).
- **QR5.3:** The system shall include comprehensive documentation (SRS, API specs, deployment guide) in the GitHub repository, updated incrementally per sprint.

Facilitates ongoing development and debugging, critical for a small team with limited resources.

QR6: Usability

- **QR6.1:** The system shall provide a simple, intuitive UI (React) with clear navigation, enabling non-tech-savvy users to complete core actions (e.g., registration, contribution setup) in fewer than 5 clicks.
- **QR6.2**: The system shall achieve a System Usability Scale (SUS) score of at least 80/100 in user acceptance testing (simulated for Demo 1 with team feedback).
- **QR6.3:** The system shall include step-by-step guidance for identity verification (e.g., ID upload, face recognition), with error messages in plain language (e.g., “Please enter a valid 13-digit SA ID”).

- **QR6.4:** The system shall be responsive, rendering correctly on screen sizes from 320px (low-end smartphones) to 1920px (desktops), tested on mid-range devices.

Ensures accessibility for rural and non-technical users, per the proposal's user verification and accessibility constraints.

QR7: Compliance

- **QR7.1:** The system shall adhere to POPIA and GDPR, allowing users to view, edit, or request deletion of their personal data via the dashboard within 24 hours of a request.
- **QR7.2:** The system shall implement user consent mechanisms for data processing during registration, with explicit opt-in for biometric verification.
- **QR7.3:** The system shall comply with AML regulations by logging all financial transactions and flagging suspicious activities for manual review.

Ensures legal compliance, critical for financial systems handling sensitive user data.

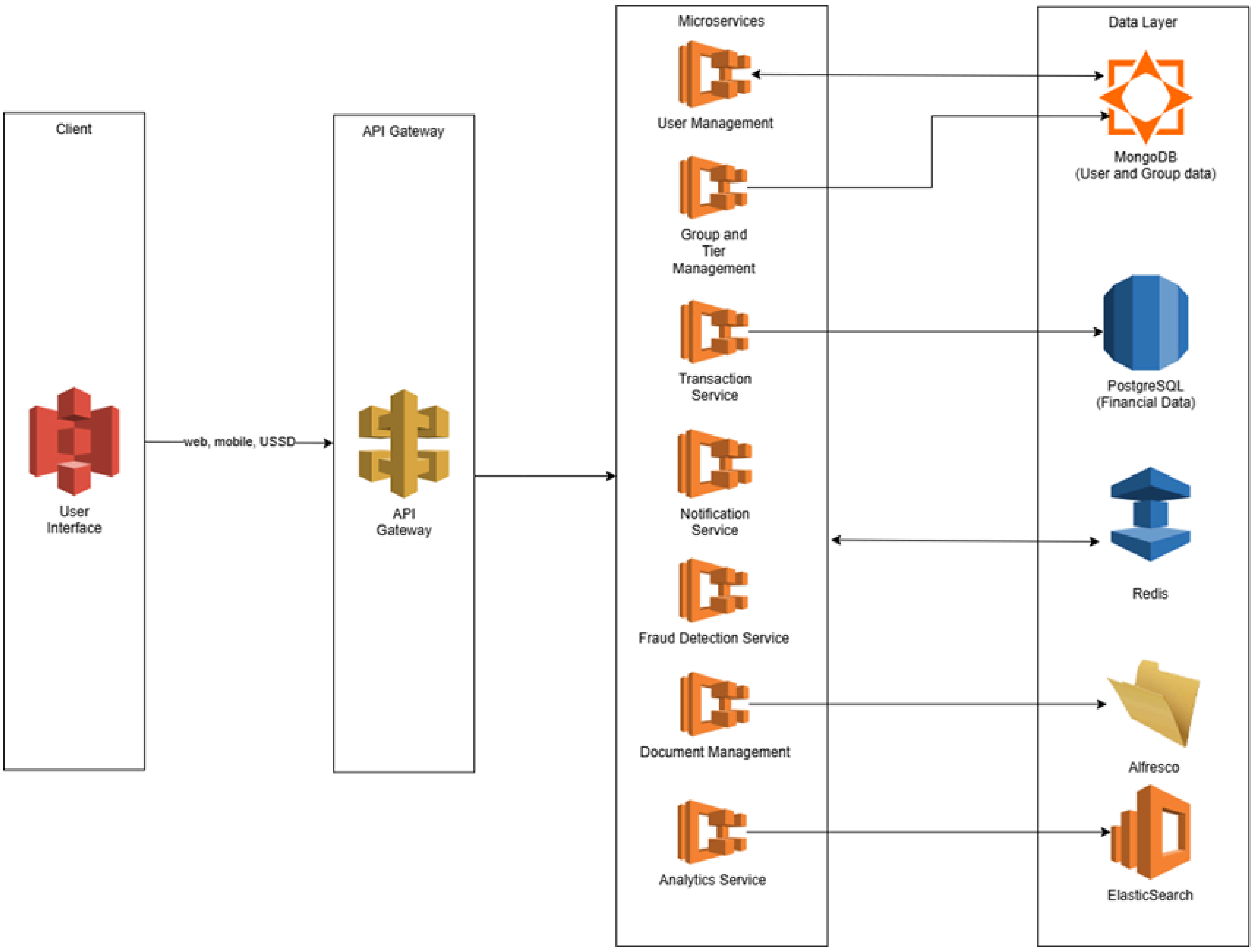
QR8: Availability

- **QR8.1**: The system shall recover from failures (e.g., service crashes) within 5 minutes, using failover mechanisms and redundant instances.
- **QR8.2:** The system shall provide monitoring via Uptime Robot, alerting the team to downtime exceeding 5 minutes.
- **QR8.3:** The system shall support rollback to a previous stable version within 10 minutes if deployment fails, documented in the deployment guide.

Guarantees continuous access to financial services, aligning with the proposal's 99.9% uptime requirement.

ARCHITECTURAL PATTERNS

We will implement a microservices architecture to ensure modularity and scalability, allowing independent scaling of services like payments and notifications, reducing costs by leveraging free cloud tiers initially (e.g., AWS Free Tier). The following architecture overview diagram (see next page) uses AWS symbols, however the actual technologies and frameworks used may vary and include in house infrastructure.



SERVICE CONTRACTS

User Registration

Service Contract Name: registerUser

Parameters: { firstName: string, lastName: string, email: string, phone: string, password: string, nationalId: string }

Pre-conditions:

- The user must not have an existing account
- All information must be provided and validated
- They must provide a valid and verifiable South African ID

Post-conditions:

- Verification process initiated via Keycloack

Actors: User

Scenario: The user accesses the registration page, enters their details, and submits the form. The verification process is initiated.

User Login

Service Contract Name: loginUser

Parameters: { email: string, password: string }

Pre-conditions:

- The user must have an existing account
- Credentials must be provided and correct

Post-conditions:

- Multi-factor authentication initiated
- User session begins
- User is navigated to dashboard

Actors: User

Scenario: The user accesses the login page, enters their details, the system verifies them and establishes a session.

Create Stokvel Group

Service Contract Name: createStokvelGroup

Parameters: { name: string, description: string, creatorId: string, private: boolean }

Pre-conditions:

- The user must have an existing account
- The user must have a good enough financial tier

Post-conditions:

- Contribution amount and frequency is set.
- New group created with rules and parameters.
- Creator of the group is set as admin.

Actors: Verified User

Scenario: The user accesses the stokvels page and creates a new stokvel group with specified parameters.

Join Stokvel Group

Service Contract Name: joinStokvelGroup

Parameters: { userId: string, groupId: string }

Pre-conditions:

- The user must have a good enough financial tier to meet the group requirements.
- Group must have space for a new user.

Post-conditions:

- Group admin is notified.
- If admin approves, the user is added to list of group members.

Actors: Verified User

Scenario: The user accesses the stokvels page, requests to join a group.

TECHNOLOGY REQUIREMENTS

- **Frontend:** React.js with Tailwind CSS for a responsive, intuitive UI accessible on web and mobile devices, can be catered to users on mid-range smartphones and low-bandwidth networks, critical for rural accessibility.
- **Backend:** Node.js with Express.js for a fast, scalable server-side application.
- **Database**: PostgreSQL for structured financial data (e.g., transactions) and MongoDB for flexible user and group data.
- Authentication: Keycloak for secure identity management with OAuth2 and MFA support.
- **Messaging:** ActiveMQ for asynchronous event-driven processing (e.g., payments, notifications).
- Document Storage: Alfresco for secure management of verification documents.
- **API**: RESTful API for integration with banks, payment gateways, and possibly blockchain services.
- Search and Analytics: ElasticSearch enables fast querying and analytics for real-time reporting
- **AI Frameworks:** TensorFlow powers fraud detection and credit scoring with robust machine learning capabilities.
- **Hosting:** AWS Free Tier, scalable and cost-effective for a student project, with security features for financial data.
- **Version Control:** GitHub, essential for collaborative development and code management.
- **CI/CD:** GitHub Actions, automates testing and deployment, ensuring quality and efficiency along with aiding the dev-ops process
- **Testing**: Jest and Cypress, provides comprehensive unit, integration and end-to-end testing to maintain reliability.