ALLAN SHARAD | 665782

IQBAL SHARIFF | 665356

MyChama

Connecting Savings Groups to Smart Financial Tools

School of Science and Technology, United States International

University-Africa

APT3065 Mid-Term Project

System Design

Table of Contents

Table of Figures

1. Introduction

1.1 Purpose of the Document

This System Design Document (SDD) provides a comprehensive overview of the technical architecture, system components, data flows, and interface designs for the MyChama platform. It serves as a bridge between the requirements gathered during the planning phase and the actual implementation. The document outlines the proposed solution's structure, behaviors, and interactions to ensure a shared understanding among team members and stakeholders. Its primary goal is to guide developers, testers, and system architects during development and maintenance of the application.

1.2 Scope of the System

MyChama is a web and mobile application designed to help informal savings groups (commonly referred to as chamas in Kenya) manage their financial and organizational activities. The system will allow members to register, contribute funds, request and approve loans, track repayment schedules, participate in meetings, vote on group decisions, and communicate efficiently through announcements and polls. The system will automate manual processes such as recordkeeping, contribution tracking, and meeting coordination, ensuring transparency, accountability, and ease of use for all group members regardless of technical expertise.

1.3 Intended Audience

This document is intended for all project stakeholders who are involved in the design, development, deployment, and validation of the MyChama system. This includes but is not limited to software developers, UI/UX designers, project managers, testers, product owners, and financial or technical stakeholders. Each section is tailored to address different roles, offering both technical depth and conceptual clarity to support informed decision-making and collaborative development.

1.4 Glossary

- **Chama** – A Swahili term referring to a self-organized savings or investment group.

- **Loan Repayment** – The act of paying back borrowed funds according to agreed terms.

- **Contribution** – A periodic amount paid by a chama member into the group fund.

- **Poll** – A voting mechanism used within the system to make group decisions.

- **API** – Application Programming Interface; allows different parts of the system to communicate.

- **M-Pesa** – A mobile money transfer service widely used in Kenya for digital payments.

2. System Overview and Design Philosophy

2.1 High-Level Description

The MyChama system is designed as a cross-platform application that enables savings groups (chamas) to digitize their financial and administrative operations. It is built on a modular client-server architecture that separates the frontend, backend, and service layers for maintainability and scalability.

The frontend will be developed using bootstrap 5, offering a responsive and consistent user experience across devices. The backend is built with Node.js and Express.js, providing RESTful APIs to handle authentication, user roles, contributions, loans, announcements, and meetings. Data is stored in MongoDB, a flexible NoSQL database suited to evolving schemas. Notifications and real-time updates are managed via Firebase Cloud Messaging (FCM) or OneSignal, and mobile payments are integrated using M-Pesa APIs. The entire system is deployed on a cloud platform (e.g., Render, Railway, or AWS), with CI/CD pipelines managed via GitHub Actions for automated testing and deployment.

This high-level architecture ensures that the system is loosely coupled, scalable, and able to adapt to growing user needs.

2.2 Design Principles
i. Modularity

The system is divided into well-defined modules, each responsible for a specific domain (e.g., finance, communication, meetings). This allows for parallel development, easier maintenance, and reuse of components.

ii. Reusability

UI components, backend services, and business logic are designed to be reusable across different parts of the application. Shared libraries and hooks are implemented to reduce duplication and improve consistency.

iii. Scalability

The system supports horizontal scaling through stateless APIs and microservice-ready structures. MongoDB's document-based model allows the application to handle high volumes of transactions and data growth over time without performance degradation.

iv. Performance

Caching strategies, asynchronous operations, and efficient API routing ensure fast response times and minimal load on the server. The use of lazy loading and efficient state management in the frontend further optimizes user experience.

v. Security

User authentication is secured using JWT (JSON Web Tokens) with role-based access control (RBAC) enforced throughout the API. Sensitive information such as payment data and credentials

are stored securely and environment variables are used to manage secrets. GitHub branch protection rules, secret scanning, and 2FA are enforced to secure the codebase.

Together, these principles ensure that MyChama is reliable, maintainable, secure, and capable of supporting the digital transformation of informal savings groups in Kenya and beyond.

3. Architectural Design

3.1 Architecture Diagram

The architecture of MyChama is based on a modular client-server model, designed to support future microservices expansion within the backend. The following component diagram provides a high-level view of the system's major functional units and their interactions.
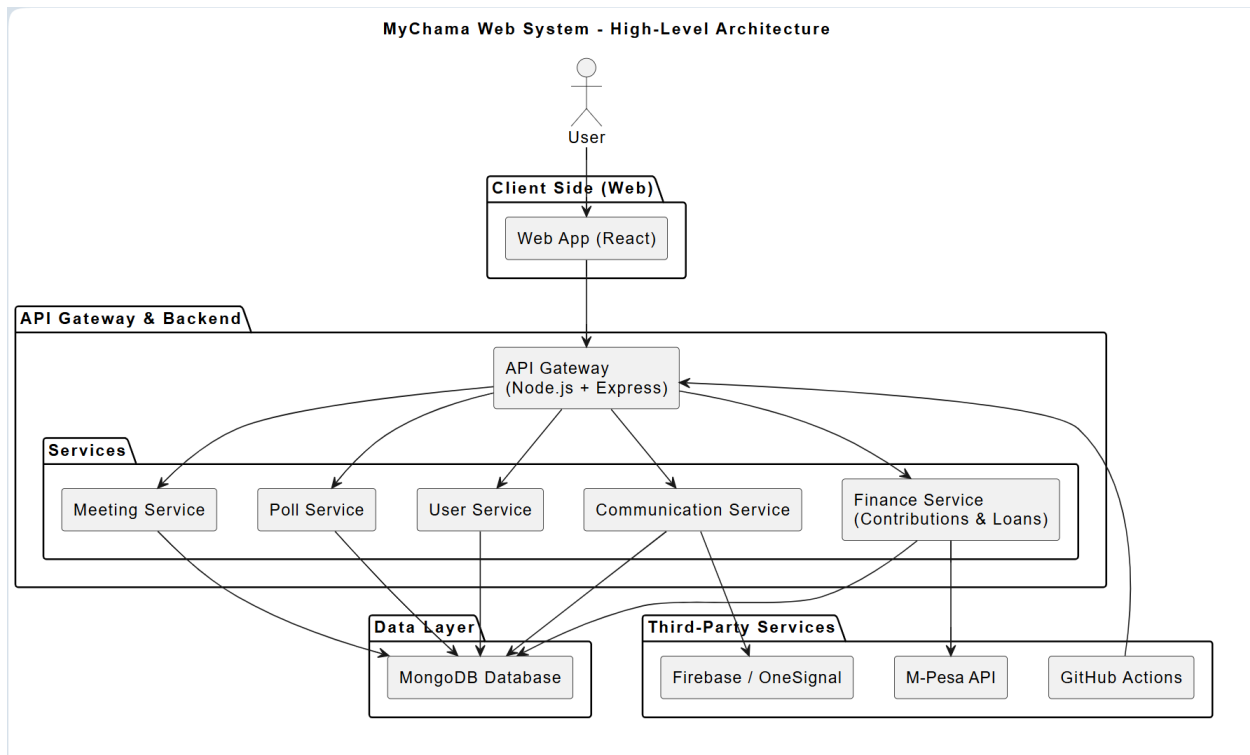

Figure 1: Component Diagram

The component diagram above illustrates the high-level architectural structure of the MyChama web application. It is based on a modular, layered architecture using a client-server model. The

system is composed of a react-based web frontend, which communicates with a centralized API Gateway built using Node.js and Express. This API gateway routes requests to specialized backend services (components) including:

- User Service – manages authentication, roles, and user data.

- Finance Service – handles contributions, loan requests, approvals, and repayments.

- Meeting Service – schedules group meetings and manages attendance.

- Poll Service – enables member voting and decision-making through polls.

- Communication Service – supports announcements and push notifications.

All services interact with a shared MongoDB database, which stores structured records for users, financial transactions, meetings, and messages. The system integrates with third-party services such as M-Pesa API for mobile payments, and Firebase/OneSignal for sending real-time notifications. Deployment automation and CI/CD pipelines are managed using GitHub Actions. This architecture ensures a clean separation of concerns, promotes scalability, and supports secure and efficient web-only operation of the MyChama platform.

3.2 Architecture Style

MyChama adopts a layered architecture built on a client-server model, promoting separation of concerns and modular development. At the presentation layer, the system features a responsive web interface developed using React, allowing users to interact with various system features through a modern and intuitive UI. The application layer consists of a centralized API gateway built with Node.js and Express, responsible for routing client requests and enforcing authentication and authorization. The service layer is composed of modular backend services that encapsulate the core business logic, including user management, finance operations (contributions and loans), meetings, communication, and polling. These services interact with the data layer, which uses

MongoDB as the primary database for storing user data, financial records, meeting schedules, and communication logs. Lastly, the external integration layer connects the platform to third-party services such as M-Pesa for mobile payments and Firebase/OneSignal for delivering real-time notifications. This structured architectural style ensures that the system remains scalable, maintainable, and adaptable to future enhancements or distributed deployments.

3.3 Component Description

i. Frontend

- Built with React web.

- Handles user interactions, form submissions, and data visualization (e.g., graphs, dashboards).

- Communicates with the backend via HTTPS and JWT-based authentication.

ii. API Gateway (Node.js + Express)

- Acts as the single-entry point for all API calls.

- Routes requests to the appropriate microservice.

- Manages authentication tokens, request validation, and role-based access control.

iii. User Service

- Handles user registration, login, profile updates, and role assignments (e.g., Chairperson, Member).

- Integrates with authentication logic (JWT issuance and verification).

iv. Finance Service

- Manages contributions, loan requests, approvals, and repayments.

- Tracks transaction history and emits notifications upon events.

- Integrates with M-Pesa APIs for payment processing.

v. Communication Service

- Enables group announcements and direct member notifications.

- Connects to Firebase Cloud Messaging (FCM) or OneSignal to deliver push notifications.

vi. Meeting Service

- Manages scheduling of meetings and stores agendas, minutes, and attendance.

- Sends calendar invites and reminders through the notification service.

vii. Poll Service

- Allows group members to create and participate in votes on group matters.

- Supports anonymous and timed polls.

viii. Database (MongoDB)

- NoSQL database storing user records, contributions, loan data, meetings, polls, and messages.

- Uses schema validation and indexing to optimize query performance.

ix. Third-Party Integrations

- M-Pesa: For sending and receiving mobile money payments.

- Firebase/OneSignal: For real-time push notifications.

- GitHub Actions: For CI/CD automation and secret management.

3.4 Interaction Flow Summary

When a user accesses the MyChama platform, they begin by logging in through the web frontend interface. After submitting login credentials, the frontend sends an authenticated request to the API Gateway. The gateway is responsible for validating the user's token and determining the

appropriate backend service to handle the request. For example, financial actions are routed to the Finance Service, while user updates go to the User Service.

Once the request reaches the designated backend service, the business logic is executed. This may involve reading from or writing to the MongoDB database, performing calculations, or validating input data. If additional communication is needed, such as confirming a contribution or loan approval, the service may trigger the Notification Service to send real-time updates using Firebase Cloud Messaging or OneSignal.

After the backend processes the request and responds, the frontend updates the user interface based on the result. This could include displaying updated dashboards, status messages, or confirmation alerts. This asynchronous and service-driven interaction model ensures a responsive user experience while maintaining system modularity and scalability.

4. Detailed Design

This section provides a breakdown of the main modules of the MyChama web application. Each module description outlines its responsibilities, data flow, rules, and interactions with other components of the system. These modules map directly to the backend services defined in the architecture.

4.1 Module Descriptions

4.1.1 User Management Module
Functionality

Handles user registration, login, authentication, profile updates, and role assignment (e.g., Member, Chairperson, Treasurer).

Inputs

- o   Registration data (name, email/phone, password)

- o   Login credentials

- o   Profile updates

- o   Role selection or assignment

Outputs

- o   JWT authentication token

- o   Profile information

- o   Success/error messages

Business Rules

- o   Each user must register with a unique phone number or email.

- o   Only admins can assign roles beyond "Member."

- o   All data must be validated (e.g., strong password, required fields).

Data Elements Involved

- o   user_id, full_name, email, phone_number, password_hash, role, date_joined

4.1.2. Contributions Module

Functionality

Allows members to view, make, and track financial contributions to the chama fund.

Inputs

- o   Contribution amount

- o   Payment method (e.g., M-Pesa)

- o   Chama ID

- o   User ID

Outputs

- o   Confirmation message

- o Contribution history

- o Updated total balance

Business Rules

- o Contributions can only be made by registered members.

- o The system must verify successful payment before recording the transaction.

- o Duplicate contributions within a defined timeframe must be flagged.

Data Elements Involved

- o contribution_id, user_id, chama_id, amount, payment_method, contribution_date, status

4.1.3. Loan Management Module
Functionality

Enables members to request loans, while chairpersons and treasurers review, approve, and track loan status and repayments.

Inputs

- o Loan amount

- o Purpose

- o User ID

- o Loan status updates (e.g., approved, rejected)

Outputs

- o Approval/rejection status

- o Repayment schedule

- o Notifications to stakeholders

Business Rules

- A member must meet eligibility requirements (e.g., consistent contributor, no outstanding loans).

- Loan approval requires both Chairperson and Treasurer confirmation.

- Interest and repayment rules must be configurable.

Data Elements Involved

- loan_id, user_id, amount, status, approved_by, repayment_schedule, issue_date

4.1.4 Meeting Management Module
Functionality

Allows group leaders to schedule meetings, track attendance, and manage agendas.

Inputs

- Meeting title

- Date and time

- Agenda notes

- Participant list

Outputs

- Calendar view

- Reminders

- Meeting records

Business Rules

- Only authorized roles (Chairperson/Secretary) can schedule meetings.

- Users must be notified at least 24 hours in advance.

- Attendance tracking is optional but supported.

Data Elements Involved

  o meeting_id, chama_id, date, time, agenda, organizer_id, attendance_status

4.1.5 Communication & Notification Module

Functionality

Facilitates group-wide announcements, reminders, and real-time alerts using Firebase or OneSignal.

Inputs

  o Message content

  o Target recipients

  o Message type (announcement, alert, reminder)

Outputs

  o Notification messages

  o Delivery status

Business Rules

  o Only Chairperson and Treasurer can post announcements.

  o Reminders are sent based on scheduled meeting or loan events.

  o Notifications must not reveal sensitive financial data.

Data Elements Involved

  o message_id, sender_id, message_type, timestamp, delivery_status

4.1.6 Polling Module

Functionality

Allows group members to vote on decisions through time-limited polls.

Inputs

  o Poll question

- List of options

- Voting deadline

Outputs

- Aggregated vote results

- Poll status

Business Rules

- One vote per member per poll

- Polls close automatically at the specified deadline

- Poll creators cannot vote after publishing

Data Elements Involved

- poll_id, creator_id, question, options[], start_time, end_time, votes[]

4.2 Interface Design

4.2.1 User Interface
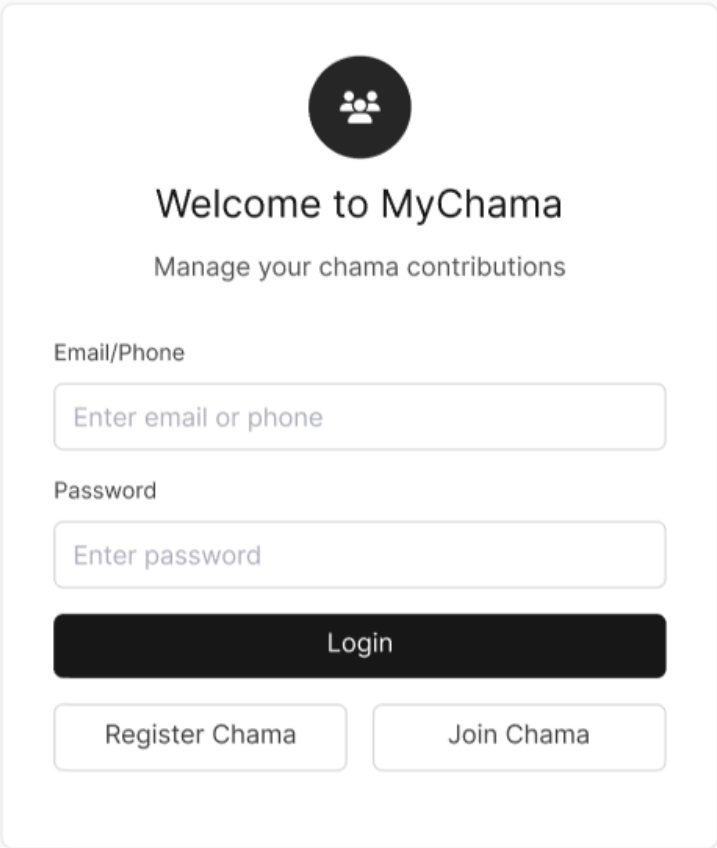
The user interface (UI) of MyChama has been designed using Figma, a collaborative design tool that enables high-fidelity wireframing and prototyping. The Figma UI focuses on delivering a clean, intuitive, and responsive experience optimized for web use. The design incorporates minimalistic layouts, Swahili-English hybrid labels, and consistent visual hierarchies for both desktop and mobile browser views.

The primary screens designed in Figma include:

- Login and registration pages with authentication input fields and role selection.

- A dashboard that summarizes key metrics such as total savings, active loans, and upcoming meetings.

- Contribution and loan interface that allow users to input amounts, view history, and submit requests.

- A Meetings calendar where users can view, schedule, and receive reminders for upcoming events.

- Communication modules such as announcements, group chat, and polls for collective decision-making.

- A reports page that presents contribution summaries and loan trends using visual charts and downloadable files.

Each Figma screen is structured using a modular grid system and reusable components to maintain visual consistency across the application. The designs also account for accessibility, with clear buttons, readable fonts, and logical navigation flows.

MyChama

- **Dashboard**
- Contributions
- Loans
- Meetings
- Chat
- Reports

Dashboard

Welcome back to Umoja Chama

Total Savings	Outstanding Loans	Next Meeting
KSh 45,000	KSh 8,500	Jan 15
+12% from last month	Due in 15 days	2:00 PM at Community Hall

[+ Contribute] [👤 Request Loan] [📅 Schedule Meeting] [⬇ Download Report]

Recent Contributions

Mary Wanjiku Jan 10, 2025	KSh 2,000
John Kamau Jan 9, 2025	KSh 1,500
Grace Muthoni Jan 8, 2025	KSh 2,500

Upcoming Events

- **Monthly Meeting**
 Jan 15, 2025 at 2:00 PM
 Community Hall, Kiambu
- **Loan Committee Review**
 Jan 20, 2025 at 10:00 AM
 Online Meeting
- **Training Workshop**
 Jan 25, 2025 at 9:00 AM
 Financial Literacy

Michango - Contributions

Track member contributions and payments

[+ Make Payment]

Member Contributions

Member	This Month	Total	Status
Mary Wanjiku	KSh 5,000	KSh 35,000	Paid
John Kamau	KSh 3,000	KSh 28,000	Pending

Mikopo - Loans

Request and manage loans

Request Loan

Amount (KSh)

[Enter amount]

Reason

[Describe purpose of loan]

[Submit Request]

Loan Status

1. **Application Submitted**
 Jan 10, 2025
2. **Under Review**
 Pending
3. **Decision**
 Pending

Mikutano - Meetings

Schedule and track meetings

[+ Create Meeting]

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18

Meeting Attendance

Grace Njeri ☐

Peter Mwangi ☐

Mazungumzo - Group Chat
Communicate with your chama members

📌 Pinned: Next meeting Jan 15, 2025
Please confirm attendance

Sarah Wanjiru
Nimeshalipa contribution yangu ya January
2 hours ago

Poll: Best day for next meeting?
○ Saturday (5 votes)
○ Sunday
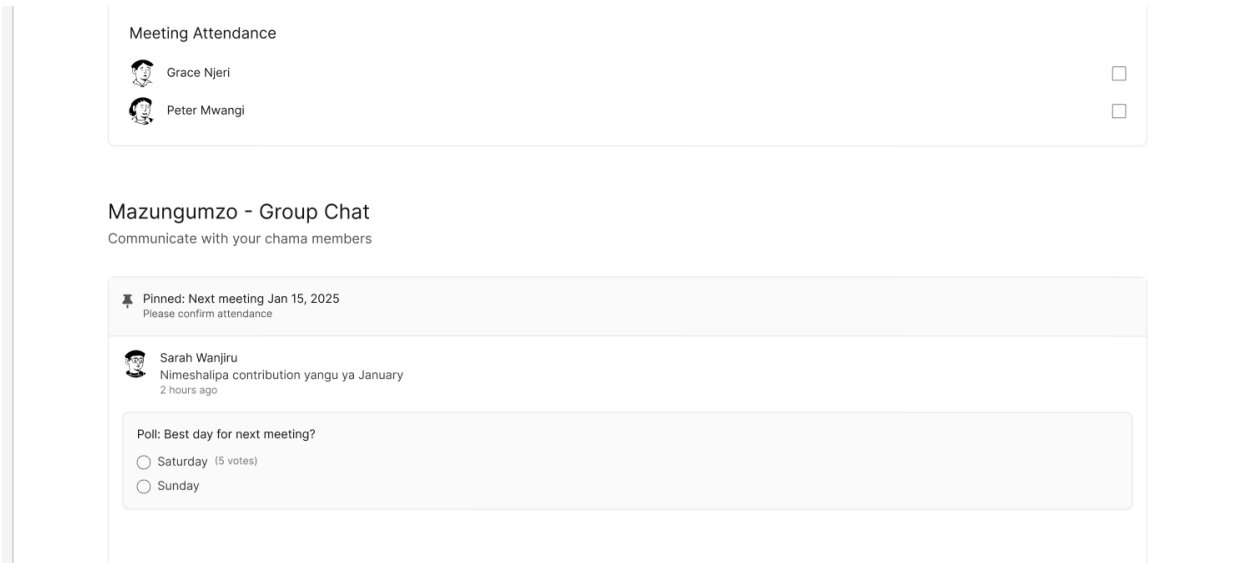
4.2.2 Navigation Flow

1. Login / Registration Page

→ Upon successful login, redirect to Dashboard.

2. Dashboard

→ Access to:

▪ My Contributions

▪ Loan Requests

▪ Meetings & Events

▪ Announcements / Polls

▪ Reports

▪ Settings / Logout

3. Contributions

→ View history → Click "Contribute" → Enter amount → Submit.

4. Loans

→ View loan status → Click "Request Loan" → Submit request → Wait for approval.

5. Meetings

→ View calendar → Click meeting for details → RSVP or add notes.

6. Announcements / Polls

→ View list → Read / vote → View results or post new if authorized.

4.2.3 API Interface Specifications

Below are example API endpoints used in the MyChama system:

i. User Login

Endpoint URL: POST /api/auth/login

HTTP method: POST

Request body:

json

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Response:

json

```
{
  "token": "jwt-token-string",
  "user": {
    "id": "u123",
    "name": "Jane Doe",
    "role": "Member"
  }
}
```

Status codes

- 200 OK – Login successful

- 401 Unauthorized – Invalid credentials

ii. Make Contribution

Endpoint URL: POST /api/contributions

HTTP method: POST

Request body:

json

```
{
  "userId": "u123",
  "amount": 500,
  "paymentMethod": "M-Pesa"
}
```

Response:

json

```
{
  "message": "Contribution recorded successfully",
  "transactionId": "c987"
}
```

Status Codes:

- 201 Created
- 400 Bad Request – Missing fields

iii. Request Loan

Endpoint URL: POST /api/loans/request

HTTP method: POST

Request body:

json

```json
{
  "userId": "u123",
  "amount": 2000,
  "purpose": "Emergency fund"
}
```

Response:

json

```json
{
  "status": "pending",
  "loanId": "l789",
  "message": "Loan request submitted for review"
}
```

Status Codes:

- 202 Accepted
- 403 Forbidden – Not eligible

iv. Get Upcoming Meetings

Endpoint URL: GET /api/meetings/upcoming

HTTP Method: GET

Response:

json

```json
[
  {
    "meetingId": "m001",
    "title": "Monthly Planning",
    "date": "2025-08-10T14:00:00Z",
    "location": "Chama Hall"
```

```
  }
]
```

Status Codes:

- 200 OK
- 404 Not Found – No meetings scheduled

5. Database Design

5.1 Entity-Relationship Diagram (ERD)

The MyChama system uses MongoDB, a NoSQL database, where each entity is modeled as a collection. Relationships are represented through references (ObjectIds) or embedded subdocuments depending on access patterns and normalization.
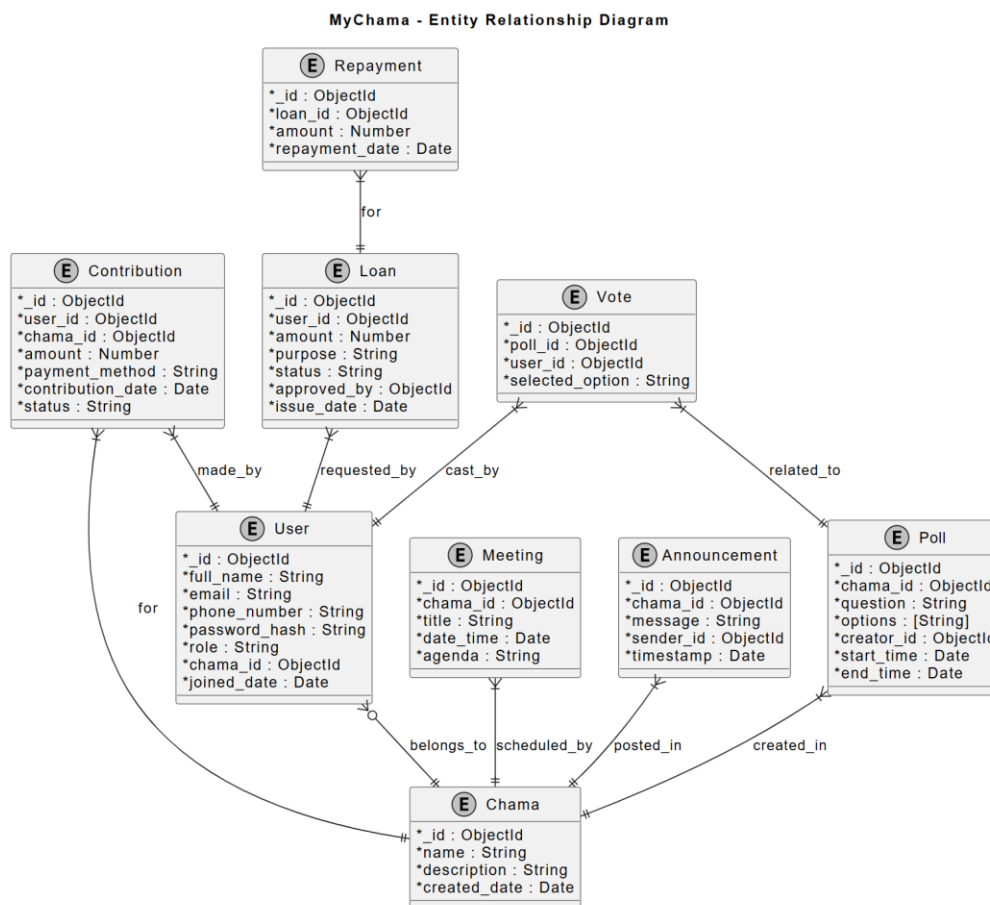

Figure 2: ERD

Figure 2 illustrates the entity relationship diagram (ERD) for the MyChama system, outlining the logical data model and the relationships among key entities. At the center of the system is the user entity, which represents registered members and administrators of a chama group. Each user is associated with a single chama, while a chama can have multiple users, forming a one-to-many relationship.

The contribution entity captures all financial contributions made by users to their respective chama. A user can have many contributions, and each contribution record includes details such as amount, payment method, and date. The loan entity allows users to request loans, which are linked to both the user who requested the loan and the group it belongs to. Each loan may have multiple repayment records tracking periodic payments over time.

The meeting entity stores scheduled events for each chama, including date, agenda, and participants. The announcement entity allows authorized users to broadcast messages to all group members, while the poll entity enables voting on key decisions. Each poll stores options, timestamps, and votes cast by individual users.

This ERD reflects a normalized data model, supports efficient data retrieval and relationship integrity, and serves as the foundation for structuring MongoDB collections in the implementation phase.

5.2 Data Dictionary (MongoDB Collections)
Below is a data dictionary showing key collections and their fields:

Collection: users

Field Name	Type	Nullable	Description
_id	ObjectId	No	Unique identifier

full_name	String	No	Full name of the user
email	String	No	Email address
phone_number	String	No	Unique phone number
password_hash	String	No	Hashed password
role	String	No	User role (Member, Chairperson, etc.)
chama_id	ObjectId	Yes	Reference to chama
joined_date	Date	No	Date the user joined

Collection: contributions

Field Name	Type	Nullable	Description
_id	ObjectId	No	Unique identifier
user_id	ObjectId	No	ID of the contributing user
chama_id	ObjectId	No	Associated chama group
amount	Number	No	Amount contributed
payment_method	String	No	Payment mode (e.g., M-Pesa)
contribution_date	Date	No	Date of contribution
status	String	No	Status (e.g., confirmed, pending)

Collection: loans

Field Name	Type	Nullable	Description
_id	ObjectId	No	Unique loan ID
user_id	ObjectId	No	Requesting member
amount	Number	No	Loan amount

purpose	String	Yes	Reason for loan
status	String	No	pending, approved, or rejected
approved_by	ObjectId	Yes	ID of approver (Chairperson)
issue_date	Date	Yes	Date the loan was issued
repayment_schedule	Object	Yes	Embedded repayment details

Collection: meetings

Field Name	Type	Nullable	Description
_id	ObjectId	No	Meeting ID
chama_id	ObjectId	No	Related chama
title	String	No	Meeting topic
date_time	Date	No	Scheduled date and time
agenda	String	Yes	Meeting agenda or notes
attendees	[ObjectId]	Yes	List of members who attended

Collection: announcements

Field Name	Type	Nullable	Description
_id	ObjectId	No	Announcement ID
chama_id	ObjectId	No	Related chama
message	String	No	Announcement content
sender_id	ObjectId	No	Who sent it
timestamp	Date	No	When it was sent

Collection: polls

Field Name	Type	Nullable	Description
_id	ObjectId	No	Poll ID
chama_id	ObjectId	No	Related chama
question	String	No	Poll question
options	[String]	No	Answer options
creator_id	ObjectId	No	Member who created the poll
start_time	Date	No	Start date
end_time	Date	No	End date
votes	[Object]	Yes	List of vote records

5.3 Normalization Level

The database design for the MyChama system adheres to the principles of Third Normal Form (3NF) to ensure data integrity, eliminate redundancy, and promote efficient storage. Each logical entity such as user, chama, contribution, loan, and meeting; is represented as a separate collection, with relationships maintained through references (ObjectIds). Attributes within each collection depend solely on the primary identifier and are free from transitive dependencies. In specific cases where performance optimization is necessary (such as embedding repayment records within loan documents or votes within polls), controlled denormalization is applied, in line with best practices for NoSQL document-based databases like MongoDB. This hybrid approach balances normalization with system efficiency, scalability, and real-time access patterns.

6. Data Flow and Control Flow

6.1 Level 1 Data Flow Diagram (DFD)

The Level 1 Data Flow Diagram provides a high-level view of how data moves through the MyChama system. It highlights the major processes involved, the external actors interacting with the system, and the core data stores. This DFD follows the Yourdon and DeMarco notation and illustrates how users, chairpersons, and treasurers engage with various functional modules such as contributions, loans, meetings, and polls.
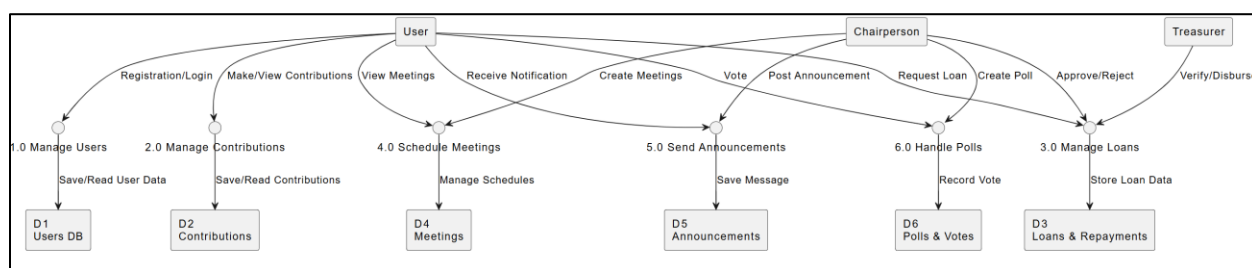

Figure 3: Level 1 DFD

Figure 3 illustrates the Level 1 Data Flow Diagram (DFD) for the MyChama system. This diagram presents a high-level overview of how data flows between the system's primary actors (users, chairpersons, and treasurers), core functional processes, and data stores.

The diagram follows the Yourdon and DeMarco notation, where circles represent major processes such as user management, contributions, loans, meetings, announcements, and polls. Arrows indicate the direction of data flow between external entities, system processes, and internal data repositories. Data stores include user records, financial transactions, loan and repayment logs, scheduled meetings, announcements, and poll results.

This DFD helps visualize how MyChama handles key user interactions and system operations, ensuring all input and output data paths are logically and functionally defined in the system's architecture. It serves as a bridge between the requirements and implementation stages by mapping real-world actions to system processes and information flows.

6.2 Control Flow Diagram

The Control Flow Diagram outlines the logical sequence of operations within the MyChama system. It captures decision-making paths based on user roles and system conditions, particularly during login and post-authentication access. This diagram is useful for visualizing how control is passed between system components and user-specific modules, depending on authentication and role-based access control.
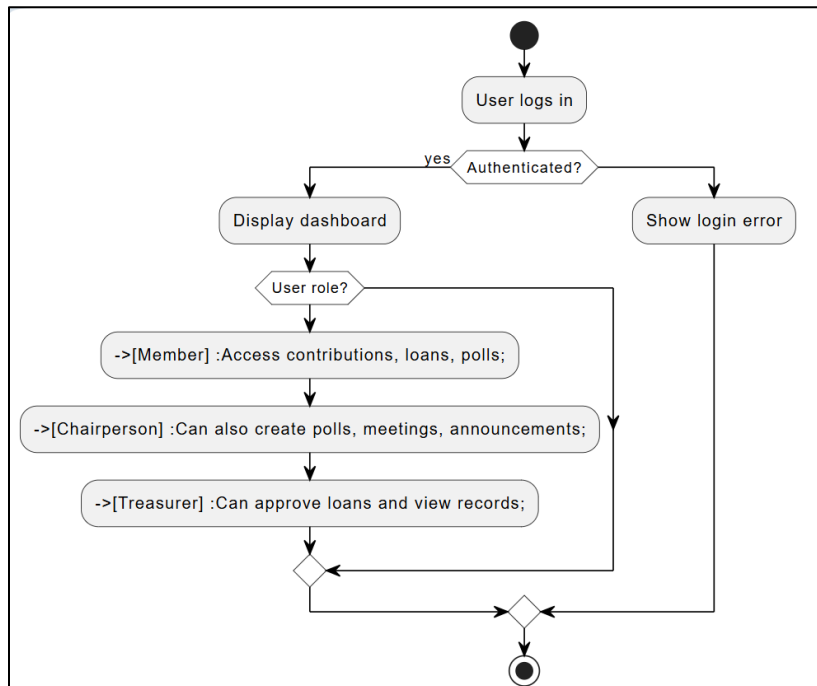

Figure 4: Control Flow Diagram

Figure 4 presents the Control Flow Diagram for the MyChama system. It captures the sequence of operations that occur after a user accesses the platform, with a focus on authentication and role-based access. The diagram illustrates how the system determines user permissions and routes the user to the appropriate features based on their role—whether they are a regular member, chairperson, or treasurer.

The flow begins with user login and includes conditional paths that define system behavior for both successful and unsuccessful authentication attempts. Once authenticated, users are granted

access to features relevant to their role. Members can view and submit contributions or request loans, while chairpersons can also manage polls, meetings, and announcements. Treasurers are given access to loan approvals and financial record management.

This diagram is critical for understanding how control logic is enforced across the system and how user navigation is dynamically adjusted based on access rights. It provides developers and testers with a clear overview of the conditional flow within the application.

6.3 State Diagram – Loan Lifecycle

The Loan State Diagram represents the various stages a loan object undergoes in the system, from the time it is requested by a member to its final state either repaid or rejected. This diagram helps model dynamic behavior and conditional transitions based on actions performed by the chairperson and treasurer, and reflects business logic embedded in the loan approval and repayment processes.
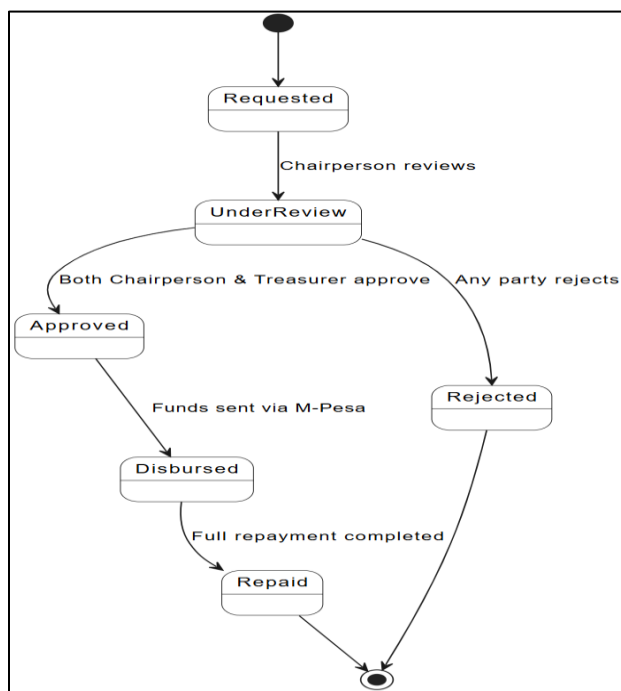

Figure 5:State Diagram - Loan Lifecycle

Figure 5 illustrates the state diagram representing the lifecycle of a loan in the MyChama system. This diagram captures the key states a loan passes through, beginning from the moment it is

requested by a member, through its review and approval process, to final states such as repayment or rejection.

The loan lifecycle starts in the requested state. It then transitions to under review, where both the chairperson and treasurer evaluate the request. Based on their actions, the loan either moves to the approved state or is rejected. Once approved, it proceeds to the disbursed state, indicating that funds have been successfully transferred, typically through M-Pesa integration. Finally, the loan enters the repaid state after full settlement of the outstanding amount.

This state diagram helps to clearly define system behavior and transitions related to loan processing, ensuring that appropriate validations and business rules are enforced at each stage. It also supports the development of automated workflows and status-based logic in the backend.

6.4 State Diagram – Poll Object

The Poll State Diagram illustrates the life cycle of a poll created within a chama. It defines how a poll progresses from creation to being open for voting and then transitions to a closed state upon expiration or manual intervention. This visual helps clarify time-based state transitions and supports the implementation of polling logic in the communication module.
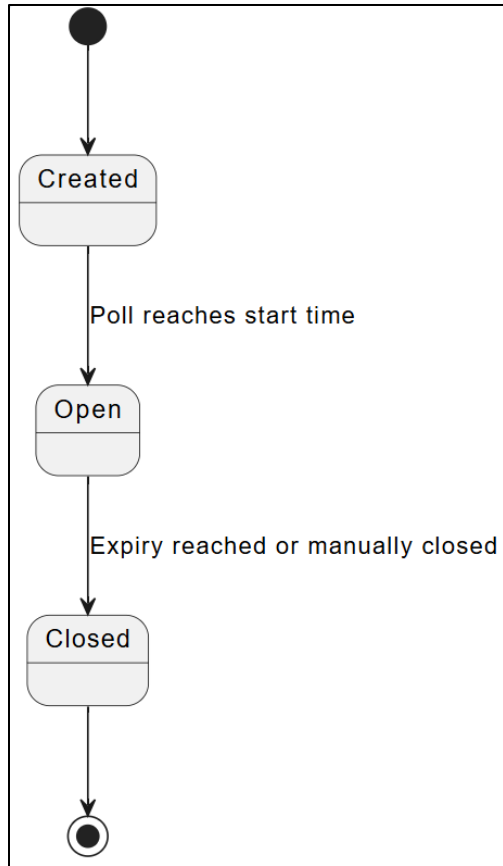
Figure 6: State Diagram – Poll Object

Figure 6 illustrates the state diagram for a poll object within the MyChama system. This diagram outlines the different states a poll transitions through, from its creation to closure. The poll lifecycle ensures structured participation from members during decision-making processes within a chama group.

A poll begins in the created state, where it is defined by the chairperson or an authorized user but not yet active. When the specified start time is reached, the poll automatically transitions to the open state, during which members can cast their votes. After the deadline passes, or if the poll is closed manually by the creator or system admin, it enters the closed state. Once closed, the poll is locked from further edits or voting, and the final results can be viewed by participants.

This state diagram is essential for modeling time-based and event-driven logic related to polls. It ensures that voting is conducted in a transparent and orderly manner, with clear enforcement of deadlines and access controls throughout the poll's lifecycle.

7. Non-Functional Design Considerations

The design of the MyChama system takes into account several critical non-functional aspects to ensure that the platform is secure, performant, scalable, reliable, and user-friendly. These considerations are essential to supporting real-world usage at scale and maintaining user trust and satisfaction.

7.1 Performance Optimization

The system is designed with performance in mind across both frontend and backend layers. On the frontend, lazy loading is used to defer loading of non-critical components, improving page load speed and perceived performance. On the backend, MongoDB indexes are created on frequently queried fields such as user_id, chama_id, and status in contributions and loans collections to accelerate query execution. Additionally, caching mechanisms such as in-memory caching (e.g., Redis) are planned for future integration to reduce repeated database calls for frequently accessed data like user profiles or dashboard summaries.

7.2 Security Design

Security is enforced at multiple layers of the MyChama system. All user authentication is handled using JWT (JSON Web Tokens), which securely encode user identity and roles. Passwords are stored as cryptographic hashes using modern hashing algorithms like bcrypt. The system communicates over HTTPS to ensure encrypted data transmission. For sensitive operations such as financial transactions, role-based access control (RBAC) is implemented to restrict functionality based on user privileges. Furthermore, OAuth2 integration is planned for future compatibility with

third-party identity providers, and all credentials and secrets are managed securely using environment variables and GitHub Secrets.

7.3 Scalability Plans

MyChama is built to support both horizontal and vertical scaling. Backend services are stateless and container-friendly, enabling deployment in distributed environments using Docker or Kubernetes. The system is intended to be hosted on cloud platforms such as Render, Railway, or AWS, which support autoscaling and global deployment. MongoDB is deployed using a cloud-managed service (e.g., MongoDB Atlas) that supports sharding and replication, allowing the database to grow with system demand without degrading performance.

7.4 Availability and Fault Tolerance

To ensure high availability, the system architecture supports load balancing across multiple backend instances. This allows requests to be distributed evenly and maintains uptime during peak usage. Built-in retry logic is implemented in critical operations such as loan disbursement and M-Pesa transactions to handle temporary failures. Data is periodically backed up using automated backup services provided by the hosting platform and MongoDB Atlas, reducing the risk of permanent data loss.

7.5 Usability and Accessibility Design

The MyChama interface is designed with WCAG 2.1 accessibility guidelines in mind to support users of diverse abilities. The application supports keyboard navigation, color contrast standards, and responsive design for use on both desktop and mobile browsers. Font sizes and interactive elements are optimized for readability and ease of use. Forms include real-time validation feedback to reduce user frustration and error rates. The UI is developed using modular React components, ensuring a consistent experience throughout the platform.

8. Deployment and Infrastructure Design

This section outlines how the MyChama system will be deployed using Firebase Hosting, along with backend services and database infrastructure hosted on compatible platforms. The deployment process is designed to be secure, maintainable, and scalable for long-term use.

8.1 Target Platforms

MyChama is a web-only application developed using Bootstrap 5 for the frontend UI and Node.js with Express for backend functionality. The frontend will be deployed on Firebase Hosting, a globally distributed CDN that serves static and dynamic content securely over HTTPS.

Backend APIs and dynamic features will be hosted using either:

- Firebase Cloud Functions *(for serverless Express backend)*, or

- Render/Railway *(for full Express server deployment)*

Data storage is handled using MongoDB Atlas, a cloud-managed NoSQL database with backup and scaling capabilities.

8.2 Infrastructure Diagram

The infrastructure diagram provides a high-level view of how the MyChama system components are deployed and interact across the cloud environment. It illustrates the flow of user requests from the browser to the Firebase Hosting layer, backend API services, and the cloud database. This architecture emphasizes modular separation between the presentation, logic, and data layers while leveraging Firebase for static content delivery and external services for backend logic and data persistence. The deployment strategy prioritizes performance, security, and scalability, supporting real-time interactions and streamlined development-to-deployment workflows.
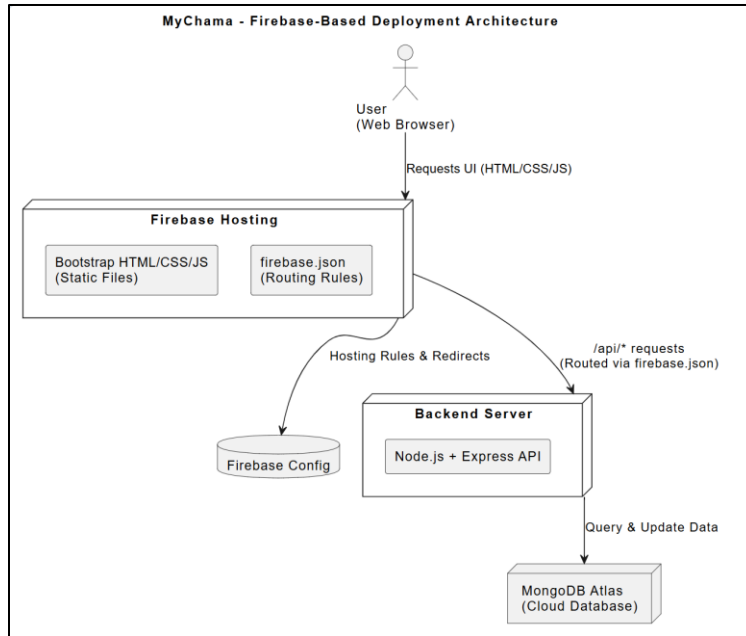
Figure 7: Infrastructure Diagram

Figure 7 illustrates the deployment infrastructure for the MyChama web application using Firebase Hosting. The system architecture is designed to support scalability, modularity, and cloud-based deployment while maintaining a lightweight and responsive user experience.

Users interact with the application through a web browser. The frontend, developed using Bootstrap and static HTML templates, is served from Firebase Hosting, which acts as a global content delivery network (CDN) for fast and secure delivery of assets. When a user performs an action that requires backend processing such as logging in, submitting contributions, or requesting a loan. Firebase routes the request to a Node.js Express backend, hosted either as a Firebase Cloud Function or through an external platform such as Render or Railway.

The backend processes the request and interacts with MongoDB Atlas, a cloud-hosted NoSQL database responsible for managing structured data related to users, contributions, loans, meetings, and polls. The flow of data between frontend, backend, and database is coordinated through

RESTful APIs. Routing rules and API endpoints are defined in the firebase.json configuration file, enabling seamless integration between the static frontend and dynamic backend services.

This infrastructure supports secure HTTPS communication, automatic scaling, and environment separation for development, staging, and production, ensuring the MyChama system remains robust, maintainable, and cloud-native.

8.3 Environments

The MyChama system is designed to operate across three distinct environments: development, staging, and production - to support reliable testing, quality assurance, and live deployment workflows. Each environment is configured separately to isolate changes, protect user data, and ensure a smooth transition from development to production.

Development Environment

The development environment is used by engineers to build and test features locally. It runs on local machines or containers and uses Firebase's local emulator suite for frontend previews and optional mock services. Developers use test credentials and sample data to validate new UI components, backend logic, and database interactions. Debug logging and live reloading are enabled to accelerate iteration.

- **Tools used**: Firebase Emulator, local Node.js server, .env.dev configs
- **Data**: Dummy/mock data only
- **Purpose**: Feature development, integration testing

Staging Environment

The staging environment closely mirrors production but is used solely for pre-release validation. It runs on Firebase Hosting Preview Channels and points to a separate MongoDB Atlas staging

database. Features from the develop branch are deployed here automatically via GitHub Actions for internal testing, stakeholder demos, and user acceptance testing (UAT).

- **Tools used**: Firebase Hosting (Preview Channels), Render (staging backend)

- **Data**: Realistic but test-safe sample data

- **Purpose**: End-to-end testing, internal QA, UAT, stakeholder review

Production Environment

The production environment is the live version of the MyChama system, publicly accessible to end-users. It is hosted on Firebase Hosting (Live Channel) with secure HTTPS, real-time monitoring, daily backups, and domain-level access control. Backend services and MongoDB Atlas are configured for high availability, performance, and scalability.

- **Tools used**: Firebase Hosting (Live), Render/Railway for backend, MongoDB Atlas (prod tier)

- **Data**: Live user and financial data

- **Purpose**: Live application used by chama members and administrators

Each environment uses separate configuration files (.env), database instances, and access credentials. This separation ensures minimal risk during deployment, quick rollback options, and a controlled deployment pipeline from feature development to end-user access.

9. Testing Design

Testing is a critical part of the MyChama system development lifecycle. The testing design ensures the reliability, security, and correctness of features before they reach end users. This section outlines the planned strategy, test data practices, automated testing setup, and integration with the CI/CD pipeline.

9.1 Testing Strategy

The MyChama system will follow a multi-level testing strategy covering unit, integration, system, and acceptance testing:

- **Unit Testing**: All core functions and modules, such as loan calculations, contribution validation, and user authentication, will be tested independently to confirm they return correct outputs for known inputs.

- **Integration Testing**: Focuses on testing interactions between components such as the API and MongoDB, or frontend and backend integration through API endpoints.

- **System Testing**: The entire application is tested end-to-end to validate workflows such as user registration, loan approvals, meeting scheduling, and poll creation.

- **User Acceptance Testing (UAT)**: Conducted on the staging environment with real chama users to validate whether the system meets functional expectations.

Manual testing will also be conducted for usability, accessibility, and edge cases not covered in automation.

9.2 Test Data Requirements

Test environments (development and staging) will use anonymized or synthetic data to replicate common chama use cases. Test datasets will include:

- Sample users with various roles (Member, Treasurer, Chairperson)

- Simulated chama groups with contribution and loan history

- Scheduled meetings and mock announcements

- Polls with prefilled options and fake voting behavior

This ensures test coverage over permission handling, financial operations, and group coordination features.

9.3 Automated Tests

To ensure code reliability and prevent regressions, the MyChama system will implement automated testing for both backend and frontend (where applicable):

Tools:

- Jest or Mocha/Chai for backend (Node.js + Express) unit and integration testing

- Supertest for API endpoint testing

- (Optional) Playwright or Cypress for end-to-end browser testing of the Bootstrap UI

Coverage Goals:

- 80%+ coverage for core backend modules

- 100% coverage for authentication and financial logic

- Smoke tests on critical user flows (login, contribute, request loan)

Test results will be monitored in every pull request to block faulty code from being merged into develop or main.

9.4 CI/CD Testing Integration

The testing framework is fully integrated into the GitHub Actions CI/CD pipeline. For every push and pull request:

- Dependencies are installed (npm ci)

- Linting is performed to ensure code quality

- Tests are executed automatically

- If any test fails, the deployment process is halted

For staging and production releases, an additional manual review step is required before pushing to Firebase Hosting or Render. This ensures that automated tests act as the first line of defense while still allowing human oversight.

10. Risks and Mitigation Plans

The following table outlines potential risks related to the system's architecture, third-party services, and deployment strategy, along with the mitigation measures designed to reduce their impact.

Risk	Description	Mitigation Strategy
Third-party API limits	Firebase Cloud Functions, M-Pesa API, or MongoDB Atlas may enforce usage or rate limits.	Implement request throttling, exponential backoff retry queues, and fallback UI messages for users. Monitor API quotas actively using alerts.
Unreliable internet connectivity	End-users may access the platform from rural areas with unstable internet.	Optimize frontend for fast loading and offline caching using service workers (where feasible). Keep page sizes light.
Data loss or corruption	Accidental deletions or unhandled server failures could lead to critical data loss.	Enable automated daily backups in MongoDB Atlas. Use transaction support for financial actions. Perform regular integrity checks.
Authentication security breaches	JWT tokens or password hashes could be compromised.	Use HTTPS, secure token lifespans, bcrypt for hashing, and implement 2FA for admin users. Log and monitor suspicious login attempts.

System-downtime during deployment	Deployments might interrupt live services if not isolated.	Use Firebase Hosting preview channels for safe staging. Schedule zero-downtime deployments. Use blue-green deployment methods if possible.
Scaling bottlenecks	A sudden increase in users may overload backend or database.	Use serverless architecture with autoscaling (Cloud Functions). Enable sharding and horizontal scaling in MongoDB Atlas. Monitor performance with dashboards.
Low user engagement	Users may not adopt the platform due to unfamiliar UI or lack of trust.	Conduct usability testing early. Include onboarding walkthroughs. Build transparency features (audit logs, real-time balances) to build user trust.
Cross-browser issues or mobile rendering problems	UI may not behave consistently across browsers/devices.	Use Bootstrap's responsive grid system. Test on Chrome, Firefox, Safari, and Android/iOS browsers before release.

11. Appendices

This section includes supporting materials referenced throughout the design document but too detailed to include inline. It provides access to diagrams, logs of key architectural decisions, and a list of tools and libraries used in the development and deployment of the MyChama system.

11.1 Diagrams

The following diagrams are included in full resolution and may be referenced from earlier sections:

- **Figure 1**: Component Diagram

- **Figure 2**: Entity Relationship Diagram (ERD)

- **Figure 3**: Level 1 Data Flow Diagram (DFD)

- **Figure 4**: Control Flow Diagram

- **Figure 5**: State Diagram – Loan Lifecycle

- **Figure 6**: State Diagram – Poll Lifecycle

- **Figure 7**: Infrastructure Diagram (Firebase Deployment)

11.2 Design Decision Logs

Key architectural and implementation decisions were documented throughout the project lifecycle.

A summary of important decisions is listed below:

Decision	*Justification*	*Date*
Use of Firebase Hosting	Lightweight deployment for static Bootstrap frontend	May 2025
Use of MongoDB Atlas	NoSQL flexibility for document-style records	May 2025
Token-based auth with JWT	Stateless and scalable across clients	June 2025
Frontend framework: Bootstrap (not React)	Easier for rapid prototyping and lower learning curve	June 2025
CI/CD using GitHub Actions	Seamless integration with GitHub and Firebase CLI	June 2025

11.3 Tools and Libraries

The following tools and libraries were used across different parts of the MyChama system:

Frontend

- **Bootstrap 5** – UI styling and layout

- **Font Awesome** – Icons

- **HTML5, CSS3, JavaScript**

Backend

- **Node.js + Express.js** – Web server and routing

- **Mongoose** – MongoDB object modeling

- **jsonwebtoken (JWT)** – Authentication tokens

- **bcrypt.js** – Password hashing

- **dotenv** – Environment configuration

Database

- **MongoDB Atlas** – Cloud database

- **MongoDB Compass** – Local DB management

Deployment & DevOps

- **Firebase Hosting** – Static frontend deployment

- **Firebase CLI** – Hosting and preview channels

- **GitHub Actions** – CI/CD automation

- **Docker (optional)** – Containerization in dev/staging

Testing

- **Jest / Mocha / Supertest** – Backend testing

- **Cypress / Playwright (optional)** – End-to-end UI tests

All tools are listed with their versions in package.json and devDependencies.