ChatGPT

Codex Development Manual: Next-Gen Loan Management System (LMS) for South African Micro-Lenders

Table of Contents

- C11. Prompt – Implement Third-Party Integrations
- C12. Prompt – Implement Mobile App Support
- C13. Prompt – Implement Admin Tools Module
- C14. Prompt – Implement Security Enhancements
- C15. Prompt – Implement AI & ML Features

SECTION A: PROJECT FOUNDATION

A1. Executive Summary

This manual is a comprehensive, step-by-step development guide for a **next-generation Loan Management System (LMS)** tailored to **South African micro-lenders**. It is designed for use with OpenAI's Codex (an AI code generator), enabling feature-by-feature development of a robust LMS backend and frontend. The system's mission is to **surpass existing platforms like Mobiloan** – currently regarded as one of South Africa's best LMS solutions [1] – by providing more detail, better design, stronger compliance, and superior usability. Every aspect of the loan lifecycle, from client onboarding to final repayment, will be covered in extreme detail, ensuring nothing is left to guesswork. The goal is a **"plug-and-play" development manual**: Codex can be fed each module's prompt in sequence to build out the entire system without losing context or introducing bugs.

South African Context: The LMS is built with a South African perspective, aligning with local regulations and practices. South Africa's microfinance industry operates under strict laws (National Credit Act, FICA, POPIA) aimed at responsible lending and consumer protection [2]. This manual emphasizes compliance with these laws (e.g. **affordability assessments**, interest rate caps, data privacy) to ensure the system not only functions well but also **operates within the legal framework** [3]. The solution accounts for South African specifics like the National Credit Regulator (NCR) guidelines, **Protection of Personal Information Act (POPIA)** for data privacy, and **Financial Intelligence Centre Act (FICA)** for customer identification and anti-money-laundering checks. It also integrates local fintech services (such as payment gateways and credit bureaus) commonly used by micro-lenders in SA.

Target Users: The primary users are **micro-lending businesses** in South Africa, including short-term loan providers and small credit lenders. The system will support **multiple user roles** (admin, loan officers, credit/risk analysts, collectors, etc.) with a UI and workflows optimized for both **desktop and mobile** usage. Borrowers (clients) are indirect users via a self-service mobile app or web portal for loan applications and account management. By catering to both internal staff and client self-service, the LMS provides end-to-end digital loan management.

Feature Development Approach: The manual is organized in a logical sequence, **starting with foundational features (user accounts & security)** and progressively adding complexity (client data, loans, payments, etc.). Each feature/module is fully specified with clear requirements: every screen's fields, every role's permissions, interactions between modules, and compliance checkpoints are explicitly listed. **No vague language** – each "etc." from typical specs is replaced with concrete details. The development sequence ensures that **dependencies are handled in the correct order** (e.g. set up user auth before restricted features, create client records before loan origination, process payments before collections, etc.).

Outcome: Following this manual, Codex will generate a **state-of-the-art, cloud-based LMS** that is **more detailed and compliant** than existing solutions. It will be *"plug-and-play"* – meaning the code for each feature can be integrated immediately into the project – and modular, so adding one feature won't break others. The finished product will be a **secure, scalable, and user-friendly** LMS platform that handles the full loan lifecycle (origination to closure) and meets all regulatory requirements. This includes advanced capabilities like **paperless document handling, mobile accessibility, real-time credit checks, automated payments, robust analytics, and AI-driven insights**, exceeding the current industry offerings in South Africa and globally.

A2. System Architecture

The system is designed with a **modern, serverless architecture** that leverages **Google Firebase** for the backend and a **cross-platform frontend**. The architecture ensures scalability, maintainability, and secure data handling, crucial for a financial application. Below is an overview of the key architectural components:

- **Client Applications (Frontend):** The LMS will have a web-based frontend and a mobile application interface. The **web app** (desktop and mobile web) is built as a single-page application (SPA) that is responsive to different screen sizes. The **mobile app** is built with a cross-platform framework and will reuse much of the web app logic, ensuring *"one solution"* works across smartphones, tablets, and PCs [4]. This unified approach means the same core features are available on all devices. The frontend communicates with backend services primarily through Firebase SDKs and RESTful APIs (for third-party services).

- **Backend (Firebase):** Firebase serves as the backbone:

- **Firebase Authentication** is used for managing user accounts and authentication (email/password login, with support for multi-factor authentication as needed).
- **Cloud Firestore** (NoSQL database) stores application data (users, clients, loans, payments, etc.). It's structured into collections and documents (detailed in A4) to reflect our data model.
- **Firebase Cloud Functions** contain server-side logic (business rules, integrations, scheduled jobs). Cloud Functions are written in TypeScript/Node.js and enforce business constraints (e.g., calculating loan schedules, enforcing compliance rules, calling external APIs for credit checks or payments).
- **Firebase Storage** is used for storing documents and files (e.g., uploaded client documents, generated loan agreements). This includes robust security rules to restrict access to files by role.
- **Firebase Security Rules** govern read/write access to Firestore and Storage, implementing row-level security based on user roles and ensuring data privacy.

- **Firebase Cloud Messaging (FCM)** may be used for push notifications to the mobile app (e.g., payment reminders, notifications of loan approval).

- **Third-Party Service Integrations:** The architecture includes integration with external services via secure APIs:

- **Payment Gateways:** e.g., **ALLPS** and **NuPay** for debit order collections, **Ozow** and **PayFast** for electronic payments. These services are accessed either via direct API calls (from Cloud Functions) or via secure file exchange if API isn't available. For instance, the system can schedule debit orders through Allps – a certified payment provider offering various debit order options (authenticated and

non-authenticated) [5] – by invoking Allps' API from a Cloud Function. Similarly, it can integrate with NuPay (a leading provider of early debit order solutions in SA [6]) to handle DebiCheck or NAEDO transactions.

- **Credit Bureaus:** e.g., **TransUnion, Experian** (and possibly XDS). The system uses their APIs via Cloud Functions to pull credit reports or scores during the risk assessment phase. For example, the integration with XDS returns a comprehensive credit report to the LMS [7].
- **Communication APIs:** e.g., an SMS gateway (or Twilio) for SMS, email service (SMTP or SendGrid API), and the WhatsApp Business API for sending WhatsApp messages. These allow the system to automatically send notifications and ensure omnichannel communication with clients.

- **Other Integrations:** The system is designed to easily plug in additional services (e.g., ID verification services, bank account verification, or analytics tools) through Cloud Functions and modular service classes.

- **Modular Design:** Each feature module (user management, client management, loans, etc.) is **encapsulated** so it can be developed and tested independently. Frontend uses a modular structure (for example, Angular modules or React components separated by feature) and the backend uses separate Cloud Function sets or microservices for each domain. This modularity prevents new features from breaking existing ones and makes maintenance easier. Inter-module communication happens through well-defined interfaces (for example, loan module calls payment module's functions to schedule debits, rather than sharing data in unpredictable ways).

- **Workflow Overview:** In a typical user flow, a staff user on the frontend (web or mobile) will trigger operations that hit Firebase or cloud functions. For example, when a loan officer submits a new loan application on the frontend, the app writes to Firestore (creating a loan entry in a "Applications" collection) and calls a Cloud Function to perform credit checks and compute an amortization schedule. The Cloud Function updates Firestore with results (approval status, repayment schedule). A scheduled Cloud Function periodically monitors due loans and triggers payment instructions via the integrated payment provider. All user interactions happen in real-time with Firestore's updates (the frontend listens to changes for live updates, e.g., seeing a loan's status change to "Approved").

- **Scalability & Reliability:** Firebase's serverless infrastructure scales automatically with usage. The stateless Cloud Functions scale out as needed for heavy operations (e.g., sending batch communications or performing nightly interest accrual calculations). Data is replicated across Firebase's zones for reliability. We leverage Firebase's robust infrastructure (with Google Cloud's compliance certifications) so that our LMS data is secure and reliably available (99.9% uptime). We also design idempotent and transactional operations in Cloud Functions to avoid double-processing (for example, ensuring a payment is not recorded twice if a function retried).

- **Security & Compliance in Architecture:** Security is baked into the architecture. All communication between frontend and Firebase is over HTTPS. Sensitive operations (like writing a loan approval or performing a payout) are done in Cloud Functions with server-side checks on the user's role and the compliance rules. Data is partitioned so that users only access what they should: e.g., clients are tagged by branch so branch-specific officers only see their branch's clients, if applicable. Strong encryption is applied where needed; Firebase by default encrypts data at rest and in transit, and we add extra encryption for particularly sensitive fields (like password hints, or personal IDs) in the database. The architecture supports **audit logging** via Cloud Functions that log key events (login,

data changes) to an audit trail store. This combination of measures aligns with POPIA's requirement for technical security controls (encryption, access control, audit logs) [8] [9].

In summary, the architecture leverages cloud services and modular design to achieve a **flexible, scalable LMS platform**. It ensures that as new features are developed, they can plug into the existing structure with minimal friction. The design supports a seamless experience across devices and locations (enabling *"business mobility"* for loan origination inside or outside the office [4]) while maintaining centralized control and data integrity. The following sections detail the technology choices, data schema, and each feature module.

A3. Technology Stack

The project employs a modern technology stack optimized for **full-stack TypeScript development** and cross-platform deployment. Key components of the stack include:

- **Frontend Framework: Angular 15+** (TypeScript) with **Angular Material UI** for a responsive, accessible design. Angular is chosen for its structured architecture (useful for a large app with many modules) and built-in support for forms, routing, and internationalization. The UI will follow **Material Design** principles, ensuring a clean and familiar interface for users. *Alternative:* A comparable stack could use **React** with libraries like Material-UI or **Vue.js**; however, Angular's out-of-the-box tooling for enterprise apps suits our needs well.

- **Mobile Framework: Ionic Capacitor** (with Angular) to package the web app as a native mobile app for Android/iOS, or **React Native** as an alternative dedicated mobile front-end. The aim is code-reuse: using Ionic, much of the Angular codebase can be reused to build a mobile app that has access to device features (camera, GPS, etc.) while sharing business logic with the web app. This aligns with competitor approaches where the same app works on phone, tablet, and PC [4]. If using React Native instead, a separate codebase would implement similar UI and calls to Firebase, but for this manual, we'll assume the Ionic/Angular unified approach for efficiency.

- **State Management:** NgRx (Redux pattern for Angular) or Reactive Services to manage application state across components, particularly for complex data like the currently loaded client or loan details. This helps keep the app performance optimal and data consistent.

- **Backend Platform: Google Firebase** on Google Cloud:

- **Cloud Firestore** as the database (NoSQL, schema-less but we'll enforce a schema via validation). It offers real-time data sync which is great for things like seeing updates to a loan status live.
- **Firebase Authentication** for user auth (supports email/password, phone OTP, etc. We'll use email/password + optional 2FA).
- **Cloud Functions (Node.js with TypeScript)** for server-side logic (APIs, triggers, scheduled tasks). Using TypeScript keeps our code consistent end-to-end.
- **Firebase Storage** for file storage (documents, images).
- **Cloud Messaging** for push notifications.

- **Cloud Scheduler** (if needed for CRON-like jobs) to trigger functions (e.g., daily summary jobs or payment retries).

- **External APIs & Libraries:**

- **Payments:** Integrations with **Allps** and **NuPay** (debit orders) likely via REST APIs or SDK provided by those companies. Also, **Ozow** and **PayFast** via their REST APIs for initiating and confirming payments. We might use their sandbox environments for development/testing.
- **Credit Bureau APIs:** Integrating **TransUnion** and **Experian** via REST/SOAP web services (depending on what they offer) for credit checks. This may involve using an SDK or building HTTP requests with authentication (likely API keys or certificates).
- **Communication APIs:** Use **Twilio** (as an example) to send SMS and WhatsApp messages, and possibly SendGrid or a similar service for emails. Twilio's APIs allow WhatsApp messaging in a sanctioned way, or alternatively Meta's WhatsApp Cloud API can be used.
- **Mapping & Location:** If capturing GPS coordinates, we might use Google Maps API for reverse lookup (address verification) or just to display location.
- **OCR & Document Processing:** Google Cloud Vision API for OCR (to read text from ID documents, etc.), if we implement auto-extraction of info from uploaded images.

- **Machine Learning:** TensorFlow.js or Python TensorFlow (in Cloud Functions via AI Platform) for any ML models we incorporate (like credit risk model). Alternatively, use a service like Vertex AI or an AutoML model if needed in future. Initially, we may just simulate ML outcomes.

- **Development Tools & Practices:**

- **IDE & Version Control:** Use Visual Studio Code (with Firebase Emulators for local testing) and Git for version control. A Git branching strategy (feature branches for each module) will align with our modular development approach.
- **Testing:** Jasmine/Karma for unit tests in Angular; Jest or Mocha for Cloud Functions. End-to-end testing with a framework like Protractor or Cypress (simulate user flows like applying for a loan).
- **CI/CD:** Optionally, setup CI with GitHub Actions or GitLab CI to run tests and deploy to Firebase Hosting/Functions on merge. Firebase CLI will be used for deployments.

- **Localization:** Use Angular i18n or ngx-translate for multilingual support. The default language is English, but the app will be built to allow easy addition of South African languages (e.g., Afrikaans, Zulu) in the future. All user-facing strings will be externalized in a JSON or XLIFF file for translation.

- **Security & Compliance:** Use **ESLint/TSLint** for code linting (including rules to avoid insecure code). Dependencies will be monitored for vulnerabilities (e.g., using `npm audit`). All API keys and secrets for third-party integrations will be stored securely (Firebase Functions config or a secure key management service) – not hardcoded. The stack ensures compliance with PCI-DSS for payment data by not handling card data directly (we redirect to PayFast/Ozow for card entry) and by using providers (Allps, NuPay) that are PCI-DSS certified [10].

Overall, this technology stack provides a **robust foundation** to implement the LMS. It emphasizes consistency (TypeScript across front and back), scalability (serverless backend), and a rich user interface that works on **both desktop and mobile** environments. By using widely supported tools and frameworks, we ensure the longevity of the solution and ease of finding developer support.

A4. Database Schema

Although Firestore is schemaless, we define a **clear schema structure** for our data collections and fields. This schema will be enforced through application logic and security rules. All data is stored using **document collections** in Firestore, identified by unique IDs. Below is the proposed schema with key collections and their fields (all data is in *English* by default, with some fields possibly supporting localization if needed):

- **Users Collection (`users`):** Stores application user (staff) profiles and role info.
  Fields:
- `userId` (string, Firestore document ID or same as Firebase Auth UID)
- `email` (string, user's login email, unique)
- `passwordHash` (string, if we store or we rely on Firebase Auth solely)
- `name` (string, full name)
- `role` (string, e.g., "Admin", "Manager", "LoanOfficer", "Collector", etc.)
- `branchId` (string, reference to branch if user is tied to a branch, else null for head office)
- `isActive` (boolean, to disable an account without deleting)
- `lastLogin` (timestamp, last login time)
- `createdAt` (timestamp)
- `createdBy` (userId of who added the user)
  Access control: Only Admins (and perhaps Managers) can create or modify users. Non-admins can read limited info (maybe just their own profile).

- **Roles Collection (`roles`):** (Optional) If we allow dynamic role definitions. Each role document defines the permissions for that role.
  Fields:

- `roleId` (string, e.g., "Admin", "LoanOfficer")
- `description` (string)
- `permissions` (map of permission flags, e.g., `{ "CREATE_USER": true, "APPROVE_LOAN": false, ... }`).
  Note: We may hard-code roles in code instead of storing in DB. If stored, only Admin can modify.

- **Branches Collection (`branches`):** (If the micro-lender has multiple branches or outlets)
  Fields:

- `branchId` (string)
- `name` (string)
- `address` (string)
- `region` (string, maybe province or area)
- `managerUserId` (string, userId of branch manager)
- `createdAt` (timestamp)
- etc.
  Use: Used to filter clients/loans by branch and for reporting. Optional for a single-office lender.

- **Clients Collection (`clients`)**: Stores each borrower/client's details.
  Fields: *(Captured during client onboarding or loan application)*

- `clientId` (string, primary key)
- `firstName` (string)
- `lastName` (string)
- `southAfricanID` (string, 13-digit SA ID number or passport for foreign nationals)
- `birthDate` (date)
- `gender` (string, optional; could be "Male", "Female", "Other")
- `contact.mobile` (string, cellphone number)
- `contact.email` (string, if any)
- `contact.address` (nested object with `line1`, `line2`, `city`, `postalCode`, etc. for residential address)
- `employment.status` (string, e.g., "Employed", "Unemployed", "Self-Employed", "Pensioner")
- `employment.employerName` (string, if employed)
- `employment.income` (number, monthly net income in ZAR)
- `employment.payday` (number, day of month typically paid, if applicable)
- `expenses.estimatedMonthly` (number, estimated total monthly expenses in ZAR, used for affordability)
- `bank.accountNumber` (string)
- `bank.bankName` (string, e.g., "FNB", "Standard Bank")
- `bank.branchCode` (string, if needed)
- `bank.accountType` (string, e.g., "Cheque", "Savings")
- `consent.creditCheck` (boolean, whether client gave consent for credit bureau check – required by law)
- `consent.marketing` (boolean, consent to receive marketing communications, per POPIA)
- `createdAt` (timestamp)
- `createdBy` (userId of officer who added)
- `branchId` (string, link to branch if applicable)
- `status` (string, e.g., "Active", "Blacklisted", "Closed") – perhaps to mark if client is barred from loans due to past defaults, etc.
- `notes` (string, general notes about client)

Additionally, certain data might be filled later:
- `creditReport` (object) – summary of latest credit bureau info: e.g., `score` (number), `bureau` (which bureau), `datePulled`, `accountsInArrears`, etc. (This is updated via Risk module when a credit check is run, and could also be stored in a separate subcollection e.g. `clients/{id}/creditChecks`).
- `FICA.verified` (boolean) – whether KYC documents (ID, proof of address) are verified. Possibly also store references to documents in storage.
- `popiaDeletionDate` (timestamp or date) – if client requested deletion or end of retention period, mark when their data can be purged per POPIA.

Indexes: We will index on `southAfricanID` (unique constraint, to avoid duplicate client entries for one ID) and on `lastName` / `mobile` for search.
Relationships: We expect related subcollections: e.g., documents (`clients/{id}/documents`) and loans

(`clients/{id}/loans` could be a logical nesting, though we might store all loans in a central collection referencing clientId for simplicity).

- **Loans Collection (`loans`)**: Stores loan accounts and applications. Each loan document goes through various statuses from application to closure.
  Fields:
- `loanId` (string, unique ID)
- `clientId` (string, reference to the borrower in `clients`)
- `loanProductId` (string, reference to a product in loan products setup; defines interest rate, fees)
- `principalAmount` (number, the principal loan amount in ZAR)
- `termMonths` (number, term of loan in months, or could be weeks/days for short-term loans – we will store as number of months for standardization; for shorter loans, use fraction of a month or store in days if needed)
- `interestRate` (number, annual nominal interest rate as percentage, e.g., 28% for unsecured, or monthly rate if short-term; specify which – likely annual % to comply with NCR expression)
- `interestType` (string, e.g., "Fixed", "Declining Balance", etc. – determines how interest is calculated)
- `fee.initiation` (number, upfront initiation fee charged)
- `fee.service` (number, monthly service fee)
- `fee.other` (object, any other fees like insurance premium, if applicable)
- `schedule` (array of objects) – the repayment schedule. Each element has `dueDate` (date), `amountDue` (number), `interestPortion` (number), `principalPortion` (number), `status` (e.g., "Due", "Paid", "Missed"), and `paymentId` if it was paid. Alternatively, we use a subcollection for installments (e.g., `loans/{loanId}/installments`). Using a subcollection might be cleaner for variable-length schedules.
- `status` (string) – current status of loan. Possible values: `"Application"` (just created, not yet approved), `"PendingApproval"`, `"Approved"`, `"Rejected"`, `"Active"` (disbursed and currently being repaid), `"Closed"` (fully paid), `"WrittenOff"`, `"Collections"` (if handed over or in hard collections). We will refine states in the Loan Origination and Collections sections.
- `applicationDate` (timestamp) – when loan application was created.
- `approvalDate` (timestamp, if approved)
- `approvedBy` (userId of approver)
- `rejectionReason` (string, if rejected)
- `disbursementDate` (timestamp, when funds disbursed)
- `disbursedBy` (userId or system if automated)
- `closedDate` (timestamp, when loan closed/settled)
- `outstandingBalance` (number, remaining amount to be paid, auto-calculated as payments are made)
- `paidToDate` (number, total amount paid so far)
- `nextDueDate` (date, next installment due date, for convenience)
- `lateDays` (number, how many days past due if status is delinquent)
- `flags` (object) – various boolean flags: e.g., `inCollections` (if sent to collections), `rescheduled` (if loan was restructured), etc.
- `affordabilityCheck` (object) – summary of affordability results at origination (e.g., `income` vs `expenses`, calculated installment, and a boolean `pass` indicating if loan was granted per affordability rules).

- `creditScoreAtOrigination` (number) – credit score of client at time of decision.
- `loanPurpose` (string, e.g., "Education", "Business", etc. if captured)
- `comments` (string) – any comment from loan officer on this application.

Relationships: Each loan is linked to a client. Could also link to branch via client or a separate field if needed (for branch-level portfolio). Also, we might have subcollections: - `loans/{loanId}/payments` for payment transactions (alternatively a global Payments collection). - `loans/{loanId}/documents` for documents specifically for that loan (e.g., signed contract, any supporting docs). - `loans/{loanId}/collections` for collection efforts logs (or that can be in a global Collections tickets collection referencing the loan).

Note: Firestore allows subcollections; we might use subcollections for payments and collections notes to keep them logically grouped by loan, or use separate top-level collections with references. The choice can depend on query patterns (for reporting, a top-level `payments` collection might be easier to aggregate across loans).

- **Payments Collection (`payments`)**: Stores payment transactions (repayments from clients, or disbursements to clients). Alternatively, separate subcollections as noted. But we define one for clarity.
  Fields:
- `paymentId` (string)
- `loanId` (string, reference to loan)
- `clientId` (string, redundancy for quick access)
- `type` (string, e.g., "Disbursement", "Repayment")
- `method` (string, e.g., "DebitOrder", "EFT", "Cash", "Card")
- `provider` (string, e.g., "Allps", "PayFast", if via an integration)
- `date` (timestamp of payment)
- `amount` (number)
- `status` (string, e.g., "Successful", "Failed", "Pending")
- `reference` (string, external reference like transaction ID from provider or bank reference)
- `retryOf` (paymentId, if this payment is a retry of a failed one)
- `comments` (string, e.g., reason if failed or any notes)
- `recordedBy` (userId if someone manually recorded a payment, else system)

Usage: This log allows reconciliation. Each repayment should correspond to an installment or multiple installments. We might store a mapping in each payment record to which installment(s) it covered or update installments status via payment triggers.

- **Collections (Tickets) Collection (`collections`)**: Manages collection efforts on overdue loans. Each entry could represent a *collections ticket* or case.
  Fields:
- `ticketId` (string)
- `loanId` (string)
- `clientId` (string)
- `openedDate` (timestamp when collections case opened)
- `openedBy` (userId, typically system or collector who initiated)
- `status` (string, e.g., "Open", "In Progress", "Closed", "Closed-Paid", "Closed-WriteOff")

- `assignedTo` (userId of collections agent handling it, if using assignment)
- `lastActionDate` (timestamp of last action taken on this case)
- `nextFollowUpDate` (date for next follow-up action)
- `currentBalance` (number, amount due at time case opened; can update as payments come in)
- `promisedPaymentDate` (date if client promised to pay)
- `promisedAmount` (number promised)
- `notes` (string or array of notes – though better to have a subcollection for logs of each action)

Possibly a subcollection `collections/{ticketId}/actions` for each contact or action:
- Each action doc with `actionDate`, `actionType` (call, SMS, email, etc.), `agentId`, `notes`, `result` (e.g., "Customer will pay on X date" or "No answer").
This keeps a timeline of collection efforts.

- **Loan Products Collection (`loanProducts`)**: Configurations for different loan offerings. (Admin Tools maintain this.)
  Fields:
- `productId` (string)
- `name` (string, e.g., "Short-term Loan 30-day", "Personal Loan 6-month")
- `maxAmount` (number)
- `minAmount` (number)
- `maxTerm` (number, in months)
- `minTerm` (number)
- `interestRateAnnual` (number, e.g., 28.5 for 28.5% per annum, or if short-term, maybe store monthly cap)
- `serviceFeeMonthly` (number)
- `initiationFee` (number or formula reference, often initiation fee is regulated: e.g., R165 + 10% of amount above R1000 up to R1050 max)
- `vatRate` (number, VAT on fees if applicable – in SA, certain fees may include VAT)
- `requiresCreditLifeInsurance` (boolean, if this product mandates an insurance)
- `insuranceRate` (number, if any insurance premium rate)
- `gracePeriodDays` (number, if any grace days for payments)
- `earlySettlementPolicy` (string, e.g., "No penalty", or if fees apply for early payoff)
- `description` (string, brief description of product)
- `active` (boolean, to deactivate a product from new issuance)

- `createdBy` / `createdAt` for audit.

- **Repayment Methods (`repaymentMethods`)** and **Payout Methods (`payoutMethods`)**: These config collections list allowed methods (e.g., "Debit Order", "Cash", "EFT") which can be referenced by loans or chosen during processing. Fields might include a name and maybe integration mapping (like map "Debit Order" to provider Allps vs Nupay depending on branch or choice).

- **Documents Collection (`documents`)**: While files are in Firebase Storage, metadata can be stored in Firestore. Possibly organized under each client or loan, or a single collection referencing owner. For clarity:
  Each document record has:

- `docId` (string)
- `ownerType` (string, "client" or "loan")
- `ownerId` (string, the clientId or loanId it belongs to)
- `type` (string, e.g., "ID", "Proof of Residence", "Payslip", "LoanAgreement")
- `fileName` (string, stored file name or path)
- `fileUrl` (string, Storage URL or path token)
- `uploadedBy` (userId or "client" if via self-service)
- `uploadedAt` (timestamp)
- `verified` (boolean, if an admin has verified this document as valid)
- `verifiedBy` (userId, if verified)
- `notes` (string, any note like "ID expired, needs update")

- **Audit Logs Collection (`auditLogs`)**: Logs of important events for security and compliance (could also use Firebase's native logging, but storing some events in Firestore helps build an audit trail accessible in-app).
  Fields:

- `eventId`
- `timestamp`
- `userId` (who performed action, if applicable)
- `action` (string, e.g., "LOGIN", "CREATE_USER", "APPROVE_LOAN", "EDIT_CLIENT")
- `details` (string or object, e.g., which record was affected and old vs new values if needed)
- `ipAddress` (if we capture the IP of user – from frontend, might not be direct since backend Cloud Functions have it)
- `success` (bool, e.g., login success/fail)
  This is mainly for internal review and compliance audits. It helps demonstrate accountability [9].

- **Miscellaneous Config Collections:**

- **Company Info (`company`)**: e.g., store company name, registration number, NCR license number, physical address, support contact details, etc., for use in documents or reports.
- **Commission Setup (`commissions`)**: if needed to configure commission rates for agents or branches.
- **Insurance Products (`insuranceProducts`)**: if offering credit life insurance, details of the insurance coverage and premium rates.
- **Categories (`categories`)**: maybe categorize clients or loans (could be for internal use like risk categories or marketing).
- **Transaction Codes (`transactionTypes`)**: define types of transactions for accounting interface (if needed).
- **Integration Settings (`integrations`)**: store API keys or settings for external services (though these likely stored securely in function config rather than Firestore, to avoid exposure. But non-sensitive flags can be stored).

Each collection will have appropriate **indexes** for queries (Firestore auto-indexes single fields and some compound indexes will be defined for common queries e.g., loans by status and branch, clients by ID number).

Relational considerations: Firestore is not relational, but we ensure to store references (IDs) for linking documents. We will use *denormalization* where helpful (for example, storing client name in loan record for quick display, while also keeping the canonical data in `clients`). We must keep such data in sync via functions or careful updates.

Data Volume & Partitioning: The schema supports potentially thousands of clients and loans. If the scale grows (e.g., millions), we might consider sharding certain collections, but for micro-lenders the scale is typically manageable. We will use subcollections (like installments, actions) to distribute reads/writes if needed. Also, separating active vs closed loans might be considered (e.g., an `activeLoans` and `closedLoans` collection) to optimize queries for active portfolios. Our design, however, will treat them uniformly but could add a filter.

Security (data perspective): Every collection will have a corresponding security rule. For example, rules will enforce that a loan officer can only read clients and loans from their branch (if multi-branch setup), a collector can update collection notes but not alter loan amounts, etc. The schema is structured to make those rules straightforward (having branchId fields, owner relationships, etc.). Sensitive personal info like identification numbers and bank accounts will be carefully protected – possibly even stored in encrypted form in Firestore (e.g., encryption via Cloud KMS in a Cloud Function before writing) and decrypted on the client side only for authorized users. This reduces risk if the database were ever exposed, aligning with POPIA's expectation of protecting personal info by technical means [8].

This schema will be referenced in each feature section to show how data flows and which fields are used in each module.

A5. API Specifications

Even though much of the front-end will interact directly with Firestore (using Firebase SDK) and leverage Cloud Functions for backend logic, it is important to define the **API surface** of our system. This includes Cloud Function endpoints (which act like REST API calls) and any custom REST endpoints needed for third-party callbacks. Below we outline key APIs and their specs (input/output), structured by feature/module:

A5.1 Authentication & User APIs:

- `POST /api/auth/registerUser` (Cloud Function callable or HTTPS endpoint): *Registers a new user.*
  Input: `{ email, password, name, role, branchId }` (all fields required, role must be one of predefined roles, only Admin can call this).
  Process: Creates a new Firebase Auth user (email & password), then creates a user document in `users` collection with provided details. Sets custom claims (Firebase) for role for use in security rules.
  Output: Success message or error. In case of error (e.g., email in use), returns error code/message.

- `POST /api/auth/login`: *User login (if not using Firebase's built-in methods directly).*
  Input: `{ email, password }`.
  Process: Verifies credentials via Firebase Auth.
  Output: Returns a Firebase Auth token (JWT) or uses client SDK to sign in (in web/mobile, we'll use Firebase SDK directly, so this API may not be needed explicitly).

- `POST /api/auth/requestPasswordReset`: *Initiate password reset.*
  Input: `{ email }`.
  Process: Triggers Firebase to send password reset email.
  Output: Success or error.

- `GET /api/users/list`: *Retrieve user list (admin only).*
  Input: Optional filters (e.g., by role or branch).
  Output: Array of user objects (excluding sensitive info like password). This could also be done via Firestore query from the front-end (with security rules ensuring only Admin can query all users).

- `PUT /api/users/{userId}`: *Update a user's information or deactivate.*
  Input: userId as URL or param, and body can contain fields to update (name, role, branch, isActive).
  Process: Check permissions (only Admin or self for certain fields). Update Firestore and possibly Firebase Auth (for email change or if setting password via an admin reset).
  Output: Success or error.

- `DELETE /api/users/{userId}`: *Remove a user.*
  Actually, rather than deleting (which might cascade in Auth), we may just deactivate. But if needed: checks that current user is Admin and not deleting self inadvertently.
  Output: success.

(In practice, many of these can be achieved with direct Firestore operations if rules permit, but using Cloud Functions adds an extra layer of control and logging for critical changes. We'll often prefer Cloud Functions for actions that involve multiple steps or require secure enforcement.)

A5.2 Client Management APIs:

- `POST /api/clients`: *Create a new client record.*
  Input: A JSON payload with all client fields (see B2 for full list, including personal info, employment, etc.). If coming from a self-service channel, some fields like `consent.creditCheck` must be true and an initial application might accompany it.
  Process:
  - Validate ID number format (South African 13-digit ID has a known pattern and check-digit; ensure it passes or mark as potentially invalid).
  - Ensure the ID or mobile isn't already registered (to avoid duplicates) by searching `clients` collection.
  - If unique, write client doc to Firestore. Possibly also create an initial credit check request or mark as unverified.

    - If using Cloud Functions for this, it can also create a stub credit profile or call an ID verification service asynchronously. **Output:** Created client object or ID. If duplicate or validation fails, returns error code (e.g., "DUPLICATE_ID" or specific message).

- `GET /api/clients/{clientId}`: *Retrieve client profile.*
  Output: Full client data (excluding or masking sensitive info depending on caller's role). For example, a loan officer can see everything, but maybe certain data like full ID number could be masked for a general staff if policy requires.

- `PUT /api/clients/{clientId}`: *Update client info.*
  Input: Fields that changed (e.g., address update, employment update).
  Process: Validate changes (e.g., if updating ID number, re-check uniqueness). Log an audit event.
  Output: Updated data.

- `GET /api/clients/search`: *Search clients.*
  Input: query params like `idNumber` or `name` or `mobile`.
  Process: Query Firestore (with appropriate indexes) for matching documents the user is allowed to see (e.g., within their branch).
  Output: List of matching client records (with basic info: name, ID, mobile, status).
  (Alternatively, this can be done via Firestore queries from frontend if security rules allow search by those fields. Firestore doesn't support wildcards well, so search might be exact match or starting-with. For more advanced search (like partial name), an integration with Algolia or a cloud function to filter might be used.)

- `POST /api/clients/{clientId}/flag`: *Flag client (blacklist or special status).*
  Input: e.g., `{ "status": "Blacklisted", "reason": "Fraud suspected" }`.
  Process: Only compliance or admin roles can do this. Update client status, and optionally propagate this info (e.g., to not allow new loans). Possibly notify concerned parties.
  Output: Success.

- `POST /api/clients/{clientId}/documents`: *Upload a document for client.* (Could also be handled by direct Firebase Storage upload and metadata via separate call – see Document Handling). This might be covered under Document APIs below, but essentially: Input: file (binary) plus metadata (type, etc.), output: stored doc reference.

A5.3 Loan Origination & Servicing APIs:

- `POST /api/loans`: *Submit a new loan application.*
  Input: Payload including `clientId` (or full client info if client not created yet via self-service), `amount`, `term`, `productId`, and any application details. For self-service, might also include client's declared income/expense if not already in system.
  Process:

    1. Validate that client exists (if clientId given) or create a new client first if needed (if allowed).
    2. Check that client is eligible: not blacklisted, not having a pending application that conflicts, etc.
    3. Fetch product details (interest rates, fees) based on `productId`.

4. Calculate a preliminary repayment schedule and monthly installment using the formula for that product (ensuring interest and fees comply with the regulated max: e.g., cap interest at 28% or 5%/month for short term [11]).
5. Perform an affordability assessment: using client's income and expense info, ensure that the new installment + existing obligations ≤ a certain percentage of net income (common practice ~30-40% guideline). If fails, return an error or mark application as "Pre-declined: affordability".
6. (If integrated) Optionally initiate a credit bureau check here or flag it for the risk officer to do manually.
7. Save the loan application in `loans` collection with status e.g. `"PendingApproval"` and all calculated fields (schedule, fees, etc.). Possibly save the schedule in a subcollection or as an array.
8. If this API is called by a public self-service form, also trigger an alert for officers to review.
Output: Application ID and status. If further action needed (like documents upload), include that info.

- `GET /api/loans/{loanId}`: *Get loan details.*
Output: All loan info including schedule, current status, client info (maybe denormalized or via separate call).

- `POST /api/loans/{loanId}/approve`: *Approve a loan application.*
Input: `{approved: true/false, approvedAmount (if different), approvedTerm (if changed), notes}`.
Process:

  - Only Manager or designated approver roles can call. If approved:
  - Set loan status to `"Approved"`.
  - Record `approvedBy` and timestamp.
  - Finalize the repayment schedule (in case any adjustments made).
  - Initiate **disbursement** process: create a disbursement payment entry and either mark it pending (to be executed by finance or automatically via PayFast etc.). Possibly call an integration to send money (if using something like an EFT service; most micro-lenders disburse via EFT manually or via a banking integration batch).
  - Generate a loan agreement document (PDF) with all terms, and save it to Storage (this can be done by another function call or queue, but from API perspective might just trigger it).
  - If digital signature is required from client: send to client via email/WhatsApp a link or document to sign (or code to confirm).
  - Set up the **repayment schedule** in the system: e.g., schedule Cloud Functions or create entries in a `installments` subcollection and possibly queue them for debit order processing on due dates.
  - If any welcome communication is to be sent, trigger that (SMS "Your loan is approved and will be paid out…").
  - If rejected:
  - Set status `"Rejected"`, record reason.
  - Optionally log or communicate rejection to client.
  - Ensure all these actions are atomic or properly rollback on failure (Firestore transactions if multiple docs, or carefully ordered so that if one step fails, the system can recover). **Output:**

Confirmation of approval or rejection. For approval, could return some summary (like "Loan approved, disbursement scheduled").

- `POST /api/loans/{loanId}/disburse`: *Mark loan as disbursed.*
  If we separate approval from actual funding, this API handles the funding confirmation. **Process:** Possibly triggered after an external payment (like bank transfer) is completed. It sets loan status to `"Active"` (from Approved), logs disbursement details (date, transaction ref), and the loan is now officially in repayment phase. If approval API already did this immediate, this might not be separate.

- `GET /api/loans?status=Active` (and other filters): *List loans.*
  Output: List of loans, filterable by status, branch, client, etc. Used for screens like Active Loans listing, Pending Approvals listing, etc. Might implement via Firestore queries on front-end with composite indexes (e.g., filter by branch and status).

- `POST /api/loans/{loanId}/settle`: *Settle an active loan early.*
  Input: If a borrower wants to payoff early, input may include a date of settlement.
  Process: Calculate outstanding balance with possibly reduced interest (as per NCA, interest should only be charged up to settlement date plus possibly an early termination fee if any). Mark loan as Closed, create a payment entry for the settlement.
  Output: Settlement statement or confirmation.

- `POST /api/loans/{loanId}/reschedule`: *Restructure a loan.*
  Input: new terms (e.g., extending term, new installment plan).
  Process: Only allowed for certain roles, recalc schedule, update status perhaps to reflect restructured. Keep history if needed. (This is advanced and maybe optional, but listed for completeness.)

- `POST /api/loans/{loanId}/writeoff`: *Write off a loan.*
  Process: Marks loan as WrittenOff (closed for further collection) with write-off date and reason. Triggers any required accounting entries (if integrated to accounting). Only high-level roles can do.

A5.4 Payment & Reconciliation APIs:

- `POST /api/payments/collectDue`: *Initiate collection of due payments (automatic process).*
  This could be a Cloud Scheduler triggered function (no direct user input). It will:
- Find all installments due today (or overdue and scheduled for retry).
- For each, call the respective integration (Allps/NuPay) to initiate a debit, or if using a more manual process, compile them.

- Mark in the `payments` collection a pending entry. This might not be an externally called API but rather an internal job.

- `POST /api/payments/manualReceipt`: *Record a manual payment.*
  Input: `{ loanId, amount, date, method, reference }`.
  Process: If a client paid cash or EFT outside the automated system, a user (cashier or admin) can record it. The function will:

- Create a `payments` record (type "Repayment", method as provided).
- Update the loan's outstanding balance and next due info.
- Mark relevant installment(s) as paid or partially paid.

- Trigger a receipt generation (could email receipt to client). **Output:** Payment recorded confirmation.

- `GET /api/payments?loanId=XYZ`: List payments for a loan (or possibly this is just fetched as subcollection).

- `POST /api/payments/reconcile`: *Reconcile payments from provider.*
  This could handle input from a file or webhook:

- For PayFast/Ozow: a **webhook endpoint** (e.g., `POST /api/webhook/payfast`) will receive confirmation of payment status. It will parse the message (with security checks like signature), find the corresponding loan/invoice, mark payment success or failure in our system (update `payments` record, adjust loan, etc.).
- For debit orders (Allps/NuPay): Typically, the results of debit orders come as a report at end of day. We might have to upload a return file or use an API call to fetch results. If Allps/others provide a webhook/callback, we use that. Otherwise, an admin might upload a CSV of results to a UI that calls an API like `POST /api/payments/uploadResults` with the file. The system then processes each line, updating each payment status accordingly.

- These reconciliation processes will also handle partial payments or fees if returned (e.g., note if a debit failed due to insufficient funds, possibly schedule a retry or add a flag).

- `POST /api/payments/{paymentId}/retry`: *Retry a failed payment.*
  Process: Mark the previous payment as failed (if not already), then essentially call the debit order again or schedule for next available date. Could allow a collector to manually trigger an out-of-cycle debit attempt via integration (some systems allow ad-hoc debits).

A5.5 Communication APIs:

- `POST /api/notify`: *Send a communication to a client.*
  Input: `{ clientId or loanId, type: ("SMS"|"Email"|"WhatsApp"), templateId, placeholders }`.
  Process: Looks up client contact info, loads the message template (from templates collection or config), fills in placeholders (like {name}, {amount}, {dueDate}), sends via appropriate provider API (Twilio for SMS/WhatsApp, or SMTP/SendGrid for email). Logs the communication in a history (maybe in a subcollection `clients/{id}/communications`).
  Output: Success or error from provider. This API would be used for manual sends (e.g., staff clicking "Resend welcome email"), whereas most routine communications are automated by triggers.

- **Templates Management:**

- `POST /api/templates` to create a new message template (Admin only).
- `GET /api/templates?type=SMS` list templates.

- `PUT /api/templates/{id}` update, etc.

- `POST /api/notifications/pushTest`: Possibly an API to test push notifications to a device, if we use FCM for the mobile app.

A5.6 Document Handling APIs:

- `POST /api/clients/{clientId}/documents/upload` (or an endpoint to get a signed upload URL for direct upload):
  Input: file binary + metadata or request for URL.
  Process: Usually, we use Firebase Storage SDK directly from frontend to upload and then call a function to save metadata. But if using an API:
- Accept file upload (maybe via an HTTPS function).
- Store in Storage (generating a filename perhaps containing clientId for structure).
- Create a metadata record in Firestore (in `documents` or `clients/{id}/documents`).

- If document type is something like ID or proof of address and we have an OCR step, trigger an OCR function to extract text and compare (for verification). **Output:** Document ID/URL.

- `GET /api/clients/{clientId}/documents`: *List documents for a client.*
  Output: List of docs with their metadata and URLs (possibly time-limited download URLs if we want to avoid public access). The actual content is fetched from Storage with a secured link.

- `POST /api/clients/{clientId}/documents/{docId}/verify`: *Mark document as verified.*
  Process: The compliance officer or admin uses this to approve a KYC document after review. It updates the document record (`verified=true, verifiedBy, verifiedAt`). If the doc was an ID or address proof, also update `clients` collection fields like `FICA.verified=true`.

- `POST /api/loans/{loanId}/documents/generateContract`: *Generate loan agreement document.*
  Process: Takes loan data and produces a PDF (maybe using a template with placeholders for client name, loan terms, etc.). Could use a library like PDFKit or an HTML to PDF service. Store the PDF in Storage and attach to loan's documents. Optionally, if e-sign is to be done by client, we might use an external e-sign service or embed a signature field to be signed on device. For now, at least generate and store.
  Output: Contract document URL.

A5.7 Reporting & Analytics APIs:

- `GET /api/reports/portfolioSummary`: *Portfolio summary data.*
  Output: Key stats: number of active loans, total principal outstanding, total arrears, etc. Possibly grouped by branch or product. This data might be pre-aggregated via Cloud Function scheduled daily or computed on the fly via queries.

- `GET /api/reports/collectionsSummary`: e.g., total overdue, recovery rate, etc.

- `GET /api/reports/ncrSubmission`: Perhaps produce data needed for regulatory submission, like a list of all loans booked in a period, etc., in required format (CSV). This could gather data from Firestore and format as needed.

- `GET /api/reports/transactionHistory?clientId=X`: Payment history for a client, etc.

(Reports may also just be generated client-side by querying the data. However, heavy aggregation (like computing aging of accounts, or cohort analysis) might be done on server side due to limitations of Firestore queries. In such cases, we'll use Cloud Functions to perform aggregation (maybe writing results to a `reports` *collection or returning on request).)*

A5.8 Risk & AI APIs:

- `POST /api/risk/pullCreditReport`: *Initiate a credit bureau pull for a client.*
  Input: `{ clientId, bureau }` (bureau might be "TransUnion" or "Experian").
  Process: Cloud Function calls external API with client's ID number and returns credit data. Save the result (score, summary) in `clients/{id}/creditReport` and possibly store the full report (maybe in a separate secured storage or as a file, since credit reports can be lengthy). Mark that a credit check was done at a certain time (for compliance record).
  Output: Credit score or report summary.

- `POST /api/risk/scoreApplication`: *Calculate internal risk score for a loan application.*
  Input: `{ loanId }`.
  Process: Could run an internal logic or machine learning model to produce a risk grade or probability of default. Uses data like credit score, client's history (if repeat borrower), and application details (income, loan amount). The result might be a score (e.g., 1-100) or category ("Low/Med/High risk"). Save to the loan record (e.g., `riskRating`). Possibly auto-approve or decline based on thresholds (with override possible).
  Output: Risk score and decision recommendation.

- `POST /api/ai/trainDefaultModel`: *Train an AI model on past loans to predict default.*
  Process: This would gather historical loan outcomes and train a model. Likely done offline or as a long-running function. For manual, we note that such capability exists, but implementation might be beyond immediate scope. Output would be a model stored (if using AI Platform).
  Output: Model info or success.

- `POST /api/ai/getRecommendations`: *Get AI-based recommendations.*
  This could yield things like "client X is eligible for an upsell" or "loan Y shows signs of potential default risk, consider action". It would query patterns (maybe using ML results or simple rule for v1).
  Output: List of recommendations (each with a type and message, e.g., `{"clientId": 123, "recommendation": "Offer top-up loan of R500"}`). These can be displayed on dashboards.

A5.9 Admin & Configuration APIs:

- `POST /api/config/loanProducts`: Create a new loan product config (Admin only).
  Input: product details as in schema.
  Output: New product ID.

- `PUT /api/config/loanProducts/{id}`: Update product (e.g., interest rate if regulation changes, ensuring no impact on existing loans except maybe future ones). Possibly versioning might be needed if terms change, but keep it simple.

- `GET /api/config/*`: Various gets (list branches, list roles, etc. for dropdowns in UI).

- `POST /api/admin/generateReports`: A trigger to generate certain reports or backups on demand.

- `POST /api/admin/exportData`: Perhaps export all data for backup/compliance (POPIA data access requests) – this could gather all info related to a client to fulfill a data access or deletion request.

Every API will be secured by Firebase Authentication (and Firebase will pass an `Auth` token to Cloud Functions). The Cloud Function will verify the user's role/permissions server-side as well, not relying solely on the front-end checks. This double enforcement (client-side via UI hiding and security rules / server-side via token claims) ensures **strict access control**, satisfying both security best practices and compliance requirements for protecting personal data [12].

Moreover, each important API action will create entries in the **auditLogs** so we have a record of who did what and when (especially for things like approvals, data changes, login attempts). The system's API design thus balances *ease of use* (for the front-end to call and integrate) with *robust security and logging*, providing a transparent and accountable system.

SECTION B: DETAILED FEATURES INVENTORY

In this section, we detail each functional module of the LMS, enumerating all features, screens, fields, user interactions, and compliance checks relevant to that module. Each subsection corresponds to a major component of the system. The approach is exhaustive: **every user right, field, and process is explicitly described**, ensuring Codex (and developers) have a clear spec to implement. We also highlight how modules interact and where regulatory compliance steps must occur.

B1. User Management System

The User Management System handles **authentication, user profiles, roles, and access control**. It ensures that only authorized personnel can access the LMS and that each user's capabilities are limited to their role's permissions. This module will be developed first, as it forms the security foundation for all other features.

Key Features: - User registration (by admins), authentication (login/logout), and password management. - Definition of user roles and granular permissions. - Role-based access control in the UI and backend (which menus/actions each role sees and can perform). - User profile management and audit of user activities. - Multi-factor authentication (for added security, optional). - Device management (optional: if needing to restrict which devices can be used, could integrate, but not a core requirement unless specified).

User Roles and Permissions: Each role in the system comes with specific rights. Below is an explicit list of roles and what each can do:

- **Administrator (Admin):** Full access to all system features. Admins can manage users (create, edit, deactivate accounts), configure system settings (all Admin Tools in B13), view and edit all data (clients, loans, etc.), and perform high-level actions (approve loans, write-off loans, view all reports). Essentially, Admin can do everything, including tasks of all other roles. Typically reserved for a few senior users.
  Permissions: Create/edit/delete users, assign roles; configure products and integrations; override any restrictions; view all branches; approve large loans; access audit logs; initiate system-wide actions.

- **Branch Manager:** (If branches are used) Oversee operations at a specific branch. Can do almost everything an Admin can but limited to their branch's data. For example, a Branch Manager can approve loans up to a certain amount for their branch, view branch reports, and manage branch staff accounts (except maybe not create new Admins). They cannot change global settings or view other branches.
  Permissions: Approve/decline loan applications for their branch, manage clients and loans within branch, view branch analytics, add new loan officers or collectors (but maybe not other managers or admins), etc.

- **Loan Officer (Credit Officer):** Responsible for loan origination and client management. Loan Officers can create new client entries, input loan applications, and see through the loan evaluation process (though they might not have final approval authority depending on policy). They can upload client documents and run preliminary affordability checks or credit checks if allowed. They cannot manage other users or alter system settings. They only see clients and loans relevant to their branch (or to them, depending on assignment).
  Permissions: Create/edit client profiles; initiate new loan applications; view their own pipeline of applications; upload documents; (possibly) run credit checks; see credit reports for their clients; save loan recommendations. They **cannot** approve loans beyond maybe recommending, cannot see other branches' info, cannot manage users or configurations.

- **Risk Officer / Underwriter:** (If separate from loan officer) Focus on risk assessment. They can view applications marked for risk review, access full credit bureau reports, and input decisions or recommendations. They might update risk scoring fields and either clear the app for approval or flag for rejection.
  Permissions: View all pending applications (or those assigned for risk review); pull credit reports via integration; update risk assessment fields (like risk rating, recommended decision); cannot final approve if policy requires manager sign-off, but can mark an application as recommended approve/reject. They cannot change core client info (maybe read-only on that), cannot manage users, etc.

- **Collections Officer:** Manages overdue accounts. They have access to the Collections module. They can view loans in arrears (probably only for their branch if applicable, or all if central collections team). They can update collection tickets (log calls, promises, outcomes), send communication to delinquent borrowers (SMS, call via the system, etc.), and recommend further action (e.g., handover to external collections). They typically cannot create or approve new loans or edit client info, except updating maybe contact details if discovered during calls.
  Permissions: View list of overdue loans; access each collections case; add notes and mark follow-ups; change collection status of a loan (e.g., from active to legal handover); trigger reminder messages; view payment history for loans in collections. They **cannot** modify loan terms or create loans, and cannot see non-delinquent loan details beyond what's necessary.

- **SuperUser / System Admin:** (Optional role if we differentiate from Admin) Could be used for technical maintenance users with access to everything including raw data (for troubleshooting). Possibly combined with Admin role if not separate.

- **Auditor / Read-Only:** (Optional) A role that can view all data but not edit. This might be used for internal/external auditors. They can run reports and view client/loan details but cannot change anything.
  Permissions: Read-only access across system or within scope, can export data, view logs.

- **Client (Borrower) Self-Service:** Although not a staff role, if clients use a mobile app or portal, they have a role in Auth. This role is very restricted: they can only access their own data (their profile, their loans), and perform actions like applying for new loan, viewing statements, making payments via provided channels. They obviously cannot access other clients' data or internal screens. Their experience is separate (the mobile app context in B12).
  Permissions: Register/login to their account; update limited personal info (perhaps address, contact details); apply for loans; upload documents; view loan status and balances; make payments via integrations; receive notifications. They cannot see internal notes, internal workflows or any other client data.

Each user account will be assigned one (or multiple, if needed) roles. Roles are enforced at multiple levels: - **UI Level:** The frontend will show/hide menu items and buttons based on the user's role. For instance, only Admin sees the "User Management" menu, only Loan Officers see the "New Loan Application" form, etc. - **Route Guarding:** We set up route guards (in Angular, e.g., `canActivate` guards) to prevent navigation to unauthorized routes. If a non-Admin somehow tries to navigate to the admin URL, they'll be blocked. - **Backend Security:** Regardless of UI, the backend (Firestore rules and Cloud Function checks) will enforce permissions. E.g., Firestore rules might allow a loan officer to write a loan application but not to alter loan approval status field (which only manager or system can). Similarly, a collections officer can update collections subcollection but not modify the core loan amount. We will carefully design these rules to mirror the permissions above.

Authentication Flow: The LMS uses Firebase Auth for login: - Users log in with their email and password (set up when their account is created). On the first login or when password is reset, they must set a new password. - We implement **"Forgot Password"** functionality: on the login screen, user can click "Forgot Password", enter their email, and the system emails them a reset link (via Firebase's built-in email template, customized with company branding). - Optional **Two-Factor Authentication (2FA):** For enhanced security (especially for Admins or sensitive roles), we can enable 2FA. If enabled, after password login, the system

sends an OTP (one-time PIN) via SMS or email which the user must enter to complete login. We'll integrate this via Firebase Auth multi-factor or our own OTP service if needed. (This is recommended under POPIA and general security best practices for systems hosting sensitive data). - After successful login, the user's session token includes their role info (Firebase custom claims or a roles field we attach). We store minimal session info on client side (maybe JWT in memory or local storage for re-auth). The session may timeout after a period of inactivity (we can rely on Firebase refresh tokens which expire, and explicitly implement an auto-logout after X minutes of inactivity for security). - **Device Trust (optional):** If required, we could implement device recognition (like sending an email if a new device logs in, or allowing Admin to see active login devices). Not a core feature, but worth noting as a future enhancement for security.

User Interface for User Management: - **Login Screen:** Fields: *Email*, *Password*, and a "Login" button. Also a "Forgot Password?" link. Simple and branded with the company logo. Validation: requires a valid email format and non-empty password. On incorrect credentials, show a clear error message. If 2FA is enabled, after hitting Login, show a screen to enter OTP (with option to resend OTP). - **User Dashboard:** After login, user lands on a Home/Dashboard showing summary info (like tasks or relevant overview). For now, relevant to user management, maybe a welcome and quick links. (Detailed dashboards come later in Reports section.) - **User Management Screen (Admin Only):** - A *User List* view: table of all users with columns: Name, Email, Role, Branch, Status (Active/Inactive), Last Login. Support sorting and filtering (e.g., show only active users, filter by branch or role). Each row might have actions "Edit" and if permitted "Deactivate/Activate". - An *Add New User* form: fields: **Name**, **Email**, **Role** (dropdown of roles), **Branch** (if applicable, dropdown of branches), **Temporary Password** (option to either auto-generate or let Admin set a password). Possibly we auto-generate and email the new user an activation link. The form has validation (unique email, required fields). - An *Edit User* form: similar to add, but also can toggle Active status, and possibly a "Reset Password" button that triggers a password reset email to that user. - Deactivation simply sets `isActive=false` on the user; deactivated users cannot log in (Firebase Auth custom claim or check in our code). We won't physically delete users unless necessary (to keep history). - If we allow role changes, Admin can update a user's role here. Changing roles should update their Firebase claims immediately (we'll implement a function or use Firebase Admin SDK to set custom claims). - **Profile Settings (All Users):** A screen where a logged-in user can view and edit their own profile: e.g., change their display name, change password. Changing password uses Firebase Auth's functions (which require re-authentication). This screen might also allow toggling 2FA if implemented (like enable/disable OTP for login). - Possibly a **Roles Management** UI (Admin Tools): If roles are not fixed but configurable, an Admin Tools screen to create roles and assign permissions. However, in our design, roles are mostly fixed by code. We might not include a UI for editing permissions for simplicity (given explicit listing we have), unless we want to be extremely flexible. We'll assume fixed roles for now to avoid complexity and potential security holes from misconfiguration.

Access Control Implementation: - In Firestore Security Rules: We will implement rules such as: - Only Admin can read/write `users` collection (non-Admin maybe can read their own user doc if needed). - In `clients` collection, a loan officer can write if `resource.data.createdBy == auth.uid` for new clients, or if in same branch for existing – or better, we assign branch in user claims and ensure `client.branchId == user.branchId` for reads/writes except Admin which can access all. - For loans, similar approach: an officer can create if they own or same branch, but cannot approve (the `status` field transition to "Approved" might be blocked unless user has role Manager). - Using custom claims: e.g., `request.auth.token.role == "Admin"` in rules to allow certain actions. - We'll also use Cloud Functions as a gatekeeper for some actions (like approve loan) to have more complex logic rather than solely relying on rules. - Frontend will use route guards as mentioned: e.g., a guard that checks

`currentUser.role` and if not allowed, redirect to not-authorized page or home. Also, each component can have an `*ngIf` around sensitive buttons (like only show "Approve" button if role is Manager). - UI will provide feedback: if a user somehow tries an action they're not allowed to (maybe by a crafted request), the backend will reject (security rule or function throws error), and we'll catch that and show an error "You are not authorized to perform this action."

Audit & Logging: We maintain logs of user activities: - Log every login (success or failure) with timestamp and user ID (store in `auditLogs` and possibly increment a "login count" in user profile). - Log creation of new users or changes in roles (with who did it). - These logs are available to Admins perhaps in an Audit screen (or at least exportable if needed). - This helps in monitoring and is a POPIA requirement to track access to personal data [9].

Compliance Considerations (User Management): - **POPIA:** We must protect personal information of users (though users are staff, we still treat their data carefully). Passwords are never stored in plaintext (Firebase Auth handles hashing). Any personal info (like user's own phone or email) is kept secure. If a staff user leaves (and is deactivated), we retain minimal data (like their name on past audit logs) but can anonymize if required over time. Also, ensure user accounts have least privilege – by defining roles strictly – to comply with the principle of minimum necessary access to personal client data [12]. - **Access Control:** It's a violation if, say, a loan officer could see loans from another branch without need – that could expose client info unnecessarily. Our role-based filters ensure users only access data relevant to their role, satisfying POPIA's need-to-know principle and NCR's expectation of controlled access to credit data. - **Security:** Multi-factor auth is recommended for admin accounts because a breach of an admin account can expose all client data. We'll implement it or leave the hooks to implement easily. Password policies (min length, complexity) will be enforced when setting passwords. - **Audit:** Having an audit trail of user actions (especially any data exports or deletions) addresses compliance by showing accountability in handling sensitive data [9]. - **Device Security:** If the LMS is used on mobile devices, ensure the app has appropriate security (e.g., auto-logout on inactivity, ability for admin to remotely sign out a user if device lost, etc.). And advise users not to share accounts.

With the User Management system properly implemented, we set a secure stage for building all other features. Next, we address client management, which will build on the user roles defined here (e.g., only certain users can add clients).

B2. Client Management System

The Client Management System handles all information and actions related to **borrowers (clients)**. This module is crucial as it feeds the loan origination process with verified customer data. It includes capturing extensive client information (personal, contact, employment, financial), updating that info, and verifying compliance requirements like KYC (Know Your Customer). It ensures a single, clean record for each client, which is referenced by their loans.

Data Fields (Client Profile): The system maintains a comprehensive profile for each client. All fields captured and stored were outlined in the schema (A4), but to reiterate in the context of UI:

- **Personal Details:** Title (Mr/Ms), First Name, Last Name, Date of Birth, Gender, National ID number (13-digit South African ID or passport number if foreign). The ID number field has validation: exactly

13 digits for SA ID and passes the checksum logic; if passport, we allow alphanumeric and require capturing nationality. There's also an optional field for **Marital Status** (if needed for affordability).

- **Contact Details:** Mobile Phone (required), Alternate Phone (optional, could be home or work number), Email Address (optional but good to have), Physical Address (residential). Address will be broken down into: Street Address, City, Postal Code, and perhaps Province. Possibly also a Postal Address if different (for mailing statements, though likely not used if we go fully electronic). For compliance (FICA), a *Proof of Address* document will be required later to verify this address.
- **Employment & Income:** Employment Status (dropdown: Employed full-time, Part-time, Self-Employed, Unemployed, Pensioner, etc.), Employer Name (if employed), Employer Contact (maybe telephone or HR email, optional), Monthly Income (numeric, net income after tax), Income Frequency (e.g., Monthly, Weekly if applicable), Next Pay Date (for scheduling first debit – many lenders align first debit to pay date). Also Monthly Expenses (numeric, an estimate of client's total monthly expenses). The system might break expenses down (rent, transport, etc.) or take a single figure for affordability calculation. To be thorough, some lenders capture number of dependents, which can factor into affordability. We could include *Number of Dependents* as a field and possibly *Household expenses* separate from personal, but to keep it manageable, we can just use an aggregate expense figure provided by the client.
- **Banking Details:** Bank Name (dropdown of major banks: ABSA, FNB, Standard Bank, Nedbank, Capitec, etc., plus "Other"), Account Number, Account Type (Savings/Cheque(Current)/Transmission), Branch Code (or a universal branch identifier, though for big banks a generic code is fine; but we'll capture it for smaller ones). These details are needed for both disbursement (to pay out the loan into their account) and setting up debit orders for repayment. We will ensure the system can integrate with account verification (some services can verify that the account belongs to the client's ID – but that might be a later enhancement). At minimum, a signed debit order authorization or DebiCheck mandate will be needed; the system will produce one in the loan agreement.
- **Identification & Documents:** While documents themselves are handled in B9, from the client profile perspective, we store references or statuses: e.g., "ID Copy Received (yes/no)", "Proof of Address Received", "Payslip Received" (if income needs verification). These could be boolean flags or a status field that automatically updates when a document is uploaded and marked verified.
- **Additional Info:** Loan Purpose (if collected at client level or in each application? Typically per application, but we might also record general intent of borrowing if needed). Also, some micro-lenders record things like how the client heard about them (marketing source), which can be a field for reporting. It's optional (e.g., Source: Walk-in, Referral, Online).
- **Consent and Declarations:** There should be fields indicating the client has agreed to certain terms:
- Consent to credit check (must be true and timestamped; ideally a digital signature or recorded proof is kept – e.g., they tick a box or sign a form which we store).
- Consent to data processing (POPIA consent: acknowledging their data will be used for loan processing). Possibly part of the application form terms.

- Marketing consent (whether they agree to be contacted with promotional offers). These can be stored as booleans and date of consent.

- **Client Status:** An internal status flag: e.g., Active (eligible), Inactive, Blacklisted, Under Debt Review.

- If a client goes under debt review (a legal process in SA where they consolidate debt, no new credit should be given), we might mark them accordingly. This info can come from credit bureau or client declaration. If flagged, system should prevent new loans to them and maybe mark existing loans for special handling.

- Blacklisted might mean we internally decided not to lend to them (e.g., due to fraud or prior default).
- Deceased could be another status if that info is available.
- These statuses heavily influence what can be done (system should block creating new loans for blacklisted/debt review, etc.).

All these fields appear on the **Client Detail Screen**, which is the central place to view/edit client info.

User Interface & Flows:

- **New Client Creation:**
- A form for loan officers or any authorized user to **Add New Client**. This form will contain all the personal, contact, employment, and banking fields above. It might be divided into sections (tabs or accordion): Personal Info, Contact, Employment & Finance, Banking, Documents.
- Validation is key: The form will enforce correct formats (e.g., ID number 13 digits and likely perform Luhn check or date-of-birth match within ID number; email format; phone number digits; no letters in numeric fields; etc.). It will also ensure required fields are filled (first name, last name, ID, mobile, etc. are mandatory). For numeric fields like income, disallow negatives, maybe set reasonable ranges (e.g., income must be above 0).
- On submitting a new client, the system checks if the ID or mobile already exists: If yes, it should alert user "Client with this ID or phone already exists" to avoid duplicates (the user could then cancel and search for that client). This check can be done by querying Firestore for matching ID (which we'll index).
- If unique, it saves the client to Firestore. If documents are being uploaded at the same time (we might allow attaching files in the add form), those get uploaded and linked.

- Post-creation, perhaps automatically navigate to the new client's profile page.

- **Client Profile Page:**

- Shows all the client's details in read-only mode initially, with an "Edit" button to enable editing.
- It also contains sub-sections or tabs:
    - **Profile Info:** displays the fields nicely (maybe grouped by Personal, Contact, Employment, Banking).
    - **Documents:** listing all documents related to client (IDs, proof of address, etc.) with status (verified or not). This section will allow uploading new docs or viewing existing ones (opening an image or PDF in a modal).
    - **Loans:** a list of all loans that this client has (with key info: loan ID, date, amount, status, balance). Possibly separate lists for Active vs Closed loans. Each entry might link to the loan detail page.
    - **Credit Checks:** (if we store history) we can list any credit bureau checks run for this client: date, bureau, score, summary.
    - **Notes:** freeform notes section for any miscellaneous remarks (e.g., "Client prefers communication via WhatsApp", or internal warnings). Loan officers or managers can add notes here. Possibly timestamp each note with user.
- The profile page allows editing: user clicks "Edit", then fields become editable (with similar validation as creation). Some fields might be locked from edit after initial entry (like ID number should rarely change – if it was input wrong or client got an ID change due to error, we might allow admin to change but normally not easily editable by officers).

- After editing, saving updates the Firestore doc and possibly triggers verification flows if needed (e.g., if they changed address, maybe mark proof of address as required again).

- The UI should highlight if something is missing: e.g., if no ID doc uploaded yet, show a warning "ID document not on file" perhaps in documents section.

- **Search and Selection:** A **Client Search** feature will be provided so that staff can quickly find a client:

- A search bar on top of client list or a dedicated search page. You can search by ID number, name, or phone. Ideally it supports partial name search (which might require an external search index if needed). At minimum, entering an exact ID or phone will find the client. For names, we might allow starting letters search if we index name fields with prefixes.
- The result is a list of matching clients (showing maybe Name, ID, phone, status, and maybe branch). Clicking one opens the profile page.

- This search is crucial to avoid duplicate creation and to speed up loan origination (officer should search client first, if found, then directly start loan application for them).

- **Client List View:** In addition to search, an Admin or Manager might list clients (e.g., to see all clients in the branch). We could have a paginated list sorted by last name or ID. But since client base could be large, we likely rely more on search or filters. If implemented, it should allow filtering by status or creation date, etc.

- **Editing & Role Permissions:**

- Creating a client: allowed for Loan Officers, Branch Managers, Admin. Perhaps not allowed for Collections or Risk roles (they consume the info, not create).
- Editing client info: allowed for Loan Officers (to update contact info etc.), but possibly restrict certain fields to management (e.g., only a manager can flag a client as blacklisted or change a status to active again). So if a loan officer tries to mark a client "Blacklisted", it might be disabled for them – only Admin can do that via a separate action (maybe on profile page an Admin sees a "Blacklist client" button).

- Viewing clients: any user with reason should view. Loan Officers can view their branch's clients. Collections can view clients in collections obviously (they need contact info to call). Risk officers can view all clients with pending applications. We ensure through role checks that one can't randomly browse clients outside their scope.

- **Integration within Loan Origination:**

- When starting a new loan application (B3), if the client exists, the officer will search and select the client. If not, they will create a new client as above and then proceed. We can streamline that: e.g., a "New Application" button might first prompt to search or create client.

- Once an application is created for a client, that link is permanent (loan references the clientId).

- **Data Integrity and Duplicate Handling:**

- If duplicates slip in (maybe slight name differences or ID entry mistakes), the Admin should have a way to merge or mark duplicates. A *Merge Client* tool could be in Admin Tools to consolidate records (this is advanced, but to surpass others, mention it). Merging would move all loans from one client to another and remove the duplicate. We will not implement initially but design such that if needed, it's possible (like loans reference clientId, so one could update those references).
- For sanity, we rely on ID number as a unique key. If two different people share something (like same name), ID distinguishes. Only scenario might be client doesn't have SA ID (like immigrants) – but then passport and maybe date-of-birth combination can be used for uniqueness.

Compliance Considerations (Client Management): - **FICA (Know Your Customer):** Before any loan is granted, FICA law requires verifying client's identity and address: - We must **collect ID document** (SA ID book/card or passport). The system facilitates uploading the ID doc (which will be done typically at origination or client onboarding). A staff member must then verify that the ID matches the person (if in person) or at least that it's a valid ID copy. We mark `ID Verified = true`. Optionally, integration with Department of Home Affairs could verify ID number validity or that name matches ID (some services or via credit bureau). If available, we can integrate that into risk checks. For now, manual verification is assumed. - We must **collect Proof of Address** (utility bill, etc. not older than 3 months). The system allows uploading such a document. We track that we have it and it's verified. This satisfies anti-money-laundering address verification. - We should record the **method of verification**: e.g., "ID verified by John Doe on 2025-07-01 against original document" in notes. - If any required doc is missing, the system should flag it and possibly prevent loan approval until resolved. - **POPIA (Data Privacy):** - When capturing client data, we must ensure we have the client's consent to store and process it. Typically, the loan application form (digital or paper) will have a clause the client agrees to. We store that consent as a boolean and timestamp as mentioned. If a client ever requests to see what data we have on them or to delete it (a right under POPIA), the system should be able to retrieve all their stored info easily (maybe via a report) and accommodate deletion if no legal reason to retain. Likely we keep data of loans for at least the retention period required by NCR (which might be 5 years after account closure). - Also, we must safeguard this data: ensure only authorized access (which our role controls do). We should also mask or not display full ID numbers on screens unnecessarily - for example, some systems display ID as `******0857` (last 4 digits) unless a user clicks to view (and only if they have permission). We might implement masking for highly sensitive fields on the UI to minimize shoulder-surfing risks. - **NCR (National Credit Act):** - One aspect: making sure the client is not under debt counseling or has multiple concurrent short-term loans that could make lending reckless. The credit report integration will usually show if the person is under debt review or has other loans in arrears. We incorporate that in risk, but at client management level, maybe a field "Under Debt Review" can be set if we know it, to block new loans. - Another aspect: the act forbids discrimination – ensure we gather necessary info but not using it in a prohibited way (we are not, we're just storing). - The client's credit profile (like credit score) is personal information – showing it around should be limited. Only roles that need to see credit score (risk officers, approvers) should see it. We implement such restrictions. - **Affordability assessment input:** The captured income and expense data for the client is critical for calculating affordability. Regulations require that we obtain **proof of income** for a loan (like payslips or bank statements for big loans). Our system will ensure that before approval, for employed clients, at least one *Payslip document* is uploaded. For self-employed, perhaps a bank statement or financials. We'll reflect that requirement in compliance checks (Loan Origination will check docs). - Also by capturing dependents and expenses, we can compute a **disposable income** figure. E.g., Disposable = Income - Expenses - current debt obligations. The loan installment must be such that disposable remains >= some threshold. NCR doesn't give a fixed formula, but guidelines must be set by each credit provider. We'll for example enforce that after paying all debts including the new loan, the client should have at least e.g. 20% of income remaining or some realistic amount. This can be a parameter that Admin sets and risk module uses. - **Record Keeping:** According to

regulations, we need to keep client records and documents for a certain period (I believe at least 5 years from loan closure or account closure). Our system's data retention policy (could be a background function) might keep data indefinitely unless purged. But POPIA says not to keep longer than needed. We'll likely not implement auto-deletion in code, but mention that policy exists and can be executed if needed. We ensure backups and that deletion, when required (like client requests after that period), is possible (Admin could delete or anonymize the client record and purge documents).

Interactions with Other Modules: - Client management feeds into **Loan Origination** (B3). The loan module will retrieve client info like income to use in calculations. We ensure that whenever a loan is being processed, it pulls the latest client data (or references it). - It also interacts with **Risk** (B4) – e.g., a Risk officer might update something on client like marking them as high risk or adding an internal credit score. Possibly after a bad experience, risk could blacklist a client, which would update client status. - With **Collections** (B5), collectors use client contact info to get in touch. If a client updates their phone or address, collectors see the updated info. Collectors might also add alternate contact details (like a new phone number found) – perhaps the system allows adding multiple phone numbers or a "contacts" sublist (like next of kin contact which some credit apps ask for). We did not list next-of-kin earlier but often micro-loan forms have fields for a **Referee or Next of Kin** (someone not living with the client). That could be additional fields or even a subcollection of references (name, relation, phone). This info is used if client disappears. We might include at least one alternate contact field in client info. - With **Communication** (B8), the client's preferences (SMS, email opt-in) are considered. The communication module will use client's stored email/phone to send notifications. If client opts out of marketing, the system should exclude them from bulk campaigns (not asked but good practice). - **Admin Tools** (B13) might have settings influencing client module (e.g., configurable mandatory fields, or the list of allowed ID types, etc.). - **AI** (B15) could analyze client data to give recommendations (like "this client has taken loans consecutively, maybe offer a product upgrade").

By implementing the client management system with the above detail, we ensure the LMS has a solid, compliant customer database. Next, we proceed to the **Loan Origination System**, which uses these client records to create and process loans.

B3. Loan Origination System

The Loan Origination System covers the end-to-end process of taking a loan request from initial application through to approval (or rejection) and readying it for disbursement. This is the core operational workflow of the LMS. We will detail capturing loan details, underwriting steps, approval flows, and how the system ensures consistency and compliance at each step.

Loan Application Workflow: The typical stages in origination are: 1. **Loan Application Entry:** A loan officer (or the client via self-service) fills out a loan application form for a specific client. 2. **Preliminary Assessment:** System calculates installments, checks basic criteria (e.g., client's affordability, any immediate disqualifiers). 3. **Credit & Risk Checks:** (Detailed in B4, but part of origination flow) – obtaining credit reports, scoring. 4. **Approval Decision:** A user with approval authority (manager) reviews and approves or declines the application. 5. **Post-Approval:** If approved, preparing for disbursement (payout) and moving the loan to active status (which then falls under loan servicing). 6. **Loan Agreement Signing:** Getting the client's acceptance/signature on the terms (this can be integrated if digital, or assumed manual signing that is then uploaded).

We will address features in roughly this order.

Loan Application Form (New Loan Request): This is a crucial UI where all loan details are captured: - Accessible via a "New Loan" or "Apply Loan" button from either the client's profile (preferred, so client is pre-selected) or from a main Origination menu where first you choose a client. - If started from scratch, the first step is selecting or creating a client (we might have a client search modal, or instruct user to create client first). - Fields in the loan application form: - **Client:** If not already selected via context, choose the client (by search, then lock it in). - **Loan Product:** Choose from available loan product types (populated from `loanProducts` config). Once a product is selected, some fields might auto-set or restrict (like max amount, allowed terms). - **Loan Amount (Principal):** Input field for desired amount. Should enforce min/max of selected product, and maybe round to nearest whole Rand (no cents usually for simplicity). - **Term:** Input the term of loan. Possibly in months (for personal loans) or days/weeks (for short-term). We will unify to months for UI but allow e.g. 1 month = 30 days loan. Alternatively, have separate UI for short-term vs installment loans. To keep it simple, term in months, but for 1-month payday loan, just select 1. If product is short-term, maybe we restrict term to 1 or 2. - **Interest Rate:** This might auto-fill from product (or product has standard rate). If rate can vary per client or promotion, allow edit if user has permission (maybe managers can adjust interest within legal limits). Typically short-term loans have fixed max rates (5%/month or so), and personal loans might vary with credit score. In advanced scenario, system could decide interest based on risk (for now, assume fixed per product). - **Fees:** Initiation Fee and Service Fee should calculate automatically based on amount and term as per product config and legal formula. For example, if a R1000 loan, initiation fee might be R165 + 10% of (1000-1000) = R165 (with cap R1050) [13]; service fee e.g. R60/month regulated max. The form should display these fees for transparency. The fees can be included in the loan or added on top – typically initiation fee is capitalized into loan principal or deducted from payout; we need to clarify how it's treated (commonly, initiation fee can be added to amount financed or borrower pays upfront; we might choose to finance it as part of loan). - **Repayment Schedule:** As soon as amount, term, interest are input, system computes a draft repayment schedule. If it's an installment loan (term > 1 month), compute equal monthly installment using amortization formula (taking into account interest and fees). If it's a single-payment loan (e.g., 1 month), then schedule is one payment of principal+interest+fees at due date. We generate schedule entries (maybe not all shown in the form if too many, but at least show the monthly installment amount and number of payments). Possibly show a table if term is small or just a summary if term is long. For example: - If 6 months loan, show monthly installment = X Rand and maybe a small table with each month's due date and amount. - If short-term, show "Due on [date]: [amount]". - **Purpose of Loan:** (optional but many forms have) Dropdown (e.g., "Education, Rent, Business, Medical, Other"). If product requires capturing it for NCR purposes. - **Payment Method:** How will the client repay? Typically default is Debit Order. Could also be "Cash at branch" or "Salary deduction" if applicable. Choose an option (populated from `repaymentMethods`). If debit order, ensure we have bank details (the system will use client's bank info already on file). - **Disbursement Method:** How funds will be given to client. Likely default is EFT to their bank (which we have). Could be cash (then branch will give cash) or e-wallet. This might be just informational but triggers different processes (e.g., if PayFast integration for instant pay to bank card? Not common, but maybe just EFT). - **Collateral / Guarantor:** Usually microloans are unsecured, but if the product involves collateral (like maybe they pawn something or have guarantor), we could have extra fields. For now, likely not, since micro-lenders seldom require collateral except maybe car title in some cases. We skip unless needed for completeness. - **Insurance:** If credit life insurance is required for this loan (some products require it for term > 6 months or amount above X). If so, the form can show the premium or ask if client accepts insurance. If insurance is optional (like they can opt out if they have their own cover), include a checkbox "Include credit life cover". If checked, add premium to costs and reflect in schedule. - **Notes/Comments:** A text area for loan officer to add any remarks or special considerations (these will be seen by the approver). - Possibly **Attachments:** If any additional document is needed at application (like a new

payslip if last on file is old), allow attaching here too (this would go to documents under client or specific to this loan).

The form will be dynamic: e.g., when you pick product, it populates default term or allowed terms, when you enter amount it recalculates fees and schedule. This can be done via front-end logic calling a small interest calculation service (we can write a function or do it client-side).

Affordability check in form: The system should display the result of affordability calculation as the officer fills the form: - For example, show “Monthly income: R X, Expenses: R Y, Existing debt payments: R Z, Proposed installment: R N. Disposable income after new loan: R D.” And highlight if R D is positive enough. Possibly color-code pass/fail. If it seems the loan would cause over-indebtedness, the system can warn “Client may not afford this loan (disposable income would be only R D). Consider reducing amount or declining.” - We incorporate existing debt obligations from credit report if available: e.g., if credit report says client pays R500 on other loans monthly, include that. - This addresses the regulation against reckless lending by making the officer aware of the affordability outcome.

The loan officer can still submit an application that fails affordability (maybe client insists or there is other household income not counted), but the system will flag it for sure during approval and require a motivation if proceeding.

After filling the form, officer clicks **Submit Application**. On submission: - Frontend validates all required fields present and consistent. - The data is sent to the backend (Cloud Function or Firestore). If using direct Firestore, we’d add a new document in `loans` with status “Application” or “PendingApproval” and fields as filled. - The backend logic (or security rules) will ensure the user can create this loan (role check, branch consistency). - Cloud Functions might trigger on new loan doc: one function could auto-calc some fields to ensure consistency (like re-check the installment calculation server-side for trust, or attach an affordability result, or call credit check if configured to auto-run). - Alternatively, instead of direct Firestore, we call a function `applyLoan()` as earlier API spec to handle all logic including credit check asynchronously. - For our design, let's say we do immediate partial processing: * Save application with all details and a field like `affordabilityPass: true/false`, `DTI: xx%` (debt-to-income) computed. * Possibly mark `autoApproved: true/false` if we had a rule to autoapprove low-risk small loans (not likely yet, but we can include as future). * If we have integrated an instant credit score fetch, this could be triggered and attached to the application.

- **Credit Bureau Check (during application):** Many lenders will do a credit pull before approving. There are two approaches:
- *Manual by risk officer:* After application is in, a risk officer clicks "Run Credit Check" in the application view (if not already run) which calls integration and populates the result.
- *Auto at submission:* The system could automatically call credit bureau (if we have an API and consent) upon application submission, to save time. However, credit bureau calls cost money, so some might hold off until initial review. But to surpass others, auto integration is good. We can implement it with a Cloud Function triggered on new loan doc that calls TransUnion API with the client's ID number and then updates the loan doc with `creditScore` and maybe some summary flags (like number of defaults etc).
- If the credit pull fails (e.g., bureau down or no record), we handle gracefully (maybe set a flag and try again or require manual).

Loan Evaluation & Approval Interface: - Once an application is submitted, it appears in the **Pending Approvals** list (for managers/risk to review). - This list (menu item “Loan Applications” or “Pending Approvals”) shows all loans with status "PendingApproval"/"Application" that are awaiting decision. Each entry might show Client Name, Amount, Term, Date applied, Officer name, maybe an initial risk flag (like credit score or "affordability fail" highlight). - A manager or risk officer clicks an entry to open the **Loan Application Detail View**.

- **Loan Application Detail View:** This screen provides a comprehensive view to assess and decision the loan:
- Top section: Client info snapshot (name, ID, age, maybe credit score if fetched, client’s loan history like how many previous loans, and any notes like "Blacklisted" if that were true).
- Next section: Application details (the fields from the form: amount, term, product, purpose, etc.).
- Show the calculated repayment schedule clearly (table of installment dates and amounts).
- Affordability summary: show income, expenses, existing debt from credit report, resulting disposable, and perhaps a recommended maximum loan amount if our system can calculate one (some advanced systems do that).
- Credit bureau information: If we have a credit report summary, show it – e.g., *Credit Score: 610 (Fair)*, *Accounts in arrears: 1 (30 days)*, *Existing total debt: R50,000*, *Other Loans: 2 active*, *Judgments: none*, *Debt Review: no*. Possibly an option to view full credit report (if we stored a PDF or detailed info). This section is crucial for compliance – demonstrates that credit information is considered [7].
- Documents: Ensure KYC docs are present. The approver will want to see the ID copy and payslip. So, link or embedded thumbnails of the uploaded docs for quick access. If something is missing, highlight “Document X required is missing”. They might then send back to officer to complete or put application on hold.
- Officer’s notes: display any comment the loan officer made.
- **Decision controls:** Approver has buttons or options at bottom: **Approve**, **Decline**, maybe **Pend** (on hold, if they need more info).
    - If Approve: maybe allow editing some terms before finalizing (like they might approve but for a lower amount or shorter term than requested). So the approver UI could allow adjusting amount or term or interest within allowed range. If changed, system recalculates schedule accordingly.
    - If Decline: require selecting a reason (like drop-down: "Failed affordability", "Bad credit history", "Fraud suspicion", etc., plus a text box for comments).
    - If Pend: they could mark as "More info needed" and give notes (like "upload latest payslip, then notify me"). The application stays pending but maybe flagged or returned to loan officer queue as “Needs Attention”.

- Only users with appropriate role (Manager or Credit Committee or Admin) see these decision buttons. Loan officers opening this view would see it read-only or see "Awaiting Approval".

- **Approval Process & Compliance:**

- If approved, the system will prompt for confirmation and then execute what we outlined in A5 for approval: update loan status to "Approved", stamp approvedBy and date. Possibly require the approver to input their credentials again or 2FA confirm (for high amounts, some systems require a second factor to approve to ensure it’s intentional).
- The system should automatically generate the **Loan Agreement** document upon approval. This includes all terms (interest rate, repayment schedule, fees, total cost of credit, etc.) and client info. As

per NCA, a **Pre-Agreement Statement and Quotation** should be provided to client before they accept the loan, and then the **Credit Agreement** when final. Our system can use the application data to produce a Quotation PDF (even before final approval, if needed to give to client to sign) and the final Agreement after approval. In practice, in branch lending, they might print these or show on a tablet for e-sign. For digital, we can send it via email/WhatsApp for e-sign.

- We should also ensure the interest and fees on the contract do not exceed regulated maxima: e.g., check that interestRate <= allowed (which in config it is) and the initiation fee charged <= formula result [13]. The system calculates those so it should be compliant by design.
- The client must **sign/accept** the agreement. If in person, they'd sign paper or on device. If remote, perhaps they click an acceptance link or enter an OTP. Our system could implement an e-sign by sending a unique link that shows the contract PDF and has an "Accept" button that records timestamp (or ask client to reply with a code). For completeness, mention that if using WhatsApp or email, we could use a one-time PIN that client sends back or a service like DocuSign integration for a more formal e-sign. Given competitor highlights electronic signatures [14], we incorporate an **E-Signature** step: perhaps use a touchscreen signature capture in the mobile app or a simple "I accept" OTP flow. We will at least capture a field `acceptedByClient` (boolean) and `acceptedAt` timestamp when they do so.
- Only after client acceptance (which might be immediate if they are present or slightly later if remote) do we mark the loan as ready to disburse.

- If the client refuses to accept terms, then it's effectively not going to disburse (they might want changes or cancel).

- **Rejection Handling:**

- If declined, system sets status "Rejected", stores reason. Optionally, generate a rejection notice letter (NCR requires providing reasons if asked, at least verbally or written).
- That application is closed. The client could apply again later, but maybe if reason was something fixable (like need guarantor), they might come back.
- For tracking, keep the record of rejected apps. Possibly in client profile show "Previous Applications: 1 rejected on [date] reason X".

- If an application is rejected due to credit issues, some systems update an internal blacklist or score for client (e.g., to easily decline in future if nothing changed).

- **Loan Activation and Disbursement:**

- Once approved (and accepted), the next step is disbursement (paying out the loan). This might be done automatically by the system or manually by finance:
    - If automatically, integration with PayFast or bank API could send money to client's bank. More realistically, micro-lenders often just do an EFT from their bank account and then mark it done in system. We have a choice: either have a "Mark Disbursed" action that the finance person clicks when they have sent money, or if integrated with a payment API (some payment gateways do payouts, e.g., PayFast has a Payout feature for merchants or one could use a bank's API).
    - For thoroughness, let's incorporate: The system can produce an EFT payment batch or use an API to pay out. We will detail in Payment System (B6).

    - In origination context, we simply note that after approval, the loan's status is still "Approved/ Pending Disbursement" until the funds actually go out. Once funds are sent, status changes to "Active".
    - We may have a separate small UI for finance to handle multiple disbursements at once (like list of approved loans that need payout, with bank details and amounts, and a button to confirm each or generate a batch file for bank).
- When marking as disbursed, we also record actual first payment due date (which might depend on when money was given, possibly align to next pay cycle).

- The borrower is notified (SMS/email) that their loan is approved and funds are on the way or ready for pickup if cash.

- **Loan Servicing Start:** Once active, the loan moves out of origination. However, the origination module might still display active loans in a list (like "All Active Loans") primarily for reference or quick actions like early settlement.

- **Quotations:** NCR requires that if a consumer asks, the lender must provide a quotation (valid for 5 business days) before they sign the actual contract. Mobiloan had a Quotation menu [15]. Our system can allow generating a quote without formally submitting an app:

- E.g., a tool where you input amount and term, and it outputs the payment schedule and total cost in a formatted PDF. This is not stored as a loan app (unless we want to keep it). It's more a simulation. Could be used when client is shopping around.
- We'll include a "Generate Quote" feature: basically the loan form can be filled and then click "Generate Quote" which produces a PDF with a reference number, but not saved in loans DB (or saved as temporary). If the client decides to proceed within 5 days, the officer can convert that quote to a full application (perhaps by saving it then).
- This ensures compliance to provide quotes easily.

Compliance Considerations (Loan Origination): This module has the most compliance points: - **Responsible Lending (NCA):** We incorporate the mandated checks: - **Affordability Assessment:** As described, using income vs expenses vs debt. Document the outcome. Ideally, the client should sign that the info they provided is true and that the assessment was explained. We store income/expense and outcome on the loan record so an auditor can see it was done. If the loan is approved contrary to a failing affordability, that's potential "reckless lending" which is illegal. Our system should either block it or require a special override with motivation recorded by the manager. Possibly implement an override code so that if affordability is negative, only Admin can approve after writing a justification in notes field. - **Credit Assessment:** The law requires checking credit bureau for most loans. Our integration ensures a credit check is done and considered. We store at least the credit score and maybe number of accounts, etc. in the loan file. If no credit info, at least the attempt is logged. - **Interest & Fee Caps:** The system by design calculates interest and fees within regulatory caps (e.g., as per [NCA regs] maximum interest for unsecured = 28% p.a. and initiation fee caps [11], service fee cap ~R60+ inflation). By using the product config that adheres to these, and by maybe double-checking on submission (we could recalc and ensure no one tampered), we ensure compliance. If user tries to input a custom rate beyond allowed, system should refuse. - **Disclosure:** The loan agreement produced must clearly disclose the principal, interest rate, fees, total repayable, installment amount, number of installments, and any security or insurance details, as per NCA's requirements for a credit agreement. We will use templates to ensure all required info is present. For example, include phrases like "This agreement is governed by the National Credit Act... interest is calculated

daily and charged monthly… etc." - **Cooling-off:** Not explicitly NCA for microloans, but NCA allows a consumer some rights to rescind in certain situations within a short time (usually for intermediated transactions). Not too common in microloans, but we could note if any such right, system should handle if client cancels after signing but before disbursement (just mark loan cancelled, don't charge anything). - **POPIA:** Ensure client's personal and financial details on the application are protected. Only relevant staff can see them. Also, when sending loan agreements or info to the client, use secure channels (we wouldn't email a PDF with ID number visible without encryption maybe – WhatsApp is end-to-end encrypted, email is less so but we can password protect PDFs or use secure portal links). - **Record Keeping:** Keep copies of agreements and communications. The system storing the signed loan agreement and proof that client accepted covers us. Also keep the quote if given (perhaps store quotes as records that were provided, maybe in documents with date, in case a dispute). - **NCR Reporting:** Lenders must report new credit agreements to credit bureaus typically within 7 days. Our system can facilitate that by generating a data file of all loans opened (with details like ID, amount, term, installments) to send to bureaus or NCR's system monthly. That integration can be considered in Reporting or integration section. - **Anti-fraud:** During origination, watch for fraud signs: e.g., if one user is creating too many loans quickly (maybe internal fraud) – an admin report can catch that. Also verifying documents authenticity (OCR the ID to see if number matches provided, etc., as planned). - **Loan Stacking:** If client already has an outstanding loan with us, do we allow a second concurrent loan? Many micro-lenders do not unless it's a top-up (closing old and opening new bigger one). Our system should warn if client has an active loan already. Possibly block a new application if one is currently active, unless Admin override to allow multiple loans. That rule can be configured. If allowed, incorporate that into affordability (two loans payments). - **Agreement Signing Compliance:** If using e-sign, ensure the method complies with SA ECT Act (Electronic Communications and Transactions Act) for legal signing. An electronic signature is valid, especially if it's an advanced one. OTP confirmation or a tick + OTP likely suffices as an electronic signature for such agreements. (For high value maybe they'd prefer a physical or biometric, but up to the lender.) - **Insurance (if any):** If we include credit life insurance, NCA requires disclosing it, showing premium, and that client can choose provider (we might default one but let them provide proof of own if they have it). We should have a checkbox "I choose to take the credit life cover from [Partner] at R X per month included in installment. Or I have my own policy: [details]." If they have their own, we might require uploading that policy. This is detail to surpass others perhaps.

Interactions: - Origination interacts with **Risk Management (B4)** seamlessly – credit checks, scoring, etc., which we partly covered here but will detail next. - After a loan becomes Active, it falls under **Payment System (B6)** which handles scheduling those installment payments we set up. The Origination module should pass the schedule info to the payment subsystem (like maybe create the installment entries which the payment module picks up on due dates). - If a loan is not paid and goes overdue, that transitions to **Collections (B5)** – not part of origination but outcome. Possibly, the origination module's data is used by collections for reference (like interest rate to calculate further interest on arrears if any). - **Reporting (B7)** will include metrics from origination (like approval rates, number of applications by status). - The **Communication (B8)** module will send out notifications triggered by origination events: e.g. - "Loan Application Received" to client, - "Your loan has been approved" or "declined" messages, - If pending info, maybe an automated message: "Please upload your payslip to complete your application." - **Admin Tools (B13)**: manage products which origination uses; manage user limits possibly (some systems allow setting user-specific approval limits – e.g., a branch manager can approve up to R5000, beyond that goes to head office. We might not implement fully but mention it's possible). - **AI (B15)**: Could be used to auto-score or even auto-approve low-risk loans instantly (some advanced lenders do instant loan decisions). We could incorporate a simple version: e.g., if credit score above X and affordability pass and amount small, mark as

“AutoApproved” so that when manager sees it’s basically done or it might even auto-approve and notify manager it happened. But human in loop is usual for micro-lenders, so maybe mention future AI could do auto underwriting.

Now with origination covered, we move to Risk which overlaps but has additional tasks like portfolio risk monitoring.

B4. Risk Management System

The Risk Management System focuses on assessing and mitigating the credit risk associated with loan applications and the portfolio at large. It includes tools and processes for credit checking, scoring, setting risk-based pricing (if applicable), fraud detection, and monitoring the health of the loan book. This module works closely with Origination (B3) but adds specialized capabilities for risk officers or automated risk engines.

Features and Components:

- **Credit Bureau Integration:** As partially described, risk management relies on integrating with credit bureaus (TransUnion, Experian, etc.). The system will:
- Provide a function to **pull a credit report** for a client (either automatically at application or manually by a risk officer).
- Store key data from the report, like credit score, credit bureau scorecard (if any), and summary stats (e.g., number of defaults, total outstanding debt, any court judgments, whether under debt review).
- Display these on the loan application and possibly on the client profile risk tab.
- Possibly allow downloading/viewing the full report PDF (if provided by bureau) from within the system for detailed perusal.
- The integration must handle responses securely (the report contains sensitive data, handle as personal info).

- We will track that credit check was done and which bureau and on what date (since repeated checks can negatively impact score if done too often, we may have a rule not to pull a second report within X days for the same client unless necessary).

- **Internal Credit Scoring:** We define an internal scoring model or rules:

- For example, create a composite score or grade (A/B/C) for each application. This could combine the external credit score and other factors like: client’s income stability, existing relationship (if they had previous loans paid on time, that’s positive), and maybe behavioral data (not much at first).
- A simple rule-based approach: e.g., if TransUnion score > 650 and affordability margin > 20%, classify risk as Low; if score 550-650 Medium; below 550 High risk. Or use an actual logistic regression if data available (maybe later a ML model).
- This internal score might influence the decision or required approvals (e.g., high-risk loans must be approved by senior manager, low-risk can be auto-approved by system or by junior manager).

- Also could influence pricing: some lenders charge higher interest for riskier clients (within allowed max). If we wanted to surpass others, we could incorporate risk-based pricing: e.g., define in loanProducts a base rate and a possible variation. If risk = High, maybe charge max interest; if Low,

maybe give a discounted rate to attract good clients. This can be an optional feature toggled on by admin policy. For now, mention it as an advanced feature.

- **Fraud Detection:** The system should help catch potential fraud:

- **ID Verification:** We may integrate with the Department of Home Affairs or a third-party (like *XDS's identification verification* or *Experian's ID Verify*) to check that the ID number is valid and matches the name/DOB given. If integrated, when a new client is added or a credit check is done, we could do this verification. If not integrated, at least the system can validate the ID format and check digit which catches simple forgeries of ID numbers.
- **Blacklist/Watchlist:** There's a South African Fraud Prevention Service (SAFPS) where individuals reported for fraud or identity theft are listed. We could integrate to check if an ID is on that list. If yes, flag application for likely decline.
- **Device or IP patterns:** If we had an online portal, we could monitor if multiple applications come from same device/IP with different identities (sign of loan stacking fraud). This is advanced; maybe log IP/device in application and have a report for suspicious overlaps.
- **Application data validation:** e.g., if client claims very high income but is young with credit history that doesn't support it, that could be flagged. We can create some rule anomalies checks (like ratio of requested loan to income too high, or expenses too low to be believable).
- **Manual flagging:** Provide an option for a risk officer or system to flag an application as "Potential Fraud" which might require extra verification (like calling employer, etc.). Mark such in the system so that approver knows to be extra careful or to demand more proof.
- Possibly incorporate a stolen ID database check (some credit bureaus flag if an ID was reported stolen).

- If multi-lender data sharing was possible (not in our scope except through credit bureau's data), but at least using the credit bureau info we can see if the client recently took many loans (which could be a "loan shopping" pattern).

- **Risk Workflow:**

- Applications that meet certain criteria might be automatically routed to a *risk officer review queue* before reaching final approval. For example, if the score is low or first-time borrower or high amount, maybe it requires risk team review. The system can mark `riskReviewRequired = true` on such apps.
- Risk officers have a screen "My Evaluations" (as seen in Mobiloan menu [16]) where they see those flagged applications. They then perform the credit check (if not auto), verify documents, maybe call references if needed, and put a recommendation.
- The risk officer can fill a form on the application page with fields: `recommendation` (Approve/Deny), `recommendedAmount` (maybe lower than asked if they feel to reduce risk), and their `comments`. They then mark it as "Evaluated".
- Then it moves to manager for final decision with that input.

- If an application is straightforward (low risk), maybe it bypasses risk officer and goes directly to manager for decision or auto-approves if policy allows.

- **Portfolio Risk Monitoring:** Beyond individual loans, the risk module would keep an eye on the entire loan book risk:

- Provide **Analytics** such as:
    - Average credit score of active borrowers.
    - Distribution of loans by risk category.
    - PAR (Portfolio at Risk) metrics: e.g., what portion of portfolio is >30 days delinquent (which is a key microfinance indicator).
    - Concentration risk: e.g., how much of portfolio in one product or region, if relevant.
    - This overlaps with Reporting (B7), but from a risk perspective, these help adjust strategies.
- **Early Warning System:** Identify accounts showing signs of trouble early:
    - e.g., if a client misses a payment or even pays late or their credit bureau file gets a new adverse (some bureaus offer monitoring alerts). If integrated, could set up alerts for any new negative info on our clients. If not, at least watch payment behavior. If someone misses an installment, risk might want to review if they should cut off additional credit or flag it.
    - Could incorporate a **Behavioral Score**: after a few months of payments, re-score the client based on how they paid (common in credit).
- **Regulatory Capital / Provisions:** If needed, the system can calculate provisions for bad debt based on risk (like IFRS9 or some internal policy). Possibly beyond scope, but mention that risk officers might want to categorize loans by expected loss.

- The system can allow risk users to change status of a loan to different risk categories (like Stage 2 accounts).

- **Risk-based Collections Prioritization:** Risk module might inform Collections which accounts are higher risk (likely to default) so they can prioritize. For instance, if a client's credit score dropped or they defaulted elsewhere, or their behavior pattern (like frequently late) suggests they will default, we might mark them for preemptive action. This could be an automated list of high-risk active loans for collection team to remind or follow up before they miss payments.

- **AI in Risk:** We can leverage machine learning for risk:

- Use historical data to train a model to predict probability of default (PD) for new applications. As mentioned, we could eventually have a model that takes in application info and outputs PD or risk grade.
- Initially, we might implement a basic logistic regression or decision tree via Python and store the formula or use a simple rule approximation. For example: if no prior credit and low income -> high PD; if high credit score and stable job -> low PD. Codex could generate a basic model code for us or we can simulate with a weighted scoring approach.

- *Fraud detection ML:* pattern recognition (like clustering accounts by contact info to catch rings). Possibly too advanced to fully implement now, but mention as a future addition.

- **User Interface for Risk:**

- Risk Dashboard: A screen showing overall risk metrics (like number of loans by grade, average PD, alerts for risky trends).
- Risk Review Queue: As described, list of applications needing risk review.

- Risk Details on Application: Already integrated in the loan application view in B3 (score, etc.).
- A **Credit Bureau Inquiry** tool: risk officer can input an ID and fetch a credit report even outside an application (maybe to pre-qualify or to check an existing client's status mid-loan). Those inquiries should be logged to avoid misuse.

- Possibly an **Override** function: risk officer can approve a deviation from policy with explanation. For example, policy says max debt-to-income 40%, but client is at 45% because they have other household income uncounted – risk officer can override and recommend approval, capturing rationale. The system should log that and maybe require a higher-up to countersign or at least keep in audit.

- **Compliance (Risk Management):**

- The **National Credit Act** requires that lending decisions consider credit bureau data and affordability – our risk module ensures that.
- All credit inquiries must be done with client consent and recorded – we store the `consent.creditCheck` as true and log inquiries by user and date (so if client pulls their data usage report, we know when we accessed their bureau info).
- There are also **NCR affordability regulations** (like there's a general requirement to have "a predetermined criteria and model" for credit assessment; our system provides a consistent model).
- If we adjust interest by risk, we must still abide max interest and we should not discriminate unfairly (the criteria should be credit risk based, which is fair use).
- **Data handling:** Credit reports contain info on other credit – treat that with confidentiality. Only risk/management roles should see full credit report. Possibly even the loan officer shouldn't see the full bureau details, just maybe a summary or none at all. We might restrict the full credit report view to risk officers and above, whereas loan officer only sees maybe that one was done and a high-level "score was X, recommended risk grade Y".
- **Audit & Model Governance:** If we use an algorithm or model, regulators might ask to see how decisions are made. We should document the scoring criteria (in code and maybe in an internal document). For every decline due to credit/risk, ideally list reason codes (like "Reason for decline: Inadequate affordability, Adverse credit history" etc.), which we do capture. This helps demonstrate compliance with not lending recklessly and giving reason for rejection, as required if customer asks.
- **Limit management:** Sometimes NCA sets special rules, e.g., maximum loan amount for short-term credit is R8000 (for loans < 6 months) – our product config and input validation should enforce that. Or cannot charge interest beyond certain points if account in default (like once handed over, interest might freeze depending on actions). Risk module might enforce such limits too (like don't approve beyond allowed amount for category).

Interactions: - Risk management is tightly integrated with Origination (B3, as above). - It informs **Collections (B5)** by highlighting accounts likely needing attention. - Works with **Integrations (B11)** for pulling credit data. - Possibly uses **AI (B15)** for advanced scoring – we will detail in AI section any overlap (like we might mention training risk models there). - Risk output might feed into **Reporting (B7)** for regulatory reports like "Credit granted vs credit rejected stats" and "Risk profile of lending" which some internal risk committees or NCR might require. - Risk might also interface with **Compliance (B10)**, since compliance will ensure we followed risk procedures properly (like we did check credit, we did do affordability, etc., which are compliance checkpoints).

By implementing a thorough Risk Management System, we not only help protect the lender from losses but also ensure compliance with responsible lending obligations. The next module, Collections, will pick up when risk materializes as delinquency.

B5. Collections System

The Collections System deals with loans that have entered **delinquency** – meaning missed or late payments. Its goal is to manage the process of recovering overdue amounts from borrowers in an organized, efficient, and legally compliant manner. This includes tracking overdue loans, assigning collection tasks to agents, contacting borrowers through various channels, and escalating cases as needed (to legal action or external collections agencies).

Scope of Collections: It typically starts when a loan payment is past due (often defined as a certain number of days past due date, e.g., 1 day overdue triggers soft collections). The Collections module will manage everything from day 1 delinquency through charge-off or settlement of the debt.

Key Features and Workflow:

- **Delinquency Monitoring:**
- The system automatically flags loans as overdue if a scheduled payment is missed. For instance, if an installment due on 2025-08-01 was not marked paid by 2025-08-02, the loan (or that installment) gets a status "Missed" and triggers collections.
- For each loan, we might keep a field `daysPastDue` which updates daily or when payments come. Or simply compute from due dates and payment data.
- Based on `daysPastDue`, we categorize delinquencies: e.g., 1-30 days = early delinquency, 31-60 = mid, 61-90 = late, >90 = critical. These can correspond to collection strategies and regulatory definitions (NCR might require issuing a formal letter before 20 business days, etc., which we cover).

- The system can generate an **Overdue Loans List** showing all loans currently with any overdue amount, sorted by severity (longest overdue first perhaps, or largest amount overdue).

- **Collections Tickets / Cases:**

- As soon as a loan goes overdue (and perhaps past any grace period), the system creates a **collections case** (or "ticket") for it. This is an entry in the `collections` collection (schema in A4).
- The ticket will contain the current info: loanId, client, amount due, days overdue, and it's marked Open. Optionally, assign automatically to a collections agent (if we have multiple, maybe round-robin or by branch).
- If the loan cures (payment made) quickly, the case can be closed automatically or marked resolved.

- If still open, collections agents will work on these cases.

- **Collections Dashboard:**

- A specialized dashboard for collectors showing metrics: total overdue count, total overdue amount, broken down by category (like X accounts 1-30 days, Y accounts 31-60 days, etc.), their personal assignments, etc.

• Possibly display performance like how many promises to pay obtained, how many kept, etc.

• **Collections List/Queue:**

• **All Active Tickets** list (for a manager or central view): shows all open cases (like Mobiloan's "All Active Tickets" [17]).
• **My Assigned Tickets** list (for a specific collector): shows only those assigned to them.
• Each entry might show: Client Name, Loan ID, Days Overdue, Amount Overdue (maybe total arrears), Last action date, Next action due.

• Ability to filter/sort by days overdue, amount, region, etc., and search by client.

• **Collections Case View (Loan Collections Screen):**

• When collector opens a ticket, they see detailed information necessary to negotiate and resolve:
    ◦ Borrower's contact info (phone numbers, address, email, and perhaps alternative contacts like next of kin if available).
    ◦ Loan details: original amount, installment amount, interest rate, collateral if any, etc.
    ◦ Payment history: which installments were paid or missed, how much outstanding. Possibly a breakdown: e.g., "2 installments missed: July and Aug, total arrears R2000, next installment due on ..., total outstanding RX including arrears."
    ◦ Any notes from the loan or risk (like risk grade or if client had any special condition).
    ◦ A **History of Collection Actions**: all previous interactions recorded. For a new case, it might be empty or show automated actions (like "System sent SMS on due date", etc.).
    ◦ The interface to **log a new action**:
    ◦ E.g., a dropdown of action type (Call, SMS, Email, WhatsApp, Letter, In-person visit, Other),
    ◦ an outcome (Reached, No answer, Left message, etc.),
    ◦ a promise to pay (if client promises, input amount and date promised),
    ◦ or a dispute raised by client (maybe they claim they paid or request extension).
    ◦ and a notes field for free text (e.g., "Client said will pay R500 on Friday").
    ◦ Also a follow-up date field (system can default e.g., if promised Friday, set follow-up next Monday to check).
    ◦ Once they log an action, it gets saved to the actions subcollection and updates the ticket (like lastActionDate, status if needed).
• The collector can also update the **status** of the ticket:
    ◦ For instance, if the client pays and account is now up-to-date, mark ticket as Closed (Resolved).
    ◦ If unable to reach for X days, escalate to "Pre-legal" perhaps.
    ◦ If client is refusing or absconded, escalate to "Legal Handover".
    ◦ We can have statuses: Open (in progress), PTP (promised to pay, awaiting that date), Broken PTP (if they promised but didn't pay by promised date), Resolved (paid), Closed - Handover (if sent to external collections or legal).

• Possibly incorporate triggers: if a promised pay date passes and payment not received, system can reopen or mark Broken PTP and put back on top of queue.

• **Automated Collections Actions:**

- The system should automate initial reminders:
    - For example, on the day a payment is missed, automatically send an SMS or WhatsApp: "Dear [Name], we noticed your payment due on [date] was not received. Please pay ASAP or contact us."
    - Perhaps send an email if available too.
    - These communications can be templated and handled via Communication module (B8) triggered by Payment System (B6).
    - System might schedule another reminder at 5 days overdue, etc.
    - The communications should be logged in the collection actions history for reference.
- If still no payment after X days (e.g., 20 business days in SA), the law requires sending a formal **Section 129 notice** (this is a legal notice of default giving the consumer a chance to remedy or face legal action). Our system should:
    - Generate this Section 129 letter automatically (with details like amount due, etc.), possibly as a PDF.
    - Send it via registered mail or preferably both physical and electronic (though WhatsApp/email might not suffice legally for this notice, but at least as additional).
    - Log that it was sent and the date (important for legal compliance).
    - Possibly track if client responded to it.

- After sending that letter and waiting the required period, the loan can be handed to attorneys or external collectors. The system should mark that and close internal case (or mark as handed-over, which might freeze interest accrual or change how it's tracked).

- **Integration with Payment System:**

- Collections agents need to possibly take payments from the client on the spot or change payment arrangements:
    - They might want to trigger an immediate debit attempt via integration (like using NuPay MPS - once-off debit now, or send PayFast link to pay). The UI could have a button "Collect Now" which triggers a debit (calls payment API).
    - They could also reschedule future debit attempts, e.g., client says "debit my account again on the 15th". The agent can schedule a retry on that date easily in the system (which essentially is creating a new entry in payments for that date).
    - If client wants to pay via card on phone, agent could send them a PayFast/Ozow link via SMS from the system. The outcome of that would update the payment and then the collections case if payment is successful.
- If partial payments are made, the system should update the arrears amount accordingly in real-time so collectors see updated balances. Possibly reflect new arrangement (like if client can't pay full arrears, and we allow them to pay over next 3 installments, maybe structure that).
- The Collections module might also allow for certain modifications: e.g., waiving a late fee or doing a refinance. Some lenders at collections stage might negotiate to extend term or restructure the loan.
    - If we do **restructures**: It could be done by closing current loan and creating a new loan (with remaining amount) or adjusting schedule. That’s complex, but perhaps within Collections, an authorized user could click "Reschedule Loan" – which then perhaps calls the Loan Origination module to create a modified schedule (with maybe additional interest or not). Our manual mention the possibility but maybe not implement full details. We'll note restructure as an option where a new schedule is created, interest possibly continues, and we mark loan as Restructured.

- If **write-off**: After a certain period or if debt is deemed uncollectable, an admin might decide to write it off. In system, that means mark loan status "WrittenOff", create an accounting entry, and close collections case with status "Unresolved/Write-off". We track the loss. Possibly also flag that client as blacklisted for future.

- **User Roles in Collections:**

- Collections Agents (Collectors) have rights to view overdue accounts and update collections actions. They should NOT be able to modify core loan details (like they cannot change loan amount or due dates arbitrarily; any change must be through defined processes).
- They can trigger communications, and maybe manually adjust a promise date etc., but typically not change interest or fees.
- Managers of collections can see all cases, reassign them to different agents, and perhaps have rights to negotiate settlements (e.g., allow a discount on interest if client pays now – such an action could be built in, where manager can waive certain fees or interest).
- If integrated with a dialer system or call center phone system, roles could have click-to-call if using VOIP. (Sudonum integration possibly, as Mobiloan mentions Sudonum which might be a call tracking solution).

- Possibly integration with an auto-dialer or just logging calls suffice for now.

- **Compliance (Collections):**

- **NCA / Debt Collection Regulations:**
    - As mentioned, Section 129 letter (notice of default) is a legal requirement typically before legal enforcement. Our system generating and tracking that ensures compliance. We should not escalate to court without that step.
    - There are rules on **collection fees**: e.g., lender can't charge exorbitant collection charges beyond actual costs, etc. If we charge a default admin fee or letter fee, it should be within allowed limits (NCA allows a once-off default admin charge + possibly interest on arrears can continue at contracted rate until judgment).
    - The system can automatically apply a *penalty fee* or interest on overdue amounts if allowed (some do, but as per NCA, interest can continue up to certain point, but not sure if additional penalty interest is allowed beyond contract interest).
    - Keep track of communications. Under POPIA and ethical collection practices, calls should be made in certain hours (8am-5pm typically). The system could log call times to ensure compliance with not calling at 9pm for example. We can prompt collectors with appropriate times or restrict scheduling follow-ups outside business hours.
    - Also, ensure we have consent to contact references if we do (some loan forms let you contact their employer or kin if they default – need consent).
    - If client goes under **debt counseling** (debt review) after loan was given, by law you cannot collect normally; the matter goes to a debt counselor who arranges reduced payments. Our system should mark if we receive notice of debt review (could come via credit bureau update or client letter). Then collections status might change to "Under Debt Review" and the case might be suspended or handled differently (like stop interest or accept reduced payment plan as per court order). We might not implement that deeply, but note it if needed.

- **POPIA:** Collections must handle personal info carefully too. E.g., if sending WhatsApp messages, be careful not to include too much sensitive info that could be seen by others (like perhaps don't include full ID or account details in an SMS). Also, verify you are speaking to the right person on call (to not disclose debt info to someone else).

- All collection communications should remain respectful and within legal boundaries (no harassment or threats beyond stating legal consequence). The system can use templated language to help ensure compliance.

- **External Collections / Legal:**

- If internal collection fails at say 90 days, and after Section 129, the next step might be to hand over to a law firm or collection agency. The system should handle that by:

    - Marking loan as handed over. Possibly create a record of to whom and when (agency name, reference).
    - Stop internal interest if required (some stop interest at handover, some continue until judgment).
    - Possibly export the case data to a format to give the external party.
    - If the external party collects money, finance can record those payments in the system as well, to update the balance.
    - If legal action leads to judgment or write-off, mark accordingly.
    - This process ensures we formally close out internal tracking and avoid double efforts.

- **Collections Strategy & Efficiency:**

- The system can help managers track collector performance: number of calls made, promises vs kept, money recovered per collector etc. These are management reports to optimize strategy.
- Could incorporate automated prioritization: e.g., sort who to call not just by oldest, but maybe by collectability (if a large amount but likely uncollectable vs a smaller but likely collectable, maybe focus on the latter first – depending on strategy).
- Possibly integrate with a schedule or calendar view: a collector could have a daily worklist (cases with follow-ups due today, new missed payments from yesterday).

Interactions: - Collections uses data from **Payments (B6)** to update status. Ideally, Payment system triggers an update to collections if a payment fails or succeeds: - E.g., if a retry payment succeeded, the collections case might auto close or at least update arrears. - We might have a Cloud Function that listens to `payments` collection for outcomes and updates the `loans` and `collections` accordingly. - Collections heavily uses **Communication (B8)** for reaching out to borrowers. We might have integrated sending of SMS/WhatsApp or even automated voice messages. The templates used should be specific (like "Your account is X days overdue..."). - Collections ties into **Compliance (B10)** in ensuring legal steps (like Sec129 letter) are done. - Data from collections flows to **Reporting (B7)**: e.g., recovery rates, how many went to legal, average days to collect, etc. - **AI & ML (B15)** can help collections too: e.g., predictive modeling to see which accounts are unlikely to pay vs likely if nudged, so you focus efforts. Or recommending the best time to call someone (some systems find patterns like person tends to pay on 15th, etc.). Or even a chatbot for collections via WhatsApp. We can mention these as forward-looking enhancements.

With the Collections System in place, the LMS can manage the back-end of the loan lifecycle (post-disbursement issues). Now, let's detail the Payment System which runs in parallel to ensure all the financial transactions (disbursements and repayments) are executed and tracked properly.

B6. Payment System

The Payment System handles all monetary transactions in the LMS, including **loan disbursements (payouts to borrowers)** and **loan repayments (incoming payments from borrowers)**, along with related processes like handling failed payments, retry logic, and reconciling accounts. This is a critical module that ensures money flows are executed timely and accurately, and that the system's financial records match the actual payments processed.

Components of Payment System:

- **Disbursement Process:** How loan funds are delivered to the client after approval.
- Once a loan is approved and accepted, the disbursement process kicks in. We have a few methods:
  - **EFT (Electronic Funds Transfer):** The most common method – transferring funds to the client's bank account. The system will generate a payout instruction using the banking details from the client profile. If integrated with a provider (like using PayFast Payouts or even Allps if they have a disbursement function), the system can call an API to send money. Alternatively, generate a file for batch import into the bank's system if manual.
  - **Cash or Cheque:** Some micro-lenders dispense cash at the branch. If this is done, the system should record that "Cash disbursed by [User] on [Date]" and possibly print a receipt voucher.
  - **Mobile wallet (eWallet):** It's mentioned Allps even supports eWallet payments [18], which could be an option. If client doesn't have bank, maybe send to a mobile wallet (like FNB eWallet or similar).
- The Payment system should allow an authorized user (like a finance officer or admin) to **initiate and confirm disbursements**. Possibly a screen listing loans "Ready to Disburse".
  - For each, show details (client, amount, bank info) and a button to 'Pay Now' or 'Mark Paid'.
  - If integrated (like through PayFast's API or a direct bank API), clicking 'Pay Now' triggers the transfer and returns success/fail. If success, mark disbursed and date.
  - If manual, the user might do the EFT on bank's side, then come back and click 'Mark as Paid'.
  - The system must ensure double payments don't occur. So maybe include a check: once marked paid, disable further actions. If integrated, also check response.
- Each disbursement is recorded as a transaction in `payments` collection (type "Disbursement") with reference details (like transaction ID or bank reference number).
- Reconcile disbursements: ideally, the system should confirm that the money actually left the account. If integrated, you'll get a confirmation. If not, maybe they check bank statement and tick off manually.

- Compliance: Confirm that we only disburse to the verified bank account of the client (to prevent fraud or mistakes). The system can integrate an account verification step before disbursement (some services verify account ownership by ID). Not explicitly required but it's a prudent step (some lenders do a "penny drop" verification).

- **Repayment Scheduling:** After disbursement, all expected repayments are scheduled.

- Based on loan schedule, for each installment due the system should plan to collect it via the chosen method.
- If **Debit Order (NAEDO/DebiCheck)** is the method:
    - The system will register a mandate and schedule with the debit order provider (Allps or NuPay). Possibly done right after loan approval:
    - If DebiCheck (authenticated debit orders), the client must approve the mandate via their bank (maybe they get an SMS from bank to confirm). Our integration could trigger that request. Allps/NuPay provide such flows.
    - If NAEDO (non-authenticated early debit), it's more old system where multiple attempts can be made in early morning after salary day.
    - We gather needed info: client bank, account type, branch code, etc. and send via API a mandate setup for specific dates or a recurring rule.
    - On an ongoing basis, near the due date, the provider will attempt the debit:
    - Our system either asks the provider to process on that date (maybe we call an API "run debit for mandate # on date").
    - Or if schedule was pre-registered, the provider does it and sends result.
    - We will require feedback of success/failure. Ideally via an API callback or by us fetching a report.
- If **PayFast or Ozow** (usually for one-time card or EFT payments initiated by client):
    - We might not schedule these automatically, as they require client action (like clicking a link or entering card details).
    - But we can facilitate by sending a payment reminder with a link where they can pay. Possibly our system can host a payment page or redirect to PayFast with loan reference and amount for them to complete.
    - If a client saved card details (tokenization) via PayFast, we might even charge automatically if they gave consent (less common due to needing CVV unless token with debits).
    - But for recurring, debit order is the main method; card/ozow is often an alternative if debit fails or client prefers to pay manually.

- If **Cash payments**:

    - Some borrowers might pay in person at branch. Then the cashier would use a “Receive Payment” screen to log the payment (like we have manual receipt).
    - Or they deposit into lender’s bank. Then reconciliation needed to match that deposit to the client.
    - Could integrate with systems like Pay@ or EasyPay where clients can pay at supermarkets with a barcode. If we wanted to, could generate a payment barcode for each loan that clients use at retailers, but that's beyond current scope. (Mention to surpass maybe, but let's skip heavy detail.)

- **Payment Collection Execution:**

- We likely set up a **Scheduled Cloud Function (cron)** daily to check for any installments due that day (or earlier not paid):
    - For each, if method is debit order, and if not already processed, call the debit API (or mark for file output).
    - If method is internal manual, nothing to do except wait for branch to collect.

    - If multiple installments on same day (rare unless multiple loans for one or multiple clients with same date), handle accordingly (just loop).
    - If an installment is in arrears (past date and not paid), our collections module has it, but Payment system might still attempt another method if available (like maybe try card on file if debit failed).
- **Retries for Failed Payments:**
    - Typically with NAEDO, the system will automatically re-run in the same tracking period (like next early morning window). But to be thorough:
    - If a debit fails due to insufficient funds (code from bank), the micro-lender often tries again on the next salary date or a few days later.
    - Our system will have a policy: e.g., if fail on due date, set a retry maybe 3 days later or when client said money will be in.
    - We can either rely on the collections agents to schedule a retry (they talk to client and then schedule via the UI), or have automated second attempt (some do two attempts by default).
    - If using DebiCheck, might allow certain reattempts within same mandate but limited.
    - The system should track number of failed attempts. Perhaps after 2 fails, escalate to collections.
    - Possibly auto switch method: if debit fails multiple times, send an Ozow link via SMS as alternate to prompt client to pay via any bank at that moment.
- **Partial Payments:**
    - If a client pays not the full installment but some part (maybe manually), the system should credit that, reduce outstanding, and mark installment partially paid.
    - We might either break an installment into two records or update one with remaining due.
    - Many systems treat an installment as paid only when full amount is covered, but we should account partial to not lose the money tracking.
    - For interest accrual, partial pay might reduce interest going forward if daily interest.
- **Pre-payments:** If client pays earlier or extra, the system should either advance their schedule (next payment may be smaller or next due moved forward) or allow them to settle early (closing loan as discussed).
    - If someone tries to pay early via online, our system should accept and then logic to recalc maybe (some lenders will still charge full term interest unless settled in full; by law in SA, if settle early, you can charge up to maybe 3 months extra interest maximum, but likely we just calculate up to closure date).

- **Loan Settlement**:

    - If client wants to settle loan mid-way, the system (via loan origination or payment) calculates a settlement quote: principal remaining + accrued interest since last payment + perhaps a small settlement fee if allowed. Provide that number to client and if they pay it, mark loan closed.
    - The Payment module processes that as one final payment and updates loan status to closed.

- **Reconciliation:**

- Ensuring that every payment that is supposed to happen is accounted for:
    - The system's records (payments collection, loan balances) need to match the bank account or payment provider records.

- For **inbound payments**:
    - If integrated via API, success/failure is known immediately or via callback, so we update right away.
    - For things like EFT deposits: we might need to manually reconcile by checking bank statements. Possibly provide a screen to input bank deposits.
    - If we had many manual payments, maybe an import function: e.g., upload a CSV of bank transactions, system tries to match to loans by reference or amount (some systems give each loan a unique reference number to use when paying via EFT).
    - Or an interface with something like Xero (accounting) or a tool to tick off.
- For **outbound (disbursements)**:
    - We ensure the disbursed amounts match what we intended to pay (like no duplicate).
    - Possibly integrate an acknowledgement: E.g., PayFast Payout returns success and a batch report of which paid.

- The Payment system should provide a **Reconciliation Report**:

    - Listing all payments (in and out) within a period, their status (pending, success, failed), and allow an admin to mark unresolved ones if needed.
    - For example, if an EFT deposit is noticed on bank statement but system didn't record it (maybe client paid without correct reference), an admin can find whose loan amount matches and record that payment manually to reconcile.
    - Conversely, if system shows a debit as successful but later it bounced (some NAEDO might give provisional success then bounce after 2 days if reversed?), the system must correct that (maybe provider sends a late fail update).
    - So keep track of provisional vs final status if needed. But with DebiCheck/NAEDO, usually you'll know result by end of day.

- **Handling Fees & Charges:**

- Some payments come with fees (e.g., debit order fees charged to lender per attempt, or card processing fees). The system might not track that in client account, that's internal expense. Possibly an admin view for cost analysis but not needed in client view.
- If we charge clients any penalty fees (like a late fee of R50 after X days), the system should add that to their account balance and include it in amount due. But SA NCA limits collection charges (I recall a default admin fee ~ R50 allowed one-time). If we implement, do it once and record it so not charged repeatedly.
- Also monthly service fees continue to accrue if loan goes beyond term? Actually, if they default, they might still owe service fees each month until settled depending on contract. That can accumulate. Our system might continue to add service fee each month past due (if legally allowed) until we freeze or handover.
- Interest on arrears: The contract rate likely continues to apply on unpaid balance. We should accumulate interest daily on overdue principal. If the loan is on amortization schedule, once they miss, interest still accrues. Our schedule approach was fixed installments, but after a miss, interest for that period might not have been fully covered, so we need to recompute or add the extra interest to arrears. This is complex but necessary for precise finance: typically the next time they pay, interest is first taken for however long since last payment. Possibly easier: move to a daily interest accrual model once delinquent.

- The Payment system (maybe as part of daily job) can add interest for each day on overdue amounts, updating the loan balance. But for microloans, often they freeze schedule and just count the installment as missed and maybe charge extended interest. We'll attempt to track interest if beyond schedule. Simpler: we can keep an “accruedInterest” field that increments if payments are late.

- **User Interface (Payment Management):**

- **Disbursement Approval Screen:** For disbursing funds, as discussed, likely only accessible by finance or admin roles. Could incorporate multi-factor or dual approval for large amounts (one person prepares batch, another approves release).
- **Payment Recording Screen:** For branch cashier to input a payment made in cash or via POS. Fields: select loan or client (search), amount paid, date, method (cash, card swipe in branch, etc.), and any note. Then commit, which creates a payment record and updates balances.
- **Payment List:** Possibly a log of all payments received (like a bank ledger) which can be filtered by date, method, etc.
- **Scheduled Payments Calendar:** A calendar view or list of upcoming scheduled debits might be useful for an admin to see how many will run on a given day and total expected.
- **Failed Payment Handling:** A specific interface or alert for failed payments. E.g., a dashboard widget "5 Debit Orders failed today". Clicking shows which ones, then an action: maybe retry or mark for collections. The system likely auto did collections ticket, but the finance team might manually retry a small portion same day (some re-present attempts midday if funds came in).

- **Adjustments:** If any adjustment is needed (e.g., waiving interest), an authorized user could have a form to adjust a loan balance (with reason). That should be tightly controlled and logged.

- **Integration (as enumerated in B11 but recapping):**

- **ALLPS** and **NuPay** for debit orders:
    - We'll integrate the API or use their system's file upload. Possibly with Allps, since it's a web-based collection solution [19], maybe they have an API where we send a list of debits and get results.
    - If API: we store credentials, call endpoints to add debit orders. If file: system generates the correct format (e.g., NAEDO CSV) that user downloads and uploads to Allps portal.
    - We prefer API for automation. The system can also query Allps for result of each transaction.
- **Ozow, PayFast** for other payments:
    - For immediate payments, use their web redirect. Our system will have to implement their callback endpoints to update payment status once the client completes payment.
    - Possibly we embed Ozow in our web portal for clients or send them to their site with reference.
    - PayFast for card, likely just sending invoice links or embed their payment page in mobile app as webview.
    - We must ensure references and amounts align to loan.
- **TransUnion/Experian** (less direct for payments, but Experian has something called “CreditVision” that includes bank account info analysis, maybe not needed).
- **Payfast Payout** for disbursement:
    - PayFast offers a Mass Pay or Payout API where a business can send money to a recipient's email or phone (but that requires receiver to have PayFast account or they give bank details).

Might not be straightforward for us. We might stick to direct bank EFT or using bank's own integration if available.

- Possibly **SMS for OTP** if requiring to confirm a payment or something (some systems confirm if a large payment arrangement, they send OTP to confirm identity).

- **Compliance (Payments):**

- **NCR Interest & Fee compliance:** Payment system must enforce that we do not collect more interest than allowed. For example, NCA says you cannot keep charging interest indefinitely after a certain point (for short term loans, once it's defaulted you might not charge more interest after 2 months or something; also for all loans, once a judgment is obtained, future interest is often at a defined rate or stops).
    - We should ensure if a loan goes to write-off or handover, we freeze further interest in system, unless otherwise legally allowed.
    - Also, ensure we cease monthly service fees after a certain number of missed installments (I think monthly fee can still be charged until contract termination, but if account is handed over, they might not).
- **Debit Order mandates:** As of a few years ago, DebiCheck is required for new mandates. The system using it ensures compliance (no debiting without authorization).
- Provide easy evidence of payment history to client: NCA requires that if the client requests a statement, the lender must provide a detailed statement of all payments and fees. Our system can generate that on demand from the payments data (this could be part of reporting or a button on client portal).
- **Refunds:** If any situation requires refunding the client (overpayment, or loan cancelled within cooling period), the system should allow to process a refund (which is basically a disbursement in opposite direction and adjusting balances). Keep track and ensure proper authorization for refunds.
- **Security:** Handling bank accounts etc, ensure those fields maybe masked when displayed (e.g., show only last 4 digits of account to general view, full to authorized). Because bank details are sensitive personal info under POPIA.
- **Financial integrity:** Possibly external audit will check our system to ensure no money is unaccounted for. Having a complete audit trail of every transaction, who initiated it, and ideally dual control for big transactions is important. The manual instructs that we isolate features, but in practice, something like releasing disbursements might require a second user sign-off (we could incorporate a rule that disbursements above R X need two different admin approvals in the system, but that might be beyond our scope).
- **PCI-DSS:** If we were directly handling card details, we would need to comply with PCI standards (but we are not storing card numbers, we redirect to PayFast/Ozow which handle it, thus we stay out of scope of PCI largely).
- **Record retention:** Keep payment records for required time (likely 5 years min) for audit. We will, unless data is purged by request, not delete these.

Interactions: - Payment System supports **Client Portal/Mobile** by enabling them to pay (the mobile app might have a "Make Payment" button that uses Payment system integration with Ozow/Payfast). - Feeds **Reporting (B7)**: key stats like total disbursed, total collected, daily cash flow, etc. - **Collections (B5)**: as discussed, two-way interaction - Payment informs Collections of fails or success, Collections may schedule new payments. - Ties with **Integrations (B11)** for actual API usage of external. - Works under rules from **Compliance (B10)** for interest and others. - Payment events could also feed **AI (B15)** for analyzing patterns

(like who is likely to default based on payment behavior). - Payment system might integrate with general ledger if needed (some lenders want to feed data into an accounting system for financial statements, but that might be separate integration outside our direct scope. We mention if needed "Integrate with accounting for journal entries of interest, etc." possibly in Admin Tools or integration).

With Payment System described, next the Reporting & Analytics section will utilize data from all these modules to generate insights.

B7. Reporting & Analytics

The Reporting & Analytics module provides insights into the LMS data through various reports, summaries, and data visualizations. It serves both operational needs (daily reports, tracking KPIs) and strategic analysis (portfolio performance, trends). It also includes any regulatory reporting outputs required. Our goal is to surpass typical platforms by offering comprehensive, easy-to-use reporting capabilities, with possible real-time dashboards and the ability to drill down into data.

Types of Reports:

We can categorize the reports into a few groups:

- **Operational Reports:** Day-to-day or periodic reports for managing the business:
- Loan Portfolio Report
- Collections Report
- Payments Received Report
- User Activity Report
- etc.
- **Regulatory/Compliance Reports:** Specific outputs required by regulators or for audits:
- NCR regulatory returns (like the quarterly submission of loan book statistics, if applicable).
- Credit Bureau submissions (monthly feed of all loans, though that might be automated via integration, but we can generate a file).
- POPIA data access logs (who accessed personal info, if needed).
- Audit logs (list of changes and by whom, as needed by internal audit).
- **Analytical Dashboards:** Graphs and charts for trends:
- e.g., New loans per month, Repeat vs New customers, Portfolio growth.
- Delinquency trends (PAR by month).
- Income (interest & fees earned) vs Write-offs.
- Efficiency metrics (approval rate, average time from application to approval).
- **Ad-hoc Query Capability:** Possibly allow exporting raw data (to CSV/Excel) for deeper analysis outside system or for custom reports.

Let's detail some key specific reports:

A. Executive Summary Dashboard: (Accessible to management on login perhaps) - Displays top-level metrics: - *Total Active Loan Accounts*: count and total principal outstanding. - *Total Arrears*: amount and percentage of portfolio in arrears (e.g., PAR30 – Portfolio at Risk over 30 days). - *New Loans This Month*: count and value, vs last month (could show arrow up/down). - *Collected Amount This Month*: vs amount due (collection efficiency %). - *Applications Approved vs Declined*: maybe as a ratio or funnel graph (how many applied, how many approved). - *Average Credit Score of Borrowers*: possibly show distribution. - Graphs: - *Disbursements over time*: line or bar chart by month. - *Repayments over time* similarly. - *Delinquency trend*: a line showing PAR30 or number of accounts >30 days late each month. - *Loans by status pie chart*: e.g., % Active, % Closed, % In Collections. - This would be a nice visual, likely using a chart library (like Chart.js or Highcharts). - Ensure it updates either in real-time or refresh on load (computing some aggregates on the fly, which Firestore can handle with small queries or pre-aggregate with Cloud Functions updating a summary collection).

B. User Management Reports: - *User Activity Report:* List of logins, actions performed by each user (this can be based on audit logs). Good for audit and to monitor if system usage meets expectations. Fields: User, action, date/time, description. - *Productivity Report:* e.g., number of loans created by each Loan Officer in a period, number of approvals by each Manager, etc. This helps identify top performers or training needs.

C. Loan Portfolio Reports: - *Active Loans Listing:* Possibly filterable report listing each active loan with key fields (client, amount, balance, next due date, arrears if any, etc.). This is basically a data dump of active accounts, often used by management or could be exported for credit bureau if needed. - *Loans in Arrears:* List all loans that are past due, with how much and how long. This complements collections but in report format (for, say, management or regulatory). - *Loan Book Quality:* Shows distribution of loans by risk category (like how many high-risk vs low-risk loans, maybe by credit score brackets). - *Vintage Analysis:* If we really want advanced: group loans by the quarter they were issued and track their performance (like default rate per vintage). This requires capturing e.g., what % of loans from Jan-Mar 2025 have defaulted by now. That's advanced analytics often used by finance teams. - *Maturity Schedule:* Project future cash flows (sum of expected payments each month from the existing portfolio). This helps with liquidity planning. We can get that from sum of schedules of all loans.

D. Collections Reports: - *Collections Effectiveness:* e.g., how much was recovered vs how much was due in a period. Show by collector perhaps. - *Aging Report:* Standard aged arrears: e.g., total amount 1-30 days late, 31-60 days, etc. Possibly a table:

```
Age Bucket        | Number of loans | Amount Outstanding
1-30 days         | 50 loans        | R100,000
31-60 days        | 20 loans        | R60,000
...
```

- *Collector Performance:* how many calls made, promises taken, promises kept, amounts collected by each agent (especially for those taking payments). - *Closed vs open tickets:* e.g., how many cases resolved this month, how many new ones came in (collections inflow/outflow). - *Write-off Report:* listing loans written off in a period, with details (perhaps needed by accounting).

E. Financial Reports: - *Interest & Fees Earned:* Calculate total interest accrued and fees (initiation, service fees) in a period (for income recognition). - *Outstanding Balance Report:* aggregate outstanding principal and interest as of a date. Useful for accounting provisions or investor reporting. - *Provisioning Report:* If we have a formula for loss reserves (like 5% of current, 20% of 90+ day arrears, etc.), report how much provision would be required. (This is more an internal risk/finance usage). - *Cash Flow Report:* Money in vs out by day/week. Could pair disbursements vs repayments to see net cash movement (important for lender's treasury).

F. Compliance Reports: - *NCR Schedule:* Possibly prepare a regulatory report. For instance, NCR might require a quarterly report on credit granted: e.g., number of loans granted, total value, number of clients, and perhaps breakdown by category (like short-term vs unsecured etc). We can generate those aggregates: - number of new loans in quarter, - total principal, - average loan size, - maybe demographic if needed (some regulators track how many were first-time borrowers vs repeat). - *Credit Bureau Submission File:* as mentioned, we might output a file (say a CSV or XML) of all open loans each month to send to bureaus. Fields required would be: client ID, account number, opening date, outstanding balance, status (current or in arrears and how many days). - This is something credit bureaus require for their data updates. Our system can create that at end of each month. - *FICA Compliance Report:* list of all clients onboarded in period and whether their KYC docs were verified within required time. (Banks often have to report if any KYC pending, but micro-lenders too should ensure KYC done). - *Audit Trail Report:* Possibly extract from auditLogs for a given period or filter by user or by type of action (like show all changes to interest rates or all deletions etc.). Useful for an auditor to verify no unauthorized changes. - *Data Privacy report:* If a customer requests "what data do you have on me", the system could compile a report of all their data (client profile, loans, documents, communications). That might be more of an export function in client profile - could generate a PDF/zip with all info. We mention this since POPIA gives that right.

User Interface & Functionality: - We will have a **Reports menu** in the application (for roles that have access, likely managers and admins; some reports may be specialized for collectors or officers too). - It can be organized by categories or a list of available reports. Possibly a sidebar or list with "Portfolio Summary, Loan List, Arrears, etc." - Many reports will have filters (e.g., date range, branch, loan product, etc). We should allow the user to specify criteria, then generate the report. - Reports can be displayed on screen in tables and charts, and also offer options to **Export (PDF, Excel)** or print. - For charts, provide legend and the ability to hover for details. Possibly allow toggling between chart and table view. - On large data (like listing all active loans) we might page or allow export full because on-screen could be heavy. - Consider adding a **Report Builder** (very advanced, likely outside immediate scope): some systems let users create custom queries by selecting fields. We likely won't implement fully but we may mention that we allow exporting of data for those who want to do further analysis externally. - The system should ensure reports show **up-to-date data**. Real-time queries from Firestore might handle many easily but for heavy aggregation (like total interest over time) we might either compute on the fly by summing relevant fields or maintain some counters (like update total disbursed each time a loan is disbursed). - Possibly use Cloud Functions to maintain some summary documents: - e.g., a doc that keeps track of total counts (could update incrementally). - Or use Firestore's new Aggregation queries (if available). - We might also offload analytics to an integrated service if needed, but since we want everything in-house, we stick to our logic.

Compliance (Reporting): - Ensure all regulatory reports are accurate and have all required fields as per latest guidelines (e.g., if government changes interest ceilings, update product config accordingly and ensure reports reflect correct categories). - Keep archival copies of reports generated for regulatory submission (e.g., store a PDF of each quarter's NCR report as record of what was filed). - POPIA: when exporting personal data (like client list to Excel), caution that such exports are handled properly. Possibly

restrict who can export or password protect exported files if they contain personal info. - Provide a **Audit logging** of report generation if containing PII (maybe not necessary, but one might want to know if someone exported the entire client list). - In terms of surpassing competition, offering more analytics than typical is a plus (like integrating external data sources or advanced modeling results).

Interactions: - Almost every module feeds into reporting: - User mgmt (for user activity reports). - Client mgmt (for counts of new clients, etc. maybe we can show growth in client base). - Origination (for disbursement stats, approval rates). - Risk (maybe a risk report showing average credit score, risk distribution which we mentioned). - Collections (for arrears stats). - Payments (for financial reports). - Compliance (for regulatory logs). - Admin Tools might include configurations that affect report calculations (like if interest is inclusive/exclusive of VAT, etc. – our reports should account for tax if any). - Could incorporate external data for benchmarks: maybe show how our default rate compares to industry average if we feed such data (likely not in this manual's scope though).

Implementing robust reporting ensures transparency and enables better decision-making, which is essential for surpassing other LMS offerings. Next, we handle Communication, which overlaps by delivering many of the notifications needed in processes described.

B8. Communication Module

The Communication module manages all system-driven or user-driven communications with clients (and potentially with users) via multiple channels: **SMS, Email, WhatsApp**, and possibly push notifications. It handles the creation of message templates, sending of messages (automated and manual), and tracking communication history. Effective communication is key for user experience and ensuring clients are informed at each stage (e.g., application status, payment reminders), and it can also be used for marketing or announcements in a compliant way.

Channels Covered: - **SMS (Text Messages):** Good for short, immediate notifications like OTPs, payment reminders, overdue notices. - **Email:** Useful for sending detailed communications, loan statements, legal notices (though legally, some notices may require physical, but email is additional). - **WhatsApp:** Increasingly important in SA – using WhatsApp Business API to send messages to clients (which could include rich content, PDFs, etc.). Many clients prefer WhatsApp as it feels more personal and is cost-effective. - **Push Notifications:** If the client mobile app is installed, we can send push notifications (via Firebase Cloud Messaging) for things like "Your payment is due tomorrow" or "Loan approved!". - **In-app notifications:** possibly for staff users, e.g., a manager gets a notification in the system if an important thing happens (like a high-value loan is pending approval).

Message Templates: - We will create a library of templates for common messages. These templates can include placeholders for dynamic data. For example: - *Loan Application Received:* "Dear {name}, your loan application (Ref {loanId}) has been received on {date}. We will update you soon." - *Loan Approved:* "Good news {name}! Your loan of R{amount} has been approved. Please check your email for the contract and reply 'Accept' to confirm." - *Loan Declined:* "Dear {name}, we regret to inform you that your loan application (Ref {loanId}) was not approved. Reason: {reason}. You may contact us for more info." - *Payment Reminder:* "Reminder: Your loan payment of R{amount} is due on {dueDate}. Please ensure sufficient funds. Thank you." - *Overdue Notice:* "URGENT: Your loan payment due on {dueDate} is now {daysLate} days late. Amount overdue: R{overdueAmount}. Please pay to avoid further action. Contact us if you need assistance." - *Section 129 Notice (Formal):* likely this is more formal letter email attachment or SMS short "Please check your email

for an important notice regarding your loan." (We probably deliver the letter via email or physical, but could notify via SMS). - *Receipt Confirmation:* "Thank you {name}, we received your payment of R{amount} on {date}. Your remaining balance is R{balance}." - *General Announcement:* e.g., "We will be closed on public holiday" or marketing "New loan product available..." (Only send to those who opted in for marketing). - *OTP Code:* "Your verification code is 123456 (valid for 5 minutes)."

- Templates will be stored (maybe in a `templates` collection or configuration file). Fields: templateID, channel (SMS/Email/WhatsApp if different content per channel), content with placeholders, maybe a subject (for email).
- Possibly multi-language support: Could have template text in English, and if needed we could have versions in other languages (the system would choose based on client's preferred language if we had that data). But at least English default. We can plan for translations of templates if needed since SA has multiple languages – e.g., provide an Afrikaans version of key templates if target audience demands. Our multilingual support means we can have template placeholders replaced and content possibly in chosen language (but likely English default in absence of selection).

Automated Communications: - The system triggers messages automatically at various events: - After user registration: email with welcome and maybe login details (for staff user? or if client self-registers). - Application events: * On submission of application (maybe send to client a confirmation). * On approval (send SMS and email with details). * On decline (SMS+email with possibly reason). * If application is pending additional info (could send a message "Please upload X to continue your application"). - Disbursement: * On disbursement, send a message "Funds have been disbursed to your account ending 1234. Please confirm receipt." * Possibly attach payment schedule or summary in email. - Repayment reminders: * Usually an SMS 1-3 days before due date: "Your payment is due on Monday, please ensure funds." * On due date morning, maybe another reminder or if first one hasn't been sent. - Overdue: * SMS/WhatsApp at 1 day overdue politely reminding. * Another at, say, 5 days if still not paid, a bit firmer. * Section 129 formal notice via email after 20 business days (with SMS alert). - Collections agent actions: * If a collector logs a promise to pay, system might schedule an SMS confirmation: "As per our call, you promised to pay R{amt} by {date}. Thank you." * If promise date passes, maybe an automated follow-up message: "We did not receive your promised payment. Please contact us immediately." - General: * OTP for login or for verifying changes (if we implement 2FA). * Password reset emails through Firebase (template customized to company branding). * System alerts to staff: e.g., email manager if a large loan is waiting approval for too long, etc. Or daily summary emails of important metrics (some managers like daily portfolio report via email).

- We will utilize triggers in Cloud Functions or scheduled functions for many of these:
- E.g., a Cloud Function on loan status change to "Approved" triggers sending approved template.
- A scheduler runs every morning to send due date reminders for that day and next (depending on config).
- Payment failures trigger an immediate SMS ("Your debit on {date} was unpaid, please fund your account. We'll retry or contact you.").

Manual Communications: - Staff may want to send a custom message to a client or a group: - From a client's profile or loan page, a "Send Message" option where they can choose SMS/WhatsApp/email, pick a template or free-type (with caution). They might do this for things like asking for missing docs or responding to a client's query in writing. - If free-typing allowed, we should log it and possibly have some canned responses to maintain professionalism. - Bulk messaging: maybe an admin might want to send an SMS to all clients in a certain status ("We have a new branch open...") or to all overdue accounts (a generic reminder). If doing this, ensure respect to those who opted out of marketing. - We may integrate a bit of

two-way: e.g., if a client replies to an SMS, do we capture it? Twilio or similar can allow incoming messages to a virtual number. Out-of-scope possibly, but a nice feature is a conversation view. At least for WhatsApp, might consider a basic bot or allow staff to see responses. This gets complex as then it's a mini CRM messaging. Possibly not fully implemented, but mention that replies could be directed to an official phone or email for manual handling.

- **WhatsApp specifics:**
- Because WhatsApp Business API requires pre-approved message templates for outbound notifications to people who haven't initiated chat in last 24h (outside support conversations). So templates have to be registered. The ones we plan (like loan approved, reminders) would need to be approved by WhatsApp. We'll ensure our content is compliant (no personal info that violates their policy, etc.).
- We can use a BSP (Business Solution Provider) like Twilio, or direct Cloud API by Meta. For our manual, Twilio integration is easier conceptually. Twilio can unify SMS and WhatsApp sending code with slight differences.

- For WhatsApp interactive messages (like quick reply buttons), maybe not needed but could be useful (like a "Pay Now" button in a WhatsApp message that opens payment link).

- **Email specifics:**

- The system will send emails for things like loan agreements (with PDF attached), monthly statements (if we choose to send statements monthly, some lenders do).
- We can set up an email service like SendGrid or using Firebase's email extension. We'll store email templates too (with subject and HTML content possibly).
- Branding: all emails should have company logo, proper header/footer (like "you're receiving this because... contact info, unsubscribe link for marketing emails").

- Unsubscribe handling: For marketing or general info emails/SMS, POPIA requires honoring opt-outs. If a client opts out of marketing, our system will mark that and exclude them from any bulk marketing sends. But transactional emails/SMS (like statements or overdue notices) can still be sent as they are necessary for contract fulfillment.

- **Communication History:**

- All communications sent to a client should be logged. Possibly in a subcollection `communications` per client or just an array of notes.
- For example, after sending an SMS, log: {date, channel, type (template ID or manual), content snippet, status (sent/delivered if we get DLRs), user who triggered (if manual or system if automated)}.
- This allows, say, a collector to see "We already sent 3 reminders" so they can adjust approach, or a client says "I never got reminder" we can see if it was sent.

- If possible, capture delivery status: SMS and WhatsApp via Twilio provide delivery callbacks (delivered, failed). Email too (open tracking if enabled, or bounce info). We can update the log with that info. E.g., if an SMS fails (wrong number), mark it and maybe alert staff to update contact info.

- **Staff Communications:**

- Perhaps aside from client comms, internal comms: e.g., system might email an admin if an integration fails or error logs. Or notify all users of system updates (less critical here).
- Could integrate a support chat for staff, but not needed likely.

Compliance (Communications): - **POPIA & direct marketing:** We must manage consent: - We have a field `consent.marketing`. If false, exclude from any non-essential communications (like product offers or newsletters). Only send what's necessary for contract (like reminders, statements). - Provide an easy way for clients to opt out: e.g., "Reply STOP to opt out" on marketing SMS (then if we get such reply, manually or automatically update their record). For WhatsApp, maybe instruct "send STOP to opt out". - Ensure we only send communications relevant to the client's language preference if known (not required but good). - Data privacy: content of comms should not reveal too much sensitive info to avoid risk if phone stolen etc. E.g., avoid including full ID or account number. It's fine to say "loan payment" or partial details. - For overdue or legal notices, ensure compliance with debt collection rules: e.g., can only send between 8am-8pm (maybe our scheduler should avoid sending SMS at 2am, schedule to morning). - Section 129 letter ideally delivered physically or registered means. Email might not suffice legally unless client agreed to e-communication. But practically, many send via email these days with consent clause in contract. We can do both: email and also generate for physical mail. The system could output letters for mailing if needed (but that would be outside direct digital). - **WhatsApp opt-in:** Officially, businesses should get opt-in from users to initiate WhatsApp. If user gave their number for contact, that might count as implicit consent for service messages, but for marketing via WhatsApp, need explicit opt-in. We'll treat overdue, reminders, etc. as service messages (should be fine). - **Record keeping:** Keeping copies of communications (especially any that are legally relevant like the Section 129 letter content as sent). Good to archive those for proof. - **Language:** SA requirement is often to communicate in plain language and in a language the consumer understands where possible. Some credit providers provide contracts in English but explain verbally in local language. If we can, maybe having templates in a couple of major languages (English and possibly Afrikaans, Zulu) could be a plus. We can allow the template to exist in multiple languages and choose based on client's preferred language field. If no preference, default English. - **Security:** If sending documents like ID copies or statements, be cautious: maybe password protect PDF statements with the client's ID number or something known, as an extra measure (some companies do that).

Interactions: - As noted, Communication ties into: - Origination (sending out application status updates). - Payment (reminders and receipts). - Collections (overdue and follow-ups). - It uses **Integrations (B11)** such as SMS gateway (maybe a local provider or Twilio), Email SMTP/SendGrid, WhatsApp API (via Twilio or another). - The templates might be managed in **Admin Tools (B13)**: there could be an interface for Admin to edit templates content (with caution that they keep placeholders etc.). Possibly a rich text editor for email templates and plain text for SMS. - It leverages **Multilingual support** as per initial requirement, fulfilling the "default English but translatable" part. - Also, the communication logs might feed into some report (like how many SMS were sent, delivery rates, etc., maybe not primary but could be).

By building a comprehensive Communication module, we improve client engagement and keep borrowers informed and businesses efficient. Next, the Document Handling module ensures all the paperwork is managed digitally.

B9. Document Handling

The Document Handling module covers the uploading, storage, management, and verification of documents associated with clients and loans. This includes identity documents, proofs of address, income proof, signed contracts, and any other files. It ensures these documents are stored securely (given personal

and sensitive info), easily retrievable, and can be marked as verified or processed. It also optionally includes OCR capabilities to extract information from documents to reduce manual data entry.

Document Types to Handle: - **KYC Documents:** - Government-issued ID (South African ID card/book or passport). Possibly also marriage certificate if needed for married signatory in some cases, but typically just ID. - Proof of Residence (utility bill, bank statement, etc. with client's address). - **Financial Documents:** - Payslips (to verify income). - Bank statements (some lenders require 3 months bank statements to verify income and expenses). - If self-employed, perhaps financial statements or tax returns. - **Loan Documents:** - Loan Application form (if signed by client physically, maybe scanned copy). - Loan Agreement / Contract (final signed copy). - Quotation / Pre-agreement statement (if separate). - Debit Order Mandate form (if separate from contract). - Insurance policy documents (if any credit life insurance). - **Communications/ Notices:** - Section 129 letters or any important notices (we might generate these as PDFs and store them). - Any correspondence the client sends (maybe they email a dispute letter; staff can upload that to client file). - **Miscellaneous:** - Client's photo (some systems store a photo of client for identity verification, especially in branch micro lending – they might snap a picture of client holding ID). - Consent forms (like a POPIA consent form if taken separately, though likely included in contract). - For field visits, perhaps property pictures if loan was for something collateral (less likely in microloans though).

Storage and Security: - All documents will be stored in **Firebase Storage** which offers secure file storage. - We'll organize storage paths maybe as: - `/clients/{clientId}/ID.jpg` - `/clients/{clientId}/proof_address.pdf` - `/loans/{loanId}/contract.pdf` etc., or some naming convention. Alternatively, a flat structure with unique filenames using clientId/loanId metadata in Firestore. - Access to these files is controlled by Firebase Security Rules: e.g., only users with appropriate roles can access certain docs. Possibly: - Loan officers can upload and view docs for their clients. - Risk officers can view all because they need to verify. - Collections can view ID and maybe contract (so they know terms) but maybe not things like payslips (not needed for collections). - We ensure no unauthorized user can get them (like borrowers should not see others' docs, obviously). - Also, ensure if we allow client to upload via a portal, they can only put files to their own ID path (we set rules to allow writes to `clients/{theirId}/...` for that user's auth). - We might consider encrypting files for extra security (Firebase Storage is already encrypted at rest by Google, which might suffice for compliance). If needed, could encrypt file content before upload, but that complicates viewing. Likely not needed beyond what Firebase does, plus access rules. - Retention: per POPIA, after certain time (like, 5 years after account closure, KYC docs should be destroyed if not needed). We might implement a scheduled function to purge docs of clients who have been inactive for X years (with admin override). Or at least flag them. This is advanced, but mention that we have retention policies we can adhere to by configuration.

Uploading and Managing Documents (User Experience): - **Client Onboarding (internal):** When adding a new client, the officer can immediately upload ID and proof of address via the form or after saving. Perhaps in the client profile page, there's a Documents section with an upload button for each required doc type. - **Loan Application:** On the loan form, after entering details, an officer might upload supporting docs (like latest payslip). Or do it as separate step. Possibly have a checklist: ID ✔, Address ✔, Payslip ✔, so they know what's provided. - **Client Self-Service Upload:** If clients can apply online, they'd need the ability to upload photos of their ID, etc. Our mobile app (B12) will cover that (like use phone camera to capture). These should come into the same storage and be flagged for verification. - After uploading, a thumbnail or filename appears on UI. - Provide a **View** function: clicking a doc shows it (in an image viewer or PDF viewer, perhaps open in new tab if web). - Provide a **Download** option for admins if needed (maybe to forward to an auditor etc). - Provide a **Delete/Replace** function if a wrong file was uploaded (with appropriate rights,

e.g., admin or the uploader within a short timeframe). - Possibly allow tagging or adding notes to documents (e.g., note that "ID copy is blurred, need clearer one" or "Payslip shows bonus included" etc).

Verification Process: - Have a mechanism for authorized users (like risk or compliance officer) to mark documents as verified: - For each doc, a checkbox or status "Verified" that they can toggle once they've reviewed the doc for authenticity and completeness. - Example: risk officer opens ID image, checks that name matches system and photo looks like the person (if they saw person or if not, at least that ID is valid format). Then mark "ID Verified = yes". - They might do the same for address (check date and address lines) and income (check that income on payslip matches what client stated). - Once verified, maybe the interface shows a green check icon next to doc name. The `documents` Firestore record gets `verified=true, verifiedBy, date`. - If doc is not acceptable, could either leave not verified and request another. Possibly have a status like "Rejected" on a doc with reason: e.g., "Proof of address too old or not clear, please provide newer one." The system could notify the client to upload a new one if client has portal. - Only when all required docs are verified should an application proceed to final approval ideally (we could enforce that in loan approval process – if not all required docs verified, prevent approval or at least warn manager).

- **OCR (Optical Character Recognition) & Data Extraction:**
- We aim to reduce manual entry by extracting text from documents:
  - Use Google Cloud Vision or other OCR on ID documents:
  - For SA ID, the newer ID card has a Machine Readable Zone (MRZ) and a QR code; the older ID book has a barcode and printed text. If we wanted to be fancy, we could scan the barcode via device if using mobile (some apps do that to capture ID number quickly).
  - Alternatively, OCR the printed text: get the ID number, name, surname, etc.
  - Compare with what's in system. If mismatch, flag possible error or fraud (like if ID number on doc is different from what client gave, something's off).
  - Also, check ID number validity: maybe our system does that anyway by formula, but OCR could confirm the actual doc's number.
  - For proof of address: OCR can attempt to find text like an address or the person's name on a utility bill. Harder, but at least it could detect the person's name to ensure it matches, and maybe date on the bill. Could be challenging but an advanced feature mention: some companies do automated address verification by reading the PDF bill and extracting address lines then comparing to what’s on file.
  - Payslips/bank statements:
  - Very advanced, but we could use OCR or specialized parsing to read a payslip total and date, or to scan a bank statement for income entries to auto-calc average income. That is high complexity and perhaps beyond scope to fully implement. But we mention it: e.g., integrate with a service that can parse bank statements (some bureaus offer that as an API to output an affordability assessment from a PDF statement).
  - Or simpler, just store the docs and rely on officer eyeballing for now, which is typical for micro-lenders manually.
  - Implementation: We can have a Cloud Function that triggers when a file is uploaded to certain storage path. It calls Vision API to extract text, then maybe saves the text result in a field in the Firestore doc or triggers some verification logic.
  - E.g., after ID upload, function extracts ID number and maybe the name, then cross-check with client record. If mismatched, mark doc as "verificationFailed" or something.

    - If matched, could auto-mark verified (or at least inform the verifier that it matches).
    - This level of automation would put us ahead of many systems.

- **Document Generation & Storage:**

- The module also deals with creating documents like the loan agreement or letters:
    - After loan approval, our system will produce a PDF contract. We'll save that to storage (e.g., `/loans/{loanId}/LoanAgreement.pdf`). Mark it as generated and not verified (since needs client signature).
    - If the contract gets signed digitally, maybe we generate a new version "LoanAgreement_Signed.pdf" or embed a signature image and re-upload.
    - For the Section 129 letter, similarly generate a PDF and store.
    - So, the document handling also includes output documents not just uploads.
    - Possibly integrate an e-signature: We could use a third-party like DocuSign or SignRequest API, but could also do a simpler approach with OTP. If using an external, that might produce a signed doc which we then store.
    - But we can do something simpler: accept an "I agree" via OTP code, then mark contract accepted and consider that as good as signature (some debate in legality, but let's assume it's okay with proper audit log).
- It's important to maintain version control if needed: e.g., if a contract is regenerated due to terms change, keep old draft too possibly. Or we can override, but version could avoid confusion.

- We'll likely just generate final version after terms final and use that as single contract file.

- **Document Viewing and Download by Clients:**

- In the mobile app, a client may want to view their contract or statements or any doc like that. We can allow that: e.g., "My Documents" section for clients showing their own docs (like loan agreement, perhaps receipts).
- Under POPIA, they have right to get copy of anything they signed, so providing contract in their app or via email is good.
- Also, maybe store receipts as PDF if needed? Usually an SMS or small confirmation might suffice for receipts, but some might want official receipt for a payment, which could be a PDF as well.
- These would just be additional conveniences.

Compliance (Document Handling): - **FICA & KYC:** Ensuring we collect and verify ID and address meets FICA. We should mark date of verification and who did it (for audit). - We should store documents for at least 5 years from end of relationship, per FICA retention rules. Then they should be destroyed. We should implement or plan a purge after that retention time, as holding ID copies longer could violate law. Possibly we set up an automated flag to prompt admin after time passed. - **Privacy & Security:** These documents contain highly sensitive personal data. We must: - Limit access to need-to-know roles. - Never send them over insecure channels without encryption (for instance, not emailing an ID copy without at least password protection, unless absolutely needed). - Possibly watermark documents that staff download with "Confidential". Eh, maybe not needed. - **Integrity:** We want to ensure documents aren't altered maliciously: - If storing digitally, likely fine. If we allow deletion or replacement, log it (who replaced an ID doc and when, to avoid someone swapping documents incorrectly). - Could also use file hashes to ensure the stored file remains unchanged (not critical, but if wanting to detect tampering). - **E-sign Legal:** If using digital loan agreement, ensure it contains appropriate clauses that by electronic acceptance it's binding, as allowed by

ECT Act for credit agreements (credit agreements likely allowed to be electronic, except maybe some complexities for suretyship not in microloans). - **Audit:** Keep track of who viewed what document and when if possible (this is heavy, but for privacy compliance, nice to have an audit log if an employee opens a client's ID copy, so any inappropriate viewing can be tracked. Possibly beyond typical scope, but it's a gold-star compliance practice).

Interactions: - Document module touches: - **Client Management:** where documents are linked to client and required for verification. - **Loan Origination:** requiring docs for approval, generating loan docs. - **Risk Management:** verifying docs as part of risk assessment and fraud prevention (like checking ID and proof of income). - **Collections:** if legal action, documentation might be needed (like contract copy for lawyers, ID, etc., which we have stored). - If integrated with external checks: We might add an integration like a credit bureau's bank statement analyzer (some credit bureaus have a product where you upload bank statements and they return an affordability analysis). That could be triggered by doc upload (like run analysis on PDF). - Admin Tools might include Document types config (though we can hardcode typical ones). - **AI** might come into play by reading documents (as mentioned OCR and maybe verifying authenticity of ID via machine learning comparing photo to user selfie or something advanced). - **Mobile App** (B12) will provide a user-friendly way to capture and upload these docs.

This thorough document handling ensures the platform is paperless and efficient (Mobiloan bragged "100% Paperless" [20], which we match by digital docs and e-sign). Next, we cover Compliance and Regulatory module that ties together the policies enforced in all others and tracks those specifically.

B10. Compliance & Regulatory

The Compliance module ensures that the entire system and lending operation adhere to all relevant laws, regulations, and industry standards (especially in South Africa). Rather than a separate functional process, it's a set of checks and balances integrated across features, but we list them here explicitly. This module covers data protection (POPIA), credit regulations (NCA/NCR requirements), anti-money laundering (FICA), affordability rules, and any required regulatory reporting or audit support. It overlaps with prior sections, but here we enumerate all compliance-related features per the request to list "all compliance checks required per feature."

Compliance Areas and System Implementation:

1. **POPIA – Data Privacy & Protection:**
2. *Consent Management:* The system obtains consent from clients for processing their personal data at application time (likely included in contract terms or a separate tick box). We store `consent.personalData=true` with timestamp. Also separate marketing consent, as discussed.
3. *Data Security:* All personal data stored (contact info, financials, documents) is protected via several measures: encryption at rest (Firebase does this), secure access controls (role-based restrictions, as implemented in user management and security rules) [12], and network security (all data transfer is over HTTPS/SSL).
4. *Access Control:* Only authorized roles can access or edit sensitive data. For instance, only certain roles can see full ID numbers or download documents, etc., thereby enforcing least privilege principle.
5. *Audit Trails:* The system maintains logs of key actions (viewing or exporting sensitive data, data changes) [9]. If an admin exports a client list or a user views a hundred client IDs, these can be flagged in logs to detect potential privacy breaches.

6. *Data Subject Rights:* The system can assist in fulfilling client rights:
    - Right to Access: We can generate a report of all information we have on a client on request (as noted, possibly a one-click assemble of profile, loans, comms, docs).
    - Right to Correction: If a client finds incorrect info (say wrong address), our system allows editing it (with appropriate verification for critical fields like ID).
    - Right to Deletion: If a client requests deletion (and we have no overriding legal need to keep data), the system should allow anonymizing or deleting their personal info. Because of NCA retention requirements, we likely cannot delete loan records for 5 years, but after that or if data is not tied to an active loan, an admin could anonymize the personal identifiers. We might implement a function to scrub personal details (replace name with "Removed", etc.) while keeping transactional records for stats.
    - Right to Object to marketing: We have opt-out flags and we honor them by excluding from marketing communications.
7. *Breach Notification:* Not a direct system feature, but we should be prepared to log and detect unusual data access which could indicate a breach. If a breach occurred, we have audit logs to analyze what data was accessed.
8. *PII Minimization:* In displays and communications, we avoid unnecessary full PII exposure (e.g., mask ID and account numbers where possible as mentioned). This reduces risk if, say, an email is intercepted or a user sees a screen in public.
9. *Third-Party compliance:* Ensure our integration partners (SMS, WhatsApp provider, etc.) are also compliant (e.g., using Twilio which is GDPR/POPIA compliant). We'll mention in docs that we chose providers with strong data protection track records.

10. *Policy & Training:* Out of software scope, but the system can have a pop-up or acknowledgment for staff to confirm they understand POPIA responsibilities on first login or annually.

11. **National Credit Act & NCR (Responsible Lending):**

12. *Registration:* The system should store the micro-lender's NCR registration number and include it in relevant documents (contracts, etc.).
13. *Interest & Fee Caps Enforcement:* The system ensures that loan products are configured with interest and fee parameters that do not exceed the maximum allowed [11]. For example:
    - Unsecured loans: max interest ~28% p.a. If admin tries to set 30%, system should reject or warn.
    - Short-term loans: enforce 5%/month on first short-term loan, 3% on subsequent as per reg [21].
    - Initiation fees: must follow formula (we could code that formula and if someone tries to set a higher fee, override it to cap).
    - Service fees: cannot exceed the regulated monthly cap (which is R60 + VAT currently).
14. *Affordability Assessment:* As described in Origination (B3), before approving a loan, the system calculates affordability and ensures it meets the criteria. The criteria themselves (like allowable disposable income percentage) should align with NCR guidelines (the law says consider existing debts and living expenses; many use around 30% of net as a safe harbor). We might let admin set this threshold, e.g., in config "Minimum Disposable Income % = 20%" and "Max debt-to-income = 40%" and system checks against those. Any override of a failed affordability is logged with reason to defend the decision (in case of later dispute, we have rationale).
15. *Credit Bureau Check:* NCA requires checking credit history for new credit (especially for amounts above a small threshold or any term credit). Our integration ensures a credit check is done for each

loan, or if not, require a manual override with reason. We log date of check and reference number of report to prove compliance. If client's credit profile indicates they're under debt review or had recent defaults, system flags it and likely prevents auto-approval.

16. *Reckless Lending Prevention:* The combination of credit check + affordability + proper KYC should prevent giving a loan that the client cannot afford or shouldn't receive. If our system is used properly, an approval implies these were green. For audit, we can produce for each loan a "credit assessment record" that shows income, expenses, outcome, credit score, etc.
17. *Contract & Disclosure:* The loan agreement generated includes all required information as per NCA Section 93 (like all costs, installment schedule, interest calculation method, rights and obligations, cancellation rights if any, cooling-off if applicable, etc.). We likely use a standard contract template vetted for compliance. The Pre-agreement quote given to client is also as per regulations (with 5-day validity, which we note).
18. *Consumer Rights Info:* Perhaps include in communications the statements of rights (like "If you experience difficulty, you have the right to consult a debt counselor" etc., which are sometimes in the contract).
19. *Regulatory Reporting:* The system compiles required reports for the regulator:
    - Possibly NCR Monitoring returns (somewhat optional for small credit providers, but if asked, we can easily extract summary info).
    - **Credit Bureau Data Sharing:** It's required that credit providers supply data to all registered credit bureaus. Our system helps by producing that data (as above in reports, monthly submission file).
20. *Arrear Handling Compliance:* As covered in Collections:
    - Issue Section 129 letter after 20 business days of default (we schedule ~ default+20 days).
    - After letter, wait at least 10 business days before legal action. We ensure in system that the "legal handover" action cannot be done until that period passed. Perhaps the system can alert "Cannot handover until dd/mm/yyyy to comply with S129 timeline".
    - The letter content should include necessary wording (like advising the consumer of their right to approach a debt counselor, etc. as required by NCA).
21. *Interest in Default:* NCA (or regulations) may limit interest once a loan is in default or handed over. For example, sometimes if a matter goes to court, the interest rate might change to prescribed rate if judgment given. Our system isn't tracking legal judgment stage explicitly unless we extend to that, but at least if we mark a loan as "Handed Over", we might freeze further interest accrual or ensure any further interest doesn't exceed what's allowed. Actually, under NCA, interest can continue to accrue at the rate in contract until a certain point or until settlement in many cases (except for certain short-term scenarios).
    - However, there is an "in duplum" rule in SA common law (incorporated in NCA somewhat): once interest accrued equals principal, interest stops accruing (the total interest that can accrue is capped at the outstanding principal at time of default, effectively). So our system should enforce the **in duplum rule**: e.g., if a client stops paying, the interest and fees added after default cannot exceed the amount that was due when they defaulted. We'll implement a check: if accrued interest is about to exceed principal due at default, stop accruing further interest. This prevents an infinite growth of debt, as required by law.
22. *Marketing compliance:* If the system is used for sending promotional messages (maybe in Communication), ensure they comply with NCA marketing rules (like no false or misleading advertising, interest and fee must be clearly stated if any ads; though our system likely not generating ads, just maybe cross-sell messages).

23. *NCR compliance in agreements:* If loans are small and short-term, ensure we apply the correct maximum initiation fees etc. This is already in our config but just reiterating that each product is set up as either "short-term credit transaction" vs "unsecured credit transaction", etc., with relevant rules.

24. **FICA – Anti-Money Laundering (KYC/AML):**

25. *Customer Identification:* We do KYC (Know Your Customer) by collecting ID and verifying it. Also, capturing addresses. The system does not allow loan disbursement until FICA docs are provided and verified (we can enforce: if `FICA.verified != true`, then disburse button is disabled).
26. *PEP/Sanctions Screening:* Not explicitly asked, but an AML measure is to check if the client is on a sanctions or politically exposed persons list. Some micro-lenders might not do this rigorously, but to surpass, we can mention integration with a watchlist (maybe via a service or the government's list). If integrated, on client creation, run their name through a sanctions API (there are databases like OFAC, UN sanctions; or local Treasury list). If a match, alert compliance officer.
27. *Record-keeping:* FICA requires keeping KYC records for at least 5 years from end of relationship and being able to retrieve them to show regulators. Our system's document storage and retention policy covers that (we won't delete earlier).
28. *Reporting Suspicious Activity:* If the system notices suspicious behavior (like someone making strange large payments or multiple accounts using same info, etc.), it could flag it for compliance. Possibly out-of-scope to automate much, but we can allow manual flag by staff "Mark as suspicious", and log it. Reporting to regulators (Financial Intelligence Centre) is usually outside core system (done by compliance officer manually), but our system can help by providing data needed (client profile, transactions).
29. *Large Cash Transactions:* If a client makes or receives cash above a threshold (like R25k, FIC Act requires a CTR report), our system can flag that. If we recorded that a disbursement or repayment was done in cash and > threshold, trigger an alert "Cash transaction of R30k by X on Y date - consider regulatory report". The compliance officer can then use that info to file the report.
30. *Source of Funds Declaration:* Sometimes required to ask how the client's funds are sourced (especially if large loan). Not typically for microloans, but could incorporate if needed (like a field "Source of income" which is essentially employment already).
31. *Branch compliance:* We might in Admin Tools have fields to mark which branch has which FICA officer, etc., but not necessary to detail.

32. *Ongoing monitoring:* If a client has multiple loans or unusual patterns (like paying off and re-borrowing quickly repeatedly, could that be money laundering?), probably not main scenario in microloans, but at least track multi-loan use.

33. **Affordability (some points covered under NCA but highlighting):**

34. For each loan, the system ensures:
    - Income and expenses are captured and that resulting net is positive post-loan.
    - If client has multiple loans with us, include their existing installments in expense.
    - If credit report shows other debts, ideally incorporate those obligations (some advanced integration can fetch monthly obligations from bureau data).
    - Possibly maintain a rule: total monthly debt repayments (including this new loan) should not exceed, say, 50% of net income. If exceed, mark as fail.

    - The system won't allow loan approval if check fails, unless override. And if override, require a recorded motivation like "Client has additional household income not reflected" or "Will be reducing other expenses significantly" (thin justification but something).
35. This is a direct measure to avoid reckless lending and is indeed mandated by regulation that a proper affordability assessment is done with evidence (like a payslip or bank statements).

36. The system prints on the assessment some outcome that could be stored in loan file or client file to show compliance.

37. **System & Data Security (IT compliance):**

38. While not a law, adhering to best practices like OWASP for web security is important (avoid XSS, injection etc. in our code).
39. Ensure user passwords are stored properly (we use Firebase Auth so it's taken care of).
40. Possibly enforce password strength and periodic change if internal policy demands (Firebase can't force change, but we could remind or make an admin toggle to reset password).
41. Multi-factor (we have that option).
42. Session management: auto-logout after period of inactivity, to prevent unauthorized access if user leaves terminal open.
43. Logging out devices: admin ability to force sign-out a user (if suspect compromise).
44. Data backups: we should keep backups of critical data (though in Firebase, backups can be scheduled). Ensure we can recover in case of accidental deletion or corruption (maybe beyond our dev but mention scheduled backup).
45. Business continuity: The system being cloud and available in multiple zones covers uptime, but mention our architecture inherently achieves high availability.
46. Audit & Compliance checks: If an external auditor or NCR comes, our system can easily generate whatever data needed to demonstrate compliance (the reports and logs).
47. Privacy by design: We could mention we've done PIA (Privacy Impact Assessment) internally to ensure compliance features, which we basically have enumerated.

In summary, **Compliance is woven through all modules**. For each feature as requested: - **User Management:** deals with user access and privileges (prevent unauthorized data access) [12]. - **Client Management:** covers KYC (FICA) and POPIA consent. - **Loan Origination:** covers NCA (affordability, interest caps, disclosures). - **Risk Management:** covers credit checks (NCR) and AML flags. - **Collections:** covers NCA collections process (Section 129, in duplum rule) and POPIA (respect times, privacy in communications). - **Payment System:** covers DebiCheck compliance, in duplum (stop interest once threshold), fee limits. - **Reporting:** covers regulatory reports (NCR, credit bureau) and audit trails. - **Communication:** covers POPIA (marketing opt-out) and consumer communications fairness (no misleading info, proper language, time). - **Document Handling:** covers FICA recordkeeping, contract contents (NCA required content) and data protection of docs.

By centralizing compliance considerations here and also integrating them into each relevant feature, our LMS will stand out as a system that not only is functionally rich but also inherently ensures legal compliance and data security at every step. This significantly reduces risk for the lender using it.

B11. Integrations

The Integrations section outlines how our LMS connects with external systems and services to extend functionality and streamline operations. We will discuss each integration required: **NuPay**, **ALLPS**, **Ozow**, **TransUnion**, **Experian**, **PayFast**, as well as mention any others implied (e.g., possibly credit life insurance provider, maybe Sudonum or something for phone, though not explicitly asked). Each integration will include what it is used for, how it works within our system, and any specific considerations.

B11.1 NuPay Integration (Debit Order Processing): - **Purpose:** NuPay is a payment solutions provider specializing in debit order collections, particularly known for Authenticated Early Debit Orders (AEDO) and Non-authenticated (NAEDO) as well as newer DebiCheck mandates [6]. It's widely used in microfinance for collecting loan installments. - **Integration Functionality:** - **Mandate Creation:** When a new loan is finalized, our system will communicate with NuPay to create a debit order mandate. If using DebiCheck (authenticated), NuPay will handle the request to the client's bank. We provide the client's details and schedule (dates and amounts). The client typically authorizes via their bank app or an SMS code (which is outside our system but initiated by NuPay). - **Debit Scheduling/Submission:** Prior to each due date (or as a batch), our system sends a debit request to NuPay: * If AEDO: Our user might have a NuPay terminal or interface if card-present, but AEDO in modern context likely replaced by DebiCheck. We focus on automated via API: we send the account details, amount, and desired debit date. * If NAEDO: We send instructions to queue debit attempts on salary date's early window. NAEDO allowed multiple attempts and random priority, which is older system. Possibly still in use for some lenders. NuPay would handle the multiple attempt logic, we just give the instruction once. - **Real-time/Batch API:** We likely use a web-based product of NuPay (they have a web portal and possibly APIs). According to ITWeb, NuPay also has web-based e-commerce and NAEDO via API [22]. We'll assume an API where we can POST debit instructions and query status. - **Result Handling:** After each scheduled debit, NuPay will return success or failure information. Possibly: * For DebiCheck/AEDO, result might be immediate (if card present AEDO, it's immediate approval or decline). * For NAEDO, result may come only after processing (later in day). * We'll implement a listener or a polling: maybe daily retrieve a report of all transactions, or NuPay can call a webhook with results. * Our system updates the Payment record with the status. If failed, triggers collections as described. - **Terminal Integration (if any):** Some micro-lenders use physical NuPay terminals for AEDO where client inserts bank card and enters PIN to authorize debit mandate (that's older AEDO process). If needed, our system might not directly integrate that hardware (that's more offline), but we can record the outcome. We'll lean on web integration and assume if a branch has a terminal, they use it separately and then log reference in our system. - **Security:** We'll store NuPay API credentials securely (in Cloud Functions config). Communications with NuPay API are via HTTPS, and possibly we must handle signing or encryption if their API requires (some older might use certificate-based SOAP or sFTP). - **Testing:** We'll likely have a sandbox mode for development. Possibly a toggle in Admin to use test credentials. - **UI Integration:** * On loan screens, maybe show mandate status (e.g., "Debit Order Mandate active via NuPay" or if not authorized yet, "Awaiting DebiCheck confirmation"). * Provide a button "Verify Mandate" or "Send Mandate Request Again" if needed (for DebiCheck, if expired). * On the Payments or Disbursement screen, maybe show upcoming debits and allow admin to cancel a debit request if needed (if client pays manually ahead of time). - **Reconciliation:** NuPay likely provides a daily settlement report (which accounts were successfully debited and amount deposited to our account minus fees). We should reconcile that with what our system expects. Typically, our Payment records for each due would align with that, but any discrepancy (like partial payment or a reversal later) should be updated.

B11.2 ALLPS Integration (Payment Collections): - **Purpose:** ALLPS (All Payment Solutions) is another debit order and payment management provider. ALLPS is PASA registered and offers AEDO, NAEDO, conventional EFT debits, social grant debits, and eWallet payouts [5]. It seems to have a web interface (Amplifin/ Maxstore). - **Integration Functionality:** - Very similar to NuPay in that ALLPS can schedule debit orders. Possibly chosen by some lenders as an alternative or in addition. We integrate it to provide flexibility; maybe lender can use one or the other depending on cost or success rates. Or maybe one for certain bank sets and another for others. - *Debit Order Setup:* When a loan is approved, we register a debit order with ALLPS API: * Provide bank account and debit dates. They handle DebiCheck as well (Allps mention "Authenticated Early Debit Order" too). - *Multiple Streams:* ALLPS supports multiple streams: * AEDO (authenticated via card and PIN). * NAEDO (non-auth). * EFT Debits (standard debit orders on random time, usually processed midday). * Debits against Grindrod accounts (for social grants) – interesting special case if client's income is social grant, they handle differently (we might not go that deep but good to mention that integration can handle social grant debit intricacies). * eWallet payments – Allps can also pay out to eWallet (which we can use for disbursement if needed). - Our system could allow which type to use (maybe default NAEDO and if client is grant recipient, use the Grindrod method). - *API vs File:* ALLPS might allow uploading a batch file or using an API. Ideally API to automate. If file (like a daily CSV of debits) then our system can generate it and possibly SFTP to them. But that introduces more manual steps. We assume API via an integration partner like Amplifin's Maxstore interface. - *Result Handling:* As with NuPay, we retrieve statuses from ALLPS. They likely provide a return file or API with results (paid, bounced, reason codes). - *UI Consideration:* Possibly in admin settings, choose the provider for debit orders (NuPay vs ALLPS). Or even by branch if needed (some branches might use one vs another). - *Payouts with ALLPS:* We could also use ALLPS to do disbursements via eWallet. If that's desired, when disbursing, an option "Pay via eWallet" triggers ALLPS's eWallet function, which presumably sends an SMS to client with a withdrawal code. This is useful if client unbanked. If we include this, we'd have to capture client's mobile and ID for eWallet and integrate that. It's a nice extra: eWallet (like FNB eWallet) means we deposit money to a voucher they withdraw at ATM. * We'll mention: Our integration with ALLPS enables alternative payout methods such as eWallet for unbanked customers [18]. - *Fees:* ALLPS might deduct a fee per transaction or monthly. The system could store those in a config and perhaps record in a ledger, but that's more accounting. We can at least track number of transactions for cost analysis.

B11.3 Ozow Integration (Instant EFT Payments): - **Purpose:** Ozow is a South African fintech enabling **instant EFT** payments. It allows clients to pay directly from their bank account (internet banking) in real-time without needing to manually send proof. It's used as an alternative to card for online payments, often with lower fees. - **Integration Use Cases:** - **Repayments by Clients:** If a client wants to make a loan payment manually (outside the debit order schedule or if debit failed), our system can present an Ozow payment option. The client would choose their bank via Ozow, log in, and authorize a transfer that Ozow confirms instantly to us. - This is especially useful for on-demand payments: e.g., on the client portal "Pay Now" button leads to Ozow. - Or for initial disbursement, some use Ozow to verify bank and deposit but that's not typical (disburse via EFT is fine). - Perhaps for application fee or such if we had (we don't have application fees as separate due to NCA possibly). - Ozow can also be used by collections: e.g., collector sends an Ozow payment link through SMS/WhatsApp, client clicks and pays. - **Integration Functionality:** - We integrate via Ozow's API or inline widget: * We generate a transaction request with amount, reference, and callback URL. * We either redirect the client (in web) or open an in-app webview (in mobile) to Ozow's interface where they complete payment. - **Callback Handling:** Ozow will call our system (to a specified endpoint) with status once payment is successful (or failed). We update the corresponding Payment record (initially marked pending). - *Security:* We have merchant credentials (Ozow id, private key) to sign requests. We store them securely. - *UI Flow:* For an internal user (like collector or staff sending link), the system might

generate a link and present it so they can send to client. For client using portal, it's direct. - *Reconciliation:* Ozow likely also has a dashboard, but our callback updates our records in real-time (or we might also fetch unsettled transactions daily to double-check none missed). - *Benefits:* It's instant, so we can credit the payment immediately and possibly if a client is making an overdue payment, unlock their account or remove them from collections quickly. - *Consideration:* If a client doesn't complete the payment or it fails (maybe no funds or they cancel), we handle that gracefully (maybe notify them or allow retry).

B11.4 PayFast Integration (Card & EFT Payments + Payouts): - **Purpose:** PayFast is a popular SA payment gateway for credit/debit card payments and also provides instant EFT and other methods. It can serve two roles: receiving payments (like Ozow, but primarily card) and sending payouts (they have a Mass Pay feature). - **Integration for Incoming Payments:** - **Card Payments:** Many customers might prefer paying via card (maybe a debit card, which ultimately pulls from bank but through card rails). * Our system can integrate PayFast to accept card payments for loan repayments (for example, a "Pay with Card" option on the portal). * Also, initial down-payments if any (some lenders require first installment upfront) could be via card. - **Alternative EFT (PayFast EFT):** PayFast also offers Instant EFT similar to Ozow. We might not need both Ozow and PayFast's EFT; we could pick one. But to surpass, we integrate both giving clients choice. - **Wallets:** PayFast supports some wallets like Masterpass, but not critical to mention. - **Process:** * We create a payment order via PayFast API, including amount and client details, and a redirect URL for them to complete payment on PayFast's site. * The client goes to PayFast’s payment page, enters card details or chooses EFT, etc. * PayFast returns to our site with a token. Also, they send a secure callback (ITN - Instant Transaction Notification) to our server with the result. * We verify the signature and update the payment record as success or failure. - **Recurring:** PayFast offers recurring billing, but since we handle recurring via debit orders, we might not use PayFast for scheduled recurring (though they do have a tokenization option to charge cards monthly, but that would circumvent DebiCheck and might cause higher decline rates and fees; still an option if debit orders fail and client consents to card charge). * If we wanted, we could implement storing a token of client's card on first payment and then charging it for installments, but need their consent and it's a plan B usually.

- **Integration for Payouts (Disbursements):**
- PayFast has a Payouts API (Mass Pay) that allows sending money to multiple recipients either to their bank accounts or maybe as a voucher. It's typically used to pay sellers, but a lender could use it to pay loan amounts to clients.
- If we use PayFast for disbursements:
    - We send through PayFast the recipient’s bank info and amount, they process the EFT. It could be immediate or same-day.
    - Benefit: We handle all through one platform (they take a small fee per payout).
    - PayFast would notify once payout is completed. We then mark loan as disbursed.
    - This might simplify not dealing with our own bank payments, but it adds a cost. Up to the lender if they want to use that or manual EFT. It's an offering we can mention as integrated.
- We have to fund our PayFast account to do payouts if needed, which is a financial operations point, but system-wise just ensuring we know if a payout fails (like incorrect account).

- **Security & PCI:**

- PayFast handles card data on their side, we never see card numbers, so compliance burden is on them (we just integrate via their secure forms).

- We ensure our IPN/ITN listener is secure (verify with PayFast's signature as per their docs).

B11.5 TransUnion Integration (Credit Bureau): - **Purpose:** Retrieve credit information for clients from TransUnion, one of the major credit bureaus in SA. This is vital for risk assessment. - **Integration Functionality:** - **Credit Report Pull:** Provide an API call to get either a full credit report or at least a summary (score + key indicators). - We send client's identification details: typically ID number, full name, maybe DOB and address for verification. (TransUnion needs those to locate consumer file). - Options: * We might get a full PDF report and a structured response with score and some data. Or some product like TransUnion's Consumer Credit Score and some attributes (like number of defaults, etc). * We integrate via TransUnion's web service (likely SOAP or REST with XML). Many bureaus still use SOAP with secure certificates. * Possibly use a middleman integration (like XDS aggregator or a service like LendProtect) if direct is complex. - **Response Handling:** * On success, we parse the data: extract credit score, any critical flags (e.g., "Has Judgment", "Under Debt Review", credit summary totals). * Save those to our database: e.g., in `loans/application` object or `clients/creditReport`. * Attach the PDF or raw report in our storage if needed for later review (maybe store it encrypted or restrict who can open). - **Display:** Show the credit score and perhaps a grade or description ("Score 600 - Medium risk"). Possibly also number of accounts, delinquent accounts, etc., in the UI for risk and approval managers to see. * We might not show the full report to loan officer if not needed (just score). But risk officer or manager can click "View Full Credit Report" which opens the PDF or a detailed view. - **Timing:** Ideally do it in real-time when needed (like user clicks "Pull Credit" or system triggers at application submission). * If bureau is slow, we might show a "Fetching credit info..." spinner. It should typically return within seconds. * If bureau is down or returns no data (thin file), handle that by alerting user or allow proceed with caution flag. - **Cost management:** Each pull costs money for lender. Some systems allow risk officer to choose to pull or not for borderline cases. We may allow only one pull per app unless explicitly triggered again (to avoid duplicate charges). * Possibly track usage count and show a monthly count in admin for reconciliation. - **Alternate ID:** If some clients have no ID (rare, maybe passport), TransUnion might not have data. Then we might try Experian or use alternate identification (like drivers license not common for credit).

- **Extra TransUnion features:**
- They may offer a product called TransUnion Affordability or something that estimates disposable income. Or "TransUnion Summit" platform for micro lenders (just speculation).
- They likely have a service to get a suggested credit limit or risk segment. If available, we could incorporate that in our risk logic.
- **Integration Setup:**
- We'll have credentials for TransUnion (username, password, possibly a client code or certificate).
- Use a staging environment for testing if provided, else demo data.
- Probably only accessible from backend server (we do it in Cloud Function, not from client side, since it's sensitive).
- **Data Protection:**
- The credit data is also personal info. We must treat it as confidential. Only store what's needed (score, key points). The full report if stored should be highly restricted, maybe encrypted or only accessible to risk role.

- We should not expose bureau data to the client (like they might dispute something, but that's done through bureau, not our system).
- Comply with credit bureau usage policies: e.g., must have consent (we have that from client), must use data only for the purpose of credit assessment, etc.

B11.6 Experian Integration (Credit Bureau): - **Purpose:** Redundant or supplemental credit data from Experian, another major bureau. Some lenders query multiple bureaus to get a fuller picture (since not all data is on one bureau). - **Integration Similar to TransUnion:** - Provide client identification to Experian's API, retrieve credit score and report. - Possibly Experian's scoring and info might differ (they have their own scoring system). - We could use Experian as an alternate if TransUnion fails, or use both and perhaps consider the worst-case or something. Or choose one as primary. - Or maybe integrate both and let admin decide which to use (or use both for more thorough check). - **Differences:** - Experian's API likely similar complexity. They might also provide a PDF or just structured data. - If integration overhead is high, in an MVP one might choose just one bureau. But to "surpass all global", having multi-bureau support is a plus. - We can implement an abstraction in our code: e.g., a "CreditCheckService" that can call either TransUnion or Experian or both. Possibly cross-verify data (like if one bureau says score X and other says Y, maybe flag discrepancy). - In SA, there are also other bureaus like XDS and Compuscan (which is now Experian I think acquired), but focusing on TransUnion & Experian covers majority. - **Usage:** - Possibly, if one bureau reports debt review and another doesn't, trust the positive (i.e. assume they are under review). - If scores differ significantly, maybe take the lower/higher, up to policy. - **Cost & Efficiency:** - Hitting two bureaus costs double, but yields more data. Possibly only do Experian if TransUnion returned thin file, or vice versa. - Could have a rule: if TransUnion score is borderline, fetch Experian for more insight. - Or run both for every app for thoroughness.

B11.7 Other Integrations Mentioned or Implied: - The user explicitly listed the above, but let's ensure coverage: - **PayFast** and **Ozow** for payments (done). - **NuPay** and **ALLPS** for payments (done). - **TransUnion** and **Experian** for credit (done). - They also mentioned "NCR, FICA, affordability" under compliance, which we covered but not an integration - those are policies. - Not mentioned but competitor had "Sudonum Setup" in their system: - Sudonum might be a call tracking/number masking service (the quote from mobiloan site suggests they partner with them [23]). Possibly to allow collection agents to call clients via a proxy number to protect personal numbers or to record calls. - If we want to surpass, we could mention integration with a telephony service: * e.g., integrate with Twilio or local provider to enable click-to-call and call logging. Or to allocate virtual numbers to track calls (Sudonum does dynamic number insertion). * This is more of a nice-to-have: like call center integration. Could allow calls initiated from system and automatically log call duration etc. * It's advanced and not required by user query, so maybe a brief mention: "Integration with telephony (e.g., Sudonum/Twilio) to allow one-click calls and record call outcomes" in passing. - **Credit Scorecards/Decision systems**: - Possibly XDS (Xpert Decision Systems) was referenced in user guide. Actually, we saw an XDS integration in mobiloan user guide [7] . * XDS is a bureau too (third largest in SA) and they also provide decision analytics. * If some lenders prefer XDS, our architecture can integrate any bureau with appropriate API. We won't detail all, but note that our system is flexible to integrate with any credit data provider as needed, thanks to modular design. - **Fraud database**: * There's SAFPS (fraud prevention service) which has a database of victims of identity fraud. Possibly through XDS or direct, could integrate to check ID against that. * Not asked, so skip depth but mention maybe "We can also integrate with fraud databases if provided (like SAFPS) for extra checks". - **ID Verification (Department of Home Affairs)**: - The Dept of Home Affairs has an interface (via services like *Home Affairs National Identification System (HANIS) or a web service to verify an ID's existence and match name). - Some credit providers use third-party providers (e.g., Experian and others offer ID verification built in). - If we can use the credit bureau's ID verification that suffices. Or direct if available. - It can be considered part of KYC integration.*

*Possibly mention "The system can integrate with DHA data to validate identity in real-time when available." - Bank Account Verification: - Services like LexisNexis or some bank API can verify that an account number belongs to a certain ID. - That ensures the bank details given are truly the client's. - This prevents fraud (someone giving someone else's account). - We can integrate a product often called "AVS - Account Verification Service" which banks provide or via a bureau. - Possibly if Allps/NuPay can verify the account as part of DebiCheck process, they might (DebiCheck inherently is confirmed by client via bank, so probably okay). - But for payout, verifying account can avoid sending money to wrong account. - Could mention "Integration with Bank Account Verification services to confirm client's bank details (ensuring payouts and debits target correct accounts)." - Credit Bureau Data Sharing**: - Integration to push data to bureaus monthly might be needed (like sending new loan info). Usually done via SFTP batch files. - We can just generate file for manual upload by compliance team. If an API existed, we could push automatically. - Possibly not in this scope to automate.

Technical Integration Approach: - We'll create separate integration service modules in our codebase: - e.g., `nupayService`, `allpsService`, `payfastService`, `ozowService`, `creditBureauService`. - Each will handle the specific API calls and parse responses. - Use Cloud Functions for server-side calls to keep keys safe (not expose in front-end). - The system is modular, so if the micro-lender later switches provider (say from NuPay to a new one), we can implement that easily by adding a new service and updating config. - Logging: - We should log all integration calls and responses (except sensitive details) for troubleshooting. E.g., log "Debit instruction sent to NuPay for loan X, response success" or "Credit check API error: code 500". - Also possibly queue retries for failed calls (like if bureau API times out, try again after a minute).

UX considerations: - If an integration fails or is slow, user should get a meaningful message, not just spin forever. E.g., if credit check fails, show "Unable to fetch credit info, please retry or contact support" and allow saving app maybe in pending state. - For payments, if an external payment (like PayFast) is pending, show pending status to user until callback confirms.

Compliance with External Systems: - Ensure that integration uses allowed data (for example, only send client ID to bureau if we have consent and permissible purpose, which we do as part of credit application). - For payments, ensure we comply with NACHA/PASA rules via our providers (they largely handle that, e.g., DebiCheck compliance is handled by provider). - Keep integration systems' credentials secure and update them regularly if required by providers.

By integrating all these third-party services, our LMS significantly extends its capabilities without reinventing the wheel: leveraging specialized services for payments and credit data. This provides a seamless experience (e.g., automated payments and quick credit decisions) that outpaces competitors that may not have such breadth of integration.

B12. Mobile App Features

The LMS includes a mobile application (or mobile-responsive frontend) to support both borrowers and possibly staff in the field, ensuring that the system is accessible on smartphones and tablets. This section outlines the mobile app's functionality, which largely mirrors many web features but optimized for smaller screens and on-the-go usage. It will also highlight any mobile-specific features (like camera usage for document upload, push notifications, offline capabilities).

Given the user's requirement, the mobile app should serve at least the **borrowers (clients)**, and potentially could have a mode or separate app for **staff** (like loan officers doing field visits or collections agents in the field). Mobiloan emphasizes one solution on all devices [4], so we'll design a **single app** with role-based views.

Technology Approach: - We'll likely implement the mobile app using a cross-platform framework. As noted, using Ionic/Angular allows one codebase for web and mobile (just different build outputs). Alternatively, a separate React Native app but that means duplicate logic. We prefer a unified approach where possible. - We will ensure the UI is responsive (Material design is mobile-friendly by default) and use Cordova/Capacitor plugins for native features (camera, file storage for offline, fingerprint auth, etc.).

Mobile App for Borrowers (Client Self-Service):

Key features for clients via the mobile app: - **Account Creation & Login:** A client can register an account on the app (if the lender allows self sign-up). They would need to verify their phone/email (possibly OTP). Alternatively, an account might be pre-created when they first apply through an agent, and they receive login credentials (or link to set password). - Use Firebase Auth for this as well (could use phone number auth with SMS OTP for simplicity, which is common in apps). - After authentication, use JWT/cookies similarly to identify them for database access, with rules restricting them to only their own data.

- **Dashboard/Home:** After login, client sees an overview:
- If they have a current loan: show loan status, next payment amount and due date, outstanding balance.
- If multiple loans (rare in micro-lending simultaneously, unless one is settled maybe), list them or show active one and an option to view closed ones.
- Options like "Make a Payment", "View Statements", "Apply for New Loan".
- **Loan Application via Mobile:**
- The client can apply directly from the app:
    - They fill an application form similar to what an officer would on web: amount, term, reason.
    - They provide or confirm personal info (if new, they'd fill everything, if existing, maybe it's already there and they just confirm/update).
    - Upload required documents using phone camera:
    - Take photo of ID, proof of address, etc., within the app. We'll integrate device camera via Cordova plugin for that. Possibly auto-crop/resize for clarity.
    - The app can use OCR or barcode scanning (for SA ID, scanning the barcode on back of ID card to get ID number quickly).
    - E-sign: They could sign on screen (finger sign on touchscreen) for the application form. If needed, we capture that as an image.
    - Submit application. The app will call backend to create client (if new) and loan record.
    - Provide feedback: "Application submitted, you will be notified once a decision is made."
- **Real-time updates:** Using Firestore real-time, if an admin approves the loan, the client app could get a push notification or in-app update "Approved!" without needing to constantly refresh.

• **Loan Management:**

• If client has an active loan, they can:

  ◦ See details: original amount, interest rate, schedule of payments (perhaps a list of installments with paid vs due).
  ◦ View/download their loan agreement (PDF).
  ◦ Check current balance (maybe a dynamic calculation if interest accrues daily).
  ◦ Option to request an early settlement quote (the app can calculate current payoff amount and display it).
  ◦ Possibly request a top-up or new loan if eligible (some lenders allow top-up after certain repayments; the app could show "You are eligible for an additional R500" etc. Maybe ties to AI suggestions).

• **Payments:**

• The app provides ways to pay installments:

  ◦ Show "Pay Now" button linking to Ozow or PayFast integration. The user can choose to pay via their bank or card right in the app.
  ◦ If a debit order is scheduled, they might not need to manually pay, but in case they want to pay early or different method, they have this option.
  ◦ If they make a manual payment (like EFT outside app), they could upload proof of payment, but better to encourage using integrated methods for instant confirmation.
  ◦ They can also see payment history (which installments paid on which date, maybe receipts).

• **Notifications (Push):**

• The app will receive push notifications for:
  ◦ Payment reminders (e.g., a day before due).
  ◦ When a payment is successful or fails (e.g., "Your payment was received" or "Your debit order bounced").
  ◦ Loan status changes (application approved/declined).
  ◦ General announcements (if allowed, like system messages or marketing if opted in).
• We'll implement this using Firebase Cloud Messaging (FCM). The client app registers for the user's device token, and our backend Cloud Functions send messages triggered at events.
• Example: a scheduled function on due date could send an FCM push "Payment due today - click to pay now".

• These push notifications complement SMS/email but are free and direct if the app is installed.

• **Profile & Settings:**

• User can view and edit their contact details (address, email perhaps). If they update something, it writes to Firestore (which might require re-verification, e.g., if they change email, send new confirm).
• Change password or enable biometric login (fingerprint/face unlock). We can use device biometric auth via plugin to avoid typing password each time.

- Notification preferences: maybe toggle which notifications they want (except mandatory ones like overdue).
- Language preference: Could allow them to choose app language (English/Afrikaans/etc), leveraging our i18n support. This would also govern which language templates they receive in communication.
- **Support/Contact:**
- Provide a contact page with branch info or a support chat. Perhaps integrate a simple chat (maybe using WhatsApp link to message the lender's number or an in-app messaging using something like Intercom if integrated).
- Or at least list a phone number and email for help.
- **Multilingual Support (for client app):**
- We'll have the app's UI ready to display in multiple languages. Initially, content in English, but having the groundwork to add translations for labels and messages.
- We might default to English but allow switching. Possibly detect device locale to auto-set (if we have translations available for that locale).
- **Security (Client App):**
- They can enable fingerprint login for convenience after first sign-in (store token in secure storage).
- The app will have user sessions - we might keep them logged in but require re-auth for critical actions or show a PIN.
- We ensure the app uses secure communication (HTTPS).
- Data caching: minimal personal data caching on device unless needed offline (maybe basic profile info can be cached). For extra privacy, maybe provide a quick "logout" or auto-logout if app unused for a long period.
- **Offline Considerations:**
- Possibly allow filling application form offline and submit when back online (Cache data then sync). Using Firestore, offline mode can be enabled (it will queue writes), so that is helpful if connectivity is spotty in rural areas.
- For reading data, Firestore also caches reads offline by default if enabled.
- So a client could open app without connection and see last known loan info, but actions like payments would require online.

Mobile App for Field Staff (if applicable): We might also consider features if a loan officer or collector uses the mobile app: - A loan officer could use a tablet/mobile to register clients and capture applications on the go (which is basically using the same app as client maybe but in an admin mode, or a separate login that gives them more screens). - They would log in with their staff account, the app would detect their role and present a different interface (like an agent dashboard: assigned leads or ability to search clients). - They could use the camera to scan documents out in the field and upload directly, same as client would. - They

could have offline capabilities: e.g., if they are in a remote area without signal, capture an application and save locally, then sync when they get connectivity. This is advanced (we could store in local storage and then push to Firestore later). - Collections agents could use a mobile app to view their assigned cases, mark calls, and even collect payments via card using a card-swipe device or by sending an Ozow/PayFast link on the spot. Possibly integrate a Bluetooth card reader if needed (for instance, some collectors carry a mobile POS - integrating that is complex, but we could note that the app can integrate with such devices if needed, since there are APIs for certain readers). - Given not explicitly requested, it's a plus if our system supports staff mobility, which it effectively does if web is responsive or with minimal changes.

User Experience & UI: - We will design with simple, clean UI, large buttons for common tasks (like "Apply for Loan", "Make Payment"). - For readability, especially since micro-lending often deals with users of varying tech literacy, make the app intuitive: use step-by-step processes for application (maybe a progress indicator at top through steps: Personal Info -> Documents -> Terms -> Submit). - Use local language where possible or at least plain terms (not too legalistic in the app screens). - Possibly incorporate some gamification or positive reinforcement (like when they pay on time consistently, show a congratulatory message, or a progress bar for building their credit trust within the platform). - Also ensure UI is brand-customizable if needed (maybe white-label ability if the micro-lender has branding, but probably not needed here).

Integration with Device Features: - **GPS location capture:** Mobiloan mention GPS location usage [14]. We could capture device location when they apply (with permission) to help detect fraud (e.g., client claims to live somewhere but location is far off, or just for internal record). Also, if agent in field, record where they initiated app (for risk or verifying field agent visits). - **Camera & Gallery:** For doc upload, covered. Possibly also allow scanning barcode on ID for quicker input. - **Contacts/SMS reading (not likely needed)**: Some unscrupulous loan apps read user's contacts or SMS to judge behavior (and sadly to harass defaulters by messaging contacts). We will NOT do such privacy-invasive stuff, as it's problematic ethically and under POPIA likely not allowed without extreme consent. We focus on legit means.

Testing & Deployment: - App will be published likely on Google Play (and possibly Apple App Store if needed). Many microloan clients use Android primarily, but we should not exclude iOS for completeness. - We ensure the app runs on low-end Android devices commonly used in target market (lightweight, not too heavy on memory). - Possibly make it backward compatible to Android 5+ (still see those around) or at least 7+. - Provide updates for new features easily through app releases.

Security on Mobile (client side): - If device is rooted or compromised, can't do much beyond advising. But ensure our app doesn't store sensitive stuff in plain text. Use secure storage for any tokens. - Perhaps provide a quick app lock or logout in case user fears phone theft, though if phone stolen, attacker could open app if it's still logged in. We could add an app PIN as extra if needed or rely on device security. - The user can always revoke session by contacting support or something if phone lost (we could implement remote logout by removing auth token). - We should disable screenshots on sensitive screens to prevent data leak (optional, but some apps do that).

Compliance (Mobile App for Clients): - Terms and conditions of using app should be present (maybe at sign-up, an agreement to certain usage terms including consent to certain communications). - Ensure any info presented to client like their loan terms matches the official contract and is accurate. - Provide access to contract and maybe a link to NCR's pamphlet on rights (some lenders do give a "Know your rights under NCA" doc). - If multi-language, ensure translations are correct legally. - For e-sign, ensure process is

recorded (like we log that at this timestamp user X accepted via app, maybe send a confirmation email too). - Data usage: inform user about data usage if needed (though if they installed the app, it's expected). - POPIA: as a user themselves, they have rights, but since it's their own data they're viewing, not an issue. But we protect their data from others by login. Also, we should confirm their identity at registration (maybe OTP to phone ensures it's them).

Conclusion of Mobile: The mobile app dramatically improves accessibility – borrowers can apply and manage loans without visiting branch, and staff can operate on the move, aligning with modern fintech trends. This adds to usability and sets our platform ahead of older systems that might be desktop-only.

B13. Admin Tools & Configuration Modules

The Admin Tools section covers all the configuration and maintenance features that administrators use to tailor and manage the system. These include setting up loan products, interest rates, branches, user roles (if configurable), and other master data (like dropdown options). It also covers maintenance tasks like system monitoring, backups, and any internal utilities.

We gleaned from Mobiloan's user guide and anticipated needs a set of configuration modules: - Company/Branch setup - Product setup (loan products) - Fee and commission setup - Document type setup - Role/permission setup (if dynamic) - Integration settings - Misc. master data (loan purposes, categories, etc.) - Admin utilities (like data export, logs viewing, system security settings)

Let's detail each:

A13.1 Company Information Setup: - A page to configure global info about the lending company: - Company Name, Address, Contact details. - Registration numbers (NCR registration no., perhaps VAT no., CIPRO reg no.). - This info populates documents (contracts, letters) and perhaps branding (like email templates might use company name). - Possibly upload a company logo to display on client-facing docs or portal. - Also set business rules such as: * Business working hours (for controlling communications/notifications times). * Default branch (if one branch scenario). * Currency and locale settings (if expansion beyond ZA, but likely ZAR fixed). - Maybe toggles like "enable multilingual support" or "enable experimental features" if any (not necessary though).

A13.2 Branch Management: - If multiple branches: - Create/Edit Branch records: Branch name, code, address, contact phone. - Assign a branch manager (link to a user). - Possibly set branch-specific settings: working hours, region, or targets (some branches might have loan targets which can be tracked in reports). - Branch could also define numbering series for contracts if needed (some prefer contract IDs per branch). - Used for filtering data by branch in UI and limiting user access.

A13.3 Loan Product Setup: - Arguably one of the most important configs: - Admin can define each loan product type the system offers. For each product: * Name (e.g., "30-Day Payday Loan", "6-Month Personal Loan", "Business Loan 12m"). * Description (for internal info or to show clients maybe). * Product Category/Code (maybe align with regulatory categories: e.g., "Short Term Credit (<=6m)", "Unsecured Personal", etc.). * Minimum and Maximum Principal Amount. * Allowed Term range (Min/Max term in months or days). * Interest rate: - Could be fixed rate or range. Possibly allow either fixed per product (e.g., this product always at 5%/month) or a base plus variant (if risk-based pricing allowed). - Option for interest calculation method: e.g., "flat", "declining balance, amortized", "daily interest" etc. Typically personal loans are amortized

(reducing balance interest). Payday loan is flat for one period. Our system should know how to compute schedule accordingly. - Possibly interest frequency: daily vs monthly compounding. But microloans usually charge either simple interest or straight for term. We'll allow selection: "Simple (no compounding)", "Compound monthly". * Fees: - Initiation Fee: either a fixed amount or a formula. Possibly we just let admin input the max allowed and our system ensures it stays within law. + Maybe allow input like "R165 + 10% of amount above R1000 capped at R1050" or simpler have them fill certain fields and we enforce the formula behind scenes. Possibly simpler: ask for maximum allowed, and whether they want to actually charge the maximum or lower. Many will just do max allowed by law for revenue. - Service Fee (Monthly): a fixed monthly fee or maybe 0 for short term if they choose not to. Default likely R60 (plus VAT). - Early Settlement Fee or penalty (some allow a small fee if settle early, though NCA allows charging an early termination fee of up to 3 months interest on long term credit if settled early, but not beyond that. Could specify if they want to charge that or waive). - Late Fee: if allowed (maybe R50 once-off as per NCA reg, we could allow config). * Credit Life Insurance: - If applicable, link an insurance product. Possibly fields: "Insurance required if term > X or amount > Y", and a premium either as percentage of outstanding or per month fixed. - Or maybe just a toggle "include insurance" and premium formula. The system then will add that to fees and also handle payout of insurance if needed. * Grace Period / First payment rules: - E.g., "First installment in 30 days" vs allow customizing. Maybe not needed, but if a product allows a grace period (like first payment only after 60 days), admin could set that. * Max number of concurrent loans of this type allowed (maybe for controlling re-borrowing, but usually it's generic per customer, not product-specific). * Any collateral requirement indicator (if this product requires collateral, but microloans typically unsecured, skip). * Active flag: can deactivate a product (so it won't show up for new applications, though existing loans remain unaffected). * Order/priority: just to list them nicely in dropdowns. - This interface should be user-friendly because product setup errors could cause compliance issues. Possibly include a "Preview" or "Validate" button that calculates a sample scenario to show interest and fees to confirm correctness. - Only Admin role can access this section.

A13.4 Loan Purpose & Category Setup: - Many lenders have predefined reasons (Loan Purpose) like: Education, Medical, Small Business, Household, etc. - We can let admin edit that list (to add new categories or disable some). - It populates a dropdown in application forms. Useful for internal stats or NCR report (they often ask breakdown by purpose). - Also, categories for client or loan classification: - e.g., client categories like "New", "Repeat", "VIP" could be defined if they want to tag. - Or loan categories like "Staff Loan" (if they lend to their staff differently). - Possibly risk categories: not needed to set manually, but they could define some labels for risk bands if they want (like "Gold, Silver, Bronze" that correspond to score ranges). - Commission Setup: - If they have agents that get commission per loan, a config might be needed: * Commission structure could be flat per loan or % of loan amount or interest. * Possibly tiered by product or volume. * We saw "Commission Setup" in mobiloan menus [24] . * If applicable, the system could calculate an agent commission for each loan disbursed, which could then be paid out (some micro-lenders use external agents). * Admin would configure like: For Product A, agent commission = 5% of principal; or maybe a table if they have thresholds. * The system then can generate a Commission Report summing per agent. * If only internal staff, maybe they don't do commission or it's salary-based, but we allow the feature. - Insurance Setup: - If the micro-lender offers credit life insurance through a partner, an admin tool to configure insurance: * Add Insurance provider name, policy number series, premium rates (like x% of loan per month or R y per R1000). * Possibly whether it's mandatory for certain loans (like by term >6 months). * Possibly integration info if they'd want to send data to insurer (some integrate insurance claims). * At least we can use config to automatically calculate premium and add it to loan cost and maybe separate track that portion for remittance to insurer.

A13.5 User Role & Permission Management: - If roles are not fixed, admin might manage roles: - They could create custom roles or adjust permissions of existing roles if needed. - For example, maybe add a sub-role "Assistant Loan Officer" who can create applications but not approve even small ones. Or "SuperAdmin" separate from Admin. - We can present a matrix of permissions: down one side list all actionable permissions (like "Create Client", "Approve Loan up to X", "Approve Loan unlimited", "View Financial Reports", "Manage Users", etc.), and columns for each role with checkboxes. - Admin can tick/untick to grant or revoke a permission from a role. - Could also handle field-level or branch-level restrictions (like a branch manager sees only branch data, which we do via branchId linking, but if needed they could allow a role to see all branches). - Because we enumerated roles, dynamic management is an extra. Many smaller systems just hardcode roles, but to surpass global systems, flexibility is good. We can implement but caution: messing with it could break something or cause compliance issues (like if someone removes a required permission inadvertently). So we might provide a default set and allow advanced config behind a warning or even hidden unless needed. - On user creation, Admin chooses a role from those configured.

A13.6 Transaction (Accounting) Setup: - Mobiloan had "Transaction Setup" [25] which might refer to how financial transactions are categorized or integrated with accounting: - Perhaps define types of transactions (principal, interest, fee, payment, reversal, etc.) and map them to accounting ledger codes. - Possibly link to an accounting system like Sage/Pastel or QuickBooks: They might allow exporting a journal of transactions monthly. - If we were to implement, admin could enter ledger codes for each category (like interest income code, fee income code, loan receivable asset code), then our system can produce an export of accounting entries. - Without going full accounting, at least being able to categorize transactions is helpful. Or we might not implement unless specifically needed, but mention readiness to integrate with accounting (e.g., generate a CSV of transactions that can be imported).

A13.7 Integrations Settings: - We integrated a lot of third-parties; an admin interface is needed for managing API keys, credentials, and preferences: - For example: * NuPay: enter username, password, or certificate file upload. * ALLPS: API URL, keys. * TransUnion: endpoint URL, credentials. * Experian: credentials. * Twilio (or SMS provider): API key, sender IDs if any. * Email SMTP settings (if using SMTP rather than API). * WhatsApp Business API: maybe Facebook Business credentials or Twilio sandbox info. * PayFast: Merchant ID, Key, passphrase. * Ozow: Site code, private key. * If some providers have test vs live mode, allow switching (like PayFast has sandbox vs production). - Possibly a test button for each integration to ensure credentials working (like "Test SMS send" or "Test bureau connection" that pings and returns ok/fail). - This centralizes integration config rather than burying in code. - Only very few super-admin should access this, as keys are sensitive (we might mask them after entry). - We might store them in Firestore in a secure collection, or in environment config (for keys maybe better not to store in normal DB due to risk of exposure, but if UI can set them, they'd end up in DB unless we have a more secure storage or require server dev to input them).

A13.8 System Monitoring & Logs: - Provide an interface to view system logs or statuses: - E.g., an "Integration Logs" screen listing recent integration calls (especially failed ones, with error messages). - A "SMS/Email Log" showing count sent per day and any failed sends. - Possibly a "Scheduled tasks status" to ensure jobs like daily debits are running (if one fails, show alert). - If using Firebase, maybe integrate with their monitoring or just rely on Cloud Function logs externally, but an admin might not go to Firebase console, so some basic monitoring in-app could help. - Also environment info: version of the system, last deployment date, etc. for transparency. - If multiple branches, maybe a quick view of how many loans created today, etc., (though that’s more reports).

A13.9 Data Management Tools: - Bulk import/export: - If migrating from old system, admin might need to import clients or loans. We could provide an import tool (CSV format defined). - For example, "Import Clients" (CSV columns: name, ID, etc.) -> will create client records. - "Import Loans" might be trickier (we'd need all loan details). - Or at least an initial import of balances if mid-cycle transitioning. - Similarly, allow exporting data to CSV for backup or analysis (maybe beyond reports, like full data dump of certain collections if needed). - Data correction: - If some data need to be adjusted due to an issue (like interest calculation error), an admin might use a special tool to correct balances or adjust interest rate for a loan. Possibly not a separate interface but maybe allow editing those fields in the loan record with appropriate warnings (only if absolutely needed). - Or a "Loan Modification" tool if renegotiating interest (rare in microloans except restructure).

A13.10 Backup & Archive: - Automated backup probably handled by cloud, but maybe admin might want to download a backup of all data at intervals (for peace of mind or regulatory requirement to store data locally). - Could integrate with Google Cloud Storage to dump Firestore data daily. Or just have a button that triggers a Cloud Function to export to JSON and let admin download. - Also possibly an "Archive closed loans after X years" function if they want to remove them from active DB to archive storage (to keep DB lean if thousands of records after many years). We can mention archival, but Firestore can probably handle quite a lot, so not urgent unless performance suffers.

A13.11 User Support Tools: - If a client or user has trouble, admin might need to impersonate or reset: - "Reset Client Password" or send reset link for clients from admin panel (if client calls saying can't login). - Impersonate login: not common due to security, but maybe a "View as client" for support to help them step through app issues without needing their password. - Manage client consent or legal flags (if a client withdraws marketing consent via offline, admin can update in system). - Possibly a complaint logging if needed: sometimes required to record customer complaints. Could have a simple log or note addition for compliance, but might not be considered now.

A13.12 Audit & Compliance Tools: - We have many compliance things but admin tools might include: - Running an internal audit report (like see if any loans were approved without credit check or without docs, as a check on policy adherence). - The system could have a "Compliance Checklist" view summarizing for each loan or each recent loan whether all steps done (credit check Y/N, affordability done Y/N, contract signed Y/N, etc.). That helps compliance officer to spot any issues quickly. Possibly an advanced one, but good for surpassing others. - A central place to manage regulatory requests: e.g., if someone requests data deletion, admin can mark it and system will anonymize after retention period automatically.

User Permissions for Admin Tools: - Likely only "Admin" role has access to most of these config pages. - Maybe branch managers can access some limited config like updating branch info or viewing some branch-specific settings but not global ones. - We ensure these pages are hidden from unauthorized roles, and actions are protected by backend checks too.

Interactions: - Admin configurations ripple through the system: - E.g., if interest rate changed in product config, all new applications use that. Existing loans are unaffected unless we explicitly allow applying changes (rarely do retroactive changes, except maybe floating rates if concept existed, but we assume fixed per loan). - Changes in integration settings immediately affect how system communicates externally. - The user management ties here a bit if roles are dynamic; creating new roles can affect security rules (we would have to design rules to allow unknown roles with certain permissions, which is doable if we make permission lists in DB or as claims). - Commission setup influences how loan origination might calculate an

"agent commission" field per loan (if needed). - Document type setup: could list required docs per product or scenario. For instance, we could allow admin to specify: for product "Business Loan", require "Business registration doc" in addition. Then the system would enforce upload of that before approval. This could be a nice flexible feature: * in Document Setup, define doc categories and optionally link them to either all loans or certain product or client type. Then system uses that to present needed doc placeholders in UI and to do completeness checks.

By offering comprehensive Admin Tools, we empower the lender to adapt the system to their business and regulatory environment easily, without needing developer intervention for every small change. This configurability and oversight capability helps our LMS stand out relative to more rigid systems.

B14. Security Modules

Security is critical in an LMS handling sensitive financial and personal data. While we've touched on security in each section (like roles, encryption, etc.), here we consolidate and detail security-specific features and modules, ensuring the system is robust against threats and misuse. This includes authentication security, authorization controls, data protection measures, auditing of access, and any special fraud prevention beyond normal risk (like application fraud detection, etc.).

Key Security Features:

- **Authentication Security:**
- We use Firebase Auth (which is secure and battle-tested) to manage user sign-ins. But beyond that, we implement:
    - **Multi-Factor Authentication (MFA):** Particularly for admins or sensitive roles. We can use Firebase's phone OTP 2FA or an authenticator app if supported. For our design, the simplest is SMS 2FA at login for admins.
    - **Session Management:** We limit sessions. For example, if using web, after 15 minutes of inactivity maybe auto-logout (this might annoy some, but optional or maybe just for admin). On mobile, since usually personal device, perhaps keep logged in but require fingerprint or PIN to open app each time to ensure device owner.
    - **Password Policies:** Enforce strong password on user creation (min length, mix of characters). Also encourage periodic changes or at least provide ability to change. Possibly disallow reusing last N passwords (not sure if Firebase can enforce that, likely not easily without storing hashes - might skip that due to complexity).
    - **Account Lockout:** If some internal staff tries wrong password 5 times, we could temporarily lock or require reset (though Firebase might handle some of that, but not sure if default). We can implement a simple tracking and then disable user or require admin unlock. This prevents brute force.
    - **Login Notifications:** Optionally, email user when a new device logs in to their account. Useful especially for admin accounts - if someone logs in from unknown IP, alert them. Could integrate an IP geolocation to detect new location vs typical.
    - **Device/Session Logs:** Let users (especially admins) see where they're logged in (list of active sessions with IP and device) and allow remote sign-out of others. This is advanced but

Firebase does allow multiple sessions with their tokens. We might not implement fully but mention as a possibility to surpass (like how Google shows "logged in devices").

- **Authorization & Access Control:**

- We robustly enforce access permissions throughout:
    - On front-end, route guards and UI hide/show.
    - On back-end, Firestore Security Rules and Cloud Function checks validate the user's role/permissions on each data access or operation.
    - We implement principle of least privilege in data rules: e.g., a loan officer's read query for loans is filtered by branch in rules, so they physically cannot pull other branches' data via Firestore even if they manipulate the client.
    - Use custom claims in Firebase Auth tokens to carry role and branch info to rules (so rules can do like `request.auth.token.branchId == resource.data.branchId` etc.).

- **Segregation of Duties:** Possibly ensure that no single user can circumvent controls:

    - E.g., the person who creates a loan is not the one approving if possible (we can't strictly enforce, but many lenders have policy that only a manager approves, which we do).
    - Perhaps require two managers to approve very large loans (system could enforce dual approval if above a threshold: one approves, then another with equal or higher role must confirm). This could be a config "dualApprovalLimit". If implemented, the system holds loan in "Pending Secondary Approval" state until second person approves.
    - Another: a user cannot approve their own loan if they applied as a customer or such (not typical, but maybe block staff from giving loan to themselves by checking ID match).

- **Data Encryption:**

- As earlier, all traffic is TLS encrypted by default (Firebase endpoints).
- Data at rest: Firestore and Storage automatically encrypted on Google servers. For extra, we could encrypt certain fields ourselves:
    - e.g., maybe encrypt ID numbers in Firestore with a key such that even if Firestore is compromised, without our key it's gibberish (maybe overkill since Firestore already secure, but some might want their own managed encryption).
    - More realistically, we might encrypt very sensitive fields in certain contexts or for compliance maybe keys like an OTP secret if we stored or such.
    - If we do, we'd use a secure key store (Google KMS) to handle encryption keys.
- We ensure not to store any plaintext sensitive PII in logs or error messages inadvertently.

- If in future, to integrate blockchain or decentralized (the IRJMETS mention in search [18†L21-L29] was "Decentralized LMS", but likely out of scope now, just focus on conventional encryption).

- **Audit Trails & Logging:**

- We have an audit log for all critical actions: login, logout, data changes (create/update/delete on important collections: e.g., if someone edits a loan or client, log old vs new values maybe).
- Might integrate with Firebase Cloud Firestore triggers: on any write to certain collections, write an entry in `auditLogs`.

- Or simpler, for key operations done via Cloud Functions (like approveLoan, editUser), we add a log entry via code.
- The log records who did what, when, and possibly from what IP (we can capture user's IP in cloud function context).
- These logs can be reviewed by admins in an interface or exported for external audit.
- They help in forensic analysis if something goes wrong (like if data was changed, we see by whom).

- Possibly highlight unusual events (like someone trying to access unauthorized data though ideally our rules prevent it, or many failed logins).

- **Fraud Detection (beyond credit risk):**

- We mentioned some:
    - App and documentation fraud: if an ID was used before by another name in our system (we can have the system detect duplicate ID submissions under different name → flag possible identity theft).
    - If many applications come from same device or IP with different IDs, flag that (especially relevant if we have an online portal open to public).
    - We could use device fingerprinting (like via browser fingerprint or mobile device ID) to correlate.
    - Staff fraud prevention: Ensure one user isn't tampering with data unnoticed. Audit logs cover part. Also restrict database direct access (only through app and secure rules) so staff can't manipulate via some hacky route.

- Possibly incorporate a rule: to prevent internal collusion, maybe certain data fields are read-only after entry unless admin changes. E.g., once loan terms are set and contract generated, a loan officer can't secretly edit amount or interest later. Our design anyway restricts who can change those (only admin might via special tool).

- **Security Monitoring:**

- We might integrate with Google's security center or external SIEM:
    - For example, export logs to a SIEM that monitors for anomalies (like multiple failed logins, or data access outside business hours by an employee).
    - Or simpler, set triggers:
    - If admin login from a new country, alert.
    - If a large number of records exported, alert.
    - If unusual volume of tasks by a user (like someone approving 100 loans in an hour, maybe investigate).

- We allow admin to set some threshold alerts in config if needed (not mandatory but a plus).

- **Penetration Testing & Hardening:**

- Before going live, system should undergo pentesting. We in development consider common vulnerabilities:
    - Prevent XSS by properly escaping any user input in UI.

    - Use Firestore rules to avoid injection (NoSQL injection not typical but ensure queries are controlled).
    - Parameterize any queries if used in Cloud Functions with SQL (we likely not using SQL).
    - Rate limiting: Because it's behind Firebase, we get some built-in limits. But if needed, e.g., throttle certain expensive operations or login attempts. Firebase Auth might throttle on its own. For functions, could implement basic if an IP hitting too often, but as internal mostly usage, low risk.
    - DDoS: Rely on Google infra to absorb. Possibly use Cloudflare if needed but probably not needed.
    - Ensure backups and contingency: If something happens, we can restore data (we addressed backup).
    - On code level, no hard-coded secrets in client code, etc.

• Use security reviews for third-party libraries (ensuring none are compromised).

• **Device Security for Mobile:**

• Provided some points in mobile about fingerprint etc. Could also ensure the mobile app doesn't store sensitive data unencrypted.

• If sending documents to phone (like contract PDF), maybe consider marking them or having them expire if possible, but once user downloads, can't control that.

• **Security Modules UI:**

• Possibly an admin "Security Console":

    - See list of active user sessions (who is currently logged in).
    - Force logouts, disable accounts quickly if needed (account lock).
    - Manage roles/permissions as earlier.
    - View recent audit logs, filter by event type or user.
    - Possibly manage some firewall rules if needed (like restricting admin login to certain IP ranges – advanced. Some companies want that: e.g., admin can only login from office network VPN. We could incorporate IP allowlisting for admin user roles: could store allowed IP ranges in config and our login function checks it).
    - If integrated with an OTP or authenticator app, maybe a screen to manage/regenerate QR for admin 2FA etc.

• **Compliance in Security:**

• Many points overlap with POPIA (data protection), which we covered.
• For PCI (if handling card, but we actually offload to PayFast, so minimal PCI scope).
• IFRS etc. not relevant in IT security context.
• Being secure also helps with external audits - the system should allow an external auditor read-only access, maybe via a special Auditor role that can see everything but not change (we did mention read-only role).
• Ensuring logs and data is kept for some time (if logs are needed for audit, keep them at least a year or more).

- The system design and code should follow best practices, which we are detailing as part of manual - any developer reading knows where to apply security measures.

Summary of Security Integration with other Modules: - All modules incorporate role-based checks (User mgmt defines roles, all features check those). - Compliance and audit ties with security (almost same sometimes). - Admin Tools includes permission config as described. - Communication: ensure notifications don't leak sensitive info (security by design on comm). - Document handling: restrict who can see which docs. - Integration keys stored safely (less accessible to average user). - AI/ML if used should also be done securely (like any personal data used in ML models anonymized if possible, though we likely just use internal model). - In training Codex to generate code, we would stress including security aspects for each feature prompt to ensure code is secure.

By consolidating these security features, our LMS ensures robust protection against unauthorized access and data breaches, giving confidence to both the micro-lender and borrowers that their data and funds are safe. This level of thorough security is essential and expected in modern financial software, and we aim to not just meet but exceed those expectations.

B15. AI & Machine Learning

The AI & Machine Learning module explores the incorporation of intelligent features like recommendations and fraud detection to enhance decision-making and efficiency. This is a forward-looking component that leverages historical data and patterns to provide insights or automate certain decisions. By including AI/ML capabilities, our LMS becomes a "next-gen" system, potentially offering predictive analytics that competitors may not.

We will detail possible AI/ML use cases in our LMS:

B15.1 Credit Scoring & Risk Prediction: - Beyond simple rules, we can develop a machine learning model to predict the probability of default for new loan applicants. - Data for model: historical loan outcomes (paid on time vs default) along with features like applicant demographics, financials, credit bureau info, loan terms, etc. - Model type: Possibly a logistic regression or decision tree model initially. Could use Python to train offline, then implement coefficients or a simple form in code for real-time scoring. - *Process:* - We gather training data from our database (loans that have completed or at least 6 months performance). - Train a model that outputs a score (0-1 probability of default). - Then in our system, when a new application comes, we compute its features and run it through the model to get a predicted default probability or risk grade. - Use that to assist decision: * e.g., If probability of default > 50%, auto-reject or require strong motivation to approve. * If low, maybe auto-approve if all else fine (some micro-lenders do automated decision for top-tier applicants). - Also possibly use it to adjust interest: higher risk -> charge near the max interest to compensate, lower risk -> maybe could offer slightly discounted rate to attract good borrowers (if competitive market, not always done in microloans but in theory). - *Implementation:* - We could embed a pre-trained model. For example, a logistic regression could be implemented as a formula in code (weights and coefficients stored). - Or use TensorFlow Lite or tf.js in Cloud Function to load a model and predict. - Since our environment is Python/Node for Cloud Functions, we might have to handle accordingly. Python function (via Cloud Functions triggered or an API endpoint) could run a model. - Or simpler, do it offline: train model externally, then just create a scorecard (like a point system) for implementation. - Many credit models can be approximated by a scorecard (like assign points for each factor and sum). - We can present the result as part of risk evaluation: e.g., "AI-predicted default risk: 20% (Low)".

B15.2 Recommendations: - *Loan Upsell/Cross-sell:* - If a client has a good track record, the system could recommend offering them a larger loan or another product. - For example, after they pay 3 installments on time, system flags "Eligible for top-up of R500" or "Pre-approved for new loan after this one ends". - Could appear in the client mobile app ("You have earned a higher credit limit due to good repayment!") and in the staff interface so they know to mention it. - The logic can be rule-based (like good performance) or ML-based (predict which clients are likely to take and successfully repay an upsell). - Another recommendation might be if a client was declined, suggest when they can reapply or what conditions would change that (maybe not exactly ML, but a helpful feature). - *Collections Recommendations:* - ML could analyze which collection actions yield results for which type of customer (e.g., some respond better to WhatsApp vs calls). - Could then suggest to collector "This customer likely to respond to a WhatsApp message based on past similar profiles". - Or prioritize accounts in collections by likelihood of recovery (ML prediction of recovery). - Or predict which accounts might roll into default so to escalate earlier. - *Financial Health Insights:* - For the client side, maybe the app could give them insight "Customers like you typically borrow X, consider not overextending..." (though that might conflict with lending interest, maybe skip telling them not to borrow, but could do responsible education). - Possibly integrate with a credit score education where the AI explains factors affecting their credit (like "Your on-time payment of last loan improved your credit score by Y", which we could do if we see bureau data improvement).

B15.3 Fraud Detection (AI approach): - We discussed some rule-based in Security, but ML could help identify patterns: - e.g., a clustering algorithm on application data to find groups of apps with suspicious commonalities (like many different clients using same bank account or phone number, which likely is caught by simple queries too). - Or train a model on fraudulent vs legitimate cases (if we had that data) to detect subtle patterns. - For identity theft, maybe an AI that compares the ID document photo to an uploaded selfie to ensure it's the same person (face recognition). We could integrate an AI service for face match (there are APIs for that). This is advanced but effective: * The app or in-branch captures client's face photo, the system compares to ID card photo (if ID card has a photo in digital form, not easily unless scanning the newer card's chip or QR which is not accessible externally). * Or if passport, there's photo which can be OCR'd (maybe not easily, more like image processing). * Possibly skip deep face match due to complexity and potential biases. But mention that some systems do biometric checks and we could integrate those if needed (like fingerprint scanning device or so). - Another area: **AML suspicious transactions**: - If someone tries to pay off a loan with a huge amount from an unrelated third party, could be money laundering. Our system is not a bank so less likely to see that, but if we allowed overpayments or something, or prepayments bigger than loan amount, we should flag. - We can set a rule: if payment > X or multiple small loans rapidly taken and paid, maybe suspicious layering behavior. - Hard to implement properly without false positives, but could use ML anomaly detection on payment patterns.

B15.4 Machine Learning for Operations: - *Portfolio trends:* ML can forecast future credit losses or demand: - e.g., use time series forecasting for how many loans might default next month given current data. Could help in provisioning or capital planning. - Or predict cash flow – though we can do deterministic since schedule known, but ML can adjust prediction if historically some fraction always pays late etc. - *Chatbot/ Virtual Assistant:* - Perhaps integrate an AI chatbot for customer service: e.g., using something like GPT (though heavy for local deployment, maybe an API to a service). - Could be on WhatsApp or in app: answer common questions ("How do I apply?", "What is my balance?"). It would need training on a knowledge base. Possibly beyond our dev scope but an idea. - Or a simple Q&A bot on website. - *Document OCR & Classification (AI):* - Already covered in Document Handling that we might use AI to read and classify docs. - Could extend that: e.g., automatically detect if a document uploaded is actually an ID vs some other doc

(some AI vision can classify document types). - Or extract employer name from payslip and cross-check if matches what's in form, etc.

Integration & Implementation of AI: - We might leverage cloud AI services: - Google Cloud AutoML or BigQuery ML to train models on data stored in Firestore (we might export to BigQuery for analysis). - Or use Python notebooks for initial training and then embed results. - Possibly incorporate open-source models or frameworks for credit scoring (some libraries exist for credit risk). - The manual prompts (in Section C) can instruct Codex to implement simpler rule-based versions first or scaffolding for ML integration, given that actual ML training might be beyond Codex's capability (lack of actual data). - But we can ask Codex to include hooks like a function `calculateRiskScore()` that either uses a placeholder model or calls an external model. - We could also foresee using an external credit scoring service if provided (some Fintechs offer a combined scoring API).

Governance: - If using AI for decisions, ensure compliance with any regulations about automated decision: - Under NCA, declines should ideally still involve human oversight or at least provide reason. We are giving reasons anyway (like credit score too low, affordability). - POPIA has a clause about not being subject to purely automated decision that significantly affects them without human intervention, unless they've given consent. For credit, they likely know scoring is used but typically there's human final check (we have risk officer/manager, so it's not fully automated except possibly small loans but even then we can have someone quickly verify). - So if we auto-decline via model for low amounts, ensure we inform the client they were scored out and maybe allow them to request review, satisfying fairness. - Keep models updated and avoid biases (e.g., ensure model isn't inadvertently biased on protected attributes like race or gender; we likely don't even include those in features).

AI Impact on Roles: - It helps risk officers by giving them a tool, but doesn't replace them. Possibly we reduce the workload for very clear cases. - It helps collections by focusing efforts. - It improves customer experience by potentially quicker decisions and personalized offers. - We should educate users (the staff) on what AI outputs mean, so they trust and properly use them (not blindly follow if something seems off).

Example Summaries to Show in UI: - For risk scoring: "AI Risk Grade: B (Expected default rate ~5%)." - For recommended actions: on a client's profile: "Recommendation: Eligible to borrow additional R1,000." or for a delinquent client: "Recommendation: Offer a restructure plan" if model suggests they'd respond to that better than legal action. - These should come with explanation if possible (like "because of good payment history and increased income, we recommend upsell" or "client missed last 2 payments, high risk of default - escalate quickly").

By incorporating AI/ML, the LMS not only automates processes but can potentially improve decision quality and business outcomes (higher approval of good loans, lower bad debt, improved recovery). Many global platforms are starting to include such features, and we'll ensure our system is on the cutting edge by design.

With Section B completed, which enumerated all features, interactions, and details, we proceed to Section C, where we'll craft the Codex prompts that can be used to implement each module in isolation but in sequence.

Now that we have done Section B thoroughly, the last part is Section C: Ultra-detailed Codex prompts for each feature (C1–C15). These prompts instruct Codex how to implement everything we described, one by one.

We should ensure: - Each prompt (C1 to C15) covers the feature's context and requirements enough for Codex to code it. - Mention relevant technology (like "use Firebase Auth for X", "use Angular and Ionic components for Y", etc.) - Provide details of fields, logic, as enumerated, so Codex knows what to do.

We will include front-end and back-end in each prompt: - Likely focusing on the key code for that feature. Might not list every file, but instruct what to create or modify. - Ensure each prompt says "without breaking existing features, ensure to integrate properly."

We may not actually run code but the prompt should be ready to produce workable code if fed to Codex with context of previous ones.

We will also include i18n note and multi-language in each prompt (like "make text translatable").

Now let's structure each prompt concisely but thoroughly: C1: Implement user management (auth, roles, access control). C2: Implement client management (client CRUD, fields, UI etc). ... up to C15: Implement AI & ML features.

We'll keep them imperative and possibly enumerated as steps: That would likely be easiest for Codex to follow (some say Codeex liked clear numbered steps or bullet points with specifics).

Alright, let's prepare those in final answer now.## SECTION C: ULTRA-DETAILED CODEX PROMPTS

Now we provide step-by-step prompts for OpenAI Codex to develop each feature module. These prompts assume you feed them to Codex sequentially, carrying forward the code context built so far. Each prompt is crafted to ensure the new module is added **modularly** (isolated, not breaking previous features) and includes both frontend (UI) and backend (Firebase/Cloud Functions) aspects, with all necessary details (fields, logic, Firebase references, etc.). We also instruct Codex to implement text in a way that supports future multi-language translation (e.g., by keeping UI strings in a config for easy localization, default English).

C1. Prompt – Implement User Management (Authentication & Roles)
Implement the **User Management System** for the LMS with the following requirements:

1. **Authentication with Firebase:** Use **Firebase Authentication** to handle user sign-in. Enable email/password login for staff users. Set up a login page (web frontend) with fields: "Email" and "Password", and buttons "Login" and "Forgot Password". Validate inputs (proper email format, password not empty). On login form submit, authenticate via Firebase Auth (`signInWithEmailAndPassword`). If credentials are wrong, show an error message ("Invalid email or password."). Implement "Forgot Password" by linking to Firebase's password reset (e.g., via `sendPasswordResetEmail`) and inform the user ("Password reset link sent to your email").

2. **User Roles & Access Control:** After login, restrict functionality based on user's role. The roles are: **Admin**, **BranchManager**, **LoanOfficer**, **RiskOfficer**, **CollectionsOfficer**, **ReadOnly/Auditor** (as defined in Section B1). Store each user's role in their Firebase Auth **custom claims** or a Firestore `users` document. Implement a backend function (Cloud Function) `setUserRole(userId, role)` that an Admin can call to assign roles (this function will add a custom claim `role=...` for that user in Firebase Auth, and also update the `users` collection). On frontend, after login, fetch the current user's role (e.g., from a Firestore `users/{uid}` document or via `user.getIdTokenResult()`) and store it in the app state. Use this to control UI:

3. Show an **Admin Dashboard** link only if role is Admin.
4. Only Admins can see the "User Management" section (list of users, create user form, etc.).
5. Loan officers should see client and loan menus but not user admin.
6. Risk officers see risk evaluation pages, etc. Implement route guards: e.g., a route for "/admin/users" should only allow Admin role, otherwise redirect to no-access page or home.

7. Also set Firestore Security Rules: ensure that only users with `role=="Admin"` can read/write the `users` collection (except each user can read their own user profile). Define rules for other collections to restrict sensitive fields by role (for now, implement basic: e.g., in a loan document, allow create if auth.token.role is "LoanOfficer" or higher; allow updates to approval status only if role is "BranchManager" or "Admin"; we will refine in later prompts but put placeholders).

8. **User Registration (Admin only):** Create an **Admin Panel** page "Manage Users". Here, an Admin can add new staff users. Provide a form with fields: Name, Email, Password, Confirm Password, Role (dropdown of roles), Branch (dropdown, if branches are used; can be optional for now). On submit, use Firebase Admin SDK via Cloud Function to create the user account (with email & password), then call the `setUserRole` function to set their role and save other info in a `users/{uid}` document (fields: name, email, role, branch, createdAt, isActive true). Also implement toggling a user's active status: in the user list, Admin can "Deactivate" a user, which sets a field `isActive=false` in Firestore and perhaps disables login (you can implement by another custom claim or by using Firebase `updateUser` to disable). The front-end should reflect if a user is inactive (e.g., greyed out). Prevent login on inactive accounts (e.g., in a Cloud Function onAuth trigger, check `users/{uid}.isActive`; or simpler, after sign-in, if their `isActive` is false, force logout and show "Account disabled").

9. Also implement "Edit User": Admin can change a user's role or branch via a similar form. Update the Firestore doc and call `setUserRole` to update Auth claims if role changed.
10. Provide a "Reset Password" option for Admin on each user (maybe a button that triggers Firebase's password reset email to that user).

11. Provide basic form validation and success/error feedback (e.g., "User created successfully", or show errors like "Email already in use" from Firebase).

12. **Session Security & Multi-Factor:** Ensure the auth flow is secure. Use Firebase's built-in protections for brute force (it will throttle). Additionally, implement **Multi-Factor Authentication** for Admin users: e.g., if a user with role Admin logs in, require an extra step: send an OTP code to their email or phone (since we might not have phone on file for staff, use email). For simplicity, implement an email OTP: generate a 6-digit code in a Cloud Function and send via Firebase's email (or use SMS if phone

available via Twilio integration later). On front-end, show an OTP input if user is Admin after initial login, verify code via a Cloud Function (`verifyAdminOTP(uid, code)`). Only grant access if correct within 5 minutes. (If code wrong or expired, deny and possibly log event).

13. This step can be optional based on config; ensure to write code such that enabling/disabling MFA is easy (maybe a flag in a config collection like `security.mfaForAdmin=true` which Admin can toggle).

14. Also set up the front-end to use secure storage for session (use HttpOnly cookies or Firebase's built-in token management; since it's single-page app, store the token in memory or localStorage as needed, but guard against XSS by sanitizing inputs everywhere).

15. **Access Logging:** Implement an **Audit Log** for user activities related to auth & user management:

16. Each successful login: create a log entry in Firestore `auditLogs` with userId, action "LOGIN", timestamp, and maybe IP (you can get client IP from `X-Forwarded-For` header in Cloud Functions context and store it).
17. Likewise for logout, user creation, role change, deactivation: log those events.
18. We'll create an `AuditService` (Cloud Function) that is called by these operations to write log entries. Ensure sensitive data (like passwords) are never logged.

19. This sets up for compliance tracking. (We will later add a UI for admins to view logs, but just ensure logging happens now.)

20. **Front-end UI Structure:**

21. Build a navigation menu that adapts to role. For instance, a sidebar with sections: *"Clients"*, *"Loans"*, *"Collections"* for loan officers; plus *"Reports"* for managers; *"Admin"* for admin, etc. Use Angular route guard (if using Angular) or conditional rendering (if React) to only show allowed sections.
22. After login, redirect user to a Dashboard page appropriate to their role (e.g., Admin goes to an admin dashboard summary, LoanOfficer to a loan pipeline dashboard). For now, you can create a simple placeholder dashboard component that just says "Welcome [Name] – Role: [Role]" and maybe shows a few relevant quick stats we can fill in later.
23. In the Admin's "Manage Users" page: display the list of users in a table with columns: Name, Email, Role, Branch, Status (Active/Inactive), Last Login. We will populate Last Login if we store it; for now, as a stretch, update each user's `lastLogin` field in Firestore on successful login (via Cloud Function onAuth or as part of login flow).

24. Provide buttons per row: "Edit", "Deactivate/Activate", "Reset Password". Wire these to the functionalities above (Edit opens a form, Deactivate toggles isActive with confirmation, etc.).

25. **Backend Cloud Functions:**

26. Write a function `onUserCreated` (Firestore trigger) or just incorporate in the create-user function to also create a Firestore `users` doc. (Because using Admin SDK to create user returns uid; then set doc with that uid).

27. Write `setUserRole` function (callable or REST) that verifies the caller is an Admin (`context.auth.token.role == 'Admin'`) before setting another user's claims. It should use Firebase Admin SDK `auth().setCustomUserClaims(uid, { role: 'LoanOfficer' })` etc. Also update Firestore `users/uid/role`.
28. Write `disableUser` function for deactivation (again only Admin): uses Admin SDK `auth().updateUser(uid, {disabled: true})` and sets Firestore `isActive=false`.
29. On front-end, ensure to handle Firebase "user disabled" error on login to show appropriate message.
30. Use Firebase Functions config or environment to store any sensitive default (though mostly user management not heavy on secrets).

31. Ensure all these functions return success/failure messages for UI to display.

32. **Multilingual/Text:** For all UI text (labels like "Email", "Password", messages like "Invalid password", etc.), make them easy to translate later. For example, in Angular, use an i18n JSON or a dictionary object. Or simply put them in a constants file. Default is English. (We won't implement actual switching now, but structure for it.)

33. e.g., define `const MSG_INVALID_LOGIN = "Invalid email or password."` so we can later localize that.

34. **Isolation & Non-breaking:** This user management module should be added without breaking existing code. Place new components under appropriate module or folder (e.g., in Angular, create an `auth` module for login components, an `admin` module for user management pages). Do not hardcode values that conflict with other modules. Ensure styling is consistent with overall app (use a common CSS framework like Bootstrap/Material for form styling).

35. Ensure that until other features are built, if a non-admin logs in, they might see mostly empty sections – that's fine (we will implement those later). Just make sure navigation works and unauthorized routes are guarded. For now, you can stub out pages (e.g., a blank Clients page) so navigation doesn't break.
36. Test the flow: create an Admin user (maybe manually first via Firebase console or a setup script) so you can log in and then use the UI to create other users. Verify an Admin can login, add a LoanOfficer, log out, then LoanOfficer can login and sees limited menu. Ensure security rules prevent LoanOfficer from accessing admin data (test by trying to navigate to admin URL – should redirect or deny).
37. Also test the password reset: try clicking it and see if email received (if we configure Firebase email templates correctly).
38. Do not break future context: keep the code modular (e.g., separate services for Auth, separate component for login, separate for user list). We will expand on this base in subsequent prompts.

Use the above details to implement the user management system. Remember to thoroughly comment the code for clarity, and to handle error cases (e.g., creating a user with existing email returns an error, show "Email already exists."). Begin coding now.

---

1 23 Mobiloan Loan Management System
https://mobiloan.io/

2 Credit agreements in South Africa - Wikipedia
https://en.wikipedia.org/wiki/Credit_agreements_in_South_Africa

3 12 9 Key Features to Consider in Loan Management Software
https://lendfusion.com/blog/loan-management-software-features/

4 14 20 Mobiloan – Apps on Google Play
https://play.google.com/store/apps/details?id=xyz.appinstall.mobiloan&hl=en_ZA

5 10 18 Maxstore-paymentsystems | Mycomax MicroFinance Solutions
https://www.mycomax.com/maxstore-paymentsystems/

6 22 NuPay About & Contacts | ITWeb
https://www.itweb.co.za/office/nupay/about

7 Xpert Decision Systems | Mobiloan User Guide
https://docs.mobiloan.io/integration-partners/xpert-decision-systems

8 9 POPIA | Digital Guardian
https://www.digitalguardian.com/compliance/popia

11 13 21 Maximum Interest Rates Allowed in Terms of the National Credit Act
https://blanckenberglaw.co.za/maximum-interest-rates-allowed-in-terms-of-the-national-credit-act/

15 16 New Loan | Mobiloan User Guide
https://docs.mobiloan.io/main-menu/origination-menu/new-loan.origination_status

17 24 25 Introduction | Mobiloan User Guide
https://docs.mobiloan.io/getting-started/introduction

19 ALLPS Integration: Streamlining Payment Management - Amplifin
https://amplifin.co.za/amplifin-allps-i-the-power-of-integration/