Rent Reminder App – Project Blueprint

Author: Nabasa Amos

Platform: React Native (Mobile App)

Database: SQLite (Embedded)

Architecture: Offline-first, Embedded Node.js Logic

1️⃣ System Architecture & Boundaries

Overview:

- The app is offline-first.
- Embedded Node.js logic will handle all calculations, reminders, and data manipulation within the mobile app.
- SQLite is used as the local database to store tenants, payments, reminders, and settings.
- No external backend is required initially, but the system is future-proof for cloud sync.

System Components:

1. UI Layer (React Native)
    - Screens: Dashboard, Tenants List, Tenant Details, Add Payment, Reminders, Settings
2. Business Logic Layer (Embedded Node.js)
    - Core functions: calculate months covered, next due dates, tenant status, create reminders
3. Data Layer (SQLite)
    - Tables: Tenants, Payments, Reminders, Settings
4. Notification Manager
    - Handles local notifications for due rent, configurable by user settings

Embedded vs API Architecture:

- The app uses Embedded Logic rather than a separate Node.js server.
- Pros: simpler offline deployment, faster, private.
- Future Cloud Mode: can add sync via Node.js or Firebase with minimal changes to logic.

2️⃣ Data Models (SQLite Schema)

2.1 Tenants Table

Field	Type	Description
tenant_id	INTEGER PK AUTOINCREMENT	Unique tenant ID
name	TEXT	Full name
phone	TEXT	Contact number
room_number	TEXT	Room/unit identifier
start_date	TEXT (ISO)	Move-in date
monthly_rent	REAL	Rent per month
status	TEXT	Paid, Due Soon, Overdue
notes	TEXT	Optional remarks

2.2 Payments Table

Field	Type	Description
payment_id	INTEGER PK AUTOINCREMENT	Unique payment record
tenant_id	INTEGER FK → tenants	Tenant reference
amount_paid	REAL	Amount paid
months_paid_for	REAL	Calculated as amount_paid / monthly_rent
payment_date	TEXT (ISO)	Date of payment
next_due_date	TEXT (ISO)	Calculated based on months paid
payment_method	TEXT	Cash, Mobile Money, etc.
notes	TEXT	Optional

2.3 Reminders Table

Field	Type	Description
reminder_id	INTEGER PK AUTOINCREMENT	Unique reminder ID
tenant_id	INTEGER FK → tenants	Tenant reference
due_date	TEXT (ISO)	Rent due date
reminder_date	TEXT (ISO)	Scheduled reminder date
status	TEXT	Pending, Sent, Acknowledged
message	TEXT	Optional custom message

2.4 Settings Table

Field	Type	Description
setting_id	INTEGER PK AUTOINCREMENT	Unique setting record
reminder_days_before_due	INTEGER	Days before due date to notify
reminder_time	TEXT	Time of day for reminders (e.g., "09:00")
notification_enabled	BOOLEAN	Enable/disable reminders
data_backup_path	TEXT	Path for local export/backup
currency	TEXT	e.g., "UGX"
theme	TEXT	Light/Dark

3 Data Sync Strategy

Offline Mode:

- App works fully offline using SQLite
- Data is persistent and private
- Manual export/import of .json or .csv is possible

Future Cloud Sync / Hybrid Mode:

- Optional backend (Node.js/Express + MongoDB, Firebase, or Supabase)
- Each record includes:
  - last_updated timestamp

- sync_status (pending, synced, failed)

- Enables auto-sync, multi-user access, and remote backup

Backup / Export:

- JSON structure example:

```
{
  "tenants": [...],
  "payments": [...],
  "reminders": [...],
  "settings": {...}
}
```

Security:

- Optional encryption for sensitive fields (phone numbers)
- Offline app still respects privacy

4️⃣ Notification Logic

Reminder Manager:

- Scans tenants daily (or on app open) for upcoming due dates
- Creates reminders in reminders table

Notification Handler:

- Reads pending reminders
- Triggers local notifications (popup, vibration, sound)
- Updates reminder status to "sent"

Configuration:

- Reminder days before due (settings.reminder_days_before_due)
- Time of day for notifications (settings.reminder_time)
- Repeat notifications until payment is recorded

Edge Cases:

- Tenant pays before reminder → reminder cancelled

- Tenant pays after reminder sent → status auto-updated

5️⃣ App Flow & Navigation

5.1 User Journey

1. Dashboard/Home
   - Summary: Paid, Due Soon, Overdue tenants
   - Quick links: Add Tenant, Record Payment, View Reminders
2. Tenants List
   - Search, filter, sort tenants
   - Tap → Tenant Details
3. Tenant Details
   - Info: Name, room, phone, start date, next due date
   - Payment history
   - Actions: Add Payment, Edit Info, Send Reminder
4. Add Payment
   - Fields: Tenant, Amount, Date, Method, Notes
   - Updates next due date and reminders
5. Reminders
   - List of upcoming/pending reminders
   - Quick actions: Mark as Paid/Acknowledged
6. Settings
   - Reminder preferences, backup/export, theme, currency

5.2 Navigation Structure

```
Tab Navigator
│
├── Dashboard / Home (Stack)
│    └── Tenant Details
│
```

```
├── Tenants (Stack)
|     └── Tenant Details
|         └── Add Payment
|
├── Reminders (Stack)
|     └── Reminder Details
|
└── Settings
```

UX Principles:

- Color-coded tenant status: Green = Paid, Yellow = Due Soon, Red = Overdue
- Swipe actions on lists for quick payment acknowledgment
- Offline-friendly feedback ("Saved locally")

---

6 Core Business Logic Functions

1. calculateMonthsCovered(amountPaid, monthlyRent) → monthsCovered
2. calculateNextDueDate(lastPaymentDate, monthsCovered) → nextDueDate
3. determineTenantStatus(nextDueDate, today, reminderDaysBeforeDue) → status
4. createReminder(tenantId, nextDueDate, reminderDaysBeforeDue, message)
5. updateReminderStatus(reminderId, newStatus)
6. recordPayment(tenantId, amountPaid, paymentDate, monthlyRent)
   - Updates payments table
   - Recalculates next due date
   - Updates tenant status
   - Creates reminder
7. dailyReminderCheck(today, reminderTime)
   - Triggers notifications for pending reminders
8. Helper Functions:
   - getTenantPaymentHistory(tenantId)

- getUpcomingReminders()
- calculateTotalPaid(tenantId)
- calculateOutstanding(tenantId)

7️⃣ Optional Future Enhancements

- Graphs on dashboard: monthly collection vs expected
- Multi-user access with role-based permissions
- Cloud sync / remote backups
- Multi-language support
- Automatic SMS reminders via Twilio or WhatsApp API

✅ This document now fully encapsulates:

- Architecture & system boundaries
- Database schema
- Data sync & backup strategy
- Notification logic
- App flow & navigation
- Core business logic functions
- Future-proofing and optional enhancements