NyumbaniConnect – System Design Document (SDD)

1. Introduction

1.1 Purpose of the Document
This document outlines the system design for SmartNyumba, a mobile-first platform aimed at solving trust, verification, and service integration challenges in the Kenyan real estate market. It translates business needs into a structured technical framework to guide developers, testers, stakeholders, and administrators through development, testing, and deployment.

1.2 Scope of the System
SmartNyumba enables tenants to find verified rental listings, landlords to connect with pre-screened tenants, and developers/agents to showcase properties with value-added services like WiFi, cleaning, and lease tracking. It supports:
- Tenants searching for homes
- Landlords managing rentals
- Developers & Agents promoting sales-ready homes
- Service Providers offering utility bundles for homes

1.3 Intended Audience
• Software Developers (Frontend & Backend)
• QA Testers
• Product Stakeholders (landlords, agents, developers)
• System Administrators (platform and hosting)

1.4 Glossary
• Verified Listing: Property confirmed by owner credentials.
• Smart Match: Recommendation system matching tenants with listings.
• Service Bundle: Package with utilities like internet, cleaning, or gas.
• MVP: Minimum Viable Product

2. System Overview and Design Philosophy

2.1 High-Level Description
SmartNyumba is composed of:
1. Tenant Web/Mobile App: For verified property search, booking, service selection.
2. Landlord/Developer Dashboard: To list and manage properties.
3. Agent Portal: For managing multiple listings and client interactions.
4. Admin Panel: For platform control, reports, and user verification.

2.2 Design Principles
• Modularity: Separated components (auth, listings, payments)
• Scalability: Firebase/AWS-backed for easy scaling
• Security: Role-based access, encrypted transactions
• User-Centered Design: Mobile-first, minimal learning curve

3. Architectural Design

3.1 Architecture Style
Client–server model with REST APIs. Firebase (Auth + Firestore), Node.js backend, React/Flutter frontend.

3.2 Architecture Diagram
(To be created via Lucidchart or Draw.io — not included here).

3.3 Component Descriptions
• Frontend (React/Flutter): Responsive UI for tenants, landlords, agents
• Backend (Node.js + Firebase): Handles logic, verification, communication
• Database (Firestore/MongoDB): Stores listings, user profiles, bookings
• Admin Dashboard: Access controls, verification logs, reporting

4. Detailed Design

4.1 Module Descriptions
Authentication Module: Firebase Auth with role-based routing (Tenant, Landlord, Agent)
Listings Module: Create, manage, and view property listings
Smart Match Module: Recommend listings based on location, price, and preferences
Booking Module: Allow tenants to request a property, communicate, and confirm interest
Service Integration: Select add-on utilities during the move-in process
Chat/Notification Module: Secure in-app communication and alerts

5. Database Design

5.1 ER Diagram
(To be created using dbdiagram.io or MySQL Workbench).

5.2 Data Dictionary (Sample)
Users Table: user_id, name, email, role, phone, verified
Listings Table: listing_id, title, type, landlord_id, price, location
Bookings Table: booking_id, user_id, listing_id, status, date
Services Table: service_id, name, type, partner_id, cost

6. Data Flow and Control Flow

6.1 Data Flow
Tenant views listings → selects → sends booking → landlord accepts → service bundle attached → dashboard updated.

6.2 Control Flow
Tenant: Search → Book → Chat → Move-in
Landlord: List → Verify → Respond to requests
Agent: Bulk list → Manage bookings → Track analytics

7. Non-Functional Requirements
Performance: Listings and chat must load < 2s under normal network
Security: Encrypted login, HTTPS only, Firebase rules, role-based access
Availability: 99.9% uptime (hosted on Firebase or AWS)
Accessibility: Font sizes, Swahili localization, offline viewing mode

8. Deployment Strategy
Frontend: Firebase Hosting / Vercel
Backend: Node.js on Render or Firebase Functions
Mobile App: Flutter app (Android & iOS)
Versioning: GitHub + CI/CD via GitHub Actions

9. Testing Strategy
Unit Testing: Jest (backend), Flutter Test (mobile)
Integration Test: Postman, Firebase Emulator
UI Testing: Figma Prototypes, Manual QA
Beta Testing: Nairobi + Kiambu Pilot Groups (5 agents, 10 tenants)

10. Risks and Mitigation
Fake property listings: High likelihood, High impact → Mitigation: Manual verification + ID upload
Low app adoption: Medium likelihood, High impact → Mitigation: Marketing campaigns, incentives
Service provider unreliability: Medium likelihood, Medium impact → Mitigation: Allow reviews & ratings
Tech limitations in rural areas: Low likelihood, Medium impact → Mitigation: Lightweight UI + offline mode

11. Appendices

Appendix A: UI Wireframes (via Figma)
Appendix B: Architecture Diagram
Appendix C: API Specs (Swagger/Postman)
Appendix D: Firebase Security Rules
Appendix E: Sprint Board on GitHub