ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTING

Department of Software Engineering program

System Design Specification Document (SDS)

Project Name: AgriLink

Author(s): Agrilink Development Team

Table of Contents Pages

1. Introduction

1.1 Purpose

This System Design Specification (SDS) outlines the technical architecture, components, and implementation strategies for Agrilink Ethiopia – a secure, mobile-first marketplace connecting farmers with buyers in Ethiopia. It ensures the platform is scalable, offline-capable, multilingual, and compliant with local regulations.

1.2 Scope

The document focuses on the MVP for launch in 2026, including frontend, backend, database, integrations and future scaling.

Out of scope: native iOS apps, AI features and input marketplace.

1.3 Audience

- Technical team (developers, architects)
- Product and business stakeholders
- Investors and grant providers
- Government and regulatory bodies (Ministry of Agriculture, ATA)
- External partners (Telebirr, Ethio-Telecom)

1.4 References

- Agrilink SRS Version 1.0 (December 01, 2025)
- AWS Architecture Best Practices
- MongoDB Design Patterns for Scalable Apps
- Ethiopian Digital Strategy 2025

2. Design Principles

The Agrilink architecture is guided by:

- **User-Centric Reliability**: Offline-first for rural farmers with limited connectivity.
- **Performance Priority**: < 3 seconds load time on 3G networks.
- **Inclusivity**: Full multilingual support for English, Amharic, and Afaan Oromoo.
- **Security First**: End-to-end encryption and escrow for transactions.
- **Scalability**: Horizontal scaling to support 500,000+ users by 2030.
- **Cost-Efficiency**: Cloud-native with optimized resource usage.

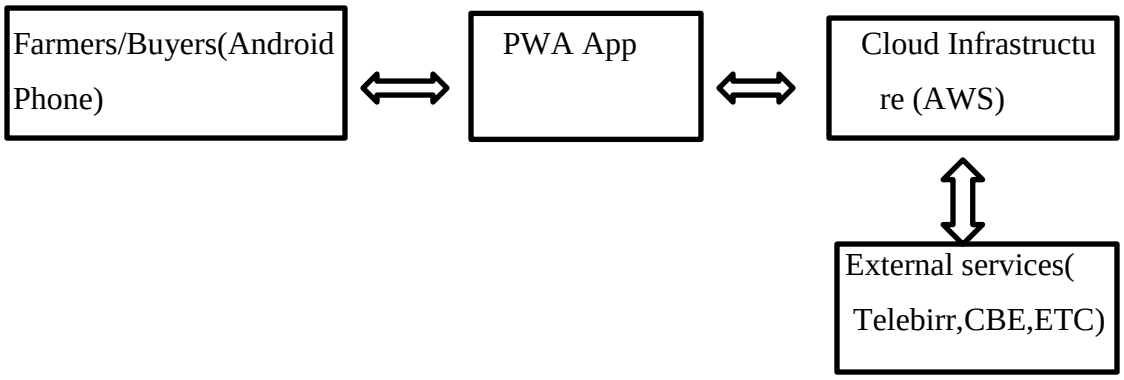
3. High-Level Architecture

3.1 System Context Diagram

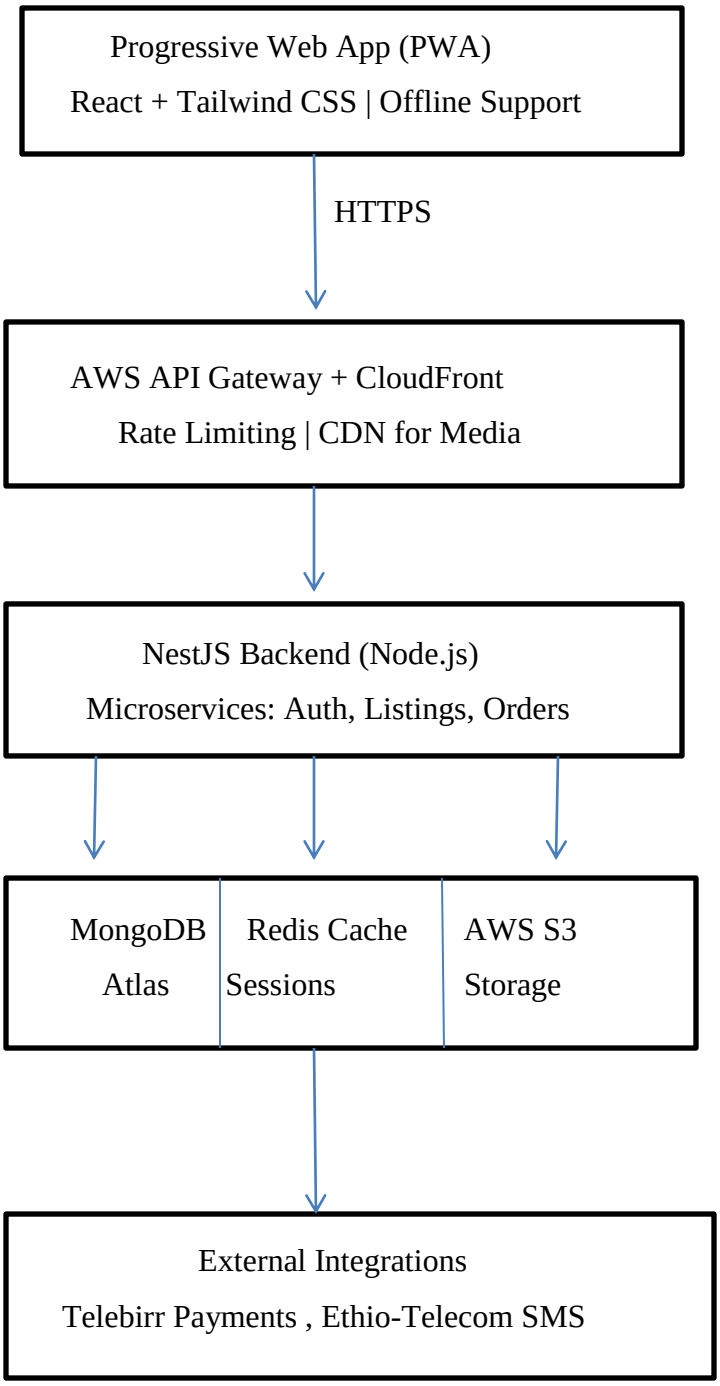
The platform interacts with users (farmers, buyers, admins) via mobile devices, integrates with payment and SMS providers, and runs on cloud infrastructure.

Text-based Diagram:

Text



3.2 Container Diagram Text-based Diagram (Formatted for Word):



3.3 Deployment Architecture

Hosted on AWS Mumbai for low latency. Multi-AZ setup for high availability. Future migration to local Ethiopian data centers for compliance.

4. Technology Stack

4.1 Frontend

- React 18 + Tailwind CSS + Vite: Fast, responsive UI with PWA support.
- Workbox: Offline caching and background sync.

4.2 Backend

- NestJS (Node.js 20 + TypeScript): Modular, scalable framework.
- Socket.io: Real-time chat and notifications.

4.3 Database and Storage

- MongoDB Atlas: Flexible NoSQL for listings and orders.
- Redis (ElastiCache): Caching and session management.
- AWS S3: Media storage with pre-signed URLs.

4.4 Real-Time and Integrations

- Socket.io + Redis adapter: Live features.
- Telebirr API: Payment escrow.
- Ethio-Telecom API: SMS/OTP.
- OpenStreetMap: Location services.

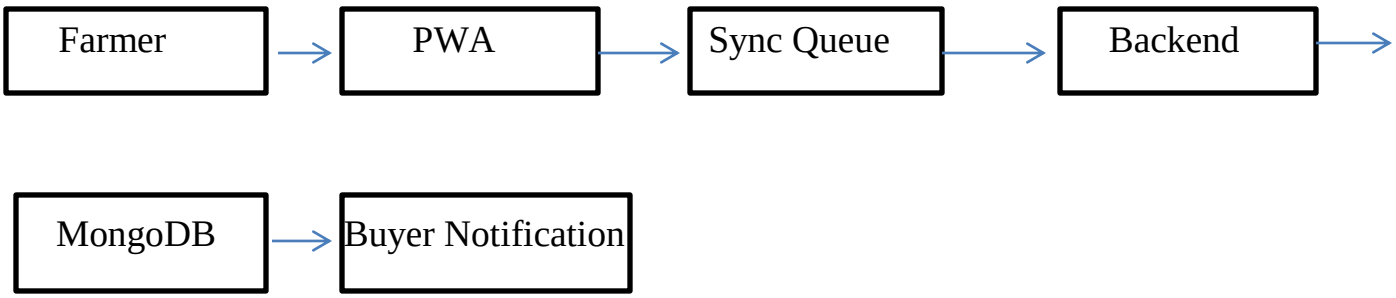
5. Detailed System Design

5.1 Component Breakdown

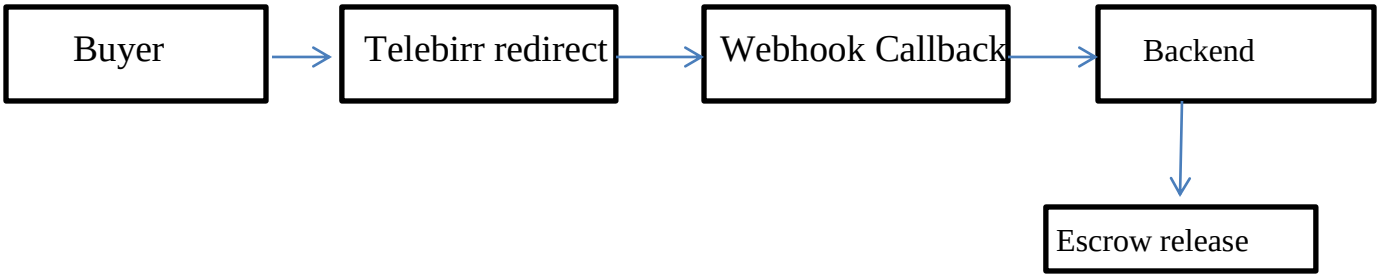
- ✓ **Auth Component**: Handles OTP-based login.
- ✓ **Listings Component**: Offline creation and sync.
- ✓ **Marketplace Component**: Multilingual search and filtering.
- ✓ **Orders Component**: Escrow logic and status tracking.
- ✓ **Admin Component**: Web dashboard for monitoring.

5.2 Data Flow Diagrams

Listing Creation Flow:



Payment Flow:



.

5.3 Offline-First Mechanism

- Service Worker intercepts requests.
- IndexedDB stores local data.
- Background Sync API handles uploads.
- Conflict Resolution: Timestamp-based with server priority.

5.4 Multilingual Support

- i18next library for translations.
- Atlas Search analyzers for Amharic/Afaan Oromoo.
- Language persisted in user profile.

6. Data Model

6.1 MongoDB Schema Overview

Flexible schema with embedded documents for performance. Indexes on location, crop, and status for fast queries.

6.2 Key Collections

- **users**: {phone, role, lang, profile, rating}
- **listings**: {farmerId, crop, quantity, price, photos, geo, status}
- **orders**: {buyerId, listingId, price, escrowStatus, qrCode}
- **transactions**: {orderId, amount, telebirrRef, timestamp}
- **chats**: {orderId, messages[array]}

7. API Design

7.1 Endpoint Structure

- ✓ Base: /api/v1
- ✓ Examples: POST /auth/otp, GET /listings?crop=tomato&zone=Oromia

7.2 OpenAPI Integration

Hosted at /docs (Swagger UI). Auto-generated from NestJS decorators.

8. Security and Compliance

8.1 Authentication and Authorization

- OTP via SMS for login.
- JWT for sessions (httpOnly cookies).
- Role-based access control.

8.2 Data Privacy

- Encryption at rest (Atlas).
- GDPR-like deletion requests.
- Compliance with Ethiopian data laws.

9. Performance and Scalability

9.1 Optimization Strategies

- Caching frequent queries (Redis).
- Progressive image loading.
- Horizontal scaling of backend instances.

9.2 Scaling Roadmap

- Year 1: 50,000 users – Basic cluster.
- Year 2: 200,000 – Read replicas.
- Year 3: 500,000 – Sharding by region.

10. Deployment and Operations

10.1 CI/CD Pipeline

- GitHub Actions + Terraform.
- Blue-green deployments.

10.2 Monitoring and Logging

- Grafana + Prometheus.
- Sentry for errors.

11. Risk Assessment

- Connectivity: Mitigated by offline design.

- Payment Failure: Multiple gateways.
- Fraud: Ratings and verification.

12. Appendix – Additional Diagrams

12.1 Sequence Diagrams

12.2 Infrastructure Cost Estimate

- Year 1: $4,200/month (AWS + Atlas).

13. Glossary

- ✓ PWA: Progressive Web App
- ✓ OTP: One-Time Password
- ✓ CDN: Content Delivery Network