Nyambika – Technical Specification Document

Project Name: Nyambika

Purpose: Al-powered mobile-first e-commerce platform for Rwanda's clothing and fashion

sector.

Target Audience: Local fashion businesses, cooperatives, individuals, and customers.

1. 🗩 System Components and Responsibilities

Component	Description
Frontend	Mobile-friendly user interface for browsing, virtual try-on, placing orders, account management
Backend API	Handles business logic, data management, authentication, and user interactions
Admin Dashboard	Role-based access for sellers, tailors, and platform admins
Al Services	Virtual try-on engine, size suggestion engine
Database	Centralized data storage for users, products, orders, and media
Payment Gateway	Mobile money and card-based payment integrations (MTN, Airtel, PayPal, etc.)
Notification Service	Push, email, and SMS notifications for order status
Delivery Tracking	Logistics coordination with transport providers
CI/CD Pipelines	Automated testing, builds, and deployment routines
Cloud Storage	Scalable image/video storage (e.g., product images, customer try-ons)

2. * Technologies and Frameworks

Layer Technology

Frontend React Native (mobile-first), Tailwind CSS (via NativeWind), Expo (for

quick builds)

Backend Node.js + Express or NestJS (preferred for structure), GraphQL or

REST

Database PostgreSQL (relational), Redis (caching, sessions)

AI/ML TensorFlow.js or MediaPipe for virtual try-on; custom ML model for

body size suggestion

Authentication Firebase Auth or Passport.js (email/OTP/social login)

Storage Firebase Storage / AWS S3 for images

CI/CD GitHub Actions / GitLab CI / Bitbucket Pipelines

Cloud AWS / GCP / Vercel / Railway (for API hosting)

Infrastructure

Monitoring Sentry (errors), LogRocket (user session recording), Prometheus +

Grafana (metrics)

3. API Structure and Key Endpoints

Base URL: https://api.nyambika.rw/

Authentication

POST /api/auth/register POST /api/auth/login POST /api/auth/verify-otp

User & Profile

GET /api/user/me

PATCH /api/user/update-profile

Product Catalog

GET /api/products
GET /api/products/:id

POST /api/products (admin/seller)
PATCH /api/products/:id
DELETE /api/products/:id

Virtual Try-On & Size Suggestion

POST /api/ai/tryon (upload image, select cloth) POST /api/ai/size-suggest (user profile data)

Orders & Payments

POST /api/orders
GET /api/orders/:userId
POST /api/payment/initiate
POST /api/payment/confirm

Notifications

GET /api/notifications
POST /api/notifications/send (admin)

4. 🧠 Al Model Usage

Virtual Try-On Engine

- Approach: Use customer photo (Snapchat-style capture) + cloth image
- Frameworks: MediaPipe (Face + Body landmark detection), OpenCV, TensorFlow.js
- Outcome: Overlay clothing on customer photo, adjust for posture

Body Measurement and Size Suggestion

- **Inputs:** Height, weight, photo-based measurements (shoulders, chest, waist)
- Output: Recommended size (e.g., S, M, L, XL)
- **Model:** Pretrained model fine-tuned on Rwandan demographics

5. Authentication Strategy

- Methods Supported:
 - Email/password with OTP verification
 - o Google & Facebook sign-in
 - Phone number (via Firebase or Twilio)
- Roles: Admin, Seller, Customer
- Session Management: JWT for API, refresh tokens stored securely

6. Responsive and Mobile-First UI

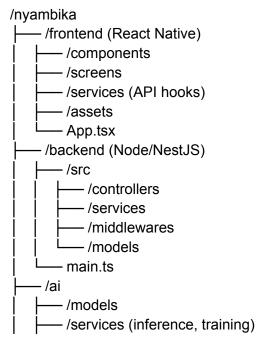
- Framework: React Native (Expo), responsive layout via Tailwind CSS (NativeWind)
- Design Language: Material UI or custom minimal UI kit
- Mobile UX Goals:
 - One-click photo capture
 - Vertical scroll layout
 - Easy checkout
 - Bottom navigation tabs
- Accessibility: Large buttons, local language support (Kinyarwanda, French, English)

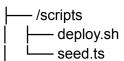
7. A Deployment and CI/CD Strategy

$\textbf{Dev} \rightarrow \textbf{Staging} \rightarrow \textbf{Production Flow}$

- Branching Model: GitFlow (feature/dev/main)
- CI Tools: GitHub Actions or GitLab CI
- Pipelines:
 - Linting + Formatting (ESLint, Prettier)
 - Tests (Jest, React Testing Library)
 - o Build & Deploy
- Frontend Deployment: Vercel / Expo Hosting
- Backend Deployment: Railway, AWS Lambda (for serverless), or traditional VM
- Database: Supabase/PostgreSQL managed instance
- Storage: AWS S3 / Firebase Storage

Sample Developer Folder Structure





Next Steps for Developers

- 1. Set up local environments (mobile emulator + backend server)
- 2. Connect React Native app with backend endpoints
- 3. Integrate AI SDK or services for try-on and size recommendation
- 4. Implement payment and delivery APIs
- 5. Set up CI/CD pipelines for auto-deployments
- 6. Launch beta and gather feedback from initial users