■# Milestone 1 Completion + System Architecture & Technical Design

Project: **UK2ME Revamp (Uk2meonline)**

Date: **January 25, 2026**

Milestone: **Milestone 1 – Planning & Initial Development (?300,000)**

1) Milestone 1 Completion Statement

This document confirms that Milestone 1 has been completed and delivered. All items listed in the milestone scope have been addressed, reviewed, and integrated into the repository.

Milestone Scope (Completed)

1) **Requirements analysis & finalization**

- Functional requirements for client, admin, and backend services documented and aligned.

- Core flows defined: product resolution, quote creation, order creation, payment webhook handling, order tracking.

- Authentication requirements defined (cookie-based sessions for v1, gated pages, admin access).

- Non-functional requirements defined: TypeScript everywhere, App Router, Prisma + Postgres, Redis + BullMQ, Docker, CI/CD, AWS ECS Fargate.

2) **System architecture & technical design**

- Architecture documented (see sections below).

- Component boundaries, data flow, and deployment model documented.

- Security posture defined (session cookies, SSM secrets, IAM roles, OIDC for CI/CD).

3) **UI/UX wireframes (if applicable)**

- Baseline UI screens created in Next.js for client and admin.

- Navigation, onboarding, and order tracking flows designed and implemented.

- Dashboard, stores directory, and profile flows exist with coherent layout and theming.

4) **Project setup (frontend & backend)**

- Monorepo with pnpm workspaces established.

- Next.js App Router projects created for client and admin.

- Next.js API-only backend created.

- Dockerfiles for each app + root `docker-compose.yml` added.

- GitHub Actions CI/CD workflow for backend deploy added.

5) **Database schema & core logic setup**

- Prisma schema defined and migrated.

- Core models implemented: User, Address, Order, OrderItem, ProductSnapshot, Quote, Payment, PurchaseAttempt, Shipment, OrderEvent, Session, EmailVerificationToken, PasswordResetToken.

- Queue scaffolding in place (BullMQ) for resolve product, purchase attempt, and tracking jobs.

Result: Milestone 1 is complete and production-ready for iteration into Milestone 2.

2) System Architecture

2.1 High-Level Architecture

```
+--------------------+        HTTPS        +------------------------+
|  Client (Next.js)  |  <----------------> |  Backend API (Next.js)  |
|  Vercel            |                     |  ECS Fargate + ALB      |
+--------------------+                     +------------------------+
           |                                           |
           |                                           |
           |                                           v
           |                              +------------------------+
           |                              | PostgreSQL (RDS)        |
           |                              +------------------------+
           |                                           |
           |                                           v
           |                              +------------------------+
           |                              | Redis (ElastiCache)     |
           |                              +------------------------+
           |
           |
           v
+--------------------+
|  Admin (Next.js)    |
|  Vercel             |
+--------------------+
```

2.2 Component Responsibilities

Client (Next.js App Router)

- Customer-facing UI for browsing stores, submitting product links, requesting quotes, paying, tracking orders, and managing profile.

- Auth-gated routes: dashboard, orders, profile, pay, quote, track order, invoices.

- Client uses API proxy routes to reach backend securely via HTTPS.

Admin (Next.js App Router)

- Admin operations dashboard.

- Order and shipment management pages.

- Role management page for internal permissions planning (v1 stub).

Backend (Next.js API-only)

- REST endpoints for product resolution, quotes, orders, payments, admin actions.

- Session-based auth with cookie storage and server-side validation.

- Prisma ORM integration with Postgres.

- BullMQ queues for background jobs.

PostgreSQL (RDS)

- Primary system of record for users, orders, product snapshots, payments, events.

Redis (ElastiCache)

- Queue and lock storage for background jobs.

2.3 Deployment & Infra

- **Backend**: AWS ECS Fargate (containerized), fronted by ALB.

- **Images**: AWS ECR.

- **Secrets**: AWS SSM Parameter Store (DB URL, Redis URL, SMTP, cookie names).

- **Logging**: CloudWatch Logs.

- **CI/CD**: GitHub Actions with OIDC to AWS IAM role.

- **Frontend**: Vercel deployments for client/admin.

2.4 Security & Compliance

- Admin access via cookie session stub (v1) with secure cookie attributes.

- Client auth uses session tokens stored in database with TTL.

- Secrets stored in SSM and injected into ECS task definition at runtime.

- CI/CD uses OIDC to avoid static AWS keys.

3) Technical Design

3.1 Monorepo Structure

```
/apps
  /client     (Next.js App Router)
  /admin      (Next.js App Router)
  /backend    (Next.js API-only)
/packages
  /shared     (shared types + utils)
  /ui         (optional shared UI primitives)
/infra
  /terraform  (AWS infrastructure)
/docs
/tasks
```

3.2 Backend API Endpoints (v1)

Public / Client:

- `POST /v1/resolve-product` ? ProductSnapshot (mock extraction)

- `POST /v1/quotes` ? Quote

- `POST /v1/payments/webhook` ? Payment (idempotent)

- `POST /v1/orders` ? Order

- `GET /v1/orders/:id`

- `GET /v1/orders`

Auth:

- `POST /v1/auth/signup`

- `POST /v1/auth/login`

- `POST /v1/auth/logout`

- `GET /v1/auth/session`

- `POST /v1/auth/verify-email`

- `POST /v1/auth/resend-verification`

- `POST /v1/auth/forgot-password`

- `POST /v1/auth/reset-password`

Profile:

- `GET /v1/me`
- `PATCH /v1/me`
- `POST /v1/me/addresses`
- `PATCH /v1/me/addresses/:id`

Admin:

- `GET /v1/admin/orders?status=`
- `POST /v1/admin/orders/:id/status`
- `POST /v1/admin/orders/:id/purchase-attempts`
- `POST /v1/admin/shipments`

3.3 Data Model (Prisma)

Core models:

- **User** (id, email, name, phone, passwordHash, emailVerifiedAt)
- **Address** (line1, line2, city, state, postalCode, country, type: SHIPPING/BILLING, isDefault)
- **ProductSnapshot** (url, title, imageUrl, price, currency, raw)
- **Quote** (snapshot, size, color, qty, totals)
- **Order** (status, total, currency, relations)
- **OrderItem**
- **Payment**
- **PurchaseAttempt**
- **Shipment**
- **OrderEvent**
- **Session**
- **EmailVerificationToken**
- **PasswordResetToken**

3.4 Auth & Session Design

- Sessions stored in DB with TTL.
- Cookies: `client_session` (client), `admin_session` (admin).
- Email verification enforced at login.

- Password reset tokens stored with expiry.

3.5 Queue Design (BullMQ)

- `resolveProduct` job: fetch product metadata (stub).
- `purchaseAttempt` job: execute purchase flow (stub).
- `trackShipment` job: update shipment statuses (stub).

3.6 Frontend UI/UX Scope (Client)

Implemented pages:
- `/` Landing
- `/preview`, `/quote`, `/pay`
- `/orders` and `/orders/[id]`
- `/dashboard`
- `/track-order`
- `/invoices`
- `/profile`
- `/stores`
- `/login`, `/signup`, `/verify-email`, `/forgot-password`, `/reset-password`

3.7 CI/CD Overview

- GitHub Actions on `main` push:
1. Install deps (pnpm)
2. Lint, typecheck, tests
3. Build backend image
4. Push to ECR
5. Update ECS task definition and service

4) Evidence of Completion

- Monorepo created with required structure and tooling.

- Backend deployed to AWS ECS Fargate.
- Client and admin deployed to Vercel.
- Prisma schema implemented and migrations created.
- API endpoints implemented and integrated with client flows.
- Auth and session system implemented.

5) Acceptance

Milestone 1 is complete and ready for stakeholder sign-off.

Prepared by: **Engineering Team (Mr Femi & Olukayode Soliu)**

Approved by: **Client**