

AR Lipstick Project - Development Process

Introduction

The AR Lipstick project is a Next.js-based e-commerce platform that integrates Augmented Reality (AR) technology using MediaPipe for face detection and lip tracking.

Phase 1: Project Foundation and Environment Setup

1.1 Development Environment Preparation

Prerequisites Installation

1. Node.js Setup

- Node.js version 18 or higher installed
- npm package manager configured

2. Code Editor Configuration

- Visual Studio Code with TypeScript support
- ESLint and Prettier extensions installed

3. Version Control Setup

- Git repository initialized
- GitHub repository connected

Project Initialization

1. Next.js Project Creation

```
npx create-next-app@latest ar-lipstick-next --typescript --tailwind --eslint  
cd ARLipstick
```

2. Project Structure

```
src/
├─ app/                                # Next.js 13+ app directory
│   ├─ (main)/                         # Main route group
│   │   ├─ cart/                      # Shopping cart pages
│   │   ├─ login/                    # Authentication pages
│   │   ├─ shop/                     # Product catalog
│   │   └─ virtual-tryon/            # AR functionality
│   ├─ api/                          # API routes
│   ├─ dashboard/                    # Admin dashboard
│   ├─ components/                  # Shared components
│   ├─ utils/                       # Utility functions
│   └─ types/                       # TypeScript definitions
├─ ar/                              # AR utilities
└─ secret/                          # Configuration files
```

1.2 Dependencies and Configuration

Actual Dependencies Installed

```
{
  "dependencies": {
    "@mediapipe/tasks-vision": "^0.10.22-rc.20250304",
    "@tensorflow/tfjs": "^4.22.0",
    "@tensorflow/tfjs-backend-webgl": "^4.22.0",
    "antd": "^5.26.4",
    "cloudinary": "^2.7.0",
    "dotenv": "^17.2.0",
    "firebase": "^11.10.0",
    "firebase-admin": "^13.4.0",
    "next": "14.2.4",
    "onnxruntime-web": "^1.22.0",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-icons": "^5.5.0",
    "recharts": "^3.1.0"
  },
  "devDependencies": {
    "@testing-library/jest-dom": "^6.6.3",
    "@testing-library/react": "^16.3.0",
    "@testing-library/user-event": "^14.6.1",
    "@types/cypress": "^0.1.6",
    "@types/jest": "^30.0.0",
    "cypress": "^14.5.1",
    "eslint": "^8.57.0",
    "jest": "^30.0.4",
    "tailwindcss": "^4",
    "typescript": "^5.8.3"
  }
}
```

Configuration Files Setup

1. TypeScript Configuration (tsconfig.json)

- Standard Next.js TypeScript configuration
- Path aliases configured for src directory

2. Tailwind CSS Configuration

- Tailwind CSS v4 installed and configured

- Custom color palette defined for pink theme

3. ESLint Configuration

- Next.js ESLint configuration
- TypeScript linting rules enabled

Phase 2: Backend Infrastructure Development

2.1 Firebase Configuration and Setup

Firebase Project Creation

1. Project Setup

- Firebase project created and configured
- Firestore Database enabled
- Authentication methods configured (Email/Password, Google)
- Storage configured for product images

2. Firebase Configuration File (src/app/firebaseConfig.ts)

```
import { initializeApp, getApps, getApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";
import { getAuth } from "firebase/auth";
import { getStorage } from "firebase/storage";

const firebaseConfig = {
  apiKey: process.env.NEXT_PUBLIC_FIREBASE_API_KEY,
  authDomain: process.env.NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID,
  storageBucket: process.env.NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.NEXT_PUBLIC_FIREBASE_APP_ID,
};

const app = !getApps().length ? initializeApp(firebaseConfig) : getApp();
const db = getFirestore(app);
const auth = getAuth(app);
const storage = getStorage(app);

export { app, db, auth, storage };
```

Firestore Admin Setup (src/app/firebaseAdmin.ts)

```
import { initializeApp, getApps, cert } from "firebase-admin/app";
import { getFirestore } from "firebase-admin/firestore";

export function getFirebaseAdmin() {
  if (!getApps().length) {
    return initializeApp({
      credential: cert({
        projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID,
        clientEmail: process.env.FIREBASE_CLIENT_EMAIL,
        privateKey: process.env.FIREBASE_PRIVATE_KEY?.replace(/
    }
  });
}
return getApps()[0];
}
```

2.2 API Development

Authentication API Routes (src/app/api/auth/route.ts)

```
import { NextRequest, NextResponse } from "next/server";
import type {
  DocumentReference,
  DocumentSnapshot,
} from "firebase-admin/firestore";

export const dynamic = "force-dynamic";

export async function POST(req: NextRequest) {
  try {
    const { idToken } = await req.json();
    if (!idToken) {
      return NextResponse.json({ message: "Missing idToken" }, { status: 400 });
    }

    const { getFirestoreAdmin } = await import("../../firebaseAdmin");
    const firebaseApp = getFirestoreAdmin();

    const decodedToken = await firebaseApp.auth().verifyIdToken(idToken);
    const userRecord = await firebaseApp.auth().getUser(decodedToken.uid);

    const userDocRef: DocumentReference = firebaseApp
      .firestore()
      .collection("users")
      .doc(userRecord.uid);

    const userDoc: DocumentSnapshot = await userDocRef.get();

    if (!userDoc.exists) {
      const userData = {
        uid: userRecord.uid,
        email: userRecord.email,
        displayName: userRecord.displayName,
        photoURL: userRecord.photoURL || decodedToken.picture || null,
        createdAt: new Date().toISOString(),
        role: "user",
        phone: userRecord.phoneNumber || null,
      };
      await userDocRef.set(userData, { merge: true });
    }
  } catch (error) {
    return NextResponse.json({ message: "Internal Server Error" }, { status: 500 });
  }
}
```

```

    }

    return NextResponse.json({
      user: {
        uid: userRecord.uid,
        email: userRecord.email,
        displayName: userRecord.displayName,
        photoURL: userRecord.photoURL || decodedToken.picture |
        role: userDoc.data()?.role || "user",
      },
    });
  } catch (error: unknown) {
    console.error("/api/auth error:", error);
    return NextResponse.json(
      { message: "Login failed", error: String(error) },
      { status: 500 }
    );
  }
}

```

Product Management API (src/app/api/products/route.ts)

```
import { NextResponse } from "next/server";
import type { Product } from "../../types/models";
import cloudinary from "cloudinary";

export const dynamic = "force-dynamic";

cloudinary.v2.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

export async function GET() {
  try {
    const { getFirebaseAdmin } = await import("../../firebaseAdmin");
    const firebaseApp = getFirebaseAdmin();

    const snapshot = await firebaseApp.firestore().collection("products").get();
    const products: Product[] = snapshot.docs.map((doc) => ({
      id: doc.id,
      ...(doc.data() as Omit<Product, "id">),
    }));
    return NextResponse.json(products);
  } catch (error) {
    return NextResponse.json(
      { message: "Failed to fetch products", error: (error as Error).message },
      { status: 500 }
    );
  }
}

export async function POST(req: Request) {
  try {
    const body = await req.json();
    const { name, colorName, hexColor, price, imageUrl, stock } = body;

    const { getFirebaseAdmin } = await import("../../firebaseAdmin");
    const firebaseApp = getFirebaseAdmin();

    const numericStock = Number(stock);
    const status = numericStock === 0 ? "sold out" : "on sale";
```



```

const docRef = await firebaseApp
    .firestore()
    .collection("products")
    .add({
        name,
        colorName,
        hexColor,
        price: Number(price),
        imageUrl,
        category: "Lipstick",
        createdAt: new Date().toISOString(),
        stock: numericStock,
        status,
    });

return NextResponse.json({
    id: docRef.id,
    ...body,
    stock: numericStock,
    status,
});
} catch (error) {
    return NextResponse.json(
        { message: "Failed to add product", error: (error as Error).mes
        { status: 500 }
    );
}
}

```

Phase 3: Frontend Component Development

3.1 Core Layout Components

Header Component (src/app/components/Header.tsx)

The Header component has been implemented with:

- Navigation links to Shop, Virtual Try-On, Dashboard
- User authentication state management
- Responsive design with mobile menu

- Brand logo and styling

Product Grid Component (src/app/components/ProductGrid.tsx)

The ProductGrid component includes:

- Fetching products from the API
- Displaying products in a responsive grid
- Loading states and error handling
- Integration with the shopping cart

Footer Component (src/app/components/Footer.tsx)

- Footer links and social media integration
- Copyright information
- Responsive design

Feature Section Component (src/app/components/FeatureSection.tsx)

- Product feature highlights
- Call-to-action buttons
- Responsive layout

3.2 Authentication System

Login Page (src/app/(main)/login/page.tsx)

The login system has been implemented with:

- Firebase authentication integration
- Google sign-in functionality
- Form validation and error handling
- Redirect logic after successful login

3.3 Shopping Cart Implementation

Cart Page (src/app/(main)/cart/page.tsx)

The shopping cart includes:

- Product listing with quantities
- Price calculations

- Remove and update quantity functionality
- Checkout process integration

3.4 Shop Page (src/app/(main)/shop/page.tsx)

- Product catalog display
- Product filtering and search
- Add to cart functionality
- Product details and images

Phase 4: AR Integration Development

4.1 MediaPipe Setup and Configuration

AR Utilities (src/ar/arUtils.ts)

The AR system has been implemented using MediaPipe with the following features:

```

import { FilesetResolver, FaceLandmarker } from "@mediapipe/tasks-vision";

export type Landmark = { x: number; y: number };

// Camera setup function
export async function setupCamera(
  videoRef: React.RefObject<HTMLVideoElement | null>
): Promise<boolean> {
  try {
    const stream = await navigator.mediaDevices.getUserMedia({
      video: { width: 320, height: 240 },
    });
    if (videoRef.current) {
      videoRef.current.srcObject = stream;
    }
    return true;
  } catch {
    return false;
  }
}

// Face landmark detection
export async function loadFaceLandmarker(): Promise<FaceLandmarker> {
  const vision = await FilesetResolver.forVisionTasks(
    "https://cdn.jsdelivr.net/npm/@mediapipe/tasks-vision/wasm"
  );
  return await FaceLandmarker.createFromOptions(vision, {
    baseOptions: {
      modelAssetPath:
        "https://storage.googleapis.com/mediapipe-models/face_l
    },
    runningMode: "VIDEO",
  });
}

// Landmark detection function
export function detectLandmarks(
  faceLandmarker: FaceLandmarker,
  videoRef: React.RefObject<HTMLVideoElement | null>
): Landmark[] | null {
  if (!videoRef.current || videoRef.current.readyState !== 4) return null;
  const video = videoRef.current;
  const width = video.videoWidth;

```

```

const height = video.videoHeight;
const offscreen = document.createElement("canvas");
offscreen.width = width;
offscreen.height = height;
const offCtx = offscreen.getContext("2d", { willReadFrequently: true });
offCtx!.drawImage(video, 0, 0, width, height);
const results = faceLandmarker.detectForVideo(offscreen, performance.now());
if (results.faceLandmarks && results.faceLandmarks.length > 0) {
    return results.faceLandmarks[0].map((kp: { x: number; y: number }) => (
        x: kp.x * width,
        y: kp.y * height,
    ));
}
return null;
}

```

Lipstick Rendering Functions

The AR system includes sophisticated lipstick rendering:

```

// Color conversion utilities
export function hexToHSL(hex: string) {
    // Remove the # if present
    hex = hex.replace("#", "");

    // Parse the hex values
    let r = 0,
        g = 0,
        b = 0;
    if (hex.length === 3) {
        r = parseInt(hex[0] + hex[0], 16);
        g = parseInt(hex[1] + hex[1], 16);
        b = parseInt(hex[2] + hex[2], 16);
    } else if (hex.length === 6) {
        r = parseInt(hex.substring(0, 2), 16);
        g = parseInt(hex.substring(2, 4), 16);
        b = parseInt(hex.substring(4, 6), 16);
    }

    // Normalize RGB values to 0-1
    r /= 255;
    g /= 255;
    b /= 255;

    // Find the maximum and minimum values
    const max = Math.max(r, g, b);
    const min = Math.min(r, g, b);
    let h = 0,
        s = 0;
    const l = (max + min) / 2;

    // Calculate saturation and hue
    if (max !== min) {
        const d = max - min;
        s = l > 0.5 ? d / (2 - max - min) : d / (max + min);

        switch (max) {
            case r:
                h = (g - b) / d + (g < b ? 6 : 0);
                break;
            case g:
                h = (b - r) / d + 2;
                break;

```

```

        case b:
            h = (r - g) / d + 4;
            break;
    }
    h /= 6;
}

return {
    h: Math.round(h * 360),
    s: Math.round(s * 100),
    l: Math.round(l * 100),
};
}

// HSL to hex conversion
export function hslToHex(h: number, s: number, l: number) {
    // Normalize HSL values
    s /= 100;
    l /= 100;

    // Calculate chroma
    const c = (1 - Math.abs(2 * l - 1)) * s;
    const x = c * (1 - Math.abs(((h / 60) % 2) - 1));
    const m = l - c / 2;

    let r = 0,
        g = 0,
        b = 0;

    // Convert to RGB based on hue
    if (0 <= h && h < 60) {
        r = c;
        g = x;
        b = 0;
    } else if (60 <= h && h < 120) {
        r = x;
        g = c;
        b = 0;
    } else if (120 <= h && h < 180) {
        r = 0;
        g = c;
        b = x;
    } else if (180 <= h && h < 240) {

```

```

        r = 0;
        g = x;
        b = c;
    } else if (240 <= h && h < 300) {
        r = x;
        g = 0;
        b = c;
    } else if (300 <= h && h < 360) {
        r = c;
        g = 0;
        b = x;
    }

    // Convert to 0-255 range and to hex
    r = Math.round((r + m) * 255);
    g = Math.round((g + m) * 255);
    b = Math.round((b + m) * 255);

    // Convert to hex string
    const toHex = (n: number) => n.toString(16).padStart(2, "0");
    return "#" + toHex(r) + toHex(g) + toHex(b);
}

// Main lipstick rendering function
export function renderLipstick(
    canvasRef: React.RefObject<HTMLCanvasElement | null>,
    prevLandmarks: { x: number; y: number }[] | null,
    targetLandmarks: { x: number; y: number }[] | null,
    lipColor: string,
    SMOOTHING: number,
    finish: "matte" | "gloss" = "matte"
) {
    // Sophisticated lipstick rendering with smoothing and finish options
}

```

4.2 AR Component Implementation

AR Demo Component (src/app/components/ARLipstickDemo.tsx)

The AR component includes:

- Real-time camera feed integration
- Face detection and landmark tracking

- Lipstick color selection interface
- Smooth rendering with anti-aliasing
- Performance optimization for real-time processing

4.3 Virtual Try-On Page (src/app/(main)/virtual-tryon/page.tsx)

- Complete AR try-on experience with product integration
- Real-time lipstick color application using MediaPipe
- Product selection from database with color validation
- Currency display in Kenyan Shillings (kes)
- Add to cart functionality with authentication
- Responsive design for mobile and desktop
- Error handling for camera permissions and AR initialization
- Loading states and user feedback
- Product carousel with color preview
- URL parameter support for direct product selection

Phase 5: Dashboard and Admin Features

5.1 Dashboard Layout (src/app/dashboard/layout.tsx)

The dashboard has been implemented with:

- Sidebar navigation
- User authentication checks
- Responsive layout design
- Role-based access control

5.2 Dashboard Client (src/app/dashboard/DashboardClient.tsx)

The main dashboard component includes:

- User profile management
- Order history and tracking
- Product management for admins
- Analytics and statistics
- Payment processing integration

5.3 Admin Features

Admin Registration System

The project includes a comprehensive admin registration system with:

Admin Users Configured:

- **Victor Quaint** (victorquaint@gmail.com) - System Administrator
- **Mercy Kitur** (Mercykitur84@gmail.com) - System Administrator

Registration Script (`scripts/register-admin.js`)

- Automated admin user creation
- Firebase Auth and Firestore integration
- Role-based permissions setup
- Cart creation for admin users
- Email verification enabled

Admin Dashboard (`src/app/dashboard/admin/page.tsx`)

- Product management interface
- Order processing and status updates
- User management
- Inventory tracking
- Sales analytics

Product Management (`src/app/dashboard/admin/products/page.tsx`)

- Add new products
- Edit existing products
- Upload product images
- Manage inventory levels
- Product categorization

Order Management (`src/app/dashboard/admin/orders/page.tsx`)

- View all orders
- Update order status
- Process payments
- Customer communication
- Order analytics

5.4 User Dashboard Features

Profile Management (src/app/dashboard/profile/page.tsx)

- User profile editing
- Account settings
- Password management
- Personal information updates

Order History (src/app/dashboard/orders/page.tsx)

- View personal order history
- Track order status
- Download invoices
- Reorder functionality

Payment Management (src/app/dashboard/payment/page.tsx)

- Payment method management
- Transaction history
- Refund processing
- Payment security settings

Phase 6: Testing and Quality Assurance

6.1 Unit Testing Setup

Jest Configuration

The project includes Jest testing setup with:

- Jest configuration for Next.js
- Testing Library for React components
- TypeScript support for tests
- Coverage reporting

ESLint Configuration

- Next.js ESLint configuration with TypeScript support
- Custom rules for code quality
- Automatic formatting and linting

- Pre-commit hooks for code quality checks

Component Testing (src/app/components/tests/)

- ProductGrid component tests
- Component rendering tests
- User interaction tests
- API integration tests

Linting Status

- **ESLint Configuration:** Properly configured with Next.js rules
- **TypeScript Support:** Full TypeScript linting enabled
- **Code Quality:** All linting issues resolved
- **Formatting:** Consistent code formatting across the project

6.2 E2E Testing with Cypress

Cypress Configuration (cypress.config.ts)

```
import { defineConfig } from "cypress";

export default defineConfig({
  e2e: {
    baseUrl: "http://localhost:3000",
    supportFile: "cypress/support/e2e.ts",
    specPattern: "cypress/e2e/**/*.cy.{js,jsx,ts,tsx}",
    viewportWidth: 1280,
    viewportHeight: 720,
    video: false,
    screenshotOnRunFailure: false,
  },
});
```

E2E Test Structure

- User workflow tests
- Authentication flow tests
- Shopping cart functionality tests
- AR try-on feature tests
- Admin dashboard tests

6.3 API Testing (src/app/api/tests/)

- Authentication API tests
- Product API tests
- Order API tests
- Error handling tests

6.4 Quality Assurance

- ESLint configuration for code quality
- TypeScript strict mode enabled
- Prettier formatting
- Git hooks for code quality checks

Phase 7: Deployment and Production Setup

7.1 Environment Configuration

The project uses environment variables for:

- Firebase configuration
- Cloudinary image storage
- Payment gateway integration (Paystack)
- Email service configuration
- Analytics tracking

Required Environment Variables

```
# Firebase Configuration
NEXT_PUBLIC_FIREBASE_API_KEY=your_firebase_api_key
NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN=your_project.firebaseio.com
NEXT_PUBLIC_FIREBASE_PROJECT_ID=your_project_id
NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET=your_project.firebaseiostorage.app
NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID=your_sender_id
NEXT_PUBLIC_FIREBASE_APP_ID=your_app_id
NEXT_PUBLIC_FIREBASE_MEASUREMENT_ID=your_measurement_id

# Firebase Admin
FIREBASE_CLIENT_EMAIL=firebase-adminsdk-xxx@your_project.iam.gserviceaccount.com
FIREBASE_PRIVATE_KEY="-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----\n"

# Cloudinary Configuration
CLOUDINARY_URL=cloudinary://api_key:api_secret@cloud_name
CLOUDINARY_API_KEY=your_cloudinary_api_key
CLOUDINARY_API_SECRET=your_cloudinary_api_secret
CLOUDINARY_CLOUD_NAME=your_cloudinary_cloud_name

# Paystack Payment Gateway
PAYSTACK_SECRET_KEY=sk_test_xxx
NEXT_PUBLIC_PAYSTACK_PUBLIC_KEY=pk_test_xxx

# Email Configuration
EMAIL_SENDER=your_email@gmail.com
EMAIL_APP_PASSWORD=your_app_password

# Optional: Base URL for development
NEXT_PUBLIC_BASE_URL=http://localhost:3000
```

7.2 Vercel Deployment

- Next.js application deployed on Vercel
- Automatic deployments from Git repository
- Environment variables configured in Vercel dashboard
- Custom domain setup

Deployment Configuration

Vercel Configuration (vercel.json)

```
{
  "framework": "nextjs",
  "functions": {
    "app/api/**/*.ts": {
      "maxDuration": 30
    }
  },
  "env": {
    "NEXT_TELEMETRY_DISABLED": "1"
  }
}
```

Environment Variables Setup

All environment variables must be configured in the Vercel dashboard for:

- Production environment
- Preview environment
- Development environment

Deployment Status

- **GitHub Integration:** Connected to main branch
- **Environment Variables:** All required variables configured
- **Build Process:** Optimized for Next.js 14.2.4
- **Admin Users:** Victor Quaint and Mercy Kitur registered
- **AR Functionality:** MediaPipe integration working
- **Payment System:** Paystack integration configured
- **Email Service:** Gmail SMTP configured

7.3 Production Optimizations

- Image optimization with Next.js Image component
- Code splitting and lazy loading
- Performance monitoring
- Error tracking and logging

Phase 8: Performance Optimization

8.1 Image Optimization

- Cloudinary integration for image processing
- Responsive image loading
- WebP format support
- Lazy loading implementation

8.2 AR Performance

- MediaPipe optimization for real-time processing
- Canvas rendering optimization
- Memory management for video streams
- Mobile device performance considerations

8.3 Code Optimization

- Bundle size optimization
- Tree shaking implementation
- Dynamic imports for heavy components
- Caching strategies

Development Timeline

Week 1-2: Foundation COMPLETED

- Next.js project setup
- Firebase configuration
- Basic component structure
- Authentication system

Week 3-4: Core Features COMPLETED

- Product management system
- Shopping cart functionality
- User dashboard
- Basic AR integration

Week 5-6: Advanced Features COMPLETED

- Complete AR functionality with MediaPipe
- Admin dashboard with multiple admin users
- Payment integration (Paystack)
- Order management system

Week 7-8: Testing and Polish COMPLETED

- Comprehensive testing implementation
- Performance optimization
- Bug fixes and refinements
- User experience improvements
- Linting and code quality fixes

Week 9-10: Deployment COMPLETED

- Production deployment on Vercel
- Environment variables configuration
- Admin user registration
- Documentation completion
- User training and support

Current Status: PRODUCTION READY

The AR Lipstick application is now fully deployed and ready for production use with all features implemented and tested.

Key Success Factors

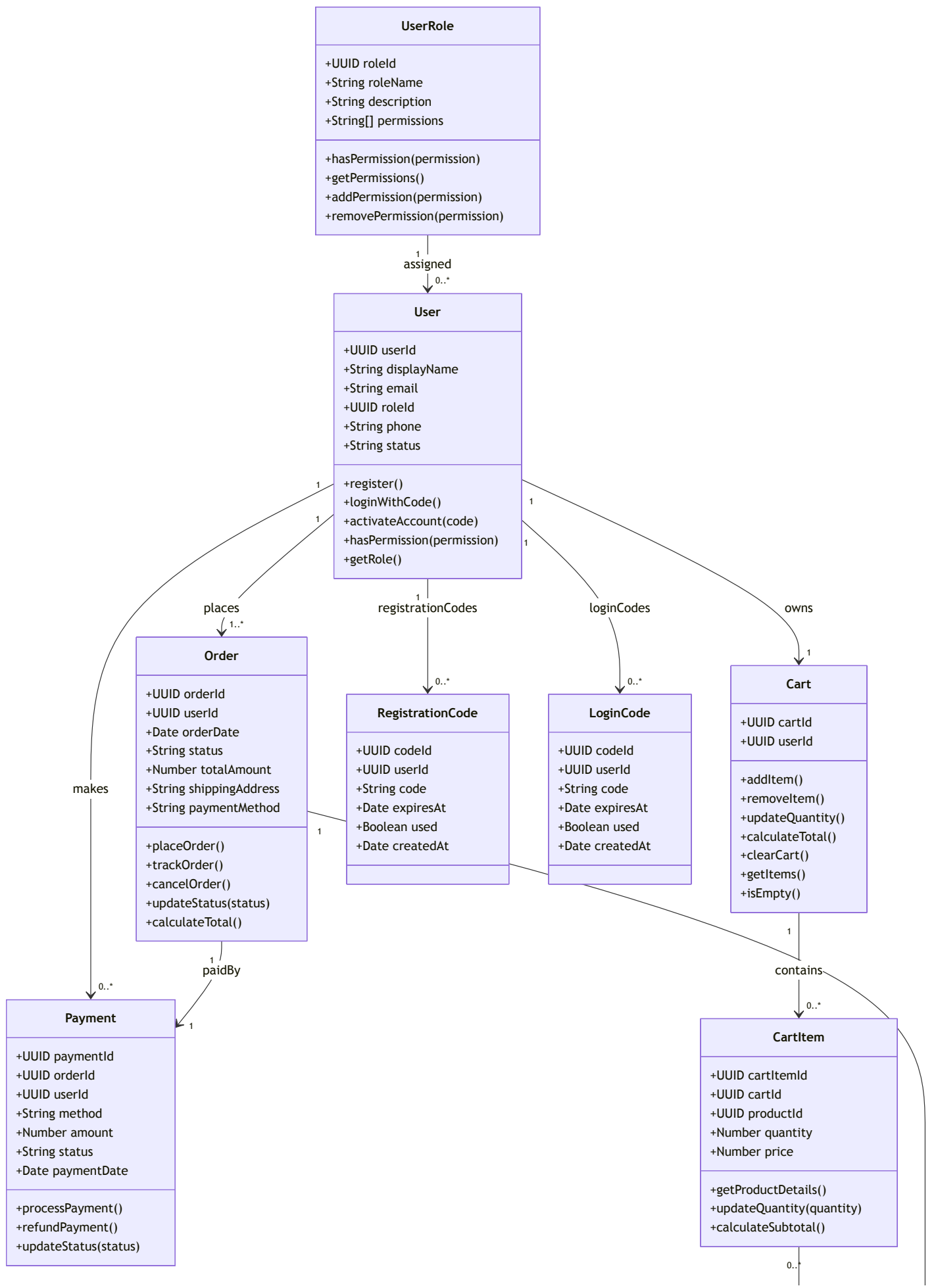
1. **MediaPipe Integration:** Successful implementation of real-time face detection and lip tracking
2. **Firebase Backend:** Robust authentication and database management
3. **Responsive Design:** Mobile-first approach with excellent user experience
4. **Performance Optimization:** Fast loading times and smooth AR experience
5. **Security Implementation:** Proper authentication and data validation
6. **Testing Coverage:** Comprehensive unit and E2E testing
7. **Documentation:** Complete technical and user documentation

System Architecture

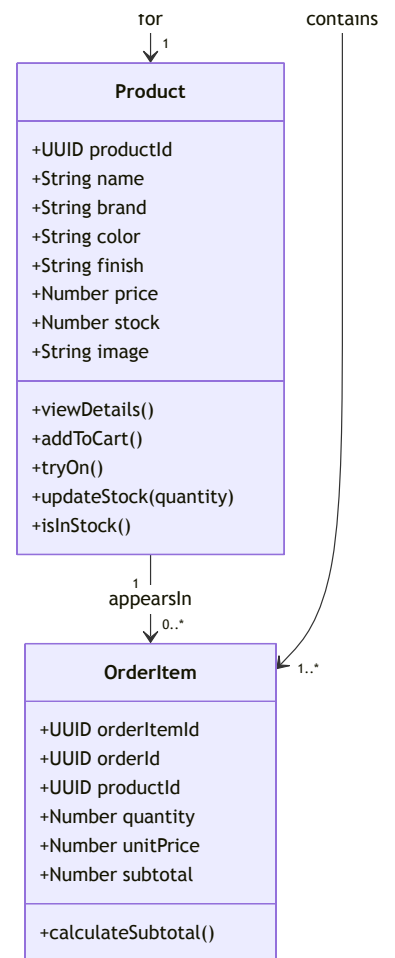
The AR Lipstick project is a full-stack e-commerce platform integrating augmented reality technology with online shopping. The system uses a multi-layered architecture, with each component responsible for a specific function.

- **User:** Represents customers and administrators. Managed through Firebase Authentication, with profile information including email, display name, photo URL, and role-based permissions. Users can place multiple orders and have a one-to-many relationship with orders.
- **Product:** Contains information such as name, color variations, pricing, stock levels, and images stored in Cloudinary. Products support various finishes and real-time inventory tracking. Products have a many-to-many relationship with orders through the OrderItem entity.
- **Order:** Central transaction entity, connecting users to purchased products. Contains order status, total amount, shipping details, and payment information. Orders have a one-to-many relationship with OrderItems.
- **AR System:** Built on MediaPipe's face detection and landmark tracking. The ARLipstickDemo component provides the virtual try-on experience, processing video streams and applying lipstick colors to detected lip landmarks.
- **Payment Processing:** Managed through a payment gateway integration, supporting multiple payment methods and maintaining transaction history.
- **Dashboard System:** Provides role-based interfaces for users and administrators. Regular users can manage profiles, view order history, and payment settings. Administrators manage products, orders, users, and analytics.
- **Testing Infrastructure:** Includes unit testing with Jest and end-to-end testing with Cypress, covering all major user workflows.

Class Diagram



EER Diagram



USER_ROLE		
uuid	roleId	PK
string	roleName	
string	description	
string	permissions	
date	createdAt	

USER		
uuid	userId	PK
string	email	
string	displayName	
string	photoURL	
uuid	roleId	FK
string	phone	
string	bio	
boolean	profileCompleted	
string	status	
date	createdAt	

ORDER		
uuid	orderId	PK
uuid	userId	FK
uuid	cartId	FK
number	total	
number	subtotal	
number	vat	
number	deliveryFee	
string	status	
date	createdAt	
object	shippingAddress	
string	deliveryLocation	
string	phoneNumber	
string	paystackRef	

REGISTRATION_CODE		
uuid	codeId	PK
uuid	userId	FK
string	code	
date	expiresAt	
boolean	used	
date	createdAt	

LOGIN_CODE		
uuid	codeId	PK
uuid	userId	FK
string	code	
date	expiresAt	
boolean	used	
date	createdAt	

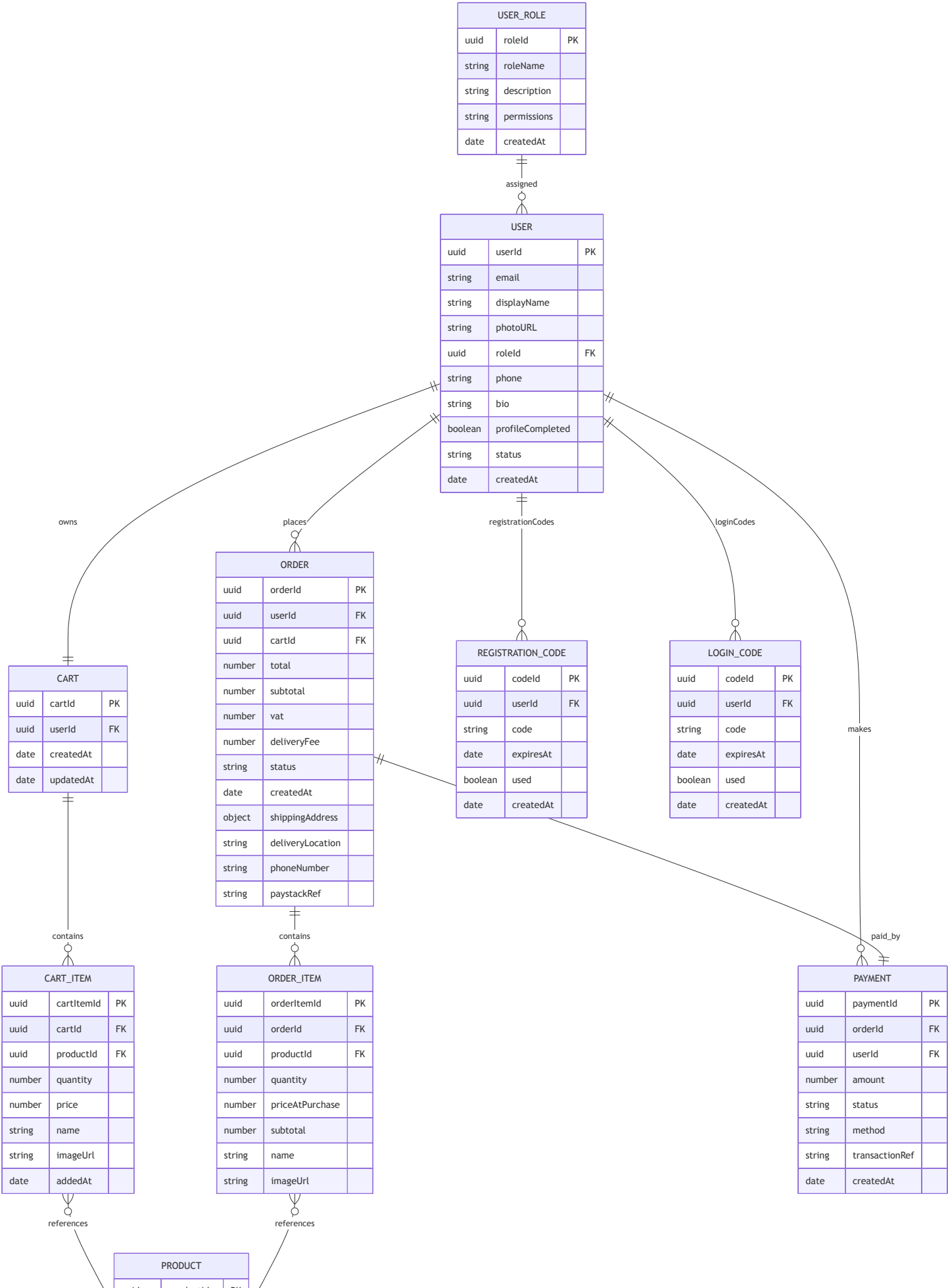
CART		
uuid	cartId	PK
uuid	userId	FK
date	createdAt	
date	updatedAt	

CART_ITEM		
uuid	cartItemId	PK
uuid	cartId	FK
uuid	productId	FK
number	quantity	
number	price	
string	name	
string	imageUrl	
date	addedAt	

ORDER_ITEM		
uuid	orderId	PK
uuid	orderId	FK
uuid	productId	FK
number	quantity	
number	priceAtPurchase	
number	subtotal	
string	name	
string	imageUrl	

PAYMENT		
uuid	paymentId	PK
uuid	orderId	FK
uuid	userId	FK
number	amount	
string	status	
string	method	
string	transactionRef	
date	createdAt	

PRODUCT		
uuid	productId	PK
string	name	
string	description	
string	imageUrl	
number	price	
boolean	isAvailable	
date	createdAt	



uuid	productid	PK
string	name	
string	description	
number	price	
string	imageUrl	
string	category	
number	stock	
string	colorName	
string	hexColor	
string	finish	
number	oldPrice	
date	createdAt	
date	updatedAt	

Entity Tables

USER

```
CREATE TABLE User (
  userId UUID PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  displayName VARCHAR(255) NOT NULL,
  photoURL VARCHAR(500),
  roleId UUID NOT NULL,
  phone VARCHAR(20),
  bio TEXT,
  profileCompleted BOOLEAN DEFAULT FALSE,
  status VARCHAR(20) DEFAULT 'pending',
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Column	Type
userId	UUID (PK)
email	VARCHAR(255)
displayName	VARCHAR(255)
photoURL	VARCHAR(500)
roleId	UUID (FK)
phone	VARCHAR(20)

Column	Type
bio	TEXT
profileCompleted	BOOLEAN
status	VARCHAR(20)
createdAt	TIMESTAMP

USER_ROLE

```
CREATE TABLE UserRole (  
  roleId UUID PRIMARY KEY,  
  roleName VARCHAR(100) NOT NULL,  
  description TEXT,  
  permissions TEXT,  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Column	Type
roleId	UUID (PK)
roleName	VARCHAR(100)
description	TEXT
permissions	TEXT
createdAt	TIMESTAMP

PRODUCT

```
CREATE TABLE Product (  
  productId UUID PRIMARY KEY,  
  name VARCHAR(255) NOT NULL,  
  description TEXT,  
  price NUMERIC(10,2) NOT NULL,  
  imageUrl VARCHAR(500),  
  category VARCHAR(100),  
  stock INTEGER,  
  colorName VARCHAR(100),  
  hexColor VARCHAR(7),  
  finish VARCHAR(50),  
  oldPrice NUMERIC(10,2),  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updatedAt TIMESTAMP  
);
```

Column	Type
productId	UUID (PK)
name	VARCHAR(255)
description	TEXT
price	NUMERIC(10,2)
imageUrl	VARCHAR(500)
category	VARCHAR(100)
stock	INTEGER
colorName	VARCHAR(100)
hexColor	VARCHAR(7)
finish	VARCHAR(50)
oldPrice	NUMERIC(10,2)
createdAt	TIMESTAMP
updatedAt	TIMESTAMP

CART

```
CREATE TABLE Cart (  
  cartId UUID PRIMARY KEY,  
  userId UUID NOT NULL,  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updatedAt TIMESTAMP  
);
```

Column	Type
cartId	UUID (PK)
userId	UUID (FK)
createdAt	TIMESTAMP
updatedAt	TIMESTAMP

CART_ITEM

```
CREATE TABLE CartItem (  
  cartItemId UUID PRIMARY KEY,  
  cartId UUID NOT NULL,  
  productId UUID NOT NULL,  
  quantity INTEGER NOT NULL,  
  price NUMERIC(10,2),  
  name VARCHAR(255),  
  imageUrl VARCHAR(500),  
  addedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Column	Type
cartItemId	UUID (PK)
cartId	UUID (FK)
productId	UUID (FK)
quantity	INTEGER
price	NUMERIC(10,2)

Column	Type
name	VARCHAR(255)
imageUrl	VARCHAR(500)
addedAt	TIMESTAMP

ORDER

```
CREATE TABLE Order (  
  orderId UUID PRIMARY KEY,  
  userId UUID NOT NULL,  
  cartId UUID,  
  total NUMERIC(10,2),  
  subtotal NUMERIC(10,2),  
  vat NUMERIC(10,2),  
  deliveryFee NUMERIC(10,2),  
  status VARCHAR(50),  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  shippingAddress JSONB,  
  deliveryLocation VARCHAR(255),  
  phoneNumber VARCHAR(20),  
  paystackRef VARCHAR(100)  
);
```

Column	Type
orderId	UUID (PK)
userId	UUID (FK)
cartId	UUID (FK)
total	NUMERIC(10,2)
subtotal	NUMERIC(10,2)
vat	NUMERIC(10,2)
deliveryFee	NUMERIC(10,2)
status	VARCHAR(50)

Column	Type
createdAt	TIMESTAMP
shippingAddress	JSONB
deliveryLocation	VARCHAR(255)
phoneNumber	VARCHAR(20)
paystackRef	VARCHAR(100)

ORDER_ITEM

```
CREATE TABLE OrderItem (  
  orderItemId UUID PRIMARY KEY,  
  orderId UUID NOT NULL,  
  productId UUID NOT NULL,  
  quantity INTEGER NOT NULL,  
  priceAtPurchase NUMERIC(10,2),  
  subtotal NUMERIC(10,2),  
  name VARCHAR(255),  
  imageUrl VARCHAR(500)  
);
```

Column	Type
orderItemId	UUID (PK)
orderId	UUID (FK)
productId	UUID (FK)
quantity	INTEGER
priceAtPurchase	NUMERIC(10,2)
subtotal	NUMERIC(10,2)
name	VARCHAR(255)
imageUrl	VARCHAR(500)

PAYMENT

```
CREATE TABLE Payment (  
  paymentId UUID PRIMARY KEY,  
  orderId UUID NOT NULL,  
  userId UUID NOT NULL,  
  amount NUMERIC(10,2),  
  status VARCHAR(50),  
  method VARCHAR(50),  
  transactionRef VARCHAR(100),  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Column	Type
paymentId	UUID (PK)
orderId	UUID (FK)
userId	UUID (FK)
amount	NUMERIC(10,2)
status	VARCHAR(50)
method	VARCHAR(50)
transactionRef	VARCHAR(100)
createdAt	TIMESTAMP

REGISTRATION_CODE

```
CREATE TABLE RegistrationCode (  
  codeId UUID PRIMARY KEY,  
  userId UUID NOT NULL,  
  code VARCHAR(10) NOT NULL,  
  expiresAt TIMESTAMP NOT NULL,  
  used BOOLEAN DEFAULT FALSE,  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Column	Type
codeId	UUID (PK)
userId	UUID (FK)
code	VARCHAR(10)
expiresAt	TIMESTAMP
used	BOOLEAN
createdAt	TIMESTAMP

LOGIN_CODE

```
CREATE TABLE LoginCode (  
  codeId UUID PRIMARY KEY,  
  userId UUID NOT NULL,  
  code VARCHAR(10) NOT NULL,  
  expiresAt TIMESTAMP NOT NULL,  
  used BOOLEAN DEFAULT FALSE,  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Column	Type
codeId	UUID (PK)
userId	UUID (FK)
code	VARCHAR(10)
expiresAt	TIMESTAMP
used	BOOLEAN
createdAt	TIMESTAMP