📘 REQUIREMENTS SPECIFICATION DOCUMENT

Blockchain-Enabled E-Commerce Platform for Artisans

---

1. PROJECT OVERVIEW

You are building a **blockchain-based marketplace** to increase:

- **Buyer trust**
- **Product authenticity**
- **Artisan visibility**
- **Traceability** of artwork ownership and condition

You already have:

- A **Solidity smart contract (Marketplace)** for listing / buying / transferring items
- A **React + Foundry + Anvil local setup** that deploys and connects a frontend to the contract
- A basic prototype that supports item listing, buying, and transfer

This requirements document describes how to scale it into a **production-ready platform** with:

- Full authentication (seller / customer / admin)
- Dashboards for different users
- Artwork authenticity + QR code verification
- Ownership transfer timelines
- Reselling module
- Integrated improved blockchain version
- All modern e-commerce features

---

2. SYSTEM ARCHITECTURE (HIGH-LEVEL)

2.1 Components

1. **Frontend (Web App / PWA)**
   - Tech: React (TypeScript recommended), TailwindCSS, Redux/React Query

- UI: Art-focused e-commerce interface
- Connects to blockchain via **ethers.js**

2. **Backend (Application Server)**

   Handles:
   - Authentication (JWT based)
   - User roles (seller, customer, admin)
   - API + business logic
   - Database interactions
   - QR code generation
   - Condition report storage **Tech options:**
   - Node.js (Express / NestJS) – best for integration with your current JS stack
   - Python (FastAPI) – if ML/analytics needed later

3. **Blockchain Layer**
   - Existing Marketplace contract (extend & improve)
   - Maintains:
     - Artwork ownership history
     - Transaction events
     - Authenticity hashes
   - Tools: Foundry (forge, anvil, cast) for dev & testing
   - Network:
     - Start with testnet (Polygon Amoy / Sepolia)
     - Final deploy → Polygon / BNB / Scroll (low gas)

4. **Database (Off-chain storage)**

   Required for:

   - User accounts
   - Art metadata
   - Images / IPFS hash references
   - Condition reports
   - QR scan logs

   Recommended DB: PostgreSQL

   (Plus Redis for caching)

5. **IPFS / Decentralized Storage**
   - Artwork images
   - Certificate of authenticity

- Metadata JSON

6. **QR Verification Service**
   - Generates QR with unique ID + IPFS/chain info
   - When scanned → displays authenticity, ownership, condition report

---

3. USER ROLES & FEATURE REQUIREMENTS

3.1 USERS

A) Artisan / Seller

Can:

- Create an account
- Setup artisan profile
- Upload artwork (image + description + price)
- Automatically mint authenticity certificate
- Print packaging QR
- View sales & earnings
- Transfer artwork condition notes
- Resell returned artwork

B) Customer / Buyer

Can:

- Create profile
- Browse, search, filter artworks
- Purchase using integrated payment options (on-chain)
- View past purchases
- View artwork authenticity details (on-chain data)
- Resell artwork (secondary market)

C) Admin

Can:

- Approve artisan accounts
- Verify reported disputes
- Manage categories

- View all blockchain transactions
- Remove fraudulent listings
- Monitor QR scan logs

---

4. CORE FEATURES (DETAILED)

4.1 Authentication & User Profiles

Requirements

- JWT authentication (access + refresh tokens)
- 3 roles: **admin, seller, customer**
- Social login optional (Google OAuth)
- Profile creation:
  - Name, bio
  - Govt ID (seller verification)
  - Wallet address (auto-linked via MetaMask)

Tools:

- Backend: NestJS Auth module
- DB: PostgreSQL
- Wallet Auth: Sign message with MetaMask

4.2 User Dashboards

Seller Dashboard

- Add new artwork
- Inventory management
- Sales analytics
- Blockchain transaction logs
- Manage QR codes
- Condition report update UI

Customer Dashboard

- Purchase history
- Ownership certificates (blockchain)
- Resell button
- Support & dispute section

Admin Dashboard

- Users list
- Approve artisans
- Ban sellers
- Dispute handling
- Blockchain explorer integrated UI

4.3 Improved Blockchain Layer

You already use the **Marketplace.sol** contract (list → buy → transfer) .

Enhance with the following:

1. **NFT-based artwork identity (optional)**
   - Each artwork = NFT representing digital certificate
2. **Ownership Timeline**

   Store on-chain:

- List of owners
- Timestamps
- Transaction IDs

Smart contract additions:

```
struct OwnershipRecord {
    address owner;
    uint256 timestamp;
    string conditionHash;  // IPFS hash of condition report
}
```

3. **Condition Reports**

Stored off-chain → IPFS

Hash stored on-chain → ensures integrity.

4. **Reselling Module**

   Add smart contract function:

- resellItem(uint itemId, uint newPrice)
- Should update:
  - isResell flag
  - previous owner

5. **QR Code Authenticity**

   Each artwork gets:

- artId
- IPFS metadata hash
- Blockchain proof hash
- QR → URL: yourdomain.com/verify/:artId

Scan result shows:

- Artwork image
- Original artist
- Ownership chain
- Condition timeline
- Checksum verification

4.4 QR Code Verification System

Workflow

1. When seller lists an artwork → system generates:
   - unique artwork ID
   - blockchain reference
   - IPFS metadata
   - QR code containing verify URL (or metadata hash)
2. Customer scans QR:
   - Request hits backend API

- System fetches:
  - On-chain item info
  - IPFS metadata
  - Ownership history
- Shows authenticity page

3. Admin can view:
   - How many times QR was scanned
   - If fake copies detected

Tools

- qrcode npm library
- Next.js API routes
- IPFS Pinning: **Web3.Storage / Pinata**

4.5 Artwork Reselling Feature

Features:

- Owner can re-list purchased item
- New price set by owner
- Smart contract handles:
  - Escrow
  - Transfer of ownership
- UI displays “Pre-Owned • Verified Authentic” tag
- Admin monitors suspicious resells

4.6 Ownership Timeline + Condition Reporting

Condition report includes:

- Current condition (excellent/good/slightly damaged)
- Image upload
- Date & location

Stored as IPFS JSON:

```
{
  "owner": "0x123..",
  "date": "2025-03-10",
  "condition": "Good",
  "comments": "No scratches",
  "photos": ["ipfs://.../photo1"]
}
```

On-chain:

- store only IPFS hash

UI:

- Timeline view (vertical chronological list)

4.7 Additional Necessary Features

Must-have e-commerce features:

- Category & tags
- Cart system
- Payment gateway (Stripe / Razorpay) for off-chain payments
- Email/SMS notifications
- Reviews & ratings
- Wishlist
- Search + filters
- Product recommendations
- Secure admin panel

Security:

- Role-based access control
- Input validation
- Rate limiting
- Smart contract audits
- QR spoof protection

Scalability:

- Deploy backend on AWS/GCP
- Use CDN for frontend
- Use TheGraph for fast blockchain indexing (optional)

---

5. REQUIRED TECHNOLOGIES

5.1 Blockchain

- Solidity (smart contracts)
- Foundry (forge + anvil) — already in use
- ethers.js (frontend interaction)
- IPFS (image/metadata)
- TheGraph (optional indexing)

5.2 Backend

Recommended: NestJS

Equivalent options:

- Express.js
- FastAPI

Modules:

- Auth
- Artwork
- Ownership
- QR Verification
- Admin

5.3 Frontend

- React + TypeScript
- TailwindCSS
- Redux Toolkit or React Query
- MetaMask integration
- Responsive UI (mobile & desktop)

5.4 Database

- PostgreSQL
- Redis

5.5 Deployment

- Frontend → Vercel
- Backend → AWS EC2 / Render
- IPFS → Web3.Storage or Pinata
- Smart contracts → Polygon/BNB chain

---

6. IMPLEMENTATION ROADMAP (STEP BY STEP)

Phase 1 — Backend Foundation

1. Create NestJS project
2. Build:
   - Auth module
   - User module
   - Artisan verification flow
3. Setup PostgreSQL
4. Integrate MetaMask login (wallet signature)

Phase 2 — Blockchain Upgrade

1. Extend Marketplace.sol:
   - Ownership timeline
   - Condition reports
   - Reselling
2. Write tests using Foundry
3. Deploy to testnet
4. Integrate updated ABI in frontend and backend

Phase 3 — Frontend Application

1. Build multi-role login
2. Seller dashboard
3. Customer dashboard
4. Admin dashboard
5. UI for:
   - Add artwork
   - Resell
   - Transfer
   - Condition reports
   - Ownership timeline
6. Implement responsive art-themed UI

Phase 4 — QR Verification Module

1. Generate QR after listing
2. Print/export QR for packaging
3. Create verify/:artId route
4. Connect on-chain + IPFS
5. Display authenticity check page

Phase 5 — E-Commerce Features

- Cart
- Wishlist
- Reviews
- Notifications
- Payments integration

Phase 6 — Testing & Deployment

1. Unit tests (Foundry, Jest)
2. Load testing

3. Security audits
4. Deploy to:
    - Frontend → Vercel
    - Backend → AWS
    - Contracts → Polygon/BNB

---

7. FUTURE SCOPE

- Mobile app (React Native)
- AI-based artwork recommendation engine
- Computer vision for condition verification
- DAO for artisan governance
- Token-based reward system

---

8. FINAL OUTPUT

This document gives you:

- All system requirements
- Architecture plan
- Feature list
- Implementation plan
- Tool stack compatible with your existing Foundry + React setup

I can also create:

✅ Architecture Diagram

✅ ER Diagram

✅ Detailed Smart Contract Specification

✅ User Flow Diagram

✅ Project Timeline Gantt Chart

Just tell me!

Would you like me to turn this into a **full formal Software Requirements Specification (SRS) PDF** or **project proposal PPT** for submission?