**Project Title**: ArtisanMarket – Multi-Vendor E-Commerce Platform  
**Client**: ArtisanMarket Ltd (Fictional Client for Portfolio)  
**Developer**: NovaEdge  
**Version**: 1.0  
**Date**: July 9, 2025

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the functional and non-functional requirements, technical architecture, and design constraints for the ArtisanMarket E-Commerce Platform. The project aims to provide a scalable, responsive, and secure multi-vendor marketplace connecting artisans to global customers.

**1.2 Scope**

The application enables artisans to register, manage products, and sell through an international e-commerce platform. It supports real-time inventory tracking, multi-currency transactions, advanced filtering, user personalization, vendor analytics, and secure payment/shipping solutions.

**2. Overall Description**

**2.1 Product Perspective**

ArtisanMarket is a web-first platform with mobile-friendly capabilities (PWA), built using microservices and modern frontend/backend technologies. It functions as a **multi-vendor SaaS e-commerce solution**.

**2.2 User Classes and Characteristics**

| **User Type** | **Description** | **Access Level** |
| --- | --- | --- |
| Customer | Shops for artisan products | Public access |
| Vendor | Registers, manages products, sees sales | Authenticated dashboard |
| Admin | Manages site operations and vendors | Admin dashboard |
| Guest | Browses products, adds to wishlist | Limited access |

**3. Functional Requirements**

**3.1 Customer Features**

* Browse/search products by category, tag, or visual search
* Filter by price, rating, vendor, location
* Product details with zoom, attributes, and reviews
* Add to cart, wishlist, or compare
* Multi-currency support
* Checkout with Stripe, PayPal
* Track orders and shipping status

**3.2 Vendor Features**

* Sign up and set up store
* Add/edit/delete products with variants
* Upload product media
* Dashboard with revenue & analytics
* View customer reviews
* Commission calculation reports
* Inventory sync and low stock alerts

**3.3 Admin Features**

* Approve/reject vendors
* Monitor sales and product listings
* Access site-wide metrics and reports
* Configure commission percentages
* Manage disputes or reports

**3.4 System Features**

* Email notifications (order, delivery, etc.)
* PWA support and offline product browsing
* Auto currency conversion (based on location/IP)
* Push notifications for mobile users

**4. Non-Functional Requirements**

**4.1 Performance**

* Global CDN for static assets
* Page load under 2 seconds globally
* Auto-scaling infrastructure (Kubernetes)

**4.2 Security**

* HTTPS everywhere
* Secure API with rate limiting
* Stripe/PayPal PCI-compliant integration
* Image/media sanitization on upload
* Role-based access control (RBAC)

**4.3 Usability**

* Accessible via WCAG 2.1 standards
* Mobile-first UX (PWA enabled)
* Intuitive dashboard for vendors and customers

**4.4 Availability**

* 99.9% uptime via AWS
* Load balancing with auto-failover

**5. Technology Stack**

**5.1 Frontend**

* **React.js + TypeScript**
* **Redux Toolkit** for app state
* **React Query** for data-fetching layer
* **Tailwind CSS** for styling
* **Framer Motion** for animations
* **PWA** features enabled (manifest, service worker)

**5.2 Backend**

* **Node.js + Express.js**
* **MongoDB** (NoSQL) for dynamic data like products, orders, users
* **Redis** for session management and caching
* **Stripe & PayPal** SDKs for payment processing
* **Nodemailer** (or SendGrid) for transactional emails

**5.3 Infrastructure**

* **Docker** for containerization
* **Kubernetes** for service orchestration
* **AWS S3** for image uploads and delivery (via CDN)
* **Cloudflare** for DNS/CDN and WAF security
* **ElasticSearch** for blazing-fast search and filtering
* **Nginx** as reverse proxy/load balancer

**6. Design & Branding**

**6.1 Visual Style**

| **Element** | **Choice** |
| --- | --- |
| Color Palette | Earthy Tones: Rust, Deep Brown, Soft Cream |
| Typography | Poppins or DM Sans (modern, artisan-friendly) |
| Layout | Card-based grid system, max width 1280px, Tailwind responsive breakpoints |
| Logo | 👜 Shaped rust-orange bag + dark navy type (generated earlier) |

**6.2 User Flow Highlights**

* Fast loading → Search & filter → Product view → Cart → Payment → Track Order
* Vendors flow: Signup → Setup shop → Upload product → Manage orders

**7. Database Schema (Simplified)**

**🛒 Product Collection**

json

CopyEdit

{

"id": "uuid",

"title": "Handcrafted Mug",

"vendorId": "uuid",

"categories": ["ceramics", "kitchen"],

"price": 49.99,

"currency": "USD",

"inventory": 120,

"images": ["s3url1", "s3url2"],

"attributes": {"color": "blue", "size": "medium"},

"createdAt": "timestamp"

}

**👤 Vendor Collection**

json

CopyEdit

{

"id": "uuid",

"storeName": "Handmade by Liza",

"email": "liza@example.com",

"approved": true,

"bio": "Nature-inspired artisan pieces.",

"products": ["uuid1", "uuid2"],

"balance": 299.99

}

**8. Deployment & CI/CD**

| **Tool** | **Purpose** |
| --- | --- |
| **GitHub Actions** | CI for lint/test/deploy |
| **Docker Hub** | Image storage |
| **AWS EKS** | Kubernetes deployment |
| **Cloudflare Pages** | CDN and cache for static |

**9. KPIs & Metrics**

| **Metric** | **Target** |
| --- | --- |
| Page load time | < 2s worldwide |
| Uptime | 99.9% |
| Mobile Lighthouse score | > 90 |
| Order processing latency | < 200ms |
| SEO health (Google test) | Green status |

**10. Stretch Goals / Future Enhancements**

* AI-based vendor recommendation for users
* Multilingual UI
* Vendor livestream product demos
* Social logins (Google, Facebook)
* Native apps via React Native
* Loyalty points & coupons system
* **Project Structure Overview**
* bash
* CopyEdit
* artisanmarket/
* ├── client/ # Frontend (React + Tailwind)
* │ ├── public/
* │ └── src/
* │ ├── assets/ # Logos, icons, etc.
* │ ├── components/ # Reusable components
* │ ├── features/ # Redux slices / views (products, cart, auth, etc.)
* │ ├── pages/ # Route-based pages (Home, Shop, Product, etc.)
* │ ├── services/ # API calls (using React Query or Axios)
* │ ├── App.tsx
* │ └── main.tsx
* ├── server/ # Backend (Node + Express + MongoDB)
* │ ├── config/ # DB and env config
* │ ├── controllers/ # Request handlers
* │ ├── models/ # Mongoose schemas
* │ ├── routes/ # Express routes
* │ ├── middleware/ # Auth, error handling, etc.
* │ ├── utils/ # Helpers
* │ ├── server.js # Entry point
* │ └── .env # Env variables
* ├── docker/ # Dockerfiles and compose
* │ ├── Dockerfile.client
* │ ├── Dockerfile.server
* │ └── docker-compose.yml
* ├── README.md
* └── .gitignore
* **🧑‍💻 Step-by-Step Setup**
* **🔹 1. Initialize the project root**
* bash
* CopyEdit
* mkdir artisanmarket && cd artisanmarket
* **🔹 2. Set up the Frontend (client/)**
* bash
* CopyEdit
* npx create-vite@latest client --template react-ts
* cd client
* npm install
* # Tailwind setup
* npm install -D tailwindcss postcss autoprefixer
* npx tailwindcss init -p
* # Additional tools
* npm install react-router-dom react-query axios redux react-redux @reduxjs/toolkit framer-motion
* **tailwind.config.js**
* js
* CopyEdit
* content: ["./index.html", "./src/\*\*/\*.{js,ts,jsx,tsx}"],
* theme: {
* extend: {},
* },
* plugins: [],
* **src/index.css**
* css
* CopyEdit
* @tailwind base;
* @tailwind components;
* @tailwind utilities;
* **🔹 3. Set up the Backend (server/)**
* bash
* CopyEdit
* mkdir server && cd server
* npm init -y
* npm install express mongoose dotenv cors bcryptjs jsonwebtoken nodemailer multer stripe
* npm install --save-dev nodemon
* **server.js**
* js
* CopyEdit
* import express from 'express'
* import cors from 'cors'
* import dotenv from 'dotenv'
* import connectDB from './config/db.js'
* import productRoutes from './routes/productRoutes.js'
* import userRoutes from './routes/userRoutes.js'
* dotenv.config()
* connectDB()
* const app = express()
* app.use(cors())
* app.use(express.json())
* app.use('/api/products', productRoutes)
* app.use('/api/users', userRoutes)
* const PORT = process.env.PORT || 5000
* app.listen(PORT, () => console.log(`Server running on port ${PORT}`))
* **🔹 4. Basic Dockerization (docker/)**
* **docker-compose.yml**
* yaml
* CopyEdit
* version: '3.8'
* services:
* client:
* build:
* context: ./client
* dockerfile: Dockerfile.client
* ports:
* - '3000:3000'
* server:
* build:
* context: ./server
* dockerfile: Dockerfile.server
* ports:
* - '5000:5000'
* environment:
* - MONGO\_URI=mongodb://mongo:27017/artisanmarket
* depends\_on:
* - mongo
* mongo:
* image: mongo
* ports:
* - '27017:27017'
* **Dockerfile.client**
* Dockerfile
* CopyEdit
* FROM node:18
* WORKDIR /app
* COPY . .
* RUN npm install
* RUN npm run build
* CMD ["npm", "run", "dev"]
* **Dockerfile.server**
* Dockerfile
* CopyEdit
* FROM node:18
* WORKDIR /app
* COPY . .
* RUN npm install
* CMD ["npm", "run", "dev"]