# Campus Marketplace — Complete Specification for AI Website Builder

This document is a **machine-consumable specification** that captures all requirements for building a **Campus Marketplace** platform. It combines UI/UX, roles, backend, database, security, marketplace logic, and checkout flow, ensuring no feature is missed.

## 🎨 User Interface & UX

* **Design System**
  + Primary color: Blue #2563eb
  + Secondary colors: Neutral grays
  + Status colors: Green (success), Red (error), Yellow (warning)
  + Font: Inter (or system fallback)
  + Mobile-first responsive design
  + Rounded corners, shadows, clean icons
* **Core Pages**
  + Homepage: Header, search, hero banner, features, popular products, footer
  + Product Catalog: Search, filters (category, price, rating, availability), sort options, grid/list toggle
  + Product Detail: Gallery, info, reviews, Q&A, related products
  + Cart: Multi-shop cart support
  + Checkout: Delivery details, delivery slot selection, payment, order summary
  + Authentication: Login, Register (Customer or Shop Owner), Forgot/Reset password
  + Customer Dashboard: Orders, wishlist, reviews, account settings
  + Shop Owner Dashboard: Inventory, order queue, analytics, settings
  + Admin Console: User management, shop verification, system logs
* **Components**
  + Header with logo, navigation, search, cart count, user profile
  + ProductCard (title, price, image, rating, add-to-cart)
  + FilterSidebar
  + OrderList with status badges
  + InventoryTable with edit/bulk actions
* **Accessibility & SEO**
  + WCAG AA compliance
  + Alt text for all images
  + Server-side rendering for SEO
  + Structured data (Product, Offer, Review)

## 👤 Roles & Authentication

* **Customers**
  + Register/login with college email & ID
  + Browse catalog, add to cart, checkout, manage orders
  + Write reviews, maintain wishlist
* **Shop Owners**
  + Register/login as seller
  + Create/manage shop profile
  + Add/edit products, manage stock
  + View orders for their shop only
  + See customer details for each order (who ordered, contact, delivery address)
* **Admins** (optional)
  + Verify shops
  + Monitor system activity
  + Handle disputes
* **Authentication System**
  + JWT-based (access + refresh tokens)
  + Password hashing with bcrypt (12 salt rounds)
  + Lockout after 5 failed attempts
  + Token rotation & secure storage

## 🏪 Marketplace Logic

* **Global Catalog**
  + Customers see all products from all shops in one unified catalog.
  + Filters allow narrowing by category, price, brand, shop, or rating.
* **Multi-Shop Cart**
  + Customers can add items from multiple shops into one cart.
  + At checkout, the system automatically splits the cart into **one order per shop**.
* **Order Management**
  + Shop Owners only see their shop’s orders.
  + Customers see a unified order history in their dashboard.

## ⚙️ Backend Architecture

* **RESTful API Endpoints**
  + POST /api/auth/register → Register new user (Customer or Shop Owner)
  + POST /api/auth/login → Login
  + POST /api/auth/refresh → Refresh token
  + GET /api/products → List all products (with filters)
  + POST /api/products → Create product (shop owners only)
  + GET /api/products/:id → Get product details
  + POST /api/orders → Create new order (requires customer details)
  + GET /api/orders → List orders (customer = own orders, shop owner = shop’s orders)
  + PUT /api/orders/:id/status → Update order status (shop owner)
  + POST /api/reviews → Create review
* **Error Handling**
  + Generic user-friendly messages
  + Secure error logging for developers
* **Security Middleware**
  + Rate limiting (global + sensitive endpoints)
  + Input validation and sanitization
  + CORS + CSRF protection

## 🗄 Database Schema (PostgreSQL)

* **users**
  + id (UUID, PK)
  + name, email (unique), password\_hash
  + role: enum(customer, shop\_owner, admin)
  + phone, college, verified (bool)
  + timestamps
* **shops**
  + id (UUID, PK)
  + owner\_id → users.id
  + name, address, verified (bool)
  + timestamps
* **products**
  + id (UUID, PK)
  + shop\_id → shops.id
  + title, description, price, stock, category, tags
  + images (JSON)
  + rating\_avg (decimal)
  + timestamps
* **orders**
  + id (UUID, PK)
  + user\_id → users.id (customer)
  + shop\_id → shops.id
  + items (JSON: product\_id, name, quantity, price)
  + subtotal, tax, delivery\_fee, total
  + status: enum(processing, accepted, in\_transit, delivered, cancelled)
  + delivery\_address (JSON: recipient\_name, phone, email, address\_line1, address\_line2, campus\_location, postal\_code, notes)
  + timestamps
* **reviews**
  + id (UUID, PK)
  + product\_id → products.id
  + user\_id → users.id
  + rating (1–5), comment, timestamps
* **auth\_tokens**
  + id, user\_id, refresh\_token\_hash, expires\_at, revoked

## 📂 File Storage

* Provider: S3-compatible or local dev storage
* Buckets:
  + product\_images: max 10MB, jpg/png/webp
  + profile\_avatars: max 5MB
  + documents: shop verification (pdf/jpg)
* Features: virus scanning, thumbnail generation, automatic resizing

## 🔒 Security

* HTTPS enforced (TLS 1.3)
* Secure headers (HSTS, X-Content-Type-Options, etc.)
* CSRF protection
* Session timeout & monitoring
* SQL injection/XSS prevention via parameterized queries & sanitization
* Rate limiting:
  + 120 requests/min global
  + 20 requests/min on auth endpoints

## 📊 Analytics & Monitoring

* Shop Owner Analytics: sales, top products, customer feedback
* Admin/Platform Analytics: revenue trends, user growth
* Monitoring: Prometheus + Grafana
* Logging: access logs, audit logs, sensitive data masking

## 🚀 DevOps & Testing

* **Local Dev**
  + Docker Compose (frontend, backend, db, storage)
* **CI/CD**
  + GitHub Actions/GitLab CI pipeline
  + Lint → Test → Build → Deploy staging → Smoke test → Deploy production
* **Backups**
  + Daily encrypted DB backups, 30-day retention
* **Testing**
  + Unit tests: >80% coverage
  + E2E: Cypress (flows: register, login, purchase, order management)
  + Security testing: SAST + DAST scans

## 🧩 Tech Stack Recommendation

* **Frontend**: React (Next.js), TailwindCSS, React Query/Redux, Cypress
* **Backend**: Node.js (TypeScript, Express/NestJS), Prisma/TypeORM, Jest
* **Database**: PostgreSQL
* **Storage**: S3-compatible
* **Infra**: Docker, Kubernetes, Prometheus, Grafana

## ✅ Order Checkout Flow (Detailed)

1. Customer adds products from multiple shops into one cart.
2. At checkout, customer must provide:
   * Full name
   * Email (auto-filled from account)
   * Phone number
   * Delivery address (address lines, campus location, postal code)
   * Delivery notes (optional)
3. Backend splits the cart into multiple orders, one per shop.
4. Each Shop Owner sees:
   * Buyer’s full details (name, email, phone, address)
   * Products ordered from their shop
   * Delivery info & payment status
5. Customer sees all orders unified in their dashboard.

## 📌 Deliverables for AI Generator

* Full project scaffold:
  + frontend/ with pages, routes, components
  + backend/ with controllers, models, migrations, security middleware
  + infra/ with Docker Compose + k8s manifests
  + seed/ with sample data (1 shop, 1 product, 1 customer)
* OpenAPI docs for API
* CI pipeline config
* README with setup & usage guide