**System Design for Used Clothes Reselling SaaS Platform**

**Overview**

This system design outlines a SaaS platform for reselling used clothes, targeting individual sellers, small thrift businesses, and influencers. It uses a MERN stack for rapid development, scalability, and flexibility, with integrations for payments, authentication, and eco-friendly features to promote sustainability.

**System Architecture**

**1. Frontend**

* **Framework:** React (with Next.js for server-side rendering and SEO)
* **Styling:** TailwindCSS for responsive, mobile-first design
* **Key Components:**
  + User authentication (login/signup)
  + Product listings (image upload, description, tags)
  + Search and filter UI
  + Order management dashboard
  + Seller profile with ratings
  + Admin dashboard for platform oversight
* **Notes:**
  + Mobile-friendly UI optimized for smartphones
  + CDN-hosted React via cdn.jsdelivr.net for fast delivery

**2. Backend**

* **Framework:** Node.js + Express
* **API Structure:** RESTful APIs for CRUD operations
  + /auth - User login, signup, profile management
  + /products - Create, read, update, delete listings
  + /orders - Manage orders, shipping, returns
  + /reviews - Submit and view ratings for sellers/products
  + /admin - Monitor users, listings, disputes
* **Scalability:** Horizontal scaling with load balancer (e.g., AWS ELB or Render)
* **Security:**
  + JWT for session management
  + Input validation and sanitization
  + Rate limiting to prevent abuse

**3. Database**

* **Database:** MongoDB (NoSQL for flexible schema)
* **Collections:**
  + Users: Buyer/seller details (email, role, profile, subscription plan)
  + Products: Listings (images, description, condition, size, price, tags)
  + Orders: Order details (buyer, seller, product, status, shipping)
  + Reviews: Seller/product ratings
  + AdminLogs: Admin actions and disputes
* **Notes:**
  + Indexes on frequently queried fields (e.g., product category, price)
  + Cloud-hosted MongoDB Atlas for scalability

**4. Authentication**

* **Provider:** Firebase Auth
* **Features:**
  + Email/password, Google, and social login (e.g., X OAuth)
  + Role-based access (buyer, seller, admin)
* **Security:** Secure token storage, password hashing

**5. Payment Integration**

* **Provider:** Stripe
* **Features:**
  + Secure payment processing for buyers
  + Payouts to sellers (minus platform fee)
  + Subscription plans (free, premium) for sellers
* **Implementation:**
  + Stripe Checkout for one-time payments
  + Stripe Billing for recurring SaaS subscriptions

**6. Hosting**

* **Frontend:** Vercel (Next.js hosting, automatic scaling)
* **Backend:** Render (Node.js hosting, managed scaling)
* **Database:** MongoDB Atlas (cloud-hosted, auto-sharding)
* **CDN:** Cloudflare for static assets and caching

**7. Additional Features**

* **Eco Badge System:**
  + Sellers earn badges for sustainable practices (e.g., recycled packaging)
  + Displayed on product listings to encourage eco-conscious buying
* **Search & Filters:**
  + Elasticsearch or MongoDB text search for fast querying
  + Filters for size, category, condition, price, brand
* **Analytics:**
  + Seller dashboard with sales, views, and trends (using Chart.js or similar)
  + Admin analytics for platform-wide metrics

**MVP Features**

To launch quickly, focus on these core features:

1. User signup/login (buyer/seller roles)
2. Product listing creation (image upload, description, tags)
3. Basic search and filter by category/price
4. Stripe payment integration for purchases
5. Order management for sellers
6. Simple review system
7. Admin dashboard for monitoring listings
8. Eco badge display on listings

**System Flow**

1. **User Journey:**
   * Seller signs up, creates profile, and lists items
   * Buyer browses, filters, and purchases items
   * Seller receives order, ships, and tracks
   * Buyer leaves review/rating
   * Admin monitors disputes or fraudulent listings
2. **API Flow Example (Product Listing):**
   * POST /products → Upload images to cloud storage (e.g., AWS S3), save metadata to MongoDB
   * GET /products?category=men&price\_max=50 → Fetch filtered listings
   * POST /orders → Process payment via Stripe, create order record

**Scalability Considerations**

* **Horizontal Scaling:** Add more Node.js instances behind a load balancer
* **Database Sharding:** MongoDB Atlas supports sharding for large datasets
* **Caching:** Redis for frequently accessed data (e.g., product listings)
* **CDN:** Cloudflare for static assets (images, CSS, JS)

**Security Considerations**

* HTTPS for all communications
* Input sanitization to prevent XSS/SQL injection
* Rate limiting and CAPTCHA for signup/login
* Secure Stripe API keys and Firebase credentials

**Development Roadmap**

1. **Week 1-2:** Set up GitHub repo, initialize Next.js and Express projects, configure Firebase Auth
2. **Week 3-4:** Build product listing and search APIs, integrate MongoDB
3. **Week 5-6:** Implement Stripe payments and order management
4. **Week 7-8:** Develop frontend UI (React + TailwindCSS), test mobile responsiveness
5. **Week 9:** Add review system and eco badge logic
6. **Week 10:** Build admin dashboard, deploy to Vercel/Render
7. **Week 11-12:** Test, bug fix, and launch MVP

**Sample File Structure**

/project

├── /client (Next.js)

│ ├── /components (React components: ProductCard, SearchBar, etc.)

│ ├── /pages (Next.js pages: home, profile, listings)

│ ├── /styles (TailwindCSS config)

│ └── /public (static assets)

├── /server (Node.js + Express)

│ ├── /routes (API routes: auth, products, orders)

│ ├── /models (MongoDB schemas)

│ ├── /controllers (business logic)

│ └── /middleware (auth, validation)

├── /docs (system design, API specs)

└── README.md

**Next Steps**

* Create low-fidelity wireframes (e.g., using Figma or Excalidraw)
* Set up GitHub repo and initialize projects
* Start with authentication and product listing APIs
* Iterate based on user feedback post-MVP