

Greeting exchange



Hello

Hello! How can I help you today?



So i am making my semester Final Project. The Theme is : E-commerce. I got an cool idea about an Thrift Store for our college. But i need help to build it so first we need to plan everything and then step by step implement it. I need to do it quickly.

Here are some Info:

I am planning to use Node.js Express.js MongoDB, React, JWT Auth, bcrypt, VSCode copilot, AI Autocompletions Vite for front-end. An organized file structure.

The course of this project is : Software Engineering 101 , Basically learning good software development practices

Also i don't want to make too much complicated like production level its for demonstration and learning.

So make some concise planning for this.

Awesome idea—perfect for SE-101! Here's a tight, do-able plan to ship a functional "College Thrift Store" MERN app fast while showing good practices.

1) Scope & Goals (MVP-first)

Core flow (demo-ready):

- Browse items → search/filter → item detail
- Auth (JWT): student signup/login, protected routes
- Post a listing (title, photos, price, condition, category)
- Chatless transaction: "Request to buy" → seller gets request → mark as "Reserved/Sold"
- Basic admin actions (seed data, view users/items, ban user)

Stretch (if time):

- Favorites (wishlist)
- In-app messages

- Simple ratings after sale
- Image hosting (Cloudinary) vs local

2) Roles & User Stories

- **Guest:** view items, search, see seller profile; sign up.
- **Student (auth):** post/edit/delete own items, request to buy, see requests, mark sold.
- **Admin:** view/remove abusive items/users.

Examples (acceptance criteria)

- As a Student, I can create a listing with ≥ 1 image, required fields, and see it in the feed within 2s.
- As a Seller, I can mark my item "Reserved" then "Sold" with a single click.
- As a Guest, I can search by title/category and see ≤ 24 results per page.

3) Architecture (MERN)

Frontend (React + Vite):

- Routes: / (feed), /item/:id, /sell, /login, /signup, /profile, /admin
- State: React Query (or lightweight fetch hooks) for items/auth
- UI: simple component lib or Tailwind (fast)

Backend (Node + Express):

- REST API, JSON
- Auth: JWT (httpOnly cookie or Authorization header)
- Passwords: bcrypt
- Validation: zod or express-validator
- File storage:
 - MVP: local /uploads
 - Stretch: Cloudinary

DB (MongoDB):

- Collections: users, items, buyRequests

4) Data Models (concise)

User

```
js

{
  _id, name, email (unique), passHash, role: 'student' | 'admin',
  avatarUrl, createdAt
}
```

Item

```
js

{ _id, title, description, price, category, condition, images: [url],
  sellerId, status: 'available' | 'reserved' | 'sold',
  campus: 'Main', createdAt }
```

BuyRequest

```
js

{ _id, itemId, buyerId, status: 'pending' | 'accepted' | 'rejected',
  createdAt }
```

5) API (minimal)

- Auth
 - POST /api/auth/signup {name, email, password}
 - POST /api/auth/login {email, password} → JWT
 - GET /api/auth/me (protected)

- Items
 - GET /api/items?search=&category=&status=&page=&limit=
 - GET /api/items/:id
 - POST /api/items (auth; form-data for images)
 - PATCH /api/items/:id (owner)
 - PATCH /api/items/:id/status (owner) {status}
 - DELETE /api/items/:id (owner/admin)

- Buy Requests
 - POST /api/items/:id/requests (auth)
 - GET /api/requests/mine (auth; as buyer)
 - GET /api/requests/received (auth; as seller)
 - PATCH /api/requests/:id (seller) {status}

- Admin
 - GET /api/admin/users (admin)
 - DELETE /api/admin/users/:id (admin)

6) Frontend Pages & Components

- **Pages:** Feed, ItemDetail, SellForm, Login, Signup, Profile (my items + requests), Admin
- **Components:** ItemCard, SearchBar, CategoryFilter, Pagination, ProtectedRoute, ImageUploader, StatusBadge, Toast

- **State/data:**

- `useAuth()` (login/logout/me)
- `useItems()` (list/search/detail/mutate)
- `useRequests()` (mine/received/update)

7) File Structure (organized)

```

lua

thrift-store/
  server/
    src/
      index.js
      config/ (env, db.js)
      middleware/ (auth.js, error.js, upload.js)
      models/ (User.js, Item.js, BuyRequest.js)
      routes/ (auth.js, items.js, requests.js, admin.js)
      controllers/ (...)
      utils/ (jwt.js, validators.js)
      uploads/ (gitignored)
    package.json
  client/
    src/
      main.jsx
      App.jsx
      routes/
      components/
      pages/
      api/ (auth.js, items.js, requests.js)
      hooks/ (useAuth.js, useQuery hooks)
      styles/
      index.html
    package.json
README.md
.env.example

```

8) Implementation Order (fastest path)

1. **Repo & tooling:** init Git, ESLint/Prettier, basic README.
2. **Server base:** express, Mongo connect, error handler, health route.
3. **Auth:** signup/login/me with bcrypt + JWT; protect middleware.
4. **Items:** Item model + CRUD (no images yet), search/filter/pagination.
5. **Images:** add multer upload → store local /uploads → serve static.
6. **BuyRequests:** create/list/update; seller-only actions.
7. **Frontend skeleton:** Vite + routing; global layout; auth pages working.
8. **Feed & Detail:** list, search, filter, item detail.
9. **Sell form:** create item with image upload.
10. **My Profile:** my items, my requests, received requests; status updates.
11. **Admin (basic):** list users, delete user.
12. **Polish:** empty states, loading, toasts, 404, basic mobile responsiveness.
13. **Seed & demo data:** script to create users/items for demo.

14. (Optional) Deploy: Render/Railway (server) + Vercel/Netlify (client).

9) Commands (quick start)

Backend

```
bash

mkdir -p thrift-store/server && cd $_
npm init -y
npm i express mongoose cors dotenv bcrypt jsonwebtoken multer morgan cookie-parser
npm i -D nodemon
```

Scripts (server/package.json)

```
json

"scripts": { "dev": "nodemon src/index.js" }
```

Frontend

```
bash

cd ../../ && mkdir -p thrift-store/client && cd thrift-store/client
npm create vite@latest . --template react
npm i axios react-router-dom @tanstack/react-query
npm i -D eslint prettier
```

Run

```
bash

# terminal 1
cd thrift-store/server && npm run dev
# terminal 2
cd thrift-store/client && npm run dev
```

10) Security & Quality (SE-101 checkboxes)

- Hash passwords (bcrypt)
- JWT expiry + refresh (simple re-login acceptable for MVP)
- Validate inputs on server (zod/express-validator)
- Authorization checks (owner/admin)
- Rate limit auth routes (optional)
- Central error handler + consistent error shape
- .env for secrets; never commit
- Seed script + sample users (admin/student)
- Basic unit tests for utils (jwt, validators) and one controller
- API docs (simple README table or Swagger later)

11) Minimal Seed Script (outline)

- Create admin (`admin@college.edu / Admin123!`), two students, ~12 items across categories with placeholder images.

12) Demo Script (5 minutes)

1. Guest browsing + search → click item.
 2. Sign up/login → create listing with image.
 3. From a second test account, request to buy.
 4. Switch back to seller → accept → mark item "Sold."
 5. Show admin page removing a spam item (pretend).
-

If you want, I can generate:

- server `index.js` + auth/item routes and models,
- client routing skeleton with Auth context/hooks,
- a tiny seed script and `.env.example`.

Say the word and I'll drop the starter code so you can run it immediately.