Review and Recommended Improvements to Your Current Frontend Architecture

Your current Next.js frontend structure is a solid starting point for a multi-tenant e-commerce platform, with clear separation between routes (in `app/`), reusable components (in `components/`), and utilities/contexts (in `lib/`). It aligns well with basic Next.js conventions, using the App Router for pages like `/admin`, `/checkout`, and `/restaurant`, and leveraging Tailwind CSS for styling. However, for scalability in a multi-tenant setup (where restaurants act as tenants with isolated menus, orders, and dashboards), it could benefit from refinements: adopting a more modular, feature-based organization to reduce coupling, incorporating middleware for tenant isolation (e.g., subdomains like `restaurant1.yourapp.com`), and adding directories for types, hooks, and internationalization if needed. This draws from 2025 best practices for large-scale Next.js apps, emphasizing maintainability, reusability, and performance in e-commerce/SaaS contexts. wisp.blog +5 more

Below, I'll review your structure's strengths and gaps, then propose an optimized version with step-by-step migration plan. This setup supports your multi-tenant needs (e.g., restaurant-specific storefronts via dynamic routes or subdomains), integrates with your backend (Express/Prisma via API calls in `lib/`), and scales for features like carts, wishlists, and role-based dashboards.

Strengths of Your Current Structure

- **Route Organization**: `app/` subfolders (e.g., `admin/`, `restaurant/`, `wishlist/`) act as

route groups, enabling clean URLs like `/restaurant` or `/admin`. This is efficient for role-based access (customers, restaurants, admins).

- **Component Separation**: `components/ui/` groups reusable UI like `ProductCard.tsx` and `RestaurantCard.tsx`, while `components/auth/` isolates auth-related elements. Good for e-commerce reusability.
- **Utility Layer**: `lib/` centralizes contexts (e.g., `authContext.tsx`, `cartContext.tsx`) and helpers (e.g., `api.ts`, `backend.ts`), making state sharing and backend fetches straightforward.
- **Config Files**: Standard placement of `tailwind.config.ts`, `tsconfig.json`, etc., ensures easy maintenance.
- **Overall Simplicity**: Suitable for mid-sized apps; avoids over-engineering.

Gaps and Opportunities

- **Lack of Modularity**: Routes and components are somewhat siloed but not fully domain-driven (e.g., no grouping by features like "orders" or "menus"). For multi-tenancy, this could lead to duplication across tenants.
- **No Multi-Tenancy Isolation**: Static routes like `/restaurant` don't inherently support dynamic tenants (e.g., multiple restaurants). Middleware and dynamic segments `[tenant]` are needed for subdomain/path-based isolation.
- **Missing Directories**: No dedicated `types/` for TypeScript models (e.g., `User`, `Order`), `hooks/` for custom hooks (e.g., `useCart`), or `styles/` for global CSS beyond `globals.css`. Also, no `middleware.ts` for tenant/auth checks.
- **State Management**: Contexts in `lib/` work, but for complex e-commerce (e.g., persistent carts across tenants), consider Zustand or Redux in a `store/` folder.
- **Scalability**: As features grow (e.g., payments, tracking), a feature-based structure prevents bloat in `components/` and `lib/`.
- **Testing/Intl**: Empty `data/` and `docs/` could be repurposed; add `tests/` and `locales/` for i18n if global expansion is planned.

Recommended Optimized Folder Structure

Adopt a src/ wrapper (common in 2025 for large apps) to encapsulate code, with a feature-based approach inside. Use dynamic routes for tenants (e.g., `app/[tenant]/` for path-based or middleware for subdomains). This mirrors boilerplates like ixartz/SaaS-Boilerplate and Vercel's Platforms Starter Kit, supporting multi-tenancy via organizations (e.g., with Clerk) and RBAC. github.com +3 more

text Collapse Wrap Copy

```
restaurant-ecommerce-hub/
├── backend/                  # Your existing backend
├── frontend/
│   ├── next/                 # Next.js root (consider renaming to 'app' f
│   │   ├── src/              # New: Wraps all source code for better isol
│   │   │   ├── app/          # Routes with multi-tenant support
│   │   │   │   ├── [tenant]/ # Dynamic: For restaurant-specific routes (
│   │   │   │   │   ├── menu/
│   │   │   │   │   ├── orders/
│   │   │   │   │   ├── layout.tsx
│   │   │   │   │   └── page.tsx
│   │   │   │   ├── (auth)/   # Group: Auth routes (e.g., /login, /regist
│   │   │   │   │   ├── login/
│   │   │   │   │   └── register/
│   │   │   │   ├── (dashboard)/ # Group: Role-based dashboards
│   │   │   │   │   ├── admin/
│   │   │   │   │   ├── customer/
│   │   │   │   │   ├── restaurant/
│   │   │   │   │   └── layout.tsx
│   │   │   │   ├── api/      # Server Actions or API routes (e.g., for b
│   │   │   │   ├── checkout/
│   │   │   │   ├── faq/
│   │   │   │   ├── track-order/
│   │   │   │   ├── wishlist/
│   │   │   │   ├── globals.css
│   │   │   │   ├── layout.tsx
│   │   │   │   ├── page.tsx  # Homepage (e.g., browse all restaurants)
│   │   │   │   └── middleware.ts # New: Handles subdomains, tenant ID ex
│   │   │   ├── components/   # Reusable UI
│   │   │   │   ├── ui/       # Primitives (e.g., Button.tsx from shadcn/
│   │   │   │   ├── layout/   # New: Structural (e.g., Header.tsx, Footer
│   │   │   │   └── features/ # New: Domain-specific (e.g., auth/LoginFor
│   │   │   ├── contexts/     # New: All contexts (move from lib/: authCo
```

```
│   │   │   ├── hooks/          # New: Custom hooks (e.g., useAuthCheck.tsx
│   │   │   ├── lib/            # Utilities and integrations
│   │   │   │   ├── api/        # New subfolder: api.ts, backend.ts (backer
│   │   │   │   ├── auth.ts     # New: Auth utils (e.g., token handling)
│   │   │   │   └── utils.ts    # General helpers
│   │   │   ├── models/         # New: TypeScript types (e.g., user.ts, ord
│   │   │   ├── styles/         # New: Theme/variables (e.g., variables.css
│   │   │   ├── tests/          # New: Unit/E2E tests (e.g., components/Pro
│   │   │   └── locales/        # Optional: i18n files if multi-language
│   │   ├── public/             # Static assets (e.g., favicon.ico)
│   │   ├── .env
│   │   ├── .gitignore
│   │   ├── eslint.config.mjs
│   │   ├── next-env.d.ts
│   │   ├── next.config.ts
│   │   ├── package.json
│   │   ├── postcss.config.mjs
│   │   ├── README.md
│   │   ├── tailwind.config.ts
│   │   ├── tsconfig.json
│   │   └── vercel.json
│   └── ...                     # Other frontend files if any
└── ...                         # Root configs if shared
```

Key Changes and Rationale

- **src/ Wrapper**: Encapsulates code, aligning with enterprise standards for separation from configs

configs. wisp.blog

- **Multi-Tenancy in app/**: Use `[tenant]/` for dynamic restaurant routes (e.g., fetch tenant data via Prisma in Server Components). Add `middleware.ts` to parse subdomains (e.g., via `request.headers.get('host')`), set tenant context, and redirect unauth users. This isolates data (e.g., menus per restaurant) and supports Vercel's wildcard domains. vercel.com +2 more
- **Components Refinement**: Subfolders like `ui/`, `layout/`, `features/` prevent monolithic growth. Group e-commerce specifics (e.g., `features/products/ProductCard.tsx`) for easier testing.
- **Contexts/Hooks/Models**: Dedicated folders reduce `lib/` clutter; `models/` ensures type safety for backend DTOs.
- **Data/Docs**: Repurpose `data/` for mock data/seeds; `docs/` for API specs or diagrams.
- **Optional Additions**: If using Zustand/Redux, add `store/`. For payments (e.g., Stripe), add `lib/payments.ts`.

Step-by-Step Migration and Setup Plan

1. **Backup and Setup src/**: Create `src/` and move `app/`, `components/`, `lib/`, etc., inside. Update `tsconfig.json` paths (e.g., `"baseUrl": "./src"`).

Update `tsconfig.json` paths (e.g., `"baseUrl": "./src"`).

2 **Implement Multi-Tenancy**:

- Add `app/middleware.ts`: Extract tenant from subdomain/path, fetch tenant data from backend, and attach to headers/props.
- Convert static routes: E.g., move `restaurant/` to `app/[tenant]/` for dynamic handling.
- Integrate Auth: Use Clerk/Auth0 in `lib/auth.ts` for organizations (restaurants as orgs), with RBAC hooks.

3 **Refactor Components/Lib**:

- Move UI files to `components/features/` (e.g., `WishlistTable.tsx` to `features/wishlist/`).
- Group contexts in `contexts/`; create hooks from utils (e.g., `useAuthCheck.tsx` to `hooks/`).
- Add `models/` with types inferred from Prisma (e.g., via `prisma generate`).

4 **Enhance Data Fetching/Security**:

- Use Server Components in routes for backend fetches (e.g., `fetch('/api/menus', { headers: { tenantId } })`).
- Secure: Add protected wrappers (e.g., extend `ProtectedRoute.tsx`) with role checks from auth context.

5 **Testing and Deployment**:

- Add `tests/` with Vitest/Playwright; test tenant switches.
- Deploy to Vercel: Configure wildcard domains for subdomains.
- Performance: Use Suspense in layouts; audit with Next.js tools.

6 **Timeline**: Start with src/ and middleware (1-2 days), then refactor features iteratively (per domain like "wishlist").

This structure will make your app more robust for multi-tenancy, easier to maintain as it grows, and aligned with industry standards. If you share specific files (e.g.

grows, and aligned with industry standards. If you share specific files (e.g., `authContext.tsx`), I can suggest code snippets.