📦 KandyPack Logistics Platform - Project Overview & Development Guide

🎯 Project Description

KandyPack Logistics Platform is a modern, full-stack logistics and supply chain management system designed to streamline operations for logistics companies. The platform provides comprehensive tools for order management, shipment tracking, inventory control, and business analytics.

🏗️ Current Project Structure

```
kandypack-logistics-platform/
├── .git/                          # Git repository
├── .gitignore                     # Git ignore rules
├── .vercel/                       # Vercel deployment config
├── .vscode/                       # VS Code workspace settings
├── LICENSE                        # MIT License
├── KandyPack.docx.pdf             # Project documentation
├── PROJECT_OVERVIEW.md            # This file - comprehensive guide
├── package.json                   # Root package configuration
├── package-lock.json              # Root dependency lock
├── node_modules/                  # Root dependencies
├── backend/                       # Backend application (Planned)
│   ├── requirements.txt           # Python dependencies
│   ├── app/                       # FastAPI application
│   │   ├── main.py               # API entry point
│   │   ├── auth.py               # Authentication logic
│   │   ├── crud.py               # Database operations
│   │   ├── database.py           # Database configuration
│   │   ├── models.py             # SQLAlchemy models
│   │   ├── routes.py             # API routes
│   │   └── schemas.py            # Pydantic schemas
│   └── database/                  # Database files
│       └── schema.sql            # SQL schema
└── frontend/                      # Frontend application
    └── UI/                        # React Router v7 application
        ├── .dockerignore          # Docker ignore rules
        ├── .gitignore             # Frontend git ignore
        ├── .react-router/         # Generated React Router types
        ├── Dockerfile             # Docker containerization
        ├── README.md              # Frontend documentation
        ├── DASHBOARD_STYLING_README.md  # Custom styling guide
        ├── components.json        # Shadcn/ui component config
        ├── package.json           # Frontend dependencies and scripts
        ├── package-lock.json      # Frontend dependency lock
        ├── react-router.config.ts # React Router configuration
        ├── tsconfig.json          # TypeScript configuration
```

```
├── vite.config.ts        # Vite build configuration
├── node_modules/         # Frontend dependencies
├── public/               # Static assets
│   ├── favicon.ico       # Site favicon
│   ├── loginbg.png       # Login page background
│   └── 8121295.gif       # Loading animation
└── app/                  # Application source code
    ├── app.css           # Global styles (Tailwind CSS v4)
    ├── root.tsx          # Root layout component
    ├── routes.ts         # Route configuration
    ├── lib/              # Utility functions
    │   └── utils.ts      # Common utilities (cn, etc.)
    ├── hooks/            # Custom React hooks
    │   └── useAuth.tsx   # Authentication hook
    ├── contexts/         # React Context providers
    │   └── SidebarContext.tsx  # Sidebar state management
    ├── routes/           # Route components (Protected)
    │   ├── home.tsx      # Landing page
    │   ├── login.tsx     # Authentication page
    │   # Admin Routes
    │   ├── dashboard.tsx # Main dashboard (Protected)
    │   ├── order-management.tsx  # Orders page (Protected)
    │   ├── admin.tsx     # Admin management (Protected)
    │   ├── stores.tsx    # Store management (Protected)
    │   ├── last-mile.tsx # Last mile delivery (Protected)
    │   ├── rosters.tsx   # Roster management (Protected)
    │   ├── reports.tsx   # Reports & Analytics (Protected)
    │   ├── logs.tsx      # Activity logs (Protected)
    │   ├── rail-scheduling.tsx  # Rail scheduling (Protected)
    │   # Customer Routes
    │   ├── customer-home.tsx     # Customer home (Protected)
    │   ├── customer-new-order.tsx    # New order (Protected)
    │   ├── customer-track-order.tsx  # Track orders (Protected)
    │   ├── customer-order-history.tsx # Order history (Protected)
    │   └── customer-help-support.tsx  # Help & support (Protected)
    └── components/       # React components
        ├── ProtectedRoute.tsx    # Route protection wrapper
        ├── ErrorBoundary.tsx     # Error handling
        ├── UserAvatar.tsx        # User profile avatar
        ├── PlaceholderPage.tsx   # Generic placeholder
        ├── ui/                   # Shadcn/ui components
        │   ├── badge.tsx
        │   ├── button.tsx
        │   ├── card.tsx
        │   ├── dropdown-menu.tsx
        │   ├── input.tsx
        │   ├── label.tsx
        │   ├── select.tsx
        │   ├── separator.tsx
        │   ├── table.tsx
        │   └── tabs.tsx
        ├── dashboard/            # Admin dashboard components
        │   ├── Dashboard.tsx     # Main dashboard layout
```

```
│   ├── DashboardLayout.tsx  # Shared layout wrapper
│   ├── DashboardHeader.tsx  # Header with logout
│   ├── DashboardSkeleton.tsx # Loading skeleton
│   ├── Sidebar.tsx       # Navigation sidebar
│   ├── StatsCard.tsx     # Metric cards
│   ├── RecentOrders.tsx  # Orders table
│   └── ActivityFeed.tsx  # Activity timeline
├── customer/             # Customer dashboard components
│   ├── CustomerHome.tsx      # Customer home page
│   ├── CustomerLayout.tsx    # Customer layout wrapper
│   ├── CustomerHeader.tsx    # Customer header
│   └── CustomerSidebar.tsx   # Customer sidebar
├── order-management/     # Order management
│   └── OrderManagement.tsx  # Orders page
├── admin/                # Admin management
│   └── AdminManagement.tsx  # Admin users page
├── stores/               # Store management
│   └── StoreManagement.tsx  # Stores page
├── last-mile/            # Last mile delivery
│   └── LastMileDelivery.tsx # Delivery scheduling
├── rosters/              # Roster management
│   └── RosterManagement.tsx # Staff rosters
├── reports/              # Reports & Analytics
│   ├── Reports.tsx       # Main reports page
│   └── ReportsMap.tsx    # Leaflet map component
├── logs/                 # Activity logs
│   └── ActivityLogs.tsx  # Activity tracking
└── rail-scheduling/      # Rail scheduling
    └── RailScheduling.tsx # Rail logistics
```

🛠️ Technology Stack

Frontend (Fully Implemented) ☑

- ⚛️ **React 19.1.0** - Latest React with concurrent features
- 🚀 **React Router v7.7.1** - Full-stack framework with SSR
- ⚡ **Vite 6.3.3** - Ultra-fast build tool and dev server
- 🎨 **Tailwind CSS v4.1.4** - Latest utility-first CSS framework
- 📝 **TypeScript 5.8.3** - Type safety and enhanced DX
- 🎯 **Lucide React** - Beautiful SVG icon library
- 🔧 **Shadcn/ui** - Complete component library integrated
- 🗺️ **Leaflet + React-Leaflet** - Interactive maps for delivery tracking
- 📊 **Recharts** - Data visualization and analytics charts
- 🐳 **Docker** - Containerization for deployment
- 🔐 **Authentication System** - LocalStorage-based auth with route protection

Backend (Planned/In Development)

Based on the backend folder structure:

- **FastAPI** - Modern Python web framework
- **PostgreSQL** - Relational database
- **JWT Authentication** - Token-based auth system
- **SQLAlchemy** - Python SQL toolkit and ORM
- **Pydantic** - Data validation using Python type hints
- **Logistics APIs** - Shipping provider integrations

Design System

Color Scheme (Logistics-Focused)

- **Primary**: Deep Navy Blue (#1e3a8a) - Trust and professionalism
- **Accent**: Coral/Orange-Red (#f97316) - Action and urgency
- **Success**: Green (#22c55e) - Successful operations
- **Warning**: Yellow (#fbbf24) - Alerts and pending states
- **Info**: Blue (#3b82f6) - Information and tracking
- **Background**: Light Blue-Gray (#f8fafc) - Clean interface
- **Glassmorphic UI**: Semi-transparent cards with backdrop blur

UI Features

- **Glassmorphic Design**: Modern frosted glass aesthetic
- **Dark Mode Support**: Full dark/light theme compatibility
- **Responsive Layouts**: Mobile-first responsive design
- **Interactive Charts**: Real-time data visualization
- **Loading States**: Skeleton loaders for better UX
- **Status Badges**: Color-coded status indicators
- **Dropdown Menus**: User profile and action menus
- **Interactive Maps**: Leaflet-based delivery tracking

Component Library

Fully integrated **Shadcn/ui** components:

- **Tables** - Sortable, filterable data tables
- **Badges** - Status and category indicators
- **Buttons** - Primary, secondary, destructive variants
- **Cards** - Container components with shadows
- **Dropdowns** - Action menus and selections
- **Inputs** - Form controls with validation
- **Selects** - Dropdown selections with search
- **Tabs** - Content organization
- **Separators** - Visual dividers

Getting Started - Development Setup

Prerequisites

```
# Required software
- Node.js 20+ (LTS recommended)
- npm or pnpm
- Git
- VS Code (recommended)
```

1. Clone and Setup

```
# Clone the repository
git clone https://github.com/hhh-berzerk/kandypack-logistics-platform.git
cd kandypack-logistics-platform/frontend/UI

# Install dependencies
npm install

# Start development server
npm run dev
```

2. Development Server

```
# The app will be available at:
http://localhost:5173
```

3. Available Scripts

```
# Development (with HMR)
npm run dev

# Type checking
npm run typecheck

# Production build
npm run build

# Start production server
npm run start
```

Key Files and Their Purpose

Configuration Files

`vite.config.ts`

```
// Vite configuration with plugins
- tailwindcss()     // Tailwind CSS v4 integration
- reactRouter()     // React Router v7 plugin
- tsconfigPaths()   // TypeScript path mapping
```

`react-router.config.ts`

```
// React Router configuration
- ssr: true         // Server-side rendering enabled
- Full-stack mode   // Both client and server routing
```

`app/routes.ts`

```
// Route definitions with layout structure
- index: "routes/home.tsx"              // Landing page
- login: "routes/login.tsx"             // Authentication
- dashboard: "routes/dashboard.tsx"     // Main dashboard (Protected)
- orders: "routes/order-management.tsx" // Order management (Protected)
- admin: "routes/admin.tsx"             // Admin panel (Protected)
- stores: "routes/stores.tsx"           // Store management (Protected)
- last-mile: "routes/last-mile.tsx"     // Delivery scheduling (Protected)
- rosters: "routes/rosters.tsx"         // Roster management (Protected)
- reports: "routes/reports.tsx"         // Analytics (Protected)
- logs: "routes/logs.tsx"               // Activity logs (Protected)
```

Core Application Files

`app/root.tsx`

- **Root layout component**
- HTML structure and meta tags
- Global error boundary
- Font loading (Inter)
- Context providers (SidebarContext)

`app/app.css`

- **Global styles and design system**
- Tailwind CSS v4 configuration
- Custom dashboard utility classes
- Light/dark theme support

- Logistics-specific color palette
- Glassmorphic design classes

Authentication System

`hooks/useAuth.tsx`

- **Authentication state management**
- LocalStorage-based session persistence
- Login/logout functionality
- User state management
- Auto-rehydration on app load

`components/ProtectedRoute.tsx`

- **Route protection wrapper**
- Checks authentication status
- Redirects to login if unauthenticated
- Shows loading state during auth check
- Wraps all protected routes

Layout Components

`components/dashboard/DashboardLayout.tsx`

- **Shared dashboard wrapper**
- Sidebar integration
- Header integration
- Consistent layout across pages

`components/dashboard/DashboardHeader.tsx`

- **Navigation header**
- User profile dropdown
- Logout functionality
- Click-outside detection
- User avatar display

`components/dashboard/Sidebar.tsx`

- **Navigation sidebar**
- Retractable/collapsible
- Active route highlighting
- Icon-based navigation
- Uses SidebarContext for state

Page Components

`routes/login.tsx`

- **Authentication page**
- Glassmorphic design with background image
- OAuth integration (Google, GitHub)
- Form validation
- Redirect after login
- Prevents access when already logged in

`routes/dashboard.tsx`

- **Main dashboard**
- Statistics cards (Orders, Revenue, Shipments, Returns)
- Recent orders table
- Activity feed timeline
- Charts and metrics

`components/order-management/OrderManagement.tsx`

- **Order management page**
- Comprehensive orders table
- Status filters and search
- Pagination support
- Order details and actions

`components/admin/AdminManagement.tsx`

- **Admin user management**
- Admin users table
- Role management (Super Admin, Admin, Manager)
- User creation and editing
- Status tracking (Active, Inactive, Suspended)

`components/stores/StoreManagement.tsx`

- **Store management**
- Store locations table
- Capacity tracking
- Manager assignment
- Status management

`components/last-mile/LastMileDelivery.tsx`

- **Delivery scheduling**
- Truck schedule table
- Route planning
- Driver assignments

- Delivery status tracking

`components/rosters/RosterManagement.tsx`

- **Staff roster management**
- Working hours tracking
- Shift scheduling
- Employee management
- Schedule conflicts resolution

`components/reports/Reports.tsx`

- **Analytics dashboard**
- 6 interactive report cards:
  - Sales by Region (Line chart)
  - Order Status Distribution (Pie chart)
  - Revenue Trends (Line chart)
  - Delivery Performance (Bar chart)
  - Customer Satisfaction (Line chart)
  - Sales Breakdown Map (Leaflet map)
- Export functionality
- Date range filtering

`components/logs/ActivityLogs.tsx`

- **Activity tracking**
- System activity log table
- User action tracking
- Timestamp logging
- Severity levels (Info, Warning, Error)

🔧 Development Workflow

1. Adding New Routes

```
// In app/routes.ts
import { type RouteConfig, route, layout, index } from "@react-
router/dev/routes";

export default [
  index("routes/home.tsx"),
  route("/login", "routes/login.tsx"),

  // Protected routes with dashboard layout
  layout("components/dashboard/DashboardLayout.tsx", [
    route("/dashboard", "routes/dashboard.tsx"),
    route("/order-management", "routes/order-management.tsx"),
```

```
    route("/admin", "routes/admin.tsx"),
    route("/stores", "routes/stores.tsx"),
    route("/last-mile", "routes/last-mile.tsx"),
    route("/rosters", "routes/rosters.tsx"),
    route("/reports", "routes/reports.tsx"),
    route("/logs", "routes/logs.tsx"),
  ]),
] satisfies RouteConfig;
```

2. Creating Protected Components

```
// Wrap route components with ProtectedRoute
import { ProtectedRoute } from "../components/ProtectedRoute";
import MyComponent from "../components/MyComponent";

export default function MyPage() {
  return (
    <ProtectedRoute>
      <MyComponent />
    </ProtectedRoute>
  );
}
```

3. Component Structure

```
# Recommended organization
app/
├── components/          # Reusable UI components
│   ├── ui/             # Shadcn/ui base components
│   ├── dashboard/      # Dashboard-specific components
│   ├── order-management/ # Order management components
│   ├── admin/          # Admin management components
│   ├── stores/         # Store management components
│   ├── last-mile/      # Delivery components
│   ├── rosters/        # Roster components
│   ├── reports/        # Analytics components
│   └── logs/           # Activity log components
├── hooks/              # Custom React hooks
│   └── useAuth.tsx     # Authentication hook
├── contexts/           # React Context providers
│   └── SidebarContext.tsx  # Sidebar state
├── lib/                # Utilities and helpers
│   └── utils.ts        # Common functions
└── types/              # TypeScript type definitions
```

4. Styling Approach

```
// Use glassmorphic design patterns
<div className="bg-white/80 dark:bg-gray-800/80 backdrop-blur-lg rounded-lg
shadow-lg p-6">
  <h2 className="text-xl font-semibold mb-4">Card Title</h2>
  <p className="text-gray-600 dark:text-gray-300">Content here</p>
</div>

// Use custom status badges
<Badge variant={status === 'active' ? 'default' : 'destructive'}>
  {status}
</Badge>

// Combine with Tailwind utilities
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
  {/* Responsive grid layout */}
</div>
```

5. Authentication Flow

```
// Using useAuth hook
const { isAuthenticated, user, login, logout, loading } = useAuth();

// Login
const handleLogin = (credentials) => {
  login(credentials);
};

// Logout
const handleLogout = () => {
  logout();
  window.location.href = '/login'; // Full page reload
};

// Check auth status
if (loading) return <LoadingSpinner />;
if (!isAuthenticated) return <Navigate to="/login" />;
```

6. TypeScript Integration

- **Auto-generated types** for routes
- **Strict type checking** enabled
- **Path mapping** configured (~/ maps to app/)
- **Interface definitions** for data models
- **Type-safe component props**

🚀 Deployment Options

Docker Deployment

```
# Build Docker image
docker build -t kandypack-logistics .

# Run container
docker run -p 3000:3000 kandypack-logistics
```

Platform Deployment

The app can be deployed to:

- **Vercel** (recommended for React Router apps)
- **Netlify**
- **Railway**
- **Fly.io**
- **AWS/Azure/GCP**

🗺️ Development Roadmap

Phase 1: Frontend Foundation ☑ COMPLETED

- ☑ React Router v7 setup
- ☑ Tailwind CSS v4 configuration
- ☑ Custom glassmorphic design system
- ☑ Development environment
- ☑ Docker containerization
- ☑ Shadcn/ui component library integration

Phase 2: Core UI Components ☑ COMPLETED

- ☑ Dashboard layout structure
- ☑ Navigation sidebar (retractable)
- ☑ Dashboard header with user profile
- ☑ Order management interface
- ☑ Admin management interface
- ☑ Store management interface
- ☑ Last mile delivery interface
- ☑ Roster management interface
- ☑ Reports & analytics page
- ☑ Activity logs interface
- ☑ Status indicators and badges
- ☑ Interactive data tables
- ☑ Charts and visualizations

Phase 3: Authentication & Security ☑ COMPLETED

- [x] Login page with glassmorphic design
- [x] Authentication system (LocalStorage-based)
- [x] useAuth hook implementation
- [x] Route protection system
- [x] ProtectedRoute wrapper component
- [x] Logout functionality with dropdown
- [x] User session management
- [x] Automatic redirect for unauthenticated users

Phase 4: Advanced UI Features ☑ COMPLETED

- [x] Interactive maps (Leaflet integration)
- [x] Data visualization (Recharts)
- [x] Responsive design (mobile-first)
- [x] Loading skeletons
- [x] Error boundaries
- [x] Dropdown menus
- [x] Search and filter functionality
- [x] Pagination support
- [x] Dark mode compatibility

Phase 5: Backend Integration 🔄 IN PROGRESS

- [x] Backend folder structure created
- [ ] FastAPI backend setup
- [ ] Database design and models
- [ ] API endpoint development
- [ ] JWT authentication integration
- [ ] Data loading with React Router loaders
- [ ] Real-time updates (WebSockets)
- [ ] File upload functionality

Phase 6: Logistics Features 📋 PLANNED

- [ ] Order creation and management
- [ ] Real-time shipment tracking
- [ ] Inventory management
- [ ] Automated reporting
- [ ] Third-party shipping integrations
- [ ] Barcode/QR code scanning
- [ ] Route optimization
- [ ] Delivery confirmations

Phase 7: Advanced Features 📋 FUTURE

- [ ] Real-time notifications
- [ ] Mobile PWA support

- [ ] Multi-tenant support
- [ ] Advanced analytics dashboard
- [ ] API integrations (FedEx, UPS, DHL)
- [ ] Email notifications
- [ ] SMS alerts
- [ ] Customer portal

🛠️ Development Best Practices

Code Organization

```
# Follow these patterns:
- Use TypeScript for all new code
- Implement proper error boundaries
- Use React Router loaders for data fetching
- Follow established naming conventions
- Write responsive, mobile-first components
- Use ProtectedRoute for all authenticated pages
- Leverage Context API for shared state (SidebarContext)
- Implement proper loading states
```

Styling Guidelines

```
# Glassmorphic design patterns
- Use bg-white/80 with backdrop-blur for glass effect
- Implement dark mode with dark: variants
- Use Tailwind utilities for layout
- Combine with Shadcn/ui components
- Use status badges for order states
- Maintain consistent spacing (p-4, p-6, gap-4)
- Use shadow-lg for depth
```

Authentication Best Practices

```
# Security and session management
- Always wrap protected routes with ProtectedRoute
- Check isAuthenticated before rendering sensitive data
- Use useAuth hook for authentication state
- Clear localStorage on logout
- Use window.location.href for logout redirects
- Implement loading states during auth checks
- Show user info in header
```

Performance

```
# Optimization strategies:
- Leverage React Router's built-in code splitting
- Use loaders for efficient data fetching
- Implement proper loading states (skeletons)
- Optimize images and assets
- Use lazy loading for charts and maps
- Minimize bundle size with tree shaking
- Use error boundaries to prevent crashes
```

Data Management

```
# State and data handling
- Use Context API for global state (Sidebar, Auth)
- Keep component state local when possible
- Use TypeScript interfaces for data models
- Implement proper error handling
- Show loading states during data fetch
- Cache data when appropriate
- Use React Router loaders for route data
```

🔍 Current State Analysis

☑ What's Working

- ☑ Modern React Router v7 full-stack setup
- ☑ Tailwind CSS v4 with glassmorphic design
- ☑ TypeScript configuration with strict types
- ☑ Docker containerization ready
- ☑ Complete authentication system with route protection
- ☑ Retractable sidebar with context-based state
- ☑ User profile dropdown with logout
- ☑ 9+ fully functional management pages
- ☑ Interactive charts and analytics
- ☑ Leaflet maps for geographic visualization
- ☑ Complete Shadcn/ui component library
- ☑ Responsive, mobile-first design
- ☑ Loading states and error boundaries
- ☑ Dark mode support

🔄 What's In Progress

- 🔄 **Backend API Integration**: FastAPI backend structure ready

- 🔄 **Real Data**: Currently using mock/sample data
- 🔄 **Database Connection**: Need to connect to PostgreSQL
- 🔄 **JWT Authentication**: Need to replace localStorage with proper JWT

📋 What Needs Development

- **API Endpoints**: Create FastAPI routes for all operations
- **Database Models**: Implement SQLAlchemy models
- **Real-time Features**: WebSocket integration for live updates
- **File Uploads**: Document and image upload functionality
- **Email Notifications**: Order and shipment notifications
- **Advanced Filtering**: More sophisticated search and filters
- **Bulk Operations**: Multi-select and bulk actions
- **Export Features**: CSV/PDF export for reports
- **Mobile App**: Native mobile application
- **Third-party Integrations**: Shipping provider APIs

📋 Immediate Next Steps

1. **Backend Setup**: Initialize FastAPI application
2. **Database Design**: Create database schema and models
3. **API Development**: Build RESTful API endpoints
4. **Authentication**: Implement JWT-based auth
5. **Data Integration**: Connect frontend to backend APIs
6. **Real Data**: Replace mock data with real database queries
7. **Testing**: Write unit and integration tests
8. **Deployment**: Configure production environment

📚 Learning Resources

React Router v7

- Official Documentation
- Data Loading Guide
- SSR and Full-Stack Features

Tailwind CSS v4

- v4 Documentation
- Migration Guide
- OKLCH Color Format

Logistics Domain Knowledge

- Supply Chain Management principles
- Order lifecycle and states
- Shipping and tracking systems

- Inventory management concepts

🤝 Contributing

Development Setup

1. Follow the development setup instructions above
2. Create feature branches for new work
3. Follow TypeScript and React best practices
4. Update documentation when adding features
5. Test across different screen sizes

Code Style

- Use Prettier for formatting
- Follow ESLint rules
- Use semantic commit messages
- Write descriptive component and function names

📄 Project Status

Current Status: **🚧 Active Development - Frontend Complete**

- **Frontend**: ☑ Complete and fully functional
  - Authentication system with route protection
  - 9+ management pages with full UI
  - Interactive charts and maps
  - Responsive design
  - Dark mode support
- **Backend**: 🔄 Structure ready, implementation pending
  - Folder structure created
  - FastAPI configuration needed
  - Database schema to be implemented
- **Database**: ◯ Not yet configured
  - Schema designed (schema.sql)
  - PostgreSQL setup needed
- **Deployment**: 🚧 Docker ready, hosting not configured
  - Frontend containerized
  - Backend deployment pending

Active Branch: `main` - Production-ready frontend

Latest Updates (October 2025):

- ☑ Implemented route protection for all dashboard pages
- ☑ Created ProtectedRoute wrapper component
- ☑ Fixed logout functionality with proper redirect
- ☑ Added glassmorphic login page design
- ☑ Integrated Leaflet maps for delivery tracking
- ☑ Added comprehensive reports and analytics
- ☑ Implemented activity logging interface

🎯 Feature Highlights

Implemented Features ☑

1. Authentication & Security

- Login page with OAuth integration (Google, GitHub)
- LocalStorage-based session management
- Protected routes with automatic redirect
- User profile dropdown with logout
- Session persistence across page reloads

2. Dashboard

- Real-time statistics cards (Orders, Revenue, Shipments, Returns)
- Recent orders table with status indicators
- Activity feed with timeline
- Quick action buttons

3. Order Management

- Comprehensive order table with pagination
- Status filtering (Pending, Processing, Shipped, Delivered, Cancelled)
- Search functionality
- Order details modal
- Action buttons (View, Edit, Track)

4. Admin Management

- Admin user table
- Role management (Super Admin, Admin, Manager, Viewer)
- User status tracking (Active, Inactive, Suspended)
- Add/Edit admin functionality
- Last login tracking

5. Store Management

- Store locations table
- Capacity tracking and utilization
- Manager assignments
- Status management (Active, Inactive, Under Maintenance)
- Contact information

6. Last Mile Delivery

- Truck scheduling table
- Route assignments
- Driver management
- Delivery status tracking
- Time slot management

7. Roster Management

- Employee roster table
- Shift scheduling
- Working hours tracking
- Position management
- Schedule conflict detection

8. Reports & Analytics

- 6 interactive report cards with charts:
    - Sales by Region (Line chart)
    - Order Status Distribution (Pie chart)
    - Revenue Trends (Line chart)
    - Delivery Performance (Bar chart)
    - Customer Satisfaction (Line chart)
    - Sales Breakdown Map (Leaflet geographic map)
- Export functionality
- Date range filtering

9. Activity Logs

- System activity tracking
- User action logging
- Severity levels (Info, Warning, Error, Success)
- Timestamp tracking
- Searchable and filterable

10. UI/UX Features

- Retractable navigation sidebar
- Glassmorphic design aesthetic
- Responsive tables with sorting

- Status badges with colors
- Loading skeletons
- Error boundaries
- Dark mode support
- Interactive dropdowns
- Tooltips and hover effects

🔐 Authentication System Details

Current Implementation (LocalStorage-based)

How it works:

1. User logs in via login page
2. Credentials validated (currently client-side mock)
3. User data stored in `localStorage` with key `kandypack_user`
4. `useAuth` hook manages authentication state
5. `ProtectedRoute` wrapper checks auth before rendering
6. User can logout via dropdown menu in header

Storage Structure:

```
{
  "id": "user_id",
  "name": "John Doe",
  "email": "john@example.com",
  "role": "admin"
}
```

Security Notes:

- ⚠️ Current implementation is for development only
- 🔄 Will be replaced with JWT tokens from backend
- 🔄 Need to implement token refresh mechanism
- 🔄 Need to add HTTPS for production

Planned Implementation (JWT-based)

Future approach:

1. User logs in via API call to FastAPI backend
2. Backend validates credentials against database
3. Backend returns JWT access token and refresh token
4. Tokens stored in httpOnly cookies (more secure)
5. Frontend includes token in Authorization header
6. Backend validates token on each request
7. Automatic token refresh when expired

Data Models

Current Mock Data Models

Order Model

```
interface Order {
  id: string;
  customer: string;
  items: number;
  total: number;
  status: 'pending' | 'processing' | 'shipped' | 'delivered' | 'cancelled';
  date: string;
}
```

Admin Model

```
interface Admin {
  id: string;
  name: string;
  email: string;
  role: 'super_admin' | 'admin' | 'manager' | 'viewer';
  status: 'active' | 'inactive' | 'suspended';
  lastLogin: string;
}
```

Store Model

```
interface Store {
  id: string;
  name: string;
  location: string;
  manager: string;
  capacity: number;
  current: number;
  status: 'active' | 'inactive' | 'maintenance';
}
```

Delivery Model

```
interface Delivery {
  id: string;
```

```
  truck: string;
  driver: string;
  route: string;
  status: 'scheduled' | 'in_transit' | 'completed' | 'delayed';
  time: string;
  packages: number;
}
```

Component Architecture

Layout Components

```
DashboardLayout (Wrapper)
├── Sidebar (Navigation)
│   ├── Logo
│   ├── Navigation Items
│   │   ├── Dashboard
│   │   ├── Orders
│   │   ├── Admin
│   │   ├── Stores
│   │   ├── Last Mile
│   │   ├── Rosters
│   │   ├── Reports
│   │   └── Logs
│   └── Collapse Toggle
└── Main Content Area
    ├── DashboardHeader
    │   ├── Page Title
    │   ├── Search Bar
    │   └── User Profile Dropdown
    │       ├── Profile Info
    │       ├── Settings
    │       └── Logout Button
    └── Page Content (Route-specific)
```

Shadcn/ui Components Used

Component	Usage
`Table`	Order lists, admin lists, store lists, etc.
`Badge`	Status indicators (shipped, pending, active, etc.)
`Button`	Actions, forms, navigation
`Card`	Statistics cards, content containers
`DropdownMenu`	User profile menu, action menus

Component	Usage
`Input`	Search bars, form inputs
`Label`	Form labels
`Select`	Filters, dropdowns
`Separator`	Visual dividers
`Tabs`	Content organization

Custom Components

Component	Purpose
`ProtectedRoute`	Authentication wrapper for routes
`ErrorBoundary`	Global error handling
`DashboardSkeleton`	Loading state for dashboard
`StatsCard`	Metric display cards
`RecentOrders`	Order table component
`ActivityFeed`	Activity timeline
`ReportsMap`	Leaflet map for analytics
`UserAvatar`	User profile picture

This project is in active development with a complete, production-ready frontend and a planned backend integration using FastAPI and PostgreSQL.