Vayunex UI - Project Documentation

Version: 1.0.0
Date: January 11, 2026
Framework: React 18 + Vite + Bootstrap 5

Table of Contents

1. Project Overview
2. Technology Stack
3. Folder Structure
4. Architecture
5. Modules
6. Components
7. Services
8. Configuration
9. Data Layer
10. Authentication
11. Theming
12. Running the Application

---

1. Project Overview

Vayunex UI is a modern, modular Inventory & Billing SaaS frontend application. It provides a complete dashboard for managing inventory items, categories, units, and generating reports.

Key Features

- 🔐 **Authentication System** - Login with session persistence
- 📊 **Dashboard** - Real-time stats and recent items overview
- 📦 **Inventory Management** - Full CRUD for items with filtering
- 🏷️ **Category Management** - Organize items by categories
- 🌙 **Dark/Light Theme** - User-toggleable theme with persistence
- 📑 **Multi-Tab System** - Browser-like tab navigation

---

2. Technology Stack

Technology	Purpose
React 18	UI Library
Vite 7.x	Build Tool & Dev Server
Bootstrap 5	CSS Framework
React-Bootstrap	Bootstrap React Components
SCSS	Custom Styling & Theme
Zustand	Lightweight State Management

Lucide React	Icon Library

---

3. Folder Structure

```
vayunex-ui/
├── public/
├── src/
│   ├── assets/              # Static assets (images, fonts)
│   ├── components/
│   │   └── layout/          # Layout components
│   │       ├── AppLayout.jsx
│   │       ├── Header.jsx
│   │       ├── Sidebar.jsx
│   │       └── TabBar.jsx
│   ├── config/
│   │   └── company.js       # Company branding config
│   ├── context/
│   │   └── ThemeContext.jsx # Theme provider
│   ├── data/                # Static JSON data (Phase 1)
│   │   ├── categories.json
│   │   ├── items.json
│   │   ├── menu.json
│   │   ├── units.json
│   │   └── users.json
│   ├── lib/                 # Shared utilities
│   │   ├── AppUser.js       # Auth singleton
│   │   ├── TabManager.js    # Tab state (Zustand)
│   │   └── index.js         # Barrel export
│   ├── modules/             # Feature modules
│   │   ├── auth/
│   │   ├── categories/
│   │   ├── dashboard/
│   │   └── inventory/
│   ├── App.jsx              # Root component
│   ├── App.scss             # Global styles
│   └── main.jsx             # Entry point
├── index.html
├── package.json
└── vite.config.js
```

---

4. Architecture

4.1 Module-based Architecture

Each feature is self-contained in its own module under `src/modules/`:

```
modules/
├── auth/
│   ├── components/LoginForm.jsx
```

```
│   ├── pages/LoginPage.jsx
│   ├── services/authService.js
│   └── index.js
├── dashboard/
│   ├── components/StatCard.jsx
│   ├── components/RecentItemsTable.jsx
│   ├── pages/DashboardPage.jsx
│   ├── services/dashboardService.js
│   └── index.js
├── inventory/
│   ├── pages/ItemsListPage.jsx
│   ├── services/inventoryService.js
│   └── index.js
└── categories/
    ├── pages/CategoriesPage.jsx
    ├── services/categoryService.js
    └── index.js
```

4.2 Import Convention

Each module exports via barrel file (`index.js`):

```
// Importing from auth module
import { LoginPage, login } from './modules/auth';

// Importing from shared lib
import { AppUser, useTabStore } from './lib';
```

---

5. Modules

5.1 Auth Module (`modules/auth/`)

Handles user authentication.

File	Purpose
`components/LoginForm.jsx`	Reusable login form with validation
`pages/LoginPage.jsx`	Complete login page with branding
`services/authService.js`	`login()`, `logout()` functions

Service Functions:

```
login(email, password) → { success, data: { user, accessToken, refreshToken } }
logout() → { success }
```

5.2 Dashboard Module (`modules/dashboard/`)

Main dashboard with statistics and recent items.

File	Purpose
`components/StatCard.jsx`	Individual stat card with icon
`components/RecentItemsTable.jsx`	Table of recent inventory items
`pages/DashboardPage.jsx`	Complete dashboard layout
`services/dashboardService.js`	Stats and recent items API

Service Functions:

```
getDashboardStats() → { totalItems, lowStockItems, totalCategories, totalValue }
getRecentItems(limit) → [items]
```

5.3 Inventory Module (`modules/inventory/`)

Full inventory item management.

File	Purpose
`pages/ItemsListPage.jsx`	Items table with search/filter
`services/inventoryService.js`	Full CRUD operations

Service Functions:

```
getItems(params) → [items with category_name]
getItemById(id) → item
createItem(data) → newItem
updateItem(id, data) → updatedItem
deleteItem(id) → { success }
```

5.4 Categories Module (`modules/categories/`)

Category management with card-based UI.

File	Purpose
`pages/CategoriesPage.jsx`	Category grid with cards
`services/categoryService.js`	Full CRUD operations

Service Functions:

```
getCategories() → [categories]
getCategoryById(id) → category
createCategory(data) → newCategory
updateCategory(id, data) → updatedCategory
deleteCategory(id) → { success }
```

---

6. Components

6.1 Layout Components (`components/layout/`)

Component	Description
AppLayout.jsx	Main app shell combining Sidebar + Header + TabBar + Content
Sidebar.jsx	Left navigation with collapsible menu from `menu.json`
Header.jsx	Top bar with search, notifications, theme toggle, user menu
TabBar.jsx	Browser-like tab system for multi-document navigation

6.2 Component Hierarchy

```
App.jsx
└── ThemeProvider
    └── AppWrapper
        ├── LoginPage (if not authenticated)
        └── AppLayout (if authenticated)
            ├── Sidebar
            ├── Header
            ├── TabBar
            └── [ActivePage] (Dashboard/Items/Categories)
```

---

7. Services

7.1 Shared Library (`lib/`)

File	Description
AppUser.js	Singleton class for authentication state. Persists to `sessionStorage`.
TabManager.js	Zustand store for browser-like tab management.

AppUser API:

```
AppUser.getInstance()    // Get user object
AppUser.setUser(data, tokens)  // Set after login
AppUser.isLoggedIn()     // Check auth status
AppUser.getToken()       // Get JWT token
AppUser.hasPermission(p) // Check permission
AppUser.clear()          // Logout
```

TabManager API:

```
useTabStore().tabs              // Array of open tabs
useTabStore().activeTabId       // Current tab ID
useTabStore().openTab(tab)      // Open/switch to tab
```

```
useTabStore().closeTab(id)      // Close a tab
useTabStore().switchTab(id)     // Switch active tab
```

---

8. Configuration

8.1 Company Config (`config/company.js`)

```
export const companyConfig = {
  name: 'Vayunex Solution',
  tagline: 'Enterprise Billing & Inventory',
  logo: '/logo.png',
  primaryColor: '#22c55e',
  supportEmail: 'support@vayunexsolution.com'
};
```

8.2 Menu Config (`data/menu.json`)

Dynamic sidebar menu structure:

```
[
  {
    "id": "dashboard",
    "title": "Dashboard",
    "icon": "LayoutDashboard",
    "url": "/dashboard"
  },
  {
    "id": "inventory",
    "title": "Inventory",
    "icon": "Package",
    "children": [
      { "id": "items", "title": "Items", "url": "/items" },
      { "id": "categories", "title": "Categories", "url": "/categories" }
    ]
  }
]
```

---

9. Data Layer

9.1 Static JSON Files (Phase 1)

Currently using static JSON for rapid development:

File	Contents
`users.json`	User credentials and roles
`items.json`	Inventory items with prices

`categories.json`	Item categories
`units.json`	Measurement units
`menu.json`	Sidebar menu structure

9.2 Switching to Live API (Phase 2)

Each module service can be updated to call real APIs:

```
// Current (Phase 1)
import itemsData from '../../../data/items.json';
export const getItems = async () => ({ success: true, data: itemsData });

// Future (Phase 2)
import axios from 'axios';
export const getItems = async () => {
  const res = await axios.get('/api/items');
  return { success: true, data: res.data };
};
```

---

10. Authentication

10.1 Login Flow

1. User enters credentials on `LoginPage`
2. `authService.login()` validates against `users.json`
3. On success, `AppUser.setUser()` stores user + tokens in `sessionStorage`
4. App renders `AppLayout` with dashboard

10.2 Session Persistence

- User data stored in `sessionStorage` (survives refresh, not tab close)
- On app init, `AppUser` loads from storage automatically
- `AppUser.isLoggedIn()` checks for valid token

10.3 Test Credentials

Email	Password	Role
admin@gmail.com	admin@1234	Admin
user@vaynex.com	user@1234	User

---

11. Theming

11.1 Theme Context

```
// Usage in any component
import { useTheme } from './context/ThemeContext';
```

```
const { isDark, toggleTheme } = useTheme();
```

11.2 Bootstrap Theme Override

Custom theme in `App.scss`:

```
$primary: #22c55e;  // Vayunex Green
$body-bg: #f8fafc;
$enable-shadows: true;
$enable-gradients: true;

@import "bootstrap/scss/bootstrap";
```

11.3 Dark Mode

Toggle sets `data-bs-theme="dark"` on document, Bootstrap handles the rest.

---

12. Running the Application

12.1 Development

```
cd vayunex-ui
npm install
npm run dev
```

Opens at: **http://localhost:5173**

12.2 Production Build

```
npm run build
npm run preview
```

12.3 Environment Variables

Create `.env` for API configuration:

```
VITE_API_URL=https://inv-api.vayunexsolution.com
```

Appendix: File Reference

Path	Type	Description
`src/App.jsx`	Component	Root application component
`src/main.jsx`	Entry	React DOM render entry
`src/App.scss`	Styles	Bootstrap theme overrides

`src/lib/AppUser.js`	Class	Authentication singleton
`src/lib/TabManager.js`	Store	Zustand tab state
`src/modules/*/index.js`	Barrel	Module exports
`src/modules/*/services/*.js`	Service	API/data functions
`src/modules/*/pages/*.jsx`	Page	Full page components
`src/modules/*/components/*.jsx`	Component	Reusable UI pieces
