temp2

Build a **complete production-ready frontend** for

Vendor Cab and Driver Onboarding & Hierarchy Management System,

using **React + TypeScript + TailwindCSS + React Router DOM**.

The frontend should:

- Use **mock APIs** that perfectly mirror the real backend endpoints,
- Be **structured for easy replacement with real APIs** later,
- Include **authentication, vendor hierarchy, driver/vehicle onboarding, document verification, notifications**, and **role/permission management**.

🎯 System Overview

This system manages multi-level vendors (Super → Regional → City → Local), cab & driver onboarding, document verification, and compliance.

Vendors can delegate tasks to sub-vendors but still maintain control and visibility.

🧩 Frontend Functional Modules

1. **Authentication**
   - Pages: `/login`, `/register`, `/forgot-password`
   - Mocked JWT login, redirect based on role
2. **Dashboard**
   - Summary cards (vendors, drivers, vehicles, pending verifications)
   - Mock from `/api/hierarchy/stats/{vendorId}`
3. **Vendor Management**
   - Table + CRUD + block/unblock actions
   - Parent vendor can manage children; children can’t override parent
4. **Driver Management**

- Add/Edit/Delete drivers
- Change status (Active/Inactive)

5. **Vehicle Management**
   - Add/Edit/Delete vehicles
   - Assign driver dropdown
6. **Document Management**
   - Upload, Verify/Reject documents
   - Show expired docs
7. **Hierarchy View**
   - Expandable tree showing vendor levels
   - Click to open details
8. **Roles & Permissions**
   - CRUD roles, assign permissions
9. **Notifications & Activity Logs**
   - List unread notifications
   - Recent activities from `/api/activity`
10. **Settings**

- Profile, Change Password

🧠 Tech Stack

- React + TypeScript + TailwindCSS
- React Router DOM (for routing)
- Zustand or Context API (for global state)
- Mock backend with `api.ts`
- Vite or CRA build compatible with Vercel

🗂️ Folder Structure

```
src/
├── components/
│   ├── tables/
│   ├── forms/
│   ├── modals/
├── pages/
│   ├── Auth/
│   ├── Dashboard/
│   ├── Vendors/
│   ├── Drivers/
│   ├── Vehicles/
│   ├── Documents/
│   ├── Hierarchy/
│   ├── Roles/
│   ├── Notifications/
│   └── Settings/
├── mocks/
│   └── api.ts
├── utils/
│   └── apiClient.ts
├── context/
├── hooks/
├── App.tsx
├── main.tsx
└── index.css
```

⚙️ Mock API Layer

```
// src/mocks/api.ts
type HttpMethod = "GET" | "POST" | "PUT" | "DELETE";

export const mockApi = async (url: string, method: HttpMethod = "GET", bod
```

```
y?: any) => {
  console.log(`[MOCK] ${method} -> ${url}`, body || "");
  await new Promise((res) => setTimeout(res, 400)); // simulate delay

  // ---------- AUTH ----------
  if (url === "/api/auth/login" && method === "POST")
    return { token: "mock-jwt", email: body.email, vendorId: 1, vendorLevel: "RE
GIONAL", role: "ADMIN", permissions: ["CAN_VIEW_VENDOR"] };

  if (url === "/api/auth/signup" && method === "POST")
    return { userId: 1, email: body.email, vendorId: 1, status: "ACTIVE", createdA
t: new Date().toISOString() };

  if (url === "/api/auth/me" && method === "GET")
    return { role: "ADMIN", vendorId: 1, vendorLevel: "SUPER", email: "user@ex
ample.com" };

  // ---------- VENDORS ----------
  if (url === "/api/vendors/list" && method === "GET")
    return [
      { vendorId: 1, name: "Super Vendor", region: "India", level: "SUPER", statu
s: "ACTIVE" },
      { vendorId: 2, name: "Regional Vendor", region: "Maharashtra", level: "REG
IONAL", status: "ACTIVE" },
    ];

  if (url === "/api/vendors/create" && method === "POST")
    return { vendorId: Math.random() * 1000, ...body, status: "ACTIVE" };

  if (url.includes("/block") && method === "PUT") return "🚫 Vendor blocked";
  if (url.includes("/unblock") && method === "PUT") return "🟢 Vendor unbloc
ked";

  // ---------- DRIVERS ----------
  if (url === "/api/drivers/list" && method === "GET")
    return [{ driverId: 1, name: "Amit Kumar", licenseNo: "DL123", phone: "9876
```

```
543210", status: "ACTIVE" }];

  if (url === "/api/drivers/add" && method === "POST")
    return { driverId: Math.random() * 1000, ...body, status: "ACTIVE" };

  if (url.includes("/drivers/") && method === "DELETE") return "🗑 Driver delet
ed";

  // ---------- VEHICLES ----------
  if (url === "/api/vehicles/list" && method === "GET")
    return [{ vehicleId: 1, registrationNo: "MH12AB1234", model: "Sedan", type:
"Car", status: "ACTIVE" }];

  if (url === "/api/vehicles/add" && method === "POST")
    return { vehicleId: Math.random() * 1000, ...body, status: "ACTIVE" };

  // ---------- DOCUMENTS ----------
  if (url.includes("/api/documents/list/") && method === "GET")
    return [{ documentId: 1, fileName: "License.png", type: "DL", status: "PENDI
NG", expiryDate: "2025-12-31" }];

  if (url.includes("/api/documents/upload") && method === "POST")
    return { fileName: "DL123.png", type: "DL", status: "PENDING" };

  // ---------- HIERARCHY ----------
  if (url.includes("/api/hierarchy/stats") && method === "GET")
    return { totalVendors: 5, totalDrivers: 20, totalVehicles: 10, blockedVendors:
1 };

  if (url.includes("/api/hierarchy/") && method === "GET")
    return { vendorId: 1, vendorName: "Super Vendor", children: [{ vendorId: 2,
vendorName: "Regional Vendor", children: [] }] };

  // ---------- NOTIFICATIONS ----------
  if (url === "/api/notifications/list" && method === "GET")
    return [{ id: 1, message: "Driver license expiring soon", read: false }];
```

```
  // ---------- ACTIVITY ----------
  if (url === "/api/activity" && method === "GET")
    return [{ logId: 1, action: "Document Verified", details: "DL for Driver 1", time
stamp: new Date().toISOString() }];

  return { message: `No mock found for ${method} ${url}` };
};
```

⚙️ API Client (Mock + Real Switch)

```
// src/utils/apiClient.ts
import { mockApi } from "../mocks/api";
import axios from "axios";

const USE_MOCK = import.meta.env.VITE_USE_MOCK === "true";

const liveClient = axios.create({
  baseURL: import.meta.env.VITE_API_BASE_URL,
  headers: { "Content-Type": "application/json" },
});

export const apiClient = {
  get: (url: string) => (USE_MOCK ? mockApi(url, "GET") : liveClient.get(url).the
n(r => r.data)),
  post: (url: string, data?: any) => (USE_MOCK ? mockApi(url, "POST", data) : li
veClient.post(url, data).then(r => r.data)),
  put: (url: string, data?: any) => (USE_MOCK ? mockApi(url, "PUT", data) : live
Client.put(url, data).then(r => r.data)),
  delete: (url: string) => (USE_MOCK ? mockApi(url, "DELETE") : liveClient.delet
e(url).then(r => r.data)),
};
```

🔐 .env File Example

```
# Base URL for your live backend (Spring Boot / Docker)
VITE_API_BASE_URL=https://api.your-backend.com

# Toggle between mock and live
VITE_USE_MOCK=true

REACT_APP_API_URL=https://mockapi.local
```

To switch to real backend later:

- Change `VITE_USE_MOCK` → `false`
- Ensure your backend URL is correct in `VITE_API_BASE_URL`

💡 Final Output Expected

- Responsive, navigable, and deployable Vercel frontend.
- All data fetched from `mockApi()` now.
- Change `.env` to `VITE_USE_MOCK=false` to instantly switch to real backend.
- Clean UI with sidebar + dashboard + CRUD pages.
- Dark/light mode toggle.
- Built for multi-role users: Super Vendor, Sub Vendor, Admin.

🎯 Goal for lovable:

Deliver a **deployable frontend app** with:

- Mock backend integrated (`api.ts`)
- Swappable live backend support (`apiClient.ts`)
- Proper file structure, routing, and Tailwind design
- Clear role-based access & modular code organization.

The output should be a working React+TypeScript project ready to deploy on lovable.