**CHEMIST INVENTORY MANAGEMENT**

**A Mini Project Report**

**Submitted By**

**PAVITHRA N -230701225**

**NEHA K - 230701209**

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE OF ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

**BONAFIDE CERTIFICATE**

Certified that this project report “**CHEMIST INVENTORY MANAGEMENT**” is the bonafide work of “**NEHA K(230701209) , PAVITHRA N(230701225)**” who carried out the project work .

**Submitted for the Practical Examination held on 23.11.2024**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**Abstract**

we propose the development of a Chemist Inventory Management System (PIMS) using the MERN stack.The MERN stack, which consists of MongoDB, Express.js, React, and Node.js, offers a powerful and versatile platform for building modern web applications. By leveraging the capabilities of each component, we aim to create a comprehensive and user-friendly system that streamlines inventory management processes for pharmacies.Pharmacists can easily add new products, update existing product information, and remove discontinued items from the inventory. Each product entry includes details such as name, description, quantity, expiry date, and price.The system supports user authentication to ensure data security and privacy. Different user roles, such as pharmacists, managers, and administrators, have varying levels of access to system functionalities, helping to enforce proper data management practices.

### TABLE OF CONTENTS

1. **INTRODUCTION**  **Page no:**

1.1 Introduction 5  
1.2 Objectives 5  
1.3 Modules 6

1. **SURVEY OF TECHNOLOGIES**

2.1 Software Description 8  
2.2 Languages 8  
2.2.1 JavaScript 8  
2.2.2 HTML/CSS 9  
2.2.3 Node.js 9  
2.2.4 Express.js 9  
2.2.5 React.js 10  
2.2.6 MongoDB 10

1. **REQUIREMENTS AND ANALYSIS**

3.1 Requirement Specification 11   
3.2 Hardware and Software Requirements 11  
3.3 Architecture Diagram 13  
3.4 ER Diagram 14

1. **PROGRAM CODE 15**
2. **RESULTS AND DISCUSSION 54**
3. **CONCLUSION 57**
4. **REFERENCES 58**

**CH-1 INTRODUCTION**

* 1. **INTRODUCTION**

Pharmacy inventory management is a critical aspect of ensuring that a pharmacy operates efficiently and effectively. It involves the systematic oversight of stock, including the acquisition, storage, and distribution of pharmaceutical products. Proper inventory management ensures that medications are available when patients need them, reduces waste due to expired drugs, and optimizes the financial investment in stock. This process also helps in maintaining regulatory compliance and improving patient safety by ensuring that medications are stored and dispensed correctly. Effective pharmacy inventory management leverages technology, such as inventory management software, to streamline operations and provide real-time data for informed decision-making.

**1.2 OBJECTIVES**

The primary objective of the Pharmacy inventory management is the effective inventory management system that guarantees the essential medications are consistently available to meet patient demand, thereby avoiding shortages that can impact patient care by closely monitoring expiration dates and turnover rates, pharmacies can reduce waste caused by expired or unused medications, leading to cost savings and environmental benefits. Proper inventory management involves organizing and storing medications in a manner that maintains their efficacy and safety, such as controlling temperature and humidity levels. Accurate inventory records help prevent dispensing errors and ensure that patients receive the correct medications, thereby improving overall patient safety and health outcomes. Well-maintained inventory records simplify the auditing process, whether for internal reviews or external inspections, ensuring transparency and accountability.

* 1. **MODULES**

Creating a Chemist inventory management system involves several components to manage and check whether the medicine is there or not. Here’s a modular approach to building this project, focusing on different functionalities. Each module will be responsible for a specific part of the providing quantity medicines.

**User Authentication and Security Module:**

Secure registration and login processes, including single-factor

Authentication. Ensures all medicines data is encrypted and securely stored.

Manages personal information and preferences.

**Budgeting Module:**

A budgeting module for Chemist inventory management involves tracking

inventory levels, forecasting sales, managing expenses, and comparing

suppliers to ensure cost-effectiveness. It includes automatic reordering based on

predefined stock levels, detailed financial reporting, and integration with point-

of-sale systems. Key functionalities encompass setting reorder points,

monitoring cash flow, and providing real-time dashboards and alerts. The

module aims to streamline inventory control, optimize budget allocation, and

enhance profitability through efficient cost management and supplier

negotiations, ultimately supporting strategic decision-making and operational

efficiency in a Chemist setting.

**CH-2 SURVEY OF TECHNOLOGIES**

**2.1 SOFTWARE DESCRIPTION**

A Pharmacy Inventory Management System built with the MERN stack

(MongoDB, Express.js, React.js, and Node.js) streamlines pharmacy operations

by managing inventory, tracking stock levels, and generating reports. Key

features include secure user authentication and role-based access,

comprehensive inventory management with alerts for low stock and expiration

dates, and sales and purchase management with detailed transaction tracking.

Reporting and analytics features provide insights into inventory status, sales

trends, and supplier performance. The frontend, developed in React.js, offers a

dynamic user interface with advanced search and filter options, while the

backend, built with Node.js and Express, ensures robust API design and

integration with third-party services. MongoDB serves as the flexible,

document-oriented database managed through Mongoose, and JWT secures user

sessions. Deployment can be achieved through cloud platforms like Heroku or

AWS, with Docker used for containerization and a CI/CD pipeline for

streamlined updates. This system enhances pharmacy efficiency and accuracy,

offering a scalable, modern solution for inventory control.

**2.2 LANGUAGES**

**Java script:**

JavaScript adds interactivity and dynamic functionality to

Web pages. It is utilized for client-side scripting, form validation, and

Asynchronous data loading.

**HTML/CSS:**

HTML is used for creating the structure and content of web pages. It Provides a standard markup language for organizing information and Displaying it in web browsers. CSS is employed for styling and formatting the appearance of web pages, Ensuring a visually appealing and user-friendly interface.

A tailwind refers to a wind that blows in the same direction as the course of an object, typically enhancing its speed and efficiency. This term is commonly used in aviation and sailing, where a tailwind helps planes or boats move faster.

Metaphorically, a tailwind can also refer to any factor or influence that helps propel or support progress, growth, or success in various contexts, such as business or sports.

**NODE.JS:**

Node.js is an open-source, cross-platform JavaScript runtime environment And library for running web applications outside the client's browser.

**EXPRESS.JS:**

Express is a node.js web application framework that provides broad Features for building web and mobile applications. It is used to build a single Page, multipage, and hybrid web application.

**REACT.JS:**

React.js aims to simplify the intricate process of building interactive user

Interfaces.

**MongoDB:**

MongoDB is built on a scale-out architecture that has become popular with Developers of all kinds for developing scalable applications with evolving data Schemas.

**CH-3 REQUIEMENTS AND ANALYSIS**

**3.1 REQUIREMENT SPECIFICATION**

The software requirements for a pharmacy inventory management system are multifaceted, aiming to streamline operations, ensure regulatory compliance, and enhance patient care. Additionally, the system should provide comprehensive order management functionalities, including automated order generation and supplier integration. Alert notifications for low stock, expired medications, and recalls are crucial for timely actions. Robust reporting and analytics capabilities are needed to analyze inventory turnover, forecast demand, and assess financial impact. Regulatory compliance, especially regarding controlled substances, audit trails, and data security, is paramount.Seamless integration with other pharmacy systems, user-friendly interfaces, and scalability ensure ease of use and adaptability to evolving needs. By meeting these requirements, the pharmacy inventory management system can optimize operations, improve patient safety, and support business growth.

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

**Hardware Requirements**

**Development Environment**

**Processor:** Dual-core processor or higher

**Memory (RAM):** 4 GB or higher

**Storage:** 50 GB free disk space

**Network:** Broadband internet connection for downloading dependencies and accessing online resources

**Monitor:** 1080p resolution or higher

**Peripherals:** Keyboard and mouse

**Production Environment (Server)**

**Processor**: Quad-core processor or higher

**Memory (RAM):** 8 GB or higher

**Storage:** 100 GB SSD or higher

**Network:** Reliable high-speed internet connection

**Operating System:** Linux (Ubuntu, CentOS) or Windows Server,

**Backup:** Regular backup system in place

**Software Requirements**

**Operating System:** Windows 10 or higher, macOS, or Linux

**Node.js:** Version 14.x or higher

**NPM (Node Package Manager):** Comes with Node.js

**Database:** MongoDB (local installation or cloud service like MongoDB Atlas)

**Code Editor:** Visual Studio Code, Sublime Text, or any preferred IDE

**Version Control:** Git (with GitHub, GitLab, or Bitbucket for repository management)

**Browser:** Google Chrome, Firefox, or any modern web browser

# **Architecture Diagram:**

# 

# **ER Diagram:**

# 

**Program Code:**

## **Back End: Server.js**

require("dotenv").config();

const express=require("express");

const mongoose=require("mongoose");

const bodyParser=require("body-parser");

const cors=require("cors");

const userRoute=require("./routes/userRoute");

const errorHandler=require("./middleware/errorMiddleware");

const cookieParser=require("cookie-parser");

const productRoute = require("./routes/productRoute");

const supplierRoutes = require("./routes/supplierRoutes");

const app =express();

//middlewares

app.use(express.json());

app.use(cookieParser());

app.use(express.urlencoded({extended: false}));

app.use(bodyParser.json());

app.use(cors());

//route middlewares

const mongoURI = process.env.MONGO\_URI;

app.use(cors({

origin: 'http://localhost:5173',

credentials: true

}));

app.use("/api/users",userRoute);

app.use("/api",productRoute);

app.use('/api', supplierRoutes);

//routes

app.get("/",(req,res)=>{

res.send("Home page");

});

//error middleware

app.use(errorHandler);

const PORT =3300;

mongoose.connect(mongoURI)

.then(() => console.log('MongoDB Connected'))

.catch(err => console.error('MongoDB connection error:', err));

app.listen(PORT,()=>console.log(Server running on PORT ${PORT}))

## **Controllers:**

### User-controllers:

### const asyncHandler =require("express-async-handler")

### const User = require("../model/userModel");

### const jwt = require("jsonwebtoken");

### const bcrypt = require("bcryptjs");

### const expressAsyncHandler = require("express-async-handler");

### const Token = require("../model/tokenModel");

### const crypto = require("crypto");

### const generateToken =(id)=>{

### return jwt.sign({id}, process.env.JWT\_SECRET, { expiresIn: '1h' })

### };

### const registerUser =asyncHandler(async(req,res)=>{

### const {name, email, password, phone} = req.body;

### if(!name || !email || !password){

### res.status(400)

### throw new Error("Please fill all field")

### }

### if(password.length < 6){

### res.status(400)

### throw new Error("Password must be up to 6 characters")

### }

### const userExists=await User.findOne({email})

### if(userExists){

### res.status(400)

### throw new Error("Email has already been registered ")

### }

### const user = await User.create({

### name,

### email,

### password,

### phone,

### });

### // generate token

### const token = generateToken(user.\_id)

### //send http only cookie

### res.cookie("accessToken", token, {

### path: "/",

### httpOnly: true,

### expires: new Date(Date.now() + 1000 \* 86400),

### samSite: "none",

### secure: true,

### });

### if(user){

### const {\_id, name, email, photo, phone, bio }= user

### res.status(201).json({

### \_id, name, email, photo, phone, bio,token,

### })

### }else{

### res.status(400)

### throw new Error("Invalid user data")

### }

### });

### const loginUser = async (req, res) => {

### try {

### const { email, password } = req.body;

### console.log(req.body);

### const user = await User.findOne({ email });

### if (!user) {

### return res.status(400).json({ message: "Invalid username or password" });

### }

### const isMatch = await bcrypt.compare(password, user.password);

### if (!isMatch) {

### return res.status(400).json({ message: "Invalid username or password" });

### }

### const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET, {

### expiresIn: "1h",

### });

### res.status(200).json({ msg: 'Login successful', token });

### } catch (error) {

### res.status(500).json({ message: "Internal server error", error: ${error} });

### }

### };

### //logout

### const logout = asyncHandler(async (req, res)=> {

### res.cookie("token", "", {

### path: "/",

### httpOnly: true,

### expires: new Date(0),

### samSite: "none",

### secure: true,

### });

### return res.status(200).json({message: "Successfully Logged Out"});

### });

### //Get user

### /\*const getUser = asyncHandler(async(req, res)=>{

### const user = await User.findById(req.user.\_id)

### if(user){

### const {\_id, name, email, photo, phone, bio }= user

### res.status(200).json({

### \_id, name, email, photo, phone, bio,

### })

### }else{

### res.status(400)

### throw new Error("User Not Found")

### }

### });

### const getUser = asyncHandler(async(req, res)=>{

### const user = await User.findOne({ email: req.user.email });

### if(user){

### const {\_id, name, email, photo, phone, bio }= user;

### res.status(200).json({

### \_id, name, email, photo, phone, bio

### });

### } else {

### res.status(400);

### throw new Error("User Not Found");

### }

### });\*/

### const getUser = asyncHandler(async (req, res) => {

### const user = await User.findById(req.user.\_id);

### if (user) {

### const { \_id, name, email, photo, phone, bio } = user;

### res.status(200).json({

### \_id,

### name,

### email,

### photo,

### phone,

### bio,

### });

### } else {

### res.status(400);

### throw new Error("User Not Found");

### }

### });

### //get login status

### const loginStatus = asyncHandler(async(req, res)=>{

### const token = req.cookies.token;

### if(!token){

### return res.json(false);

### }

### // verify token

### const verified = jwt.verify(token, process.env.JWT\_SECRET);

### if(verified){

### return res.json(true);

### }

### return res.json(false);

### });

### // update user

### const updateUser = asyncHandler(async(req, res)=>{

### const user = await User.findById(req.user)

### if(user){

### const { name, email, photo, phone, bio }= user;

### user.email = email,

### user.name = req.body.name || name;

### user.phone = req.body.phone || phone;

### user.bio = req.body.bio || bio;

### user.photo = req.body.photo || photo;

### const updatedUser = await user.save();

### res.status(200).json({

### \_id: updatedUser.\_id,

### name: updatedUser.name,

### email: updatedUser.email,

### photo: updatedUser.photo,

### phone: updatedUser.phone,

### bio: updatedUser.bio,

### })

### }

### else{

### res.status(404)

### throw new Error("User not found")

### }

### });

### /\*const updateUser = async (req, res, next) => {

### // if (req.user.id !== req.params.id) {

### // return next(errorHandler(401, 'You can update only your account!'));

### // }

### try {

### if (req.body.password) {

### req.body.password = bcryptjs.hashSync(req.body.password, 10);

### }

### const updatedUser = await User.findByIdAndUpdate(

### req.params.id,

### {

### $set: {

### name: req.body.name,

### email: req.body.email,

### password: req.body.password,

### // profilePicture: req.body.profilePicture,

### },

### },

### { new: true }

### );

### const { password, ...rest } = updatedUser.\_doc;

### res.status(200).json(rest);

### } catch (error) {

### next(error);

### }

### };\*/

### // update password

### const changePassword = asyncHandler(async(req, res)=>{

### const user = await User.findById(req.user.\_id);

### const {oldPassword, Password} = req.body;

### if(!user){

### res.status(400);

### throw new Error("User not found, please signup");

### }

### //validate

### if(!oldPassword || !Password){

### res.status(400);

### throw new Error("Please add old and new password");

### }

### // check if old password is correct then

### const passwordIscorrect = await bcrypt.compare(oldPassword, user.password)

### //save new password

### if(user && passwordIscorrect){

### user.password = password;

### await user.save()

### res.status(200).send("password change successful")

### }else{

### res.status(400);

### throw new Error("Old password is incorrect")

### }

### });

### // forgot password

### const forgotPassword = asyncHandler(async(req, res)=>{

### const {email} = req.body

### const user = await User.findOne({email})

### if(!user){

### res.status(404)

### throw new Error("User does not exist")

### }

### //delete token if exist in db

### let token = await Token.findOne({userId: user.\_id})

### if(token){

### await token.deleteOne()

### }

### /// create reset token

### let resetToken = crypto.randomBytes(32).toString("hex") + user\_.id

### //hash token before to db

### const hashedToken = crypto.createHash("sah256").update(resetToken).digest("hex")

### console.log(hashedToken);

### // save token to db

### await new Token({

### userId: user.\_id,

### token: hashedToken,

### createAt: Date.now(),

### expiresAt: Date.now() + 30 \* (60 \* 1000) // thirty minutes

### }).save();

### // construct Reset url

### const resetUrl= ${process.env.FRONTEND\_URL}/resetpassword/${resetToken}

### //reset email

### const message = `

### <h2>Hello ${user.name}</h2>

### <p>Please use the url below to reset your password</p>

### <p>This link is vaild only for 30 minutes </p>

### <a href=${reseturl} clicktracking=off>${resetUrl}</a>

### <p>regards</p>`;

### const subject = "Password reset request"

### const send\_to = user.email

### const sent\_from = process.env.EMAIL\_USER

### try{

### await sendEmail(subject, message, send\_to, send\_from)

### res.status(200).json({sucess: trusted, message: "Reset Email sent"})

### }catch(error){

### res.status(500)

### throw new Error("Email not sent, please try again")

### }

### });

### module.exports={

### registerUser,

### loginUser,

### logout,

### getUser,

### loginStatus,

### updateUser,

### changePassword,

### forgotPassword,

### }

## **Product Controller:**

const asyncHandler = require("express-async-handler");

const Product = require("../model/productModel");

// Create new product

const createProduct = asyncHandler(async (req, res) => {

try {

const {name, brand, category, price, quantity, description, expiryDate} = req.body;

const product = new Product({

name, brand, category, price, quantity, description, expiryDate

});

await product.save();

res.status(201).json(product);

} catch (error) {

res.status(400).json({ message: error.message });

}

});

// Get all products

const getAllProducts = asyncHandler(async (req, res) => {

try {

const products = await Product.find();

res.status(200).json(products);

} catch (error) {

res.status(500).json({ message: error.message });

  }

});

const getProductByName = asyncHandler(async (req, res) => {

try {

const productName = req.params.name;

const product = await Product.findOne({ name: productName });

if (!product) {

return res.status(404).json({ message: 'Product not found' });

}

res.status(200).json(product);

} catch (error) {

res.status(500).json({ message: error.message });

  }

});

const updateProductByName = asyncHandler(async (req, res) => {

try {

const productName = req.params.name;

const { name, brand, category, price, quantity, description, expiryDate } = req.body;

const updatedProduct = await Product.findOneAndUpdate(

{ name: productName },

{ name, brand, category, price, quantity, description, expiryDate },

{ new: true } // To return the updated document

);

if (!updatedProduct) {

return res.status(404).json({ message: 'Product not found' });

}

res.status(200).json(updatedProduct);

} catch (error) {

res.status(400).json({ message: error.message });

}

});

// Delete a product

/\*

const deleteProduct = asyncHandler(async (req, res) => {

try {

const {name, brand, category, price, quantity, description, expiryDate} = req.body;

await Product.findByIdAndDelete(req.body);

if (!Product) {

return res.status(404).json({ message: 'Product not found' });

}

res.status(200).json({ message: 'Product deleted successfully' });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

\*/

const deleteProductByName = asyncHandler(async (req, res) => {

try {

const productName = req.params.name;

const deletedProduct = await Product.findOneAndDelete({ name: productName });

if (!deletedProduct) {

return res.status(404).json({ message: 'Product not found' });

}

res.status(200).json({ message: 'Product deleted successfully' });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

module.exports = {

createProduct,

getAllProducts,

getProductByName,

updateProductByName,

deleteProductByName,

};

**Supplier controller.js:**

const asyncHandler = require("express-async-handler");

const Supp = require("../model/suppModel")

// Create a new supplier

const createSupplier = asyncHandler(async (req, res) => {

try {

const { name, agencyname, email, phone } = req.body;

const supplier = new Supp({

name,

agencyname,

email,

phone

});

await supplier.save();

res.status(201).json(supplier);

} catch (error) {

res.status(400).json({ message: error.message });

}

});

// Get all suppliers

const getAllSuppliers = asyncHandler(async (req, res) => {

try {

const suppliers = await Supp.find();

res.status(200).json(suppliers);

} catch (error) {

res.status(500).json({ message: error.message });

  }

});

const getSupplierByName = asyncHandler(async (req, res) => {

try {

const supplierName = req.params.name;

const supplier = await Supp.findOne({ name: supplierName });

if (!supplier) {

return res.status(404).json({ message: 'Supplier not found' });

}

res.status(200).json(supplier);

} catch (error) {

res.status(500).json({ message: error.message });

}

});

// Update a supplier

const updateSupplierByName = asyncHandler(async (req, res) => {

try {

const supplierName = req.params.name;

const { name, agencyname, email, phone } = req.body;

const updatedSupplier = await Supp.findOneAndUpdate(

{ name: supplierName },

{ name, agencyname, email, phone },

{ new: true } // To return the updated document

);

if (!updatedSupplier) {

return res.status(404).json({ message: 'Supplier not found' });

}

res.status(200).json(updatedSupplier);

} catch (error) {

res.status(400).json({ message: error.message });

}

});

// Delete a supplier

const deleteSupplierByName = asyncHandler(async (req, res) => {

try {

const supplierName = req.params.name;

const deletedSupplier = await Supp.findOneAndDelete({ name: supplierName });

if (!deletedSupplier) {

return res.status(404).json({ message: 'Supplier not found' });

}

res.status(200).json({ message: 'Supplier deleted successfully' });

} catch (error) {

res.status(500).json({ message: error.message });

}

});

module.exports = {

createSupplier,

getAllSuppliers,

getSupplierByName,

updateSupplierByName,

deleteSupplierByName

};

## **Frontend:**

## **app.jsx:**

import React ,{ useState, useEffect } from 'react';

import { BrowserRouter as Router, Route, Routes, BrowserRouter } from 'react-router-dom';

import Home from './pages/home';

import Signup from './pages/Signup';

import Login from './pages/Login';

import Header from'./components/Header';

import Dash from './pages/dash';

import User from './pages/User';

import Product from './pages/product';

import Dis from './pages/Dis';

import UpdateProduct from './pages/UpdateProduct';

import Abou from './pages/abou';

import SupplierManagement from './pages/SupplierManagement';

import SuppliersList from './pages/SuppliersList';

import Logout from './pages/LogoutButton'

const App = () => {

return (

<div>

<Routes>

<Route path="/dash" element={<Dash />} />

<Route path="/dashboard" element={<Home />} />

<Route path="/signup" element={<Signup />} />

<Route path="/" element={<Login />} />

<Route path="/user" element={<User />}/>

<Route path="/product" element={<Product />}/>

<Route path="/dis" element={<Dis />}/>

<Route path="/updateproduct" element={<UpdateProduct />}/>

<Route path="/abou" element={<Abou />} />

<Route path="/supplier" element={<SupplierManagement />} />

<Route path="/suppl" element={<SuppliersList />} />

<Route path="/logout" element={<Logout/>} />

</Routes>

</div>

)

}

export default App

## **user.jsx:**

import React, { useState, useEffect } from 'react';

const User = () => {

return (

<div className="p-5 h-screen bg-gray-100">

<div className=''>

<table className='w-full'>

<thead className="bg-gray-50 border-b-2 border-gary-200">

<tr>

<th className=" p-3 text-sm font-bold tracking-wide text-left">Name</th>

<th className=" p-3 text-sm font-bold tracking-wide text-left">Email</th>

<th className=" p-3 text-sm font-bold tracking-wide text-left">Phone</th>

<th className="p-3 text-sm font-bold tracking-wide text-left">Bio</th>

<th className="p-3 text-sm font-bold tracking-wide text-left">Action</th>

</tr>

</thead>

<tbody>

</tbody>

</table>

</div>

</div>

);

}

export default User;

## **login.jsx:**

import React, { useState } from 'react';

import { Link } from 'react-router-dom';

import Header from '../components/Header';

import backgroundImage from '../img/pharmacy.jpg'; // Import the background image

const Login = () => {

const [formData, setFormData] = useState({

email: "",

password: "",

});

const [loading, setLoading] = useState(false);

const [error, setError] = useState(false);

const handleChange = (e) => {

setFormData({ ...formData, [e.target.id]: e.target.value });

}

const handleSubmit = async (e) => {

e.preventDefault();

setLoading(true);

const { email, password } = formData;

try {

const response = await fetch('http://localhost:3300/api/users/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ email, password })

});

const data = await response.json();

if (response.ok) {

console.log('Login successful');

localStorage.setItem("accessToken", data.token);

window.location.href = '/dash';

} else {

console.error('Login failed:', data.message);

setError(true);

}

} catch (error) {

console.error('Error during login:', error);

setError(true);

}

setLoading(false);

};

return (

<div

style={{

backgroundImage: url(${backgroundImage}),

backgroundSize: 'cover',

backgroundPosition: 'center',

minHeight: '100vh',

}}

>

<Header/>

<div className='p-3 max-w-lg mx-auto'>

<h1 className='text-3xl text-center font-bold text-green-500'>Login</h1>

<form onSubmit={handleSubmit} className='flex flex-col gap-4'>

<input type="text" placeholder='Email'id='email' className='bg-slate-100 p-3 rounded-lg' onChange={handleChange} />

<input type="password" placeholder='Password' id='password' className='bg-slate-100 p-3 rounded-lg' onChange={handleChange} />

<button disabled={loading} className='bg-slate-700 text-white p-3 rounded-lg uppercase hover:opacity-95 disabled:opacity-70'>{loading ? 'Loading...' : 'Login'}</button>

</form>

<div className="flex gap-2 mt-5">

<p>Don't Have an account?</p>

<Link to='/signup'>

<span className='text-red-800'>Sign Up</span>

</Link>

</div>

{error && <p className='text-red-700 mt-5'>Invalid username or password</p>}

</div>

</div>

)

}

export default Login;

## **signup:**

import React from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useState } from 'react';

import Header from '../components/Header';

import backgroundImage from '../img/pharmacy.jpg'; // Import the background image

const Signup = () => {

const [formData, setFormData] = useState({

name: "",

email: "",

password: "",

phone: "",

});

const [error, setError] = useState(false);

const [loading, setLoading] = useState(false);

const navigate = useNavigate();

const handleChange = (e) => {

setFormData({...formData, [e.target.id]: e.target.value});

};

const handleSubmit = async (e) => {

e.preventDefault();

try {

setLoading(true);

setError(false);

const res = await fetch('http://localhost:3300/api/users/register', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(formData),

});

const data = await res.json();

console.log(data); // {message: 'user created successfully'};

setLoading(false);

if (data.success === false) {

setError(true);

return;

}

navigate('/');

} catch(error) {

setLoading(false);

setError(true);

}

};

return (

<div

style={{

backgroundImage: url(${backgroundImage}),

backgroundSize: 'cover',

backgroundPosition: 'center',

minHeight: '100vh', // Ensure background covers entire viewport height

}}

>

<Header />

<div className='p-3 max-w-lg mx-auto'>

<h1 className='text-3xl text-center font-bold text-green-500'>Sign Up</h1>

<form onSubmit={handleSubmit} className='flex flex-col gap-4'>

<input type="text" placeholder='Name' id='name' className='bg-slate-100 p-3 rounded-lg' onChange={handleChange}/>

<input type="text" placeholder='Email' id='email' className='bg-slate-100 p-3 rounded-lg' onChange={handleChange}/>

<input type="text" placeholder='Password' id='password'className='bg-slate-100 p-3 rounded-lg' onChange={handleChange}/>

<input type="text" placeholder='Phone' id='phone' className='bg-slate-100 p-3 rounded-lg' onChange={handleChange}/>

<button disabled={loading} className='bg-slate-700 text-white p-3 rounded-lg uppercase hover:opacity-95 disabled:opacity-70'>{loading ? 'Loading...' : 'Sign Up'}</button>

</form>

<div className="flex gap-2 mt-5">

<p>Have an account?</p>

<Link to='/'>

<span className='text-green-500'>Login</span>

</Link>

</div>

<p className='text-red-700 mt-5'>{error && 'Name or Email already Exits!' }</p>

</div>

</div>

);

};

export default Signup;

**dash.jsx:**

import React, { useState, useEffect } from 'react';

import Sidbar from './Sidbar'; // Import the Sidebar component

const Dis = () => {

const [products, setProducts] = useState([]);

const [searchTerm, setSearchTerm] = useState('');

useEffect(() => {

const fetchProducts = async () => {

try {

const response = await fetch('http://localhost:3300/api/products');

if (!response.ok) {

throw new Error('Failed to fetch products');

}

const data = await response.json();

setProducts(data);

} catch (error) {

console.error('Error fetching products:', error);

// Handle error, show error message to the user, etc.

}

};

fetchProducts();

}, []);

const handleSearch = event => {

setSearchTerm(event.target.value);

};

const filteredProducts = products.filter(product =>

product.name.toLowerCase().includes(searchTerm.toLowerCase())

);

return (

<div className="flex relative bg-green-100 h-screen">

<Sidbar /> {/\* Include the Sidebar component \*/}

<div className="max-w-4xl mx-auto mt-8">

<h1 className="text-2xl font-bold mb-4">Products</h1>

<input

type="text"

placeholder="Search product..."

value={searchTerm}

onChange={handleSearch}

className="border border-gray-300 rounded-md px-3 py-2 mt-4 w-full" // Adjusted margin here

/>

<table className="min-w-full divide-y divide-gray-200 mt-4"> {/\* Added margin-top here \*/}

<thead className="bg-green-400">

<tr>

<th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Name</th>

<th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Brand</th>

<th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Category</th>

<th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Price</th>

<th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Quantity</th>

<th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Description</th>

<th className="px-6 py-3 text-left text-xs font-medium text-gray-500 uppercase tracking-wider">Expiry Date</th>

</tr>

</thead>

<tbody className="bg-white divide-y divide-gray-200">

{filteredProducts.map(product => (

<tr key={product.\_id}>

<td className="px-6 py-4 whitespace-nowrap">{product.name}</td>

<td className="px-6 py-4 whitespace-nowrap">{product.brand}</td>

<td className="px-6 py-4 whitespace-nowrap">{product.category}</td>

<td className="px-6 py-4 whitespace-nowrap">{product.price}</td>

<td className="px-6 py-4 whitespace-nowrap">{product.quantity}</td>

<td className="px-6 py-4 whitespace-nowrap">{product.description}</td>

<td className="px-6 py-4 whitespace-nowrap">{product.expiryDate}</td>

</tr>

))}

</tbody>

</table>

</div>

</div>

);

};

export default Dis;

**sidebar.jsx:**

import React from 'react'

import User from './User';

import { FaHome } from "react-icons/fa";

import { CiLogin } from "react-icons/ci";

import { SiGnuprivacyguard } from "react-icons/si";

import { FaUserCircle } from "react-icons/fa";

import { FcAbout } from "react-icons/fc";

import { GiMedicinePills } from "react-icons/gi";

import { GiMedicines } from "react-icons/gi";

import { GrUpdate } from "react-icons/gr";

import { FaShop } from "react-icons/fa6";

import { LuClipboardList } from "react-icons/lu";

import { RiLogoutBoxLine } from "react-icons/ri";

import LogoutButton from './LogoutButton';

const Sidbar = () => {

return (

<div className='w-64 bg-slate-700 fixed h-full px-4 py-2'>

<div className='my-2 mb-4'>

<h1 className='text-2x text-white font-bold'>Admin Dashboard</h1>

</div>

<hr />

<ul>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/dash">

<FaHome className='inline-block w-6 h-6 mr-2 -mt-2' > </FaHome>

Home

</a>

</li>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/">

<CiLogin className='inline-block w-6 h-6 mr-2 -mt-2' > </CiLogin >

Login

</a>

</li>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/signup">

<SiGnuprivacyguard className='inline-block w-6 h-6 mr-2 -mt-2' > </SiGnuprivacyguard >

Sign Up

</a>

</li>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/user">

<FaUserCircle className='inline-block w-6 h-6 mr-2 -mt-2' > </FaUserCircle >

User

</a>

</li>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/product">

<GiMedicinePills className='inline-block w-6 h-6 mr-2 -mt-2' > </GiMedicinePills >

Add Product

</a>

</li>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/dis">

<GiMedicines className='inline-block w-6 h-6 mr-2 -mt-2' > </GiMedicines >

List Product

</a>

</li>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/updateproduct">

<GrUpdate className='inline-block w-6 h-6 mr-2 -mt-2' > </GrUpdate >

UpdateProduct

</a>

</li>

<li className='mb-2 rounded hover:shodow hover:bg-green-500 py-2'>

<a className='text-white font-bold 'href="/supplier">

<FaShop className='inline-block w-6 h-6 mr-2 -mt-2' > </FaShop >

Supplier

</a>

</li>

<li className='mb-2 rounded hover:shodow py-2 hover:bg-green-500'>

<a className='text-white font-bold 'href="/suppl">

<LuClipboardList className='inline-block w-6 h-6 mr-2 -mt-2' > </LuClipboardList >

SupplierList

</a>

</li>

<li className='mb-2 rounded py-2'>

<a className='text-white font-bold '>

<RiLogoutBoxLine className='inline-block w-6 h-6 mr-2 -mt-2 ' > </RiLogoutBoxLine >

<LogoutButton/>

</a>

</li>

</ul>

</div>

)

}

export default Sidbar

**navbar.jsx:**

import React from 'react';

import { Link } from 'react-router-dom';

import LogoutButton from './LogoutButton';

import { FaKitMedical } from "react-icons/fa6";

import { FaSignInAlt } from "react-icons/fa";

const Navbar = () => {

return (

<nav className="bg-slate-700 px-4 py-4 flex justiy-center ml-64">

<div className="max-w-7xl mx-auto px-4 flex justify-between items-center ml-64">

<div>

<Link to="/" className="text-white font-bold text-lg"> MEDCARE</Link>

</div>

<div className="flex space-x-2 justify-center ml-96">

<Link to="/product" className="text-gray-300 hover:bg-green-500 hover:text-white px-3 py-2 rounded-md text-sm font-medium">AddProducts</Link>

<Link to="/updateproduct" className="text-gray-300 hover:bg-green-500 hover:text-white px-3 py-2 rounded-md text-sm font-medium">UpdateProduct</Link>

<Link to="/dis" className="text-gray-300 hover:bg-green-500 hover:text-white px-3 py-2 rounded-md text-sm font-medium">ListProduct</Link>

<Link to="/dash" className="text-gray-300 hover:bg-green-500 hover:text-white px-3 py-2 rounded-md text-sm font-medium">Home</Link>

</div>

<div className="md:hidden">

<button className="text-gray-300 hover:bg-gray-700 hover:text-white px-3 py-2 rounded-md text-sm font-medium">Menu</button>

</div>

</div>

</nav>

);

};

export default Navbar;

**createproduct.jsx:**

import React, { useState } from 'react';

import Sidbar from './Sidbar';

import Navbar from './Navbar';

const CreateProduct = () => {

const [product, setProduct] = useState({

name: '',

brand: '',

category: '',

price: 0,

quantity: 0,

description: '',

expiryDate: ''

});

const handleChange = e => {

const { name, value } = e.target;

setProduct(prevProduct => ({

...prevProduct,

[name]: value

}));

};

const handleSubmit = async e => {

e.preventDefault();

try {

const response = await fetch('http://localhost:3300/api/products', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(product)

});

if (!response.ok) {

throw new Error('Failed to create product');

}

const data = await response.json();

console.log('Product created:', data);

// Optionally, reset the form fields after successful submission

setProduct({

name: '',

brand: '',

category: '',

price: 0,

quantity: 0,

description: '',

expiryDate: ''

});

} catch (error) {

console.error('Error creating product:', error);

// Handle error, show error message to the user, etc.

}

};

return (

<div >

<Sidbar/>

<Navbar/>

<div className="flex justify-center items-center bg-green-100 h-screen">

<form onSubmit={handleSubmit} className="w-full max-w-md bg-white p-8 rounded-lg shadow-lg">

<h2 className="text-3xl font-bold mb-6 text-center text-green-500">Add Product</h2>

<div className="mb-4 flex justify-center ">

<label htmlFor="name" className="block text-sm font-medium text-gray-600"></label>

<input type="text" name="name" id="name" value={product.name} onChange={handleChange} placeholder="name" required className="input-field rounded-lg w-80" />

</div>

<div className="mb-4 flex justify-center ">

<label htmlFor="brand" className="block text-sm font-medium text-gray-600"></label>

<input type="text" name="brand" id="brand" value={product.brand} onChange={handleChange} placeholder="brand" required className="input-field rounded-lg w-80" />

</div>

<div className="mb-4 flex justify-center ">

<label htmlFor="category" className="block text-sm font-medium text-gray-600"></label>

<input type="text" name="category" id="category" value={product.category} onChange={handleChange} placeholder="category" required className="input-field rounded-lg w-80" />

</div>

<div className="mb-4 flex justify-center ">

<label htmlFor="price" className="block text-sm font-medium text-gray-600"></label>

<input type="number" name="price" id="price" value={product.price} onChange={handleChange} placeholder="price" required className="input-field rounded-lg w-80" />

</div>

<div className="mb-4 flex justify-center ">

<label htmlFor="quantity" className="block text-sm font-medium text-gray-600"></label>

<input type="number" name="quantity" id="quantity" value={product.quantity} onChange={handleChange} placeholder="quantity" required className="input-field rounded-lg w-80 " />

</div>

<div className="mb-4 flex justify-center">

<label htmlFor="description" className="block text-sm font-medium text-gray-600"></label>

<input type="text" name="description" id="description" value={product.description} onChange={handleChange}

<div className="mb-4 flex justify-center ">

<label htmlFor="expiryDate" className="block text-sm font-medium text-gray-600"></label>

<input type="date" name="expiryDate" id="expiryDate" value={product.expiryDate} onChange={handleChange} placeholder="Expiry date" required className="input-field rounded-lg w-80" />

</div>

<button type="submit" className="bg-green-500 hover:bg-slate-700 text-white font-bold py-2 px-4 rounded w-full focus:outline-none focus:shadow-outline mt-6">Add Product</button>

</form>

</div>

</div>

)

};

export default CreateProduct;

**supplier.jsx:**

import React, { useState } from 'react';

import Sidbar from './Sidbar';

import Navbar from './Navbar';

const SupplierManagement = () => {

const [name, setName] = useState('');

const [agencyName, setAgencyName] = useState('');

const [email, setEmail] = useState('');

const [phone, setPhone] = useState('');

const handleCreateSupplier = async (e) => {

e.preventDefault();

try {

await fetch('http://localhost:3300/api/suppliers', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ name, agencyname: agencyName, email, phone }),

});

setName('');

setAgencyName('');

setEmail('');

setPhone('');

} catch (error) {

console.error('Error creating supplier:', error);

}

};

return (

<div className='bg-green-100 h-screen'>

<Sidbar/>

<Navbar/>

<div className="p-3 max-w-lg mx-auto">

<h1 className="text-2xl font-bold mb-4 text-center text-green-500">Supplier Management</h1>

<form onSubmit={handleCreateSupplier} className="flex flex-col gap-4">

<input type="text" placeholder="Name" value={name} onChange={(e) => setName(e.target.value)} className="bg-slate-100 p-3 rounded-lg" />

<input type="text" placeholder="Agency Name" value={agencyName} onChange={(e) => setAgencyName(e.target.value)} className="bg-slate-100 p-3 rounded-lg" />

<input type="email" placeholder="Email" value={email} onChange={(e) => setEmail(e.target.value)} className="bg-slate-100 p-3 rounded-lg" />

<input type="text" placeholder="Phone" value={phone} onChange={(e) => setPhone(e.target.value)} className="bg-slate-100 p-3 rounded-lg" />

<button type="submit" className="mt-4 text-white px-4 py-2 bg-green-500 p-3 rounded-lg">Create Supplier</button>

</form>

</div>

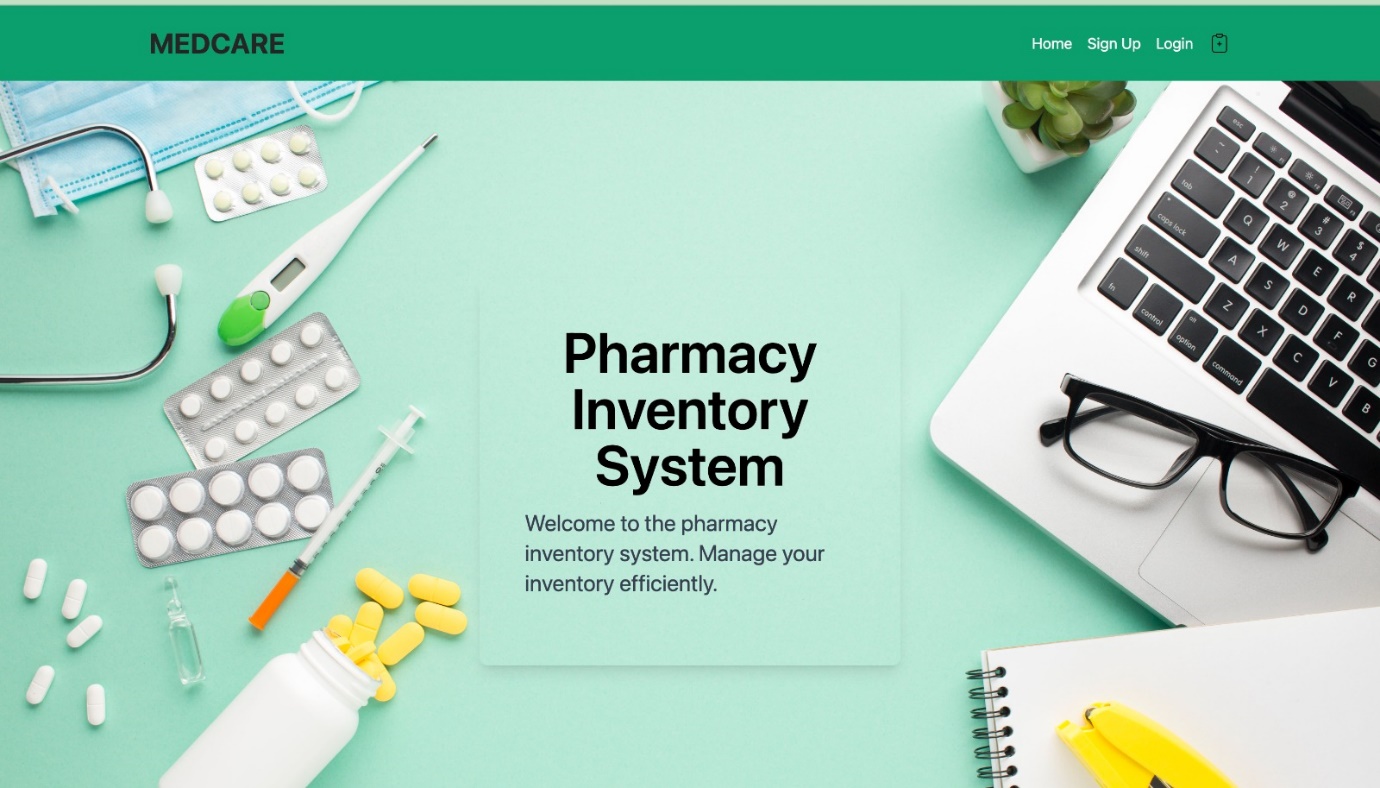
</div>

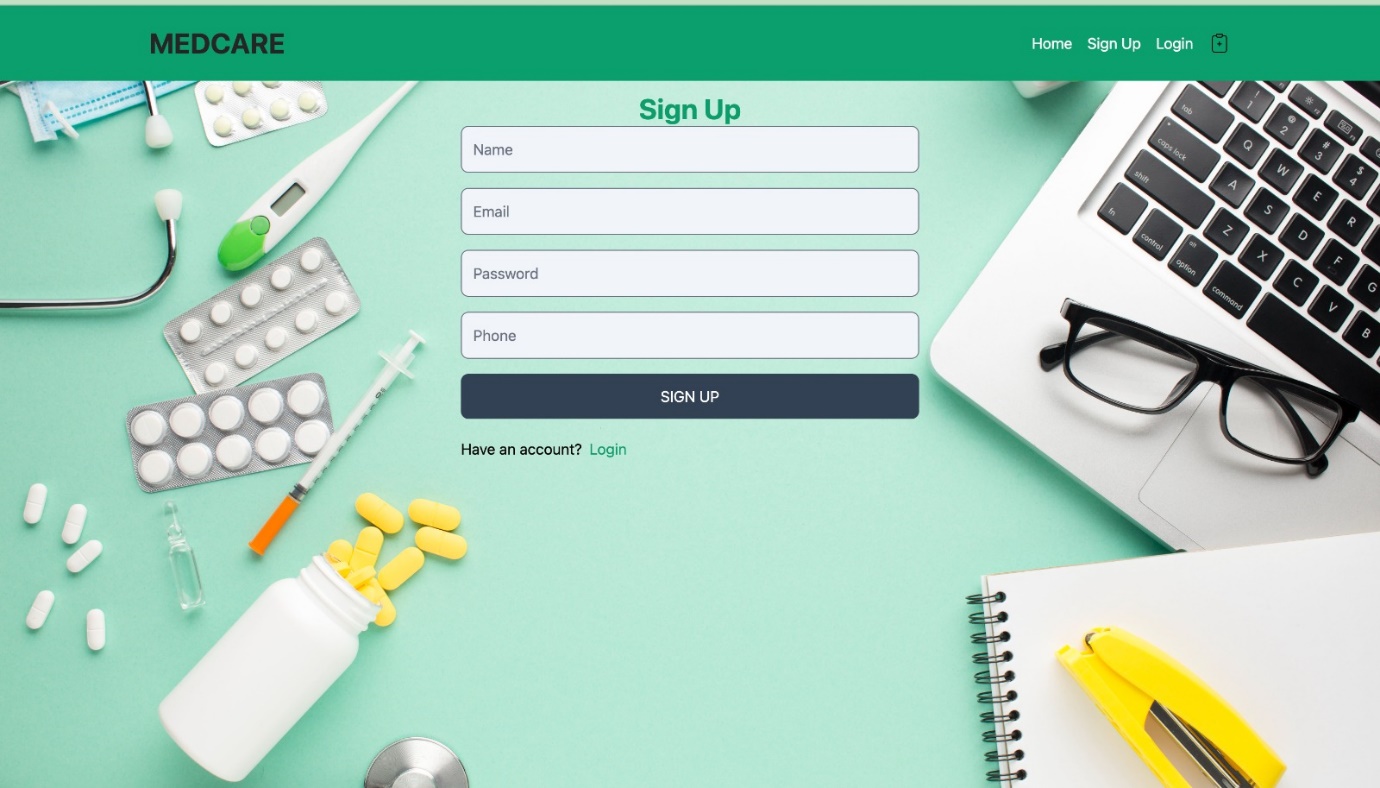
);

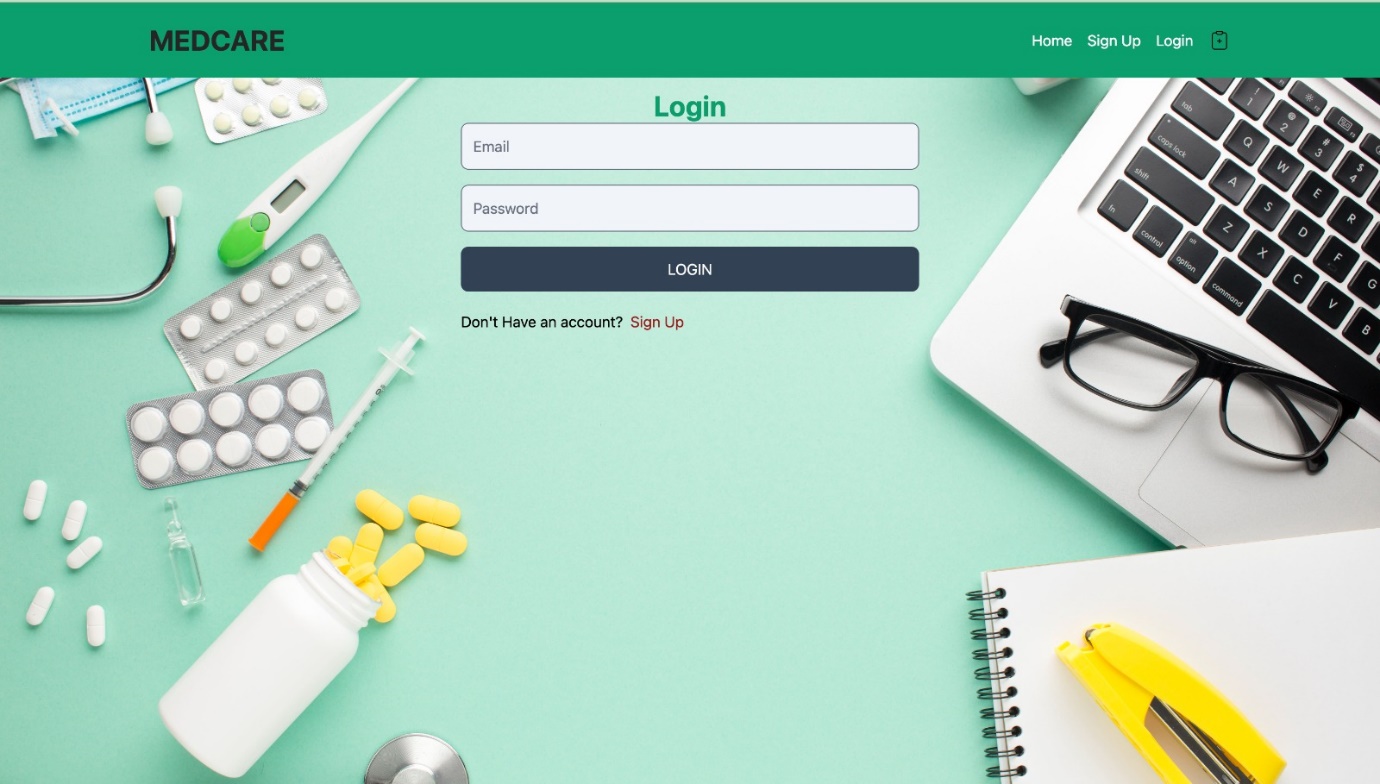
};

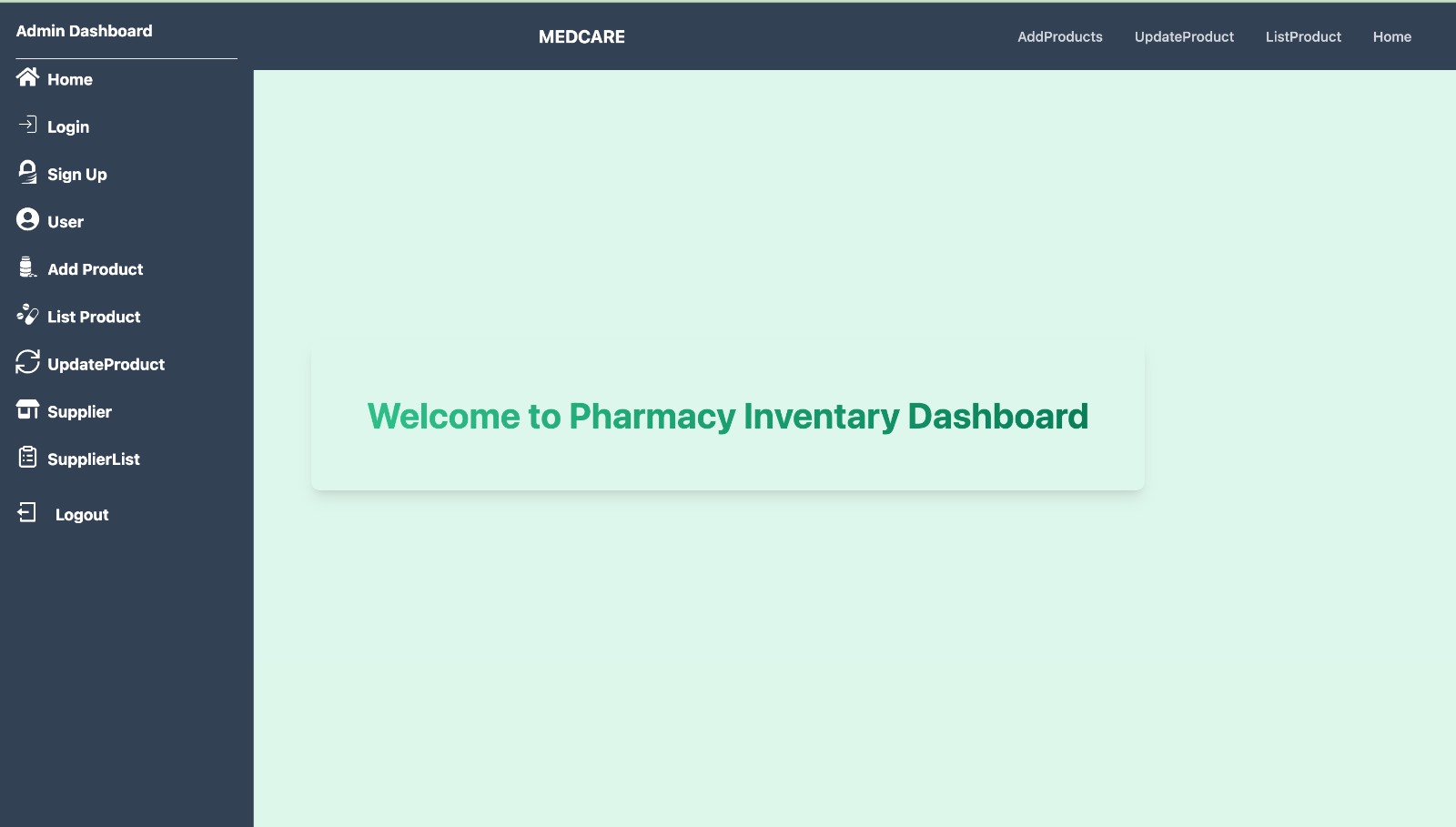
export default SupplierManagement;

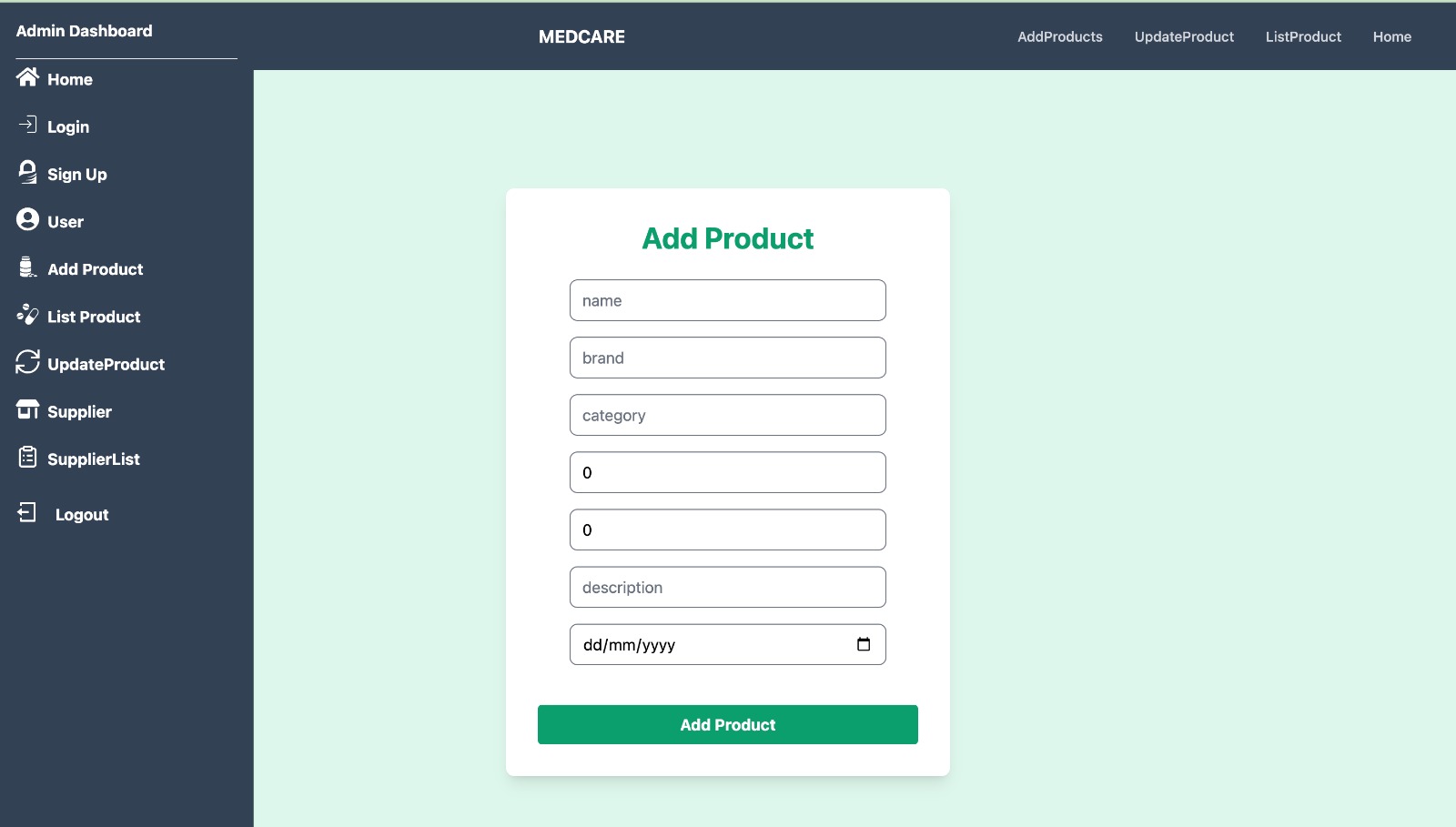
**Result:**

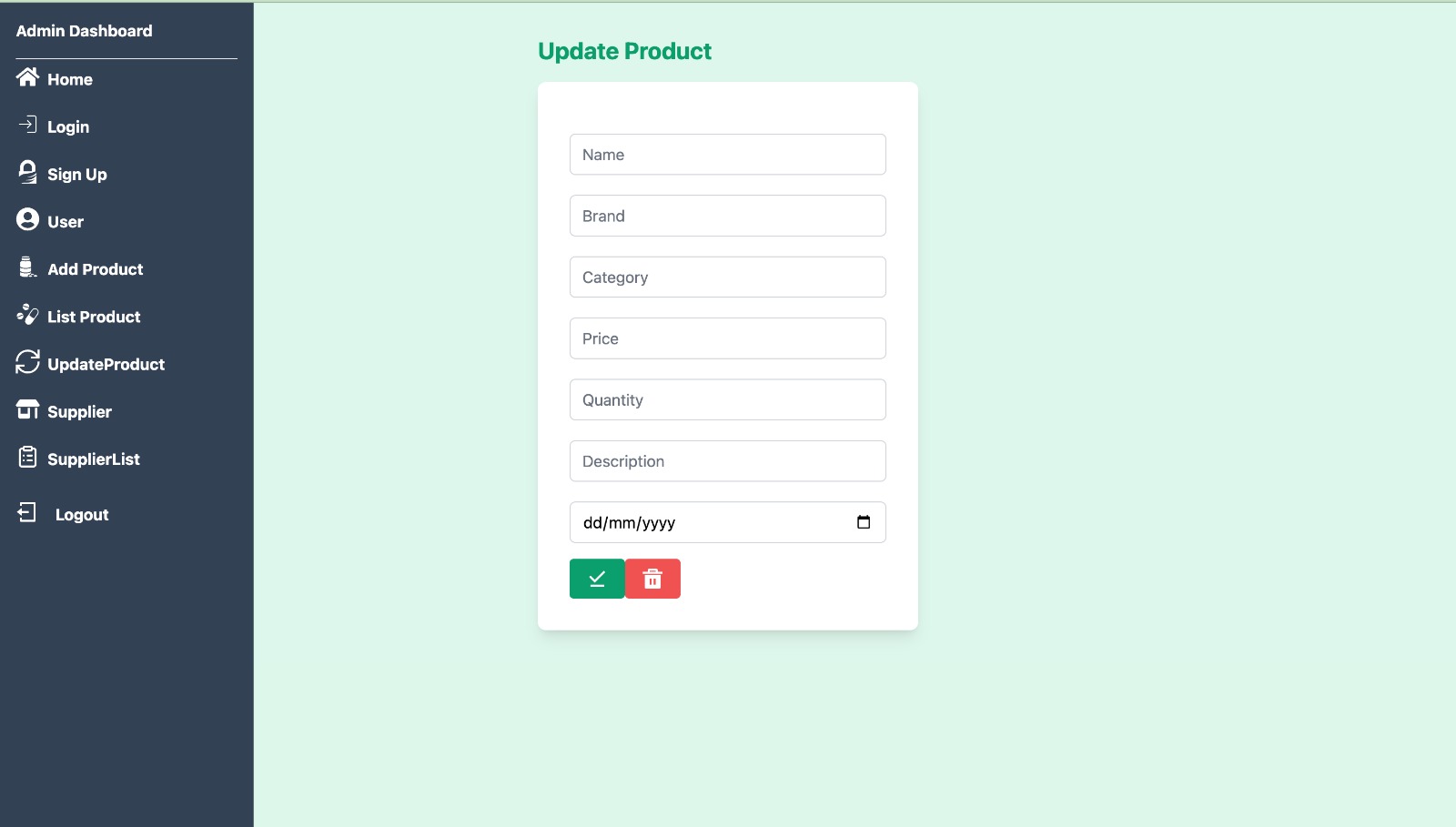


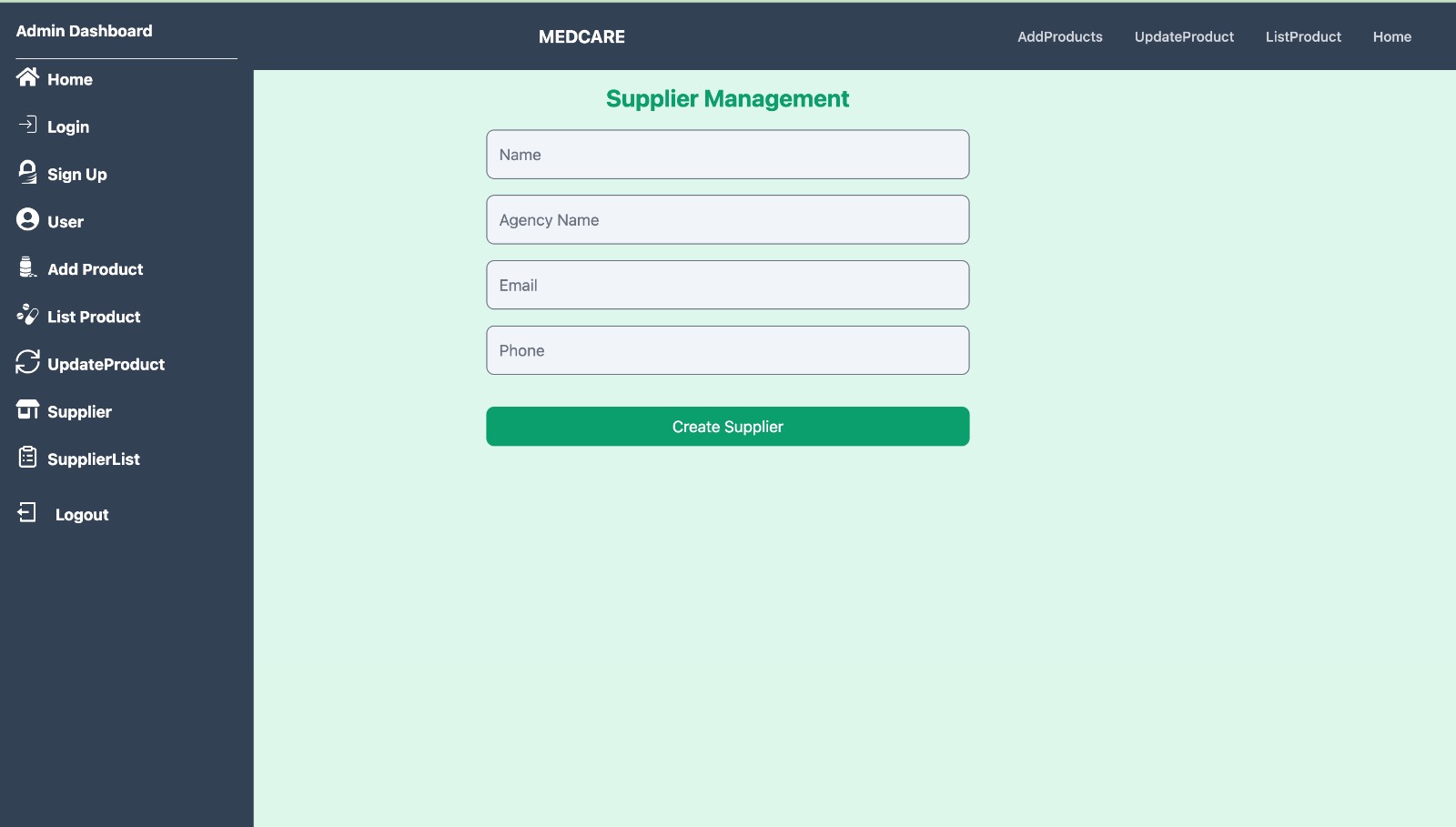












**Conclusion:**

In conclusion, the proposed Pharmacy Inventory Management System (PIMS) built on the MERN stack stands as a robust solution tailored to meet the specific needs of pharmacies. By harnessing the combined power of MongoDB, Express.js, React, and Node.js, we have developed a platform that not only simplifies inventory management but also enhances efficiency and accuracy in handling pharmaceutical products.

With its intuitive interface and comprehensive features, PIMS empowers pharmacists to effortlessly manage their inventory by adding, updating, and removing products, all while ensuring vital information such as quantity, expiry dates, and prices are readily accessible. Moreover, the implementation of user authentication and role-based access control adds an extra layer of security, guaranteeing that sensitive data remains protected and only accessible to authorized personnel.

## **Reference:**

<https://www.youtube.com/playlist?list=PL4cUxeGkcC9iJ_KkrkBZWZRHVwnzLIoUE>

<https://docs.mongodb.com/>

<https://react.dev/>

<https://expressjs.com/en/starter/installing.html>

<https://tailwindcss.com/>

<https://react-icons.github.io/react-icons/>