**Front end**

**Main.jsx**

import { StrictMode } from 'react';

import { createRoot } from 'react-dom/client';

import App from './App.jsx';

import Layout from './components/Layout.jsx';

import Home from './components/Home.jsx';

import Loan from './components/laonpayments.jsx';

import Notification from './components/Notification.jsx';

import { createBrowserRouter, RouterProvider, Navigate } from 'react-router-dom';

import AdminLayout from './AdminComponents/adminLayout.jsx';

import AdminHome from './AdminComponents/adminHome.jsx';

import Users from './AdminComponents/Users.jsx';

import PendingLoans from './AdminComponents/pendingLoans.jsx';

import DisbursedLoans from './AdminComponents/disbursedloans.jsx';

import CancelledLoans from './AdminComponents/cancelledloans.jsx';

import Overdue from './AdminComponents/overdue.jsx';

import Addnewloanproduct from './AdminComponents/addNewproduct.jsx';

import Login from './login.jsx';

import Signup from './signup.jsx';

import ApplyLoan from './components/applyloan.jsx';

function ErrorFallback() {

  return <div>Something went wrong. Please try again later.</div>;

}

const router = createBrowserRouter([

  {

    path: '/',

    element: <Navigate to="/login" replace />, // Default route to login

  },

  {

    path: '/login',

    element: <Login />,

  },

  {

    path: '/signup',

    element: <Signup />,

  },

  {

    path: '/customer',

    element: <Layout />, // Ensure Layout renders <Outlet />

    children: [

      {

        path: '',

        element: <Home />, // Changed from '/' to ''

      },

      {

        path: 'loan',

        element: <ApplyLoan/>,

      },

      {

        path: 'notifications',

        element: <Notification />,

      },

    ],

  },

  {

    path: '/admin',

    element: <AdminLayout />, // Ensure AdminLayout renders <Outlet />

    children: [

      {

        path: '',

        element: <AdminHome />, // Changed from '/' to ''

      },

      {

        path: 'users',

        element: <Users />,

      },

      {

        path: 'pendingloans',

        element: <PendingLoans />,

      },

      {

        path: 'disbursedloans',

        element: <DisbursedLoans />,

      },

      {

        path: 'cancelledloans',

        element: <CancelledLoans />,

      },

      {

        path: 'overdue',

        element: <Overdue />,

      },

      {

        path: 'newloan',

        element: <Addnewloanproduct />,

      },

    ],

  },

]);

createRoot(document.getElementById('root')).render(

  <StrictMode>

    <RouterProvider

      router={router}

      fallbackElement={<ErrorFallback />} // Add fallback for errors

    />

  </StrictMode>

);

**AdminLayout**

**import React, { useState } from "react";**

**import { Outlet, useSearchParams } from 'react-router-dom';**

**import AdminHeader from "./adminHeader";**

**import AdminDashboard from "./adminDashboard";**

**import AdminFooter from "./adminFooter";**

**function AdminLayout(){**

**const [isOpen,setIsOpen]=useState(false)**

**const togglesidebar=()=>{**

**setIsOpen(!isOpen)**

**}**

**return(**

**<>**

**<div className="d-flex flex-column vh-100 " >**

**<AdminHeader togglesidebar={togglesidebar}/>**

**<div className="d-flex flex-grow-1">**

**<AdminDashboard isOpen={isOpen}/>**

**<div className="flex-grow-1 p-3 overflow-auto">**

**<Outlet/>**

**</div>**

**</div>**

**<AdminFooter/>**

**</div>**

**</>**

**)**

**}**

**export default AdminLayout;**

**DashBoard**

import React from "react";

import 'bootstrap/dist/css/bootstrap.min.css';

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

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

import {AiFillHome} from 'react-icons/ai'

import {HiOutlineClipboardList} from 'react-icons/hi'

import {FaUser,FaRegTimesCircle,FaUserCircle} from 'react-icons/fa'

import {MdOutlineCreditCard} from 'react-icons/md'

import {FiBell} from 'react-icons/fi'

export default function AdminDashboard({isOpen}) {

    return (

        <div className={`${isOpen?"d-sm-inline-grid":"d-none"}  d-sm-inline-grid`}>

            <nav className="col-12 sidebar d-grid" style={{backgroundColor:"#F9FAFB"}}>

                <div className="position-sticky">

                    <ul className="nav d-flex flex-column">

                        <li className="nav-item p-3 m-2 d-flex align-items-center gap-2"><AiFillHome style={{color:"60A5FA"}}/><Link to='/admin' style={{textDecoration:"none",color:"#1E3A8A"}}>Home</Link></li>

                        <li className="nav-item p-3 m-2 d-flex align-items-center gap-2"><FaUser style={{color:"60A5FA"}}/><Link to='/admin/users' style={{textDecoration:"none",color:"#1E3A8A"}}>Users</Link></li>

                        <li className="nav-item p-3 m-2 d-flex align-items-center gap-2"><HiOutlineClipboardList style={{color:"60A5FA"}}/><Link to='/admin/pendingloans' style={{textDecoration:"none",color:"#1E3A8A"}}>Pending Loans  </Link></li>

                        <li className="nav-item p-3 m-2 d-flex align-items-center gap-2"><MdOutlineCreditCard style={{color:"60A5FA"}}/><Link to='/admin/disbursedloans' style={{textDecoration:"none",color:"#1E3A8A"}}>Disbursed Loans</Link></li>

                        <li className="nav-item p-3 m-2 d-flex align-items-center gap-2"><FaRegTimesCircle style={{color:"60A5FA"}}/><Link to='/admin/cancelledloans' style={{textDecoration:"none",color:"#1E3A8A"}}>Cancelled Loans</Link></li>

                        <li className="nav-item p-3 m-2 d-flex align-items-center gap-2"><FiBell style={{color:"60A5FA"}}/><Link to='/admin/overdue' style={{textDecoration:"none",color:"#1E3A8A"}}>Overdue Management</Link></li>

                        <li className="nav-item p-3 m-2 d-flex align-items-center gap-2"><FaUserCircle style={{color:"60A5FA"}}/><Link to='/admin/newloan' style={{textDecoration:"none",color:"#1E3A8A"}}>Tickets </Link></li>

                    </ul>

                </div>

            </nav>

        </div>

    )

}

**Admin Profile**

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

**function AdminProfile() {**

**const [userDetails, setUserDetails] = useState(null);**

**useEffect(() => {**

**// Retrieve admin details from session storage**

**const user = JSON.parse(sessionStorage.getItem('userDetails'));**

**console.log('Retrieved admin details from session storage:', user); // Debugging log**

**setUserDetails(user);**

**}, []);**

**if (!userDetails) {**

**return <div>Loading...</div>;**

**}**

**return (**

**<div className="container mt-5">**

**<h2>Admin Profile</h2>**

**<p><strong>Name:</strong> {userDetails.fullName}</p>**

**<p><strong>Email:</strong> {userDetails.email}</p>**

**<p><strong>Role:</strong> {userDetails.role}</p>**

**</div>**

**);**

**}**

**export default AdminProfile;**

**Disbursed Loan**

**import React, { useEffect, useState } from "react";**

**import 'bootstrap/dist/css/bootstrap.min.css';**

**import { BsTelephone } from 'react-icons/bs';**

**import Home1 from '../components/assets/home.png';**

**import axios from 'axios';**

**export default function DisbursedLoans() {**

**const [userData, setUserData] = useState([]);**

**const [showModal, setShowModal] = useState(false);**

**const [repaymentDetails, setRepaymentDetails] = useState(null);**

**useEffect(() => {**

**const fetchApprovedLoans = async () => {**

**try {**

**const response = await axios.get('http://localhost:8000/api/loans/approved');**

**if (response.status === 200 && response.data) {**

**setUserData(response.data.data || []);**

**} else {**

**console.error('Unexpected response:', response);**

**}**

**} catch (error) {**

**console.error('Error fetching approved loans:', error);**

**}**

**};**

**fetchApprovedLoans();**

**}, []);**

**const calculateEMI = (loanAmount, interestRate, loanDuration) => {**

**const principal = loanAmount;**

**const monthlyRate = interestRate / 12 / 100;**

**const months = parseInt(loanDuration); // Loan duration is already in months**

**const emi = (principal \* monthlyRate \* Math.pow(1 + monthlyRate, months)) / (Math.pow(1 + monthlyRate, months) - 1);**

**return emi.toFixed(2);**

**};**

**const handleRepaymentSchedule = (loan) => {**

**const emi = calculateEMI(loan.loanAmount, loan.interestRate, loan.loanDuration);**

**const months = parseInt(loan.loanDuration); // Loan duration is already in months**

**const acceptedDate = new Date(loan.acceptedDate); // Get the accepted date from the backend**

**const schedule = Array.from({ length: months }, (\_, i) => {**

**const dueDate = new Date(acceptedDate);**

**dueDate.setMonth(dueDate.getMonth() + i); // Calculate the due date for each month**

**return {**

**month: i + 1,**

**emi,**

**dueDate: dueDate.toLocaleDateString(), // Format the due date**

**paidDate: loan.emiPaidDates[i] || null, // Fetch paid dates from backend**

**};**

**});**

**setRepaymentDetails({ loan, schedule });**

**setShowModal(true);**

**};**

**const handlePayEMI = async (loanId, emiAmount, monthIndex) => {**

**try {**

**const response = await axios.post(`http://localhost:8000/api/loans/${loanId}/pay-emi`, { emiAmount });**

**alert(response.data.message);**

**// Update the repayment schedule locally**

**setRepaymentDetails((prevDetails) => {**

**const updatedSchedule = prevDetails.schedule.map((item, index) => {**

**if (index === monthIndex) {**

**return { ...item, paidDate: new Date().toLocaleDateString() }; // Mark as paid**

**}**

**return item;**

**});**

**return { ...prevDetails, schedule: updatedSchedule };**

**});**

**Backend**

**userControllers**

**import { asyncHandler } from "../utils/asyncHandler.js";**

**import {ApiError} from "../utils/ApiError.js"**

**import { User} from "../models/user.model.js"**

**import {uploadOnCloudinary} from "../utils/cloudinary.js"**

**import { ApiResponse } from "../utils/ApiResponse.js";**

**import jwt from "jsonwebtoken"**

**import mongoose from "mongoose";**

**const generateAccessAndRefereshTokens = async(userId) =>{**

**try {**

**const user = await User.findById(userId)**

**const accessToken = user.generateAccessToken()**

**const refreshToken = user.generateRefreshToken()**

**user.refreshToken = refreshToken**

**await user.save({ validateBeforeSave: false })**

**return {accessToken, refreshToken}**

**} catch (error) {**

**throw new ApiError(500, "Something went wrong while generating referesh and access token")**

**}**

**}**

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

**const { fullName, email, password, phone, address, role } = req.body;**

**console.log('Incoming registration request:', req.body); // Debug log**

**const missingFields = [];**

**if (!fullName?.trim()) missingFields.push('fullName');**

**if (!email?.trim()) missingFields.push('email');**

**if (!password?.trim()) missingFields.push('password');**

**if (!phone?.trim()) missingFields.push('phone');**

**if (!address?.trim()) missingFields.push('address');**

**if (missingFields.length > 0) {**

**throw new ApiError(400, `The following fields are required: ${missingFields.join(', ')}`);**

**}**

**const normalizedEmail = email.trim().toLowerCase();**

**const normalizedFullName = fullName.trim().toLowerCase();**

**const existingUser = await User.findOne({**

**email: normalizedEmail**

**});**

**if (existingUser) {**

**throw new ApiError(409, 'User with this email or username already exists');**

**}**

**try {**

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

**fullName: normalizedFullName,**

**email: normalizedEmail,**

**password,**

**phone,**

**address,**

**role,**

**});**

**console.log(user);**

**const createdUser = await User.findOne({ email: normalizedEmail }).select('-password -refreshToken');**

**const options = {**

**httpOnly: true,**

**secure: true,**

**};**

**res**

**.status(201)**

**.cookie('accessToken', user.generateAccessToken(), options)**

**.cookie('refreshToken', user.generateRefreshToken(), options)**

**.json(new ApiResponse(201, createdUser, 'User registered successfully'));**

**} catch (error) {**

**if (error.code === 11000) {**

**console.error('Duplicate key error:', error.keyValue);**

**throw new ApiError(409, `Duplicate key error: ${JSON.stringify(error.keyValue)}`);**

**}**

**console.error('Registration failed:', error);**

**throw new ApiError(500, 'Internal Server Error');**

**}**

**});**

**export const loginUser = asyncHandler(async (req, res) => {**

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

**console.log('Login request received:', { email });**

**if (!email || !password) {**

**throw new ApiError(400, 'Email and password are required');**

**}**

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

**if (!user) {**

**throw new ApiError(404, 'Invalid credentials');**

**}**

**const isPasswordValid = await user.isPasswordCorrect(password);**

**if (!isPasswordValid) {**

**throw new ApiError(401, 'Invalid credentials');**

**}**

**const accessToken = user.generateAccessToken();**

**console.log('Access token generated:', accessToken);**

**console.log('User details:', user);**

**const options = {**

**httpOnly: true,**

**secure: false,**

**sameSite: 'Lax',**

**};**

**res**

**.status(200)**

**.cookie('accessToken', accessToken, options)**

**.json(new ApiResponse(200, { user }, 'Login successful'));**

**});**

**// Get user profile**

**export const getUserProfile = asyncHandler(async (req, res) => {**

**const user = await User.findOne({ role: "customer" }).select("-password -refreshToken");**

**if (!user) {**

**throw new ApiError(404, "User not found");**

**}**

**res.status(200).json(new ApiResponse(200,user, "User profile fetched successfully"));**

**});**

**// Get admin profile**

**export const getAdminProfile = asyncHandler(async (req, res) => {**

**const admin = await User.findOne({ role: "admin" }).select("-password -refreshToken");**

**if (!admin) {**

**throw new ApiError(404, "Admin not found");**

**}**

**res.status(200).json(new ApiResponse(200, admin, "Admin profile fetched successfully"));**

**});**

**// Get current user details**

**export const getCurrentUser = asyncHandler(async (req, res) => {**

**const user = req.user; // User is already set by verifyJWT middleware**

**if (!user) {**

**throw new ApiError(404, "User not found");**

**}**

**res.status(200).json(new ApiResponse(200, user, "Current user details fetched successfully"));**

**});**

**// Logout user**

**export const logoutUser = asyncHandler(async (req, res) => {**

**res**

**.status(200)**

**.clearCookie('accessToken', { httpOnly: true, secure: true })**

**.clearCookie('refreshToken', { httpOnly: true, secure: true })**

**.json(new ApiResponse(200, {}, 'Logout successful'));**

**});**

**export const refreshAccessToken = asyncHandler(async (req, res) => {**

**const incomingRefreshToken = req.cookies.refreshToken || req.body.refreshToken;**

**if (!incomingRefreshToken) {**

**throw new ApiError(401, "Unauthorized request");**

**}**

**try {**

**const decodedToken = jwt.verify(incomingRefreshToken, process.env.REFRESH\_TOKEN\_SECRET);**

**const user = await User.findById(decodedToken?.\_id);**

**if (!user) {**

**throw new ApiError(401, "Invalid refresh token");**

**}**

**if (incomingRefreshToken !== user.refreshToken) {**

**throw new ApiError(401, "Refresh token is expired or invalid");**

**}**

**const accessToken = user.generateAccessToken();**

**const newRefreshToken = user.generateRefreshToken();**

**user.refreshToken = newRefreshToken;**

**await user.save({ validateBeforeSave: false });**

**const options = {**

**httpOnly: true,**

**secure: true, // Set to true if using HTTPS**

**sameSite: 'Lax',**

**};**

**res**

**.status(200)**

**.cookie('accessToken', accessToken, options)**

**.cookie('refreshToken', newRefreshToken, options)**

**.json(new ApiResponse(200, { accessToken, refreshToken: newRefreshToken }, "Access token refreshed successfully"));**

**} catch (error) {**

**throw new ApiError(401, error?.message || "Invalid refresh token");**

**}**

**});**

**const changeCurrentPassword = asyncHandler(async(req, res) => {**

**const {oldPassword, newPassword} = req.body**

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

**const isPasswordCorrect = await user.isPasswordCorrect(oldPassword)**

**if (!isPasswordCorrect) {**

**throw new ApiError(400, "Invalid old password")**

**}**

**user.password = newPassword**

**await user.save({validateBeforeSave: false})**

**return res**

**.status(200)**

**.json(new ApiResponse(200, {}, "Password changed successfully"))**

**})**

**export const redirectToDashboard = asyncHandler(async (req, res) => {**

**const { role } = req.user;**

**if (role === 'customer') {**

**res.status(200).json(new ApiResponse(200, {}, 'Redirect to customer dashboard'));**

**} else if (role === 'admin') {**

**res.status(200).json(new ApiResponse(200, {}, 'Redirect to admin dashboard'));**

**} else {**

**throw new ApiError(403, 'Access denied');**

**}**

**});**

**Admin Controllers**

**import { User } from '../models/user.model.js';**

**import { asyncHandler } from '../utils/asyncHandler.js';**

**import { ApiResponse } from '../utils/ApiResponse.js';**

**import { LoanApplication } from '../models/loanapplication.model.js';**

**const getAllUsers = asyncHandler(async (req, res) => {**

**try {**

**console.log('Fetching all users...'); // Debug log**

**const users = await User.find().select('-password -refreshToken');**

**console.log('Users fetched:', users); // Debug log**

**res.status(200).json(new ApiResponse(200, users, 'All users fetched successfully'));**

**} catch (error) {**

**console.error('Error fetching users:', error.message); // Debug log**

**res.status(500).json(new ApiResponse(500, {}, 'Internal Server Error'));**

**}**

**});**

**const deleteUser = asyncHandler(async (req, res) => {**

**const { userId } = req.params;**

**const user = await User.findByIdAndDelete(userId);**

**if (!user) {**

**res.status(404).json(new ApiResponse(404, {}, 'User not found'));**

**return;**

**}**

**res.status(200).json(new ApiResponse(200, {}, 'User deleted successfully'));**

**});**

**const getPendingLoans = asyncHandler(async (req, res) => {**

**const pendingLoans = await LoanApplication.find({ loanStatus: 'pending' });**

**res.status(200).json(new ApiResponse(200, pendingLoans, 'Pending loans fetched successfully'));**

**});**

**const getDashboardData = asyncHandler(async (req, res) => {**

**const userCount = await User.countDocuments();**

**const totalApplications = await LoanApplication.countDocuments();**

**const pendingApplications = await LoanApplication.countDocuments({ loanStatus: 'pending' });**

**const approvedApplications = await LoanApplication.countDocuments({ loanStatus: 'approved' });**

**const rejectedApplications = await LoanApplication.countDocuments({ loanStatus: 'rejected' });**

**res.status(200).json(new ApiResponse(200, {**

**userCount,**

**totalApplications,**

**pendingApplications,**

**approvedApplications,**

**rejectedApplications,**

**}, 'Dashboard data fetched successfully'));**

**});**

**const getOverdueLoans = asyncHandler(async (req, res) => {**

**const today = new Date();**

**// Find loans where the EMI due date has passed but the EMI has not been paid**

**const overdueLoans = await LoanApplication.find({**

**loanStatus: 'approved',**

**emiDueDates: { $elemMatch: { $lt: today } }, // Find due dates in the past**

**emiPaidDates: { $not: { $size: 1 } }, // Ensure no EMI has been paid**

**});**

**if (overdueLoans.length === 0) {**

**return res.status(200).json(new ApiResponse(200, [], 'No overdue loans found'));**

**}**

**const overdueUsers = overdueLoans.map((loan) => ({**

**name: loan.userId.fullName,**

**email: loan.userId.email,**

**phone: loan.userId.phone,**

**loanAmount: loan.loanAmount,**

**overdueDate: loan.emiDueDates.find((date) => date < today), // Find the overdue date**

**}));**

**res.status(200).json(new ApiResponse(200, overdueUsers, 'Overdue loans fetched successfully'));**

**});**

**export { getAllUsers, deleteUser, getPendingLoans, getDashboardData, getOverdueLoans };**

**UserRoutes**

**import { Router } from "express";**

**import {**

**loginUser,**

**registerUser,**

**getUserProfile,**

**getAdminProfile,**

**redirectToDashboard,**

**refreshAccessToken**

**} from "../controllers/user.controller.js";**

**import {upload} from "../middlewares/multer.middleware.js"**

**import { verifyJWT } from "../middlewares/auth.middleware.js";**

**const router = Router()**

**// Route for user registration**

**router.post("/register",**

**upload.single("profile"), // Expect the field name to be "profile"**

**registerUser**

**)**

**// Route for user login**

**router.post("/login", loginUser)**

**// Route for fetching user profile**

**router.get("/user/profile", getUserProfile);**

**// Route for fetching admin profile**

**router.get("/admin/profile", getAdminProfile);**

**// Redirect to dashboard**

**router.get('/dashboard', verifyJWT, redirectToDashboard);**

**// Route to refresh access token**

**router.post('/refresh-token', refreshAccessToken);**

**export default router**

**Admin Routes**

**import express from 'express';**

**import { asyncHandler } from '../utils/asyncHandler.js';**

**import { getAllUsers, getDashboardData, getOverdueLoans } from '../controllers/admin.controller.js';**

**import { getCurrentUser, getAdminProfile } from '../controllers/user.controller.js';**

**import { verifyJWT } from '../middlewares/auth.middleware.js';**

**import { User } from '../models/user.model.js';**

**import { LoanApplication } from '../models/loanapplication.model.js';**

**import { ApiResponse } from '../utils/ApiResponse.js';**

**const router = express.Router();**

**// Route to fetch all users**

**router.get('/users', asyncHandler(getAllUsers));**

**// Route to fetch current user details**

**router.get('/current', verifyJWT, asyncHandler(getCurrentUser));**

**// Route to fetch admin profile**

**router.get('/profile', verifyJWT, asyncHandler(getAdminProfile));**

**// Route to fetch dashboard data**

**router.get('/dashboard', verifyJWT, asyncHandler(getDashboardData));**

**// Route to fetch dashboard statistics**

**router.get('/dashboard-stats', asyncHandler(async (req, res) => {**

**const totalUsers = await User.countDocuments({ role: 'customer' });**

**const pendingLoans = await LoanApplication.countDocuments({ loanStatus: 'pending' });**

**const rejectedLoans = await LoanApplication.countDocuments({ loanStatus: 'rejected' });**

**const approvedLoans = await LoanApplication.countDocuments({ loanStatus: 'approved' });**

**const monthlyApplications = await LoanApplication.aggregate([**

**{**

**$group: {**

**\_id: { $month: '$createdAt' },**

**count: { $sum: 1 },**

**},**

**},**

**{ $sort: { \_id: 1 } },**

**]);**

**const loanTypeDistribution = await LoanApplication.aggregate([**

**{**

**$group: {**

**\_id: '$loanType',**

**count: { $sum: 1 },**

**},**

**},**

**]);**

**res.status(200).json(new ApiResponse(200, {**

**totalUsers,**

**pendingLoans,**

**rejectedLoans,**

**approvedLoans,**

**monthlyApplications,**

**loanTypeDistribution,**

**}, 'Dashboard statistics fetched successfully'));**

**}));**

**// Route to fetch overdue loans**

**router.get('/overdue-loans', verifyJWT, asyncHandler(getOverdueLoans));**

**export default router;**