**ADMIN LOGIN**

import React, { useState } from 'react';

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

import { Eye, EyeOff, Shield, Brain, ArrowLeft } from 'lucide-react';

import { Button } from '@/components/ui/button';

import { Input } from '@/components/ui/input';

import { Label } from '@/components/ui/label';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { ThemeToggle } from '@/components/ThemeToggle';

const AdminLogin = () => {

  const [showPassword, setShowPassword] = useState(false);

  // CHANGED: state now uses username

  const [loginData, setLoginData] = useState({ username: '', password: '' });

  const navigate = useNavigate();

  // CHANGED: now actually submits to FastAPI (real login logic)

  const handleLogin = async (e: React.FormEvent) => {

    e.preventDefault();

    try {

      const res = await fetch("/api/admin/login", {

        method: "POST",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify({

          username: loginData.username,

          password: loginData.password,

        }),

      });

      if (res.ok) {

        navigate('/admin/dashboard');

      } else {

        const err = await res.json();

        alert("Login failed: " + (err.detail || res.statusText));

      }

    } catch (err: any) {

      alert("Error: " + (err.message || err));

    }

  };

  return (

    <div className="min-h-screen bg-gradient-to-br from-red-50 via-orange-50 to-yellow-50 dark:from-gray-900 dark:via-red-900 dark:to-orange-900 flex items-center justify-center p-4">

      <div className="w-full max-w-md">

        {/\* Back to Home and Theme Toggle \*/}

        <div className="flex items-center justify-between mb-6">

          <Link to="/" className="inline-flex items-center text-red-600 dark:text-red-400 hover:text-red-700 dark:hover:text-red-300 transition-colors">

            <ArrowLeft className="w-4 h-4 mr-2" />

            Back to Home

          </Link>

          <ThemeToggle />

        </div>

        {/\* Header \*/}

        <div className="text-center mb-8">

          <div className="flex items-center justify-center space-x-3 mb-4">

            <div className="w-12 h-12 bg-gradient-to-r from-red-400 to-orange-400 rounded-lg flex items-center justify-center">

              <Brain className="w-7 h-7 text-white" />

            </div>

            <h1 className="text-2xl font-bold bg-gradient-to-r from-red-600 to-orange-600 bg-clip-text text-transparent">

              ApexWolf

            </h1>

          </div>

          <h2 className="text-3xl font-bold text-gray-800 dark:text-gray-100 mb-2">Admin Portal</h2>

          <p className="text-gray-600 dark:text-gray-300">Secure administrative access</p>

        </div>

        {/\* Login Card \*/}

        <Card className="bg-white/90 dark:bg-gray-800/90 border-red-200 dark:border-gray-700 backdrop-blur-sm shadow-xl">

          <CardHeader className="text-center pb-4">

            <div className="w-16 h-16 bg-gradient-to-r from-red-500 to-red-600 rounded-full flex items-center justify-center mx-auto mb-4">

              <Shield className="w-8 h-8 text-white" />

            </div>

            <CardTitle className="text-gray-800 dark:text-gray-100 text-xl">Administrator Access</CardTitle>

          </CardHeader>

          <CardContent>

            <form onSubmit={handleLogin} className="space-y-6">

              <div className="space-y-2">

                {/\* CHANGED: value and onChange now use username \*/}

                <Label htmlFor="admin-username" className="text-gray-700 dark:text-gray-300">Admin Username</Label>

                <Input

                  id="admin-username"

                  type="email"

                  placeholder="Enter admin username"

                  value={loginData.username}

                  onChange={(e) => setLoginData({ ...loginData, username: e.target.value })}

                  className="bg-gray-50 dark:bg-gray-700 border-gray-300 dark:border-gray-600 text-gray-800 dark:text-gray-100 placeholder:text-gray-500 dark:placeholder:text-gray-400 focus:border-red-400 dark:focus:border-red-500"

                  required

                />

              </div>

              <div className="space-y-2">

                <Label htmlFor="admin-password" className="text-gray-700 dark:text-gray-300">Admin Password</Label>

                <div className="relative">

                  <Input

                    id="admin-password"

                    type={showPassword ? "text" : "password"}

                    placeholder="Enter admin password"

                    value={loginData.password}

                    onChange={(e) => setLoginData({ ...loginData, password: e.target.value })}

                    className="bg-gray-50 dark:bg-gray-700 border-gray-300 dark:border-gray-600 text-gray-800 dark:text-gray-100 placeholder:text-gray-500 dark:placeholder:text-gray-400 focus:border-red-400 dark:focus:border-red-500 pr-10"

                    required

                  />

                  <button

                    type="button"

                    onClick={() => setShowPassword(!showPassword)}

                    className="absolute right-3 top-1/2 transform -translate-y-1/2 text-gray-500 dark:text-gray-400 hover:text-gray-700 dark:hover:text-gray-300"

                  >

                    {showPassword ? <EyeOff className="w-4 h-4" /> : <Eye className="w-4 h-4" />}

                  </button>

                </div>

              </div>

              <div className="bg-red-100 dark:bg-red-900/50 border border-red-300 dark:border-red-700 rounded-lg p-4">

                <p className="text-red-700 dark:text-red-300 text-sm">

                  <Shield className="w-4 h-4 inline mr-2" />

                  This is a secure admin portal. Unauthorized access is prohibited.

                </p>

              </div>

              <Button

                type="submit"

                className="w-full bg-gradient-to-r from-red-500 to-red-600 hover:from-red-600 hover:to-red-700 text-white font-semibold py-3 transition-all duration-300 transform hover:scale-105"

              >

                Access Admin Dashboard

              </Button>

            </form>

          </CardContent>

        </Card>

      </div>

    </div>

  );

};

export default AdminLogin;

**ELIGIBILITY PREDICTION LOGS**

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

type PredictionLog = {

  id: number;

  user\_type: string;

  input\_data: Record<string, string>;

  prediction\_result?: Record<string, any>;

  created\_at: string;

};

const LoanEligibilityAdmin: React.FC = () => {

  const [logs, setLogs] = useState<PredictionLog[]>([]);

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

  const [error, setError] = useState<string | null>(null);

  useEffect(() => {

    setLoading(true);

    fetch("/api/admin\_logs")

      .then((res) => {

        if (!res.ok) throw new Error("Failed to fetch admin logs");

        return res.json();

      })

      .then((data) => {

        setLogs(data.logs || []);

        setError(null);

      })

      .catch((err) => {

        setError(err.message);

        setLogs([]);

      })

      .finally(() => setLoading(false));

  }, []);

  return (

    <div className="min-h-screen bg-white p-6">

      <div className="max-w-7xl mx-auto">

        {/\* Header Section \*/}

        <div className="mb-8">

          <div className="text-center mb-6">

            <h1 className="text-2xl font-semibold text-gray-800 mb-2">

              Eligibility Prediction Logs

            </h1>

            <div className="w-24 h-1 bg-gradient-to-r from-red-500 to-orange-500 mx-auto rounded-full"></div>

          </div>

          {/\* Stats Cards \*/}

          <div className="grid grid-cols-1 md:grid-cols-3 gap-6 mb-6">

            <div className="bg-gradient-to-br from-red-50 to-orange-50 rounded-xl p-6 border border-orange-100">

              <div className="flex items-center justify-between">

                <div>

                  <p className="text-sm text-gray-600 mb-1">Total Logs</p>

                  <p className="text-2xl font-semibold text-gray-800">{logs.length}</p>

                </div>

                <div className="w-12 h-12 bg-gradient-to-br from-red-500 to-orange-500 rounded-lg flex items-center justify-center">

                  <span className="text-white font-bold text-lg">📊</span>

                </div>

              </div>

            </div>

            <div className="bg-gradient-to-br from-orange-50 to-red-50 rounded-xl p-6 border border-red-100">

              <div className="flex items-center justify-between">

                <div>

                  <p className="text-sm text-gray-600 mb-1">With Results</p>

                  <p className="text-2xl font-semibold text-gray-800">

                    {logs.filter(log => log.prediction\_result).length}

                  </p>

                </div>

                <div className="w-12 h-12 bg-gradient-to-br from-orange-500 to-red-500 rounded-lg flex items-center justify-center">

                  <span className="text-white font-bold text-lg">✅</span>

                </div>

              </div>

            </div>

            <div className="bg-gradient-to-br from-red-50 to-orange-50 rounded-xl p-6 border border-orange-100">

              <div className="flex items-center justify-between">

                <div>

                  <p className="text-sm text-gray-600 mb-1">Status</p>

                  <p className="text-sm font-medium text-green-600">

                    {loading ? "Loading..." : "Active"}

                  </p>

                </div>

                <div className="w-12 h-12 bg-gradient-to-br from-red-500 to-orange-500 rounded-lg flex items-center justify-center">

                  <span className="text-white font-bold text-lg">🔄</span>

                </div>

              </div>

            </div>

          </div>

        </div>

        {/\* Loading State \*/}

        {loading && (

          <div className="flex justify-center items-center py-12">

            <div className="flex items-center space-x-3">

              <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-gradient-to-r from-red-500 to-orange-500"></div>

              <span className="text-sm text-gray-600">Loading user logs...</span>

            </div>

          </div>

        )}

        {/\* Error State \*/}

        {error && (

          <div className="bg-red-50 border border-red-200 rounded-xl p-6 mb-6">

            <div className="flex items-center">

              <div className="w-10 h-10 bg-red-100 rounded-lg flex items-center justify-center mr-4">

                <span className="text-red-600 font-bold">⚠️</span>

              </div>

              <div>

                <h3 className="text-sm font-medium text-red-800">Error Loading Data</h3>

                <p className="text-sm text-red-600 mt-1">{error}</p>

              </div>

            </div>

          </div>

        )}

        {/\* Empty State \*/}

        {!loading && logs.length === 0 && !error && (

          <div className="text-center py-16">

            <div className="w-24 h-24 bg-gradient-to-br from-red-100 to-orange-100 rounded-full mx-auto mb-6 flex items-center justify-center">

              <span className="text-4xl">📝</span>

            </div>

            <h3 className="text-lg font-medium text-gray-800 mb-2">No Logs Found</h3>

            <p className="text-sm text-gray-600">User prediction logs will appear here once available.</p>

          </div>

        )}

        {/\* Data Table \*/}

        {!loading && logs.length > 0 && (

          <div className="bg-white rounded-xl shadow-sm border border-gray-200">

            <div className="px-6 py-4 border-b border-gray-200">

              <h2 className="text-lg font-medium text-gray-800">User Prediction Logs</h2>

              <p className="text-sm text-gray-600 mt-1">View and manage all user prediction requests</p>

            </div>

            <div className="overflow-x-auto">

              <table className="w-full">

                <thead>

                  <tr className="bg-gradient-to-r from-red-50 to-orange-50 border-b border-orange-100">

                    <th className="px-6 py-4 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">

                      ID

                    </th>

                    <th className="px-6 py-4 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">

                      User Type

                    </th>

                    <th className="px-6 py-4 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">

                      Input Data

                    </th>

                    <th className="px-6 py-4 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">

                      Prediction Result

                    </th>

                    <th className="px-6 py-4 text-left text-xs font-medium text-gray-700 uppercase tracking-wider">

                      Created At

                    </th>

                  </tr>

                </thead>

                <tbody className="bg-white divide-y divide-gray-100">

                  {logs.map((log, idx) => (

                    <tr

                      key={log.id}

                      className={`hover:bg-gradient-to-r hover:from-red-25 hover:to-orange-25 transition-all duration-200 ${

                        idx % 2 === 0 ? "bg-white" : "bg-gray-25"

                      }`}

                    >

                      <td className="px-6 py-6 whitespace-nowrap">

                        <div className="flex items-center">

                          <div className="w-8 h-8 bg-gradient-to-br from-red-500 to-orange-500 rounded-lg flex items-center justify-center text-white text-xs font-bold mr-3">

                            {log.id}

                          </div>

                        </div>

                      </td>

                      <td className="px-6 py-6 whitespace-nowrap">

                        <div className="flex items-center">

                          <span className="inline-flex items-center px-3 py-1 rounded-full text-xs font-medium bg-gradient-to-r from-red-100 to-orange-100 text-gray-800 border border-orange-200">

                            {log.user\_type}

                          </span>

                        </div>

                      </td>

                      <td className="px-6 py-6">

                        <div className="max-w-xs">

                          <FieldDisplay

                            data={log.input\_data}

                            labelColor="#dc2626"

                            valueColor="#374151"

                          />

                        </div>

                      </td>

                      <td className="px-6 py-6">

                        <div className="max-w-xs">

                          {log.prediction\_result ? (

                            <FieldDisplay

                              data={log.prediction\_result}

                              labelColor="#ea580c"

                              valueColor="#374151"

                            />

                          ) : (

                            <div className="flex items-center">

                              <span className="inline-flex items-center px-3 py-1 rounded-full text-xs font-medium bg-gray-100 text-gray-500 border">

                                No Result

                              </span>

                            </div>

                          )}

                        </div>

                      </td>

                      <td className="px-6 py-6 whitespace-nowrap text-sm text-gray-600">

                        <div>

                          <div className="font-medium">

                            {new Date(log.created\_at).toLocaleDateString("en-IN", {

                              day: "2-digit",

                              month: "short",

                              year: "numeric"

                            })}

                          </div>

                          <div className="text-xs text-gray-500 mt-1">

                            {new Date(log.created\_at).toLocaleTimeString("en-IN", {

                              hour: "2-digit",

                              minute: "2-digit"

                            })}

                          </div>

                        </div>

                      </td>

                    </tr>

                  ))}

                </tbody>

              </table>

            </div>

            <div className="px-6 py-4 bg-gray-50 border-t border-gray-200 rounded-b-xl">

              <div className="flex items-center justify-between">

                <div className="text-sm text-gray-600">

                  Showing <span className="font-medium">{logs.length}</span> results

                </div>

                <button className="bg-gradient-to-r from-red-500 to-orange-500 hover:from-red-600 hover:to-orange-600 text-white px-4 py-2 rounded-lg text-xs font-medium transition-all duration-200 shadow-sm hover:shadow-md">

                  Refresh Data

                </button>

              </div>

            </div>

          </div>

        )}

      </div>

    </div>

  );

};

// Enhanced Helper component for displaying fields

const FieldDisplay: React.FC<{

  data: Record<string, any>;

  labelColor?: string;

  valueColor?: string;

}> = ({ data, labelColor = "#dc2626", valueColor = "#374151" }) => (

  <div className="space-y-2">

    {Object.entries(data).map(([k, v]) => (

      <div key={k} className="flex flex-col space-y-1">

        <span

          className="text-xs font-semibold uppercase tracking-wide"

          style={{ color: labelColor }}

        >

          {k.replace(/\_/g, ' ')}

        </span>

        <span

          className="text-sm break-words bg-gray-50 px-2 py-1 rounded border"

          style={{ color: valueColor }}

        >

          {String(v)}

        </span>

      </div>

    ))}

  </div>

);

export default LoanEligibilityAdmin;

**LOAN SIMULATOR ADMIN**

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

export default function LoanSimulatorAdmin() {

  // State for interest rate field and result message

  const [rate, setRate] = useState("");

  const [message, setMessage] = useState("");

  // Fetch the current interest rate from backend on mount

  const fetchCurrentRate = async () => {

    const res = await fetch("http://localhost:5000/api/loan/interest");

    const json = await res.json();

    setRate(json.interest\_rate.toString());

  };

  // Validate the rate before submitting

  const validateRate = () => {

    // Interest rate must be a number and positive

    if (rate.trim() === "") {

      setMessage("Interest rate is required");

      return false;

    }

    const num = parseFloat(rate);

    if (isNaN(num) || num <= 0) {

      setMessage("Interest rate must be a positive number");

      return false;

    }

    return true;

  };

  // Handle update button click

  const updateRate = async () => {

    // Run validation before request

    if (!validateRate()) {

      return;

    }

    const res = await fetch("http://localhost:5000/api/loan/interest", {

      method: "POST",

      headers: { "Content-Type": "application/json" },

      body: JSON.stringify({ interest\_rate: parseFloat(rate) })

    });

    if (res.ok) {

      setMessage("Interest rate updated successfully");

    } else {

      setMessage("Failed to update");

    }

  };

  // Load the current rate when component mounts

  useEffect(() => {

    fetchCurrentRate();

  }, []);

  return (

    <div className="min-h-screen bg-white flex items-center justify-center p-6">

      <div className="w-full max-w-2xl bg-white">

        {/\* Header Section \*/}

        <div className="text-center mb-12">

          <h1 className="text-3xl font-medium text-gray-800 mb-3">

            Simulator Interest Rate

          </h1>

          <p className="text-sm text-gray-500">

            Manage loan interest rates for the system

          </p>

          <div className="w-24 h-1 bg-gradient-to-r from-red-500 to-orange-400 mx-auto mt-4 rounded-full"></div>

        </div>

        {/\* Main Form Card \*/}

        <div className="bg-white border border-gray-100 rounded-2xl shadow-lg p-8 mb-6">

          <div className="mb-8">

            <h2 className="text-lg font-medium text-gray-700 mb-2">

              Interest Rate Configuration

            </h2>

            <p className="text-xs text-gray-500">

              Set the current interest rate percentage for loan calculations

            </p>

          </div>

          {/\* Input Field \*/}

          <div className="mb-10">

            <label className="block text-sm font-medium text-gray-700 mb-3">

              Interest Rate (%)

            </label>

            <div className="relative">

              <input

                type="number"

                step="0.01"

                value={rate}

                onChange={(e) => setRate(e.target.value)}

                className="w-full px-6 py-4 text-gray-800 text-base border-2 border-gray-200 rounded-xl focus:outline-none focus:border-orange-300 focus:ring-2 focus:ring-orange-100 transition-all duration-200"

                placeholder="Enter interest rate..."

              />

              <div className="absolute right-4 top-1/2 transform -translate-y-1/2 text-gray-400 text-sm">

                %

              </div>

            </div>

            <p className="text-xs text-gray-500 mt-2">

              Enter a positive number (e.g., 5.25 for 5.25%)

            </p>

          </div>

          {/\* Update Button \*/}

          <div className="mb-6">

            <button

              onClick={updateRate}

              className="w-full bg-gradient-to-r from-red-500 to-orange-400 hover:from-red-600 hover:to-orange-500 text-white px-6 py-4 rounded-xl font-medium text-sm transition-all duration-200 transform hover:scale-[1.02] hover:shadow-lg active:scale-[0.98]"

            >

              Update Interest Rate

            </button>

          </div>

          {/\* Message Display \*/}

          {message && (

            <div className={`p-4 rounded-xl border-l-4 ${

              message.includes("successfully")

                ? "bg-green-50 border-green-400 text-green-700"

                : "bg-red-50 border-red-400 text-red-700"

            }`}>

              <p className="text-sm font-medium">{message}</p>

            </div>

          )}

        </div>

        {/\* Info Cards \*/}

        <div className="grid grid-cols-1 md:grid-cols-2 gap-6">

          <div className="bg-gradient-to-br from-red-50 to-orange-50 border border-red-100 rounded-xl p-6">

            <div className="flex items-center mb-3">

              <div className="w-8 h-8 bg-gradient-to-r from-red-500 to-orange-400 rounded-lg flex items-center justify-center mr-3">

                <svg className="w-4 h-4 text-white" fill="currentColor" viewBox="0 0 20 20">

                  <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7-4a1 1 0 11-2 0 1 1 0 012 0zM9 9a1 1 0 000 2v3a1 1 0 001 1h1a1 1 0 100-2v-3a1 1 0 00-1-1H9z" clipRule="evenodd" />

                </svg>

              </div>

              <h3 className="text-sm font-medium text-gray-700">Current Status</h3>

            </div>

            <p className="text-xs text-gray-600">

              The interest rate affects all new loan calculations in the system

            </p>

          </div>

          <div className="bg-gradient-to-br from-orange-50 to-red-50 border border-orange-100 rounded-xl p-6">

            <div className="flex items-center mb-3">

              <div className="w-8 h-8 bg-gradient-to-r from-orange-500 to-red-400 rounded-lg flex items-center justify-center mr-3">

                <svg className="w-4 h-4 text-white" fill="currentColor" viewBox="0 0 20 20">

                  <path fillRule="evenodd" d="M6.267 3.455a3.066 3.066 0 001.745-.723 3.066 3.066 0 013.976 0 3.066 3.066 0 001.745.723 3.066 3.066 0 012.812 2.812c.051.643.304 1.254.723 1.745a3.066 3.066 0 010 3.976 3.066 3.066 0 00-.723 1.745 3.066 3.066 0 01-2.812 2.812 3.066 3.066 0 00-1.745.723 3.066 3.066 0 01-3.976 0 3.066 3.066 0 00-1.745-.723 3.066 3.066 0 01-2.812-2.812 3.066 3.066 0 00-.723-1.745 3.066 3.066 0 010-3.976 3.066 3.066 0 00.723-1.745 3.066 3.066 0 012.812-2.812zm7.44 5.252a1 1 0 00-1.414-1.414L9 10.586 7.707 9.293a1 1 0 00-1.414 1.414l2 2a1 1 0 001.414 0l4-4z" clipRule="evenodd" />

                </svg>

              </div>

              <h3 className="text-sm font-medium text-gray-700">Validation</h3>

            </div>

            <p className="text-xs text-gray-600">

              Rate must be a positive number greater than zero

            </p>

          </div>

        </div>

        {/\* Footer \*/}

        <div className="text-center mt-12">

          <p className="text-xs text-gray-400">

            Changes take effect immediately for new loan calculations

          </p>

        </div>

      </div>

    </div>

  );

}

**RATE OPTIMIZATION REQUESTS**

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

import { Button } from "@/components/ui/button";

import { Input } from "@/components/ui/input";

import { Label } from "@/components/ui/label";

const NegotiationAdmin = () => {

  const [requests, setRequests] = useState([]);

  const [adminReplies, setAdminReplies] = useState({});

  const [finalRates, setFinalRates] = useState({});

  const [respondedRequests, setRespondedRequests] = useState([]);

  const [activeSuggestionBox, setActiveSuggestionBox] = useState(null);

  const suggestions = {

    Genuine:

    [

      "Thank you sincerely for sharing your situation in detail.",

      "We truly appreciate your openness and will give your request our full attention.",

      "Your explanation is valued and helps us understand your circumstances better.",

      "Your honesty is respected, and we will be as fair as possible in our review.",

      "We acknowledge your efforts to communicate clearly and supportively.",

      "Please know that your request is being considered with great care.",

      "We understand these matters are important, and we are here to support you.",

      "Thank you for trusting us with your story; we are taking it seriously.",

      "Your thoughtful explanation is received with gratitude.",

      "Rest assured, we will handle your case with empathy and understanding."

    ],

    Moderate:

    [

      "Thank you for taking the time to share your circumstances.",

      "Your request is important to us and will be reviewed thoughtfully.",

      "We appreciate your explanation and will look into your case promptly.",

      "Your effort in reaching out is acknowledged and respected.",

      "Thank you for clarifying your needs; we will consider all aspects.",

      "Your communication is appreciated, and we are reviewing your request.",

      "We understand these matters can be challenging and will approach with care.",

      "Thank you for your patience as we carefully review your information.",

      "Your request is on our radar, and we are working to provide a fair outcome.",

      "We will do our best to consider your situation and respond soon."

     ],

    "Likely Invalid":

     [

      "Thank you for reaching out. We encourage providing a bit more detail for clarity.",

      "We appreciate your message and are here to help however we can.",

      "Please consider sharing more information to help us understand better.",

      "Your request is important, and we welcome any further details you can provide.",

      "We are here to listen and would be glad to review more supporting information.",

      "Thank you for contacting us; every situation is unique, and we are open to hearing more.",

      "If possible, please provide any additional context that may assist us in our review.",

      "We want to support you; feel free to explain your circumstances further.",

      "We are grateful for your communication and are available for any follow-up.",

      "Your inquiry is valued, and we remain here to assist you with kindness."

     ],

    "Unclear or Not Valid":

     [

      "Thank you for reaching out. If you wish, you may share more about your situation.",

      "We're here to support you—please let us know if you'd like to provide further details.",

      "Your message is important to us. When you're ready, feel free to elaborate.",

      "If there's anything else you'd like us to know, we're listening.",

      "We appreciate your communication and welcome any additional information.",

      "Whenever you're comfortable, you can share more so we can understand better.",

      "We understand reaching out can be difficult; we're here for you.",

      "Please take your time. If you wish to explain more, we are here to help.",

      "Your well-being is important to us, and we encourage you to reach out any time.",

      "We are always available to listen and assist whenever you need."

     ]

  };

  const fetchAll = async () => {

    const pending = await fetch("http://localhost:5000/api/negotiation/admin/pending").then(res => res.json());

    const responded = await fetch("http://localhost:5000/api/negotiation/admin/responded").then(res => res.json());

    setRequests(pending.requests);

    setRespondedRequests(responded.requests);

  };

  useEffect(() => {

    fetchAll();

  }, []);

  const handleAction = async (id, action) => {

    const note = adminReplies[id];

    const rate = finalRates[id];

    if (!note || !rate) {

      alert("Please provide both admin reply and final rate.");

      return;

    }

    await fetch("http://localhost:5000/api/negotiation/admin/respond", {

      method: "POST",

      headers: { "Content-Type": "application/json" },

      body: JSON.stringify({ id, status: action, admin\_note: note, final\_rate: rate })

    });

    fetchAll();

  };

  const getSentimentBadgeColor = (sentiment) => {

    switch (sentiment) {

      case "Genuine":

        return "bg-emerald-50 text-emerald-700 border border-emerald-200";

      case "Moderate":

        return "bg-amber-50 text-amber-700 border border-amber-200";

      case "Likely Invalid":

        return "bg-orange-50 text-orange-700 border border-orange-200";

      default:

        return "bg-red-50 text-red-700 border border-red-200";

    }

  };

  return (

    <div className="min-h-screen w-full bg-white">

      {/\* Header \*/}

      <div className="bg-white border-b border-gray-100 sticky top-0 z-20">

        <div className="max-w-7xl mx-auto px-8 py-6">

          <div className="flex items-center justify-between">

            <div>

              <h1 className="text-2xl font-semibold text-gray-900">Interest Rate Flexibility Requests</h1>

              <p className="text-sm text-gray-500 mt-1">Manage and review loan negotiation requests</p>

            </div>

            <div className="flex items-center space-x-6 text-sm">

              <div className="flex items-center space-x-2">

                <div className="w-3 h-3 bg-amber-400 rounded-full"></div>

                <span className="text-gray-600">Pending: {requests.length}</span>

              </div>

              <div className="flex items-center space-x-2">

                <div className="w-3 h-3 bg-green-500 rounded-full"></div>

                <span className="text-gray-600">Processed: {respondedRequests.length}</span>

              </div>

            </div>

          </div>

        </div>

      </div>

      <div className="max-w-7xl mx-auto px-8 py-8">

        {/\* Pending Requests \*/}

        <div className="mb-16">

          <div className="flex items-center space-x-3 mb-8">

            <div className="w-1 h-8 bg-gradient-to-b from-red-500 to-orange-500 rounded-full"></div>

            <h2 className="text-xl font-semibold text-gray-900">Pending Requests</h2>

            {requests.length > 0 && (

              <span className="px-3 py-1 bg-amber-100 text-amber-800 rounded-full text-xs font-medium">

                {requests.length} awaiting review

              </span>

            )}

          </div>

          {requests.length === 0 ? (

            <div className="bg-white border border-gray-100 rounded-xl p-12 text-center">

              <div className="w-16 h-16 bg-gray-50 rounded-full flex items-center justify-center mx-auto mb-4">

                <svg className="w-8 h-8 text-gray-400" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                  <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={1.5} d="M9 12h6m-6 4h6m2 5H7a2 2 0 01-2-2V5a2 2 0 012-2h5.586a1 1 0 01.707.293l5.414 5.414a1 1 0 01.293.707V19a2 2 0 01-2 2z" />

                </svg>

              </div>

              <h3 className="text-lg font-medium text-gray-900 mb-2">No pending requests</h3>

              <p className="text-gray-500 text-sm">All negotiation requests have been processed.</p>

            </div>

          ) : (

            <div className="space-y-8">

              {requests.map((req, index) => (

                <div key={req.id} className="bg-white border border-gray-100 rounded-2xl overflow-hidden shadow-sm hover:shadow-md transition-shadow duration-200">

                  {/\* Card Header \*/}

                  <div className="bg-gradient-to-r from-gray-50 to-gray-25 px-8 py-6 border-b border-gray-100">

                    <div className="flex items-center justify-between">

                      <div className="flex items-center space-x-4">

                        <div className="w-10 h-10 bg-gradient-to-br from-red-500 to-orange-500 rounded-xl flex items-center justify-center text-white font-semibold text-sm">

                          #{index + 1}

                        </div>

                        <div>

                          <h3 className="font-medium text-gray-900">{req.loan\_type} Loan</h3>

                          <p className="text-sm text-gray-500">Request ID: {req.id}</p>

                        </div>

                      </div>

                      <span className={`px-3 py-1 rounded-full text-xs font-medium ${getSentimentBadgeColor(req.sentiment)}`}>

                        {req.sentiment}

                      </span>

                    </div>

                  </div>

                  {/\* Card Body \*/}

                  <div className="px-8 py-8 space-y-8">

                    {/\* Loan Details Grid \*/}

                    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">

                      <div className="bg-gray-50 rounded-xl p-5">

                        <label className="block text-xs font-medium text-gray-500 uppercase tracking-wider mb-2">Loan Amount</label>

                        <p className="text-lg font-semibold text-gray-900">₹{req.loan\_amount?.toLocaleString() || req.loan\_amount}</p>

                      </div>

                      <div className="bg-gray-50 rounded-xl p-5">

                        <label className="block text-xs font-medium text-gray-500 uppercase tracking-wider mb-2">Current Rate</label>

                        <p className="text-lg font-semibold text-red-600">{req.current\_rate}%</p>

                      </div>

                      <div className="bg-gray-50 rounded-xl p-5">

                        <label className="block text-xs font-medium text-gray-500 uppercase tracking-wider mb-2">Expected Rate</label>

                        <p className="text-lg font-semibold text-green-600">{req.expected\_rate}%</p>

                      </div>

                      <div className="bg-gray-50 rounded-xl p-5">

                        <label className="block text-xs font-medium text-gray-500 uppercase tracking-wider mb-2">Savings</label>

                        <p className="text-lg font-semibold text-blue-600">{(req.current\_rate - req.expected\_rate).toFixed(1)}%</p>

                      </div>

                    </div>

                    {/\* Comment Section \*/}

                    <div className="bg-gray-50 rounded-xl p-6">

                      <label className="block text-xs font-medium text-gray-500 uppercase tracking-wider mb-3">Customer Comment</label>

                      <p className="text-sm text-gray-700 leading-relaxed">{req.comment}</p>

                    </div>

                    {/\* File Upload \*/}

                    {req.file\_path && (

                      <div className="bg-blue-50 rounded-xl p-6">

                        <label className="block text-xs font-medium text-gray-500 uppercase tracking-wider mb-3">Supporting Documents</label>

                        <a

                          href={`http://localhost:5000/uploads/${req.file\_path}`}

                          className="inline-flex items-center space-x-2 text-sm font-medium text-blue-600 hover:text-blue-700 transition-colors"

                          target="\_blank"

                          rel="noopener noreferrer"

                        >

                          <svg className="w-4 h-4" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                            <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M12 10v6m0 0l-3-3m3 3l3-3m2 8H7a2 2 0 01-2-2V5a2 2 0 012-2h5.586a1 1 0 01.707.293l5.414 5.414a1 1 0 01.293.707V19a2 2 0 01-2 2z" />

                          </svg>

                          <span>View Uploaded Document</span>

                        </a>

                      </div>

                    )}

                    <div className="grid grid-cols-1 lg:grid-cols-2 gap-8">

                      {/\* Admin Reply \*/}

                      <div className="space-y-4">

                        <Label className="text-sm font-medium text-gray-700">Admin Response</Label>

                        <div className="relative">

                          <textarea

                            rows={6}

                            className="w-full border border-gray-200 rounded-xl p-4 text-sm resize-none focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 bg-white"

                            placeholder="Type your response or click a suggestion below..."

                            value={adminReplies[req.id] || ""}

                            onFocus={() => setActiveSuggestionBox(req.id)}

                            onChange={(e) =>

                              setAdminReplies({ ...adminReplies, [req.id]: e.target.value })

                            }

                          />

                          {activeSuggestionBox === req.id && suggestions[req.sentiment] && (

                            <div

                              className="absolute z-30 w-full bg-white border border-gray-200 rounded-xl shadow-lg mt-2 max-h-48 overflow-auto"

                              onMouseLeave={() => setActiveSuggestionBox(null)}

                            >

                              <div className="p-3 border-b border-gray-100 bg-gray-50">

                                <h4 className="text-xs font-medium text-gray-600 uppercase tracking-wider">Suggested Responses</h4>

                              </div>

                              {suggestions[req.sentiment].map((sug, i) => (

                                <div

                                  key={i}

                                  className="p-4 hover:bg-gray-50 cursor-pointer text-sm border-b border-gray-50 last:border-b-0 transition-colors"

                                  onClick={() => {

                                    setAdminReplies((prev) => ({ ...prev, [req.id]: sug }));

                                    setActiveSuggestionBox(null);

                                  }}

                                >

                                  {sug}

                                </div>

                              ))}

                            </div>

                          )}

                        </div>

                      </div>

                      {/\* Final Interest Rate \*/}

                      <div className="space-y-4">

                        <Label className="text-sm font-medium text-gray-700">Final Approved Interest Rate (%)</Label>

                        <div className="space-y-6">

                          <Input

                            className="p-4 text-sm border-gray-200 rounded-xl focus:ring-2 focus:ring-red-500 focus:border-transparent bg-white"

                            placeholder="e.g. 8.9"

                            type="number"

                            step="0.1"

                            value={finalRates[req.id] || ""}

                            onChange={(e) =>

                              setFinalRates({ ...finalRates, [req.id]: e.target.value })

                            }

                          />

                          {/\* Action Buttons \*/}

                          <div className="flex flex-col sm:flex-row gap-3 pt-2">

                            <Button

                              className="flex-1 bg-gradient-to-r from-green-500 to-emerald-600 hover:from-green-600 hover:to-emerald-700 text-white px-6 py-3 rounded-xl font-medium text-sm transition-all duration-200 shadow-sm hover:shadow-md"

                              onClick={() => handleAction(req.id, "approved")}

                            >

                              <svg className="w-4 h-4 mr-2" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                                <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M5 13l4 4L19 7" />

                              </svg>

                              Approve Request

                            </Button>

                            <Button

                              className="flex-1 bg-gradient-to-r from-red-500 to-orange-500 hover:from-red-600 hover:to-orange-600 text-white px-6 py-3 rounded-xl font-medium text-sm transition-all duration-200 shadow-sm hover:shadow-md"

                              onClick={() => handleAction(req.id, "rejected")}

                            >

                              <svg className="w-4 h-4 mr-2" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                                <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2} d="M6 18L18 6M6 6l12 12" />

                              </svg>

                              Reject Request

                            </Button>

                          </div>

                        </div>

                      </div>

                    </div>

                  </div>

                </div>

              ))}

            </div>

          )}

        </div>

        {/\* Processed Requests \*/}

        {respondedRequests.length > 0 && (

          <div>

            <div className="flex items-center justify-between mb-8">

              <div className="flex items-center space-x-3">

                <div className="w-1 h-8 bg-gradient-to-b from-green-500 to-blue-500 rounded-full"></div>

                <h3 className="text-xl font-semibold text-gray-900">Processed Requests</h3>

                <span className="px-3 py-1 bg-green-100 text-green-800 rounded-full text-xs font-medium">

                  {respondedRequests.length} completed

                </span>

              </div>

              {/\* Summary Stats \*/}

              <div className="flex items-center space-x-6 text-sm">

                <div className="flex items-center space-x-2">

                  <div className="w-3 h-3 bg-green-500 rounded-full"></div>

                  <span className="text-gray-600">

                    Approved: <strong className="text-green-700">{respondedRequests.filter(r => r.status === "approved").length}</strong>

                  </span>

                </div>

                <div className="flex items-center space-x-2">

                  <div className="w-3 h-3 bg-red-500 rounded-full"></div>

                  <span className="text-gray-600">

                    Rejected: <strong className="text-red-700">{respondedRequests.filter(r => r.status === "rejected").length}</strong>

                  </span>

                </div>

              </div>

            </div>

            <div className="bg-white border border-gray-100 rounded-2xl overflow-hidden shadow-sm">

              <div className="overflow-x-auto">

                <table className="w-full text-sm">

                  <thead className="bg-gray-50 border-b border-gray-100">

                    <tr>

                      <th className="text-left px-6 py-4 font-medium text-gray-700">Loan Type</th>

                      <th className="text-left px-6 py-4 font-medium text-gray-700">Amount</th>

                      <th className="text-left px-6 py-4 font-medium text-gray-700">Status</th>

                      <th className="text-left px-6 py-4 font-medium text-gray-700">Admin Response</th>

                      <th className="text-left px-6 py-4 font-medium text-gray-700">Final Rate</th>

                    </tr>

                  </thead>

                  <tbody className="divide-y divide-gray-50">

                    {respondedRequests.map((req, index) => (

                      <tr key={req.id} className={`${index % 2 === 0 ? 'bg-white' : 'bg-gray-25'} hover:bg-gray-50 transition-colors duration-150`}>

                        <td className="px-6 py-4">

                          <div className="font-medium text-gray-900">{req.loan\_type}</div>

                        </td>

                        <td className="px-6 py-4">

                          <div className="font-medium text-gray-900">₹{req.loan\_amount?.toLocaleString() || req.loan\_amount}</div>

                        </td>

                        <td className="px-6 py-4">

                          <span className={`inline-flex items-center px-3 py-1 rounded-full text-xs font-medium ${

                            req.status === "approved"

                              ? "bg-green-100 text-green-800"

                              : "bg-red-100 text-red-800"

                          }`}>

                            {req.status === "approved" && (

                              <svg className="w-3 h-3 mr-1" fill="currentColor" viewBox="0 0 20 20">

                                <path fillRule="evenodd" d="M16.707 5.293a1 1 0 010 1.414l-8 8a1 1 0 01-1.414 0l-4-4a1 1 0 011.414-1.414L8 12.586l7.293-7.293a1 1 0 011.414 0z" clipRule="evenodd" />

                              </svg>

                            )}

                            {req.status === "rejected" && (

                              <svg className="w-3 h-3 mr-1" fill="currentColor" viewBox="0 0 20 20">

                                <path fillRule="evenodd" d="M4.293 4.293a1 1 0 011.414 0L10 8.586l4.293-4.293a1 1 0 111.414 1.414L11.414 10l4.293 4.293a1 1 0 01-1.414 1.414L10 11.414l-4.293 4.293a1 1 0 01-1.414-1.414L8.586 10 4.293 5.707a1 1 0 010-1.414z" clipRule="evenodd" />

                              </svg>

                            )}

                            {req.status.charAt(0).toUpperCase() + req.status.slice(1)}

                          </span>

                        </td>

                        <td className="px-6 py-4 max-w-xs">

                          <div className="text-gray-600 truncate" title={req.admin\_note || "—"}>

                            {req.admin\_note || "—"}

                          </div>

                        </td>

                        <td className="px-6 py-4">

                          <div className="font-medium text-gray-900">

                            {req.final\_rate ? `${req.final\_rate}%` : "—"}

                          </div>

                        </td>

                      </tr>

                    ))}

                  </tbody>

                </table>

              </div>

            </div>

          </div>

        )}

      </div>

    </div>

  );

};

export default NegotiationAdmin;

**NOTIFICATION BROADCAST**

import React, { useState } from "react";

import { Toaster, toast } from "react-hot-toast";

// UUID generator for notifications

const generateUUID = (): string =>

  "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (c) {

    const r = (Math.random() \* 16) | 0,

      v = c === "x" ? r : (r & 0x3) | 0x8;

    return v.toString(16);

  });

const NotificationAdmin: React.FC = () => {

  // State for message, notification type, loading, and validation error

  const [message, setMessage] = useState("");

  const [type, setType] = useState<"urgent" | "info" | "reminder">("info");

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

  const [error, setError] = useState<string>("");

  // Validate input before sending notification

  const validate = () => {

    if (!message.trim()) {

      setError("Message cannot be empty");

      return false;

    }

    if (message.length > 500) {

      setError("Message cannot exceed 500 characters");

      return false;

    }

    setError("");

    return true;

  };

  // Send notification to all users with validation

  const sendNotification = async () => {

    if (!validate()) {

      toast.error(error || "Validation failed");

      return;

    }

    const payload = {

      id: generateUUID(),

      message,

      type,

      timestamp: new Date().toISOString(),

    };

    setLoading(true);

    try {

      const response = await fetch("http://localhost:5000/api/notifications/send", {

        method: "POST",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify(payload),

      });

      if (response.ok) {

        toast.success("Notification broadcasted to all users");

        setMessage("");

      } else {

        toast.error("Failed to broadcast notification");

      }

    } catch (err) {

      toast.error("Network error while sending");

      console.error("Notification error:", err);

    } finally {

      setLoading(false);

    }

  };

  // Render admin notification form

  return (

    <div className="min-h-screen bg-white p-8">

      <Toaster position="top-right" />

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

        {/\* Header Section \*/}

        <div className="text-center mb-12">

          <div className="inline-flex items-center justify-center w-16 h-16 rounded-full bg-gradient-to-r from-red-100 to-orange-100 mb-6">

            <svg className="w-8 h-8 text-red-600" fill="none" stroke="currentColor" viewBox="0 0 24 24">

              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M15 17h5l-5 5v-5zm-7.5-1.5L12 7l4.5 8.5H7.5z"/>

            </svg>

          </div>

          <h1 className="text-3xl font-bold bg-gradient-to-r from-red-600 to-orange-600 bg-clip-text text-transparent mb-3">

            Notification Broadcasting Center

          </h1>

          <p className="text-sm text-gray-600 max-w-2xl mx-auto">

            Send instant notifications to all users across the platform. Choose your message type and broadcast to reach everyone simultaneously.

          </p>

        </div>

        {/\* Main Form Card \*/}

        <div className="bg-white border border-gray-100 rounded-3xl shadow-lg p-10">

          <div className="space-y-8">

            {/\* Message Input Section \*/}

            <div className="space-y-4">

              <div className="flex items-center gap-3 mb-6">

                <div className="w-2 h-8 bg-gradient-to-b from-red-500 to-orange-500 rounded-full"></div>

                <h2 className="text-lg font-semibold text-gray-800">Compose Message</h2>

              </div>

              <div className="relative">

                <textarea

                  value={message}

                  onChange={(e) => setMessage(e.target.value)}

                  rows={6}

                  maxLength={500}

                  className="w-full border-2 border-gray-200 rounded-2xl p-6 text-sm resize-none focus:border-red-300 focus:ring-4 focus:ring-red-100 transition-all duration-200 placeholder-gray-400"

                  placeholder="Type your notification message here..."

                />

                <div className="absolute bottom-4 right-4 text-xs text-gray-400">

                  {message.length}/500

                </div>

              </div>

              {/\* Validation error message \*/}

              {error && (

                <div className="flex items-center gap-2 p-4 bg-red-50 border border-red-200 rounded-xl">

                  <svg className="w-4 h-4 text-red-500" fill="currentColor" viewBox="0 0 20 20">

                    <path fillRule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7 4a1 1 0 11-2 0 1 1 0 012 0zm-1-9a1 1 0 00-1 1v4a1 1 0 102 0V6a1 1 0 00-1-1z" clipRule="evenodd"/>

                  </svg>

                  <span className="text-sm text-red-700">{error}</span>

                </div>

              )}

            </div>

            {/\* Notification Type Section \*/}

            <div className="space-y-4">

              <div className="flex items-center gap-3 mb-6">

                <div className="w-2 h-8 bg-gradient-to-b from-orange-500 to-red-500 rounded-full"></div>

                <h2 className="text-lg font-semibold text-gray-800">Notification Type</h2>

              </div>

              <div className="grid grid-cols-1 md:grid-cols-3 gap-4">

                {[

                  { value: "urgent", label: "Urgent", icon: "⚡", desc: "Critical alerts requiring immediate attention", color: "from-red-500 to-red-600" },

                  { value: "info", label: "Information", icon: "ℹ️", desc: "General updates and announcements", color: "from-orange-500 to-red-500" },

                  { value: "reminder", label: "Reminder", icon: "📅", desc: "Scheduled reminders and follow-ups", color: "from-yellow-500 to-orange-500" }

                ].map((option) => (

                  <label key={option.value} className="cursor-pointer">

                    <input

                      type="radio"

                      name="type"

                      value={option.value}

                      checked={type === option.value}

                      onChange={(e) => setType(e.target.value as any)}

                      className="sr-only"

                    />

                    <div className={`p-6 border-2 rounded-2xl transition-all duration-200 ${

                      type === option.value

                        ? `border-red-300 bg-gradient-to-br ${option.color} text-white shadow-lg transform scale-105`

                        : 'border-gray-200 bg-white hover:border-gray-300 hover:shadow-md'

                    }`}>

                      <div className="text-center">

                        <div className="text-2xl mb-3">{option.icon}</div>

                        <h3 className={`font-semibold text-sm mb-2 ${type === option.value ? 'text-white' : 'text-gray-800'}`}>

                          {option.label}

                        </h3>

                        <p className={`text-xs ${type === option.value ? 'text-white/80' : 'text-gray-600'}`}>

                          {option.desc}

                        </p>

                      </div>

                    </div>

                  </label>

                ))}

              </div>

            </div>

            {/\* Send Button Section \*/}

            <div className="pt-6">

              <button

                onClick={sendNotification}

                disabled={loading}

                className={`w-full py-4 px-8 text-sm font-semibold text-white rounded-2xl shadow-lg transition-all duration-300 transform ${

                  loading

                    ? "bg-gray-400 cursor-not-allowed opacity-75"

                    : "bg-gradient-to-r from-red-500 to-orange-500 hover:from-red-600 hover:to-orange-600 hover:shadow-xl hover:scale-105 active:scale-95"

                }`}

              >

                {loading ? (

                  <div className="flex items-center justify-center gap-3">

                    <svg className="w-4 h-4 animate-spin" fill="none" viewBox="0 0 24 24">

                      <circle className="opacity-25" cx="12" cy="12" r="10" stroke="currentColor" strokeWidth="4"/>

                      <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135 5.824 3 7.938l3-2.647z"/>

                    </svg>

                    <span>Broadcasting Message...</span>

                  </div>

                ) : (

                  <div className="flex items-center justify-center gap-3">

                    <svg className="w-4 h-4" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                      <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M12 19l9 2-9-18-9 18 9-2zm0 0v-8"/>

                    </svg>

                    <span>Send Notification</span>

                  </div>

                )}

              </button>

            </div>

            {/\* Stats Footer \*/}

            <div className="pt-8 border-t border-gray-100">

              <div className="grid grid-cols-3 gap-6 text-center">

                <div>

                  <div className="w-12 h-12 mx-auto mb-3 bg-gradient-to-r from-red-100 to-orange-100 rounded-full flex items-center justify-center">

                    <svg className="w-6 h-6 text-red-600" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                      <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M13 10V3L4 14h7v7l9-11h-7z"/>

                    </svg>

                  </div>

                  <p className="text-xs font-medium text-gray-600">Instant Delivery</p>

                </div>

                <div>

                  <div className="w-12 h-12 mx-auto mb-3 bg-gradient-to-r from-orange-100 to-red-100 rounded-full flex items-center justify-center">

                    <svg className="w-6 h-6 text-orange-600" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                      <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M17 20h5v-2a3 3 0 00-5.356-1.857M17 20H7m10 0v-2c0-.656-.126-1.283-.356-1.857M7 20H2v-2a3 3 0 015.356-1.857M7 20v-2c0-.656.126-1.283.356-1.857m0 0a5.002 5.002 0 019.288 0M15 7a3 3 0 11-6 0 3 3 0 016 0zm6 3a2 2 0 11-4 0 2 2 0 014 0zM7 10a2 2 0 11-4 0 2 2 0 014 0z"/>

                    </svg>

                  </div>

                  <p className="text-xs font-medium text-gray-600">All Users</p>

                </div>

                <div>

                  <div className="w-12 h-12 mx-auto mb-3 bg-gradient-to-r from-red-100 to-orange-100 rounded-full flex items-center justify-center">

                    <svg className="w-6 h-6 text-red-600" fill="none" stroke="currentColor" viewBox="0 0 24 24">

                      <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="2" d="M9 12l2 2 4-4m6 2a9 9 0 11-18 0 9 9 0 0118 0z"/>

                    </svg>

                  </div>

                  <p className="text-xs font-medium text-gray-600">Reliable</p>

                </div>

              </div>

            </div>

          </div>

        </div>

      </div>

    </div>

  );

};

export default NotificationAdmin;

**ELIGIBILITY PREDICTION WEIGHTS MANAGEMENT**

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

import {

  Card,

  CardHeader,

  CardTitle,

  CardContent

} from "@/components/ui/card";

import { Input } from "@/components/ui/input";

import { Button } from "@/components/ui/button";

import { Label } from "@/components/ui/label";

import { ScrollArea } from "@/components/ui/scroll-area";

import { Loader2, CheckCircle, AlertCircle, Settings, Database } from "lucide-react";

type WeightMap = { [key: string]: number };

const BACKEND = "http://localhost:5000"; // your FastAPI server

const WEIGHTS\_PATH = "/api/loan/weights";

function parseFastAPIError(obj: any): string {

  if (!obj) return "Unknown error";

  if (Array.isArray(obj)) {

    return obj

      .map(e => (typeof e === "object" && "msg" in e ? e.msg : JSON.stringify(e)))

      .join(" | ");

  }

  if (typeof obj === "object" && "detail" in obj) {

    if (Array.isArray(obj.detail)) {

      return obj.detail.map((e: any) => e.msg || JSON.stringify(e)).join(" | ");

    }

    return String(obj.detail);

  }

  return typeof obj === "string" ? obj : JSON.stringify(obj);

}

const PredictionAdmin: React.FC = () => {

  const [features, setFeatures] = useState<string[]>([]);

  const [selectedFeature, setSelectedFeature] = useState<string>("");

  const [weights, setWeights] = useState<WeightMap>({});

  const [editing, setEditing] = useState<{ [k: string]: string }>({});

  const [loading, setLoading] = useState<{ [k: string]: boolean }>({});

  const [success, setSuccess] = useState<{ [k: string]: boolean }>({});

  const [error, setError] = useState<{ [k: string]: string }>({});

  const [fetchErr, setFetchErr] = useState<string>("");

  const [allWeights, setAllWeights] = useState<{ [feature: string]: WeightMap }>({});

  // Load feature list and all weights

  useEffect(() => {

  fetch(`${BACKEND}${WEIGHTS\_PATH}`)

    .then(res => {

      if (!res.ok) throw new Error(`HTTP ${res.status}`);

      return res.json();

    })

    .then((features: string[]) => {

      if (Array.isArray(features)) {

        setFeatures(features);

        setFetchErr("");

        // Fetch individual weights for each feature

        Promise.all(

          features.map(feature =>

            fetch(`${BACKEND}${WEIGHTS\_PATH}/${encodeURIComponent(feature)}`)

              .then(res => {

                if (!res.ok) throw new Error(`Error loading weights for ${feature}`);

                return res.json();

              })

              .then(weights => ({ feature, weights }))

          )

        ).then(allWeightData => {

          const combined: { [feature: string]: WeightMap } = {};

          for (const { feature, weights } of allWeightData) {

            combined[feature] = weights;

          }

          setAllWeights(combined);

        });

      } else {

        console.error("Expected array of features, got:", features);

        setFeatures([]);

        setFetchErr("Invalid data format from server");

      }

    })

    .catch(err => {

      console.error(err);

      setFeatures([]);

      setFetchErr("Could not load feature list. Backend unreachable?");

    });

}, []);

  // When feature changes, load its weight-map from cached data

  useEffect(() => {

    if (!selectedFeature) {

      setWeights({});

      setEditing({});

      setSuccess({});

      setError({});

      setLoading({});

      return;

    }

    if (allWeights[selectedFeature]) {

      setWeights(allWeights[selectedFeature]);

    } else {

      setWeights({});

    }

    setEditing({});

    setSuccess({});

    setError({});

    setLoading({});

  }, [selectedFeature, allWeights]);

  const handleEdit = (option: string, value: string) => {

    if (/^-?\d\*\.?\d\*$/.test(value) || value === "") {

      setEditing(prev => ({ ...prev, [option]: value }));

    }

  };

  const handleUpdate = async (option: string) => {

    const val = editing[option];

    if (val === undefined || val === "" || isNaN(Number(val))) {

      setError(prev => ({ ...prev, [option]: "Enter a valid number" }));

      return;

    }

    setLoading(prev => ({ ...prev, [option]: true }));

    setSuccess(prev => ({ ...prev, [option]: false }));

    setError(prev => ({ ...prev, [option]: "" }));

    try {

      const resp = await fetch(`${BACKEND}${WEIGHTS\_PATH}/${encodeURIComponent(selectedFeature)}`, {

        method: "POST",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify({ option, value: Number(val) })

     });

      if (resp.ok) {

        setWeights(prev => ({ ...prev, [option]: Number(val) }));

        setAllWeights(prev => ({

          ...prev,

          [selectedFeature]: { ...prev[selectedFeature], [option]: Number(val) }

        }));

        setSuccess(prev => ({ ...prev, [option]: true }));

        setEditing(prev => ({ ...prev, [option]: "" }));

      } else {

        let msg = "Failed to save";

        try {

          const data = await resp.json();

          msg = parseFastAPIError(data);

        } catch {

          msg = "Server error";

        }

        setError(prev => ({ ...prev, [option]: msg }));

      }

    } catch {

      setError(prev => ({ ...prev, [option]: "Network error" }));

    }

    setLoading(prev => ({ ...prev, [option]: false }));

    setTimeout(() => setSuccess(prev => ({ ...prev, [option]: false })), 1500);

  };

  return (

    <div className="min-h-screen w-full bg-white">

      {/\* Header Section \*/}

      <div className="w-full border-b border-gray-100 bg-white shadow-sm">

        <div className="max-w-7xl mx-auto px-8 py-8">

          <div className="flex items-center justify-center gap-4 mb-6">

            <div className="p-3 bg-gradient-to-r from-red-500 to-orange-500 rounded-xl shadow-lg">

              <Settings className="h-8 w-8 text-white" />

            </div>

            <div>

              <h1 className="text-2xl font-bold text-gray-800">Weights Management</h1>

              <p className="text-sm text-gray-600">Loan Weight Management System</p>

            </div>

          </div>

          {/\* Feature Selection \*/}

          <div className="max-w-md mx-auto">

            <Label htmlFor="feature-select" className="block text-sm font-medium text-gray-700 mb-3">

              Select Feature to Configure

            </Label>

            <div className="relative">

              <Database className="absolute left-3 top-1/2 transform -translate-y-1/2 h-4 w-4 text-gray-400" />

              <select

                id="feature-select"

                className="w-full pl-10 pr-4 py-3 bg-white border border-gray-200 rounded-lg text-sm text-gray-800 focus:ring-2 focus:ring-orange-500 focus:border-transparent shadow-sm"

                value={selectedFeature}

                onChange={e => setSelectedFeature(e.target.value)}

              >

                <option value="">-- Choose a Feature --</option>

                {features.map(feat => (

                  <option key={feat} value={feat}>

                    {feat}

                  </option>

                ))}

              </select>

            </div>

            {!selectedFeature && (

              <p className="text-xs text-gray-500 mt-2 text-center">

                Choose a feature from the dropdown to view and edit its weight options

              </p>

            )}

          </div>

        </div>

      </div>

      {/\* Main Content \*/}

      <div className="max-w-7xl mx-auto px-8 py-8">

        {fetchErr && (

          <div className="max-w-2xl mx-auto mb-8">

            <div className="flex items-center gap-3 p-4 bg-red-50 border border-red-200 rounded-lg">

              <AlertCircle className="h-5 w-5 text-red-600 flex-shrink-0" />

              <div>

                <p className="text-sm font-medium text-red-800">Connection Error</p>

                <p className="text-xs text-red-600 mt-1">{fetchErr}</p>

              </div>

            </div>

          </div>

        )}

        {selectedFeature && (

          <div className="space-y-6">

            {/\* Feature Header \*/}

            <div className="text-center py-6">

              <h2 className="text-xl font-semibold text-gray-800 mb-2">

                Configuring: <span className="text-orange-600">{selectedFeature}</span>

              </h2>

              <p className="text-sm text-gray-600">

                Adjust the weights for different options below

              </p>

            </div>

            {/\* Weights Grid \*/}

            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 gap-6">

              {Object.entries(weights).map(([option, value]) => (

                <Card

                  key={option}

                  className="bg-white border border-gray-100 rounded-xl shadow-sm hover:shadow-md transition-shadow duration-200"

                >

                  <CardContent className="p-6">

                    <div className="space-y-4">

                      {/\* Option Name \*/}

                      <div>

                        <h3 className="text-sm font-semibold text-gray-800 mb-2 leading-tight">

                          {option}

                        </h3>

                        <div className="flex items-center gap-2">

                          <span className="text-xs text-gray-500">Current:</span>

                          <span className="font-mono text-sm text-gray-700 bg-gray-50 px-2 py-1 rounded">

                            {value}

                          </span>

                        </div>

                      </div>

                      {/\* Input Section \*/}

                      <div className="space-y-3">

                        <Input

                          type="number"

                          step="0.01"

                          min="0"

                          value={editing[option] ?? ""}

                          placeholder="Enter new weight"

                          className="w-full text-sm border-gray-200 focus:ring-2 focus:ring-orange-500 focus:border-transparent"

                          onChange={e => handleEdit(option, e.target.value)}

                          onKeyDown={e => {

                            if (

                              e.key === "Enter" &&

                              editing[option] &&

                              !loading[option] &&

                              !isNaN(Number(editing[option]))

                            ) {

                              handleUpdate(option);

                            }

                          }}

                        />

                        <Button

                          className="w-full bg-gradient-to-r from-red-500 to-orange-500 hover:from-red-600 hover:to-orange-600 text-white text-sm font-medium shadow-sm transition-all duration-200"

                          disabled={

                            loading[option] ||

                            !editing[option] ||

                            isNaN(Number(editing[option]))

                          }

                          onClick={() => handleUpdate(option)}

                        >

                          {loading[option] ? (

                            <div className="flex items-center gap-2">

                              <Loader2 className="h-4 w-4 animate-spin" />

                              <span>Updating...</span>

                            </div>

                          ) : (

                            "Update Weight"

                          )}

                        </Button>

                      </div>

                      {/\* Status Messages \*/}

                      <div className="min-h-[20px]">

                        {success[option] && (

                          <div className="flex items-center gap-2 text-green-600">

                            <CheckCircle className="h-4 w-4" />

                            <span className="text-xs font-medium">Successfully updated!</span>

                          </div>

                        )}

                        {error[option] && (

                          <div className="flex items-center gap-2 text-red-600">

                            <AlertCircle className="h-4 w-4 flex-shrink-0" />

                            <span className="text-xs">{error[option]}</span>

                          </div>

                        )}

                      </div>

                    </div>

                  </CardContent>

                </Card>

              ))}

            </div>

            {Object.keys(weights).length === 0 && (

              <div className="text-center py-16">

                <div className="max-w-sm mx-auto">

                  <div className="w-16 h-16 bg-gray-100 rounded-full flex items-center justify-center mx-auto mb-4">

                    <Database className="h-8 w-8 text-gray-400" />

                  </div>

                  <h3 className="text-lg font-medium text-gray-800 mb-2">No Options Available</h3>

                  <p className="text-sm text-gray-600">

                    This feature doesn't have any weight options configured yet.

                  </p>

                </div>

              </div>

            )}

          </div>

        )}

        {!selectedFeature && !fetchErr && (

          <div className="text-center py-20">

            <div className="max-w-md mx-auto">

              <div className="w-20 h-20 bg-gradient-to-br from-red-100 to-orange-100 rounded-full flex items-center justify-center mx-auto mb-6">

                <Settings className="h-10 w-10 text-orange-600" />

              </div>

              <h3 className="text-xl font-medium text-gray-800 mb-3">Ready to Configure</h3>

              <p className="text-sm text-gray-600 leading-relaxed">

                Select a feature from the dropdown above to start managing its weight configurations.

                Each feature has different options with adjustable weights that affect loan calculations.

              </p>

            </div>

          </div>

        )}

      </div>

    </div>

  );

};

export default PredictionAdmin;

**REWARDS HUB TASKS & BADGES**

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

import axios from "axios";

import DatePicker from "react-datepicker";

import "react-datepicker/dist/react-datepicker.css";

interface Task {

  id: string;

  label: string;

  xp: number;

  expire\_date: string;

}

interface Reward {

  id: number;

  name: string;

  cost: number;

  type: string;

  expire\_date: string;

}

const RewardsHubAdmin: React.FC = () => {

  const [tasks, setTasks] = useState<Task[]>([]);

  const [rewards, setRewards] = useState<Reward[]>([]);

  const [taskForm, setTaskForm] = useState({

    id: "",

    label: "",

    xp: 0,

    expire\_date: ""

  });

  const [taskExpireDate, setTaskExpireDate] = useState<Date | null>(null);

  const [rewardForm, setRewardForm] = useState({

    id: 0,

    name: "",

    cost: 0,

    type: "",

    expire\_date: ""

  });

  const [rewardExpireDate, setRewardExpireDate] = useState<Date | null>(null);

  useEffect(() => {

    fetchAll();

  }, []);

  const fetchAll = async () => {

    const [tRes, rRes] = await Promise.all([

      axios.get("/api/rewards/admin/tasks"),

      axios.get("/api/rewards/admin/rewards")

    ]);

    setTasks(tRes.data);

    setRewards(rRes.data);

  };

  const formatDateForBackend = (d: Date) => {

    // produce "YYYY-MM-DDTHH:mm:ss"

    return d.toISOString().split(".")[0];

  }

  const createTask = async (e: React.FormEvent) => {

    e.preventDefault();

    try {

      await axios.post("/api/rewards/admin/tasks", taskForm);

      setTaskForm({ id: "", label: "", xp: 0, expire\_date: "" });

      setTaskExpireDate(null);

      fetchAll();

      alert("Task created");

    } catch (err: any) {

      alert(err.response?.data?.detail || "Error creating task");

    }

  };

  const createReward = async (e: React.FormEvent) => {

    e.preventDefault();

    try {

      await axios.post("/api/rewards/admin/rewards", rewardForm);

      setRewardForm({ id: 0, name: "", cost: 0, type: "", expire\_date: "" });

      setRewardExpireDate(null);

      fetchAll();

      alert("Reward created");

    } catch (err: any) {

      alert(err.response?.data?.detail || "Error creating reward");

    }

  };

  return (

    <div className="min-h-screen bg-white">

      {/\* Header \*/}

      <div className="bg-gradient-to-r from-red-50 to-orange-50 border-b border-red-100">

        <div className="max-w-7xl mx-auto px-6 py-8">

          <div className="flex items-center space-x-3">

            <div className="w-10 h-10 bg-gradient-to-r from-red-500 to-orange-400 rounded-lg flex items-center justify-center text-white text-lg">

              🛠️

            </div>

            <h1 className="text-2xl font-semibold text-red-700">

              Rewards & Tasks Admin Panel

            </h1>

          </div>

        </div>

      </div>

      <div className="max-w-7xl mx-auto px-6 py-8">

        {/\* Creation Forms \*/}

        <div className="grid grid-cols-1 lg:grid-cols-2 gap-12 mb-16">

          {/\* Task Creator \*/}

          <div className="bg-white border border-red-100 rounded-xl shadow-sm">

            <div className="bg-gradient-to-r from-red-500 to-orange-400 px-8 py-6 rounded-t-xl">

              <h2 className="text-lg font-medium text-white flex items-center space-x-2">

                <span>📝</span>

                <span>Create New Task</span>

              </h2>

            </div>

            <form onSubmit={createTask} className="p-8">

              <div className="space-y-8">

                <div>

                  <label htmlFor="task-id" className="block text-sm font-medium text-red-600 mb-3">

                    Task ID

                  </label>

                  <input

                    id="task-id"

                    type="text"

                    value={taskForm.id}

                    onChange={(e) =>

                      setTaskForm({ ...taskForm, id: e.target.value })

                    }

                    required

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    placeholder="Enter unique task identifier"

                  />

                </div>

                <div>

                  <label htmlFor="task-label" className="block text-sm font-medium text-red-600 mb-3">

                    Task Label

                  </label>

                  <input

                    id="task-label"

                    type="text"

                    value={taskForm.label}

                    onChange={(e) =>

                      setTaskForm({ ...taskForm, label: e.target.value })

                    }

                    required

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    placeholder="Describe the task"

                  />

                </div>

                <div>

                  <label htmlFor="task-xp" className="block text-sm font-medium text-red-600 mb-3">

                    XP Reward Amount

                  </label>

                  <input

                    id="task-xp"

                    type="number"

                    value={taskForm.xp}

                    onChange={(e) =>

                      setTaskForm({ ...taskForm, xp: +e.target.value })

                    }

                    required

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    placeholder="Enter XP amount"

                  />

                </div>

                <div>

                  <label htmlFor="task-expire" className="block text-sm font-medium text-red-600 mb-3">

                    Expiration Date & Time

                  </label>

                  <DatePicker

                    id="task-expire"

                    selected={taskExpireDate}

                    onChange={(date: Date) => {

                      setTaskExpireDate(date);

                      setTaskForm(f => ({

                        ...f,

                        expire\_date: formatDateForBackend(date),

                      }));

                    }}

                    showTimeSelect

                    timeFormat="hh:mm aa"

                    timeIntervals={15}

                    dateFormat="yyyy-MM-dd hh:mm aa"

                    placeholderText="Select expiration date & time"

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    required

                  />

                </div>

                <button

                  type="submit"

                  className="w-full py-4 bg-gradient-to-r from-red-500 to-orange-400 text-white text-sm font-medium rounded-lg hover:from-red-600 hover:to-orange-500 transform hover:scale-[1.02] transition-all duration-200 shadow-lg hover:shadow-xl"

                >

                  Create Task

                </button>

              </div>

            </form>

          </div>

          {/\* Reward Creator \*/}

          <div className="bg-white border border-red-100 rounded-xl shadow-sm">

            <div className="bg-gradient-to-r from-red-500 to-orange-400 px-8 py-6 rounded-t-xl">

              <h2 className="text-lg font-medium text-white flex items-center space-x-2">

                <span>🎁</span>

                <span>Create New Reward</span>

              </h2>

            </div>

            <form onSubmit={createReward} className="p-8">

              <div className="space-y-8">

                <div>

                  <label htmlFor="reward-id" className="block text-sm font-medium text-red-600 mb-3">

                    Reward ID

                  </label>

                  <input

                    id="reward-id"

                    type="number"

                    value={rewardForm.id}

                    onChange={(e) =>

                      setRewardForm({ ...rewardForm, id: +e.target.value })

                    }

                    required

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    placeholder="Enter unique reward ID"

                  />

                </div>

                <div>

                  <label htmlFor="reward-name" className="block text-sm font-medium text-red-600 mb-3">

                    Reward Name

                  </label>

                  <input

                    id="reward-name"

                    type="text"

                    value={rewardForm.name}

                    onChange={(e) =>

                      setRewardForm({ ...rewardForm, name: e.target.value })

                    }

                    required

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    placeholder="Enter reward name"

                  />

                </div>

                <div>

                  <label htmlFor="reward-cost" className="block text-sm font-medium text-red-600 mb-3">

                    Cost (XP Required)

                  </label>

                  <input

                    id="reward-cost"

                    type="number"

                    value={rewardForm.cost}

                    onChange={(e) =>

                      setRewardForm({ ...rewardForm, cost: +e.target.value })

                    }

                    required

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    placeholder="Enter XP cost"

                  />

                </div>

                <div>

                  <label htmlFor="reward-type" className="block text-sm font-medium text-red-600 mb-3">

                    Reward Type

                  </label>

                  <input

                    id="reward-type"

                    type="text"

                    value={rewardForm.type}

                    onChange={(e) =>

                      setRewardForm({ ...rewardForm, type: e.target.value })

                    }

                    required

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    placeholder="e.g., badge, voucher, discount"

                  />

                </div>

                <div>

                  <label htmlFor="reward-expire" className="block text-sm font-medium text-red-600 mb-3">

                    Expiration Date & Time

                  </label>

                  <DatePicker

                    id="reward-expire"

                    selected={rewardExpireDate}

                    onChange={(date: Date) => {

                      setRewardExpireDate(date);

                      setRewardForm(f => ({

                        ...f,

                        expire\_date: formatDateForBackend(date),

                      }));

                    }}

                    showTimeSelect

                    timeFormat="hh:mm aa"

                    timeIntervals={15}

                    dateFormat="yyyy-MM-dd hh:mm aa"

                    placeholderText="Select expiration date & time"

                    className="w-full px-4 py-4 border border-red-200 rounded-lg focus:ring-2 focus:ring-red-500 focus:border-transparent transition-all duration-200 text-sm"

                    required

                  />

                </div>

                <button

                  type="submit"

                  className="w-full py-4 bg-gradient-to-r from-red-500 to-orange-400 text-white text-sm font-medium rounded-lg hover:from-red-600 hover:to-orange-500 transform hover:scale-[1.02] transition-all duration-200 shadow-lg hover:shadow-xl"

                >

                  Create Reward

                </button>

              </div>

            </form>

          </div>

        </div>

        {/\* Existing Items Display \*/}

        <div className="grid grid-cols-1 lg:grid-cols-2 gap-12">

          {/\* Current Tasks \*/}

          <div className="bg-white border border-red-100 rounded-xl shadow-sm">

            <div className="bg-gradient-to-r from-red-500 to-orange-400 px-8 py-6 rounded-t-xl">

              <h2 className="text-lg font-medium text-white flex items-center space-x-2">

                <span>📋</span>

                <span>Current Tasks ({tasks.length})</span>

              </h2>

            </div>

            <div className="p-8">

              {tasks.length === 0 ? (

                <div className="text-center py-12">

                  <div className="text-red-300 text-4xl mb-4">📝</div>

                  <p className="text-sm text-red-400">No tasks created yet</p>

                </div>

              ) : (

                <div className="space-y-4 max-h-96 overflow-y-auto">

                  {tasks.map((t) => (

                    <div key={t.id} className="bg-red-50 border border-red-100 rounded-lg p-6">

                      <div className="flex justify-between items-start mb-3">

                        <div className="flex items-center space-x-3">

                          <div className="w-8 h-8 bg-gradient-to-r from-red-500 to-orange-400 rounded-full flex items-center justify-center text-white text-xs font-medium">

                            {t.xp}

                          </div>

                          <div>

                            <h3 className="font-medium text-red-700 text-sm">{t.id}</h3>

                            <p className="text-red-600 text-xs mt-1">{t.label}</p>

                          </div>

                        </div>

                      </div>

                      <div className="flex justify-between items-center text-xs text-red-500">

                        <span>{t.xp} XP Reward</span>

                        <span>Expires: {new Date(t.expire\_date).toLocaleDateString()} at {new Date(t.expire\_date).toLocaleTimeString()}</span>

                      </div>

                    </div>

                  ))}

                </div>

              )}

            </div>

          </div>

          {/\* Current Rewards \*/}

          <div className="bg-white border border-red-100 rounded-xl shadow-sm">

            <div className="bg-gradient-to-r from-red-500 to-orange-400 px-8 py-6 rounded-t-xl">

              <h2 className="text-lg font-medium text-white flex items-center space-x-2">

                <span>🏆</span>

                <span>Current Rewards ({rewards.length})</span>

              </h2>

            </div>

            <div className="p-8">

              {rewards.length === 0 ? (

                <div className="text-center py-12">

                  <div className="text-red-300 text-4xl mb-4">🎁</div>

                  <p className="text-sm text-red-400">No rewards created yet</p>

                </div>

              ) : (

                <div className="space-y-4 max-h-96 overflow-y-auto">

                  {rewards.map((r) => (

                    <div key={r.id} className="bg-red-50 border border-red-100 rounded-lg p-6">

                      <div className="flex justify-between items-start mb-3">

                        <div className="flex items-center space-x-3">

                          <div className="w-8 h-8 bg-gradient-to-r from-red-500 to-orange-400 rounded-full flex items-center justify-center text-white text-xs font-medium">

                            {r.cost}

                          </div>

                          <div>

                            <h3 className="font-medium text-red-700 text-sm">{r.name}</h3>

                            <p className="text-red-600 text-xs mt-1 capitalize">{r.type}</p>

                          </div>

                        </div>

                      </div>

                      <div className="flex justify-between items-center text-xs text-red-500">

                        <span>{r.cost} XP Cost</span>

                        <span>Expires: {new Date(r.expire\_date).toLocaleDateString()} at {new Date(r.expire\_date).toLocaleTimeString()}</span>

                      </div>

                    </div>

                  ))}

                </div>

              )}

            </div>

          </div>

        </div>

      </div>

    </div>

  );

};

export default RewardsHubAdmin;