My-App

Commit and Deploy

CD\Users\tshie\backend\rent-to-own-calculator\my-app>

git add .

git commit -m "Column Headings"

git push origin main

If you'd like, you can:

* Add a route fallback with React Router later
* Create shareable links using state (without changing URLs)
* Switch between multiple client profiles using a dropdown or login

<https://smartrent-auto.vercel.app/>

Calculator.jsx 2025/06/03 12:09 Demo Version

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 1000;

const EXPIRY\_DAYS = 365;

// const [inputErrors, setInputErrors] = useState({});

const Calculator = ({ suburbs }) => {

  const [creditScore, setCreditScore] = useState('');

  const [inputErrors, setInputErrors] = useState({});

  const [isAdmin, setIsAdmin] = useState(false);

      const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    const resetCalculator = () => {

  setInputs({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  setResults({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  setSelectedSuburbInfo(null);

  setFilteredSuburbs([]);

  setQuery('');

};

  useEffect(() => {

const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

  const { name, value } = e.target;

  setInputs((prev) => ({

    ...prev,

    [name]: name === "creditScore"

      ? value === "" ? "" : Number(value)

      : value,

  }));

  if (name === "suburb") setSelectedSuburbInfo(null);

};

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

const errors = {};

if (!inputs.clientName) {

  errors.clientName = 'Client Name is required.';

}

if (isNaN(price) || price <= 0) {

  errors.price = 'Vehicle Price must be a valid number greater than 0.';

}

if (isNaN(mm) || mm <= 0) {

  errors.mm = 'M&M Value must be a valid number greater than 0.';

}

if (!inputs.suburb) {

  errors.suburb = 'Please select a Suburb.';

}

if (!selectedSuburbInfo) {

  errors.selectedSuburbInfo = 'Selected Suburb is invalid or not found in the database.';

}

if (!inputs.riskProfile) {

  errors.riskProfile = 'Please select a Risk Profile.';

}

if (isNaN(terms) || terms <= 0) {

  errors.terms = 'Terms must be a valid number greater than 0.';

}

if (

  inputs.creditScore === "" ||

  isNaN(inputs.creditScore) ||

  inputs.creditScore < 0 ||

  inputs.creditScore > 999

) {

  errors.creditScore = 'Credit Score must be a number between 0 and 999.';

}

if (Object.keys(errors).length > 0) {

  Object.values(errors).forEach(msg => toast.error(msg));

  setInputErrors(errors);  // State to trigger red borders

  return;

}

// If all inputs are valid

setInputErrors({});  // Clear any previous errors

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCost = distance \* 10;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 0;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

  setLoadingPdf(true);

  try {

    const doc = new jsPDF();

    // 🟥 Add watermark using manual rotation

    doc.setTextColor(255, 0, 0);

    doc.setFontSize(80);

    doc.setFont("helvetica", "bold");

    doc.text("DEMO", 35, 150, { angle: 45 });

    // Reset styles

    doc.setFontSize(16);

    doc.setTextColor(0, 0, 0);

    doc.text('SmartRent Auto™ - Calculation Summary (DEMO)', 14, 15);

    const tableBody = [

      ['DEMO Version', 'This is a DEMO version.'],

      ['Client Name', inputs.clientName],

      ['Vehicle Price', `R${inputs.vehiclePrice}`],

      ['Retail Value', `R${inputs.mmValue}`],

      ['DEMO Version', 'This is a DEMO version.'],

      ['Suburb', inputs.suburb],

      ['Town', selectedSuburbInfo?.MP\_NAME || ''],

      ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

      ['DEMO Version', 'This is a DEMO version.'],

      ['Province', selectedSuburbInfo?.Province || ''],

      ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

      ['DEMO Version', 'This is a DEMO version.'],

      ['Risk Profile', inputs.riskProfile],

      ['Loading', `R${results.loading.toLocaleString()}`],

      ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

      ['DEMO Version', 'This is a DEMO version.'],

      ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

      ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

      ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

      ['DEMO Version', 'This is a DEMO version.'],

      ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

      ['License & Registration', `R2500`],

      ['Document Fees', `R1500`],

      ['DEMO Version', 'This is a DEMO version.'],

      ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

      ['Terms (months)', inputs.termsInMonths],

      ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`],

      ['DEMO Version', 'This is a DEMO version.']

    ];

    autoTable(doc, {

      startY: 25,

      head: [['Field', 'Value']],

      body: tableBody,

      didParseCell: function (data) {

        if (data.cell.raw === 'This is a DEMO version.') {

          data.cell.styles.fontStyle = 'bold';

          data.cell.styles.fontSize = 12;

          data.cell.styles.textColor = [255, 0, 0]; // Red text

        }

      }

    });

    doc.save('SmartRentAuto\_Calculation\_DEMO.pdf');

    toast.success('PDF exported with DEMO watermark!');

  } catch (err) {

    toast.error('Failed to export PDF.');

  } finally {

    setLoadingPdf(false);

  }

};

  return (

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

      <Toaster />

      <div className="fixed top-10 left-1/2 transform -translate-x-1/2 opacity-10 text-6xl font-extrabold text-red-600 pointer-events-none z-0">

        DEMO

      </div>

      <div className="bg-yellow-200 border border-yellow-400 text-yellow-900 px-4 py-2 rounded mb-4 text-center font-semibold">

        DEMO VERSION – For demonstration purposes only. Not for production use. Formulas are not set to calculate correctly

      </div>

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo Left" className="h-16" />

        <h1 className="text-2xl font-bold text-center flex-1">SmartRent Auto™ Calculator</h1>

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo Right" className="h-16" />

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Client Name

        </label>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      </div>

         <div className="mb-3">

    <label htmlFor="vehiclePrice" className="block text-sm font-medium text-gray-700 mb-1">

      Vehicle Price

    </label>

    <div className="relative flex items-center">

      <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

      <input

        id="vehiclePrice"

        type="number"

        name="vehiclePrice"

        placeholder="Vehicle Price"

        value={inputs.vehiclePrice}

        onChange={handleChange}

        className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

      />

    </div>

  </div>

        <div className="relative mb-3">

          <label htmlFor="mmValue" className="block text-sm font-medium text-gray-700 mb-1">

            Retail Value

          </label>

            <div className="relative flex items-center">

            <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

          <input

            id="mmValue"

            type="number"

            name="mmValue"

            placeholder="Retail Value"

            value={inputs.mmValue}

            onChange={handleChange}

            className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

         />

        </div>

       </div>

      <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

    <input

      type="number"

      name="manualDeposit"

      placeholder="Enter manual deposit if needed"

      value={inputs.manualDeposit}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Suburb name

        </label>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      </div>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Risk Profile

        </label>

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Credit Score

        </label>

      <input

        type="number"

        name="creditScore"

        className="w-full p-2 border border-gray-300 rounded"

        value={inputs.creditScore}

        onChange={handleChange}

        placeholder="e.g. 650"

        min="0"

        max="999"

      />

      </div>

  <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Terms in months</label>

    <input

      type="number"

      name="termsInMonths"

      placeholder="Enter terms in months"

      value={inputs.termsInMonths}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

  </div>

  <button

    onClick={calculate}

    className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

    disabled={loadingCalc}

  >

    {loadingCalc ? 'Calculating...' : 'Calculate'}

  </button>

  <button

    onClick={resetCalculator}

    className="w-full bg-gray-600 text-white py-2 rounded mb-4 hover:bg-gray-700"

    disabled={loadingCalc}

  >

    Reset

  </button>

  <div className="bg-gray-100 p-4 rounded mb-6">

  <h2 className="font-semibold mb-2">Results</h2>

  <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

  <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

  <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

  <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

  <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

    </div>

  </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/06/03 11:14

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 1000;

const EXPIRY\_DAYS = 365;

// const [inputErrors, setInputErrors] = useState({});

const Calculator = ({ suburbs }) => {

  const [creditScore, setCreditScore] = useState('');

  const [inputErrors, setInputErrors] = useState({});

  const [isAdmin, setIsAdmin] = useState(false);

      const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    const resetCalculator = () => {

  setInputs({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  setResults({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  setSelectedSuburbInfo(null);

  setFilteredSuburbs([]);

  setQuery('');

};

  useEffect(() => {

const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

  const { name, value } = e.target;

  setInputs((prev) => ({

    ...prev,

    [name]: name === "creditScore"

      ? value === "" ? "" : Number(value)

      : value,

  }));

  if (name === "suburb") setSelectedSuburbInfo(null);

};

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

const errors = {};

if (!inputs.clientName) {

  errors.clientName = 'Client Name is required.';

}

if (isNaN(price) || price <= 0) {

  errors.price = 'Vehicle Price must be a valid number greater than 0.';

}

if (isNaN(mm) || mm <= 0) {

  errors.mm = 'M&M Value must be a valid number greater than 0.';

}

if (!inputs.suburb) {

  errors.suburb = 'Please select a Suburb.';

}

if (!selectedSuburbInfo) {

  errors.selectedSuburbInfo = 'Selected Suburb is invalid or not found in the database.';

}

if (!inputs.riskProfile) {

  errors.riskProfile = 'Please select a Risk Profile.';

}

if (isNaN(terms) || terms <= 0) {

  errors.terms = 'Terms must be a valid number greater than 0.';

}

if (

  inputs.creditScore === "" ||

  isNaN(inputs.creditScore) ||

  inputs.creditScore < 0 ||

  inputs.creditScore > 999

) {

  errors.creditScore = 'Credit Score must be a number between 0 and 999.';

}

if (Object.keys(errors).length > 0) {

  Object.values(errors).forEach(msg => toast.error(msg));

  setInputErrors(errors);  // State to trigger red borders

  return;

}

// If all inputs are valid

setInputErrors({});  // Clear any previous errors

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCost = distance \* 10;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 0;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo Left" className="h-16" />

        <h1 className="text-2xl font-bold text-center flex-1">SmartRent Auto™ Calculator</h1>

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo Right" className="h-16" />

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Client Name

        </label>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      </div>

         <div className="mb-3">

    <label htmlFor="vehiclePrice" className="block text-sm font-medium text-gray-700 mb-1">

      Vehicle Price

    </label>

    <div className="relative flex items-center">

      <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

      <input

        id="vehiclePrice"

        type="number"

        name="vehiclePrice"

        placeholder="Vehicle Price"

        value={inputs.vehiclePrice}

        onChange={handleChange}

        className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

      />

    </div>

  </div>

        <div className="relative mb-3">

          <label htmlFor="mmValue" className="block text-sm font-medium text-gray-700 mb-1">

            Retail Value

          </label>

            <div className="relative flex items-center">

            <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

          <input

            id="mmValue"

            type="number"

            name="mmValue"

            placeholder="Retail Value"

            value={inputs.mmValue}

            onChange={handleChange}

            className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

         />

        </div>

       </div>

      <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

    <input

      type="number"

      name="manualDeposit"

      placeholder="Enter manual deposit if needed"

      value={inputs.manualDeposit}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Suburb name

        </label>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      </div>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Risk Profile

        </label>

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Credit Score

        </label>

      <input

        type="number"

        name="creditScore"

        className="w-full p-2 border border-gray-300 rounded"

        value={inputs.creditScore}

        onChange={handleChange}

        placeholder="e.g. 650"

        min="0"

        max="999"

      />

      </div>

  <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Terms in months</label>

    <input

      type="number"

      name="termsInMonths"

      placeholder="Enter terms in months"

      value={inputs.termsInMonths}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

  </div>

  <button

    onClick={calculate}

    className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

    disabled={loadingCalc}

  >

    {loadingCalc ? 'Calculating...' : 'Calculate'}

  </button>

  <button

    onClick={resetCalculator}

    className="w-full bg-gray-600 text-white py-2 rounded mb-4 hover:bg-gray-700"

    disabled={loadingCalc}

  >

    Reset

  </button>

  <div className="bg-gray-100 p-4 rounded mb-6">

  <h2 className="font-semibold mb-2">Results</h2>

  <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

  <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

  <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

  <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

  <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

    </div>

  </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/06/03 11:02 Logo 2 kante

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 1000;

const EXPIRY\_DAYS = 365;

// const [inputErrors, setInputErrors] = useState({});

const Calculator = ({ suburbs }) => {

  const [creditScore, setCreditScore] = useState('');

  const [inputErrors, setInputErrors] = useState({});

  const [isAdmin, setIsAdmin] = useState(false);

      const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    const resetCalculator = () => {

  setInputs({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  setResults({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  setSelectedSuburbInfo(null);

  setFilteredSuburbs([]);

  setQuery('');

};

  useEffect(() => {

const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

  const { name, value } = e.target;

  setInputs((prev) => ({

    ...prev,

    [name]: name === "creditScore"

      ? value === "" ? "" : Number(value)

      : value,

  }));

  if (name === "suburb") setSelectedSuburbInfo(null);

};

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

const errors = {};

if (!inputs.clientName) {

  errors.clientName = 'Client Name is required.';

}

if (isNaN(price) || price <= 0) {

  errors.price = 'Vehicle Price must be a valid number greater than 0.';

}

if (isNaN(mm) || mm <= 0) {

  errors.mm = 'M&M Value must be a valid number greater than 0.';

}

if (!inputs.suburb) {

  errors.suburb = 'Please select a Suburb.';

}

if (!selectedSuburbInfo) {

  errors.selectedSuburbInfo = 'Selected Suburb is invalid or not found in the database.';

}

if (!inputs.riskProfile) {

  errors.riskProfile = 'Please select a Risk Profile.';

}

if (isNaN(terms) || terms <= 0) {

  errors.terms = 'Terms must be a valid number greater than 0.';

}

if (

  inputs.creditScore === "" ||

  isNaN(inputs.creditScore) ||

  inputs.creditScore < 0 ||

  inputs.creditScore > 999

) {

  errors.creditScore = 'Credit Score must be a number between 0 and 999.';

}

if (Object.keys(errors).length > 0) {

  Object.values(errors).forEach(msg => toast.error(msg));

  setInputErrors(errors);  // State to trigger red borders

  return;

}

// If all inputs are valid

setInputErrors({});  // Clear any previous errors

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCost = distance \* 10;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 0;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo Left" className="h-16" />

        <h1 className="text-2xl font-bold text-center flex-1">SmartRent Auto™ Calculator</h1>

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo Right" className="h-16" />

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Client Name

        </label>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      </div>

         <div className="mb-3">

    <label htmlFor="vehiclePrice" className="block text-sm font-medium text-gray-700 mb-1">

      Vehicle Price

    </label>

    <div className="relative flex items-center">

      <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

      <input

        id="vehiclePrice"

        type="number"

        name="vehiclePrice"

        placeholder="Vehicle Price"

        value={inputs.vehiclePrice}

        onChange={handleChange}

        className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

      />

    </div>

  </div>

        <div className="relative mb-3">

          <label htmlFor="mmValue" className="block text-sm font-medium text-gray-700 mb-1">

            Retail Value

          </label>

            <div className="relative flex items-center">

            <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

          <input

            id="mmValue"

            type="number"

            name="mmValue"

            placeholder="Retail Value"

            value={inputs.mmValue}

            onChange={handleChange}

            className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

         />

        </div>

       </div>

      <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

    <input

      type="number"

      name="manualDeposit"

      placeholder="Enter manual deposit if needed"

      value={inputs.manualDeposit}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Suburb name

        </label>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      </div>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Risk Profile

        </label>

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Credit Score

        </label>

      <input

        type="number"

        name="creditScore"

        className="w-full p-2 border border-gray-300 rounded"

        value={inputs.creditScore}

        onChange={handleChange}

        placeholder="e.g. 650"

        min="0"

        max="999"

      />

      </div>

<div className="bg-gray-100 p-4 rounded mb-6">

  <h2 className="font-semibold mb-2">Results</h2>

  <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

  </div>

  <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Terms in months</label>

    <input

      type="number"

      name="termsInMonths"

      placeholder="Enter terms in months"

      value={inputs.termsInMonths}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

  </div>

  <button

    onClick={calculate}

    className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

    disabled={loadingCalc}

  >

    {loadingCalc ? 'Calculating...' : 'Calculate'}

  </button>

  <button

    onClick={resetCalculator}

    className="w-full bg-gray-600 text-white py-2 rounded mb-4 hover:bg-gray-700"

    disabled={loadingCalc}

  >

    Reset

  </button>

  <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

  <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

  <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

  <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

</div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/06/03 10:43

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 1000;

const EXPIRY\_DAYS = 365;

// const [inputErrors, setInputErrors] = useState({});

const Calculator = ({ suburbs }) => {

  const [creditScore, setCreditScore] = useState('');

  const [inputErrors, setInputErrors] = useState({});

  const [isAdmin, setIsAdmin] = useState(false);

      const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    const resetCalculator = () => {

  setInputs({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  setResults({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  setSelectedSuburbInfo(null);

  setFilteredSuburbs([]);

  setQuery('');

};

  useEffect(() => {

const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

  const { name, value } = e.target;

  setInputs((prev) => ({

    ...prev,

    [name]: name === "creditScore"

      ? value === "" ? "" : Number(value)

      : value,

  }));

  if (name === "suburb") setSelectedSuburbInfo(null);

};

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

const errors = {};

if (!inputs.clientName) {

  errors.clientName = 'Client Name is required.';

}

if (isNaN(price) || price <= 0) {

  errors.price = 'Vehicle Price must be a valid number greater than 0.';

}

if (isNaN(mm) || mm <= 0) {

  errors.mm = 'M&M Value must be a valid number greater than 0.';

}

if (!inputs.suburb) {

  errors.suburb = 'Please select a Suburb.';

}

if (!selectedSuburbInfo) {

  errors.selectedSuburbInfo = 'Selected Suburb is invalid or not found in the database.';

}

if (!inputs.riskProfile) {

  errors.riskProfile = 'Please select a Risk Profile.';

}

if (isNaN(terms) || terms <= 0) {

  errors.terms = 'Terms must be a valid number greater than 0.';

}

if (

  inputs.creditScore === "" ||

  isNaN(inputs.creditScore) ||

  inputs.creditScore < 0 ||

  inputs.creditScore > 999

) {

  errors.creditScore = 'Credit Score must be a number between 0 and 999.';

}

if (Object.keys(errors).length > 0) {

  Object.values(errors).forEach(msg => toast.error(msg));

  setInputErrors(errors);  // State to trigger red borders

  return;

}

// If all inputs are valid

setInputErrors({});  // Clear any previous errors

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCost = distance \* 10;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 0;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Client Name

        </label>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      </div>

         <div className="mb-3">

    <label htmlFor="vehiclePrice" className="block text-sm font-medium text-gray-700 mb-1">

      Vehicle Price

    </label>

    <div className="relative flex items-center">

      <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

      <input

        id="vehiclePrice"

        type="number"

        name="vehiclePrice"

        placeholder="Vehicle Price"

        value={inputs.vehiclePrice}

        onChange={handleChange}

        className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

      />

    </div>

  </div>

        <div className="relative mb-3">

          <label htmlFor="mmValue" className="block text-sm font-medium text-gray-700 mb-1">

            Retail Value

          </label>

            <div className="relative flex items-center">

            <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

          <input

            id="mmValue"

            type="number"

            name="mmValue"

            placeholder="Retail Value"

            value={inputs.mmValue}

            onChange={handleChange}

            className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

         />

        </div>

       </div>

      <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

    <input

      type="number"

      name="manualDeposit"

      placeholder="Enter manual deposit if needed"

      value={inputs.manualDeposit}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Suburb name

        </label>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      </div>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Risk Profile

        </label>

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Credit Score

        </label>

      <input

        type="number"

        name="creditScore"

        className="w-full p-2 border border-gray-300 rounded"

        value={inputs.creditScore}

        onChange={handleChange}

        placeholder="e.g. 650"

        min="0"

        max="999"

      />

      </div>

<div className="bg-gray-100 p-4 rounded mb-6">

  <h2 className="font-semibold mb-2">Results</h2>

  <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

  </div>

  <div className="mb-3">

    <label className="block text-sm font-medium text-gray-700">Terms in months</label>

    <input

      type="number"

      name="termsInMonths"

      placeholder="Enter terms in months"

      value={inputs.termsInMonths}

      onChange={handleChange}

      className="w-full p-2 border rounded mt-1"

    />

  </div>

  <button

    onClick={calculate}

    className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

    disabled={loadingCalc}

  >

    {loadingCalc ? 'Calculating...' : 'Calculate'}

  </button>

  <button

    onClick={resetCalculator}

    className="w-full bg-gray-600 text-white py-2 rounded mb-4 hover:bg-gray-700"

    disabled={loadingCalc}

  >

    Reset

  </button>

  <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

  <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

  <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

  <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

</div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/06/02 20:35 Column Headings “R” center

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 1000;

const EXPIRY\_DAYS = 365;

const Calculator = ({ suburbs }) => {

  const [creditScore, setCreditScore] = useState('');

  const [isAdmin, setIsAdmin] = useState(false);

      const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    const resetCalculator = () => {

  setInputs({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  setResults({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  setSelectedSuburbInfo(null);

  setFilteredSuburbs([]);

  setQuery('');

};

  useEffect(() => {

const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

const errors = {};

if (!inputs.clientName) {

  errors.clientName = 'Client Name is required.';

}

if (isNaN(price) || price <= 0) {

  errors.price = 'Vehicle Price must be a valid number greater than 0.';

}

if (isNaN(mm) || mm <= 0) {

  errors.mm = 'M&M Value must be a valid number greater than 0.';

}

if (!inputs.suburb) {

  errors.suburb = 'Please select a Suburb.';

}

if (!selectedSuburbInfo) {

  errors.selectedSuburbInfo = 'Selected Suburb is invalid or not found in the database.';

}

if (!inputs.riskProfile) {

  errors.riskProfile = 'Please select a Risk Profile.';

}

if (isNaN(terms) || terms <= 0) {

  errors.terms = 'Terms must be a valid number greater than 0.';

}

if (isNaN(inputs.creditScore) || inputs.creditScore < 0 || inputs.creditScore > 999) {

  errors.creditScore = 'Credit Score must be a number between 0 and 999.';

}

if (Object.keys(errors).length > 0) {

  Object.values(errors).forEach(msg => toast.error(msg));

  setInputErrors(errors);  // State to trigger red borders

  return;

}

// If all inputs are valid

setInputErrors({});  // Clear any previous errors

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCost = distance \* 10;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 0;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Client Name

        </label>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      </div>

         <div className="mb-3">

    <label htmlFor="vehiclePrice" className="block text-sm font-medium text-gray-700 mb-1">

      Vehicle Price

    </label>

    <div className="relative flex items-center">

      <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

      <input

        id="vehiclePrice"

        type="number"

        name="vehiclePrice"

        placeholder="Vehicle Price"

        value={inputs.vehiclePrice}

        onChange={handleChange}

        className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

      />

    </div>

  </div>

        <div className="relative mb-3">

          <label htmlFor="mmValue" className="block text-sm font-medium text-gray-700 mb-1">

            Retail Value

          </label>

            <div className="relative flex items-center">

            <span className="absolute left-3 text-gray-500 pointer-events-none">R</span>

          <input

            id="mmValue"

            type="number"

            name="mmValue"

            placeholder="Retail Value"

            value={inputs.mmValue}

            onChange={handleChange}

            className="w-full pl-7 p-2 border rounded focus:ring-2 focus:ring-blue-500 focus:outline-none"

         />

        </div>

       </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Suburb name

        </label>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      </div>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Risk Profile

        </label>

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      </div>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Credit Score

        </label>

        <input

          type="number"

          className="w-full p-2 border border-gray-300 rounded"

          value={creditScore}

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

          placeholder="e.g. 650"

          min="0"

          max="999"

        />

      </div>

      <div className="mb-3">

        <label className="block text-sm font-medium text-gray-700">Terms in months</label>

        <input

          type="number"

          name="termsInMonths"

          placeholder="Enter terms in months"

          value={inputs.termsInMonths}

          onChange={handleChange}

          className="w-full p-2 border rounded mt-1"

        />

      </div>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

<button

  onClick={resetCalculator}

  className="w-full bg-gray-600 text-white py-2 rounded mb-4 hover:bg-gray-700"

  disabled={loadingCalc}

>

  Reset

</button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/06/02 14:31

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 1000;

const EXPIRY\_DAYS = 365;

const Calculator = ({ suburbs }) => {

  const [creditScore, setCreditScore] = useState('');

  const [isAdmin, setIsAdmin] = useState(false);

      const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    const resetCalculator = () => {

  setInputs({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  setResults({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  setSelectedSuburbInfo(null);

  setFilteredSuburbs([]);

  setQuery('');

};

  useEffect(() => {

const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

const errors = {};

if (!inputs.clientName) {

  errors.clientName = 'Client Name is required.';

}

if (isNaN(price) || price <= 0) {

  errors.price = 'Vehicle Price must be a valid number greater than 0.';

}

if (isNaN(mm) || mm <= 0) {

  errors.mm = 'M&M Value must be a valid number greater than 0.';

}

if (!inputs.suburb) {

  errors.suburb = 'Please select a Suburb.';

}

if (!selectedSuburbInfo) {

  errors.selectedSuburbInfo = 'Selected Suburb is invalid or not found in the database.';

}

if (!inputs.riskProfile) {

  errors.riskProfile = 'Please select a Risk Profile.';

}

if (isNaN(terms) || terms <= 0) {

  errors.terms = 'Terms must be a valid number greater than 0.';

}

if (isNaN(inputs.creditScore) || inputs.creditScore < 0 || inputs.creditScore > 999) {

  errors.creditScore = 'Credit Score must be a number between 0 and 999.';

}

if (Object.keys(errors).length > 0) {

  Object.values(errors).forEach(msg => toast.error(msg));

  setInputErrors(errors);  // State to trigger red borders

  return;

}

// If all inputs are valid

setInputErrors({});  // Clear any previous errors

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCost = distance \* 10;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 0;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Credit Score

        </label>

        <input

          type="number"

          className="w-full p-2 border border-gray-300 rounded"

          value={creditScore}

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

          placeholder="e.g. 650"

          min="0"

          max="999"

        />

      </div>

      <div className="mb-3">

        <label className="block text-sm font-medium text-gray-700">Terms in months</label>

        <input

          type="number"

          name="termsInMonths"

          placeholder="Enter terms in months"

          value={inputs.termsInMonths}

          onChange={handleChange}

          className="w-full p-2 border rounded mt-1"

        />

      </div>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

<button

  onClick={resetCalculator}

  className="w-full bg-gray-600 text-white py-2 rounded mb-4 hover:bg-gray-700"

  disabled={loadingCalc}

>

  Reset

</button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/06/02 09:31 before increasing limits to 1000 and 365 days. Change formulas

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 1000;

const EXPIRY\_DAYS = 365;

const Calculator = ({ suburbs }) => {

  const [creditScore, setCreditScore] = useState('');

  const [isAdmin, setIsAdmin] = useState(false);

      const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    const resetCalculator = () => {

  setInputs({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  setResults({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  setSelectedSuburbInfo(null);

  setFilteredSuburbs([]);

  setQuery('');

};

  useEffect(() => {

const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 0

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

if (!inputs.clientName) {

  toast.error('Client Name is required.');

  return;

}

if (isNaN(price) || price <= 0) {

  toast.error('Vehicle Price must be a valid number greater than 0.');

  return;

}

if (isNaN(mm) || mm <= 0) {

  toast.error('M&M Value must be a valid number greater than 0.');

  return;

}

if (!inputs.suburb) {

  toast.error('Please select a Suburb.');

  return;

}

if (!selectedSuburbInfo) {

  toast.error('Selected Suburb is invalid or not found in the database.');

  return;

}

if (!inputs.riskProfile) {

  toast.error('Please select a Risk Profile.');

  return;

}

if (isNaN(terms) || terms <= 0) {

  toast.error('Terms must be a valid number greater than 0.');

  return;

}

if (isNaN(inputs.creditScore) || inputs.creditScore < 0  || inputs.creditScore > 999) {

  toast.error('Credit Score must be a valid number between 0 and 999.');

  return;

}

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCost = distance \* 10;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 0;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      <div className="mb-4">

        <label className="block text-sm font-medium text-gray-700 mb-1">

          Credit Score

        </label>

        <input

          type="number"

          className="w-full p-2 border border-gray-300 rounded"

          value={creditScore}

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

          placeholder="e.g. 650"

          min="0"

          max="999"

        />

      </div>

      <div className="mb-3">

        <label className="block text-sm font-medium text-gray-700">Terms in months</label>

        <input

          type="number"

          name="termsInMonths"

          placeholder="Enter terms in months"

          value={inputs.termsInMonths}

          onChange={handleChange}

          className="w-full p-2 border rounded mt-1"

        />

      </div>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

<button

  onClick={resetCalculator}

  className="w-full bg-gray-600 text-white py-2 rounded mb-4 hover:bg-gray-700"

  disabled={loadingCalc}

>

  Reset

</button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/26 20:29 Admin added

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [isAdmin, setIsAdmin] = useState(false);

  useEffect(() => {

    const handleKeyCombo = (e) => {

      if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

        setIsAdmin(true);

        toast.success('🔓 Admin access granted. Limit removed.');

      }

    };

    window.addEventListener('keydown', handleKeyCombo);

    return () => window.removeEventListener('keydown', handleKeyCombo);

  }, []);

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 580

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    // Enforce max entries unless admin

    if (!isAdmin && storedHistory.length >= MAX\_ENTRIES) {

      toast.error(`Maximum of ${MAX\_ENTRIES} calculations reached. Please wait or activate admin mode.`);

      return;

    }

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo ||

      isNaN(terms) || terms <= 0

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCostRaw = distance \* 10;

      const repoCost = repoCostRaw < 2000 ? 0 : repoCostRaw;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 580;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin + other;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['G19', `R${results.G19.toLocaleString()}`],

          ['I22', `R${results.I22.toLocaleString()}`],

          ['I21', `R${results.I21.toLocaleString()}`],

          ['G21 / Deposit', `R${results.deposit.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Base Payment', `R${results.monthlyBasePayment.toFixed(2)}`],

          ['Monthly Insurance', `R${results.monthlyInsurance.toFixed(2)}`],

          ['Profit Margin', `R${results.profitMargin.toFixed(2)}`],

          ['Other', `R${results.other}`],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

 <div className="mb-3">

        <label className="block text-sm font-medium text-gray-700">Terms in months</label>

        <input

          type="number"

          name="termsInMonths"

          placeholder="Enter terms in months"

          value={inputs.termsInMonths}

          onChange={handleChange}

          className="w-full p-2 border rounded mt-1"

        />

      </div>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/26 19:33

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const [isAdmin, setIsAdmin] = useState(false);

useEffect(() => {

  const handleKeyCombo = (e) => {

    if (e.ctrlKey && e.altKey && e.key.toLowerCase() === 't') {

      setIsAdmin(true);

      toast.success('🔓 Admin access granted. Limit removed.');

    }

  };

  window.addEventListener('keydown', handleKeyCombo);

  return () => window.removeEventListener('keydown', handleKeyCombo);

}, []);

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 580

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo ||

      isNaN(terms) || terms <= 0

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCostRaw = distance \* 10;

      const repoCost = repoCostRaw < 2000 ? 0 : repoCostRaw;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 580;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin + other;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['G19', `R${results.G19.toLocaleString()}`],

          ['I22', `R${results.I22.toLocaleString()}`],

          ['I21', `R${results.I21.toLocaleString()}`],

          ['G21 / Deposit', `R${results.deposit.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Base Payment', `R${results.monthlyBasePayment.toFixed(2)}`],

          ['Monthly Insurance', `R${results.monthlyInsurance.toFixed(2)}`],

          ['Profit Margin', `R${results.profitMargin.toFixed(2)}`],

          ['Other', `R${results.other}`],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

 <div className="mb-3">

        <label className="block text-sm font-medium text-gray-700">Terms in months</label>

        <input

          type="number"

          name="termsInMonths"

          placeholder="Enter terms in months"

          value={inputs.termsInMonths}

          onChange={handleChange}

          className="w-full p-2 border rounded mt-1"

        />

      </div>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/26 16:37

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 580

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo ||

      isNaN(terms) || terms <= 0

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCostRaw = distance \* 10;

      const repoCost = repoCostRaw < 2000 ? 0 : repoCostRaw;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 580;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin + other;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['G19', `R${results.G19.toLocaleString()}`],

          ['I22', `R${results.I22.toLocaleString()}`],

          ['I21', `R${results.I21.toLocaleString()}`],

          ['G21 / Deposit', `R${results.deposit.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Base Payment', `R${results.monthlyBasePayment.toFixed(2)}`],

          ['Monthly Insurance', `R${results.monthlyInsurance.toFixed(2)}`],

          ['Profit Margin', `R${results.profitMargin.toFixed(2)}`],

          ['Other', `R${results.other}`],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

 <div className="mb-3">

        <label className="block text-sm font-medium text-gray-700">Terms in months</label>

        <input

          type="number"

          name="termsInMonths"

          placeholder="Enter terms in months"

          value={inputs.termsInMonths}

          onChange={handleChange}

          className="w-full p-2 border rounded mt-1"

        />

      </div>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/26 15:15

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 580

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo ||

      isNaN(terms) || terms <= 0

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCostRaw = distance \* 10;

      const repoCost = repoCostRaw < 2000 ? 0 : repoCostRaw;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 580;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin + other;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['G19', `R${results.G19.toLocaleString()}`],

          ['I22', `R${results.I22.toLocaleString()}`],

          ['I21', `R${results.I21.toLocaleString()}`],

          ['G21 / Deposit', `R${results.deposit.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Base Payment', `R${results.monthlyBasePayment.toFixed(2)}`],

          ['Monthly Insurance', `R${results.monthlyInsurance.toFixed(2)}`],

          ['Profit Margin', `R${results.profitMargin.toFixed(2)}`],

          ['Other', `R${results.other}`],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

 <div className="mb-3">

        <label className="block text-sm font-medium text-gray-700">Terms in months</label>

        <input

          type="number"

          name="termsInMonths"

          placeholder="Enter terms in months"

          value={inputs.termsInMonths}

          onChange={handleChange}

          className="w-full p-2 border rounded mt-1"

        />

      </div>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator 2025/05/26 13:53

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 580

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo ||

      isNaN(terms) || terms <= 0

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCostRaw = distance \* 10;

      const repoCost = repoCostRaw < 2000 ? 0 : repoCostRaw;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 580;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin + other;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['G19', `R${results.G19.toLocaleString()}`],

          ['I22', `R${results.I22.toLocaleString()}`],

          ['I21', `R${results.I21.toLocaleString()}`],

          ['G21 / Deposit', `R${results.deposit.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Base Payment', `R${results.monthlyBasePayment.toFixed(2)}`],

          ['Monthly Insurance', `R${results.monthlyInsurance.toFixed(2)}`],

          ['Profit Margin', `R${results.profitMargin.toFixed(2)}`],

          ['Other', `R${results.other}`],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Terms in months</label>

          <input

            type="number"

            name="termsInMonths"

            placeholder="Enter terms in months"

            value={inputs.termsInMonths}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/25 18:22

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

    manualDeposit: '',

    termsInMonths: ''

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

    loading: 0,

    riskFactor: 0,

    totalRentalAmount: 0,

    G19: 0,

    I21: 0,

    I22: 0,

    G21: 0,

    netRentalAmount: 0,

    monthlyBasePayment: 0,

    monthlyInsurance: 0,

    profitMargin: 0,

    other: 580

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    const terms = parseInt(inputs.termsInMonths);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo ||

      isNaN(terms) || terms <= 0

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      // Loading

      const loading = price \* 0.10;

      // Risk Factor

      let riskFactor = loading;

      if (inputs.riskProfile === 'Low') riskFactor = loading \* 0.9;

      if (inputs.riskProfile === 'High') riskFactor = loading \* 1.1;

      // Total Rental Amount

      const totalRentalAmount = price + riskFactor;

      // Repo Cost

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const repoCostRaw = distance \* 10;

      const repoCost = repoCostRaw < 2000 ? 0 : repoCostRaw;

      // G19

      const G19 = totalRentalAmount \* 0.20 + repoCost;

      // I22

      const I22 = repoCost - 2000;

      // I21

      const I21 = totalRentalAmount - 110000 + 4000;

      // G21

      const G21 = I21 + I22;

      // Deposit

      const deposit = inputs.manualDeposit

        ? parseFloat(inputs.manualDeposit)

        : G21;

      // Net Rental Amount

      const netRentalAmount = totalRentalAmount - (deposit + repoCost);

      // Upfront Cost

      const licenseAndRegistration = 2500;

      const documentFees = 1500;

      const upfrontCost = netRentalAmount + licenseAndRegistration + documentFees;

      // Monthly base payment

      const monthlyBasePayment = upfrontCost / terms;

      // Monthly insurance

      const monthlyInsurance = mm \* 0.008167;

      // Profit Margin

      const profitMargin = monthlyBasePayment \* 1.05;

      // Other

      const other = 580;

      // Monthly Installment

      const monthlyInstallment =

        monthlyBasePayment + monthlyInsurance + profitMargin + other;

      const newResults = {

        loading,

        riskFactor,

        totalRentalAmount,

        repoCost,

        G19,

        I22,

        I21,

        G21,

        deposit,

        netRentalAmount,

        upfrontCost,

        monthlyBasePayment,

        monthlyInsurance,

        profitMargin,

        other,

        monthlyInstallment

      };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Loading', `R${results.loading.toLocaleString()}`],

          ['Risk Factor', `R${results.riskFactor.toLocaleString()}`],

          ['Total Rental Amount', `R${results.totalRentalAmount.toLocaleString()}`],

          ['Repo Cost', `R${results.repoCost.toLocaleString()}`],

          ['G19', `R${results.G19.toLocaleString()}`],

          ['I22', `R${results.I22.toLocaleString()}`],

          ['I21', `R${results.I21.toLocaleString()}`],

          ['G21 / Deposit', `R${results.deposit.toLocaleString()}`],

          ['Manual Deposit', inputs.manualDeposit ? `R${inputs.manualDeposit}` : ''],

          ['Net Rental Amount', `R${results.netRentalAmount.toLocaleString()}`],

          ['License & Registration', `R2500`],

          ['Document Fees', `R1500`],

          ['Upfront Cost', `R${results.upfrontCost.toLocaleString()}`],

          ['Terms (months)', inputs.termsInMonths],

          ['Monthly Base Payment', `R${results.monthlyBasePayment.toFixed(2)}`],

          ['Monthly Insurance', `R${results.monthlyInsurance.toFixed(2)}`],

          ['Profit Margin', `R${results.profitMargin.toFixed(2)}`],

          ['Other', `R${results.other}`],

          ['Monthly Installment', `R${results.monthlyInstallment.toFixed(2)}`]

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

        <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

        <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

      </div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Manual Deposit</label>

          <input

            type="number"

            name="manualDeposit"

            placeholder="Enter manual deposit"

            value={inputs.manualDeposit}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Net Rental Amount: <strong>R{results.netRentalAmount.toLocaleString()}</strong></p>

        <p>Upfront Cost: <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <div className="mb-3">

          <label className="block text-sm font-medium text-gray-700">Terms in months</label>

          <input

            type="number"

            name="termsInMonths"

            placeholder="Enter terms in months"

            value={inputs.termsInMonths}

            onChange={handleChange}

            className="w-full p-2 border rounded mt-1"

          />

        </div>

        <p>Monthly Base Payment: <strong>R{results.monthlyBasePayment.toFixed(2)}</strong></p>

        <p>Monthly Insurance: <strong>R{results.monthlyInsurance.toFixed(2)}</strong></p>

        <p>Profit Margin: <strong>R{results.profitMargin.toFixed(2)}</strong></p>

        <p>Other: <strong>R{results.other}</strong></p>

        <p>Monthly Installment: <strong>R{results.monthlyInstallment.toFixed(2)}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/22 22:22

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

    const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const riskMultiplier =

        inputs.riskProfile === 'High' ? 1.3 :

        inputs.riskProfile === 'Medium' ? 1.15 : 1.0;

      const deposit = Math.round(price \* 0.1 + distance \* 30 \* riskMultiplier);

      const repoCost = Math.round(mm \* 0.05);

      const upfrontCost = Math.min(deposit + repoCost, 110000);

      const monthlyInstallment = Math.round((price - deposit) / 54);

      const newResults = { deposit, repoCost, upfrontCost, monthlyInstallment };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Deposit', `R${results.deposit}`],

          ['Repo Cost', `R${results.repoCost}`],

          ['Upfront Cost', `R${results.upfrontCost}`],

          ['Monthly Installment (54 months)', `R${results.monthlyInstallment}`],

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <div className="flex items-center mb-6 bg-white p-2 rounded shadow">

  <img src="/smart logo.jpg" alt="SmartRent Auto Logo" className="h-10 w-auto mr-3" />

  <h1 className="text-2xl font-bold text-gray-800">SmartRent Auto™ Calculator</h1>

</div>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Upfront Cost (max R110,000): <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment (54 months): <strong>R{results.monthlyInstallment.toLocaleString()}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/22 18:31

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

  });

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  const [query, setQuery] = useState('');

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    if (name === 'suburb') setSelectedSuburbInfo(null);

    setInputs(prev => ({ ...prev, [name]: value }));

  };

    const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const riskMultiplier =

        inputs.riskProfile === 'High' ? 1.3 :

        inputs.riskProfile === 'Medium' ? 1.15 : 1.0;

      const deposit = Math.round(price \* 0.1 + distance \* 30 \* riskMultiplier);

      const repoCost = Math.round(mm \* 0.05);

      const upfrontCost = Math.min(deposit + repoCost, 110000);

      const monthlyInstallment = Math.round((price - deposit) / 54);

      const newResults = { deposit, repoCost, upfrontCost, monthlyInstallment };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Deposit', `R${results.deposit}`],

          ['Repo Cost', `R${results.repoCost}`],

          ['Upfront Cost', `R${results.upfrontCost}`],

          ['Monthly Installment (54 months)', `R${results.monthlyInstallment}`],

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <h1 className="text-3xl font-bold mb-6 text-center">SmartRent Auto™ Calculator</h1>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="vehiclePrice"

          placeholder="Vehicle Price"

          value={inputs.vehiclePrice}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <div className="relative mb-3">

        <span className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-500">R</span>

        <input

          type="number"

          name="mmValue"

          placeholder="Retail Value"

          value={inputs.mmValue}

          onChange={handleChange}

          className="w-full pl-7 p-2 border rounded"

        />

      </div>

      <Combobox value={inputs.suburb} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Upfront Cost (max R110,000): <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment (54 months): <strong>R{results.monthlyInstallment.toLocaleString()}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;

Calculator.jsx 2025/05/22 16:19

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

import jsPDF from 'jspdf';

import autoTable from 'jspdf-autotable';

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

import { Combobox } from '@headlessui/react';

const MAX\_ENTRIES = 10;

const EXPIRY\_DAYS = 7;

const Calculator = ({ suburbs }) => {

  const [inputs, setInputs] = useState({

    clientName: '',

    vehiclePrice: '',

    mmValue: '',

    suburb: '',

    riskProfile: '',

  });

  const [results, setResults] = useState({

    deposit: 0,

    repoCost: 0,

    upfrontCost: 0,

    monthlyInstallment: 0,

  });

  const [query, setQuery] = useState('');

  const [filteredSuburbs, setFilteredSuburbs] = useState([]);

  const [selectedSuburbInfo, setSelectedSuburbInfo] = useState(null);

  const [history, setHistory] = useState([]);

  const [loadingPdf, setLoadingPdf] = useState(false);

  const [loadingCalc, setLoadingCalc] = useState(false);

  useEffect(() => {

    const now = new Date();

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    setHistory(validHistory);

    localStorage.setItem('calculatorHistory', JSON.stringify(validHistory));

  }, []);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setInputs(prev => ({ ...prev, [name]: value }));

  };

  const handleSuburbInputChange = (value) => {

    setQuery(value);

    setInputs(prev => ({ ...prev, suburb: value }));

    setSelectedSuburbInfo(null);

    if (value === '') {

      setFilteredSuburbs([]);

      return;

    }

    const filtered = suburbs.filter(sub =>

      sub.SP\_NAME.toLowerCase().includes(value.toLowerCase())

    ).slice(0, 10);

    setFilteredSuburbs(filtered);

  };

  const handleSuburbSelect = (value) => {

    setQuery(value.SP\_NAME);

    setInputs(prev => ({ ...prev, suburb: value.SP\_NAME }));

    setSelectedSuburbInfo(value);

    setFilteredSuburbs([]);

  };

  const saveSubmissionToLocalStorage = (submission) => {

    const now = new Date();

    const timestampedSubmission = { ...submission, timestamp: now.toISOString() };

    const storedHistory = JSON.parse(localStorage.getItem('calculatorHistory')) || [];

    const validHistory = storedHistory.filter((entry) => {

      const entryDate = new Date(entry.timestamp);

      const diffInDays = (now - entryDate) / (1000 \* 60 \* 60 \* 24);

      return diffInDays <= EXPIRY\_DAYS;

    });

    const updatedHistory = [timestampedSubmission, ...validHistory].slice(0, MAX\_ENTRIES);

    localStorage.setItem('calculatorHistory', JSON.stringify(updatedHistory));

    setHistory(updatedHistory);

  };

  const calculate = () => {

    const price = parseFloat(inputs.vehiclePrice);

    const mm = parseFloat(inputs.mmValue);

    if (

      !inputs.clientName ||

      isNaN(price) || price <= 0 ||

      isNaN(mm) || mm <= 0 ||

      !inputs.suburb ||

      !inputs.riskProfile ||

      !selectedSuburbInfo

    ) {

      toast.error('Please fill in all fields correctly before calculating.');

      return;

    }

    setLoadingCalc(true);

    setTimeout(() => {

      const distance = parseFloat(selectedSuburbInfo.DIST\_KM);

      const riskMultiplier =

        inputs.riskProfile === 'High' ? 1.3 :

        inputs.riskProfile === 'Medium' ? 1.15 : 1.0;

      const deposit = Math.round(price \* 0.1 + distance \* 30 \* riskMultiplier);

      const repoCost = Math.round(mm \* 0.05);

      const upfrontCost = Math.min(deposit + repoCost, 110000);

      const monthlyInstallment = Math.round((price - deposit) / 54);

      const newResults = { deposit, repoCost, upfrontCost, monthlyInstallment };

      setResults(newResults);

      saveSubmissionToLocalStorage({ inputs, results: newResults });

      toast.success('Calculation completed!');

      setLoadingCalc(false);

    }, 400);

  };

  const exportPDF = () => {

    setLoadingPdf(true);

    try {

      const doc = new jsPDF();

      doc.setFontSize(16);

      doc.text('SmartRent Auto™ - Calculation Summary', 14, 15);

      autoTable(doc, {

        startY: 25,

        head: [['Field', 'Value']],

        body: [

          ['Client Name', inputs.clientName],

          ['Vehicle Price', `R${inputs.vehiclePrice}`],

          ['Retail Value', `R${inputs.mmValue}`],

          ['Suburb', inputs.suburb],

          ['Town', selectedSuburbInfo?.MP\_NAME || ''],

          ['Municipality', selectedSuburbInfo?.DC\_NAME || ''],

          ['Province', selectedSuburbInfo?.Province || ''],

          ['Distance (km)', selectedSuburbInfo?.DIST\_KM || ''],

          ['Risk Profile', inputs.riskProfile],

          ['Deposit', `R${results.deposit}`],

          ['Repo Cost', `R${results.repoCost}`],

          ['Upfront Cost', `R${results.upfrontCost}`],

          ['Monthly Installment (54 months)', `R${results.monthlyInstallment}`],

        ],

      });

      doc.save('SmartRentAuto\_Calculation.pdf');

      toast.success('PDF exported successfully!');

    } catch (err) {

      toast.error('Failed to export PDF.');

    } finally {

      setLoadingPdf(false);

    }

  };

  return (

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

      <Toaster />

      <h1 className="text-3xl font-bold mb-6 text-center">SmartRent Auto™ Calculator</h1>

      <input

        type="text"

        name="clientName"

        placeholder="Client Name"

        value={inputs.clientName}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <input

        type="number"

        name="vehiclePrice"

        placeholder="Vehicle Price"

        value={inputs.vehiclePrice}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <input

        type="number"

        name="mmValue"

        placeholder="Retail Value"

        value={inputs.mmValue}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-3"

      />

      <Combobox value={selectedSuburbInfo} onChange={handleSuburbSelect}>

        <Combobox.Input

          placeholder="Start typing suburb..."

          className="w-full p-2 border rounded mb-3"

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

          displayValue={() => query}

        />

        <Combobox.Options className="border rounded shadow bg-white max-h-60 overflow-y-auto">

          {filteredSuburbs.map((suburb, idx) => (

            <Combobox.Option key={idx} value={suburb}>

              {suburb.SP\_NAME} ({suburb.MP\_NAME})

            </Combobox.Option>

          ))}

        </Combobox.Options>

      </Combobox>

      {selectedSuburbInfo && (

        <div className="text-sm text-gray-700 mb-4">

          <p><strong>Town:</strong> {selectedSuburbInfo.MP\_NAME}</p>

          <p><strong>Municipality:</strong> {selectedSuburbInfo.DC\_NAME}</p>

          <p><strong>Province:</strong> {selectedSuburbInfo.Province}</p>

          <p><strong>Distance (km):</strong> {selectedSuburbInfo.DIST\_KM}</p>

        </div>

      )}

      <select

        name="riskProfile"

        value={inputs.riskProfile}

        onChange={handleChange}

        className="w-full p-2 border rounded mb-4"

      >

        <option value="">Select Risk Profile</option>

        <option value="Low">Low</option>

        <option value="Medium">Medium</option>

        <option value="High">High</option>

      </select>

      <button

        onClick={calculate}

        className="w-full bg-blue-600 text-white py-2 rounded mb-4 hover:bg-blue-700"

        disabled={loadingCalc}

      >

        {loadingCalc ? 'Calculating...' : 'Calculate'}

      </button>

      <div className="bg-gray-100 p-4 rounded mb-6">

        <h2 className="font-semibold mb-2">Results</h2>

        <p>Deposit: <strong>R{results.deposit.toLocaleString()}</strong></p>

        <p>Repo Cost: <strong>R{results.repoCost.toLocaleString()}</strong></p>

        <p>Upfront Cost (max R110,000): <strong>R{results.upfrontCost.toLocaleString()}</strong></p>

        <p>Monthly Installment (54 months): <strong>R{results.monthlyInstallment.toLocaleString()}</strong></p>

      </div>

      <button

        onClick={exportPDF}

        className="w-full bg-green-600 text-white py-2 rounded hover:bg-green-700"

        disabled={loadingPdf}

      >

        {loadingPdf ? 'Exporting PDF...' : 'Export to PDF'}

      </button>

      <section className="mt-8">

        <h3 className="font-semibold mb-2">Calculation History</h3>

        {history.length === 0 ? (

          <p>No recent calculations.</p>

        ) : (

          <ul className="space-y-2 max-h-60 overflow-y-auto border rounded p-2 bg-white">

            {history.map((entry, idx) => (

              <li key={idx} className="p-2 border rounded hover:bg-gray-50">

                <div><strong>Client:</strong> {entry.inputs.clientName}</div>

                <div><strong>Suburb:</strong> {entry.inputs.suburb}</div>

                <div><strong>Vehicle Price:</strong> R{parseFloat(entry.inputs.vehiclePrice).toLocaleString()}</div>

                <div><strong>Risk:</strong> {entry.inputs.riskProfile}</div>

                <div><strong>Deposit:</strong> R{entry.results.deposit.toLocaleString()}</div>

              </li>

            ))}

          </ul>

        )}

      </section>

    </div>

  );

};

export default Calculator;