REACT FILES

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

import axios from 'axios';

import './App.css'; // You can add some basic CSS if you want

const API\_URL = 'http://localhost:3001/api'; // Match your backend port

function App() {

  // --- Authentication State ---

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const [username, setUsername] = useState('');

  const [password, setPassword] = useState('');

  const [authMessage, setAuthMessage] = useState(''); // For displaying signup/login messages

  // --- Library Data States ---

  const [books, setBooks] = useState([]);

  const [members, setMembers] = useState([]);

  const [publishers, setPublishers] = useState([]);

  const [suppliers, setSuppliers] = useState([]);

  const [borrowings, setBorrowings] = useState([]);

  // --- State for New Items ---

  const [newBook, setNewBook] = useState({ title: '', author: '', ISBN: '', genre: '', publication\_year: '', publisher\_id: '', supplier\_id: '' });

  const [newMember, setNewMember] = useState({ name: '', type: 'Student', contact\_info: '' });

  const [newPublisher, setNewPublisher] = useState({ name: '', contact\_info: '' });

  const [newSupplier, setNewSupplier] = useState({ name: '', contact\_info: '' });

  // --- State for Editing ---

  const [editingBookId, setEditingBookId] = useState(null);

  const [editedBook, setEditedBook] = useState({ title: '', author: '', ISBN: '', genre: '', publication\_year: '', publisher\_id: '', supplier\_id: '' });

  const [editingMemberId, setEditingMemberId] = useState(null);

  const [editedMember, setEditedMember] = useState({ name: '', type: 'Student', contact\_info: '' });

  const [editingPublisherId, setEditingPublisherId] = useState(null);

  const [editedPublisher, setEditedPublisher] = useState({ name: '', contact\_info: '' });

  const [editingSupplierId, setEditingSupplierId] = useState(null);

  const [editedSupplier, setEditedSupplier] = useState({ name: '', contact\_info: '' });

  // --- Fetch Data ---

  // Fetch data ONLY if logged in

  useEffect(() => {

    if (isLoggedIn) {

      fetchBooks();

      fetchMembers();

      fetchPublishers();

      fetchSuppliers();

      fetchBorrowings();

    }

  }, [isLoggedIn]); // Dependency array: refetch when isLoggedIn changes

  const fetchBooks = async () => {

    try {

      const response = await axios.get(`${API\_URL}/books`);

      setBooks(response.data);

    } catch (error) {

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

      // Handle error, e.g., show a message

    }

  };

  const fetchMembers = async () => {

    try {

      const response = await axios.get(`${API\_URL}/members`);

      setMembers(response.data);

    } catch (error) {

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

    }

  };

  const fetchPublishers = async () => {

    try {

      const response = await axios.get(`${API\_URL}/publishers`);

      setPublishers(response.data);

    } catch (error) {

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

    }

  };

  const fetchSuppliers = async () => {

    try {

      const response = await axios.get(`${API\_URL}/suppliers`);

      setSuppliers(response.data);

    } catch (error) {

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

    }

  };

  const fetchBorrowings = async () => {

    try {

      const response = await axios.get(`${API\_URL}/borrowings`);

      setBorrowings(response.data);

    } catch (error) {

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

    }

  };

  // --- Authentication Handlers ---

  const handleUsernameChange = (e) => {

    setUsername(e.target.value);

  };

  const handlePasswordChange = (e) => {

    setPassword(e.target.value);

  };

  const handleSignup = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post(`${API\_URL}/signup`, { username, password });

      setAuthMessage(response.data.message);

      // Clear form after successful signup

      setUsername('');

      setPassword('');

    } catch (error) {

      console.error('Signup error:', error.response ? error.response.data : error.message);

      setAuthMessage(error.response ? error.response.data.message : 'Signup failed');

    }

  };

  const handleLogin = async (e) => {

    e.preventDefault();

    try {

      const response = await axios.post(`${API\_URL}/login`, { username, password });

      setAuthMessage(response.data.message);

      setIsLoggedIn(true); // Set logged in state

      // Clear form after successful login

      setUsername('');

      setPassword('');

    } catch (error) {

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

      setAuthMessage(error.response ? error.response.data.message : 'Login failed');

    }

  };

  const handleLogout = () => {

    setIsLoggedIn(false);

    setAuthMessage('Logged out');

    // Clear any sensitive data if needed

    setBooks([]);

    setMembers([]);

    setPublishers([]);

    setSuppliers([]);

    setBorrowings([]);

  };

  // --- Handle Form Changes (for adding/editing library items) ---

  const handleNewBookChange = (e) => {

    setNewBook({ ...newBook, [e.target.name]: e.target.value });

  };

  const handleNewMemberChange = (e) => {

    setNewMember({ ...newMember, [e.target.name]: e.target.value });

  };

  const handleNewPublisherChange = (e) => {

    setNewPublisher({ ...newPublisher, [e.target.name]: e.target.value });

  };

  const handleNewSupplierChange = (e) => {

    setNewSupplier({ ...newSupplier, [e.target.name]: e.target.value });

  };

  const handleEditedBookChange = (e) => {

    setEditedBook({ ...editedBook, [e.target.name]: e.target.value });

  };

  const handleEditedMemberChange = (e) => {

    setEditedMember({ ...editedMember, [e.target.name]: e.target.value });

  };

  const handleEditedPublisherChange = (e) => {

    setEditedPublisher({ ...editedPublisher, [e.target.name]: e.target.value });

  };

  const handleEditedSupplierChange = (e) => {

    setEditedSupplier({ ...editedSupplier, [e.target.name]: e.target.value });

  };

  // --- Add Operations ---

  const addBook = async (e) => {

    e.preventDefault();

    try {

      await axios.post(`${API\_URL}/books`, newBook);

      setNewBook({ title: '', author: '', ISBN: '', genre: '', publication\_year: '', publisher\_id: '', supplier\_id: '' });

      fetchBooks(); // Refresh the list

    } catch (error) {

      console.error('Error adding book:', error);

      alert('Failed to add book'); // Simple error feedback

    }

  };

  const addMember = async (e) => {

    e.preventDefault();

    try {

      await axios.post(`${API\_URL}/members`, newMember);

      setNewMember({ name: '', type: 'Student', contact\_info: '' });

      fetchMembers(); // Refresh the list

    } catch (error) {

      console.error('Error adding member:', error);

      alert('Failed to add member');

    }

  };

  const addPublisher = async (e) => {

    e.preventDefault();

    try {

      await axios.post(`${API\_URL}/publishers`, newPublisher);

      setNewPublisher({ name: '', contact\_info: '' });

      fetchPublishers(); // Refresh the list

    } catch (error) {

      console.error('Error adding publisher:', error);

      alert('Failed to add publisher');

    }

  };

  const addSupplier = async (e) => {

    e.preventDefault();

    try {

      await axios.post(`${API\_URL}/suppliers`, newSupplier);

      setNewSupplier({ name: '', contact\_info: '' });

      fetchSuppliers(); // Refresh the list

    } catch (error) {

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

      alert('Failed to add supplier');

    }

  };

  // --- Delete Operations ---

  const deleteBook = async (bookId) => {

    try {

      await axios.delete(`${API\_URL}/books/${bookId}`);

      fetchBooks(); // Refresh the list

    } catch (error) {

      console.error('Error deleting book:', error);

      alert('Failed to delete book');

    }

  };

  const deleteMember = async (memberId) => {

    try {

      await axios.delete(`${API\_URL}/members/${memberId}`);

      fetchMembers(); // Refresh the list

    } catch (error) {

      console.error('Error deleting member:', error);

      alert('Failed to delete member');

    }

  };

  const deletePublisher = async (publisherId) => {

    try {

      await axios.delete(`${API\_URL}/publishers/${publisherId}`);

      fetchPublishers(); // Refresh the list

    } catch (error) {

      console.error('Error deleting publisher:', error);

      alert('Failed to delete publisher');

    }

  };

  const deleteSupplier = async (supplierId) => {

    try {

      await axios.delete(`${API\_URL}/suppliers/${supplierId}`);

      fetchSuppliers(); // Refresh the list

    } catch (error) {

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

      alert('Failed to delete supplier');

    }

  };

  // --- Update Operations (Basic Implementation) ---

  const startEditingBook = (book) => {

    setEditingBookId(book.book\_id);

    setEditedBook({ ...book }); // Copy book data to the edit state

  };

  const cancelEditingBook = () => {

    setEditingBookId(null);

    setEditedBook({ title: '', author: '', ISBN: '', genre: '', publication\_year: '', publisher\_id: '', supplier\_id: '' });

  };

  const updateBook = async (e) => {

    e.preventDefault();

    try {

      await axios.put(`${API\_URL}/books/${editingBookId}`, editedBook);

      cancelEditingBook(); // Exit editing mode

      fetchBooks(); // Refresh the list

    } catch (error) {

      console.error('Error updating book:', error);

      alert('Failed to update book');

    }

  };

  const startEditingMember = (member) => {

    setEditingMemberId(member.member\_id);

    setEditedMember({ ...member });

  };

  const cancelEditingMember = () => {

    setEditingMemberId(null);

    setEditedMember({ name: '', type: 'Student', contact\_info: '' });

  };

  const updateMember = async (e) => {

    e.preventDefault();

    try {

      await axios.put(`${API\_URL}/members/${editingMemberId}`, editedMember);

      cancelEditingMember();

      fetchMembers();

    } catch (error) {

      console.error('Error updating member:', error);

      alert('Failed to update member');

    }

  };

  const startEditingPublisher = (publisher) => {

    setEditingPublisherId(publisher.publisher\_id);

    setEditedPublisher({ ...publisher });

  };

  const cancelEditingPublisher = () => {

    setEditingPublisherId(null);

    setEditedPublisher({ name: '', contact\_info: '' });

  };

  const updatePublisher = async (e) => {

    e.preventDefault();

    try {

      await axios.put(`${API\_URL}/publishers/${editingPublisherId}`, editedPublisher);

      cancelEditingPublisher();

      fetchPublishers();

    } catch (error) {

      console.error('Error updating publisher:', error);

      alert('Failed to update publisher');

    }

  };

  const startEditingSupplier = (supplier) => {

    setEditingSupplierId(supplier.supplier\_id);

    setEditedSupplier({ ...supplier });

  };

  const cancelEditingSupplier = () => {

    setEditingSupplierId(null);

    setEditedSupplier({ name: '', contact\_info: '' });

  };

  const updateSupplier = async (e) => {

    e.preventDefault();

    try {

      await axios.put(`${API\_URL}/suppliers/${editingSupplierId}`, editedSupplier);

      cancelEditingSupplier();

      fetchSuppliers();

    } catch (error) {

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

      alert('Failed to update supplier');

    }

  };

  return (

    <div className="App">

      <h1>Library Management System</h1>

      {!isLoggedIn ? (

        // --- Authentication Forms (Show when not logged in) ---

        <div>

          <h2>Signup or Login</h2>

          <form onSubmit={handleLogin}>

            <h3>Login</h3>

            <input

              type="text"

              placeholder="Username"

              value={username}

              onChange={handleUsernameChange}

              required

            />

            <input

              type="password"

              placeholder="Password"

              value={password}

              onChange={handlePasswordChange}

              required

            />

            <button type="submit">Login</button>

          </form>

          <form onSubmit={handleSignup}>

             <h3>Signup</h3>

             <input

               type="text"

               placeholder="Username"

               value={username}

               onChange={handleUsernameChange}

               required

             />

             <input

               type="password"

               placeholder="Password"

               value={password}

               onChange={handlePasswordChange}

               required

             />

             <button type="submit">Signup</button>

           </form>

          {authMessage && <p>{authMessage}</p>}

        </div>

      ) : (

        // --- Main Library Content (Show when logged in) ---

        <div>

          <h2>Welcome!</h2>

          <button onClick={handleLogout}>Logout</button>

          {/\* Add New Book Form \*/}

          <h2>Add New Book</h2>

          <form onSubmit={addBook}>

            <input type="text" name="title" placeholder="Title" value={newBook.title} onChange={handleNewBookChange} required />

            <input type="text" name="author" placeholder="Author" value={newBook.author} onChange={handleNewBookChange} />

            <input type="text" name="ISBN" placeholder="ISBN" value={newBook.ISBN} onChange={handleNewBookChange} />

            <input type="text" name="genre" placeholder="Genre" value={newBook.genre} onChange={handleNewBookChange} />

            <input type="number" name="publication\_year" placeholder="Publication Year" value={newBook.publication\_year} onChange={handleNewBookChange} />

            {/\* Use select for publisher and supplier IDs \*/}

            <select name="publisher\_id" value={newBook.publisher\_id} onChange={handleNewBookChange}>

               <option value="">Select Publisher</option>

               {publishers.map(p => <option key={p.publisher\_id} value={p.publisher\_id}>{p.name}</option>)}

            </select>

            <select name="supplier\_id" value={newBook.supplier\_id} onChange={handleNewBookChange}>

               <option value="">Select Supplier</option>

               {suppliers.map(s => <option key={s.supplier\_id} value={s.supplier\_id}>{s.name}</option>)}

            </select>

            <button type="submit">Add Book</button>

          </form>

           {/\* Add New Member Form \*/}

          <h2>Add New Member</h2>

          <form onSubmit={addMember}>

            <input type="text" name="name" placeholder="Name" value={newMember.name} onChange={handleNewMemberChange} required />

            <select name="type" value={newMember.type} onChange={handleNewMemberChange}>

               <option value="Student">Student</option>

               <option value="Teacher">Teacher</option>

            </select>

            <input type="text" name="contact\_info" placeholder="Contact Info" value={newMember.contact\_info} onChange={handleNewMemberChange} />

            <button type="submit">Add Member</button>

          </form>

           {/\* Add New Publisher Form \*/}

          <h2>Add New Publisher</h2>

          <form onSubmit={addPublisher}>

            <input type="text" name="name" placeholder="Publisher Name" value={newPublisher.name} onChange={handleNewPublisherChange} required />

            <input type="text" name="contact\_info" placeholder="Contact Info" value={newPublisher.contact\_info} onChange={handleNewPublisherChange} />

            <button type="submit">Add Publisher</button>

          </form>

           {/\* Add New Supplier Form \*/}

          <h2>Add New Supplier</h2>

          <form onSubmit={addSupplier}>

            <input type="text" name="name" placeholder="Supplier Name" value={newSupplier.name} onChange={handleNewSupplierChange} required />

            <input type="text" name="contact\_info" placeholder="Contact Info" value={newSupplier.contact\_info} onChange={handleNewSupplierChange} />

            <button type="submit">Add Supplier</button>

          </form>

          {/\* Book List \*/}

          <h2>Books</h2>

          <ul>

            {books.map(book => (

              <li key={book.book\_id}>

                {editingBookId === book.book\_id ? (

                  // Edit form for the selected book

                  <form onSubmit={updateBook}>

                     <input type="text" name="title" value={editedBook.title} onChange={handleEditedBookChange} required />

                     <input type="text" name="author" value={editedBook.author} onChange={handleEditedBookChange} />

                     <input type="text" name="ISBN" value={editedBook.ISBN} onChange={handleEditedBookChange} />

                     <input type="text" name="genre" value={editedBook.genre} onChange={handleEditedBookChange} />

                     <input type="number" name="publication\_year" value={editedBook.publication\_year} onChange={handleEditedBookChange} />

                     <select name="publisher\_id" value={editedBook.publisher\_id} onChange={handleEditedBookChange}>

                        <option value="">Select Publisher</option>

                        {publishers.map(p => <option key={p.publisher\_id} value={p.publisher\_id}>{p.name}</option>)}

                     </select>

                     <select name="supplier\_id" value={editedBook.supplier\_id} onChange={handleEditedBookChange}>

                        <option value="">Select Supplier</option>

                        {suppliers.map(s => <option key={s.supplier\_id} value={s.supplier\_id}>{s.name}</option>)}

                     </select>

                     <button type="submit">Save</button>

                     <button type="button" onClick={cancelEditingBook}>Cancel</button>

                  </form>

                ) : (

                  // Display mode

                  <>

                    {book.title} by {book.author} (ISBN: {book.ISBN}) [Publisher: {book.publisher\_name || 'N/A'}, Supplier: {book.supplier\_name || 'N/A'}]

                    <button onClick={() => startEditingBook(book)}>Edit</button>

                    <button onClick={() => deleteBook(book.book\_id)}>Delete</button>

                  </>

                )}

              </li>

            ))}

          </ul>

           {/\* Member List \*/}

          <h2>Members</h2>

          <ul>

            {members.map(member => (

               <li key={member.member\_id}>

                {editingMemberId === member.member\_id ? (

                  // Edit form for the selected member

                  <form onSubmit={updateMember}>

                     <input type="text" name="name" value={editedMember.name} onChange={handleEditedMemberChange} required />

                     <select name="type" value={editedMember.type} onChange={handleEditedMemberChange}>

                        <option value="Student">Student</option>

                        <option value="Teacher">Teacher</option>

                     </select>

                     <input type="text" name="contact\_info" value={editedMember.contact\_info} onChange={handleEditedMemberChange} />

                     <button type="submit">Save</button>

                     <button type="button" onClick={cancelEditingMember}>Cancel</button>

                  </form>

                ) : (

                  // Display mode

                  <>

                    {member.name} ({member.type}) - {member.contact\_info}

                    <button onClick={() => startEditingMember(member)}>Edit</button>

                    <button onClick={() => deleteMember(member.member\_id)}>Delete</button>

                  </>

                )}

              </li>

            ))}

          </ul>

           {/\* Publisher List \*/}

          <h2>Publishers</h2>

          <ul>

            {publishers.map(publisher => (

               <li key={publisher.publisher\_id}>

                {editingPublisherId === publisher.publisher\_id ? (

                  // Edit form for the selected publisher

                  <form onSubmit={updatePublisher}>

                     <input type="text" name="name" value={editedPublisher.name} onChange={handleEditedPublisherChange} required />

                     <input type="text" name="contact\_info" value={editedPublisher.contact\_info} onChange={handleEditedPublisherChange} />

                     <button type="submit">Save</button>

                     <button type="button" onClick={cancelEditingPublisher}>Cancel</button>

                  </form>

                ) : (

                  // Display mode

                  <>

                    {publisher.name} - {publisher.contact\_info}

                    <button onClick={() => startEditingPublisher(publisher)}>Edit</button>

                    <button onClick={() => deletePublisher(publisher.publisher\_id)}>Delete</button>

                  </>

                )}

              </li>

            ))}

          </ul>

           {/\* Supplier List \*/}

          <h2>Suppliers</h2>

          <ul>

            {suppliers.map(supplier => (

               <li key={supplier.supplier\_id}>

                {editingSupplierId === supplier.supplier\_id ? (

                  // Edit form for the selected supplier

                  <form onSubmit={updateSupplier}>

                     <input type="text" name="name" value={editedSupplier.name} onChange={handleEditedSupplierChange} required />

                     <input type="text" name="contact\_info" value={editedSupplier.contact\_info} onChange={handleEditedSupplierChange} />

                     <button type="submit">Save</button>

                     <button type="button" onClick={cancelEditingSupplier}>Cancel</button>

                  </form>

                ) : (

                  // Display mode

                  <>

                    {supplier.name} - {supplier.contact\_info}

                    <button onClick={() => startEditingSupplier(supplier)}>Edit</button>

                    <button onClick={() => deleteSupplier(supplier.supplier\_id)}>Delete</button>

                  </>

                )}

              </li>

            ))}

          </ul>

           {/\* Borrowing List (Read Only) \*/}

          <h2>Borrowings</h2>

          <p>Note: Full CRUD for Borrowings is more complex and omitted here for simplicity.</p>

          <ul>

            {borrowings.map(borrowing => (

              <li key={borrowing.borrowing\_id}>

                 {/\* Display book and member names from the joined data \*/}

                Borrowing ID: {borrowing.borrowing\_id}, Book: {borrowing.book\_title}, Member: {borrowing.member\_name}, Borrow Date: {borrowing.borrow\_date}, Return Date: {borrowing.return\_date || 'Not Returned'}

              </li>

            ))}

          </ul>

        </div>

      )}

    </div>

  );

}

export default App;

node.js

const express = require('express');

const mysql = require('mysql');

const cors = require('cors');

const app = express();

const port = 3001; // Ensure this matches your frontend API\_URL

// Middleware

app.use(cors()); // Enable CORS for all origins (for development)

app.use(express.json()); // Parse JSON request bodies

// Database Connection

const db = mysql.createConnection({

    host: 'localhost', // Your MySQL host

    user: 'root', // Your MySQL username

    password: '', // Your MySQL password

    database: 'library\_system' // The database you created

});

db.connect((err) => {

    if (err) {

        console.error('Error connecting to the database:', err.stack);

        return;

    }

    console.log('Connected to the database as ID ' + db.threadId);

});

// --- IMPORTANT: SQL for the 'users' table ---

// Make sure you have created this table in your 'library\_system' database:

/\*

CREATE TABLE users (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    username VARCHAR(50) UNIQUE,

    password VARCHAR(255)

);

\*/

// NOTE: This basic example stores passwords in plain text.

// In a real application, you MUST hash passwords for security (e.g., using bcrypt).

// Helper function to execute SQL queries (basic error logging)

const executeQuery = (sql, params, callback) => {

    db.query(sql, params, (err, results) => {

        if (err) {

            console.error('Database query error:', err.message);

        }

        callback(err, results);

    });

};

// --- User Authentication Endpoints ---

// Signup endpoint

app.post('/api/signup', (req, res) => {

    const { username, password } = req.body;

    if (!username || !password) {

        return res.status(400).json({ message: 'Username and password are required' });

    }

    // IMPORTANT: Storing plain text password - NOT SECURE for production!

    const sql = 'INSERT INTO users (username, password) VALUES (?, ?)';

    executeQuery(sql, [username, password], (err, result) => {

        if (err) {

            if (err.code === 'ER\_DUP\_ENTRY') {

                return res.status(400).json({ message: 'Username already exists' });

            }

            console.error('Signup error:', err); // Log other errors

            return res.status(500).json({ message: 'Error creating user' });

        }

        res.status(201).json({ message: 'Signup successful' });

    });

});

// Login endpoint

app.post('/api/login', (req, res) => {

    const { username, password } = req.body;

    if (!username || !password) {

        return res.status(400).json({ message: 'Username and password are required' });

    }

    const sql = 'SELECT \* FROM users WHERE username = ?';

    executeQuery(sql, [username], (err, results) => {

        if (err) {

            console.error('Login query error:', err); // Log query error

            return res.status(500).json({ message: 'Error during login' });

        }

        if (results.length === 0) {

            // User not found

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

        }

        const user = results[0];

        // IMPORTANT: Comparing plain text password - NOT SECURE for production!

        if (user.password !== password) {

            // Password doesn't match

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

        }

        // Login successful

        res.json({ message: 'Login successful', username: user.username }); // Optionally send back user info

    });

});

// --- Book API Endpoints ---

// Get all books (with publisher and supplier names)

app.get('/api/books', (req, res) => {

    const sql = 'SELECT b.\*, p.name AS publisher\_name, s.name AS supplier\_name FROM Book b LEFT JOIN Publisher p ON b.publisher\_id = p.publisher\_id LEFT JOIN Supplier s ON b.supplier\_id = s.supplier\_id';

    executeQuery(sql, [], (err, results) => {

        if (err) {

            res.status(500).json({ error: 'Failed to fetch books' });

            return;

        }

        res.json(results);

    });

});

// Create a new book

app.post('/api/books', (req, res) => {

    const { title, author, ISBN, genre, publication\_year, publisher\_id, supplier\_id } = req.body;

    const sql = 'INSERT INTO Book (title, author, ISBN, genre, publication\_year, publisher\_id, supplier\_id) VALUES (?, ?, ?, ?, ?, ?, ?)';

    const values = [title, author, ISBN, genre, publication\_year, publisher\_id, supplier\_id];

    executeQuery(sql, values, (err, result) => {

        if (err) {

            res.status(500).json({ error: 'Failed to add book' });

            return;

        }

        res.status(201).json({ message: 'Book created successfully', book\_id: result.insertId });

    });

});

// Update a book by ID

app.put('/api/books/:id', (req, res) => {

    const { id } = req.params;

    const { title, author, ISBN, genre, publication\_year, publisher\_id, supplier\_id } = req.body;

    const sql = 'UPDATE Book SET title = ?, author = ?, ISBN = ?, genre = ?, publication\_year = ?, publisher\_id = ?, supplier\_id = ? WHERE book\_id = ?';

    const values = [title, author, ISBN, genre, publication\_year, publisher\_id, supplier\_id, id];

    executeQuery(sql, values, (err, result) => {

        if (err) {

            res.status(500).json({ error: 'Failed to update book' });

            return;

        }

        if (result.affectedRows === 0) {

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

            return;

        }

        res.json({ message: 'Book updated successfully' });

    });

});

// Delete a book by ID

app.delete('/api/books/:id', (req, res) => {

    const { id } = req.params;

    const sql = 'DELETE FROM Book WHERE book\_id = ?';

    executeQuery(sql, [id], (err, result) => {

        if (err) {

            res.status(500).json({ error: 'Failed to delete book' });

            return;

        }

        if (result.affectedRows === 0) {

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

            return;

        }

        res.json({ message: 'Book deleted successfully' });

    });

});

// --- Member API Endpoints ---

// Get all members

app.get('/api/members', (req, res) => {

    const sql = 'SELECT \* FROM Member';

    executeQuery(sql, [], (err, results) => {

        if (err) {

            res.status(500).json({ error: 'Failed to fetch members' });

            return;

        }

        res.json(results);

    });

});

// Create a new member

app.post('/api/members', (req, res) => {

    const { name, type, contact\_info } = req.body;

    const sql = 'INSERT INTO Member (name, type, contact\_info) VALUES (?, ?, ?)';

    const values = [name, type, contact\_info];

    executeQuery(sql, values, (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to add member' });

            return;

        }

        res.status(201).json({ message: 'Member created successfully', member\_id: result.insertId });

    });

});

// Update a member by ID

app.put('/api/members/:id', (req, res) => {

    const { id } = req.params;

    const { name, type, contact\_info } = req.body;

    const sql = 'UPDATE Member SET name = ?, type = ?, contact\_info = ? WHERE member\_id = ?';

    const values = [name, type, contact\_info, id];

    executeQuery(sql, values, (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to update member' });

            return;

        }

         if (result.affectedRows === 0) {

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

            return;

        }

        res.json({ message: 'Member updated successfully' });

    });

});

// Delete a member by ID

app.delete('/api/members/:id', (req, res) => {

    const { id } = req.params;

    const sql = 'DELETE FROM Member WHERE member\_id = ?';

    executeQuery(sql, [id], (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to delete member' });

            return;

        }

         if (result.affectedRows === 0) {

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

            return;

        }

        res.json({ message: 'Member deleted successfully' });

    });

});

// --- Publisher API Endpoints ---

// Get all publishers

app.get('/api/publishers', (req, res) => {

    const sql = 'SELECT \* FROM Publisher';

    executeQuery(sql, [], (err, results) => {

        if (err) {

             res.status(500).json({ error: 'Failed to fetch publishers' });

            return;

        }

        res.json(results);

    });

});

// Create a new publisher

app.post('/api/publishers', (req, res) => {

    const { name, contact\_info } = req.body;

    const sql = 'INSERT INTO Publisher (name, contact\_info) VALUES (?, ?)';

    const values = [name, contact\_info];

    executeQuery(sql, values, (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to add publisher' });

            return;

        }

        res.status(201).json({ message: 'Publisher created successfully', publisher\_id: result.insertId });

    });

});

// Update a publisher by ID

app.put('/api/publishers/:id', (req, res) => {

    const { id } = req.params;

    const { name, contact\_info } = req.body;

    const sql = 'UPDATE Publisher SET name = ?, contact\_info = ? WHERE publisher\_id = ?';

    const values = [name, contact\_info, id];

    executeQuery(sql, values, (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to update publisher' });

            return;

        }

         if (result.affectedRows === 0) {

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

            return;

        }

        res.json({ message: 'Publisher updated successfully' });

    });

});

// Delete a publisher by ID

app.delete('/api/publishers/:id', (req, res) => {

    const { id } = req.params;

    const sql = 'DELETE FROM Publisher WHERE publisher\_id = ?';

    executeQuery(sql, [id], (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to delete publisher' });

            return;

        }

         if (result.affectedRows === 0) {

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

            return;

        }

        res.json({ message: 'Publisher deleted successfully' });

    });

});

// --- Supplier API Endpoints ---

// Get all suppliers

app.get('/api/suppliers', (req, res) => {

    const sql = 'SELECT \* FROM Supplier';

    executeQuery(sql, [], (err, results) => {

        if (err) {

             res.status(500).json({ error: 'Failed to fetch suppliers' });

            return;

        }

        res.json(results);

    });

});

// Create a new supplier

app.post('/api/suppliers', (req, res) => {

    const { name, contact\_info } = req.body;

    const sql = 'INSERT INTO Supplier (name, contact\_info) VALUES (?, ?)';

    const values = [name, contact\_info];

    executeQuery(sql, values, (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to add supplier' });

            return;

        }

        res.status(201).json({ message: 'Supplier created successfully', supplier\_id: result.insertId });

    });

});

// Update a supplier by ID

app.put('/api/suppliers/:id', (req, res) => {

    const { id } = req.params;

    const { name, contact\_info } = req.body;

    const sql = 'UPDATE Supplier SET name = ?, contact\_info = ? WHERE supplier\_id = ?';

    const values = [name, contact\_info, id];

    executeQuery(sql, values, (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to update supplier' });

            return;

        }

         if (result.affectedRows === 0) {

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

            return;

        }

        res.json({ message: 'Supplier updated successfully' });

    });

});

// Delete a supplier by ID

app.delete('/api/suppliers/:id', (req, res) => {

    const { id } = req.params;

    const sql = 'DELETE FROM Supplier WHERE supplier\_id = ?';

    executeQuery(sql, [id], (err, result) => {

        if (err) {

             res.status(500).json({ error: 'Failed to delete supplier' });

            return;

        }

         if (result.affectedRows === 0) {

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

            return;

        }

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

    });

});

// --- Borrowing API Endpoints (Read Only for Simplicity) ---

// Get all borrowings (joining with Book and Member for display)

app.get('/api/borrowings', (req, res) => {

    const sql = `

        SELECT

            b.borrowing\_id,

            bk.title AS book\_title,

            m.name AS member\_name,

            b.borrow\_date,

            b.return\_date

        FROM Borrowing b

        JOIN Book bk ON b.book\_id = bk.book\_id

        JOIN Member m ON b.member\_id = m.member\_id

    `;

    executeQuery(sql, [], (err, results) => {

        if (err) {

             res.status(500).json({ error: 'Failed to fetch borrowings' });

            return;

        }

        res.json(results);

    });

});

// Start the server

app.listen(port, () => {

    console.log(`Backend server running on http://localhost:${port}`);

});

// Close the database connection when the server stops (optional, but good practice)

process.on('SIGINT', () => {

    db.end((err) => {

        if (err) {

            console.error('Error closing database connection:', err.stack);

            return;

        }

        console.log('Database connection closed.');

        process.exit(0);

    });

});