12. Build an online Book Store project using MongoDB, Express.js, React, Node.js (MERN)

Server/app.js

```
 * To run this file:
 * 1. Ensure you have Node.js installed.
 * 2. Create a new directory for your project.
 * 3. Save this code as app.js'.
 * 4. Run 'npm init -y' in your terminal.
 * 5. Run 'npm install express mongoose dotenv'.
 * 6. Create a file named '.env' and add your connection string:
 *
MONGO_URI="mongodb+srv://<username>:<password>@<cluster>.mongodb.net/<database
Name>" or “mongodb://127.0.0.1:27017/BOOKSTORE
7. Run 'node app.js'.
 *
 * API Endpoints:
 * - GET /api/books          -> Get all books
 * - GET /api/books/:id      -> Get a single book by ID
 * - POST /api/books         -> Create a new book (requires JSON body)
 * - PUT /api/books/:id      -> Update a book by ID (requires JSON body)
 * - DELETE /api/books/:id   -> Delete a book by ID
 */

const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const cors = require('cors');


// Load environment variables from .env file
dotenv.config();
const app = express();
app.use(cors())
const PORT = process.env.PORT || 5000;

// Middleware for parsing JSON body in requests
app.use(express.json());

// Simple welcome route
app.get('/', (req, res) => {
    res.send('Welcome to the Book Store API. Access book data via
/api/books');
});
// ------------------------------------
// 2. MONGOOSE SCHEMA AND MODEL
// ------------------------------------
// Define the Book Schema
const BookSchema = new mongoose.Schema({
    title: {
        type: String,
        required: [true, 'Book title is required'],
        trim: true,
    },
    author: {
        type: String,
        required: [true, 'Author name is required'],
        trim: true,
    },
    isbn: {
        type: String,
        required: [true, 'ISBN is required'],
        unique: true,
        trim: true,
    },
    price: {
        type: Number,
        required: [true, 'Price is required'],
        min: [0, 'Price cannot be negative'],
    },
    description: {
        type: String,
        required: false,
        default: 'No description provided.',
    },
    publishedDate: {
        type: Date,
        default: Date.now,
    },
    // Optional field for stock tracking
    stock: {
        type: Number,
        default: 1,
        min: [0, 'Stock cannot be negative'],
    }
}, {
    timestamps: true // Adds createdAt and updatedAt fields
});

// Create the Book Model
const Book = mongoose.model('Book', BookSchema);
// ------------------------------------
// 3. MONGODB CONNECTION
// ------------------------------------
```

```
const MONGO_URI = process.env.MONGO_URI;

if (!MONGO_URI) {
    console.error("FATAL ERROR: MONGO_URI not defined. Please create a .env
file.");
    process.exit(1);
}

const connectDB = async () => {
    try {
        const conn = await mongoose.connect(MONGO_URI, {
            // These options are now deprecated/defaulted in Mongoose 6+ but
are included for clarity
            // useNewUrlParser: true,
            // useUnifiedTopology: true,
        });
        console.log(`MongoDB Connected: ${conn.connection.host}`);
    } catch (error) {
        console.error(`Error connecting to MongoDB: ${error.message}`);
        process.exit(1); // Exit process with failure
    }
};

connectDB(); // Connect to the database


// ------------------------------------
// 4. API ROUTES (CRUD OPERATIONS)
// ------------------------------------

const bookRouter = express.Router();

// @route   GET /api/books
// @desc    Get all books
bookRouter.get('/', async (req, res) => {
    try {
        const books = await Book.find({}).sort({ title: 1 }); // Find all,
sort by title
        res.status(200).json(books);
    } catch (error) {
        // Handle database or server errors
        res.status(500).json({ message: 'Server error while fetching books',
error: error.message });
    }
});

// @route   GET /api/books/:id
// @desc    Get a single book by ID
```

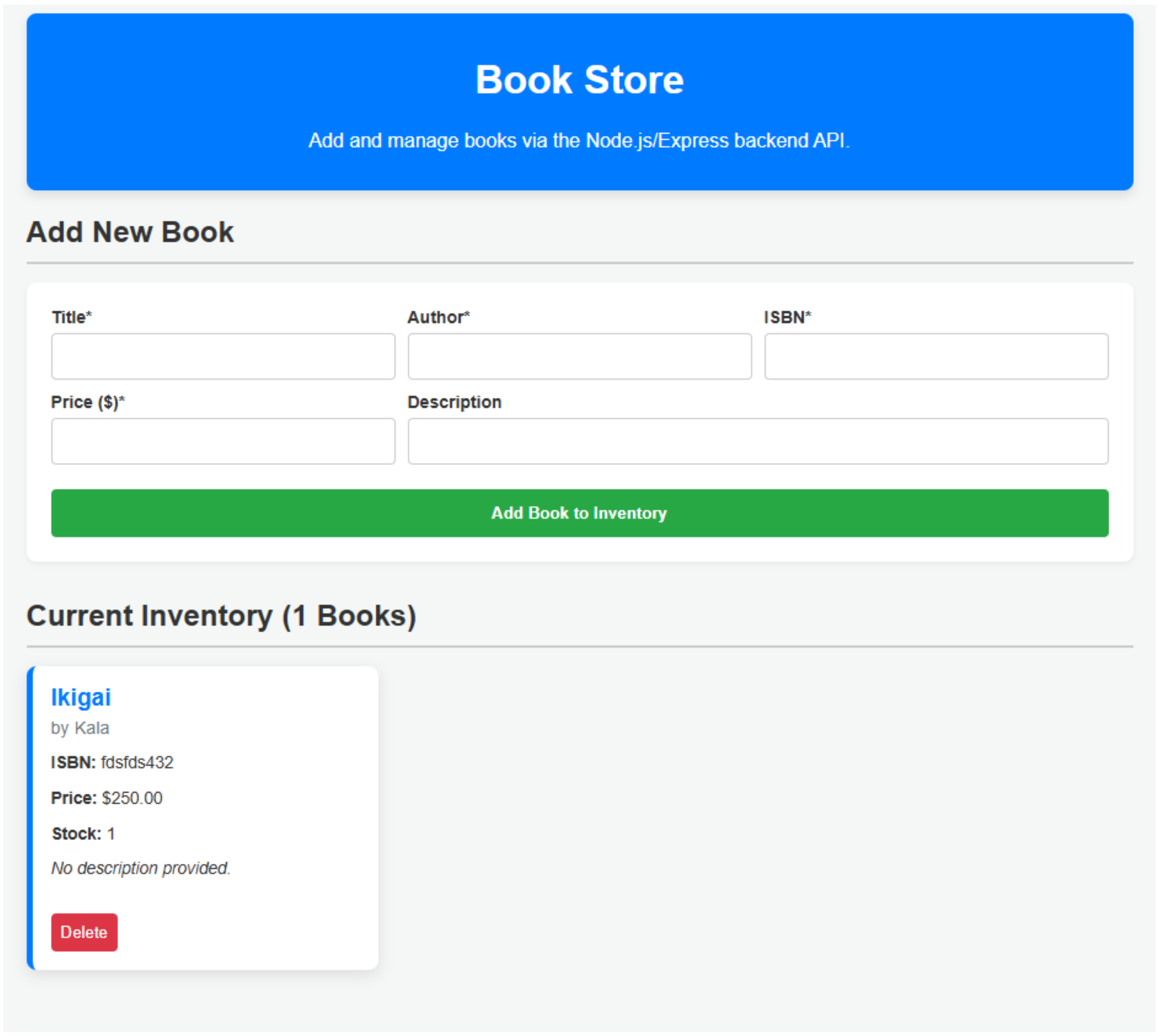
```
bookRouter.get('/:id', async (req, res) => {
    try {
        const book = await Book.findById(req.params.id);

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

        res.status(200).json(book);
    } catch (error) {
        // Handle invalid ID format or server error
        if (error.kind === 'ObjectId') {
             return res.status(400).json({ message: 'Invalid Book ID format'
});
        }
        res.status(500).json({ message: 'Server error while fetching book',
error: error.message });
    }
});

// @route   POST /api/books
// @desc    Create a new book
bookRouter.post('/', async (req, res) => {
    // req.body contains the JSON data for the new book
    try {
        const newBook = new Book(req.body);
        const book = await newBook.save();
        res.status(201).json(book); // 201 Created
    } catch (error) {
        // Handle validation errors (e.g., missing title, duplicate ISBN)
        if (error.name === 'ValidationError') {
            return res.status(400).json({ message: 'Validation failed',
errors: error.errors });
        }
        if (error.code === 11000) { // MongoDB duplicate key error (for unique
ISBN)
             return res.status(400).json({ message: 'ISBN already exists',
fields: error.keyPattern });
        }
        res.status(500).json({ message: 'Server error while creating book',
error: error.message });
    }
});

// @route   PUT /api/books/:id
// @desc    Update a book by ID

bookRouter.put('/:id', async (req, res) => {
    try {
        // Set 'new: true' to return the updated document
        // Set 'runValidators: true' to re-run schema validators on update
        const book = await Book.findByIdAndUpdate(
            req.params.id,
            req.body,
            { new: true, runValidators: true }
        );

        if (!book) {
            return res.status(404).json({ message: 'Book not found for update'
});
        }

        res.status(200).json(book);
    } catch (error) {
        if (error.name === 'ValidationError') {
            return res.status(400).json({ message: 'Validation failed',
errors: error.errors });
        }
        if (error.kind === 'ObjectId') {
             return res.status(400).json({ message: 'Invalid Book ID format'
});
        }
        res.status(500).json({ message: 'Server error while updating book',
error: error.message });
    }
});

// @route   DELETE /api/books/:id
// @desc    Delete a book by ID
bookRouter.delete('/:id', async (req, res) => {
    try {
        const book = await Book.findByIdAndDelete(req.params.id);

        if (!book) {
            return res.status(404).json({ message: 'Book not found for
deletion' });
        }

        res.status(200).json({ message: 'Book successfully deleted',
deletedBook: book });
    } catch (error) {
        if (error.kind === 'ObjectId') {
             return res.status(400).json({ message: 'Invalid Book ID format'
});
        }
        res.status(500).json({ message: 'Server error while deleting book',
error: error.message });
    }
});

// Use the book routes under the /api/books path
app.use('/api/books', bookRouter);


// ------------------------------------
// 5. START SERVER
// ------------------------------------
app.listen(PORT, () => {
    console.log(`Server running in development mode on port ${PORT}`);
});
```

Server/.env

```
MONGO_URI=mongodb://127.0.0.1:27017/BOOKSTORE
```

Client/src/app.js

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

// The base URL for the backend API
const API_URL = 'http://localhost:5000/api/books';

// Helper function for making API calls with exponential backoff
const fetchWithRetry = async (url, options, retries = 3) => {
    for (let i = 0; i < retries; i++) {
        try {
            const response = await fetch(url, options);
            if (!response.ok) {
                // If the response is not OK, throw an error to be caught
below
                const errorData = await response.json();
                throw new Error(errorData.message || `HTTP error! Status:
${response.status}`);
            }
            // For DELETE requests, response.json() might throw an error if
the body is empty.
            if (options?.method === 'DELETE' || response.status === 204) {
                return {};
            }
            return await response.json();
        } catch (error) {
            console.error(`Attempt ${i + 1} failed for ${options?.method ||
'GET'} ${url}:`, error.message);
            if (i === retries - 1) throw error; // Re-throw on last attempt
            // Wait for 2^i seconds before retrying
            await new Promise(resolve => setTimeout(resolve, Math.pow(2, i) *
1000));
        }
    }
};

// --- Styles (Inline CSS Object) ---
const styles = {
    container: {
        maxWidth: '900px',
        margin: '0 auto',
        padding: '20px',
        fontFamily: 'Arial, sans-serif',
        backgroundColor: '#f4f7f6',
        minHeight: '100vh',
    },
    header: {
        backgroundColor: '#007bff',
        color: 'white',
        padding: '15px',
        borderRadius: '8px',
        textAlign: 'center',
        marginBottom: '20px',
        boxShadow: '0 4px 8px rgba(0, 0, 0, 0.1)',
    },
    form: {
        backgroundColor: 'white',
        padding: '20px',
        borderRadius: '8px',
        marginBottom: '30px',
        boxShadow: '0 2px 4px rgba(0, 0, 0, 0.05)',
        display: 'grid',
        gridTemplateColumns: 'repeat(3, 1fr)',
        gap: '10px',
    },
    inputGroup: {
        display: 'flex',
        flexDirection: 'column',
    },
    label: {
        marginBottom: '5px',
        fontWeight: 'bold',
        fontSize: '14px',
        color: '#333',
    },
    input: {
        padding: '10px',
        border: '1px solid #ccc',
        borderRadius: '4px',
        fontSize: '14px',
    },
    submitButton: {
        gridColumn: 'span 3',
        padding: '12px',
        backgroundColor: '#28a745',
        color: 'white',
        border: 'none',
        borderRadius: '4px',
        cursor: 'pointer',
        fontWeight: 'bold',
        marginTop: '10px',
    },
    listTitle: {
        fontSize: '1.5rem',
        color: '#333',
        borderBottom: '2px solid #ccc',
        paddingBottom: '10px',
        marginBottom: '15px',
    },
    bookList: {
        display: 'grid',
        gridTemplateColumns: 'repeat(auto-fill, minmax(280px, 1fr))',
        gap: '20px',
    },
    bookCard: {
        backgroundColor: 'white',
        borderLeft: '5px solid #007bff',
        padding: '15px',
        borderRadius: '8px',
        boxShadow: '0 4px 12px rgba(0, 0, 0, 0.1)',
        display: 'flex',
        flexDirection: 'column',
        justifyContent: 'space-between',
    },
    cardTitle: {
        fontSize: '1.2rem',
        fontWeight: 'bold',
        color: '#007bff',
        marginBottom: '5px',
    },
    cardAuthor: {
        fontSize: '0.9rem',
        color: '#6c757d',
        marginBottom: '10px',
    },
    cardDetails: {
        fontSize: '0.85rem',
        color: '#333',
    },
    deleteButton: {
        marginTop: '15px',
        padding: '8px',
        backgroundColor: '#dc3545',
        color: 'white',
        border: 'none',
        borderRadius: '4px',
        cursor: 'pointer',
        alignSelf: 'flex-start',
    },
    error: {
        backgroundColor: '#f8d7da',
        color: '#721c24',
        padding: '10px',
        border: '1px solid #f5c6cb',
        borderRadius: '4px',
        marginBottom: '20px',
        textAlign: 'center',
    },
    loading: {
        textAlign: 'center',
        padding: '20px',
        fontSize: '1.2rem',
        color: '#007bff',
    },
};

/**
 * Main application component for the Book Store UI.
 * Handles state, API calls, and rendering.
 */
export default function App() {
    const [books, setBooks] = useState([]);
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState(null);
    const [newBook, setNewBook] = useState({
        title: '',
        author: '',
        isbn: '',
        price: '', // Use string for input, convert to number on submission
        description: '',
    });

    // --- API Fetching Functions ---

    // 1. Fetch all books
    const fetchBooks = async () => {
        setLoading(true);
        setError(null);
        try {
            const data = await fetchWithRetry(API_URL);
            setBooks(data);
        } catch (err) {
            setError('Failed to load books. Is the backend server running on
port 5000?');
            console.error(err);
        } finally {
            setLoading(false);
        }
    };

    // 2. Add a new book
    const addBook = async (bookData) => {
        setError(null);
        try {
            const data = await fetchWithRetry(API_URL, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                // Ensure price is converted to a number
                body: JSON.stringify({ ...bookData, price:
Number(bookData.price) }),
            });
            // Update the local state with the new book returned by the server
            setBooks(prevBooks => [...prevBooks, data]);
            // Reset the form after success
            setNewBook({ title: '', author: '', isbn: '', price: '',
description: '' });
        } catch (err) {
            setError(`Failed to add book: ${err.message}`);
            console.error(err);
        }
    };

    // 3. Delete a book
    const deleteBook = async (id) => {
        setError(null);
        try {
            await fetchWithRetry(`${API_URL}/${id}`, {
                method: 'DELETE',
            });
            // Remove the book from the local state
            setBooks(prevBooks => prevBooks.filter(book => book._id !== id));
        } catch (err) {
            setError(`Failed to delete book: ${err.message}`);
            console.error(err);
        }
    };

    // Initial load effect
    useEffect(() => {
        fetchBooks();
    }, []);

    // --- Event Handlers ---

    const handleInputChange = (e) => {
        const { name, value } = e.target;
        setNewBook(prev => ({
            ...prev,
            [name]: value,
        }));
    };

    const handleSubmit = (e) => {
        e.preventDefault();
        // Basic validation check
        if (!newBook.title || !newBook.author || !newBook.isbn ||
!newBook.price) {
            setError("Please fill in all required fields (Title, Author, ISBN,
Price).");
            return;
        }
        addBook(newBook);
    };

    // --- Renderer ---

    const BookCard = ({ book }) => (
        <div style={styles.bookCard}>
            <div>
                <div style={styles.cardTitle}>{book.title}</div>
                <div style={styles.cardAuthor}>by {book.author}</div>
                <div style={styles.cardDetails}>
                    <p><strong>ISBN:</strong> {book.isbn}</p>
                    <p><strong>Price:</strong> ${book.price.toFixed(2)}</p>
                    <p><strong>Stock:</strong> {book.stock}</p>
                    <p><em>{book.description}</em></p>
                </div>
            </div>
            <button
                style={styles.deleteButton}
                onClick={() => deleteBook(book._id)}
            >
                Delete
            </button>
        </div>
    );

    return (
        <div style={styles.container}>
            <header style={styles.header}>
                <h1>Book Store</h1>
                <p>Add and manage books via the Node.js/Express backend
API.</p>
            </header>

            {/* Error Message Display */}
            {error && <div style={styles.error}>{error}</div>}

            {/* Add New Book Form */}
            <h2 style={styles.listTitle}>Add New Book</h2>
            <form onSubmit={handleSubmit} style={styles.form}>
                <div style={styles.inputGroup}>
                    <label style={styles.label} htmlFor="title">Title*</label>
                    <input
                        style={styles.input}
                        type="text"
                        id="title"
                        name="title"
                        value={newBook.title}
                        onChange={handleInputChange}
                        required
                    />
                </div>
                <div style={styles.inputGroup}>
                    <label style={styles.label}
htmlFor="author">Author*</label>
                    <input
                        style={styles.input}
                        type="text"
                        id="author"
                        name="author"
                        value={newBook.author}
                        onChange={handleInputChange}
                        required
                    />
                </div>
                <div style={styles.inputGroup}>
                    <label style={styles.label} htmlFor="isbn">ISBN*</label>
                    <input
                        style={styles.input}
                        type="text"
                        id="isbn"
                        name="isbn"
                        value={newBook.isbn}
                        onChange={handleInputChange}
                        required
                    />
                </div>
                <div style={styles.inputGroup}>
                    <label style={styles.label} htmlFor="price">Price
($)*</label>
                    <input
                        style={styles.input}
                        type="number"
                        id="price"
                        name="price"
                        value={newBook.price}
                        onChange={handleInputChange}
                        min="0"
                        step="0.01"
                        required
                    />
                </div>
                <div style={{ ...styles.inputGroup, gridColumn: 'span 2' }}>
                    <label style={styles.label}
htmlFor="description">Description</label>
                    <input
                        style={styles.input}
                        type="text"
                        id="description"
                        name="description"
                        value={newBook.description}
                        onChange={handleInputChange}
                    />
                </div>
                <button type="submit" style={styles.submitButton}>
                    Add Book to Inventory
                </button>
            </form>

            {/* Book List Display */}
            <h2 style={styles.listTitle}>Current Inventory ({books.length}
Books)</h2>

            {loading && <div style={styles.loading}>Loading books...</div>}

            {!loading && books.length === 0 && !error && (
                <p style={{ textAlign: 'center', color: '#6c757d' }}>No books
found in the inventory. Add one above!</p>
            )}

            <div style={styles.bookList}>
                {books.map(book => (
                    <BookCard key={book._id} book={book} />
                ))}
            </div>
        </div>
    );
}
```
OUTPUT:


Book Store

Add and manage books via the Node.js/Express backend API.

Add New Book

Title*	Author*	ISBN*
Price ($)*	Description	

Add Book to Inventory

Current Inventory (1 Books)

Ikigai

by Kala

ISBN: fdsfds432

Price: $250.00

Stock: 1

No description provided.

Delete