Scantech Library Web-App (MERN)

Full-stack lesson pack: Node.js + Express + MongoDB + React

Goal: Students build a simple library management app with Students, Books, Loans (borrow/return), and a small Stats service.

Tech: Node.js (API on port 5000), MongoDB (Atlas or local), React (UI on port 3000).

Learning outcomes

After this project, students can:

• Create a clean backend structure: config, models, controllers, services, routes, middleware.

• Build REST endpoints for CRUD and connect them to MongoDB using Mongoose.

• Use a simple Service layer for reusable business logic (library statistics).

• Create a React frontend with pages/components and a reusable API client.

• Connect frontend to backend (CORS/proxy) and test with Postman/Thunder Client.

Part A — Project setup

Recommended folder structure

```
scantech-library/
  backend/
  frontend/
```

Prerequisites

• Node.js 18+ installed

• MongoDB Atlas account (recommended) OR local MongoDB

• VS Code + Thunder Client (or Postman)

Quick run commands

```
# Backend (port 5000)
cd backend
npm install
npm run dev

# Frontend (port 3000)
cd ../frontend
npm install
npm start
```

Part B — Backend (Node.js + Express + MongoDB)

1) Backend structure (clean & teachable)

```
backend/
  src/
    config/
      db.js
    models/
      Student.js
      Book.js
      Loan.js
    controllers/
      studentController.js
      bookController.js
      loanController.js
      statsController.js
    services/
      libraryStatsService.js
    routes/
      studentRoutes.js
      bookRoutes.js
      loanRoutes.js
      statsRoutes.js
    middleware/
      errorHandler.js
  server.js
  .env
```

Rule of thumb for students:

• **Routes** decide the URL paths. • **Controllers** handle HTTP requests. • **Models** talk to MongoDB. • **Services** hold reusable business logic.

2) Install backend dependencies

```
cd backend
npm init -y
npm install express mongoose cors dotenv
npm install -D nodemon
```

package.json scripts

```
{
  "scripts": {
    "dev": "nodemon server.js",
    "start": "node server.js"
  }
}
```

3) Environment variables (.env)

```
PORT=5000
MONGO_URL=mongodb+srv://<user>:<pass>@cluster0.xxxxx.mongodb.net/scantech_library?retryWrites=true&w=ma
```

Tip: never commit .env. Add it to .gitignore.

4) MongoDB connection (src/config/db.js)

```
// src/config/db.js
const mongoose = require("mongoose");

async function connectDB() {
  try {
    await mongoose.connect(process.env.MONGO_URL);
    console.log("MongoDB connected");
  } catch (err) {
```

```
    console.error("MongoDB connection error:", err.message);
    process.exit(1);
  }
}

module.exports = connectDB;
```

5) Models (Mongoose schemas)

Student model (src/models/Student.js)

```
// src/models/Student.js
const mongoose = require("mongoose");

/*
  Student = a library member.
  Keep it simple for beginners: name + email + className.
*/
const StudentSchema = new mongoose.Schema(
  {
    fullName: { type: String, required: true, trim: true },
    email: { type: String, required: true, lowercase: true, unique: true },
    className: { type: String, default: "General" }
  },
  { timestamps: true }
);

module.exports = mongoose.model("Student", StudentSchema);
```

Book model (src/models/Book.js)

```
// src/models/Book.js
const mongoose = require("mongoose");

/*
  Book = an item in the library.
  availableCopies helps prevent borrowing when 0.
*/
const BookSchema = new mongoose.Schema(
  {
    title: { type: String, required: true, trim: true },
    author: { type: String, required: true, trim: true },
    isbn: { type: String, trim: true },
    availableCopies: { type: Number, default: 1, min: 0 }
  },
  { timestamps: true }
);

module.exports = mongoose.model("Book", BookSchema);
```

Loan model (borrow/return) (src/models/Loan.js)

```
// src/models/Loan.js
const mongoose = require("mongoose");

/*
  Loan connects a Student to a Book.
  returnedAt === null means 'currently borrowed'.
*/
const LoanSchema = new mongoose.Schema(
  {
    studentId: { type: mongoose.Schema.Types.ObjectId, ref: "Student", required: true },
    bookId: { type: mongoose.Schema.Types.ObjectId, ref: "Book", required: true },
    borrowedAt: { type: Date, default: Date.now },
    returnedAt: { type: Date, default: null }
  },
  { timestamps: true }
);

module.exports = mongoose.model("Loan", LoanSchema);
```

6) server.js + middleware order

```
// server.js
require("dotenv").config();
const express = require("express");
const cors = require("cors");

const connectDB = require("./src/config/db");
const studentRoutes = require("./src/routes/studentRoutes");
const bookRoutes = require("./src/routes/bookRoutes");
const loanRoutes = require("./src/routes/loanRoutes");
const statsRoutes = require("./src/routes/statsRoutes");
const { notFound, errorHandler } = require("./src/middleware/errorHandler");

const app = express();

// 1) Global middleware
app.use(cors());
app.use(express.json()); // parse JSON bodies

// 2) Routes
app.use("/api/students", studentRoutes);
app.use("/api/books", bookRoutes);
app.use("/api/loans", loanRoutes);
app.use("/api/stats", statsRoutes);

// 3) Error middleware (must be last)
app.use(notFound);
app.use(errorHandler);

const PORT = process.env.PORT || 5000;

connectDB().then(() => {
  app.listen(PORT, () => console.log("API running on port " + PORT));
});
```

Error handler (src/middleware/errorHandler.js)

```
// src/middleware/errorHandler.js
function notFound(req, res, next) {
  res.status(404);
  next(new Error("Route not found: " + req.originalUrl));
}

function errorHandler(err, req, res, next) {
  const status = res.statusCode && res.statusCode !== 200 ? res.statusCode : 500;
  res.status(status).json({
    message: err.message,
    stack: process.env.NODE_ENV === "production" ? "hidden" : err.stack
  });
}

module.exports = { notFound, errorHandler };
```

7) Endpoints (REST API)

Resource	Method	Endpoint	What it does
Students	GET	/api/students	List students
Students	POST	/api/students	Create student
Students	PUT	/api/students/:id	Update student
Students	DELETE	/api/students/:id	Delete student
Books	GET	/api/books	List books (optional search)
Books	POST	/api/books	Create book
Books	PUT	/api/books/:id	Update book

Books	DELETE	/api/books/:id	Delete book
Loans	POST	/api/loans/borrow	Borrow a book
Loans	POST	/api/loans/return	Return a book
Stats	GET	/api/stats/summary	Counts for dashboard

8) Controllers + Routes (example: Students)

Controller: src/controllers/studentController.js

```
// src/controllers/studentController.js
const Student = require("../models/Student");

// GET /api/students
async function listStudents(req, res) {
  const students = await Student.find().sort({ createdAt: -1 });
  res.json(students);
}

// POST /api/students
async function createStudent(req, res) {
  const { fullName, email, className } = req.body;

  // Simple beginner validation
  if (!fullName || !email) {
    res.status(400);
    throw new Error("fullName and email are required");
  }

  const student = await Student.create({ fullName, email, className });
  res.status(201).json(student);
}

// PUT /api/students/:id
async function updateStudent(req, res) {
  const student = await Student.findByIdAndUpdate(
    req.params.id,
    req.body,
    { new: true }
  );
  if (!student) {
    res.status(404);
    throw new Error("Student not found");
  }
  res.json(student);
}

// DELETE /api/students/:id
async function deleteStudent(req, res) {
  const removed = await Student.findByIdAndDelete(req.params.id);
  if (!removed) {
    res.status(404);
    throw new Error("Student not found");
  }
  res.json({ message: "Student deleted" });
}

module.exports = { listStudents, createStudent, updateStudent, deleteStudent };
```

Routes: src/routes/studentRoutes.js

```
// src/routes/studentRoutes.js
const router = require("express").Router();
const c = require("../controllers/studentController");

router.get("/", c.listStudents);
router.post("/", c.createStudent);
router.put("/:id", c.updateStudent);
```

```
router.delete("/:id", c.deleteStudent);

module.exports = router;
```

9) Service layer (libraryStatsService)

Service = reusable logic. Here we calculate dashboard numbers.

```
// src/services/libraryStatsService.js
const Student = require("../models/Student");
const Book = require("../models/Book");
const Loan = require("../models/Loan");

/*
  Simple "table info" service:
  - totalStudents
  - totalBooks
  - borrowedNow (loans with returnedAt = null)
*/
async function getSummary() {
  const [totalStudents, totalBooks, borrowedNow] = await Promise.all([
    Student.countDocuments(),
    Book.countDocuments(),
    Loan.countDocuments({ returnedAt: null })
  ]);

  return { totalStudents, totalBooks, borrowedNow };
}

module.exports = { getSummary };
```

Controller + route for stats

```
// src/controllers/statsController.js
const { getSummary } = require("../services/libraryStatsService");

async function summary(req, res) {
  const data = await getSummary();
  res.json(data);
}

module.exports = { summary };

// src/routes/statsRoutes.js
const router = require("express").Router();
const c = require("../controllers/statsController");
router.get("/summary", c.summary);
module.exports = router;
```

10) Loans (borrow/return) — key idea

Borrow: create a Loan + decrease Book.availableCopies. Return: set returnedAt + increase copies.

```
// src/controllers/loanController.js (core idea)
const Book = require("../models/Book");
const Loan = require("../models/Loan");

async function borrow(req, res) {
  const { studentId, bookId } = req.body;

  const book = await Book.findById(bookId);
  if (!book) throw new Error("Book not found");
  if (book.availableCopies <= 0) {
    res.status(400);
    throw new Error("No copies available");
  }

  const loan = await Loan.create({ studentId, bookId, returnedAt: null });

  book.availableCopies -= 1;
  await book.save();

  res.status(201).json(loan);
}
```

```
async function returnBook(req, res) {
  const { loanId } = req.body;

  const loan = await Loan.findById(loanId);
  if (!loan) throw new Error("Loan not found");
  if (loan.returnedAt) {
    res.status(400);
    throw new Error("Already returned");
  }

  loan.returnedAt = new Date();
  await loan.save();

  const book = await Book.findById(loan.bookId);
  if (book) {
    book.availableCopies += 1;
    await book.save();
  }

  res.json({ message: "Returned successfully" });
}

module.exports = { borrow, returnBook };
```

Part C — Frontend (React)

1) Create the React app (CRA)

```
cd frontend
npx create-react-app .
npm install axios react-router-dom
```

2) Configure API base URL

Create a reusable Axios client so students stop copy/pasting URLs everywhere.

```
// src/api/client.js
import axios from "axios";

export const api = axios.create({
  baseURL: process.env.REACT_APP_API_URL || "http://localhost:5000/api"
});
```

For local dev, create frontend/.env

```
REACT_APP_API_URL=http://localhost:5000/api
```

3) Pages/components (minimal set)

```
src/
  api/
    client.js
  components/
    Navbar.jsx
  pages/
    Dashboard.jsx
    Students.jsx
    Books.jsx
    Loans.jsx
  App.js
```

Navbar (src/components/Navbar.jsx)

```
import { Link } from "react-router-dom";

export default function Navbar() {
  return (
    <nav style={{ padding: 12, borderBottom: "1px solid #ddd" }}>
      <b>Scantech Library</b>  | 
      <Link to="/">Dashboard</Link>  | 
      <Link to="/students">Students</Link>  | 
      <Link to="/books">Books</Link>  | 
      <Link to="/loans">Loans</Link>
    </nav>
  );
}
```

App routing (src/App.js)

```
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Navbar from "./components/Navbar";
import Dashboard from "./pages/Dashboard";
import Students from "./pages/Students";
import Books from "./pages/Books";
import Loans from "./pages/Loans";

export default function App() {
  return (
    <BrowserRouter>
      <Navbar />
      <div style={{ padding: 16 }}>
        <Routes>
```

```
          <Route path="/" element={<Dashboard />} />
          <Route path="/students" element={<Students />} />
          <Route path="/books" element={<Books />} />
          <Route path="/loans" element={<Loans />} />
        </Routes>
      </div>
    </BrowserRouter>
  );
}
```

Dashboard page (calls the Stats service)

```
// src/pages/Dashboard.jsx
import { useEffect, useState } from "react";
import { api } from "../api/client";

export default function Dashboard() {
  const [stats, setStats] = useState(null);

  useEffect(() => {
    api.get("/stats/summary").then(res => setStats(res.data));
  }, []);

  if (!stats) return <p>Loading...</p>;

  return (
    <div>
      <h2>Dashboard</h2>
      <ul>
        <li>Total students: {stats.totalStudents}</li>
        <li>Total books: {stats.totalBooks}</li>
        <li>Borrowed now: {stats.borrowedNow}</li>
      </ul>
    </div>
  );
}
```

Students page (simple CRUD)

```
// src/pages/Students.jsx
import { useEffect, useState } from "react";
import { api } from "../api/client";

export default function Students() {
  const [items, setItems] = useState([]);
  const [fullName, setFullName] = useState("");
  const [email, setEmail] = useState("");

  async function load() {
    const res = await api.get("/students");
    setItems(res.data);
  }

  useEffect(() => { load(); }, []);

  async function addStudent(e) {
    e.preventDefault();
    await api.post("/students", { fullName, email });
    setFullName(""); setEmail("");
    load();
  }

  async function removeStudent(id) {
    await api.delete("/students/" + id);
    load();
  }

  return (
    <div>
      <h2>Students</h2>

      <form onSubmit={addStudent} style={{ marginBottom: 12 }}>
        <input value={fullName} onChange={e => setFullName(e.target.value)} placeholder="Full name" />
        {" "}
        <input value={email} onChange={e => setEmail(e.target.value)} placeholder="Email" />
        {" "}
        <button type="submit">Add</button>
      </form>

      <ul>
        {items.map(s => (
          <li key={s._id}>
            {s.fullName} ({s.email})
            {" "}
            <button onClick={() => removeStudent(s._id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Books and Loans pages follow the same pattern. Start simple, then add features (search books, borrow/return actions).

Teacher checklist (how to teach this project)

• Start with data: Students, Books, Loans (show diagrams).

• Build backend first: models -> controllers -> routes -> test in Postman.

• Only then start frontend: pages call endpoints (GET/POST first).

• Add the Service layer last (Stats dashboard) to show good practice.

• Finish with deployment discussion (Render for backend, Netlify for frontend).

Student tasks (replication)

• Implement Books page CRUD (create/list/delete).

• Add a search box for books (query param ?q=).

• Implement borrow/return UI buttons on Loans page.

• Add simple validation messages in frontend (empty inputs).

• Stretch: add dueDate and show overdue loans in Stats.

Deployment notes (short)

Backend: use process.env.PORT and set MONGO_URL on Render. Frontend: set REACT_APP_API_URL to your deployed API URL on Netlify.