Digital Library Portal — Technical Documentation

A web-based library management system with role-based access (Librarian / Student) for catalog management, book lending, returns, and fine tracking.

Technology Stack

Backend (Port 5000, CommonJS): Node.js, Express ^4.21.2, SQLite, Prisma 5.22.0, jsonwebtoken ^9.0.2, bcryptjs ^2.4.3, cookie-parser ^1.4.7, cors ^2.8.5, dotenv ^16.4.7, nodemon ^3.1.14

Frontend (Port 5173, ES Modules): React ^19.2.0, Vite ^7.3.1, Tailwind CSS ^4.2.0, Zustand ^5.0.11, React Router DOM ^7.13.0, Axios ^1.13.5, React Hot Toast ^2.6.0, React Icons ^5.5.0

How It Works

- Users register with name, email, password, and role → password is **bcrypt-hashed** → stored in SQLite via Prisma.
- On login, the server issues a **JWT** (7-day expiry) as an **HTTP-only cookie** + JSON response.
- On every page load, the client calls **GET /api/auth/me** to restore the session silently.
- The **auth middleware** verifies the JWT, extracts the user payload, and attaches it to `req.user`.
- **Librarian** sees → Dashboard, Books (CRUD), All Issues, Manage Students.
- **Student** sees → Books (browse & borrow), My Books, History.
- **ProtectedRoute** redirects unauthenticated users to login.

REST API Reference

Authentication — `/api/auth`

Method	Route	Protected	Role	Description
POST	/api/auth/register	No	—	Create account
POST	/api/auth/login	No	—	Login, get JWT
POST	/api/auth/logout	No	—	Clear cookie
GET	/api/auth/me	Yes	—	Get current user

Books — `/api/books`

Method	Route	Protected	Role	Description
GET	/api/books	Yes	—	List/search books
GET	/api/books/:id	Yes	—	Get single book
POST	/api/books	Yes	Librarian	Add book

Method	Route	Protected	Role	Description
PUT	/api/books/:id	Yes	Librarian	Update book
DELETE	/api/books/:id	Yes	Librarian	Delete book
GET	/api/books/meta/categories	Yes	—	Get categories

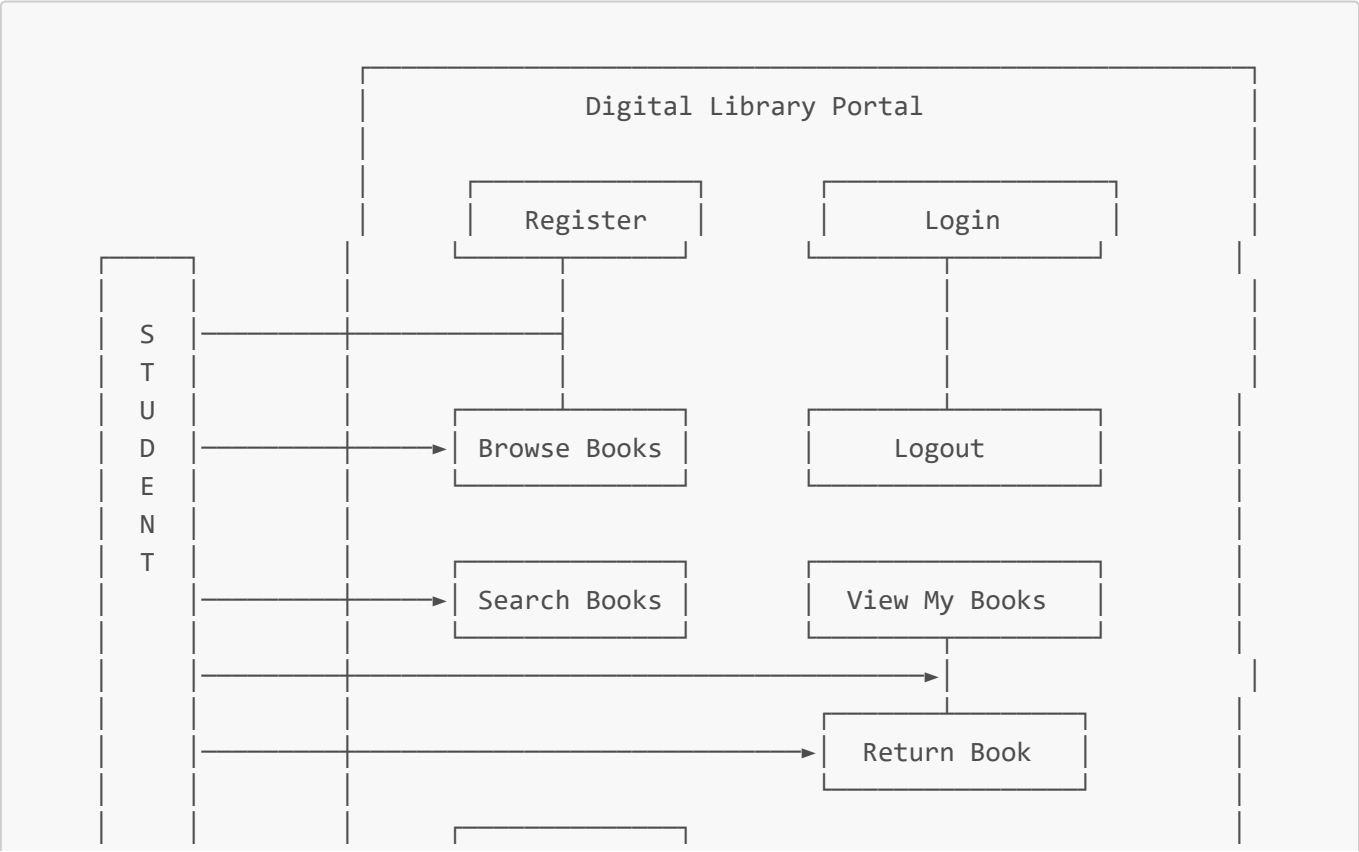
Issues & Returns — `/api/issues`

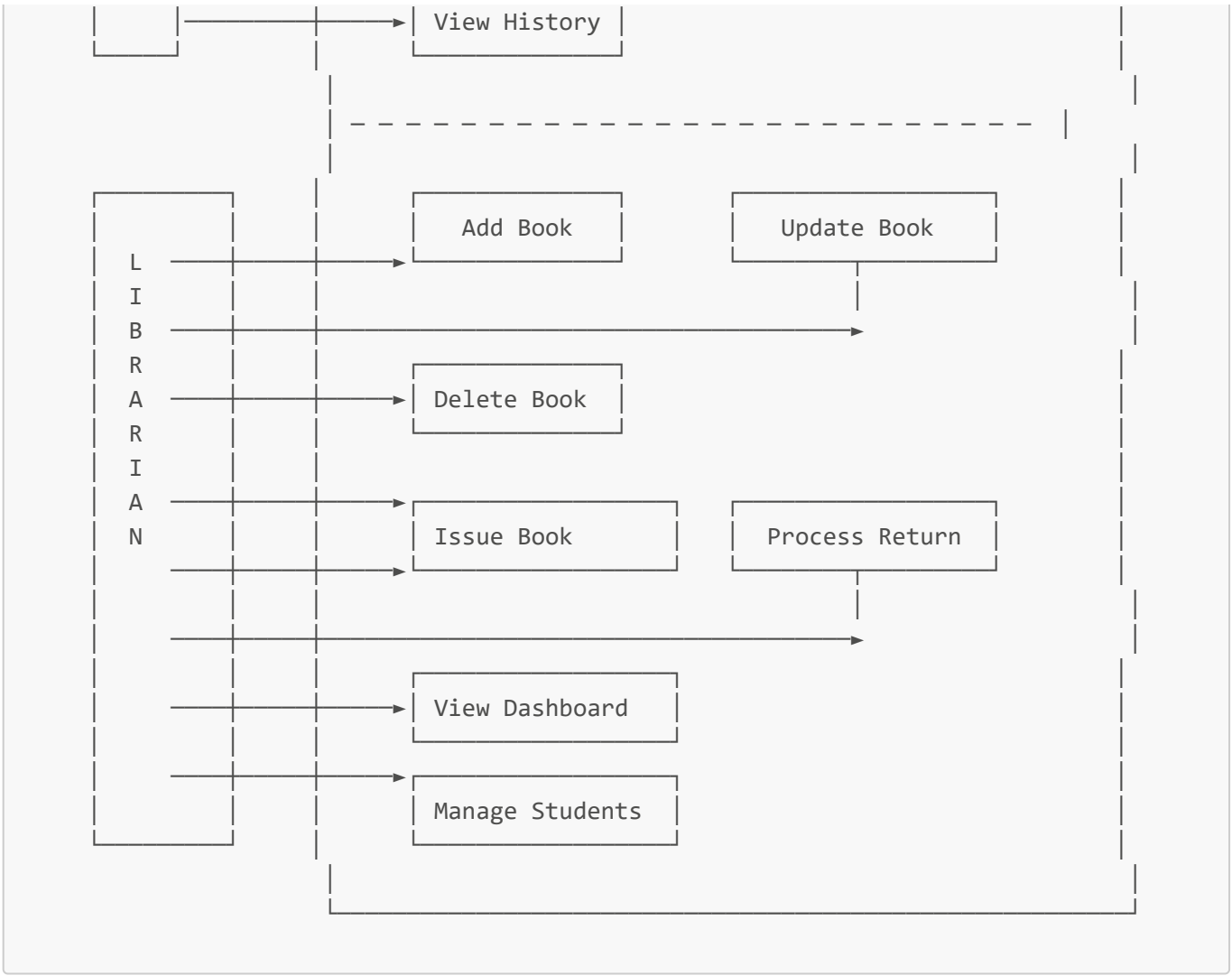
Method	Route	Protected	Role	Description
POST	/api/issues/issue	Yes	—	Issue book (14-day period)
POST	/api/issues/return/:issueId	Yes	—	Return book ($2/day fine if late)
GET	/api/issues	Yes	—	List issues
GET	/api/issues/history/:userId	Yes	—	Borrowing history
GET	/api/issues/stats/dashboard	Yes	Librarian	Dashboard stats

Users — `/api/users`

Method	Route	Protected	Role	Description
GET	/api/users	Yes	Librarian	List students
DELETE	/api/users/:id	Yes	Librarian	Delete student

Use Case Diagram





Functional Requirements

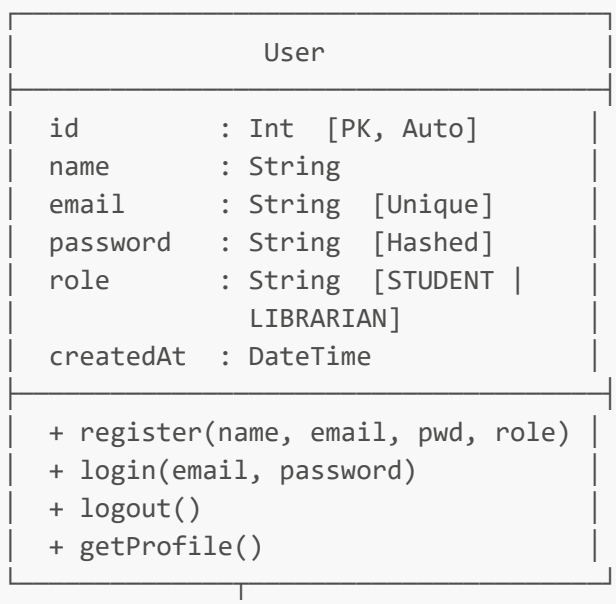
#	Feature	Details
1	Account Creation	Sign up with name, email, password, role (Student/Librarian)
2	Login	Verify credentials, return JWT
3	Logout	Clear auth cookie
4	Persistent Sessions	Auto-validate JWT on page load via /api/auth/me
5	Role-Based Permissions	Different pages and API access per role
6	Catalog Management	Librarians add, edit, delete books
7	Catalog Viewing	All users browse the book list
8	Search & Filter	Search by title/author/ISBN, filter by category
9	Book Lending	Issue with 14-day deadline using database transaction
10	Duplicate Loan Guard	Block issuing same book twice to one user
11	Return & Fine	$2/day late fee auto-calculated on return

#	Feature	Details
12	Copy Tracking	availableCopies decremented/incremented on issue/return
13	Borrowing History	Chronological log of all loans per user
14	Dashboard	Stats: total books, users, issued, returned, fines
15	Student Management	View/delete student accounts (blocked if active loans)
16	Notifications	Toast pop-ups for success/error feedback

Non-Functional Requirements

#	Area	Details
1	Security	Passwords hashed with bcrypt (10 salt rounds)
2	Security	JWT stored in HTTP-only cookies (XSS protection)
3	Security	All protected routes require valid JWT
4	Performance	Prisma generates optimized parameterized SQL
5	Data Integrity	Issue/return wrapped in $transaction blocks
6	Usability	Responsive UI via Tailwind CSS
7	Usability	Real-time toast notifications
8	Reliability	SQLite file-based DB, zero external setup
9	Maintainability	Layered architecture: routes, middleware, stores, pages
10	Portability	Config via .env file (DB path, JWT secret, port)

Data Model (Class Diagram)



Relationship	Cardinality	Description
User → Issue	1 to Many	One student can have many borrowing records
Book → Issue	1 to Many	One book can appear in many issue records
User ↔ Book	Many-to-Many	Connected through the Issue table

Summary

A full-stack library system using React + Zustand on the frontend and Express + Prisma + SQLite on the backend. Authentication uses JWT in HTTP-only cookies with bcrypt password hashing. Role-based routing gives librarians and students tailored dashboards. Prisma transactions keep book availability in sync. SQLite requires zero setup. The result is a clean, secure, and maintainable application.