📚 e-Granthalaya — Complete Project Documentation Report

Project Name: e-Granthalaya — Digital Library Management System
Institution: School of Mines KGF
Technology Stack: Node.js, Express.js, SQLite, Vanilla JavaScript, HTML5, CSS3
License: MIT
Report Generated: 23 February 2026

📑 Table of Contents

---

1. Project Overview

e-Granthalaya is a full-stack, MVC-based Digital Library Management System built for the School of Mines KGF. It allows:

- **Admins** to manage books (add, delete, restore), view all borrowing records, and monitor student activity.
- **Students** to register, log in, browse the book catalog, borrow and return books, read PDFs online, and view their borrowing history.

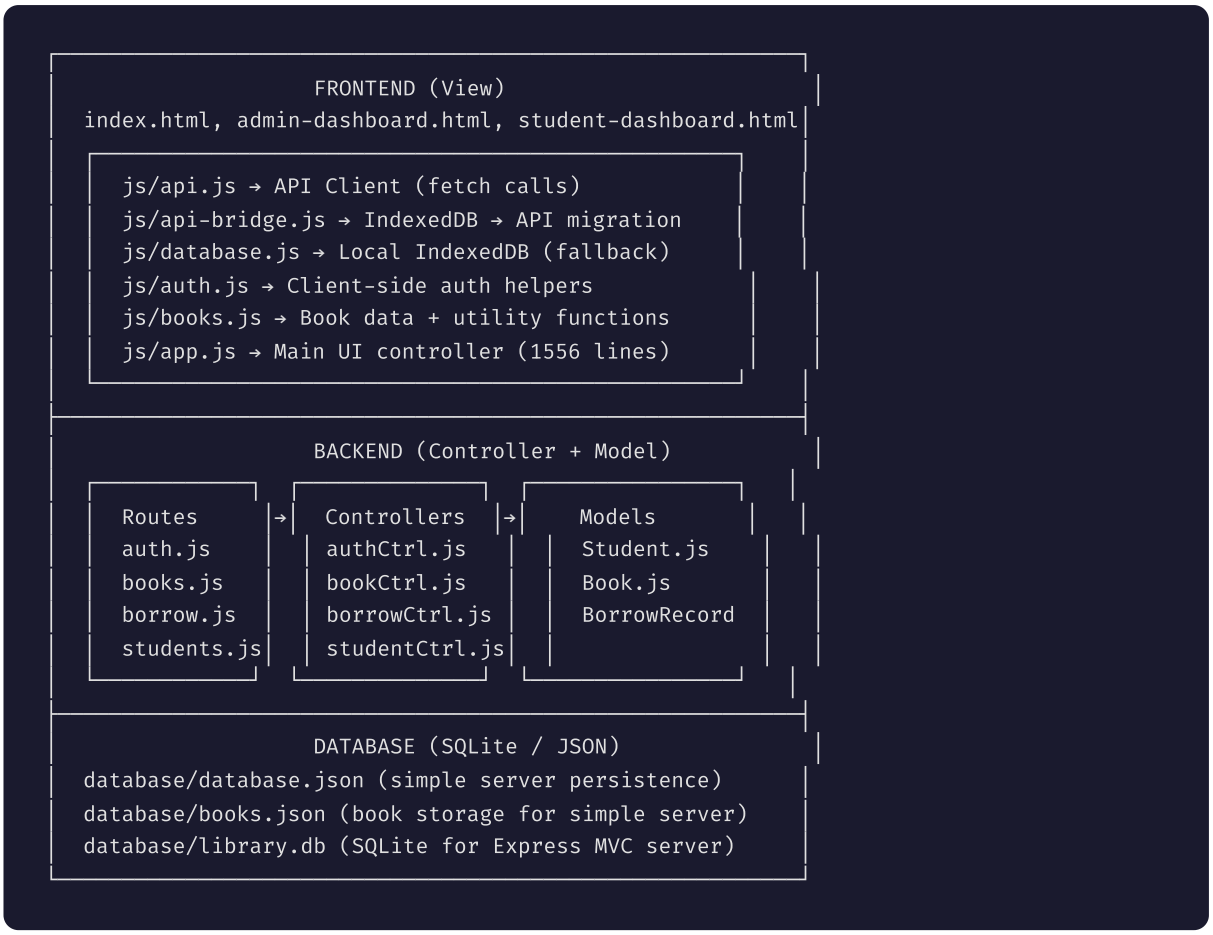
Key Features:

- ✅ Admin Panel with full book management (CRUD + soft-delete + restore)
- ✅ Student Portal with book browsing, borrowing, and inline PDF reading
- ✅ Real-time book synchronization via API polling every 5 seconds
- ✅ Department-wise organization (Computer Science, Mechanical, Mining)
- ✅ 15-day borrowing period with overdue tracking
- ✅ Cross-tab synchronization via `localStorage` events
- ✅ Dual server architecture (Express MVC + Zero-dependency simple server)
- ✅ Deployable to Vercel (serverless) and Render (full server)

---

2. Architecture & Design Pattern

The project uses a **Model-View-Controller (MVC)** pattern with a clear separation of concerns:


FRONTEND (View)
index.html, admin-dashboard.html, student-dashboard.html
js/api.js → API Client (fetch calls)
js/api-bridge.js → IndexedDB → API migration
js/database.js → Local IndexedDB (fallback)
js/auth.js → Client-side auth helpers
js/books.js → Book data + utility functions
js/app.js → Main UI controller (1556 lines)
BACKEND (Controller + Model)
Routes
auth.js
books.js
borrow.js
students.js
Controllers
authCtrl.js
bookCtrl.js
borrowCtrl.js
studentCtrl.js
Models
Student.js
Book.js
BorrowRecord
DATABASE (SQLite / JSON)
database/database.json (simple server persistence)
database/books.json (book storage for simple server)
database/library.db (SQLite for Express MVC server)

3. Complete Directory Structure

```
e-granthalaya-main/
├── .gitignore                    # Git ignore rules
├── README.md                     # Project readme
├── package.json                  # Root NPM config
├── package-lock.json             # Dependency lock file
├── vercel.json                   # Vercel deployment config
├── index.html                    # Landing page
├── admin-login.html              # Admin login page
├── admin-dashboard.html          # Admin dashboard (365 lines)
├── student-login.html            # Student login/register page (288 lines)
├── student-dashboard.html        # Student dashboard (270 lines)
│
├── assets/
│   └── logo.png                  # School of Mines logo
│
├── css/
│   └── styles.css                # Global stylesheet (2066 lines)
│
├── js/
│   ├── api.js                    # API client module (190 lines)
│   ├── api-bridge.js             # IndexedDB-to-API bridge (123 lines)
│   ├── database.js               # IndexedDB module (157 lines)
│   ├── auth.js                   # Authentication module (146 lines)
│   ├── books.js                  # Books data & logic (362 lines)
│   └── app.js                    # Main application controller (1556 lines)
│
├── backend/
│   ├── package.json              # Backend dependencies
│   ├── server.js                 # Express MVC server (177 lines)
│   ├── server-simple.js          # Zero-dependency server (418 lines)
│   ├── config/
│   │   └── database.js           # SQLite init & schema (102 lines)
│   ├── models/
│   │   ├── Book.js               # Book model (99 lines)
│   │   ├── BorrowRecord.js       # Borrow model (79 lines)
│   │   └── Student.js            # Student model (77 lines)
│   ├── controllers/
│   │   ├── authController.js     # Auth controller (115 lines)
│   │   ├── bookController.js     # Book controller (148 lines)
│   │   ├── borrowController.js   # Borrow controller (126 lines)
│   │   └── studentController.js  # Student controller (46 lines)
│   ├── routes/
│   │   ├── auth.js               # Auth routes (22 lines)
│   │   ├── books.js              # Book routes (22 lines)
│   │   ├── borrow.js             # Borrow routes (19 lines)
│   │   └── students.js           # Student routes (15 lines)
│   ├── middleware/
│   │   ├── auth.js               # Auth middleware (26 lines)
│   │   └── upload.js             # Multer file upload (47 lines)
│   └── uploads/                  # Uploaded PDF storage
│
├── database/
```

```
│   ├── database.json            # JSON database (students, history)
│   └── books.json               # JSON books database
│
├── api/
│   └── [...path].js             # Vercel serverless API (171 lines)
│
└── e-granthalaya-main/          # Duplicate frontend (GitHub Pages)
```

4. Root-Level Configuration Files

4.1 `package.json` (Root)

Purpose: Defines the project metadata, scripts, and dependencies for the root-level application.

- **`main`**: Points to `backend/server-simple.js` — the zero-dependency server is the default entry point.
- **`scripts.start`**: Runs `node backend/server-simple.js`.
- **`scripts.deploy`**: Deploys frontend files to GitHub Pages using `gh-pages`.
- **Dependencies:** `express`, `cors`, `body-parser` (for the Express server variant).
- **DevDependencies:** `gh-pages` for deployment.

4.2 `vercel.json`

Purpose: Configures Vercel deployment with two build targets:

1. `backend/server.js` → Built as a `@vercel/node` serverless function.
2. `**/*` → All other files served as static assets via `@vercel/static`.

Routes:

- `/api/(.*)` → Proxied to `backend/server.js` for API handling.
- `/(.*)` → Served as static files.

4.3 `.gitignore`

Purpose: Excludes `node_modules/`, `.env`, and `.DS_Store` from version control.

4.4 `README.md`

Purpose: Project documentation with live demo links, quick-start instructions, default credentials (`Admin: 112233`), feature list, and technology stack description.

5. Frontend — HTML Pages

5.1 `index.html` — Landing Page (44 lines)

Purpose: Entry point of the application. Displays the School of Mines branding and two navigation buttons.

Flow:

1. Renders a centered card with the school logo, title "School of Mines", subtitle "e-Granthalaya", and tagline.
2. Presents two buttons: "Admin Login" → `admin-login.html` and "Student Login" → `student-login.html`.
3. Background has three animated floating circles for visual effect.
4. Uses Poppins font from Google Fonts and links to `css/styles.css`.

5.2 `admin-login.html` — Admin Login (74 lines)

Purpose: Single-field password login for administrators.

Flow:

1. Displays a login form with a password field and a toggle-visibility button.
2. On form submit, calls `API.adminLogin(password)` which sends a POST to `/api/auth/admin/login`.
3. If successful, stores session data in `localStorage` (`sessionId`, `userType: 'admin'`) via `api.js` and redirects to `admin-dashboard.html`.
4. If failed, shows an error message below the field.
5. Includes `js/api.js` as its only script dependency.

5.3 `admin-dashboard.html` — Admin Dashboard (365 lines)

Purpose: Full admin panel with sidebar navigation and five tabbed sections.

Tabs:

1. **All Books** — Grid display of all library books with department filter and search bar. Each card shows book cover, title, author, availability status, borrower info, and a delete button.
2. **Borrowing Records** — Table with columns for Student ID, Name, Department, Book, Borrow Date, Due Date, Return Date, and Status (Active/Overdue/Returned).

3. **Students** — Table listing all registered students with their login time, books borrowed, returned, and currently holding. Below it, detailed per-student cards with full borrowing history.
4. **Add Books** — Drag-and-drop file upload area (PDF/EPUB), metadata form (title, author, department, cover URL), and recently-added books list.
5. **Deleted Books** — Grid of soft-deleted books with restore and permanent-delete options.

Authentication Flow:

1. On `DOMContentLoaded`, checks `API.isAdmin()` (reads `localStorage`).
2. If not admin, redirects to `admin-login.html`.
3. If admin, calls `initAdminDashboard()` which initializes IndexedDB, loads all books, records, and students.
4. Tab switching uses event delegation on `.nav-item` elements; each tab switch reloads its data.

Scripts loaded: `api.js`, `api-bridge.js`, `database.js`, `auth.js`, `books.js`, `app.js`.

5.4 `student-login.html` — Student Login/Register (288 lines)

Purpose: Dual-form page with tabbed Login and Register sections.

Login Flow:

1. Student enters Student ID, Full Name, and Password.
2. On submit, calls `API.studentLogin(studentId, password)`.
3. On success, stores session in `localStorage` and redirects to `student-dashboard.html`.
4. On failure, displays error message.

Registration Flow:

1. Student fills Student ID, Full Name, Email, Password, and selects Department.
2. On submit, calls `API.studentRegister({ ... })` which POSTs to `/api/auth/student/register`.
3. On success, shows a success toast and auto-switches to login tab after 2 seconds, pre-filling the Student ID.
4. On failure, shows error message.

Tab Switching Logic: `switchTab(tab)` function toggles `.active` class on tab buttons and forms, and clears all error/success messages.

5.5 `student-dashboard.html` — Student Dashboard (270 lines)

Purpose: Student portal with four tabbed sections and three modals.

Tabs:

1. **All Books** — Same book grid as admin but with Borrow/Return buttons instead of Delete. Includes department filter and search. Shows "Not Available" for books borrowed by others.
2. **My Borrowed Books** — Grid of currently borrowed books with due date status, "Read Book" and "Return Book" buttons.
3. **My History** — Timeline view of all borrowing activity showing borrow dates, due dates, and return status.
4. **My Profile** — Profile card showing name, email, department, student ID, and statistics (total borrowed, currently holding, overdue).

Modals:

- **Borrow Modal** — Confirmation dialog with book preview and due date calculation.
- **Return Modal** — Confirmation dialog for returning a book.
- **Book Reader Modal** — Full-screen PDF viewer using `<iframe>` with blob URLs, or a search-links panel for books without uploaded PDFs (links to Internet Archive, Google Books, Open Library, Google PDF Search, Library Genesis).

Authentication Flow: Same pattern as admin — checks `API.isStudent()`, redirects if not authenticated, then calls `initStudentDashboard()`.

---

6. Frontend — CSS Stylesheet

6.1 `css/styles.css` (2066 lines)

Purpose: Complete styling system using CSS custom properties and glassmorphism design language.

Key Sections:

- **CSS Variables** (`:root`) — 14 design tokens including colors, glass effects, shadows, and transitions.
- **Reset & Base** — Universal box-sizing reset, Poppins font, gradient background.
- **Background Decoration** — Three animated floating circles with `@keyframes float` (20s infinite loop).

- **Landing Card** — Glass-morphism card with `backdrop-filter: blur(20px)` and `@keyframes slideUp` entrance animation.
- **Logo** — `@keyframes pulse` animation (3s breathing effect).
- **Buttons** — Six button variants (admin, student, primary, secondary, success, logout) with gradient backgrounds, hover transforms, and an animated shimmer pseudo-element `::before`.
- **Dashboard Layout** — Flexbox layout with 280px fixed sidebar and flex-1 main content area.
- **Sidebar** — Fixed-height navigation with active state gradient, icon support, and footer logout.
- **Tab Content** — Show/hide with `.active` class and `@keyframes fadeIn`.
- **Books Grid** — CSS Grid with `auto-fill, minmax(280px, 1fr)`, hover transforms, and cover image scaling.
- **Data Tables** — Styled with glass background, uppercase headers, hover row highlighting, and status badges.
- **History Timeline** — Flexbox timeline with red left-border accent.
- **Profile Card** — Centered glass card with emoji avatar and stats row.
- **Modals** — Fixed overlay with centered content, blur backdrop, and header/body/footer layout.
- **Book Reader** — Full-width modal with embedded `<iframe>` for PDF viewing at 80vh height.
- **File Upload Area** — Dashed border dropzone with drag-over state.
- **Deleted Books** — Cards with red-tinted overlay and restore/delete buttons.
- **Notifications** — Fixed-position toast with gradient background and slide-in/out animations.
- **Responsive Design** — Media queries for screens ≤768px (sidebar collapses, grid goes single-column).

---

7. Frontend — JavaScript Modules

7.1 `js/api.js` — API Client (190 lines)

Purpose: Central API communication layer. Replaces IndexedDB with server-side RESTful API calls.

Key Logic:

- `API_BASE` = `'/api'` — Uses relative paths for same-origin requests.
- **Session Restoration** — IIFE `restoreSession()` runs on load, reads `sessionId`, `userType`, and `userData` from `localStorage` to restore state after page refreshes.
- `API.getBooks()` — GET `/api/books`, returns array of books or empty array on error.
- `API.addBook(bookData)` — POST `/api/books` with JSON body (title, author, dept, fileData, etc.).
- `API.adminLogin(password)` — POST `/api/auth/admin/login`. On success, stores `sessionId` and `userType: 'admin'` in `localStorage`.
- `API.studentRegister(data)` — POST `/api/auth/student/register`.
- `API.studentLogin(studentId, password)` — POST `/api/auth/student/login`. On success, stores session data and full user object in `localStorage`.
- `API.borrowBook(bookId, studentId, studentName)` — POST `/api/borrow`.
- `API.getBorrowingHistory()` — GET `/api/borrow/all`.
- `API.getStudents()` — GET `/api/students`.
- **Session Helpers** — `isLoggedIn()`, `isAdmin()`, `isStudent()`, `getCurrentUser()`, `logout()` — all based on `localStorage`.
- **Global Export** — `window.API = API` makes it accessible from all scripts.

7.2 `js/api-bridge.js` — API Bridge (123 lines)

Purpose: Migration layer that bridges the old IndexedDB-based code with the new server API, allowing both to coexist.

Key Logic:

- Wraps in an IIFE to avoid global pollution.
- Overrides `window.getAllBooks` and `window.addBook` to use `API.getBooks()` and `API.addBook()` instead of IndexedDB.
- If `window.dbModule` exists, patches `dbModule.dbGetAll('books')` and `dbModule.dbAdd('books', data)` to route through the API.
- Falls back to original IndexedDB functions if API calls fail.
- **Auto-Sync** — `window.startBookSync()` sets up a `setInterval` (every 5 seconds) that checks if the books tab is visible and refreshes the grid via `loadBooksGrid()`.
- **Stop Sync** — `window.stopBookSync()` clears the interval.
- Auto-starts sync 2 seconds after DOM is ready.

7.3 `js/database.js` — IndexedDB Module (157 lines)

Purpose: Local browser-based database using IndexedDB. Serves as a fallback when the server is unavailable.

Key Logic:

- **`initDatabase()`** — Opens IndexedDB `eGranthalayaDB` version 2. On `onupgradeneeded`, creates four object stores:
  - `students` (keyPath: `studentId`, indexes: `email`, `department`)
  - `books` (keyPath: `id` auto-increment, indexes: `department`, `title`)
  - `borrowingHistory` (keyPath: `id` auto-increment, indexes: `studentId`, `bookId`, `status`)
  - `deletedBooks` (keyPath: `id` auto-increment, indexes: `department`, `title`, `deletedAt`)
- **CRUD Operations** — Generic Promise-wrapped functions: `dbAdd`, `dbPut`, `dbGet`, `dbGetAll`, `dbGetByIndex`, `dbDelete`, `dbClear`.
- **Exported as** `window.dbModule` for use by other scripts.

7.4 `js/auth.js` — Authentication Module (146 lines)

Purpose: Client-side authentication helpers and department definitions.

Key Logic:

- **`ADMIN_PASSWORD`** = `'112233'` — Hardcoded for client-side validation (backup).
- **`DEPARTMENTS`** — Array of three department objects with `id`, `name`, and `icon` emoji.
- **`adminLogin(password)`** — Compares against hardcoded password, sets `localStorage.currentUser`.
- **`studentLogin(name, email, department)`** — Validates inputs, generates a student ID via `generateStudentId(email)`, creates student data, saves to `localStorage`, and calls `registerStudent()`.
- **`generateStudentId(email)`** — Creates ID from email hash: `STU` + last 4 digits of char-code sum + first 3 chars of username uppercase.
- **`registerStudent(studentData)`** — Writes to IndexedDB via `dbModule.dbPut('students', ...)`.
- **`getAllStudents()`** — Tries `API.getStudents()` first, falls back to `dbModule.dbGetAll('students')`.
- **`logout()`** — Clears `localStorage.currentUser`, redirects to `index.html`.

7.5 `js/books.js` — Books Module (362 lines)

Purpose: Book data definitions, borrowing logic, and utility functions.

Key Logic:

- `BORROWING_DAYS` = 15 days.
- `generateBookCover(title, dept)` — Creates SVG data URIs with department-colored backgrounds (purple for CS, amber for Mechanical, green for Mining) and the book title's first letter.
- `DEPARTMENT_BOOKS` — Hardcoded catalog of ~48 books across 3 departments. First 4 CS books include embedded base64 PDF file data for inline reading.
- `initializeBooks()` — Currently disabled (admin must upload manually).
- `getAllBooks()` — Tries API first, falls back to IndexedDB.
- `getBookById(id)` — Fetches all books from API, filters by ID.
- `isBookBorrowed(bookId)` — Checks borrowing history for any `status === 'active'` record matching the book ID.
- `borrowBook(bookId, studentData)` — Validates availability, creates borrow record with auto-calculated due date (15 days), saves via IndexedDB.
- `returnBook(recordId)` — Updates record status to `'returned'` with current timestamp.
- `getStudentStats(studentId)` — Computes total borrowed, currently holding, and overdue counts.
- **Date Utilities** — `formatDate()`, `formatTime()`, `formatDateTime()`, `calculateDueDate()`, `isOverdue()`, `getDaysStatus()` — all use Indian locale (`en-IN`).

7.6 `js/app.js` — Main Application Controller (1556 lines)

Purpose: The largest file in the project. Orchestrates the entire UI: data loading, grid rendering, modal management, file upload, book deletion/restoration, and the book reader.

Key Logic Sections:

Admin Dashboard Init (`initAdminDashboard`):

1. Initializes IndexedDB via `dbModule.initDatabase()`.
2. Calls `initializeBooks()` (currently no-op).
3. Loads all four data grids: books, borrowing records, students list, deleted books.
4. Initializes the file upload area.

`loadBooksGrid(isAdmin)` — Book Grid Rendering:

1. Fetches all books via `getAllBooks()`.
2. Applies department filter if selected.
3. For each book, checks borrow status via `isBookBorrowed()`.
4. Generates HTML cards with conditional buttons:
   - Admin view: Delete button, borrower info.
   - Student view: Borrow/Return/Not Available buttons based on status.
5. Uses `Promise.all()` for parallel async card generation.

`filterBooks()` — Search Filtering:

- Reads search input, iterates DOM cards, toggles `display` based on title match.

`loadBorrowingRecords()` — Admin Records Table:

- Fetches history, sorts newest-first, renders table rows with status badges (Active/Overdue/Returned).

`loadStudentsList()` — Admin Students View:

- Fetches students and history, cross-references to compute per-student statistics.
- Generates both summary table rows and detailed per-student cards with borrowed/returned book lists.

Student Dashboard Init (`initStudentDashboard`):

1. Same DB/book init as admin.
2. Updates UI with student name and ID from `getCurrentUser()`.
3. Loads all four tabs: books grid, borrowed books, history timeline, profile.

`loadStudentHistoryTimeline()` — History Timeline:

- Fetches student's history, sorts newest-first, renders timeline items with status indicators.

`loadStudentProfile()` — Profile Tab:

- Displays student info and statistics from `getStudentStats()`.

Modal System (Borrow/Return/Delete):

- `showBorrowModal(bookId)` — Shows book preview with due date, binds `confirmBorrow()`.
- `confirmBorrow()` — Calls `borrowBook()`, refreshes grids, shows success notification.
- `showReturnModal(bookId, recordId)` — Shows return confirmation, binds `confirmReturn()`.

- `confirmReturn()` — Calls `returnBook()`, refreshes grids.
- `showDeleteModal(bookId)` — Admin only. Shows warning if book is borrowed. Binds `confirmDeleteBook()`.
- `confirmDeleteBook()` — Moves book to `deletedBooks` store (soft delete), removes from active books.

Deleted Books Management:

- `loadDeletedBooks()` — Renders deleted books grid sorted by deletion date.
- `restoreBook(id)` — Moves book from `deletedBooks` back to `books` store.
- `permanentDeleteBook(id)` — Removes from `deletedBooks` after user confirmation.

File Upload System (`initAddBooksTab`):

- Click and drag-and-drop handlers on `#fileUploadArea`.
- `handleFileSelection(file)` — Validates file type (PDF/EPUB), shows file info, extracts title from filename, displays metadata form.
- `addBookToLibrary()` — Converts file to base64 via `FileReader.readAsDataURL()`, creates book object, sends to server via `API.addBook()`, updates grids.
- `fileToBase64(file)` — Promise wrapper around FileReader.

Book Reader (`openBookReader`):

- If book has PDF `fileData`: Converts base64 data URI to Blob, creates `URL.createObjectURL()`, renders in `<iframe>`.
- If no PDF: Shows book info card with external reading links (Internet Archive, Google Books, Open Library, Google PDF Search, Library Genesis).

Notifications (`showNotification`):

- Creates a fixed-position div with gradient background (green/red/purple based on type).
- Auto-removes after 4 seconds with CSS slide-out animation.

Cross-Tab Sync:

- Listens for `storage` events on `bookListUpdated` key.
- When detected, reloads book grid and shows notification about new books.

---

8. Backend — Server Files

8.1 `backend/server.js` — Express MVC Server (177 lines)

Purpose: Production-grade server using Express.js with MVC architecture and SQLite database.

Key Logic:

1. **Middleware Setup:**
   - CORS with credentials enabled.
   - JSON and URL-encoded body parsing with 100MB limit (for large PDF uploads).
   - Express-session with 24-hour cookie expiry.
2. **Database Init:** Requires `./config/database` which creates SQLite tables on load.
3. **Static File Serving:** Serves frontend from `../e-granthalaya-main/` directory.
4. **API Route Mounting:**
   - `/api/auth` → `routes/auth.js`
   - `/api/books` → `routes/books.js`
   - `/api/borrow` → `routes/borrow.js`
   - `/api/students` → `routes/students.js`
5. **HTML Routes:** Explicit routes for each HTML page.
6. **Sample Books Init:** On first run, seeds 10 sample books if database is empty.
7. **Vercel Detection:** If `process.env.VERCEL` is set, exports `app` instead of listening.
8. **Graceful Shutdown:** Handles `SIGINT` for clean exit.

8.2 `backend/server-simple.js` — Zero-Dependency Server (418 lines)

Purpose: Lightweight server using only built-in Node.js modules (`http`, `fs`, `path`, `crypto`). No npm dependencies required.

Key Logic:

1. **In-Memory Database (`db` object):** Books, students, borrowing history, admin password (SHA-256 hashed), and sessions — all stored in a JS object.
2. **File Persistence:**
   - `loadDatabase()` — Reads `database/database.json` for students/history and `database/books.json` for books.
   - `saveDatabase()` — Writes students/history to `database.json`. Skips on Vercel.
   - `saveBooksDatabase()` — Writes books to `books.json`. Skips on Vercel.

3. **HTTP Server Creation:** Raw `http.createServer()` with URL pattern matching.
4. **API Routes (inline):**
   - `GET /api/books` — Returns `db.books`.
   - `POST /api/books` — Adds book with all fields including `fileData`. Auto-assigns incremental ID.
   - `POST /api/auth/admin/login` — SHA-256 hashes input, compares with stored hash.
   - `POST /api/auth/student/register` — Validates required fields, stores password as plain text.
   - `POST /api/auth/student/login` — Plain text password comparison, records login time.
   - `POST /api/borrow` — Creates borrow record with 15-day due date.
   - `GET /api/borrow/all` — Returns all borrowing records.
   - `GET /api/students` — Returns all students.
5. **Static File Serving:** `serveFile()` reads files with content-type mapping for `.html`, `.js`, `.css`, `.json`, `.png`, `.jpg`.
6. **Security:** Path normalization to prevent directory traversal attacks.
7. **CORS:** Headers added to all API responses.

---

9. Backend — Configuration

9.1 `backend/config/database.js` — SQLite Database (102 lines)

Purpose: Initializes the SQLite database using `better-sqlite3` and creates all tables.

Key Logic:

1. **Database Connection:** Opens `database/library.db` with verbose logging.
2. **Foreign Keys:** Enabled via `PRAGMA foreign_keys = ON`.
3. **`hashPassword(password)`** — SHA-256 hashing using Node.js `crypto` module (not bcrypt, for simplicity).
4. **`initializeDatabase()`** — Creates four tables:
   - **`students`** — `id`, `student_id` (unique), `name`, `email` (unique), `password`, `department`, `last_login`, `created_at`.
   - **`admin_users`** — `id`, `username` (unique, default 'admin'), `password`, `created_at`.

- **`books`** — `id`, `title`, `author`, `department`, `image`, `description`, `file_path`, `file_name`, `file_size`, `added_by`, `added_at`, `deleted` (0/1), `deleted_at`.
- **`borrowing_history`** — `id`, `book_id` (FK), `student_id` (FK), `book_title`, `book_department`, `student_name`, `student_email`, `student_department`, `borrow_date`, `due_date`, `return_date`, `status`.

5. **Default Admin:** Inserts admin user with hashed password `112233` if not exists.
6. **Exports:** The `db` instance and `hashPassword` function.

9.2 `backend/package.json` — Backend Dependencies

Dependencies: `express`, `cors`, `express-session`, `bcryptjs`, `better-sqlite3`, `multer`.

---

10. Backend — Models (Data Layer)

10.1 `backend/models/Book.js` (99 lines)

Purpose: Database operations for the `books` table.

Methods:

- `findAll()` — `SELECT * FROM books WHERE deleted = 0 ORDER BY added_at DESC`.
- `findById(id)` — Single book lookup excluding deleted.
- `findByDepartment(dept)` — Filter by department.
- `create(bookData)` — INSERT with title, author, department, image, description, file_path, file_name, file_size, added_by.
- `update(id, data)` — UPDATE title, author, department, image, description.
- `delete(id)` — Soft delete: sets `deleted = 1` and `deleted_at` = now.
- `findDeleted()` — Returns soft-deleted books.
- `restore(id)` — Sets `deleted = 0`, clears `deleted_at`.
- `isBorrowed(id)` — Checks `borrowing_history` for active record.

10.2 `backend/models/BorrowRecord.js` (79 lines)

Purpose: Database operations for `borrowing_history` table.

Methods:

- `findAll()` — All records sorted by borrow_date DESC.
- `findActive()` — Only `status = 'active'` records.

- `findByStudentId(studentId)` — Student's full history.
- `findActiveByStudentId(studentId)` — Student's active borrowings.
- `isBookBorrowed(bookId)` — Returns boolean.
- `getBookBorrower(bookId)` — Returns the active borrow record.
- `create(borrowData)` — INSERT with all borrow details, status defaults to 'active'.
- `returnBook(recordId)` — UPDATE status to 'returned', set return_date.
- `findById(id)` — Single record lookup.

10.3 `backend/models/Student.js` (77 lines)

Purpose: Database operations for the `students` table.

Methods:

- `findAll()` — Returns all students (excludes password field).
- `findByStudentId(studentId)` — Full student record including password.
- `findByEmail(email)` — Lookup by email.
- `create(studentData)` — INSERT with SHA-256 hashed password.
- `verifyPassword(studentId, password)` — Hashes input and compares with stored hash.
- `updateLastLogin(studentId)` — Sets `last_login` to current timestamp.
- `getStats(studentId)` — Aggregation query returning `total_borrowed`, `currently_holding`, `returned` counts from `borrowing_history`.

---

11. Backend — Controllers (Business Logic)

11.1 `backend/controllers/authController.js` (115 lines)

- `adminLogin` — Hashes input password, compares with DB admin record, sets session.
- `studentRegister` — Checks for duplicate student ID and email, then creates student.
- `studentLogin` — Finds student by ID, verifies password hash, updates last login, sets session with user data.
- `logout` — Destroys session.
- `checkSession` — Returns current session user data or `success: false`.

11.2 `backend/controllers/bookController.js` (148 lines)

- `getAllBooks` — Returns all non-deleted books.

- `getBookById` — Returns single book by ID.
- `getBooksByDepartment` — Filter by department parameter.
- `addBook` — Accepts multipart form with file upload via Multer. Creates book record with file path.
- `updateBook` — Updates book metadata.
- `deleteBook` — Soft-deletes after checking book isn't currently borrowed.
- `getDeletedBooks` — Returns all soft-deleted books.
- `restoreBook` — Restores a soft-deleted book.
- `downloadBook` — Sends the uploaded PDF file via `res.sendFile()`.

11.3 `backend/controllers/borrowController.js` (126 lines)

- `borrowBook` — Validates book exists and isn't borrowed, gets student from session, creates 15-day borrow record.
- `returnBook` — Marks record as returned with current timestamp.
- `getAllRecords` — Admin: all history records.
- `getActiveRecords` — Admin: only active borrowings.
- `getStudentRecords` — Student's full history.
- `getStudentActiveRecords` — Student's currently borrowed books.

11.4 `backend/controllers/studentController.js` (46 lines)

- `getAllStudents` — Admin: returns all students.
- `getStudentById` — Admin: returns student with stats.
- `getCurrentStudent` — Student: returns own profile with stats from session.

---

12. Backend — Routes (API Endpoints)

12.1 `backend/routes/auth.js` (22 lines)

METHOD	PATH	HANDLER	AUTH REQUIRED
POST	`/api/auth/admin/login`	`adminLogin`	No
POST	`/api/auth/student/register`	`studentRegister`	No
POST	`/api/auth/student/login`	`studentLogin`	No
POST	`/api/auth/logout`	`logout`	No
GET	`/api/auth/check`	`checkSession`	No

12.2 `backend/routes/books.js` (22 lines)

METHOD	PATH	HANDLER	AUTH REQUIRED
GET	`/api/books`	`getAllBooks`	Auth
GET	`/api/books/:id`	`getBookById`	Auth
GET	`/api/books/department/:dept`	`getBooksByDepartment`	Auth
GET	`/api/books/:id/download`	`downloadBook`	Auth
POST	`/api/books`	`addBook` (with Multer)	Admin
PUT	`/api/books/:id`	`updateBook`	Admin
DELETE	`/api/books/:id`	`deleteBook`	Admin
GET	`/api/books/deleted/all`	`getDeletedBooks`	Admin
POST	`/api/books/:id/restore`	`restoreBook`	Admin

12.3 `backend/routes/borrow.js` (19 lines)

METHOD	PATH	HANDLER	AUTH REQUIRED
POST	`/api/borrow`	`borrowBook`	Student
POST	`/api/borrow/return/:id`	`returnBook`	Student
GET	`/api/borrow/my-history`	`getStudentRecords`	Student
GET	`/api/borrow/my-active`	`getStudentActiveRecords`	Student
GET	`/api/borrow/all`	`getAllRecords`	Admin
GET	`/api/borrow/active`	`getActiveRecords`	Admin
GET	`/api/borrow/student/:studentId`	`getStudentRecords`	Admin

12.4 `backend/routes/students.js` (15 lines)

METHOD	PATH	HANDLER	AUTH REQUIRED
GET	`/api/students`	`getAllStudents`	Admin
GET	`/api/students/:id`	`getStudentById`	Admin
GET	`/api/students/me/profile`	`getCurrentStudent`	Student

13. Backend — Middleware

13.1 `backend/middleware/auth.js` (26 lines)

Purpose: Three Express middleware functions for route protection.

- `requireAuth` — Checks `req.session.user` exists. Returns 401 if not.
- `requireAdmin` — Checks `req.session.user.type === 'admin'`. Returns 403 if not.
- `requireStudent` — Checks `req.session.user.type === 'student'`. Returns 403 if not.

13.2 `backend/middleware/upload.js` (47 lines)

Purpose: Configures Multer for PDF/EPUB file uploads.

Key Logic:

- **Storage:** Disk storage in `backend/uploads/` directory. Auto-creates directory if missing.
- **Filename:** `Date.now()-<random>.<ext>` for uniqueness.
- **File Filter:** Only accepts files matching `/pdf|epub/` in both extension and MIME type.
- **Size Limit:** 100MB maximum.

---

14. Database Files

14.1 `database/database.json`

Purpose: JSON-file persistence for the simple server. Stores:

- `students` array — Each with `id`, `studentId`, `name`, `email`, `password` (plain text), `department`, `createdAt`, `lastLogin`, `loginHistory[]`.
- `borrowingHistory` array — Borrow/return records.
- `adminPassword` — SHA-256 hash of `'112233'`.

14.2 `database/books.json`

Purpose: Separate JSON storage for books (to keep file sizes manageable since books may contain large base64-encoded PDF data). Contains a `books` array with book objects including optional `fileData` field.

---

15. Deployment — Vercel Serverless API

15.1 `api/[...path].js` (171 lines)

Purpose: Vercel-native serverless function that implements all API endpoints in a single file using in-memory storage.

Key Logic:

- `getBody(req)` — Helper to parse JSON request body.
- `handler(req, res)` — Default export; handles all API routes.
- `global.db` — In-memory database initialized once per cold start with 8 sample books.
- **Routes implemented:** Same as simple server (books CRUD, auth, borrow).

- **Limitations:** Data is ephemeral — resets on each cold start since Vercel functions are stateless.

16. Application Flow — End-to-End

Admin Flow:

```
index.html → admin-login.html → POST /api/auth/admin/login
  → admin-dashboard.html → initAdminDashboard()
    → loadBooksGrid(true) ← GET /api/books
    → loadBorrowingRecords() ← GET /api/borrow/all
    → loadStudentsList() ← GET /api/students
    → Add Book → POST /api/books (with base64 PDF)
    → Delete Book → Soft-delete to deletedBooks store
    → Restore Book → Move back to active books
```

Student Flow:

```
index.html → student-login.html → Register: POST /api/auth/student/register
                                → Login: POST /api/auth/student/login
  → student-dashboard.html → initStudentDashboard()
    → loadBooksGrid(false) ← GET /api/books
    → Borrow Book → POST /api/borrow → loadBorrowedBooks()
    → Read Book → openBookReader() → PDF iframe or external links
    → Return Book → returnBook(recordId) → Update status to 'returned'
    → View History → loadStudentHistoryTimeline()
    → View Profile → loadStudentProfile()
```

17. API Reference

ENDPOINT	METHOD	DESCRIPTION	REQUEST BODY	RESPONSE
`/api/auth/admin/login`	POST	Admin login	`{password}`	`{success, sessionId}`
`/api/auth/student/register`	POST	Student signup	`{studentId, name, email, password, department}`	`{success, message}`
`/api/auth/student/login`	POST	Student login	`{studentId, password}`	`{success, sessionId, user}`
`/api/books`	GET	List all books	—	`{success, data: [...]}`
`/api/books`	POST	Add new book	`{title, author, department, fileData, ...}`	`{success, bookId}`
`/api/borrow`	POST	Borrow a book	`{bookId, studentId, studentName}`	`{success, recordId, dueDate}`
`/api/borrow/all`	GET	All borrow records	—	`{success, data: [...]}`
`/api/students`	GET	All students	—	`{success, data: [...]}`

18. Security Considerations

1. **Password Storage:** The Express MVC server uses SHA-256 hashing. The simple server stores student passwords in plain text in `database.json`.
2. **Session Management:** Express-session with server-side storage. Client uses `localStorage` for session persistence across page refreshes.
3. **Admin Password:** Default `112233` — hardcoded in both frontend (`auth.js`) and backend.
4. **CORS:** Wide-open (`origin: true` / `*`) — suitable for development, should be restricted in production.
5. **File Upload:** Restricted to PDF/EPUB, max 100MB, with unique filenames to prevent overwrites.
6. **Directory Traversal:** Simple server normalizes paths to prevent `../` attacks.
7. **No HTTPS Enforcement:** Session cookies set with `secure: false`.

19. Summary Table of All Files

#	FILE PATH	LINES	SIZE	PURPOSE
1	`package.json`	22	633B	Root NPM config
2	`vercel.json`	23	434B	Vercel deployment config
3	`.gitignore`	4	29B	Git exclusions
4	`README.md`	73	1.9KB	Project documentation
5	`index.html`	44	1.6KB	Landing page
6	`admin-login.html`	74	2.7KB	Admin login form
7	`admin-dashboard.html`	365	16.3KB	Admin panel
8	`student-login.html`	288	11KB	Student login/register
9	`student-dashboard.html`	270	11.4KB	Student portal
10	`css/styles.css`	2066	38.9KB	Global stylesheet
11	`js/api.js`	190	6.6KB	API client
12	`js/api-bridge.js`	123	4.5KB	IndexedDB→API bridge
13	`js/database.js`	157	5.6KB	IndexedDB module
14	`js/auth.js`	146	4.4KB	Auth helpers
15	`js/books.js`	362	20.3KB	Books data & logic
16	`js/app.js`	1556	63.5KB	Main UI controller

#	FILE PATH	LINES	SIZE	PURPOSE
17	`backend/server.js`	177	6.2KB	Express MVC server
18	`backend/server-simple.js`	418	16.1KB	Zero-dep server
19	`backend/package.json`	28	629B	Backend NPM config
20	`backend/config/database.js`	102	3.3KB	SQLite schema & init
21	`backend/models/Book.js`	99	2.8KB	Book model
22	`backend/models/BorrowRecord.js`	79	2.6KB	Borrow model
23	`backend/models/Student.js`	77	2.4KB	Student model
24	`backend/controllers/authController.js`	115	3.7KB	Auth controller
25	`backend/controllers/bookController.js`	148	4.8KB	Book controller
26	`backend/controllers/borrowController.js`	126	4.3KB	Borrow controller
27	`backend/controllers/studentController.js`	46	1.5KB	Student controller
28	`backend/routes/auth.js`	22	574B	Auth routes
29	`backend/routes/books.js`	22	1KB	Book routes
30	`backend/routes/borrow.js`	19	842B	Borrow routes
31	`backend/routes/students.js`	15	532B	Student routes
32	`backend/middleware/auth.js`	26	784B	Auth middleware
33	`backend/middleware/upload.js`	47	1.3KB	File upload middleware
34	`database/database.json`	20	502B	JSON student DB

#	FILE PATH	LINES	SIZE	PURPOSE
35	`database/books.json`	—	149KB	JSON books DB
36	`api/[...path].js`	171	7.1KB	Vercel serverless API
37	`assets/logo.png`	—	22.4KB	School of Mines logo

Total Project Files: 37 (excluding `node_modules` and duplicate `e-granthalaya-main/` subdirectory)
Total Lines of Code: ~7,100+ lines
Total Project Size: ~350KB+ (excluding `node_modules` and large `books.json`)

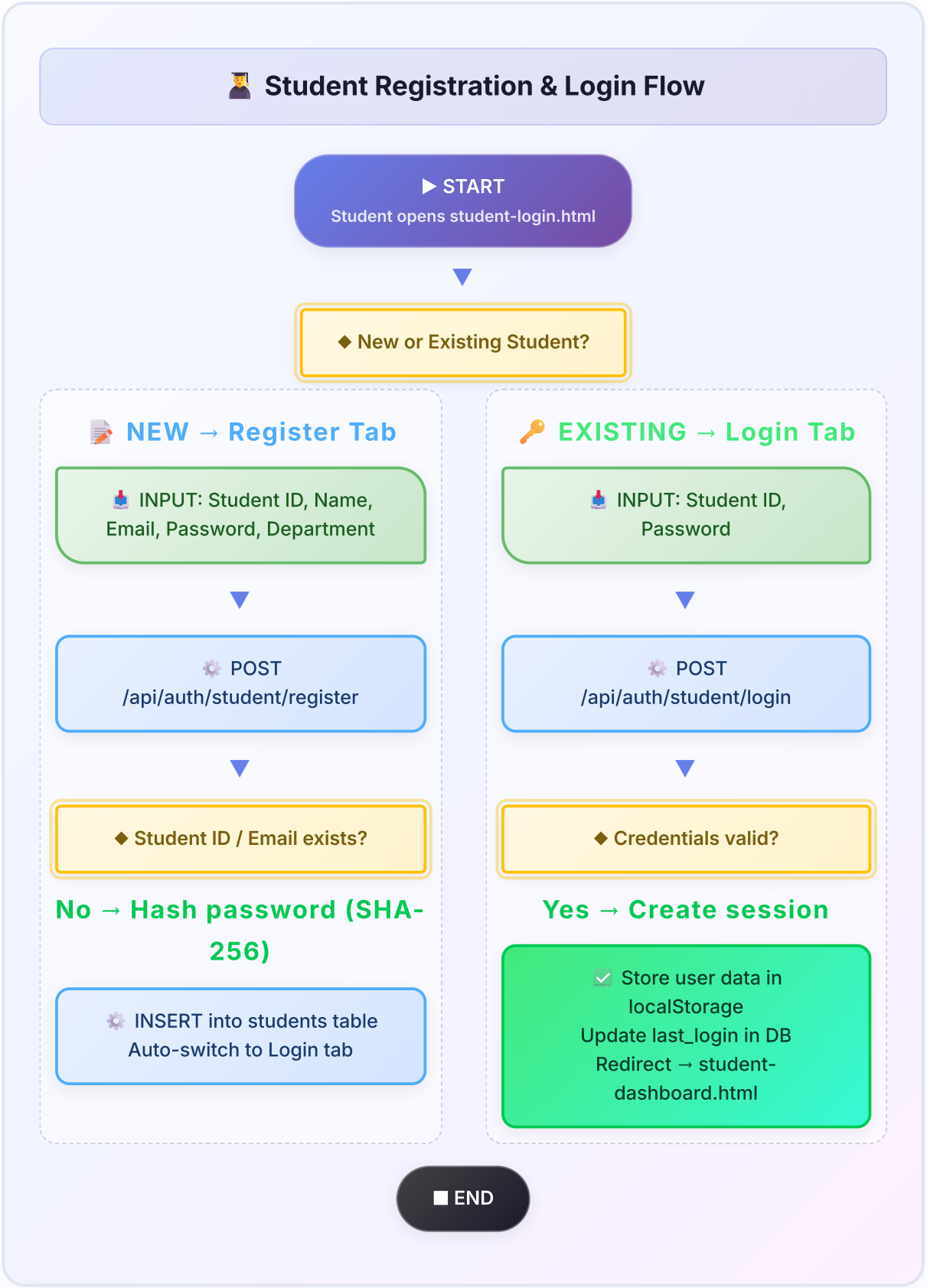
20. System Flowcharts

The following flowcharts illustrate the core workflows of the e-Granthalaya system.

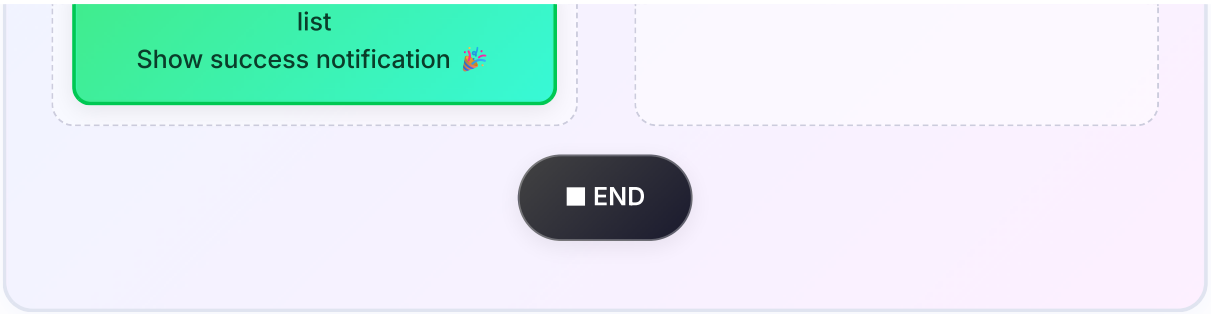
20.1 Overall System Architecture Flowchart

20.2 Admin Authentication Flowchart

20.3 Student Registration & Login Flowchart

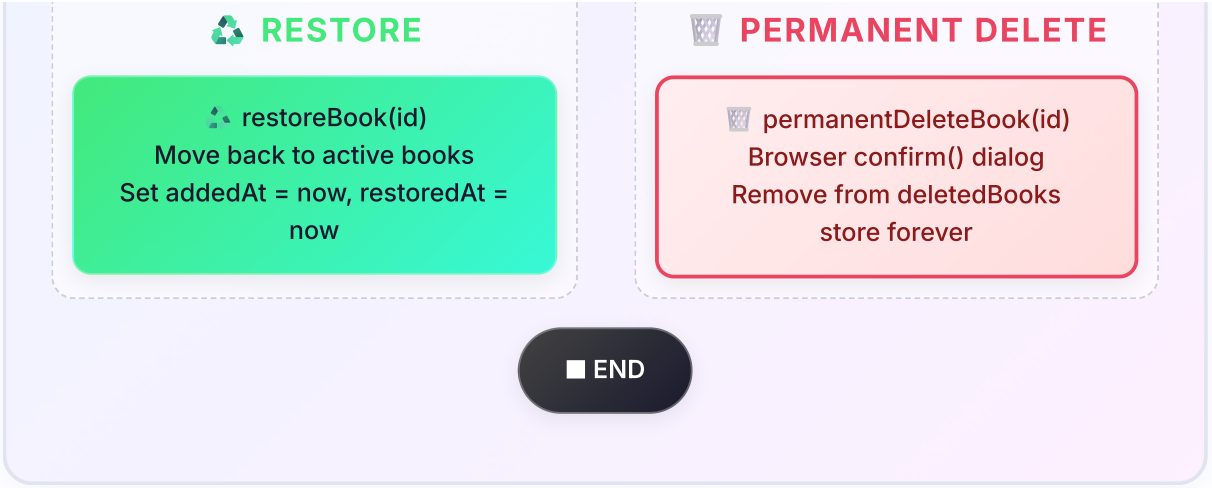


20.4 Book Borrowing Flowchart

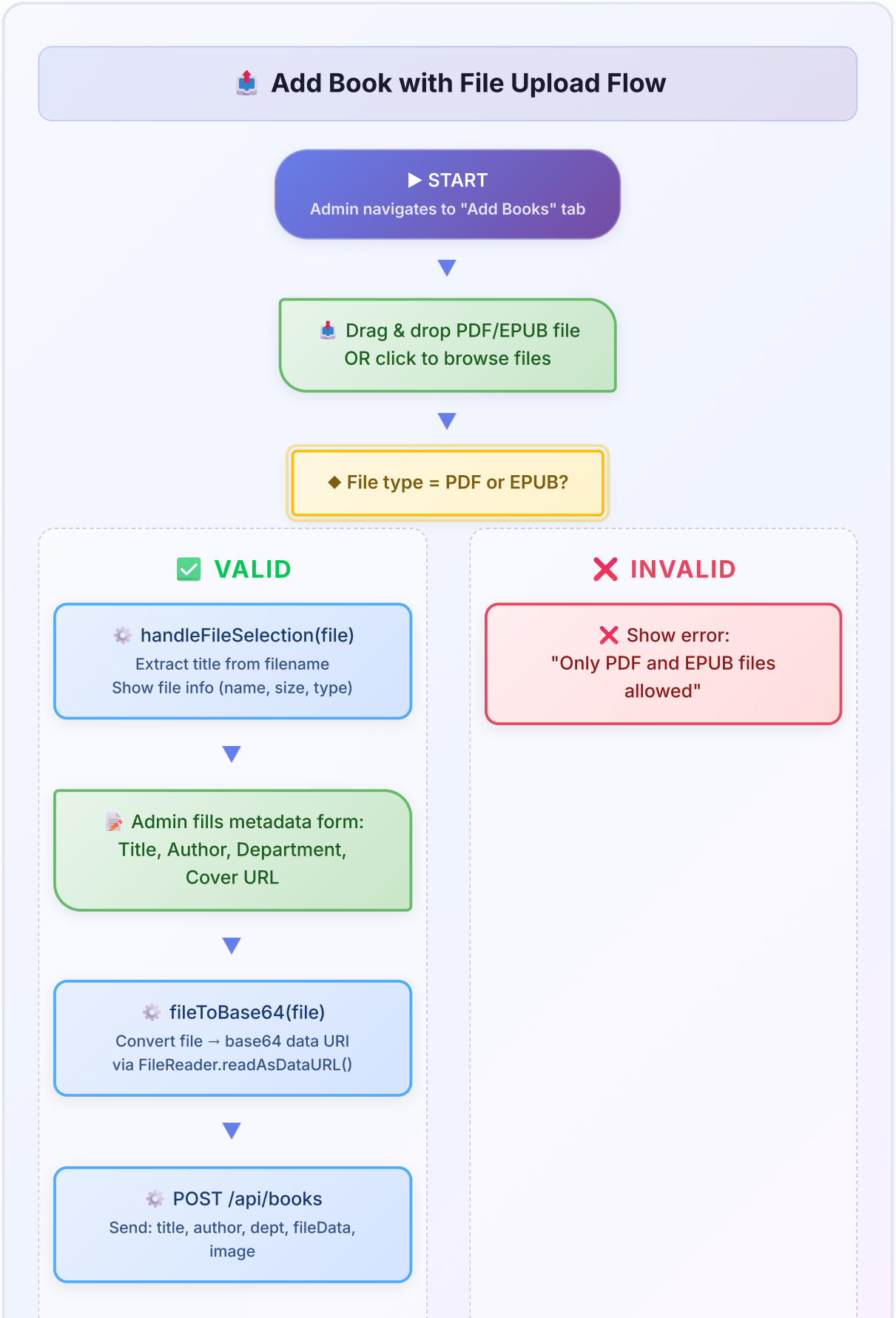

list
Show success notification 🎉
END

20.5 Book Return Flowchart

20.6 Book Deletion & Restoration Flowchart


RESTORE
restoreBook(id)
Move back to active books
Set addedAt = now, restoredAt = now
PERMANENT DELETE
permanentDeleteBook(id)
Browser confirm() dialog
Remove from deletedBooks store forever
END

20.7 Add Book (File Upload) Flowchart




✅ Book added to library!
Refresh book grid + recently added list
Reset upload form
■ END

20.8 Book Reader Flowchart

20.9 API Request Lifecycle Flowchart

21. Algorithms & Pseudocode

The following algorithms describe the core operations of the e-Granthalaya system in structured pseudocode.

Algorithm 1: Admin Authentication

```
ALGORITHM: AdminAuthentication
INPUT: password (string)
OUTPUT: sessionId (string) OR error (string)

BEGIN
    1. RECEIVE password from login form
    2. hashedInput ← SHA256(password)
    3. adminRecord ← DB.QUERY("SELECT * FROM admin_users WHERE username = 'admin'")

    4. IF adminRecord is NULL THEN
         RETURN error("Admin not found")
       END IF

    5. IF hashedInput ≠ adminRecord.password THEN
         RETURN error("Invalid password", status: 401)
       END IF

    6. sessionId ← GENERATE_RANDOM_HEX(16 bytes)
    7. SESSION.user ← { type: "admin", username: "admin" }
    8. STORE sessionId in client localStorage
    9. REDIRECT to admin-dashboard.html

    RETURN success(sessionId)
END

Time Complexity: O(1)
Space Complexity: O(1)
```

Algorithm 2: Student Registration

```
ALGORITHM: StudentRegistration
INPUT: studentId, name, email, password, department
OUTPUT: success (boolean) OR error (string)

BEGIN
    1. VALIDATE all required fields are non-empty
       IF any field is empty THEN
         RETURN error("Missing required fields")
       END IF

    2. existingById ← DB.QUERY("SELECT * FROM students WHERE student_id = ?",
studentId)
       IF existingById is NOT NULL THEN
         RETURN error("Student ID already registered")
       END IF

    3. existingByEmail ← DB.QUERY("SELECT * FROM students WHERE email = ?", email)
       IF existingByEmail is NOT NULL THEN
         RETURN error("Email already registered")
       END IF

    4. hashedPassword ← SHA256(password)

    5. DB.INSERT("students", {
         student_id: studentId,
         name: name,
         email: email,
         password: hashedPassword,
         department: department,
         created_at: CURRENT_TIMESTAMP
       })

    6. RETURN success("Student registered successfully")
END

Time Complexity: O(1) — indexed lookups
Space Complexity: O(1)
```

Algorithm 3: Student Login

```
ALGORITHM: StudentLogin
INPUT: studentId, password
OUTPUT: sessionId + userData OR error

BEGIN
    1. student ← DB.QUERY("SELECT * FROM students WHERE student_id = ?", studentId)

    2. IF student is NULL THEN
         RETURN error("Invalid student ID or password", status: 401)
       END IF

    3. hashedInput ← SHA256(password)

    4. IF hashedInput ≠ student.password THEN
         RETURN error("Invalid student ID or password", status: 401)
       END IF

    5. DB.UPDATE("students",
         SET last_login = CURRENT_TIMESTAMP
         WHERE student_id = studentId)

    6. sessionId ← GENERATE_RANDOM_HEX(16 bytes)
    7. SESSION.user ← {
         type: "student",
         studentId: student.student_id,
         name: student.name,
         email: student.email,
         department: student.department
       }

    8. STORE sessionId, userType, userData in client localStorage
    9. REDIRECT to student-dashboard.html

    RETURN success(sessionId, userData)
END

Time Complexity: O(1)
Space Complexity: O(1)
```

Algorithm 4: Book Borrowing

```
ALGORITHM: BorrowBook
INPUT: bookId, studentId (from session)
OUTPUT: recordId + dueDate OR error

BEGIN
    1. book ← DB.QUERY("SELECT * FROM books WHERE id = ? AND deleted = 0", bookId)

    2. IF book is NULL THEN
         RETURN error("Book not found", status: 404)
       END IF

    3. activeRecord ← DB.QUERY(
         "SELECT * FROM borrowing_history
          WHERE book_id = ? AND status = 'active'", bookId)

    4. IF activeRecord is NOT NULL THEN
         RETURN error("Book is already borrowed", status: 400)
       END IF

    5. student ← DB.QUERY("SELECT * FROM students WHERE student_id = ?", studentId)

    6. borrowDate ← CURRENT_TIMESTAMP
    7. dueDate ← borrowDate + 15 DAYS

    8. recordId ← DB.INSERT("borrowing_history", {
         book_id: bookId,
         student_id: studentId,
         book_title: book.title,
         book_department: book.department,
         student_name: student.name,
         student_email: student.email,
         student_department: student.department,
         borrow_date: borrowDate,
         due_date: dueDate,
         status: "active"
       })

    9. RETURN success(recordId, dueDate)
END

Time Complexity: O(1)
Space Complexity: O(1)
```

Algorithm 5: Book Return

```
ALGORITHM: ReturnBook
INPUT: recordId
OUTPUT: success OR error

BEGIN
    1. record ← DB.QUERY("SELECT * FROM borrowing_history WHERE id = ?", recordId)

    2. IF record is NULL THEN
         RETURN error("Borrow record not found", status: 404)
       END IF

    3. IF record.status = "returned" THEN
         RETURN error("Book already returned", status: 400)
       END IF

    4. DB.UPDATE("borrowing_history",
         SET status = "returned",
             return_date = CURRENT_TIMESTAMP
         WHERE id = recordId)

    5. RETURN success("Book returned successfully")
END

Time Complexity: O(1)
Space Complexity: O(1)
```

Algorithm 6: Load Books Grid (Frontend)

```
ALGORITHM: LoadBooksGrid
INPUT: isAdmin (boolean)
OUTPUT: Rendered HTML book cards in the DOM

BEGIN
    1. allBooks ← API.getBooks()          // GET /api/books
    2. selectedDept ← GET value from department filter dropdown

    3. IF selectedDept ≠ "all" THEN
         filteredBooks ← FILTER allBooks WHERE book.department = selectedDept
       ELSE
         filteredBooks ← allBooks
       END IF

    4. IF filteredBooks is EMPTY THEN
         RENDER empty state: "No books found"
         RETURN
       END IF

    5. UPDATE stats counter: "{count} Books in Library"

    6. cardPromises ← EMPTY ARRAY

    7. FOR EACH book IN filteredBooks DO
         promise ← ASYNC:
           a. isBorrowed ← isBookBorrowed(book.id)
           b. borrower ← IF isBorrowed THEN getBookBorrower(book.id) ELSE NULL
           c. bookCover ← IF book.image EXISTS THEN <img> ELSE generateBookCover()
           d. statusBadge ← IF isBorrowed THEN "Borrowed" ELSE "Available"
           e. actionButtons ←
              IF isAdmin THEN
                [Delete Button] + (IF isBorrowed THEN borrower info)
              ELSE  // Student view
                IF isBorrowed AND borrower.studentId = currentUser THEN
                  [Return Button]
                ELSE IF isBorrowed THEN
                  [Not Available - disabled]
                ELSE
                  [Borrow Button]
                END IF
           f. RETURN generated HTML card string
         ADD promise to cardPromises
       END FOR

    8. cards ← AWAIT Promise.all(cardPromises)
    9. grid.innerHTML ← JOIN all cards
END

Time Complexity: O(n × m) where n = books, m = borrow records
Space Complexity: O(n) for the card HTML array
```

Algorithm 7: Soft Delete Book (Admin)

```
ALGORITHM: SoftDeleteBook
INPUT: bookId
OUTPUT: success OR error

BEGIN
    1. book ← getBookById(bookId)

    2. IF book is NULL THEN
         RETURN error("Book not found")
       END IF

    3. isBorrowed ← isBookBorrowed(bookId)

    4. IF isBorrowed THEN
         // Clean up borrowing records
         history ← getBorrowingHistory()
         bookRecords ← FILTER history WHERE record.bookId = bookId

         FOR EACH record IN bookRecords DO
           DB.DELETE("borrowingHistory", record.id)
         END FOR
       END IF

    5. deletedBook ← COPY(book)
    6. deletedBook.originalId ← book.id
    7. deletedBook.deletedAt ← CURRENT_TIMESTAMP
    8. REMOVE deletedBook.id   // Let auto-increment assign new ID

    9. DB.INSERT("deletedBooks", deletedBook)
   10. DB.DELETE("books", bookId)

   11. REFRESH: loadBooksGrid(), loadRecentlyAddedBooks(),
                loadBorrowingRecords(), loadDeletedBooks()

   12. SHOW notification: "{title} moved to Deleted Books"

    RETURN success
END

Time Complexity: O(m) where m = borrow records for the book
Space Complexity: O(1)
```

Algorithm 8: File Upload & Add Book

```
ALGORITHM: AddBookWithFileUpload
INPUT: file (PDF/EPUB), title, author, department, coverUrl
OUTPUT: bookId OR error

BEGIN
    1. VALIDATE file.type IN ["application/pdf", "application/epub+zip"]
       IF invalid THEN
         RETURN error("Only PDF and EPUB files are allowed")
       END IF

    2. VALIDATE title AND author are non-empty
       IF empty THEN
         RETURN error("Title and author are required")
       END IF

    3. // Convert file to base64
       reader ← NEW FileReader()
       base64Data ← AWAIT reader.readAsDataURL(file)
       // Result: "data:application/pdf;base64,JVBERi0xLjQ..."

    4. bookData ← {
         title: title,
         author: author,
         department: department,
         image: coverUrl OR getDefaultCoverImage(department),
         fileName: file.name,
         fileSize: file.size,
         fileType: file.type,
         fileData: base64Data,
         addedAt: CURRENT_TIMESTAMP,
         addedBy: "admin"
       }

    5. response ← API.addBook(bookData)    // POST /api/books

    6. IF response.success THEN
         SET localStorage("bookListUpdated", timestamp)  // Cross-tab sync
         REFRESH: loadBooksGrid(), loadRecentlyAddedBooks()
         RESET upload form
         SHOW notification: "{title} added successfully!"
         RETURN response.bookId
       ELSE
         RETURN error(response.message)
       END IF
END

Time Complexity: O(n) where n = file size (for base64 encoding)
Space Complexity: O(n) for the base64 string
```

Algorithm 9: Book Search & Filter

```
ALGORITHM: SearchAndFilterBooks
INPUT: searchQuery (string), departmentFilter (string)
OUTPUT: Filtered DOM display

BEGIN
    1. query ← LOWERCASE(TRIM(searchQuery))
    2. cards ← document.querySelectorAll(".book-card")

    3. FOR EACH card IN cards DO
         title ← LOWERCASE(card.querySelector(".book-title").textContent)

         IF query is EMPTY OR title.INCLUDES(query) THEN
           card.style.display ← "block"    // Show card
         ELSE
           card.style.display ← "none"     // Hide card
         END IF
       END FOR
END

Time Complexity: O(n) where n = number of book cards
Space Complexity: O(1)
```

Algorithm 10: Overdue Detection

```
ALGORITHM: GetDaysStatus
INPUT: dueDate (ISO string)
OUTPUT: { days: number, overdue: boolean }

BEGIN
    1. due ← NEW Date(dueDate)
    2. now ← NEW Date()

    3. diffMs ← due - now                    // Milliseconds difference
    4. diffDays ← FLOOR(diffMs / (1000 × 60 × 60 × 24))

    5. IF diffDays < 0 THEN
         // Book is overdue
         RETURN { days: ABS(diffDays), overdue: true }
       ELSE
         // Book still has time
         RETURN { days: diffDays, overdue: false }
       END IF
END

Time Complexity: O(1)
Space Complexity: O(1)
```

Algorithm 11: Auto-Sync Book Data (API Bridge)

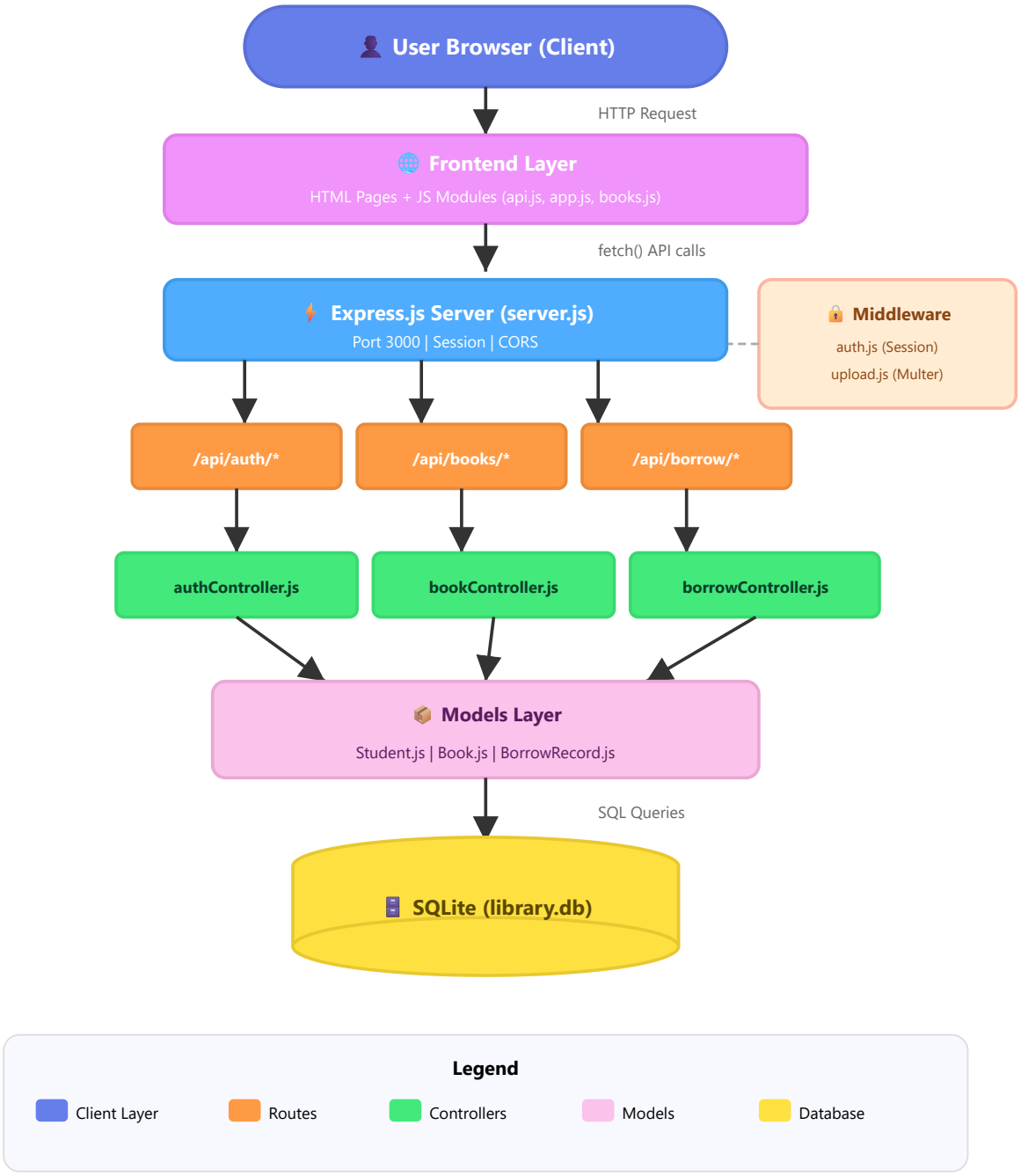
```
ALGORITHM: AutoSyncBooks
INPUT: syncInterval (default 5000ms)
OUTPUT: Periodically refreshed book grid

BEGIN
    1. syncTimerId ← NULL

    2. FUNCTION startBookSync():
         IF syncTimerId is NOT NULL THEN
           RETURN    // Already running
         END IF

         syncTimerId ← setInterval(syncFunction, 5000)
       END FUNCTION

    3. FUNCTION syncFunction():
         activeTab ← document.querySelector(".tab-content.active")

         IF activeTab.id CONTAINS "books" THEN
           isAdmin ← CHECK if admin dashboard
           CALL loadBooksGrid(isAdmin)
           PRINT "Auto-synced book data"
         END IF
       END FUNCTION

    4. FUNCTION stopBookSync():
         IF syncTimerId is NOT NULL THEN
           clearInterval(syncTimerId)
           syncTimerId ← NULL
         END IF
       END FUNCTION

    5. // Auto-start after 2 seconds
       WAIT 2000ms THEN startBookSync()
END

Time Complexity: O(n) per sync cycle (n = number of books)
Space Complexity: O(n) for book data in memory
```

This documentation was auto-generated by analyzing the complete e-Granthalaya source code.
Report Date: 23 February 2026

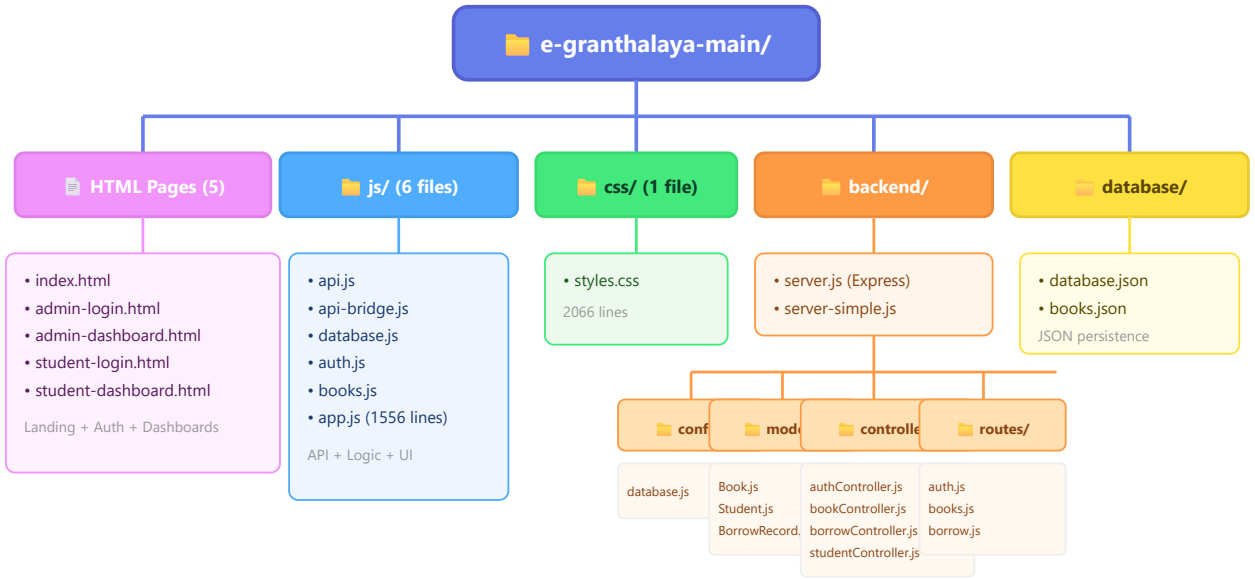
📚 e-Granthalaya — Flowcharts & Diagrams

Project Architecture, Folder Structure & Loop Analysis

1. Project Architecture Flowchart


User Browser (Client)
HTTP Request
Frontend Layer
HTML Pages + JS Modules (api.js, app.js, books.js)
fetch() API calls
Express.js Server (server.js)
Port 3000 | Session | CORS
Middleware
auth.js (Session)
upload.js (Multer)
/api/auth/*
/api/books/*
/api/borrow/*
authController.js
bookController.js
borrowController.js
Models Layer
Student.js | Book.js | BorrowRecord.js
SQL Queries
SQLite (library.db)
Legend
Client Layer
Routes
Controllers
Models
Database

2. Folder & File Structure Diagram



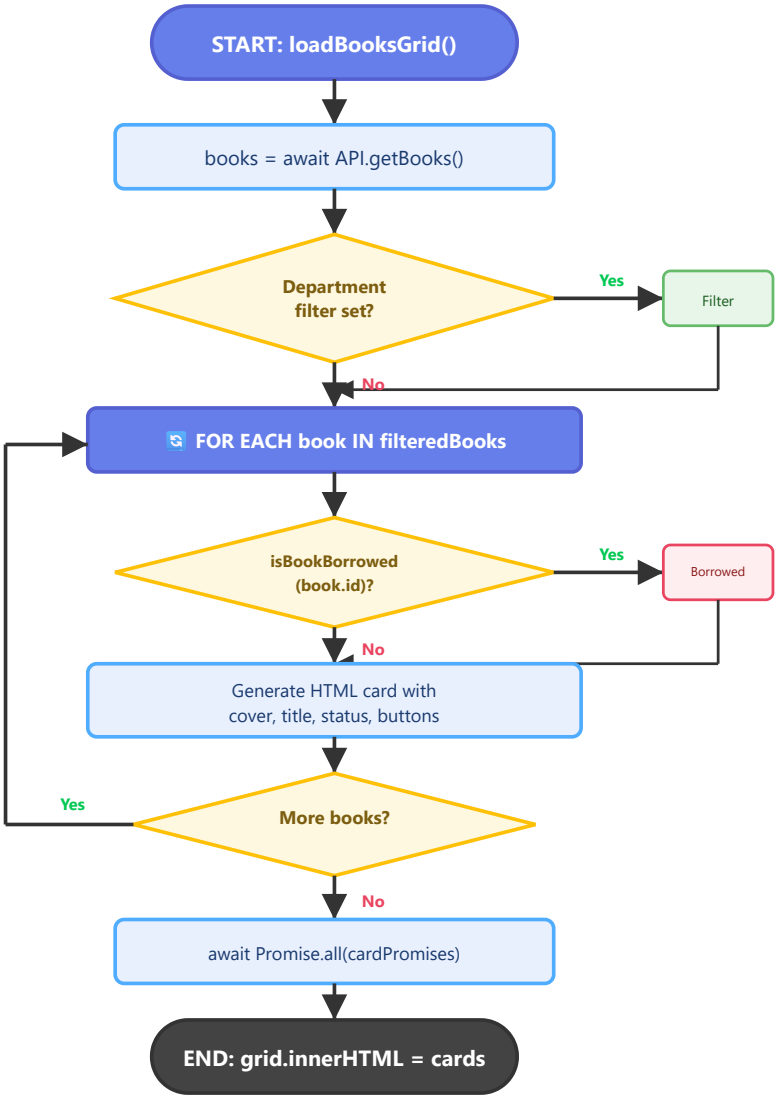
Total: 37 Files | ~7,100 Lines of Code | ~350KB

5 HTML | 6 JS | 1 CSS | 2 Servers | 3 Models | 4 Controllers | 4 Routes | 2 JSON

3. Loop Flowcharts

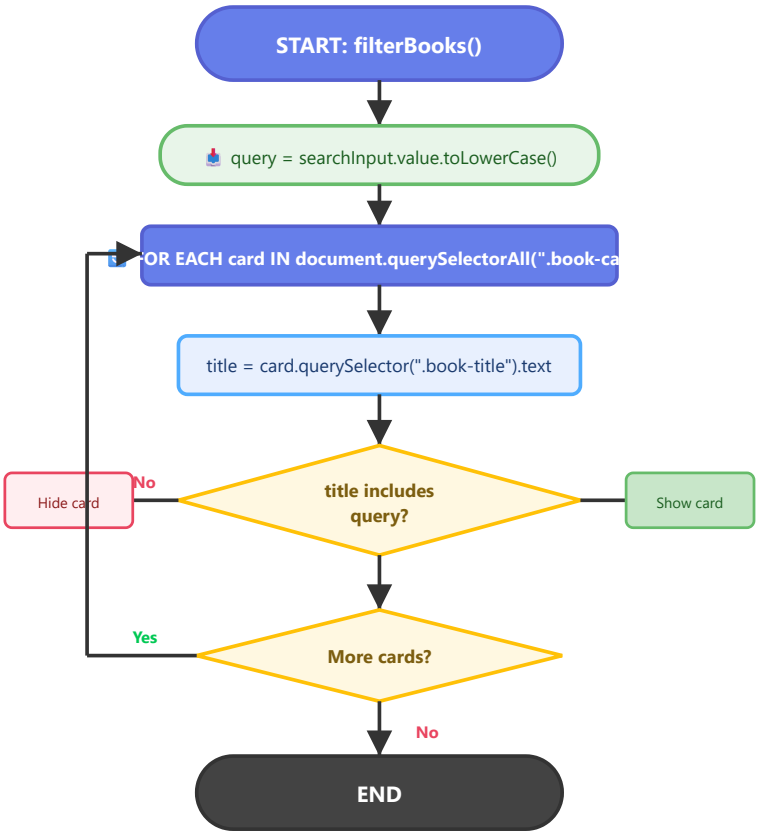
3.1 loadBooksGrid() — Book Rendering Loop

File: js/app.js | **Loop Type:** FOR EACH with Promise.all()



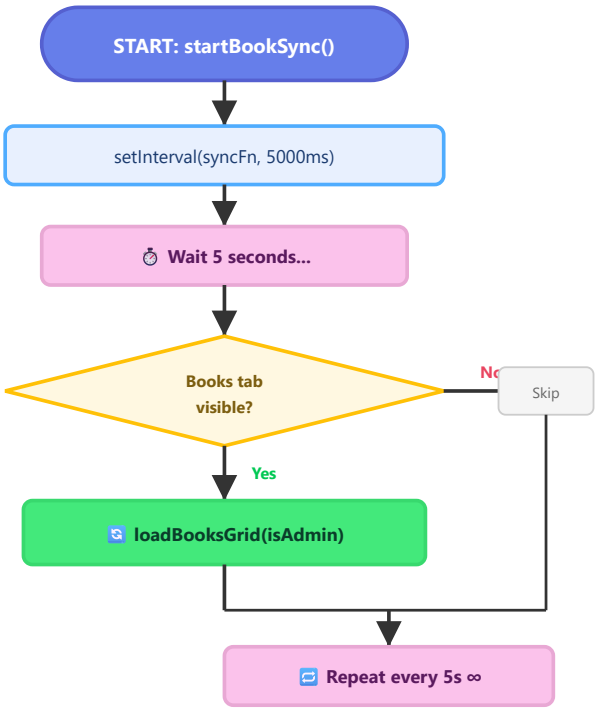
3.2 filterBooks() — Search Filter Loop

File: js/app.js | **Loop Type:** FOR EACH (DOM iteration)



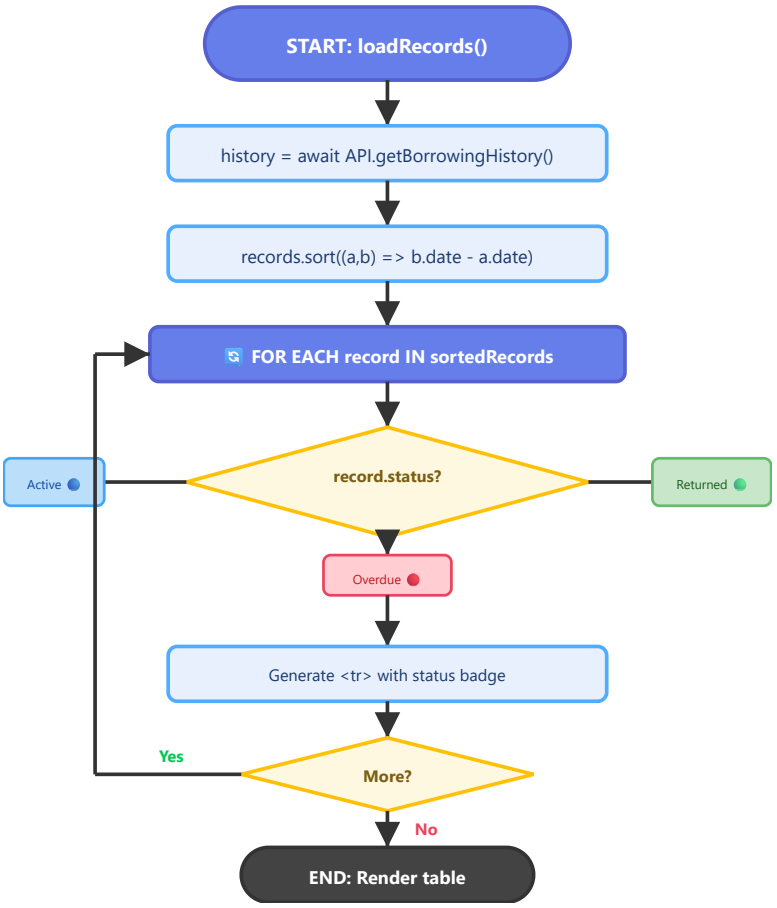
3.3 Auto-Sync Loop (api-bridge.js)

File: js/api-bridge.js | **Loop Type:** setInterval (every 5 seconds)


START: startBookSync()
setInterval(syncFn, 5000ms)
⏱ Wait 5 seconds...
Books tab visible?
No
Skip
Yes
🔄 loadBooksGrid(isAdmin)
🔁 Repeat every 5s ∞

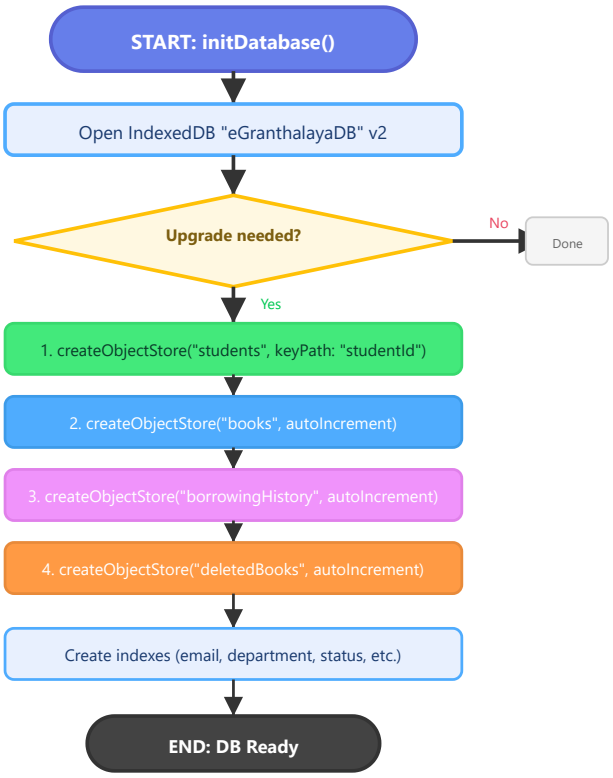
3.4 loadBorrowingRecords() — Table Rendering Loop

File: js/app.js | **Loop Type:** forEach + sort



3.5 IndexedDB Schema Creation Loop

File: js/database.js | **Loop Type:** Sequential store creation in onupgradeneeded


START: initDatabase()
Open IndexedDB "eGranthalayaDB" v2
Upgrade needed?
No
Done
Yes
1. createObjectStore("students", keyPath: "studentId")
2. createObjectStore("books", autoIncrement)
3. createObjectStore("borrowingHistory", autoIncrement)
4. createObjectStore("deletedBooks", autoIncrement)
Create indexes (email, department, status, etc.)
END: DB Ready