BookWorm — Project Analysis Report (Frontend + Backend)

1) Project Overview

BookWorm is a book recommendation + reading tracker app with two roles:

- **User**: discover books, manage personal shelves (Want to Read / Currently Reading / Read), track progress, write reviews + ratings, watch tutorials, see personalized recommendations and reading stats.
- **Admin**: manage users/roles, books, genres, reviews moderation, and YouTube tutorial links.

Core goals:

- Fully protected routes (no public homepage).
- Server-side authentication + authorization.
- Clean, modular code with strong UX for loading/error states.
- Responsive, book-themed “cozy library” design.

---

2) Functional Requirements Breakdown

2.1 Authentication & Authorization

Registration

- Inputs: `name`, `email (unique)`, `photo upload`, `password`.
- Validation:
- Duplicate email blocked.
- Strong password rules (min length, complexity).
- Required fields.
- Data stored securely (hashed password, photo URL).

Login

- Inputs: `email`, `password`.
- Verify credentials + issue tokens.

Route Protection

- Every route requires login.
- If not logged in → redirect to `/login`.

Default Route Behavior

- Normal User → redirect `/` → `/my-library`.
- Admin → redirect `/` → `/admin/dashboard`.

Role-Based Access Control (RBAC)

- Admin-only pages: manage books/genres/users/reviews/tutorials.
- User pages: library/dashboard/browse/details/tutorials.

2.2 User Features

A) Browse Books

- Book list with:
- Search: title, author
- Filters: genre (multi-select), rating range
- Sort: rating, most shelved
- Pagination or infinite scroll
- SSR/SSG using **Next.js App Router** (preferred) for good SEO/performance.

B) Book Details

- Show full details: cover, title, author, genre, description, community rating.
- Actions:
- Add to shelf: Want to Read / Currently Reading / Read
- Progress tracking (when Currently Reading)
- Write review + rating (1–5)

C) My Library (Reading Tracker)

- Shelves:
- Want to Read
- Currently Reading (with progress)
- Read
- Progress model options:
- `pagesRead / totalPages` OR `percent`
- UX:
- Inline progress update
- Quick move between shelves

D) Reviews & Ratings

- User submits review → starts as **pending**.
- Approved reviews are visible publicly on Book Details.

E) User Dashboard/Home

- Reading stats overview:
- Books read this year
- Total pages read
- Average rating given
- Favorite genre breakdown
- Reading streak
- Recommendations block (12–18 books grid/carousel)

F) Tutorials Page

- 10–12 embedded YouTube videos.
- Same page for User & Admin viewing; admin can manage.

G) Reading Challenge / Goals

- User sets annual goal: e.g., “Read 50 books in 2026”.
- Track goal completion with circular progress.

2.3 Admin Features

Admin Dashboard

- Overview cards:
- total users
- total books
- pending reviews
- Charts (Chart.js/Recharts):
- books per genre
- monthly books read (aggregate)
- pages over time (optional)

Manage Books

- CRUD:
- Create: title, author, genre(select existing), description, cover upload → store URL
- Read: table/list with thumbnails + actions
- Update: all fields editable
- Delete: confirmation modal

Manage Genres

- Add/edit genres.
- Books must reference a genre.

Manage Users

- View users list.
- Change roles: Admin ↔ Normal User.

Moderate Reviews

- Pending reviews list.
- Approve or delete.

Manage Tutorials

- Add/remove YouTube links.
- Validate URL and store `videoId`.

---

3) Recommendation System (Simple but Explainable)

3.1 Inputs

- **User's Read shelf** genres frequency.
- **User's average ratings** by genre and overall.
- **Community signals**:
- Books with high approved review ratings
- Most shelved books

3.2 Algorithm (Practical Approach)

1. Compute top genres from user's **Read** shelf.
2. Create candidate pool:
3. Books in top genres not already in user shelves.
4. Add popular books by community rating & most shelved. 3. ## Score candidates:
5. genre match weight -
    - community average rating -
    - approved reviews count/quality
6. Pick 12–18 results.

3.3 Fallback Rule

- If user has **< 3** books in Read:
- Show a mix of popular + random + trending.

3.4 "Why this book?" Tooltip

- Store a short explanation string per recommendation, e.g.
- "Matches your Mystery preference (4 books read) + high-rated reviews."

---

4) UX / UI Design Plan

Design vibe: cozy library, warm colors, paper textures, soft shadows, rounded cards.

Pages & Layout

- **Navbar**: logo + links by role.
- **Footer**: socials, links, copyright.
- Use responsive grid + skeleton loaders.

UX requirements

- Clear empty states:
- No books found
- Shelf empty
- No tutorials yet
- Loading states:
- list skeleton
- button spinner

• Error states:
• toast + inline error messages

---

5) Technical Architecture

5.1 Frontend (Next.js 14+ App Router)

Core stack

• Next.js App Router
• TypeScript
• UI: Tailwind + shadcn/ui (or similar)
• Auth: cookie-based access token + refresh flow OR Next middleware guard
• Data fetching: Server Components for lists, Client Components for interactions

Frontend modules

• `app/(auth)/login`, `app/(auth)/register`
• `app/(user)/dashboard`, `app/(user)/browse`, `app/(user)/my-library`, `app/(user)/books/[id]`, `app/(user)/tutorials`
• `app/(admin)/admin/dashboard`, `.../books`, `.../genres`, `.../users`, `.../reviews`, `.../tutorials`

Route guarding

• Middleware checks token and role.
• Redirect rules:
• `/` → role-based redirect
• non-auth user → `/login`

Image optimization

• Use `next/image` for cover + profile photos.

Charts

• Recharts/Chart.js in dashboard.

5.2 Backend (Node.js + Express + MongoDB)

Core stack

• Node.js + Express
• TypeScript
• MongoDB + Mongoose
• Auth: JWT access + refresh tokens, bcrypt password hash
• Validation: Zod/Joi
• Uploads: Cloudinary (recommended) or local

Backend Modules (Modular MVC)

- `auth` (register/login/refresh/logout)
- `user` (profile, role update by admin)
- `book` (CRUD)
- `genre` (CRUD)
- `review` (create pending, approve/delete)
- `tutorial` (CRUD)
- `library` (shelves, progress updates)
- `stats` (dashboard + analytics)
- (optional) `recommendation` (computed endpoint)

---

6) Data Model / Database Schema (MongoDB)

6.1 User

- `_id`
- `name`
- `email` (unique)
- `passwordHash`
- `photoUrl`
- `role`: `Admin | User`
- `createdAt`, `updatedAt`

6.2 Genre

- `_id`
- `name` (unique)
- `slug` (unique)

6.3 Book

- `_id`
- `title`
- `author`
- `genreId` (ref Genre)
- `description`
- `coverImageUrl`
- `totalPages` (optional but helpful for progress)
- `avgRating` (derived)
- `approvedReviewCount` (derived)
- `shelvedCount` (derived)

6.4 Review

- `_id`
- `bookId` (ref Book)
- `userId` (ref User)
- `rating` (1–5)

- `text`
- `status`: `pending | approved`
- `createdAt`

6.5 LibraryItem (User Shelves)

- `_id`
- `userId` (ref User)
- `bookId` (ref Book)
- `shelf`: `want | reading | read`
- `progressPercent` OR `pagesRead`
- `startedAt`, `finishedAt`
- Unique index: `(userId, bookId)` to prevent duplicates

6.6 ReadingGoal

- `_id`
- `userId`
- `year` (e.g., 2026)
- `goalBooks` (e.g., 50)

6.7 Tutorial

- `_id`
- `title`
- `youtubeUrl`
- `youtubeVideoId`
- `createdBy` (admin userId)

---

7) API Design (Sample Endpoints)

Prefix: `/api/v1`

Auth

- `POST /auth/register`
- `POST /auth/login`
- `POST /auth/refresh`
- `POST /auth/logout`
- `GET /auth/me`

Books

- `GET /books` (search, filter, sort, pagination)
- `POST /books` (admin)
- `GET /books/:id`
- `PATCH /books/:id` (admin)
- `DELETE /books/:id` (admin)

Genres

- `GET /genres`
- `POST /genres` (admin)
- `PATCH /genres/:id` (admin)
- `DELETE /genres/:id` (admin)

Reviews

- `POST /reviews` (user) → pending
- `GET /reviews/pending` (admin)
- `PATCH /reviews/:id/approve` (admin)
- `DELETE /reviews/:id` (admin)
- `GET /books/:id/reviews` (approved only)

Library

- `GET /library` (my shelves)
- `POST /library` (add/move shelf)
- `PATCH /library/:bookId/progress` (update reading progress)

Tutorials

- `GET /tutorials`
- `POST /tutorials` (admin)
- `DELETE /tutorials/:id` (admin)

Stats + Recommendations

- `GET /stats/user` (user dashboard stats)
- `GET /stats/admin` (admin overview)
- `GET /recommendations` (personalized)

---

8) Error Handling & Edge Cases

Auth

- Wrong credentials
- Token expired (refresh flow)
- Forbidden role access

Books

- Invalid book ID
- Genre not found
- Empty cover image

Reviews

- Prevent multiple reviews per user per book (optional rule)

- Pending reviews hidden from public

Library

- Prevent duplicate shelf items
- Progress can't exceed 100% or totalPages
- If shelf becomes `read`, auto set progress 100%

Uploads

- Validate file size + type
- Cloud upload failure fallback

---

9) Deployment Plan

Frontend: Vercel

- Configure env vars: `NEXT_PUBLIC_API_URL`, etc.
- Ensure middleware works on production.

Backend: Render/Railway/VPS

- Env vars: DB URI, JWT secrets, Cloudinary keys
- CORS allow Vercel domain
- Production cookie flags

Key checks

- No console errors
- Auth works on live
- API health endpoint works

---

10) Suggested Work Plan (High-Level)

1. Backend auth + roles + core models
2. Books/Genres CRUD
3. Reviews pending + moderation
4. Library shelves + progress
5. Recommendations + stats endpoints
6. Frontend routes + middleware guards
7. UI pages + responsive layout
8. Deploy + fix edge cases

---

11) Summary

This project demonstrates full-stack skills: secure auth + RBAC, modular backend, SSR-ready Next.js frontend, advanced UX, data modeling, and a practical recommendation engine. The system is designed for real-world reliability: protected routes, clean code, and deploy-ready production behavior.