Contents

[**LIST OF FIGURES** 3](#_Toc195143420)

[**LIST OF TABLES** 4](#_Toc195143421)

[**ABBREIVATIONS** 5](#_Toc195143422)

[1 INTRODUCTION 7](#_Toc195143423)

[1.1 Introduction 7](#_Toc195143424)

[1.2 Literature survey 7](#_Toc195143425)

[1.3 Project Background 8](#_Toc195143426)

[1.4 Objective 8](#_Toc195143427)

[1.5 Project description 9](#_Toc195143428)

[2 SOFTWARE REQUIREMENT SPECIFICATION 11](#_Toc195143429)

[2.1 Requirement Analysis 11](#_Toc195143430)

[2.2 Problem Statement 11](#_Toc195143431)

[2.3 Functional Requirements 11](#_Toc195143432)

[2.3.1 User Access: 11](#_Toc195143433)

[2.3.2 Data Management: 11](#_Toc195143434)

[2.3.3 Data Retrieval: 11](#_Toc195143435)

[2.3.4 Admin Functions: 12](#_Toc195143436)

[2.4 Software Requirement Specification 12](#_Toc195143437)

[2.4.1 Purpose 12](#_Toc195143438)

[2.4.2 Scope of the Project 12](#_Toc195143439)

[2.4.3 Technologies Used 12](#_Toc195143440)

[2.4.4 Overview 13](#_Toc195143441)

[2.5 Software Requirement 13](#_Toc195143442)

[2.6 Hardware Requirements 13](#_Toc195143443)

[2.7 Functional Requirements (Modules) 13](#_Toc195143444)

[2.8 Non-Functional Requirements 14](#_Toc195143445)

[2.8.1 Performance Requirements 14](#_Toc195143446)

[2.8.2 Design Constraints 14](#_Toc195143447)

[2.8.3 Standards Compliance 14](#_Toc195143448)

[2.8.4 Availability 14](#_Toc195143449)

[2.8.5 Portability 15](#_Toc195143450)

[2.9 External Interface Requirements 15](#_Toc195143451)

[2.9.1 User Interface 15](#_Toc195143452)

[2.10 Feasibility Study 15](#_Toc195143453)

[2.10.1 Organisational Feasibility 15](#_Toc195143454)

[2.10.2 Economic Feasibility 16](#_Toc195143455)

[2.10.3 Technical Feasibility 16](#_Toc195143456)

[2.10.4 Behavioural Feasibility 16](#_Toc195143457)

[3 ANALYSIS AND DESIGN 18](#_Toc195143458)

[3.1 Introduction 18](#_Toc195143459)

[3.1.1 Purpose 18](#_Toc195143460)

[3.1.2 Scope 18](#_Toc195143461)

[3.2 System Overview 18](#_Toc195143462)

[3.3 System Architecture 18](#_Toc195143463)

[3.3.1 Architectural Design 18](#_Toc195143464)

[3.3.2 Build and Deploy to a Web App in Cloud 19](#_Toc195143465)

[3.4 Data Design 20](#_Toc195143466)

[3.4.1 Databases 20](#_Toc195143467)

[4 MODELING 23](#_Toc195143468)

[4.1 Design 23](#_Toc195143469)

[4.1.1 Use Case Diagram 23](#_Toc195143470)

[4.1.2 Sequence Diagram 24](#_Toc195143471)

[4.1.3 Activity diagram 25](#_Toc195143472)

[4.1.4 Class Diagram 26](#_Toc195143473)

[4.1.5 Deployment Diagram 27](#_Toc195143474)

[4.1.6 ER Diagram 28](#_Toc195143475)

[5 IMPLEMENTATION 31](#_Toc195143476)

[5.1 Sample code 31](#_Toc195143477)

[5.1.1 Authentication (Login Endpoint - Node.js/Express) 31](#_Toc195143478)

[5.1.2 Book Review Verification Endpoint 31](#_Toc195143479)

[5.1.3 MongoDB Schema (Mongoose) 31](#_Toc195143480)

[5.1.4 Frontend: Book Fetch Component (React + TanStack Query) 32](#_Toc195143481)

[5.1.5 Reading List Entry API 32](#_Toc195143482)

[5.2 Screen Captures 33](#_Toc195143483)

[5.2.1 Home page 33](#_Toc195143484)

[5.2.2 Admin Login Page 33](#_Toc195143485)

[5.2.3 Admin Dashboard 34](#_Toc195143486)

[5.2.4 Users page in Admin Dashboard 34](#_Toc195143487)

[5.2.5 Active Users Page in Admin Dashboard 35](#_Toc195143488)

[5.2.6 Books Management Page in Admin Dash board 35](#_Toc195143489)

[5.2.7 Books reviews Page 36](#_Toc195143490)

[5.2.8 User liked books page in Admin Dasboard 36](#_Toc195143491)

[5.2.9 User Reading List in Admin Dashboard 37](#_Toc195143492)

[5.2.10 User Login Page 37](#_Toc195143493)

[5.2.11 User Registration Page 38](#_Toc195143494)

[5.2.12 User Home Page 38](#_Toc195143495)

[5.2.13 Explore Books Page 39](#_Toc195143496)

[5.2.14 My Reading List Page 39](#_Toc195143497)

[5.2.15 Liked Books Page 40](#_Toc195143498)

[5.2.16 Add new Book Page 40](#_Toc195143499)

[6 TESTING 43](#_Toc195143500)

[6.1 Software Testing 43](#_Toc195143501)

[6.2 Black Box Testing 43](#_Toc195143502)

[6.3 White Box Testing 43](#_Toc195143503)

[6.4 Performance Testing 43](#_Toc195143504)

[6.5 Load Testing 43](#_Toc195143505)

[6.6 Manual Testing 43](#_Toc195143506)

[7 RESULTS AND CHALLENGES 47](#_Toc195143507)

[7.1 Results 47](#_Toc195143508)

[7.2 Challenges 47](#_Toc195143509)

[8 CONCLUSION 49](#_Toc195143510)

[8.1 Conclusions 49](#_Toc195143511)

[8.2 Scope for Future Work 49](#_Toc195143512)

[8.3 Limitations 49](#_Toc195143513)

# **LIST OF FIGURES**

[Figure 1-Web Application Architecture Diagram 21](#_Toc195143514)

[Figure 2-Use Case Diagram for System 25](#_Toc195143515)

[Figure 3-Sequence Diagram for user 26](#_Toc195143516)

[Figure 4-Sequence Diagram for Admin 27](#_Toc195143517)

[Figure 5-Activity diagram for booknest 28](#_Toc195143518)

[Figure 6-Class Diagram 29](#_Toc195143519)

[Figure 7-Deployment Diagram 30](#_Toc195143520)

[Figure 8-ER Diagram 31](#_Toc195143521)

[Figure 9-Booknest Home Page 35](#_Toc195143522)

[Figure 10-Booknest Admin Page 35](#_Toc195143523)

[Figure 11-Booknest Admin Dashboard overview 36](#_Toc195143524)

[Figure 12-User Page in Admin Dashboard 36](#_Toc195143525)

[Figure 13-Active Users Page 37](#_Toc195143526)

[Figure 14-Book Management Page 37](#_Toc195143527)

[Figure 15-Book Reviews in Admin Dashboard 38](#_Toc195143528)

[Figure 16-User liked Books in AdminDashboard 38](#_Toc195143529)

[Figure 17-User Reading List in Admin Dashboard 39](#_Toc195143530)

[Figure 18-User Login Page 39](#_Toc195143531)

[Figure 19-User Registration Page 40](#_Toc195143532)

[Figure 20-User HomePage 40](#_Toc195143533)

[Figure 21-User Liked Books 41](#_Toc195143534)

[Figure 22-My Reading List Page 41](#_Toc195143535)

[Figure 23-My Liked Books Page 42](#_Toc195143536)

[Figure 24- Adding new book 42](#_Toc195143537)

[Figure 25-verification questions 43](#_Toc195143538)

[Figure 26-Books Added successfully 43](#_Toc195143539)

[Figure 24- Adding new book 46](#_Toc195143540)

[Figure 25-verification questions 46](#_Toc195143541)

[Figure 26-Books Added successfully 47](#_Toc195143542)

# **LIST OF TABLES**

# **ABBREIVATIONS**

***CHAPTER - 1***

***INTRODUCTION***

# INTRODUCTION

## Introduction

BookNest is an innovative web-based platform created to enhance the book reading and reviewing experience for literature enthusiasts. This project aims to bridge the gap between casual readers and authentic book reviews by offering a structured environment where users can explore books, write insightful reviews, and maintain personalized reading lists. The system is designed to encourage genuine interactions and improve the quality of content through a unique review verification mechanism.

The platform features a dual-interface system: a user interface for book discovery and review submission, and an admin dashboard for managing books, users, and review content. BookNest is built using the MERN stack with additional tools like Tailwind CSS for styling, Zod for input validation, and TanStack Query for efficient data fetching. The backend ensures secure and role-based authentication using hashed passwords and session management.

What sets BookNest apart is its intelligent verification system that ensures only readers who have genuinely read the book can submit reviews. This is accomplished by requiring users to answer book-specific questions created by admins. Such measures not only maintain the platform’s credibility but also promote deeper user engagement. With future plans for social features, AI-powered recommendations, and mobile support, BookNest is set to evolve into a comprehensive ecosystem for readers worldwide.

## Literature survey

The domain of online book review and discovery platforms has been shaped by prominent systems like Goodreads, Amazon, and LibraryThing, each offering unique functionalities but also revealing significant gaps. Goodreads, a widely used social cataloging platform, allows users to rate books, write reviews, and join reading communities. However, studies such as those by Worrall (2017) highlight its susceptibility to unverified reviews, where users can post opinions without proving engagement with the text, leading to potential bias or spam. Similarly, Amazon’s book review system integrates customer feedback with purchasing data, yet research by Hu et al. (2014) points out the prevalence of fake reviews, often motivated by commercial incentives rather than genuine reading experiences. These platforms lack mechanisms to ensure review authenticity, a critical limitation BookNest aims to address.

Advancements in sentiment analysis and recommender systems have offered partial solutions to filter and enhance review quality. For instance, Zhang et al. (2016) explored machine learning techniques to detect sentiment polarity in reviews, improving the ability to categorize feedback as positive or negative. Meanwhile, collaborative filtering methods, as discussed by Ricci et al. (2011) in "Introduction to Recommender Systems," leverage user behavior to suggest books, a feature prominent in Goodreads and Amazon. However, these approaches focus on post-submission analysis rather than preemptive verification, failing to confirm whether reviewers have read the books. A study by Mudambi and Schuff (2010) further emphasizes that perceived review helpfulness declines when authenticity is questionable, underscoring the need for a system like BookNest’s verification process.

Emerging platforms and research have begun exploring innovative ways to tackle these challenges. For example, BookRiser, a lesser-known platform, introduced gamified reading challenges to encourage engagement, though it lacks a review verification component (Smith, 2020). Academic work by Chen et al. (2018) on blockchain-based review systems suggests decentralized verification, but its complexity limits practical adoption. BookNest draws inspiration from these efforts while diverging with a practical, admin-driven question-based verification system. Built on the MERN stack with tools like Zod and TanStack Query, it integrates modern web development practices with a focus on credibility. Unlike existing solutions, BookNest’s proactive approach—requiring users to answer book-specific questions—ensures authentic reviews, positioning it as a novel contribution to the literary ecosystem.

## Project Background

BookNest was born out of the need to address the unreliable reviews plaguing platforms like Goodreads and Amazon, where unverified feedback often undermines trust and quality. Inspired by the limitations of existing systems and the potential of modern technology, BookNest introduces a verification system requiring users to answer book-specific questions, ensuring authentic reviews. Built on the MERN stack with tools like Tailwind CSS and Zod, it offers a secure, scalable platform with dual interfaces for users and admins, aiming to foster a credible and engaging community for book lovers while laying the groundwork for future innovations like AI recommendations and mobile apps

## Objective

The primary objective of BookNest is to create a credible and engaging book review platform that ensures authentic feedback by requiring users to pass a verification process before submitting reviews, addressing the issue of unreliable content on existing systems. Built on the MERN stack, it aims to provide an intuitive interface for readers to discover books, manage reading lists, and connect with a community, while offering admins robust tools to oversee content. BookNest seeks to enhance the reading experience and establish a foundation for future features like AI-driven recommendations and mobile support.

## Project description

BookNest is an innovative web-based platform designed to revolutionize the book reading and reviewing experience for literature enthusiasts. This project addresses the common issue of unreliable reviews found on existing platforms like Goodreads and Amazon by introducing a structured environment where authenticity is paramount. BookNest empowers users to explore a diverse catalog of books, write insightful reviews, and maintain personalized reading lists, while ensuring that only verified readers contribute feedback. The platform’s core mission is to bridge the gap between casual readers and credible book reviews, fostering a trustworthy community of book lovers through advanced technology and thoughtful design.

The system features a dual-interface architecture: a user-friendly interface for book discovery, review submission, and list management, and a powerful admin dashboard for overseeing books, users, and review content. Built on the MERN stack (MongoDB, Express.js, React.js, Node.js), BookNest integrates modern tools such as Tailwind CSS for responsive styling, Zod for input validation, and TanStack Query for efficient data fetching. Security is a priority, with role-based authentication, hashed passwords, and session management ensuring safe interactions. The standout feature is its intelligent review verification system, requiring users to correctly answer admin-generated, book-specific questions before submitting reviews, guaranteeing genuine engagement and high-quality content.

BookNest goes beyond traditional book platforms by combining technical robustness with a focus on user experience. Its RESTful API enables seamless communication between frontend and backend, supporting functionalities like book browsing, user authentication, and content management. Designed with scalability in mind, the platform lays the groundwork for future enhancements, including social features like user following and book clubs, AI-driven recommendations, and native mobile applications for iOS and Android. By blending innovation with a passion for literature, BookNest aspires to become a comprehensive ecosystem where readers can discover, discuss, and celebrate books with confidence and enthusiasm.

***CHAPTER - 2***

***SOFTWARE REQUIREMENT SPECIFICATION***

# SOFTWARE REQUIREMENT SPECIFICATION

## Requirement Analysis

The proposed web-based data access and retrieval system aims to provide a portable and efficient solution for users to manage and access data seamlessly across widely used operating systems like Windows and Ubuntu. This analysis outlines the functional and non-functional requirements to ensure the system is user-friendly, robust, and meets the goal of easy data handling, supported by necessary documentation such as problem statements, data flow diagrams, use case diagrams, and other UML diagrams.

## Problem Statement

Current online book review platforms, such as Goodreads and Amazon, suffer from a significant flaw: the abundance of unverified and often unreliable reviews, which erodes trust and diminishes the value of user feedback. Without mechanisms to confirm whether reviewers have read the books, these systems are vulnerable to biased, superficial, or even fraudulent reviews, making it challenging for readers to find authentic recommendations. This lack of credibility hampers meaningful engagement within literary communities and leaves readers struggling to navigate a cluttered landscape of opinions. BookNest aims to resolve this issue by developing a web-based platform that implements a unique verification system, requiring users to answer book-specific questions to prove their familiarity with the text before submitting reviews, thereby fostering a trustworthy and engaging environment for book enthusiasts.

## Functional Requirements

### User Access:

* Users must be able to log in using a username and password to access the system securely.
* Support a registration process for new users to create accounts.

### Data Management:

* Enable users to upload, store, and retrieve data (e.g., documents, files) through a web interface.
* Provide a search functionality to locate specific data by keywords, file names, or categories.

### Data Retrieval:

* Allow users to download or view stored data directly from the system.
* Support bulk retrieval of multiple files or datasets in a single operation.

### Admin Functions:

* Admins must have tools to manage user accounts (e.g., activate/deactivate) and monitor system usage.
* Provide options to delete or archive outdated data.

## Software Requirement Specification

BookNest is a web-based platform designed to revolutionize the book review experience by ensuring authentic feedback through a unique verification system, requiring users to answer book-specific questions before submitting reviews. Built on the MERN stack (MongoDB, Express.js, React.js, Node.js) with tools like Tailwind CSS, Zod, and TanStack Query, it offers a dual-interface application: a user interface for browsing books, submitting verified reviews, and managing reading lists, and an admin dashboard for overseeing content and analytics. The system aims to provide a scalable, secure, and user-friendly solution for book enthusiasts, addressing the issue of unreliable reviews on existing platforms, with future plans for social features and mobile support.

### Purpose

The purpose of BookNest is to create a web-based platform that enhances the book reviewing experience by ensuring the authenticity of user reviews through a unique verification system. It aims to provide a reliable space for book enthusiasts to discover books, submit credible feedback, and manage reading lists, addressing the prevalent issue of unverified reviews on platforms like Goodreads and Amazon, thereby fostering a trustworthy literary community.

### Scope of the Project

BookNest will deliver a dual-interface web application: a user interface for browsing books, submitting verified reviews, and managing reading lists, and an admin interface for managing books, users, and verification questions. The initial scope includes user authentication, a searchable book catalog, a review verification process, and admin analytics. Future expansions may incorporate social features, AI-driven recommendations, and mobile applications. The system targets desktop users on major browsers, with plans for scalability and broader accessibility.

### Technologies Used

* **Frontend:** React.js with TypeScript (UI development), Tailwind CSS (styling), TanStack Query (data fetching), React Hook Form (form management).
* **Backend:** Node.js with Express.js (server-side logic and RESTful API).
* **Database:** MongoDB (data storage for users, books, and reviews).
* **Additional Tools:** Zod (validation), bcrypt (password security), session middleware (authentication).

### Overview

BookNest is a web-based application developed to provide a reliable and engaging platform for book enthusiasts, overcoming the challenge of unverified reviews found in existing systems like Goodreads and Amazon through a unique verification mechanism that requires users to answer book-specific questions before submitting feedback. Leveraging the MERN stack (MongoDB, Express.js, React.js, Node.js) and enhanced by technologies such as Tailwind CSS, Zod for validation, and TanStack Query for data management, it offers a dual-interface system: one for users to browse books, submit verified reviews, and manage reading lists, and another for administrators to oversee books, users, and analytics. This specification outlines a secure, scalable solution that ensures authenticity and usability, laying the groundwork for future features like social integration and mobile support.

## Software Requirement

BookNest requires a robust software framework to deliver a secure and efficient book review platform, utilizing the MERN stack (MongoDB for data storage, Express.js and Node.js for backend logic, and React.js with TypeScript for a dynamic frontend) to ensure scalability and performance. The system must support user authentication with bcrypt-encrypted passwords, a RESTful API for seamless data interaction (e.g., /api/books, /api/auth), and a verification module requiring users to answer book-specific questions before submitting reviews. Additional requirements include Tailwind CSS for responsive styling, Zod for input validation, TanStack Query for optimized data fetching, and compatibility with Windows and Ubuntu browsers, providing a reliable, user-friendly experience for browsing books, managing reading lists, and administering content.

## Hardware Requirements

* 2GB RAM minimum
* 1GB storage space
* Internet connectivity

## Functional Requirements (Modules)

* **User Authentication Module:** Facilitates user registration and login using email and password, with role-based access (user/admin) tracked via secure sessions and bcrypt-encrypted passwords.
* **Book Discovery Module:** Enables users to browse and search a catalog of books by title, author, category, or ISBN, accessible publicly without login, with results displayed in a responsive interface.
* **Review Verification Module:** Requires users to answer three admin-generated, book-specific questions (each with four options) to verify reading before submitting reviews, storing verification status in sessions.
* **Review Management Module:** Allows verified users to write, submit, and view reviews for books, with options to read others’ feedback, integrated with a rating system.
* **Reading List Module:** Provides users the ability to create, edit, and delete personalized reading lists, with options to add or remove books and track reading progress.
* **Admin Dashboard Module:** Equips administrators with tools to add, edit, and delete books (including metadata like title, author, and cover image), manage verification questions, oversee user accounts, and access analytics on platform usage.

## Non-Functional Requirements

### Performance Requirements

BookNest must deliver efficient performance to ensure a smooth user experience. The system should achieve page load times of under 2 seconds for browsing books and retrieving reviews, even under moderate user traffic. Database queries, powered by MongoDB, must be optimized to handle concurrent requests efficiently, supporting at least 100 simultaneous users without degradation. API response times for critical endpoints (e.g., /api/books, /api/auth/login) should not exceed 500 milliseconds, ensuring quick access to data and seamless navigation.

### Design Constraints

The platform must be developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) to leverage its full-stack JavaScript ecosystem, limiting the choice of alternative frameworks. The frontend must utilize Tailwind CSS for styling and React.js with TypeScript for type safety, restricting deviations to other styling or scripting languages. The system is constrained to a web-based architecture initially, with server-side rendering avoided to maintain simplicity, and must integrate Zod for validation and TanStack Query for data fetching, aligning with predefined technical choices.

### Standards Compliance

BookNest must adhere to web development standards, including W3C guidelines for HTML5 and CSS3 to ensure accessibility and cross-browser compatibility. RESTful API design must follow HTTP protocol standards (e.g., proper use of status codes like 200, 401, 404). Security practices should comply with OWASP Top 10 recommendations, incorporating HTTPS, secure session management, and CSRF protection. Data storage in MongoDB must align with JSON schema standards for consistency and interoperability.

### Availability

The system should target 99% uptime, ensuring BookNest remains accessible to users with minimal disruptions. Scheduled maintenance windows, if required, must be limited to off-peak hours (e.g., 2:00 AM–4:00 AM UTC) and communicated in advance. The platform should implement error handling and logging to quickly recover from failures, with a maximum downtime of 1 hour per month, excluding planned updates, to maintain reliability for its user base.

### Portability

BookNest must be portable across Windows and Ubuntu operating systems, supporting major browsers (Chrome, Firefox, Edge, Safari) without requiring platform-specific installations. The web-based nature eliminates dependency on native apps initially, ensuring consistent functionality via standard web technologies. The system should be deployable on common cloud platforms (e.g., AWS, Heroku) with minimal configuration changes, facilitating easy migration or scaling across hosting environments.

## External Interface Requirements

### User Interface

The user interface of BookNest includes:

* **Book Browsing and Search**: Users can explore books by categories, keywords, or popularity.
* **Review Reading and Writing**: Only users who pass book-related verification questions can post reviews.
* **Reading List Management**: Users can create and manage a personal list of books they intend to read or have read.
* **Book Recommendations**: Personalized suggestions based on reading habits and preferences.
* **Engaging Layout**: Implemented using React.js with Tailwind CSS for a modern, responsive design.
* **Frontend Libraries Used**: Shadcn UI, TanStack Query, React Hook Form, and Zod for enhanced UX and form validation.

Admin interface includes:

* **Book Management**: Add, update, or delete books.
* **User Management**: Monitor and control user activities and roles.
* **Verification Question Management**: Admins can add or remove questions for book review verification.
* **Analytics Dashboard**: Visual stats on reading trends and user engagement.

## Feasibility Study

### Organisational Feasibility

BookNest aligns with organizational goals by promoting genuine reader engagement, improving content quality through review verification, and offering scalable admin controls. It supports a structured content moderation process, efficient user management, and can be integrated into educational or literary institutions aiming to digitize book reviews.

### Economic Feasibility

The project utilizes open-source technologies (React.js, Node.js, MongoDB), reducing licensing costs. Initial development and deployment can be managed by a small team. With modular architecture and cloud deployment options, maintenance and scaling are economically viable. Revenue opportunities exist via ads, premium features, and partnerships.

### Technical Feasibility

The tech stack (React.js, Express.js, MongoDB) is mature and well-supported. Features like authentication, session handling, and form validation are implemented with industry-standard tools. The component-based frontend and RESTful API backend ensure scalability, ease of integration, and maintainability.

### Behavioural Feasibility

User-friendly interfaces and intuitive workflows promote user adoption. The verification system encourages only genuine readers to contribute, improving trust in reviews. Admin tools are straightforward, supporting ease of use for non-technical staff. Overall, the system fosters positive behavior from both users and administrators.

***CHAPTER - 3***

***ANALYSIS & DESIGN***

# ANALYSIS AND DESIGN

## Introduction

### Purpose

#### Document Purpose

The purpose of this document is to provide a detailed description of the BookNest system, outlining its features, architecture, interfaces, and functional specifications. It serves as a reference for developers, testers, and stakeholders throughout the development lifecycle.

#### Project Purpose

The purpose of the BookNest project is to develop a secure, interactive, and intelligent book review platform that ensures authentic reviews through a verification system, supports personalized reading experiences, and provides administrative tools for efficient content and user management.

### Scope

#### Document Scope

This document covers the system requirements, architectural design, technical specifications, user and admin functionalities, database schema, API details, and feasibility aspects of the BookNest platform.

#### Project Scope

BookNest includes modules for user registration, book browsing, review submission (post-verification), reading list management, book recommendations, admin dashboards for content control, and secure authentication. Future extensions include social features, mobile apps, and advanced recommendation engines.

## System Overview

BookNest is a web-based platform designed for book discovery and authentic review sharing. It supports two main user roles: **Users** and **Admins**. Users can browse books, manage personal reading lists, and write verified reviews, while Admins can manage books, users, and review content through a dedicated dashboard.

The system architecture follows a modular full-stack approach:

* **Frontend**: Built with React.js and Tailwind CSS, offering a responsive, intuitive user interface.
* **Backend**: Powered by Node.js with Express, handling authentication, review verification, and data management.
* **Database**: MongoDB stores users, books, reviews, and verification questions.
* **Security**: Role-based access control, session-based authentication, input validation, and CSRF protection ensure robust security.

## System Architecture

### Architectural Design

BookNest follows a **three-tier architecture** comprising:

1. **Presentation Layer (Frontend)**
   * **Technology**: React.js with TypeScript and Tailwind CSS
   * **Role**: Provides a responsive user interface for users and admins.
   * **Functionality**: Book browsing, review submission, reading list management, admin dashboard access.
2. **Application Layer (Backend)**
   * **Technology**: Node.js with Express
   * **Role**: Handles business logic, API routing, session management, and role-based access control.
   * **Functionality**: Authentication, review verification, CRUD operations for books/users/questions.
3. **Data Layer (Database)**
   * **Technology**: MongoDB
   * **Role**: Stores user data, books, reviews, reading lists, and verification questions.
   * **Functionality**: Efficient data management and relationships via collections and schemas.

**Interaction Flow**:

* The frontend communicates with the backend through RESTful APIs.
* The backend validates and processes requests, interacts with the database, and returns responses.
* Secure sessions and validation layers protect data flow and user interactions.

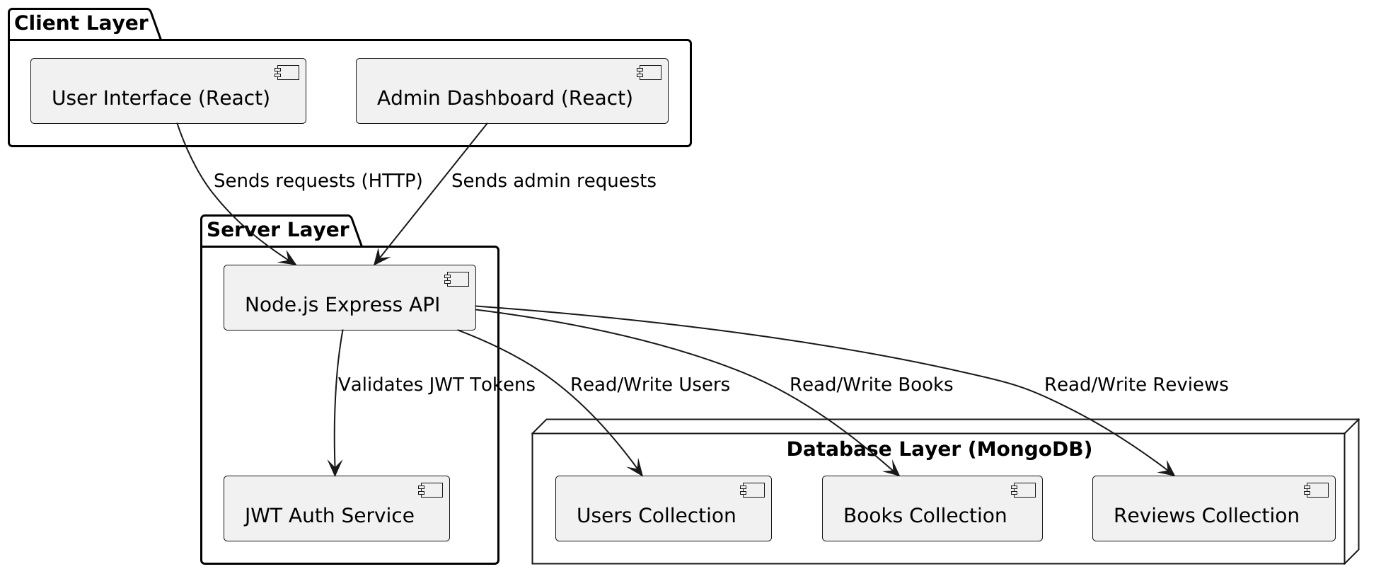


Figure 1-Web Application Architecture Diagram

### Build and Deploy to a Web App in Cloud

* To build and deploy **BookNest** as a web application in the cloud, the following steps are followed:

**1. Build Process**

* **Frontend (React.js)**
* Run: npm run build to generate optimized static files (/dist or /build folder).
* **Backend (Node.js + Express)**
* Ensure all routes and controllers are functional.
* Include environment configurations using .env file.
* Test the server with Postman or browser.

**2. Cloud Deployment Stack**

* **Cloud Provider**: [Render](https://render.com), **Vercel**, **Netlify**, or **Heroku** (Frontend)
* **Backend Host**: Render / Railway / DigitalOcean App Platform
* **Database**: MongoDB Atlas (Cloud-hosted NoSQL database)

**3. Deployment Steps**

* **Frontend**
* Connect the frontend GitHub repo to Vercel or Netlify.
* Set build command (npm run build) and output directory (build).
* Deploy directly via Git push or through the dashboard.
* **Backend**
* Push backend code to GitHub.
* Deploy using Render or Railway:
* Select "Web Service"
* Set build command: npm install
* Start command: node server.js or npm start
* Add environment variables from env
* **MongoDB Atlas**
* Create a cluster on [MongoDB Atlas](https://www.mongodb.com/cloud/atlas).
* Whitelist IP and get connection string.
* Use this URI in the backend .env file.

**4. Final Configuration**

* Ensure CORS is properly configured on backend to allow frontend requests.
* Use HTTPS for secure deployment.
* Enable auto-deployment on Git push for CI/CD.

## Data Design

### Databases

BookNest uses **MongoDB Atlas** as its cloud-based NoSQL database solution. The database design includes several key collections to support system functionality:

**1. users Collection**

* Stores user account details and roles.
* Fields:
  + id: Unique identifier
  + name: Full name
  + email: Unique email
  + password: Hashed password
  + isAdmin: Boolean for admin role
  + isActive: Boolean for active status
  + joinedAt: Timestamp of registration

**2. books Collection**

* Stores all book-related information.
* Fields:
  + id: Unique identifier
  + title: Book title
  + author: Author name
  + publisher: Publisher name
  + publishedDate: Date of publication
  + description: Book summary
  + category: Genre/category
  + isbn: ISBN number
  + coverImage: Image URL
  + addedAt: Timestamp
  + averageRating: Calculated rating
  + totalReviews: Count of reviews

**3. reviews Collection**

* Stores reviews submitted by users.
* Fields:
  + id: Unique identifier
  + bookId: Linked book reference
  + userId: Reviewer reference
  + content: Review text
  + rating: Star rating
  + createdAt: Timestamp

**4. verification\_questions Collection**

* Stores multiple-choice questions used for book verification.
* Fields:
  + id: Unique identifier
  + bookId: Linked book reference
  + question: Question text
  + options: Array of four options
  + correctOptionIndex: Index of correct answer

**5. reading\_lists Collection**

* Stores personalized book tracking for users.
* Fields:
  + id: Unique identifier
  + userId: Linked user reference
  + bookId: Book in list
  + status: Enum (e.g., “To Read”, “Reading”, “Completed”)

***CHAPTER - 4***

***MODELING***

# MODELING

## Design

Requirements gathering followed by careful analysis leads to a systematic Object Oriented Design (OOAD). Various activities have been identified and are represented using Unified Modeling Language (UML) diagrams. UML is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development

### Use Case Diagram

the Unified Modeling Language (UML), the use case diagram is a type of behavioral diagram defined by and created from a use-case analysis. It represents a graphical over view of the functionality of the system in terms of actors, which are persons, organizations or external system that plays a role in one or more interaction with the system. These are drawn as stick figures. The goals of these actors are represented as use cases, which describe a sequence of actions that provide something of measurable value to an actor and any dependencies between those use cases.

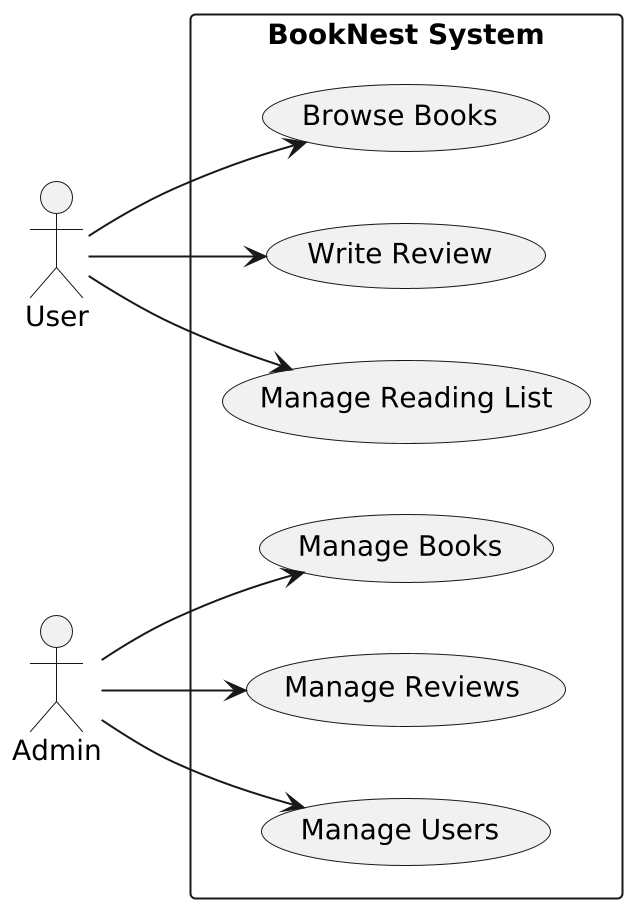


Figure 2-Use Case Diagram for System

### Sequence Diagram

A **Sequence Diagram** in UML represents the interaction between system components in a time-sequenced manner. It shows how objects communicate with each other through messages over time to carry out a specific functionality.

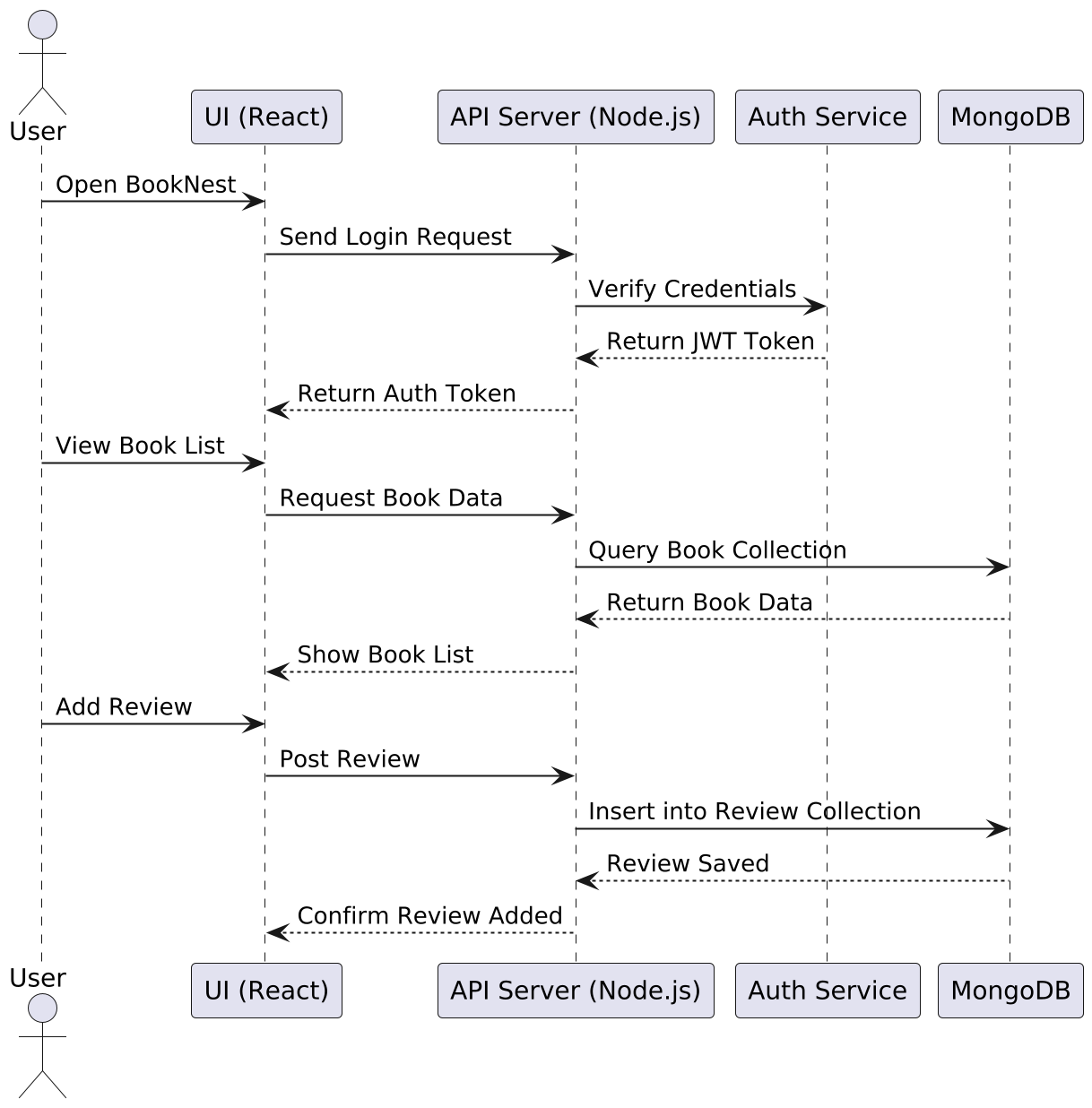


Figure 3-Sequence Diagram for user

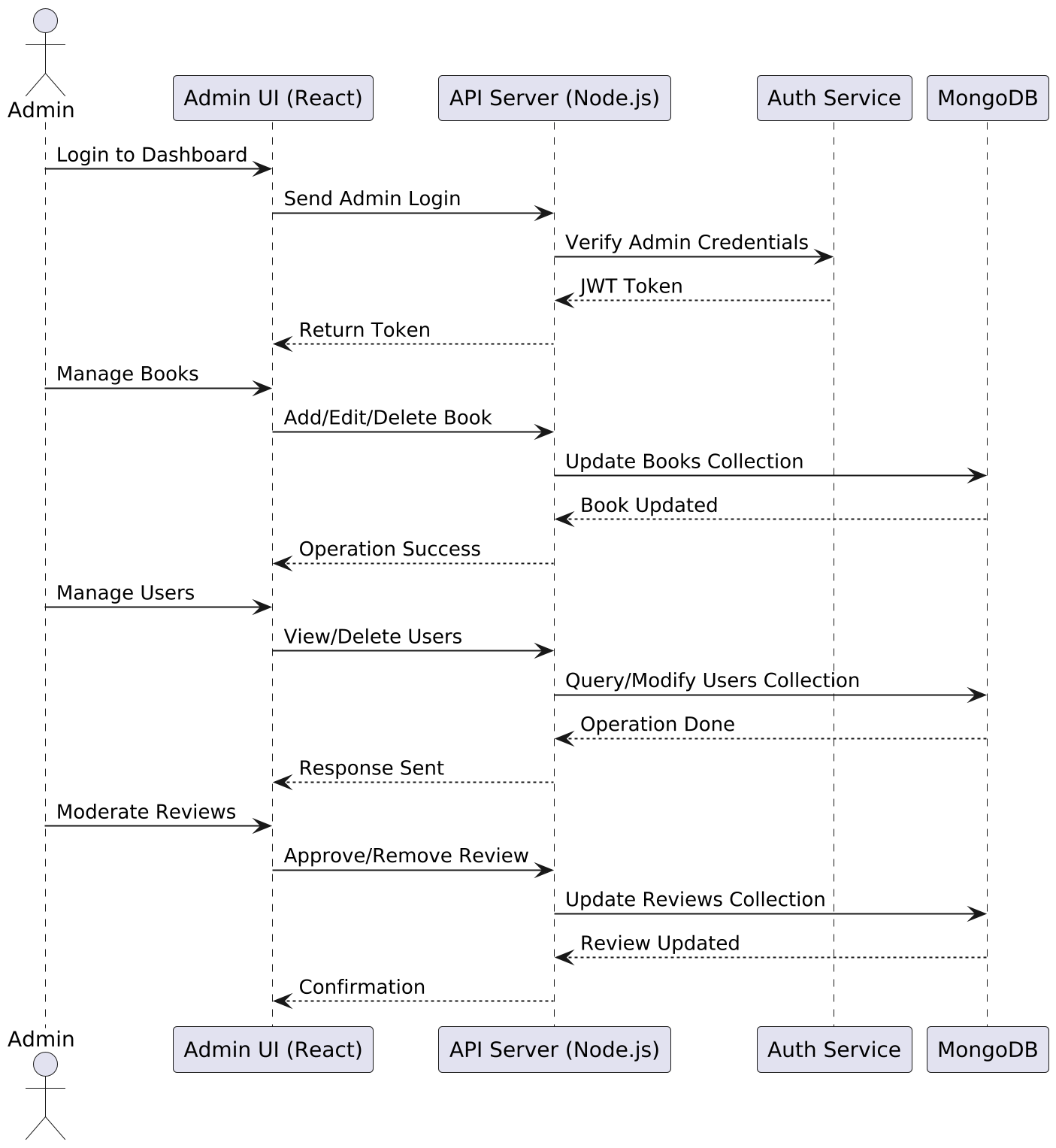


Figure 4-Sequence Diagram for Admin

### Activity diagram

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc. Activity is a particular operation of the system.

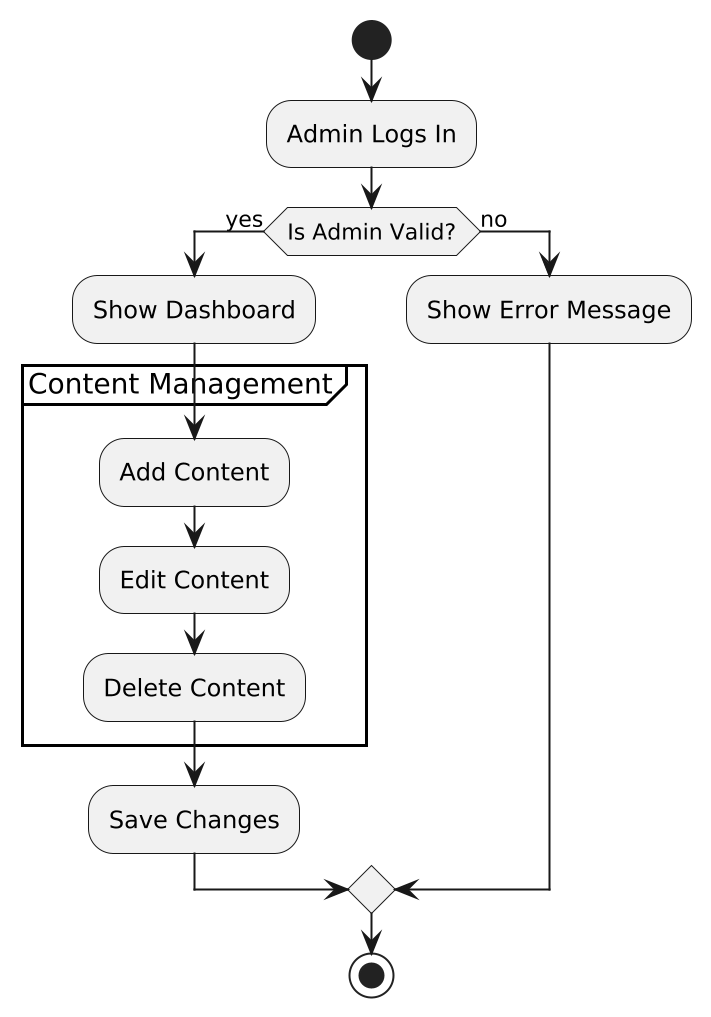


Figure 5-Activity diagram for booknest

### Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system’s classes, their attributes, operations (or methods), and the relationships among the classes.

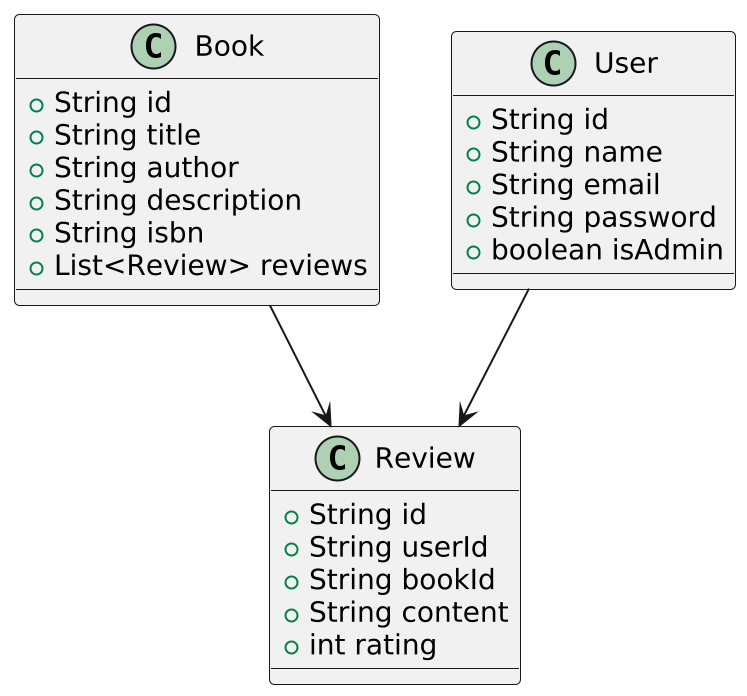


Figure 6-Class Diagram

### Deployment Diagram

Deployment diagram shows execution architecture of systems that represent the assignment (deployment) of software artifacts to deployment targets (usually nodes). Nodes represent either hardware devices or software execution environments. They could be connected through communication paths to create network systems of arbitrary complexity. Artifacts represent concrete elements in the physical world that are the result of a development process and are deployed on nodes.

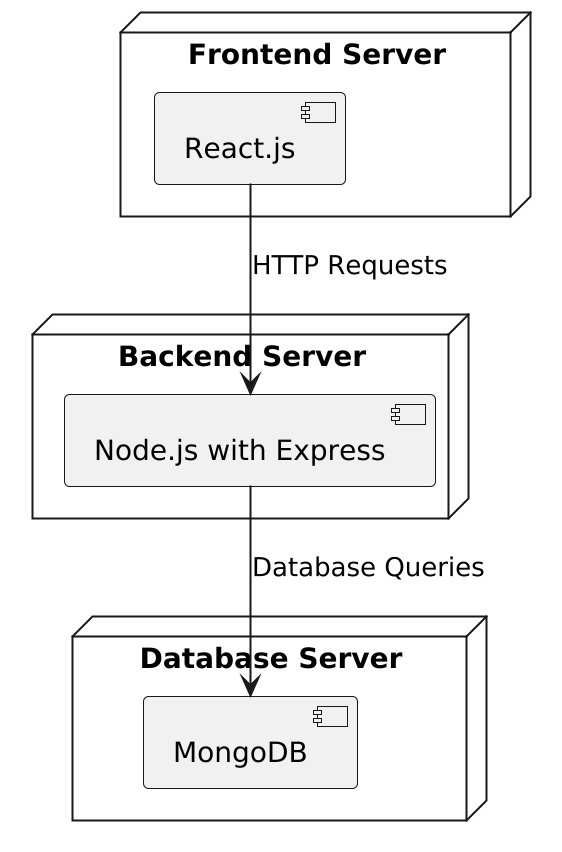


Figure 7-Deployment Diagram

### ER Diagram

An **Entity-Relationship Diagram (ERD)** represents the logical structure of the database. It identifies the entities involved in the system, their attributes, and the relationships among them. ERDs help in visualizing how data is interconnected and how it flows within the system.

In the BookNest system, key entities include **User**, **Book**, **Review**, **VerificationQuestion**, and **ReadingList**. Relationships are established between these entities to support functionalities like book review, reading list management, and review verification.

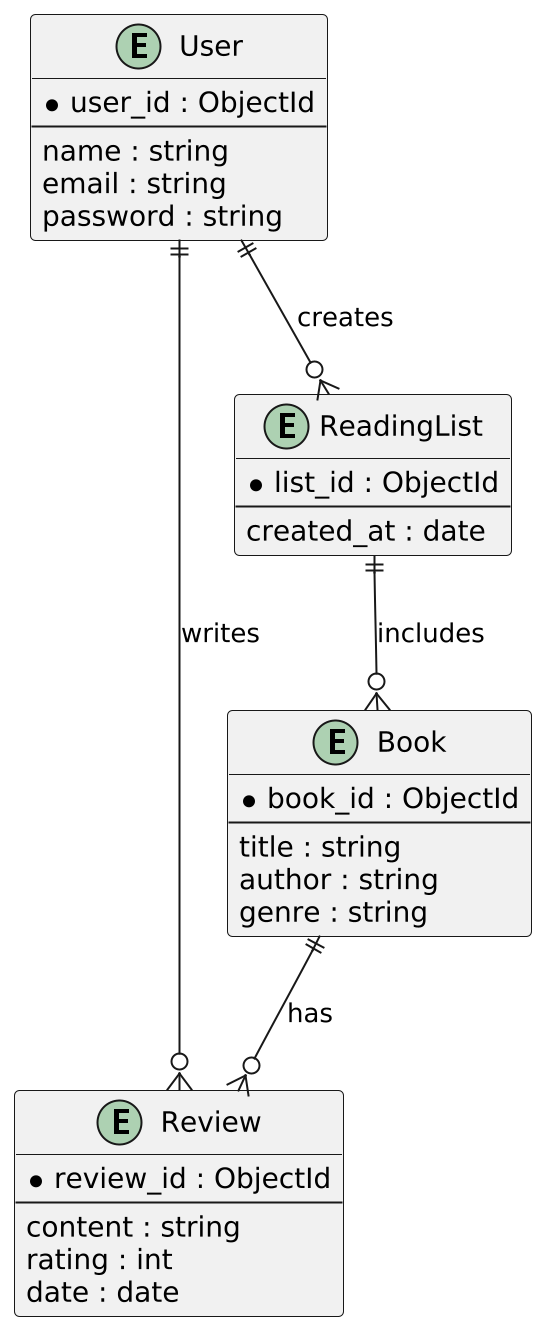


Figure 8-ER Diagram

***CHAPTER - 5***

***IMPLEMENTATION***

# IMPLEMENTATION

## Sample code

### Authentication (Login Endpoint - Node.js/Express)

app.post("/api/auth/login", async (req, res) => {

try {

const { email, password } = req.body;

const user = await User.findOne({ email });

if (!user || !(await bcrypt.compare(password, user.password))) {

return res.status(401).json({ message: "Invalid credentials" });

}

req.session.userId = user.\_id;

req.session.isAdmin = user.isAdmin;

res.status(200).json({ id: user.\_id, email: user.email, isAdmin: user.isAdmin });

} catch (err) {

res.status(500).json({ message: "Login failed" });

}

});

### Book Review Verification Endpoint

app.post("/api/books/:id/verify", requireAuth, async (req, res) => {

try {

const bookId = req.params.id;

const { answers } = req.body;

const questions = await VerificationQuestion.find({ bookId });

const allCorrect = questions.every((q, i) => answers[i] === q.correctOptionIndex);

req.session.isVerified = req.session.isVerified || {};

req.session.isVerified[bookId] = allCorrect;

res.status(200).json({ verified: allCorrect });

} catch (err) {

res.status(500).json({ message: "Verification failed" });

}

});

### MongoDB Schema (Mongoose)

#### User Schema

**const userSchema = new mongoose.Schema({**

**name: String,**

**email: { type: String, unique: true },**

**password: String,**

**isAdmin: { type: Boolean, default: false },**

**joinedAt: { type: Date, default: Date.now }**

**});**

#### Book Schema

const bookSchema = new mongoose.Schema({

title: String,

author: String,

category: String,

description: String,

isbn: String,

coverImage: String,

publishedDate: String

});

#### Review Schema

const reviewSchema = new mongoose.Schema({

bookId: mongoose.Schema.Types.ObjectId,

userId: mongoose.Schema.Types.ObjectId,

rating: Number,

content: String,

createdAt: { type: Date, default: Date.now }

});

### Frontend: Book Fetch Component (React + TanStack Query)

const Books: React.FC = () => {

const { data: books, isLoading } = useQuery(['books'], async () => {

const res = await fetch('/api/books');

return res.json();

});

if (isLoading) return <div>Loading...</div>;

return (

<div>

{books.map((book) => (

<div key={book.\_id}>

<h2>{book.title}</h2>

<p>{book.author}</p>

</div>

))}

</div>

);

};

### Reading List Entry API

app.post("/api/reading-list", requireAuth, async (req, res) => {

const { bookId, status } = req.body;

const entry = new ReadingList({ userId: req.session.userId, bookId, status });

await entry.save();

res.status(201).json(entry);

});

## Screen Captures

### Home page

Home page has two options one is Admin Access and another is user access.

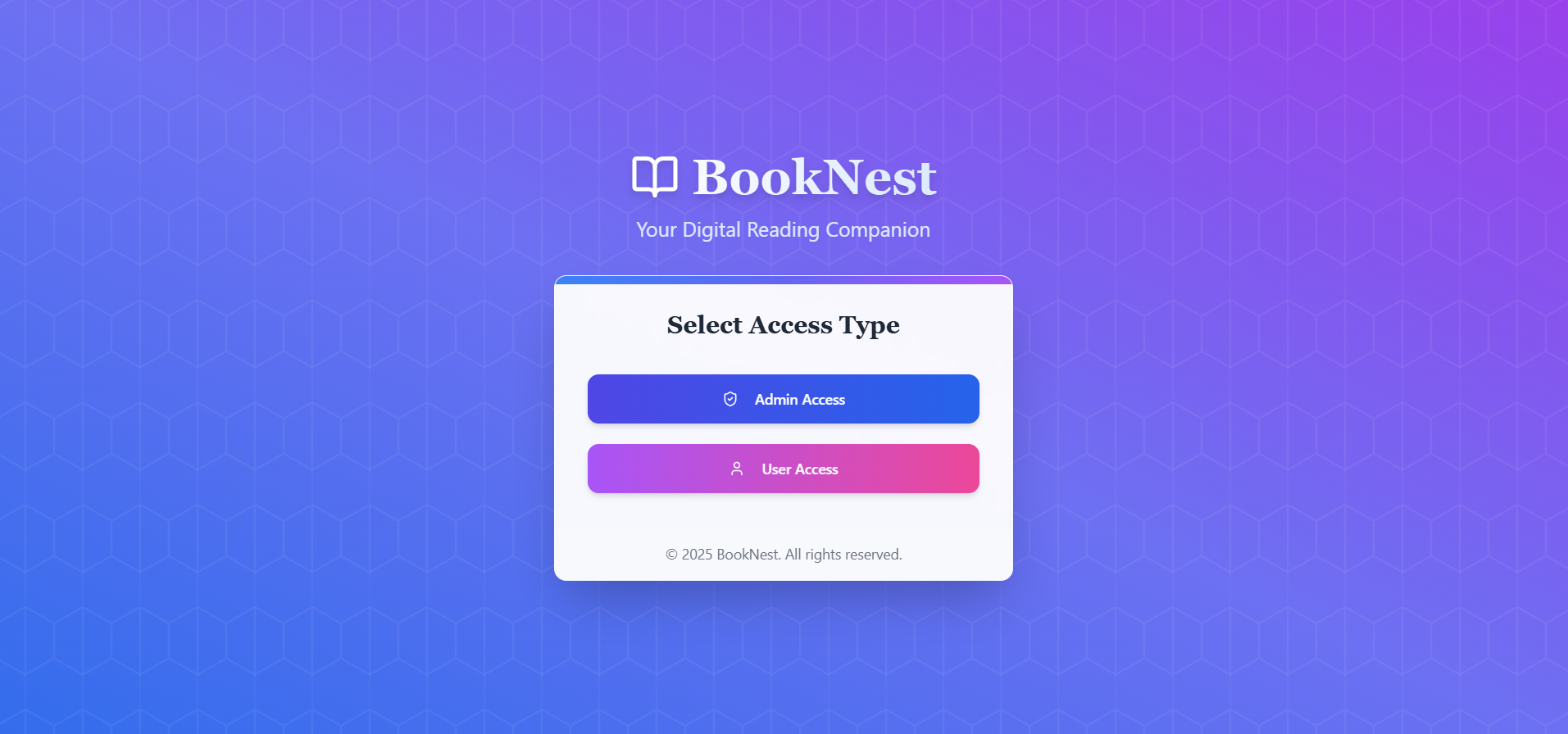


Figure 9-Booknest Home Page

### Admin Login Page

Here using admin credentials admin can login to Admin Dashboard.

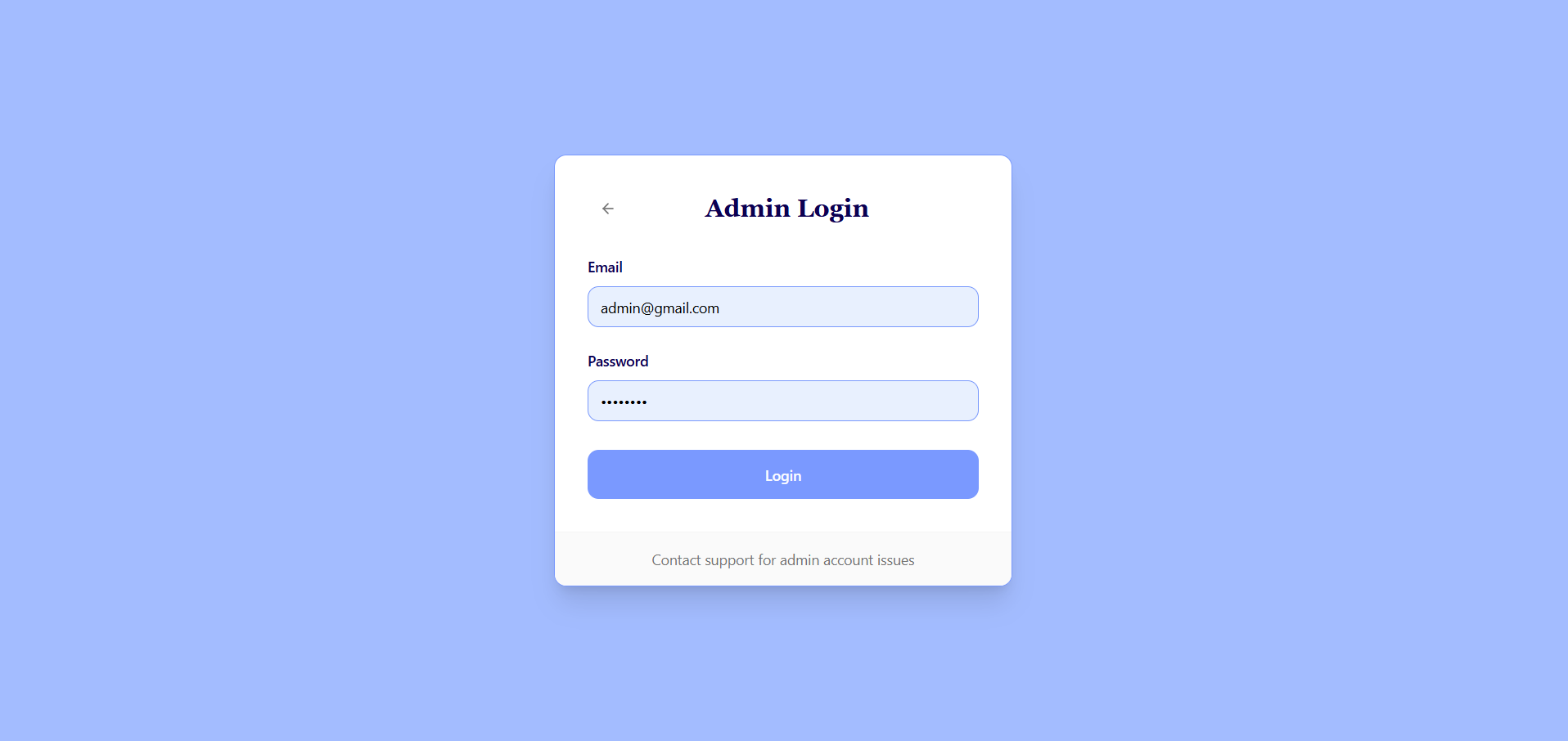


Figure 10-Booknest Admin Page

### Admin Dashboard

In this page we can see details like total users,no of books,total reviews,active users.

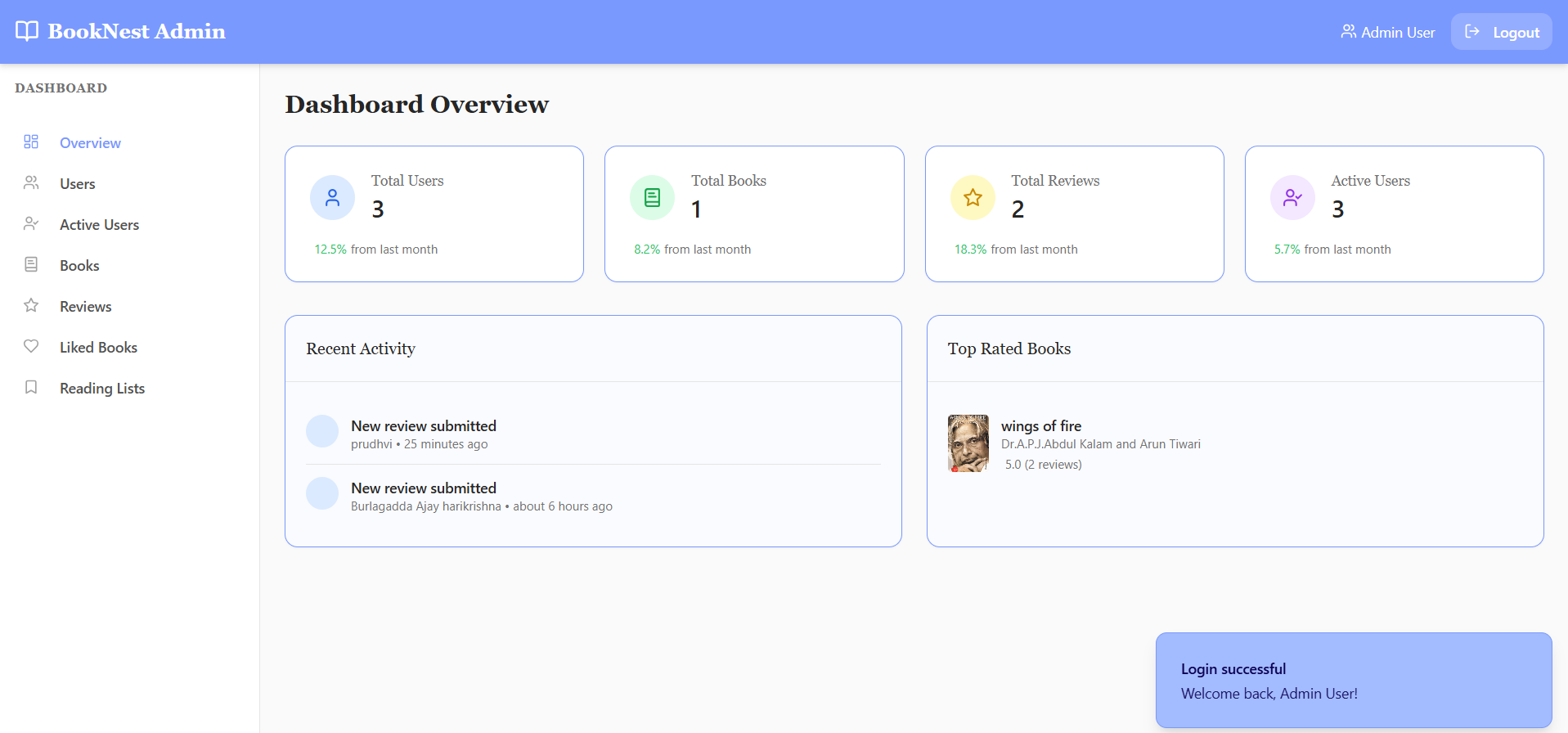


Figure 11-Booknest Admin Dashboard overview

### Users page in Admin Dashboard

In this page we can see all the users in Booknest.

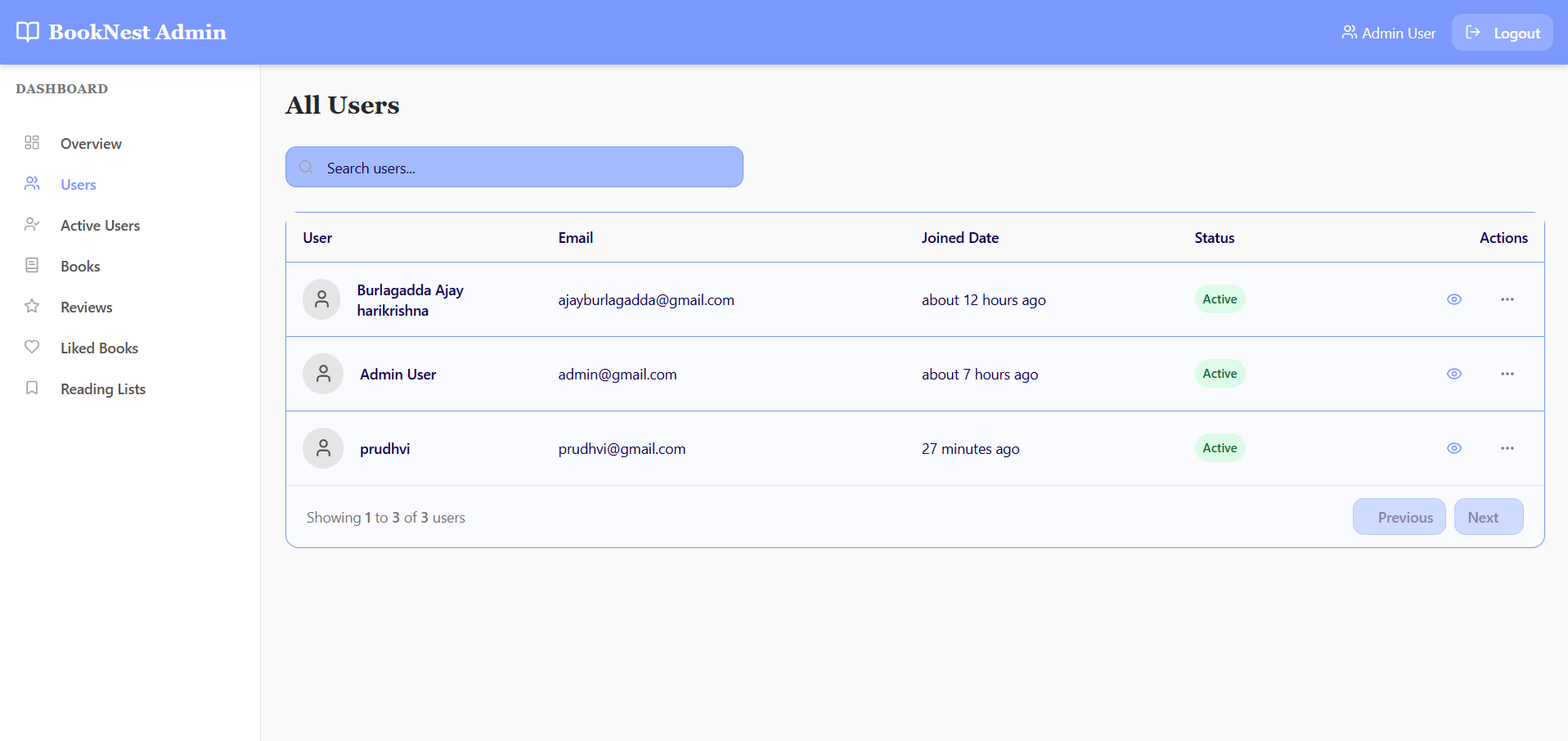


Figure 12-User Page in Admin Dashboard

### Active Users Page in Admin Dashboard

We can view the Active users

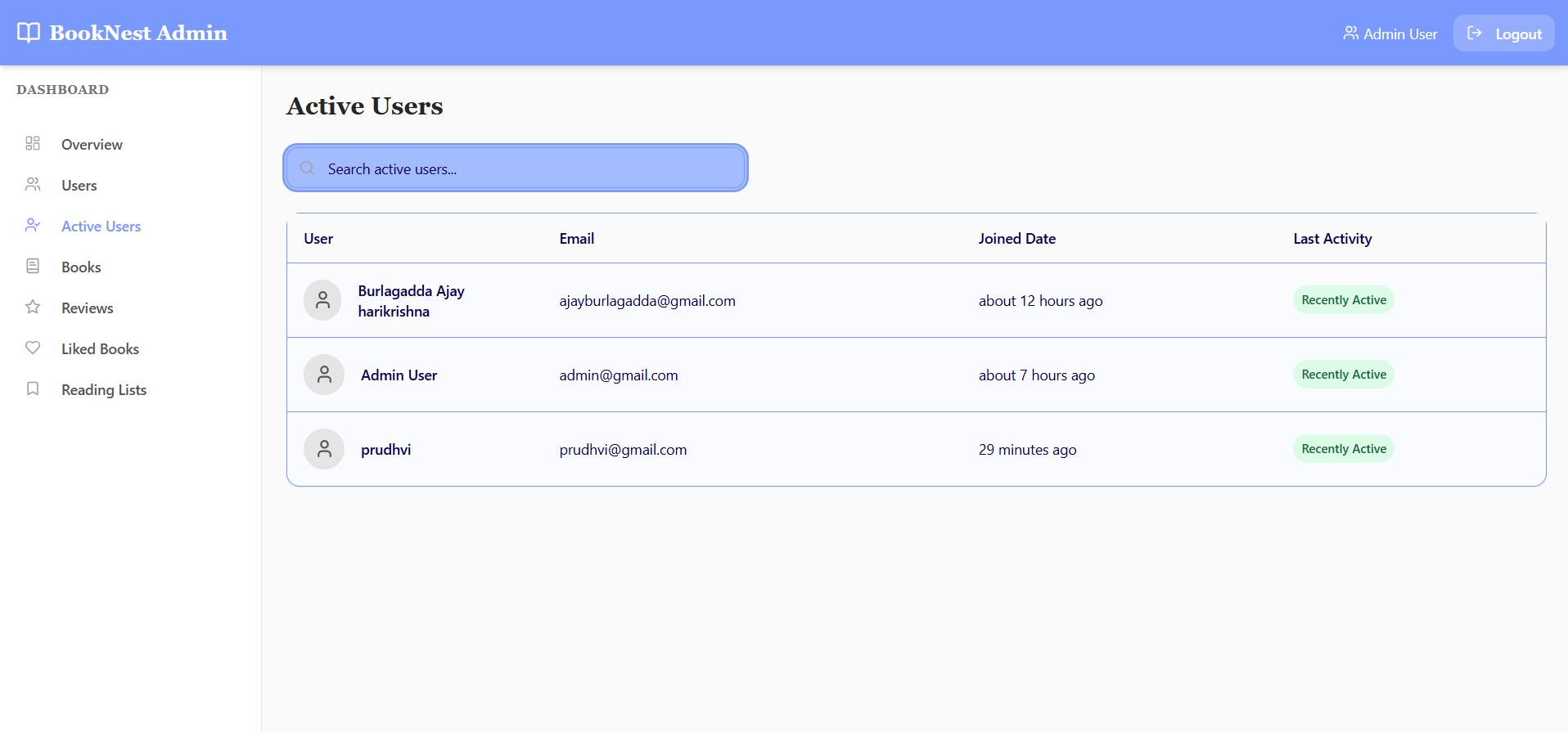


Figure 13-Active Users Page

### Books Management Page in Admin Dash board

Here we can Manage books like add books, edit books,delete books.

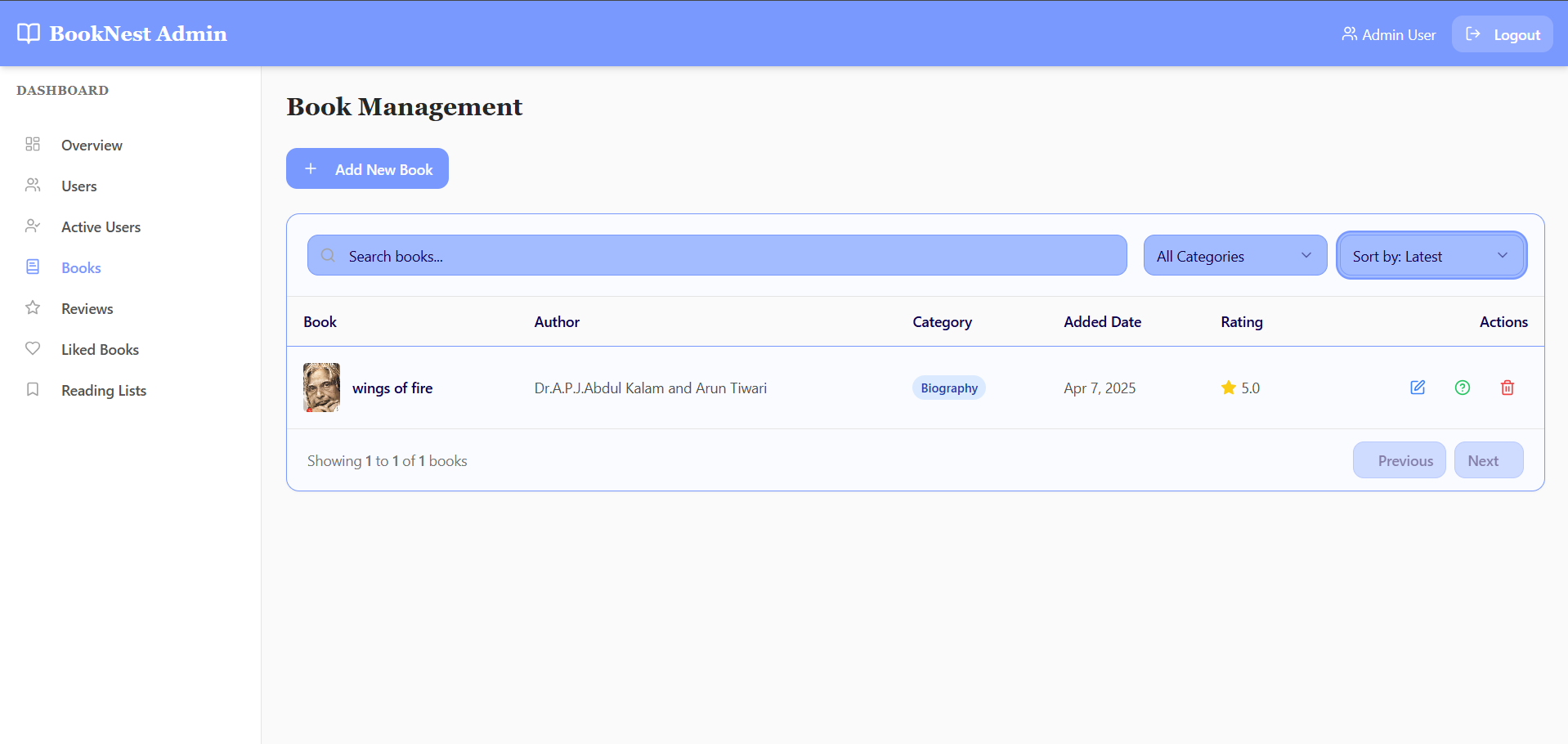


Figure 14-Book Management Page

### Books reviews Page

Here Admin can view All the reviews given by users.

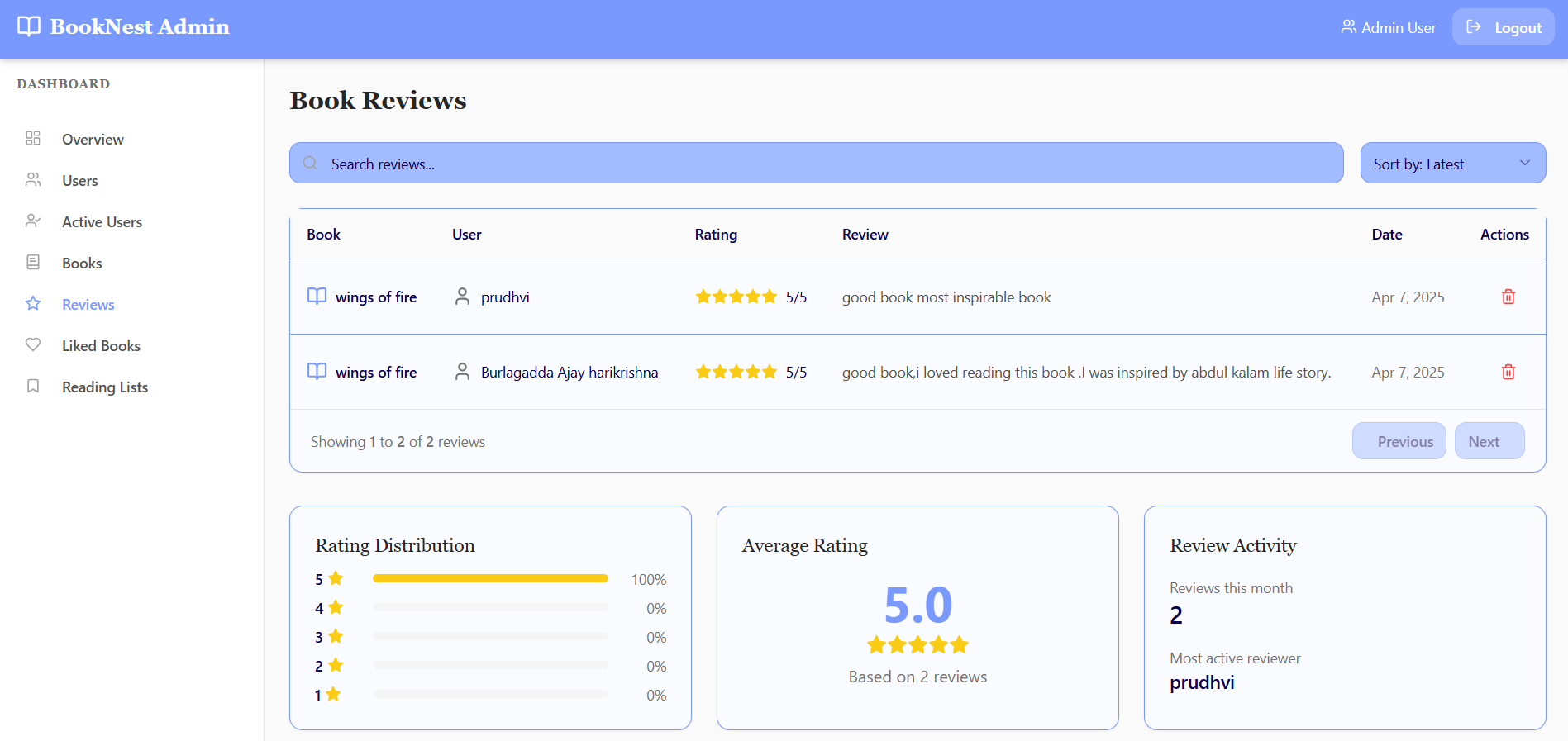


Figure 15-Book Reviews in Admin Dashboard

### User liked books page in Admin Dasboard

In this Page we can see users liked books

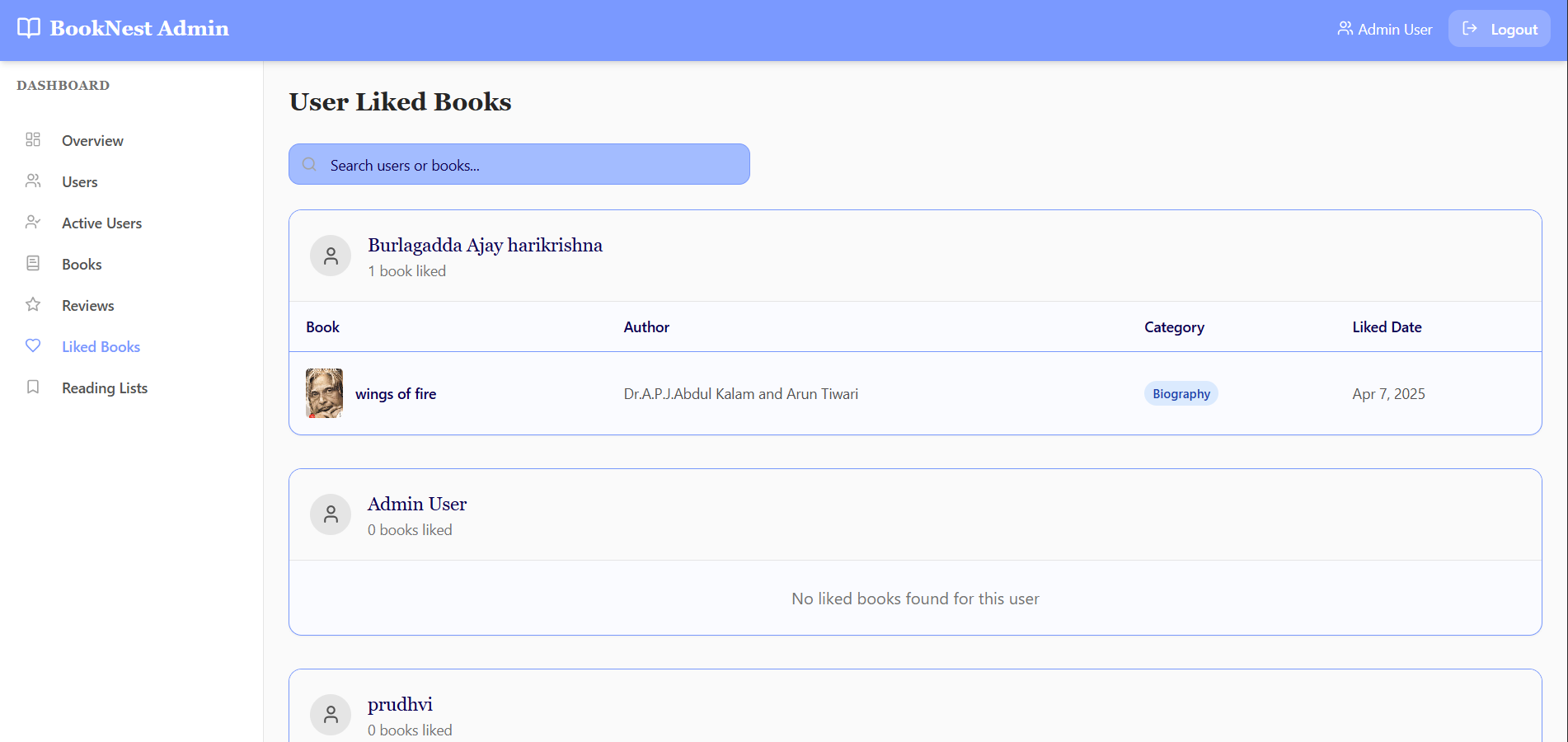


Figure 16-User liked Books in AdminDashboard

### User Reading List in Admin Dashboard

Users Reading Lists in Admin dashboards shows the users readlist books and how much percent the users have read that book.

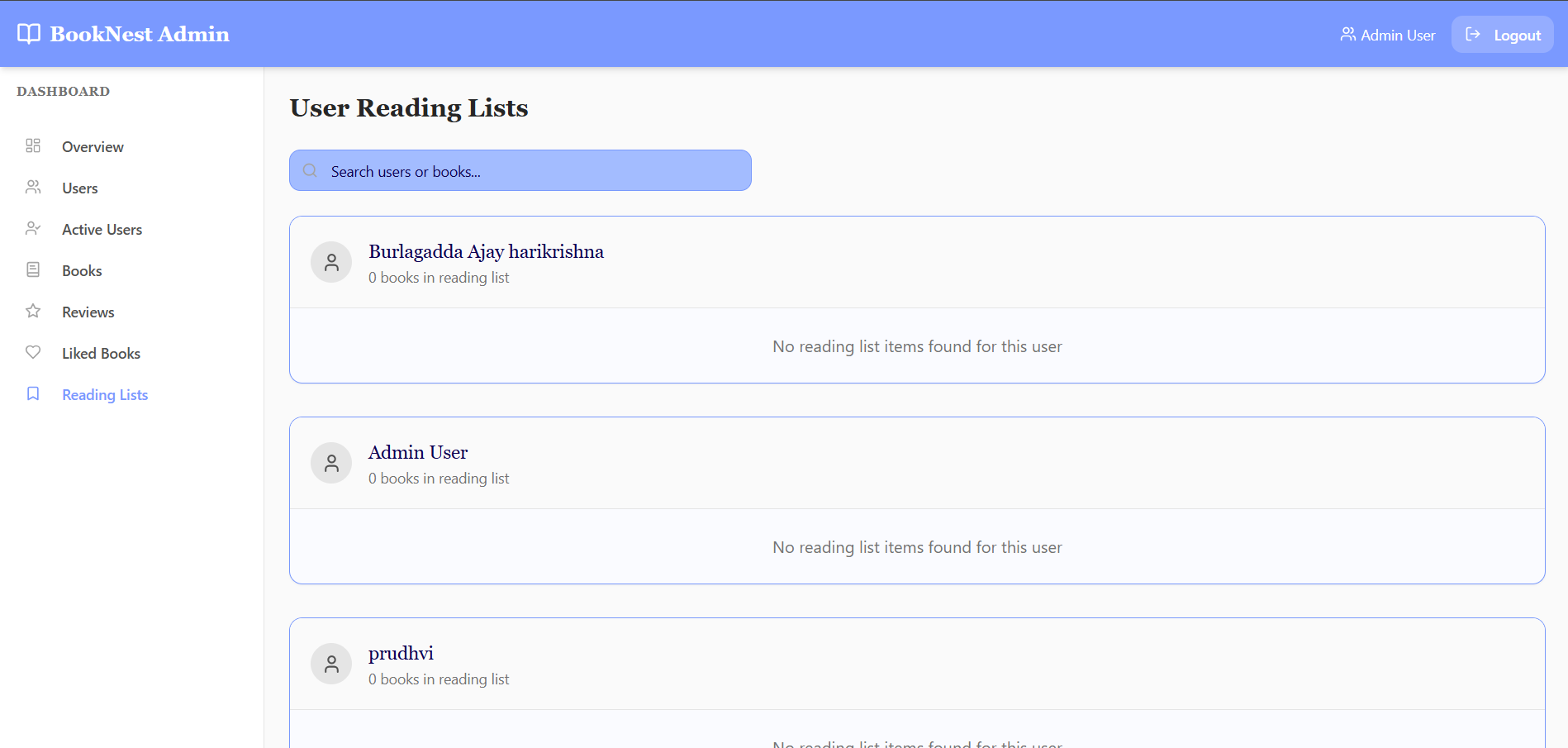


Figure 17-User Reading List in Admin Dashboard

### User Login Page

Through this page admin can login using his credentials

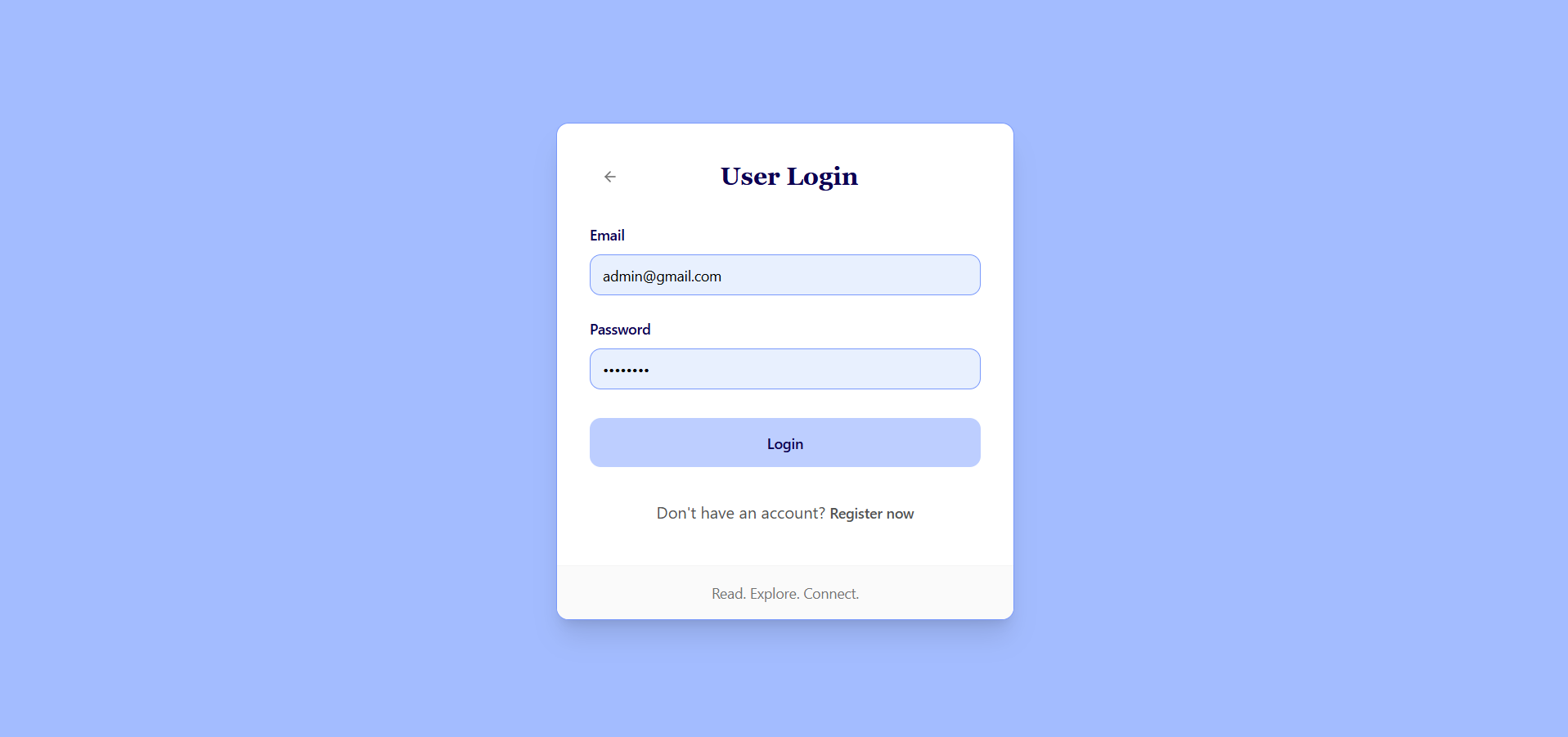


Figure 18-User Login Page

### User Registration Page

If User do not have an account we can create an account by registering in user registration page.

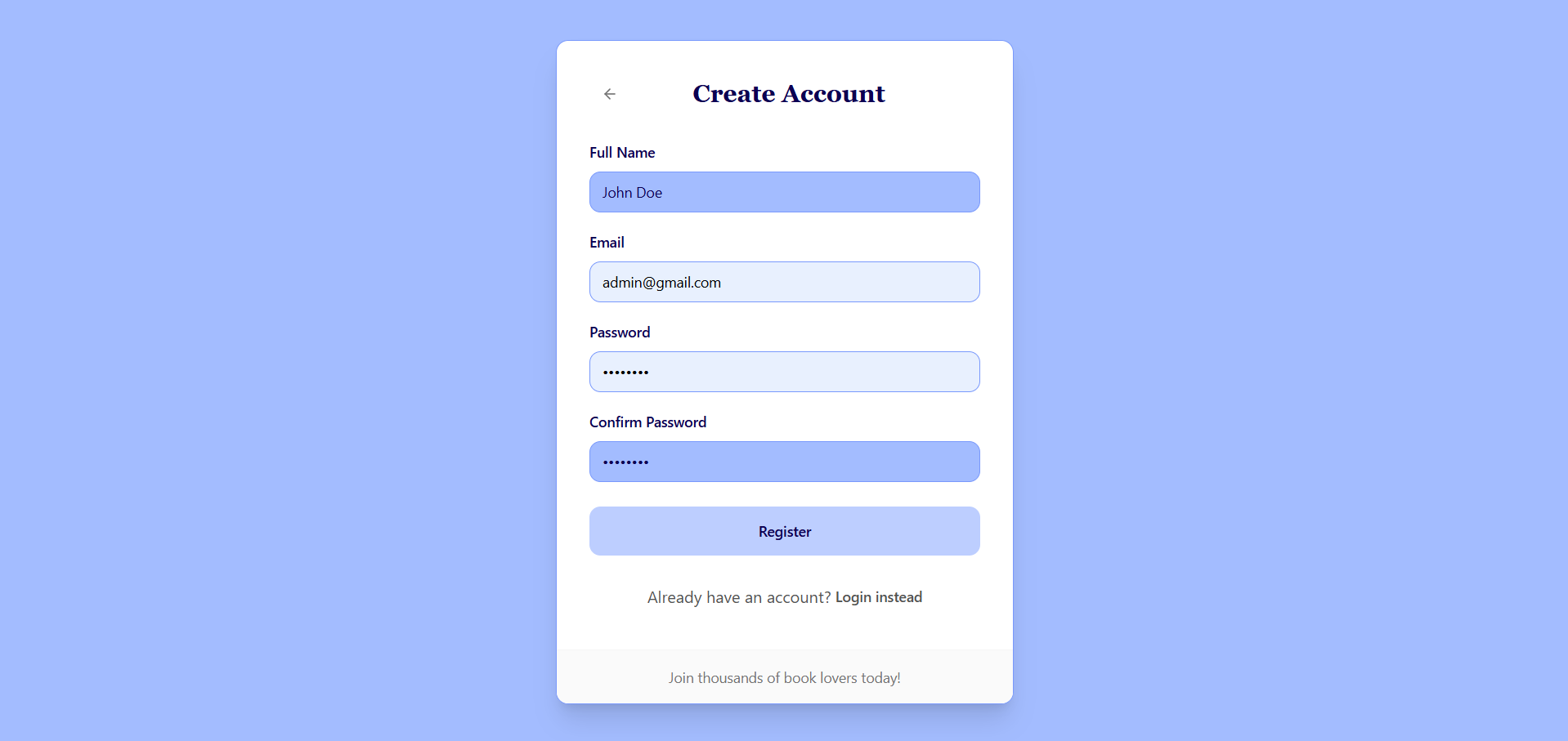


Figure 19-User Registration Page

### User Home Page

After login in as user successfully we will be redirected to user home page.

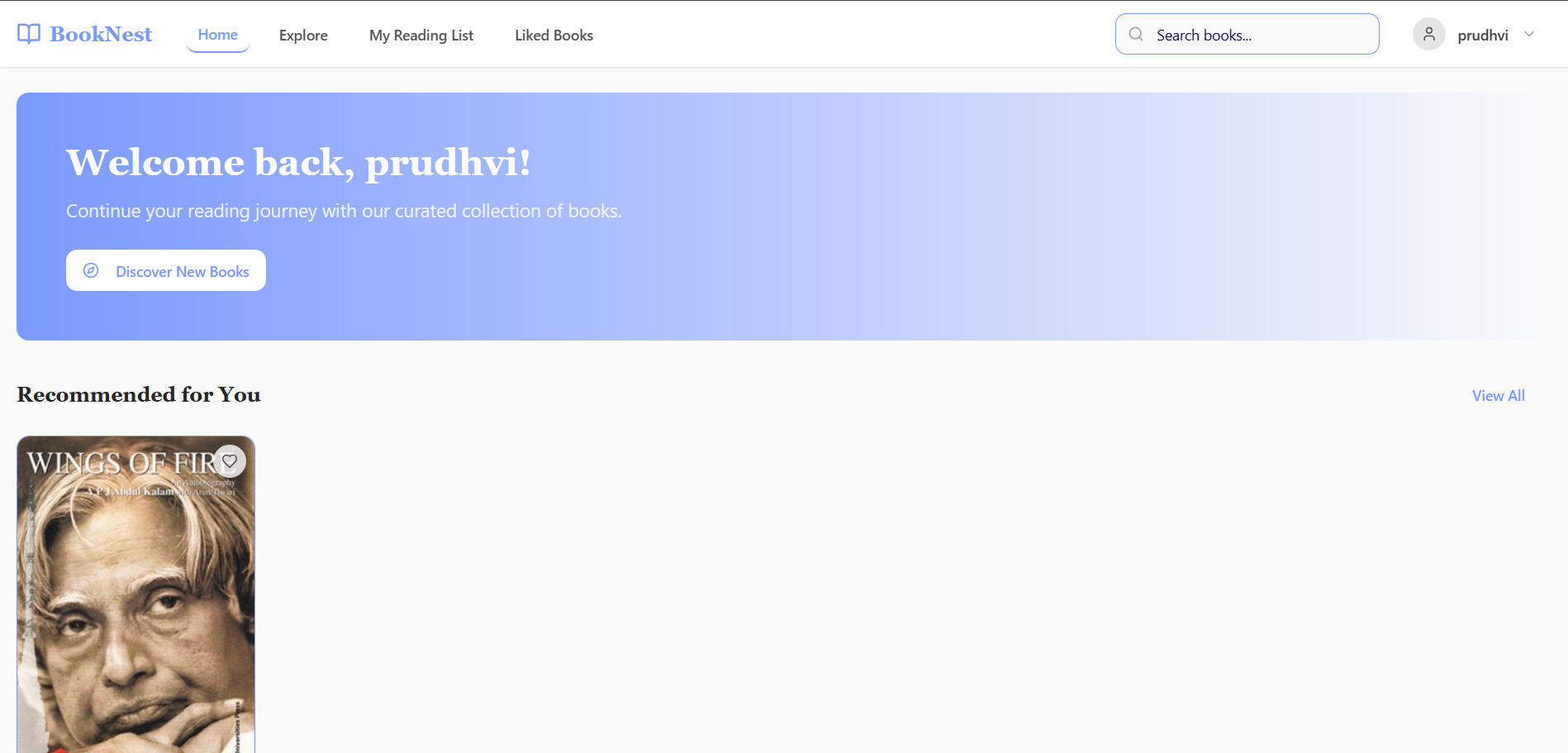


Figure 20-User HomePage

### Explore Books Page

Here user can view all the books and can comment on them.

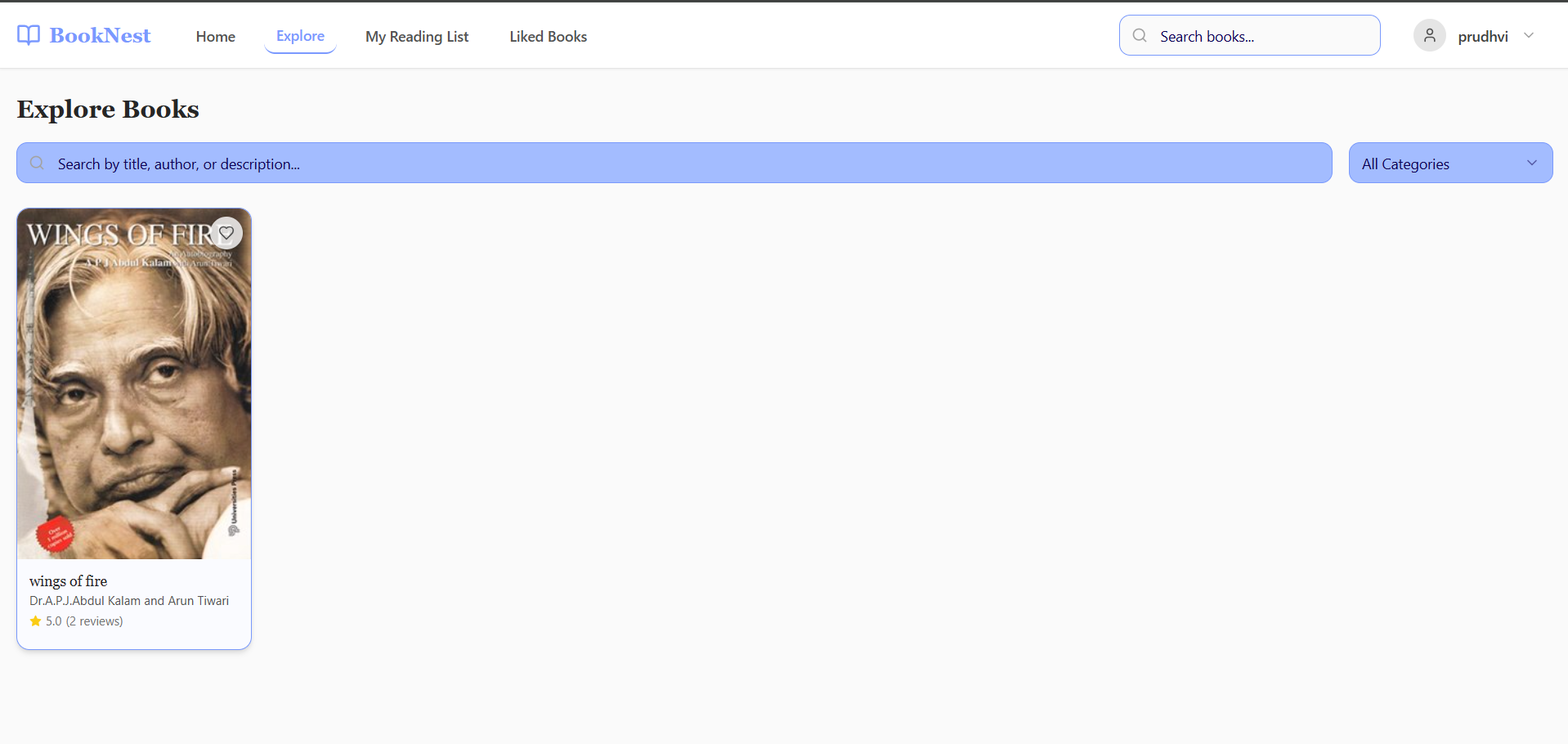


Figure 21-User Liked Books

### My Reading List Page

Here user can see all the reading list of the books the percentage of reading completed on those books buy him.

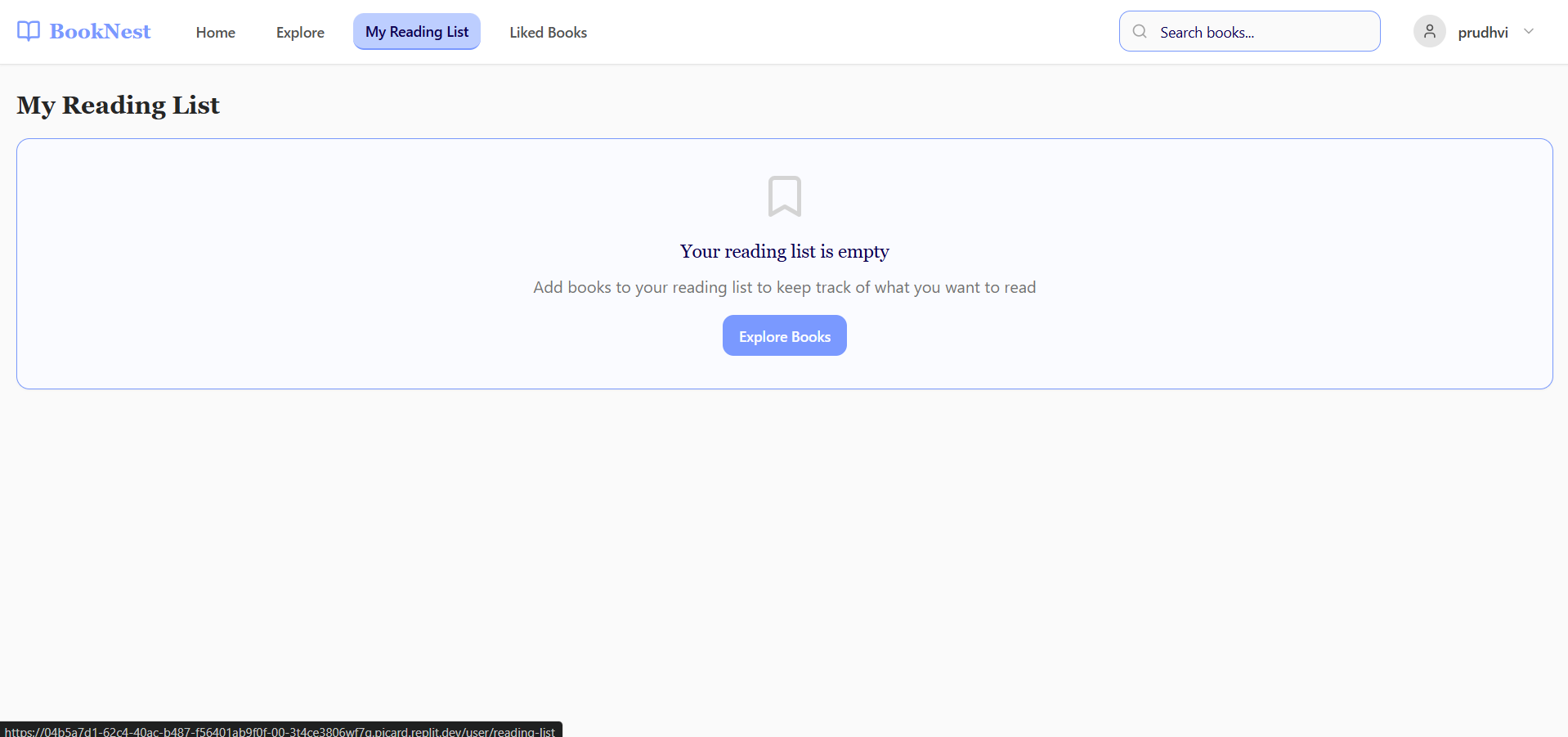


Figure 22-My Reading List Page

### Liked Books Page

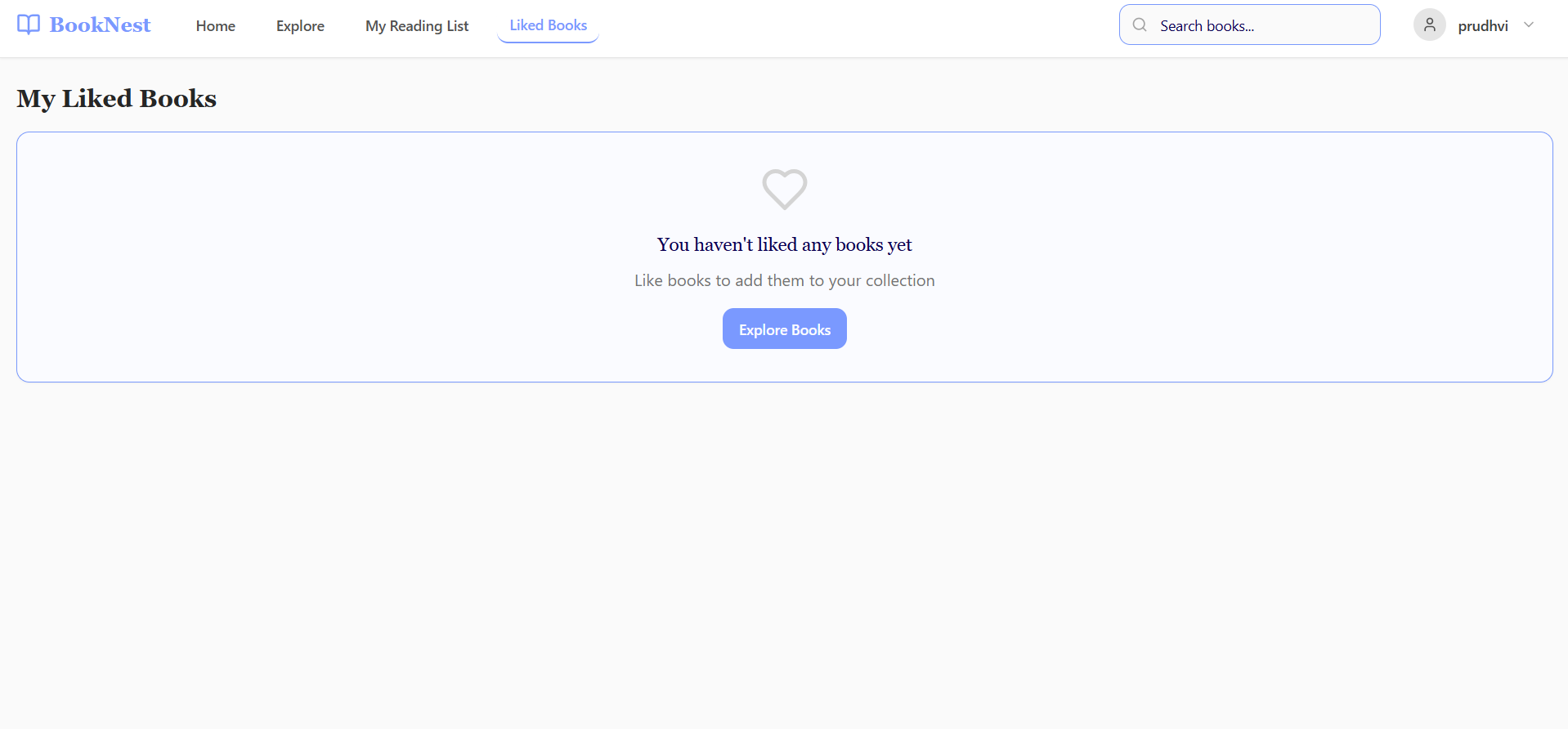
In this Page we can view all the liked books. 

Figure 23-My Liked Books Page

### Add new Book Page

By pressing add new book button in books in admin dashboard, admin can insert book details and 3 questions on books.

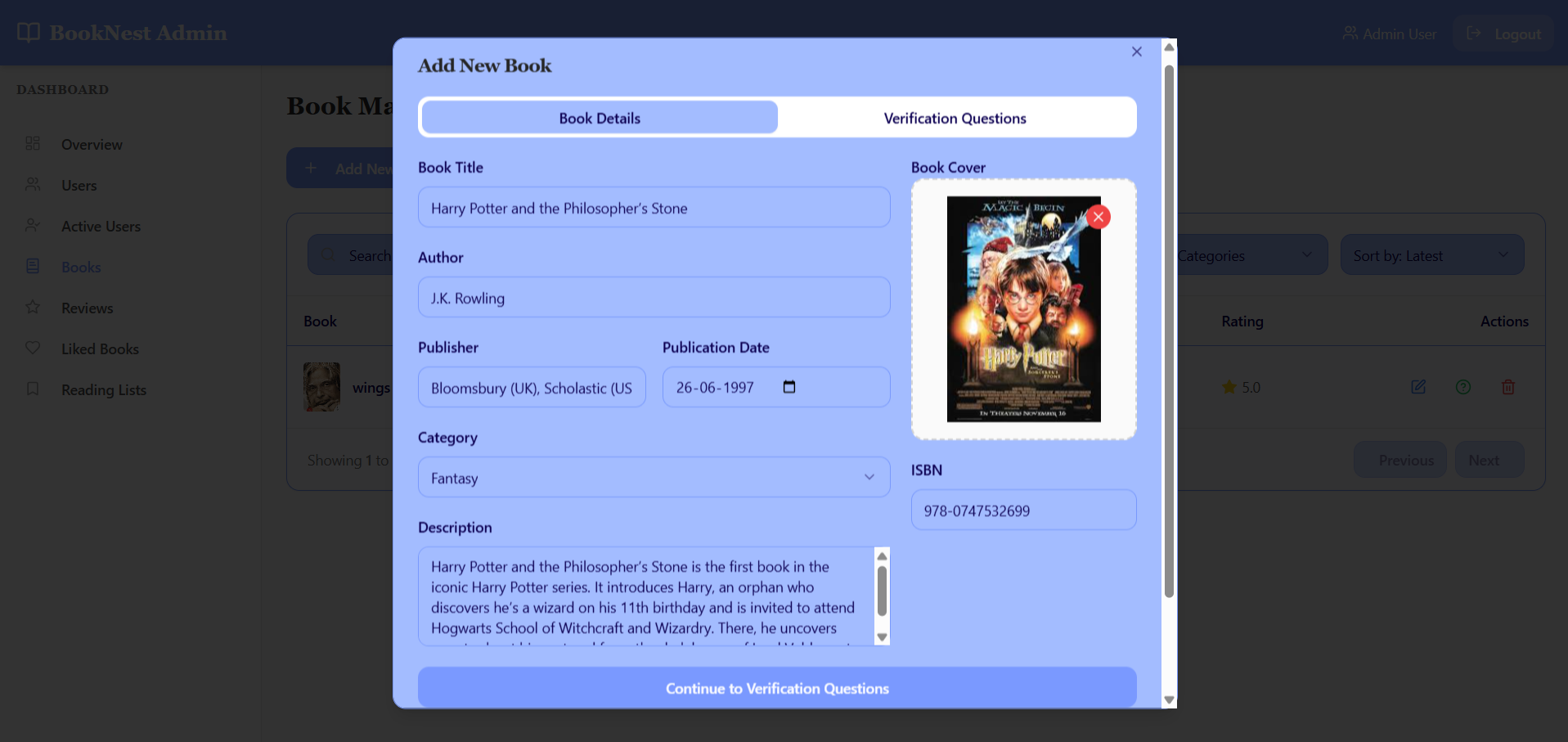


Figure 24- Adding new book

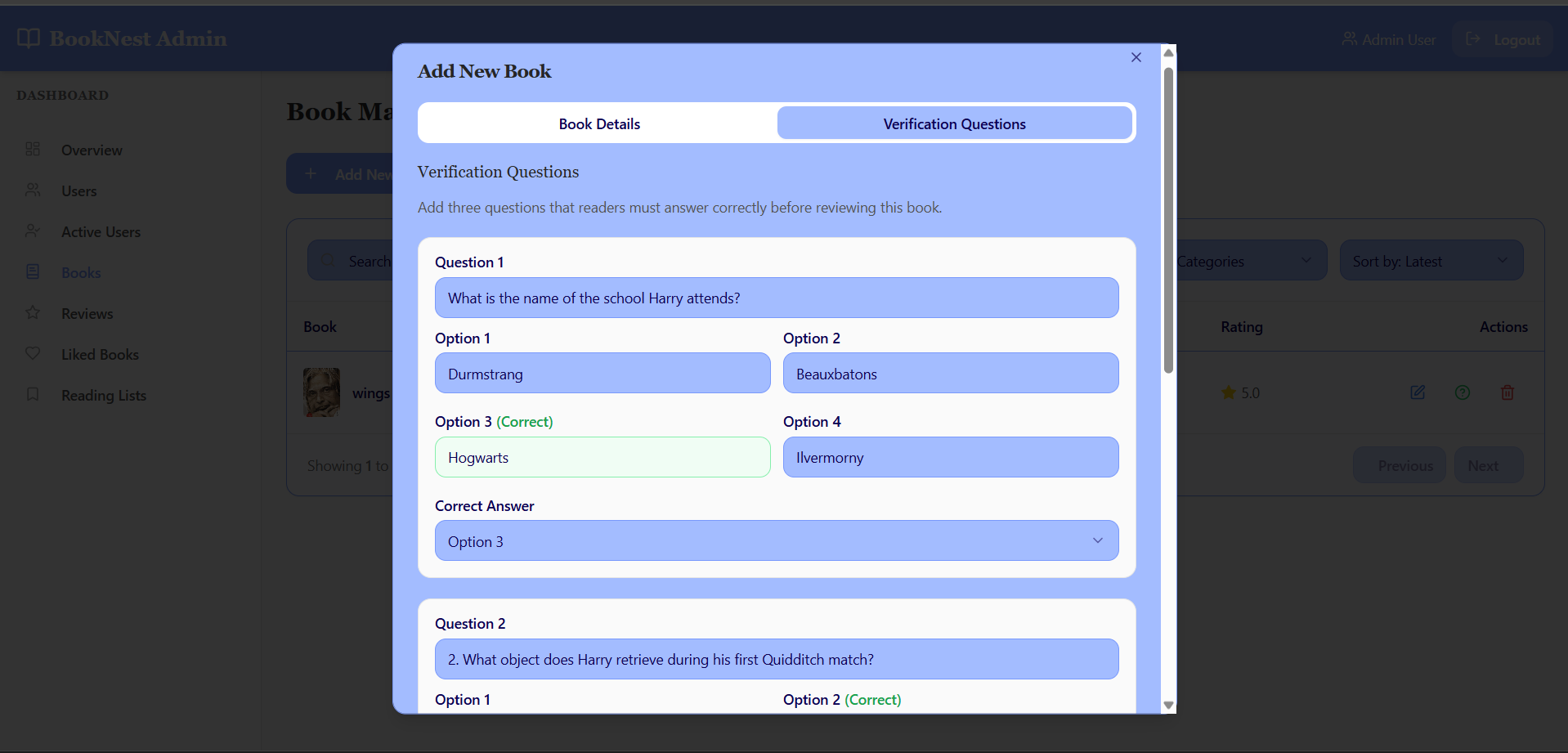


Figure 25-verification questions

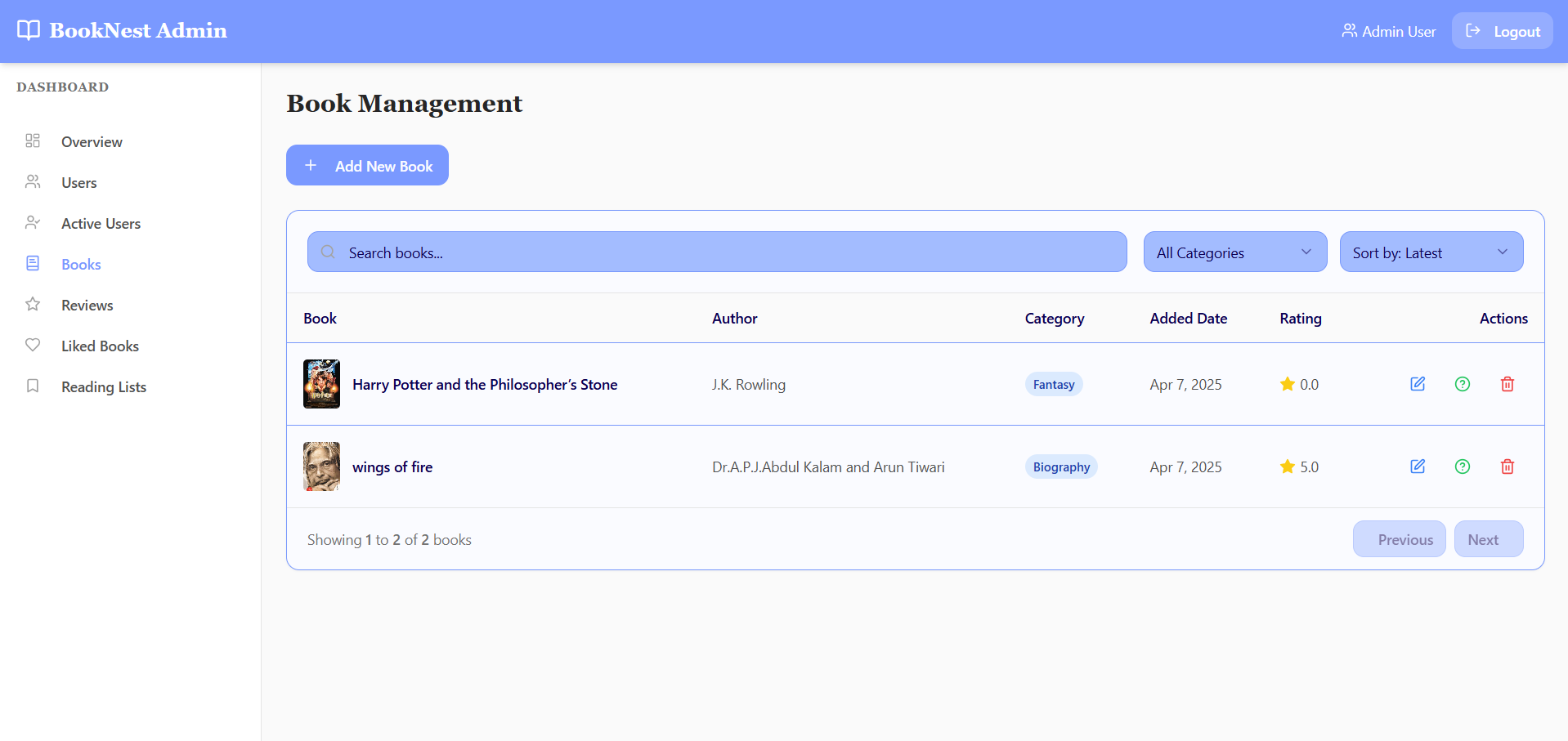


Figure 26-Books Added successfully

***CHAPTER - 6***

***TESTING***

# TESTING

## Software Testing

Software testing is the process of validating and verifying that a software application meets the technical requirements which are involved in its design and development. It is also used to uncover any defects/bugs that exist in the application. It assures the quality of the software. There are many types of testing software viz., manual testing, unit testing, black box testing, performance testing, stress testing, regression testing, white box testing etc. Among these performance testing and load testing are the most important one for an android application and next sections deal with some of these types.

## Black Box Testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

## White Box Testing

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

## Performance Testing

Performance testing is executed to determine how fast a system or sub-system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system such as scalability, reliability and resource usage.

## Load Testing

Load testing is primarily concerned with testing that can continue to operate under specific load, whether that is large quantities of data or a large number of users.

## Manual Testing

Manual Testing is the process of manually testing software for defects. Functionality of this application is manually tested to ensure the correctness. Few examples of test case for Manual Testing are discussed later in this chapter.

Test case-1

Adding a book by filling book details

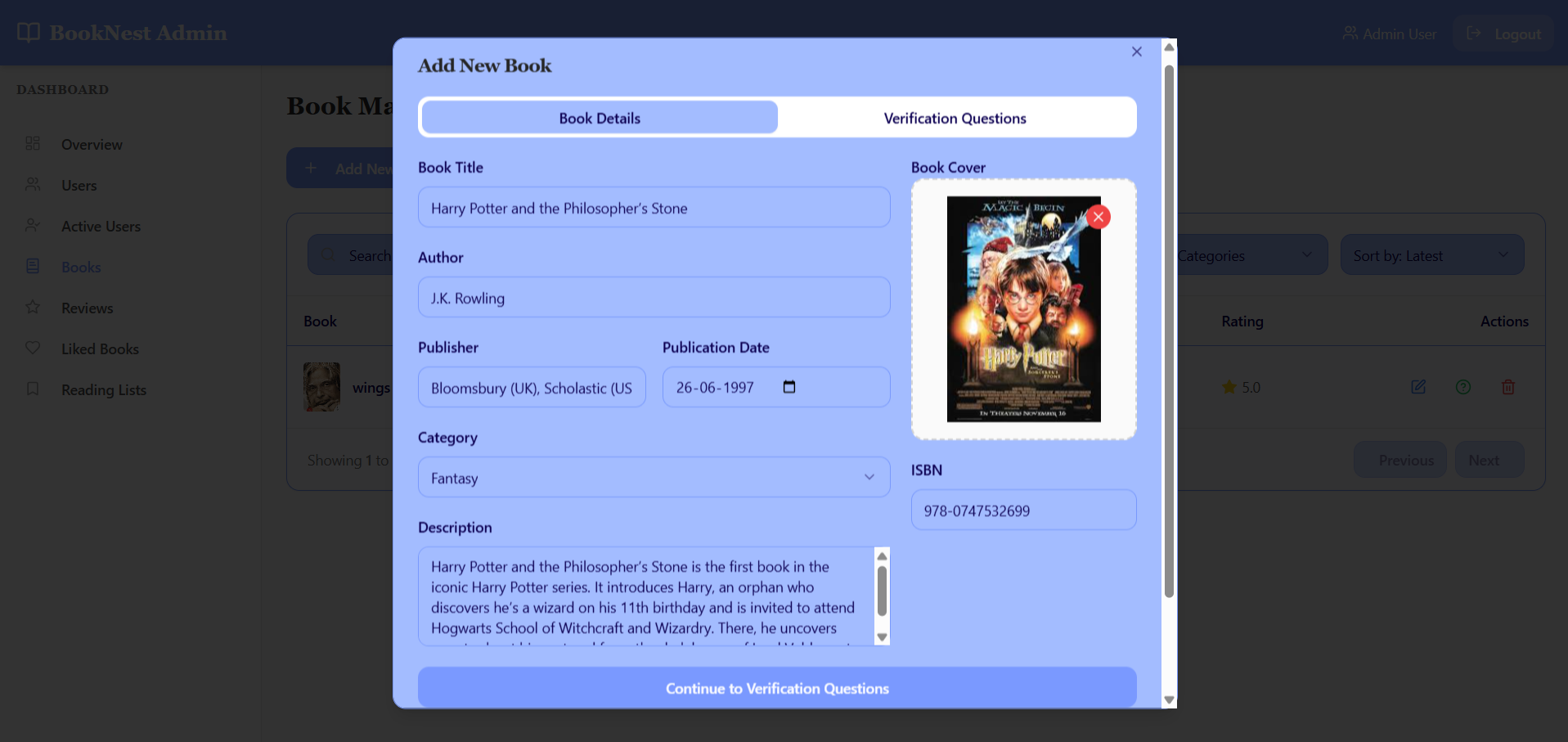


Figure 24- Adding new book

Adding questions on that book.

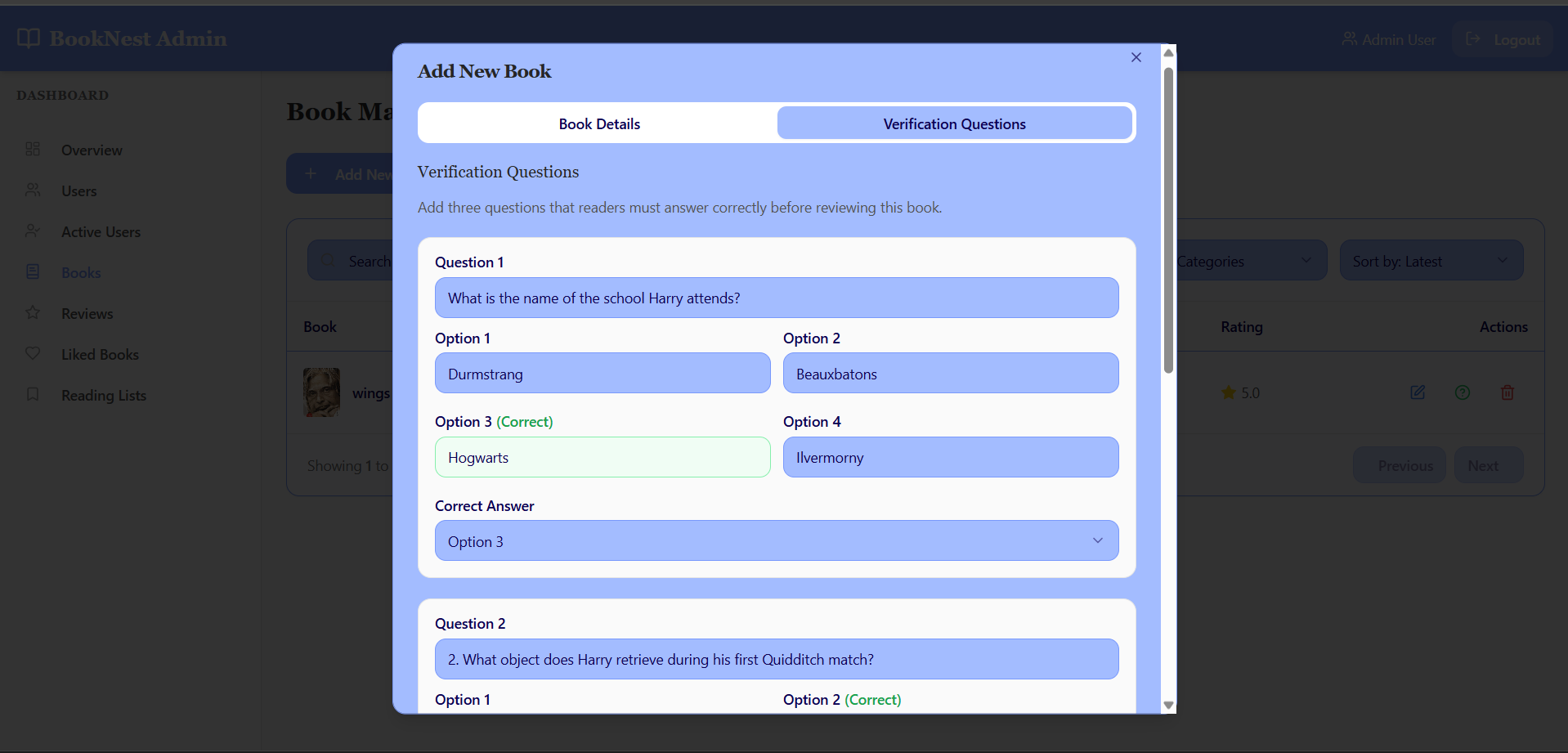


Figure 25-verification questions

Added successfully

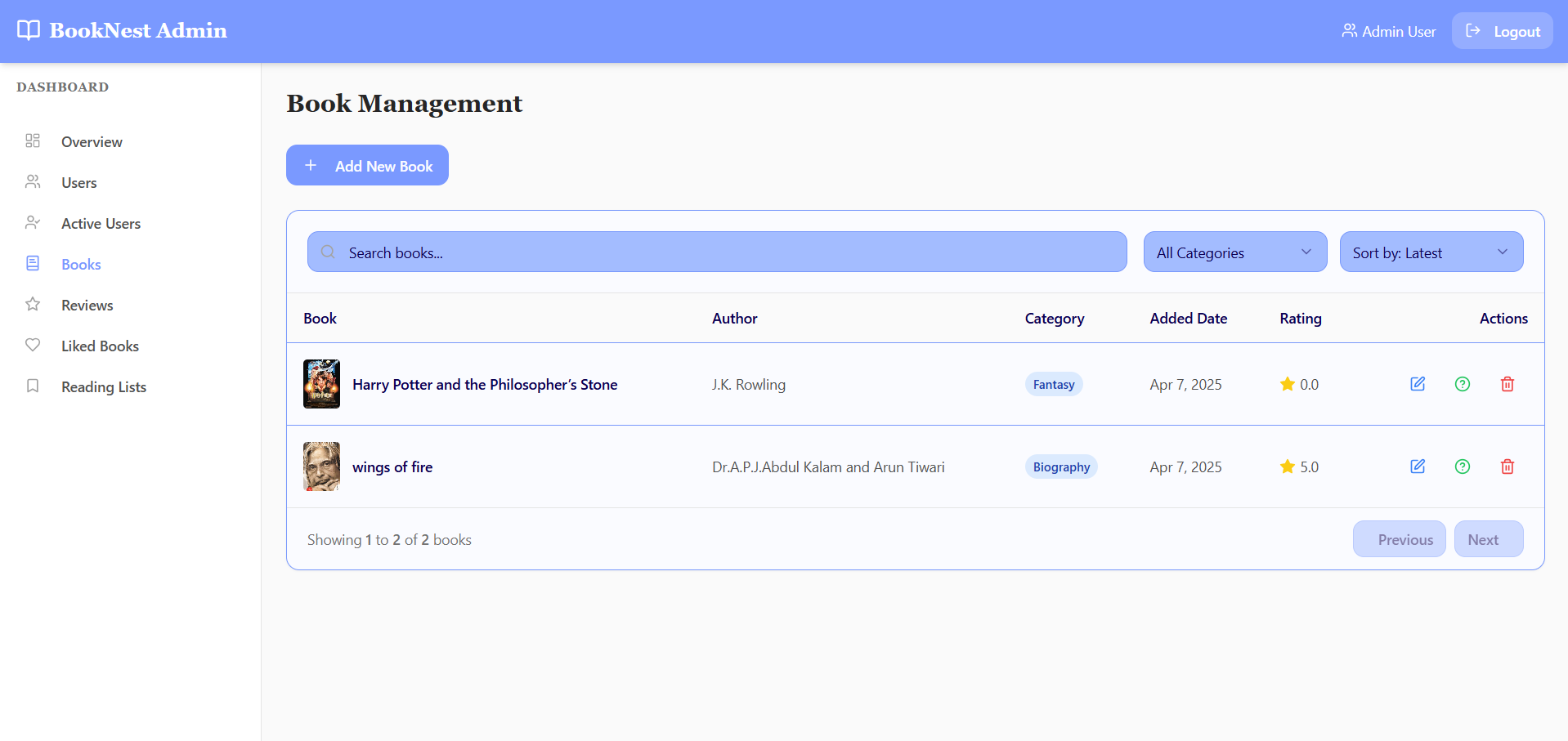


Figure 26-Books Added successfully

***CHAPTER - 7***

***RESULTS &CHALLENGES***

# RESULTS AND CHALLENGES

## Results

* Successfully developed a fully functional book review platform with verified review submission.
* Implemented role-based access for users and admins.
* Built an intuitive and responsive user interface using React and Tailwind CSS.
* Enabled secure authentication and session management.
* Deployed the web app with a cloud database (MongoDB Atlas).
* Admin dashboard provides effective tools for managing books, users, and verification questions.

## Challenges

* Designing a flexible system to manage book-specific verification questions.
* Ensuring data consistency during book deletions with associated reviews and questions.
* Handling session-based authentication securely across routes.
* Managing API state efficiently in a React frontend using TanStack Query.
* Balancing user experience with review authenticity through mandatory verification.

***CHAPTER - 8***

***CONCLUSIONS & FUTURE WORK***

# CONCLUSION

## Conclusions

BookNest successfully delivers a secure, interactive, and user-friendly platform for book discovery and verified reviews. It enhances trust by implementing a review verification mechanism and provides robust admin tools for content and user management. The modular architecture supports maintainability and scalability.

## Scope for Future Work

* Integration of social features like likes, comments, and user following
* Mobile application development for Android and iOS
* Advanced recommendation engine using machine learning
* Reading progress tracking with time and page metrics
* Virtual book clubs and community features

## Limitations

* Review verification relies solely on admin-created questions
* No real-time chat or discussion feature
* Limited analytics for admins
* Requires manual input for book data unless integrated with external APIs
* Mobile support is through responsive web design only, no native apps yet

BIBOLOGRAPHY

* MongoDB Documentation – <https://www.mongodb.com/docs>
* React.js Official Documentation – <https://react.dev>
* Node.js Documentation – <https://nodejs.org>
* Express.js Guide – <https://expressjs.com>
* Tailwind CSS Documentation – <https://tailwindcss.com/docs>
* Shadcn UI Components – https://ui.shadcn.com
* TanStack Query – https://tanstack.com/query/latest
* React Hook Form – <https://react-hook-form.com>
* Zod Schema Validation – <https://zod.dev>
* OWASP Authentication Cheat Sheet- https://cheatsheetseries.owasp.org