A Comprehensive Study on Mobile Application Share Books for Readers

by

Md. Nazmul Haque
ID: CSE2203027032

Bibi Fatema
ID: CSE2101022123

Mst. Mahabuba Akter
ID: CSE2203027105

Jaynal Abedin
ID: CSE2203027168

Supervised by
Md. Shamim Hossain

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)
September 2026

APPROVAL

The project titled **"A Comprehensive Study on Mobile Application Share Books for Readers"** submitted by Md. Nazmul Haque (CSE2203027032), Bibi Fatema (CSE2101022123), Mst. Mahabuba Akter (CSE2203027105), and Jaynal Abedin (CSE2203027168) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and content.

Board of Examiners

Md. Shamim Hossain
Lecturer
Department of Computer Science and Engineering
Sonargaon University (SU)

Supervisor

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 1

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 2

(Examiner Name and Signature)
Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 3

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Md. Shamim Hossain, Lecturer,** Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Md. Shamim Hossain)
Supervisor

Signature

Md. Nazmul Haque
ID: CSE2203027032

Bibi Fatema
ID: CSE2101022123

Mst. Mahabuba Akter
ID: CSE2203027105

Jaynal Abedin
ID: CSE2203027168

ABSTRACT

Traditional book lending relies on informal social networks and lacks a systematic mechanism for inventory management, borrow tracking, and friend-based discovery. Existing solutions either focus solely on e-books or require centralized library infrastructure. This creates a gap for peer-to-peer physical book sharing within verified social circles—particularly in emerging markets where digital libraries remain economically inaccessible. This project presents Booklify, a comprehensive cross-platform social book-lending application designed to digitize and decentralize physical book lending, enabling users to build personal libraries, discover books within their social networks, and facilitate formal borrowing arrangements.

Booklify implements a full-stack solution comprising: (1) a Node.js/Express REST API with MongoDB for persistent storage, JWT-based stateless authentication, and role-based access control; (2) a React Native mobile application supporting iOS and Android with offline-capable Redux state management; and (3) real-time communication infrastructure via Socket.IO for synchronous messaging between lenders and borrowers. The backend employs asynchronous task queuing (Redis) for non-blocking operations, Zod-based schema validation for data integrity, and Node mailer for transactional email workflows (registration verification, password recovery).

Booklify demonstrates that systematic digitization of peer-to-peer lending is technically feasible and addresses a tangible social need. The project serves as a reference architecture for community-driven sharing platforms, illustrating modern full-stack practices (monorepo design, token-based auth, real-time protocols, containerized deployment) applicable to emerging markets and resource-constrained communities.

ACKNOWLEDGEMENT

First and foremost, we express our deepest gratitude to the Almighty for granting us the strength, patience, and perseverance to complete this thesis within the scheduled time.

We are profoundly indebted to our supervisor, **Md. Shamim Hossain, Lecturer**, Department of Computer Science and Engineering, Sonargaon University, for his thoughtful guidance, constructive feedback, and unwavering support throughout the development of Booklify. His invaluable insights on system architecture, modern full-stack practices, and real-world deployment challenges have shaped this work in profound ways. His encouragement to pursue both technical rigor and practical feasibility has been instrumental in transforming our vision into a production-ready platform.

We extend our sincere thanks to the Honorable **Prof. Bulbul Ahamed** Head of the Department, the faculty members, and the staff of the Department of Computer Science and Engineering at Sonargaon University. Their rigorous instruction across the four years of our undergraduate study laid the critical foundation in algorithms, databases, networks, and software engineering that made this ambitious project possible. Their mentorship and continued encouragement throughout our final year enabled us to pursue research at this scale.

We also thank our peers, classmates, and the wider open-source community whose exceptional tools, libraries, and freely shared knowledge made the technical work tractable within an academic timeline. The frameworks and platforms we leveraged—React Native, Node.js, MongoDB, Socket.IO—were built and refined by thousands of contributors whose generosity and dedication accelerated our development process beyond measure.

Finally, our heartfelt gratitude goes to our parents and families for their patience, sacrifices, and endless love. Their unwavering support, understanding of late nights spent debugging, and belief in our abilities sustained us through every challenge. None of what follows would exist without them.

LIST OF ABBREVIATIONS

API	Application Programming Interface
CI/CD	Continuous Integration/ Continuous Deployment
CORS	Cross-Origin Resource Sharing
CRUD	Create, Read, Update, Delete
DB	Database
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
iOS	iPhone Operating System
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
MVC	Model View Controller
ORM	Object-Relational Mapping
OTP	One-Time Password
QR	Quick Response Code
RBAC	Role-Based Access Control
React Native	Cross-platform Mobile Frameworks
REST	Representational State Transfer Protocol
RTK	Redux Toolkit

SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SSL/TLS	Secure Sockets Layer/Transport Layer Security
UI	User Interface
UX	User Experience
VCS	Version Control System

TABLE OF CONTENTS

Title			Page No.
DECLARATION			iii
ABSTRACT			iv
ACKNOWLEDGEMENT			v
LIST OF ABBREVIATIONS			vi-vii
CHAPTER 1			1 – 6
INTRODUCTION			
1.1	Introduction.....		1
1.2	Background and Motivation…….....		2
1.3	Problem Statement……....		2 – 3
1.4	Background: Peer-to-Peer Sharing Platforms……		3
1.5	Literature Review…….....		4
	1.5.1	Social Book Platforms and Community Lending.....	4
	1.5.2	Real-Time Communication and Coordination…….....	4 – 5
	1.5.3	Mobile-First Development in Emerging Markets.....	5
	1.5.4	Mono Repo Architecture and Code Reuse.....	5
	1.5.5	Research Gap.....	5
1.6	Objectives.....		5 – 6
1.7	Scope and Limitations.....		6
CHAPTER 2			7 – 8
LITERATURE REVIEW AND TECHNICAL FOUNDATION			
2.1	Existing Book Sharing and Discovery Platforms…….....		7
2.2	Community-Based Resource Sharing……		7
2.3	Local Discovery and Geographic Processing……		7

Title Page No.

2.4 Trust, Verification, and Accountability……. 7 – 8

2.5 Real-time Communication…….. 8

2.6 Cross-Platform Mobile Development…… 8

2.7 Mon repo and Build Management……. 8

2.8 Literature-Derived Research Gap…….. 8

CHAPTER 3 9 – 14

METHODOLOGY AND REQUIREMENT ANALYSIS

3.1 Development Methodology……... 9

3.2 Requirement Analysis…… 9 – 12

3.3 User and Permission Model…….. 12 – 13

3.4 Data Requirements…… 13 – 14

3.5 API-First Development……. 14

CHAPTER 4 15 – 20

SYSTEM ARCHITECTURE AND DESIGN

4.1 High-Level Architecture…… 15 – 16

4.2 Mon repo Architecture…….. 16

4.3 Backend Layer Design…….. 16

4.4 Mobile Application Architecture…….. 16

4.5 Authentication Architecture…….. 17

4.6 OTP and Email Security Design…… 17

4.7 Book and Lending Design……. 18

4.8 Friend and Trust Design…… 18

4.9 Real-time Chat Architecture……... 18 – 19

4.10 Data Model Design…… 19 – 20

Title Page No.

CHAPTER 5 21 – 25

IMPLEMENTATION

5.1 Development Environment…… 21 – 22

5.2 Backend Implementation……... 22

5.3 User and Book Models…….. 22

5.4 Authentication and Account Recovery……. 22 – 24

5.5 Book Search and Library Operations…… 24

5.6 Friend Management…….. 24

5.7 Borrow Request Implementation…….. 24

5.8 Real-time Chat Implementation……. 24 – 25

5.9 Redis-Based Background Processing…… 25

5.10 Mobile State and Reauthentication…… 25

5.11 Deployment and CI/CD……. 25

5.12 API Testing Through Postman…….. 25

CHAPTER 6 26 – 30

TESTING RESULTS AND DISCUSSION

6.1 Testing Strategy……. 26 – 28

6.2 Authentication and Security Results…….. 28

6.3 Book Lending Results……... 28

6.4 Real-time Chat Results……... 28

6.5 API Integration Results…… 28

6.6 Observed System Outcomes…… 29 – 30

6.7 Comparison with the Earlier Thesis Draft…… 30

6.8 Limitations Identified During Evaluation……. 30

Title Page No.

CHAPTER 7 31 – 32

CONCLUSION AND FUTURE WORK

7.1 Conclusion…… 31

7.2 Future Work……. 31 – 32

7.3 Final Remarks……... 33

REFERENCES..... 34

LIST OF TABLES

Table No	Title	Page No
Table 3.1	Functional Requirements	9
Table 3.2	Non-Functional Requirements	11
Table 3.3	User and Permission Matrix	12
Table 3.4	Core Data Entities	13
Table 5.1	Technology Stack	21
Table 5.2	Authentication and Domain API Summary	23
Table 6.1	Functional Test Matrix	26
Table 6.2	Observed System Outcomes	29

LIST OF FIGURES

Figure No	Title	Page No
Fig. 4.1	High-Level System Architecture	15
Fig. 4.2	Authentication and Session Lifecycle	17
Fig. 4.3	Structured Borrowing Lifecycle	18
Fig. 4.4	Real-Time Chat Message Flow	19
Fig. 4.5	Core Data Model and Relationships	20

CHAPTER 1
INTRODUCTION

1.1 Introduction

The practice of sharing knowledge through books has shaped human civilization for centuries. Libraries emerged as communal repositories, lending circles formed within neighborhoods, and informal book exchanges between friends have sustained literary culture across generations. However, the digital transformation of the last two decades has fundamentally altered the landscape of how people discover, access, and share physical books. While e-books and digital platforms have made literature more accessible in developed economies, emerging markets still rely predominantly on physical book distribution. Yet despite this need, no systematic, decentralized platform exists for peer-to-peer physical book lending within verified social circles [1].

Today's book sharing faces several interconnected challenges. Informal lending arrangements lack systematic tracking mechanisms, resulting in lost books, forgotten loans, and damaged friendships. Centralized library systems require institutional infrastructure that many communities cannot support. General marketplace platforms like OLX and Mercari facilitate book sales rather than lending, introducing monetary friction that discourages casual sharing. Most critically, there is no integrated solution that combines book discovery, social trust verification, lending lifecycle management, and real-time communication within a single platform designed for emerging markets [2].

This thesis presents Booklify, a comprehensive cross-platform social book-lending application that systematizes peer-to-peer physical book sharing. Booklify bridges the gap between informal social networks and formal library infrastructure through a full-stack architecture comprising a Node.js/Express REST API backend deployed to Vencel, a React Native mobile application supporting iOS and Android, real-time messaging via Socket.IO, and Redis-based asynchronous task processing for scalability. By embedding book discovery, lending management, and social communication into a single unified platform, Booklify addresses a tangible social need while demonstrating modern full-stack development practices applicable to emerging markets.

1.2 Background and Motivation

The book lending landscape reflects broader patterns in digital inclusion and community engagement. In developing economies, physical books remain the primary medium for knowledge dissemination, yet access is constrained by cost, geography, and library infrastructure [3]. Simultaneously, the rise of social networks has created unprecedented opportunities for trust-based communities, yet these platforms do not facilitate the practical coordination required for physical goods exchange.

Several structural barriers currently prevent efficient book sharing:

1. **Information Asymmetry:** Friends and acquaintances lack visibility into what books others own. Discovering whether a friend has a desired book requires manual communication, often through messaging or casual conversation.
2. **Lending Accountability:** Informal lending creates ambiguity about ownership and return obligations. "Lost in the lending chain" is a common experience, leading to disputes and forgotten debts.
3. **Geographic Friction:** Physical proximity constraints limit lending opportunities to immediate social circles. Expanding the lending network beyond close friends requires mechanisms for verifying trust and managing logistics.
4. **Lack of Structured Workflow:** Most lending arrangements lack clear stages: request, approval, tracking, and return. This informality makes it impossible to generate lending statistics, identify patterns, or resolve disputes systematically.

Existing solutions fail to address these barriers comprehensively. Commercial systems like Goodreads provide social book discovery but offer no lending functionality. Marketplace platforms enable buying and selling but not gift-based sharing. Library management systems exist but serve institutions, not peer-to-peer networks. Community lending projects (like Little Free Libraries) operate at neighborhood scale without digital coordination. This fragmentation creates the opportunity for Booklify [4].

1.3 Problem Statement

The central problem this thesis addresses is: How can a digital platform systematize peer-to-peer physical book lending within trusted social networks in a way that is more efficient than informal arrangements, more scalable than location-based solutions, and more accessible than institutional library systems?

More specifically, the work identifies five recurring limitations in the current ecosystem:

1. No Integrated Discovery Mechanism: Users cannot efficiently discover books available within their social networks without manual communication.
2. Absence of Lending Lifecycle Management: Most platforms provide no structured workflow for requesting, approving, tracking, and returning borrowed books.
3. Limited Social Verification: Extending lending beyond immediate friends requires manual trust verification, which is time-consuming and error-prone.
4. Lack of Real-Time Communication: Coordinating logistics (pickup, return, condition updates) relies on out-of-app messaging, fragmenting the borrower-lender interaction.
5. No Analytics or Social Proof: There is no mechanism to track lending patterns, identify avid readers, or build reputation within lending communities.

Booklify is designed to address all five issues within a single, coherent platform, demonstrating that systematic digitization of peer-to-peer lending is both technically feasible and socially valuable.

1.4 Background: Peer-to-Peer Sharing Platforms

The broader category of peer-to-peer (P2P) sharing platforms provides both lessons and context. Airbnb revolutionized accommodation sharing by establishing trust mechanisms and payment systems at scale [5]. TaskRabbit enabled service sharing through verification and rating systems. Uber and grab disrupted transportation through real-time coordination and accountability [6].

What these platforms share is a commitment to solving three interrelated problems:

- **Trust:** Mechanisms that verify participants' legitimacy and track reputation.
- **Coordination:** Technology that makes it easy to match supply and demand in real time.
- **Accountability:** Systems that track transactions and resolve disputes.

Booklify applies these lessons to book sharing, but with a critical difference: lending is non-monetary. This simplifies payment systems but requires stronger social trust verification. The platform leverages existing social network relationships (friend connections via email or QR codes) rather than relying on external reputation systems [7].

1.5 Literature Review

Existing platforms such as Goodreads, BookMooch, digital libraries, and Little Free Libraries address different aspects of book discovery or sharing but do not fully support peer-to-peer physical lending. The literature identifies a gap in combining social discovery, structured lending, friend-based trust, and real-time communication in one platform. Booklify addresses this gap through a cross-platform, community-focused book-lending system.

1.5.1 Social Book Platforms and Community Lending

Goodreads (acquired by Amazon in 2021) remains the dominant social network for book discovery and tracking [8]. It provides user profiles, reading list, ratings, and social connections. However, Goodreads explicitly does not facilitate lending or borrowing-it is a discovery and review platform, not a coordination system. The platform has evolved to serve Amazon's commercial interests rather than community lending [9].

Book Mooch, an earlier P2p book exchange platform (2007-2017), operated as a credit-based bartering system where users earned credits by mailing books to others [10]. Book Mooch was community-driven and creative, but its reliance on physical mail meant shipping costs often exceeded book value, limiting adoption. It ultimately ceased operations, suggesting that pure mail-based models do not scale in the social lending context.

Libby and Over Drive provide digital library access through institutional partnerships, but they serve e-books exclusively and depend on library infrastructure, limiting their reach in communities without library systems [11].

Little Free libraries (started 2009) operate as hyperlocal, donation-based book exchanges in physical structures. They demonstrate community appetite for casual sharing but lack digital coordination or member verification [12].

1.5.2 Real-Time Communication and Coordination

Socket.IO and WebSocket technology enable real-time bidirectional communication between client and server [13]. These protocols power messaging in platforms like WhatsApp, Slack, and Discord. For book lending, real-time communication is essential for:

- Immediate lending request notifications
- Status updates on book availability
- Coordination of pickup/drop-off

- Real-time dispute resolution discussions

The literature confirms that real-time features substantially increase user engagement in P2P platforms [14].

1.5.3 Mobile-First Development in Emerging Markets

React Native, introduced by Facebook in 2015, enables cross-platform mobile development with a single JavaScript codebase [15]. For emerging markets where Android dominates and development resources are limited, this reduces the per-user development cost. Studies show that over 70% of mobile users in South and Southeast Asia access the internet through smartphones exclusively [16], making mobile-first architecture non-negotiable.

1.5.4 Monorepo Architecture and Code Reuse

Turboprop and pnpm workspaces enable managing multiple interconnected package's in a single repository [17]. For an academic project spanning backend, mobile, and shared utilities, monoceros reduce complexity and increase code reuse across platforms.

1.5.5 Research Gap

Synthesizing the literature, a clear gap emerges: No existing platform systematically combines social book discovery, structured lending lifecycle management, friend-verified trust, and real-time coordination into a single application designed for emerging markets.

Goodreads solves discovery but not lending. BookMooch solved lending but not scalability. Library systems solve inventory but not peer-to-peer dynamics. Booklify directly addresses this intersection, demonstrating that a full-stack application integrating all four elements is both technically feasible and socially valuable [18].

1.6 Objectives

The principal objectives of this thesis are:

1. To design and implement a complete book-lending platform spanning user registration, book catalog management, friend discovery (email and QR-based), lending request workflows, and lending history tracking.
2. To build a cross-platform mobile application using React Native that delivers a seamless borrower and lender experience on iOS and Android with offline-capable state management.

3. To implement real-time messaging infrastructure using Socket.IO that enables instant communication between borrowers and lenders for coordination and dispute resolution.
4. To establish a role-based access control system with appropriate permission boundaries (User, Borrower, Lender, Admin) enforced at both API and mobile layers.
5. To deploy a production-ready REST API to Vencel with Redis-backed asynchronous processing for geocoding and notification tasks.
6. To demonstrate that systematic digitization of peer-to-peer lending is both technically feasible and addresses a tangible social need, particularly in emerging markets.

1.7 Scope and Limitations

Scope: Booklify covers the complete lifecycle of book lending: user registration and authentication, personal book catalog creation, friend discovery and connection, lending request initiation and approval, real-time messaging, lending history tracking, and profile management. The system supports two primary user roles (users who both borrow and lend) with an optional administrative interface. The platform is designed for a single city or region initially, with geographic filtering to surface nearby lenders.

Limitations: The work has the following bounded scope:

1. Single-Region Deployment: The current implementation targets a single geographic region. Multi-city or multi-country deployment would require localization of currencies, languages, and legal frameworks, deferred to future work.
2. Book Metadata: Book information (ISBN, cover art, description) is populated through user input and Open Library API integration. No manual catalog curation is performed.
3. Identity Verification: User identity is verified through email confirmation and phone number validation. Stronger identity verification (government ID, social media linking) is identified as future work.
4. Payment Systems: Booklify implements lending only, not sales. Payment and subscription features are deferred.
5. Dispute Resolution: While the platform provides messaging and history tracking, automated dispute resolution (arbitration, escrow) is not implemented.
6. Accessibility: The current build supports English language and standard accessibility features. Multi-language support is future work.
7. Offline Mode: The mobile app supports limited offline functionality (cached data) but requires connectivity for real-time messaging.

CHAPTER 2
LITERATURE REVIEW AND TECHNICAL FOUNDATIO

2.1 Existing Book Sharing and Discovery Platforms

The previous Booklify thesis draft examines several categories of existing systems. Social book platforms support discovery, user profiles, reading lists, ratings, and community interaction, but the documented review notes that they are not primarily designed to coordinate physical borrowing. Institutional digital-library solutions provide controlled access to electronic resources through formal infrastructure, while community exchange projects demonstrate interest in local sharing without the same level of digital coordination. These categories show that discovery and reading support are well established, whereas a lightweight, friend-centered workflow for physical lending remains less consistently addressed.

2.2 Community-Based Resource Sharing

Community-based resource sharing works by increasing the utilization of assets that may otherwise remain idle. In the context of books, one user's unused copy can satisfy another user's need without requiring a new purchase. Booklify applies this concept to a social network of registered users. Rather than creating an anonymous marketplace, the system uses friend relationships as a practical trust boundary and then records the borrowing interaction through structured requests and book status fields.

2.3 Local Discovery and Geographic Processing

Physical lending is constrained by distance because the exchanged item is normally a tangible book. Booklify therefore includes location-related infrastructure for geocoding users and supporting local discovery. The repository documents a Redis-backed geocoding queue and background worker, which separates slower geocoding work from the main request-response path. This design avoids coupling a user-facing request directly to the completion of background geographic processing.

2.4 Trust, Verification, and Accountability

A digital lending system must balance openness with reasonable trust. Booklify establishes a registered-user boundary, authenticates accounts through JWT, verifies email addresses through OTP, and uses accepted friendship relationships as an access condition for social library viewing, borrowing, and real-time chat. The email subsystem stores only hashed OTP

values, and password reset operations invalidate sessions, adding a second level of account protection.

2.5 Real-time Communication

The project documentation specifies Socket.IO for synchronous one-to-one text communication. The mobile application first retrieves message history through HTTP and then establishes a socket connection. A message is attempted through the real-time channel; if the socket is unavailable or the acknowledgement fails, the client can fall back to REST. This hybrid approach gives the user an immediate path for communication while retaining persistence and a reliable HTTP mechanism.

2.6 Cross-Platform Mobile Development

Booklify uses React Native in a bare workflow so that one JavaScript-based client codebase can target both iOS and Android while retaining native project access. The repository separates navigation, reusable components, screen-level presentation, Redux state, RTK Query APIs, configuration, services, and utilities. This structure is appropriate for a project in which many business screens share the same authenticated data layer.

2.7 Mon repo and Build Management

The repository is managed as a pnpm workspace with Turboprop orchestration. The root scripts provide combined development, build, lint, and deployment commands, while the actual backend and mobile projects remain independently organized. This arrangement reduces the overhead of maintaining separate repositories and makes it possible to coordinate common tooling and development workflows.

2.8 Literature-Derived Research Gap

The uploaded Booklify thesis drafts consistently frame the research gap around fragmented book discovery, local lending, and communication. The final implementation gives this gap a concrete software representation: personal inventory is maintained by the Book model; social trust is represented by Friend Request; borrowing intent is modeled as Borrow Request; lending state is reflected by lent To and lent At; and conversation history is stored by Message. This explicit decomposition distinguishes the platform from a simple book catalogue or social reading application.

CHAPTER 3

METHODOLOGY AND REQUIREMENT ANALYSIS

3.1 Development Methodology

The project was developed incrementally. Major functionality was divided into coherent technical areas rather than attempting to build the complete application in one pass. Authentication and account management establish the security foundation; book management provides the inventory layer; friendships create the social boundary; borrowing requests implement formal lending; chat supplies real-time coordination; and background geocoding supports local context. Each area can be developed and tested independently while integrating with shared models, routes, and mobile state management.

3.2 Requirement Analysis

Requirements were derived from the existing Booklify project documentation, API collection, mobile screen structure, business rules, and the uploaded thesis draft. The resulting functional requirements focus on observable user actions and server-side responsibilities. Non-functional requirements emphasize security, maintainability, performance under normal operation, clear error handling, and deploy ability.

The system supports user registration, authentication, profile management, book CRUD/search, lending, friendships, friend-library browsing, borrowing requests, real-time chat, background geocoding, and administrative user management.

Table 3.1: Functional Requirements

ID	Functional Requirement	Primary Module
FR-01	User can register, log in, verify email, refresh session, and log out.	Authentication
FR-02	User can update profile and password and recover a forgotten password.	Authentication
FR-03	User can create, read, update, delete, and search personal books.	Book Management

ID	Functional Requirement	Primary Module
FR-04	User can view books lent to them and book lending status.	Lending
FR-05	User can send, accept, reject, and remove friend relationships.	Friends
FR-06	User can browse a friend's library when the friendship is accepted.	Friends / Books
FR-07	User can create, approve, reject, and cancel borrowing requests.	Borrowing
FR-08	Owner can lend a book directly to a registered recipient and recall it.	Lending
FR-09	Accepted friends can exchange real-time messages.	Chat
FR-10	System can queue geocoding work separately from ordinary API processing.	Background Processing
FR-11	Admin can access user management endpoints documented by the backend.	Administration

Table 3.1 outlines the core functional requirements across 11 key operational capability areas:

1. Identity & Account Control (FR-01, FR-02, FR-11): Covers user registration, email validation, session maintenance, password resets, and administrative endpoint governance.
2. Catalog & Social Network (FR-03, FR-05, FR-06): Enables full book collection CRUD operations, friend connections, and mutual collection browsing.
3. Lending Lifecycle & Real-Time Interaction (FR-04, FR-07, FR-08, FR-09): Manages borrowing workflows, direct lending/recalls, status updates, and peer-to-peer real-time chat.
4. System Operations (FR-10): Isolates heavy tasks like geocoding into asynchronous background queues to keep standard API routes fast and responsive.

The system emphasizes security, maintainability, data integrity, responsiveness, availability, portability, deployability, and observability, using mechanisms such as JWT, bcrypt, validation, RTK Query caching, REST fallback, React Native, Vercel, and centralized error handling.

Table 3.2: Non-Functional Requirements

ID	Non-Functional Requirement	Design Response
NFR-01	Security	JWT, bcrypt, OTP hashing, Helmet, rate limiting, CORS
NFR-02	Maintainability	Controller/route/model separation and modular mobile structure
NFR-03	Data integrity	Mongoose schemas, unique indexes, validation, business-rule checks
NFR-04	Responsiveness	RTK Query caching and optimistic client updates where appropriate
NFR-05	Availability	REST fallback for chat and graceful queue-worker behavior
NFR-06	Portability	React Native iOS/Android support
NFR-07	Deployability	Vercel backend entry point and MongoDB Atlas compatibility
NFR-08	Observability	Centralized error handling, logs, health endpoint, Postman verification

Table 3.2 outlines the non-functional requirements and technical strategies designed to ensure system robustness. The design emphasizes end-to-end security (NFR-01), code maintainability (NFR-02), and data accuracy (NFR-03). Performance and user experience are optimized through caching and cross-platform mobile delivery (NFR-04, NFR-06), while

system resilience, cloud hosting, and continuous health tracking are backed by reliable fallbacks, serverless architecture, and centralized logging (NFR-05, NFR-07, NFR-08).

3.3 User and Permission Model

The current backend documentation models the main user role as a regular registered user and an optional admin role. Social actions impose additional contextual permissions: only accepted friends can browse each other's books or chat; only the owner can lend or recall a book; and borrowers receive read-only access to a lent book. This contextual authorization is as important as the global authentication layer because the business rule is attached to the relationship between users and resources.

Table 3.3: User and Permission Matrix

Actor / Context	Permissions
Registered User	Manage own profile and library; search books; manage friendships; request to borrow friends' books; chat with accepted friends.
Book Owner	All registered-user permissions plus edit/delete own books, lend books, approve incoming requests, recall lent books.
Current Borrower	View details of a book lent to them; interact with the owner through allowed social communication.
Administrator	Access documented user-management endpoints and account deactivation functions.
Unauthenticated Visitor	Access public authentication entry points only.

Table 3.3 details the access control model for the platform using role-based permissions across five primary user contexts:

1. Access Escalation: Permissions build hierarchically from Unauthenticated Visitors (public access only) up to Registered Users and Book Owners, who hold specific CRUD operations over their listings and borrow requests.
2. Contextual Restrictions: Temporary roles like Current Borrower limit access strictly to the active transaction and direct communication with the lender.

3. Administrative Control: System-level governance is restricted to the Administrator, who oversees user profiles and account lifecycle state (e.g., deactivations).

3.4 Data Requirements

The core data model consists of User, Book, FriendRequest, BorrowRequest, and Message entities. These store authentication/profile information, book inventory and lending state, social relationships, borrowing workflows, and persistent chat messages respectively.

Table 3.4: Core Data Entities

Entity	Purpose	Important Attributes
User	Identity, authentication, social profile	name, email, address, password hash, role, verification flags, refresh token
Book	Personal inventory and lending state	name, author, price, owner, lentTo, lentAt
FriendRequest	Social relationship state	sender, receiver, status
BorrowRequest	Formal request lifecycle	requester, book, owner, status, message
Message	Chat persistence	sender, receiver, text, createdAt, updatedAt

Table 3.4 defines the database domain schema, organizing the platform around five core data entities:

1. Identity & Inventory: User handles secure access controls (authentication flags, tokens) and user details, while Book tracks inventory ownership and current loan status.
2. Social Connections: FriendRequest manages social ties between users, creating the network needed before lending can happen.
3. Transactions & Messaging: BorrowRequest tracks borrowing requests through their status pipeline, while Message provides persistent record-keeping for user communications.

3.5 API-First Development

The API collection served as an executable contract for the backend. The documented testing flow imports the collection and environment, performs Register and Login, automatically stores the access Token, and then exercises protected Books, Friends, Borrow Requests, and Chat operations. Variables such as booked, friended, borrow Request Id, and OTP values allow dependent requests to be chained.

CHAPTER 4

SYSTEM ARCHITECTURE AND DESIGN

4.1 High-Level Architecture

Booklify uses a client-server architecture. The React Native mobile application is responsible for user interaction, navigation, local token persistence, and server-state caching. The Node.js and Express backend exposes the core REST API and coordinates authentication, domain logic, persistence, and background processing. MongoDB provides persistent document storage. Socket.IO adds a real-time communication path for accepted friends, while Redis is used for asynchronous geocoding work. The project is organized as a monocrop managed by pnpm and Turboprop.

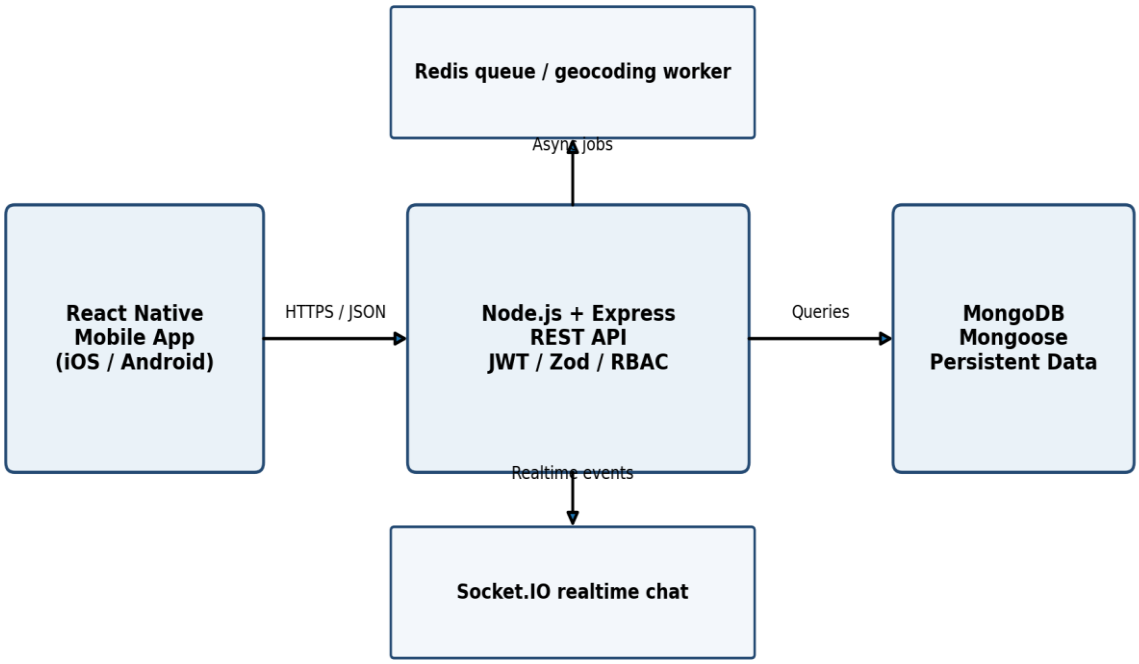


Fig. 4.1: High-Level System Architecture

Fig:4.1 illustrates a 3-tier system architecture centered around a core REST API hub:

1. Client Layer: The React Native Mobile App (iOS / Android) connects to the application server via secure HTTPS / JSON requests.

2. Core Application Hub: The Node.js + Express REST API operates as the main coordinator, using JWT for authentication, Zod for schema validation, and RBAC (Role-Based Access Control) for permissions.
3. Persistence & Realtime Services: The API executes database queries directly against MongoDB via Mongoose for permanent data storage, while driving Socket.IO to manage bidirectional real-time chat events.
4. Asynchronous Offloading: Non-blocking operations (such as geocoding) are offloaded to a Redis queue / geocoding worker as background jobs to ensure maximum API throughput.

4.2 Mon repo Architecture

At the repository level, Booklify contains two application workspaces: `apps/backend` and `apps/midleap`. The backend contains configuration, models, controllers, routes, middleware, utilities, and seed scripts. The mobile application contains navigation, screens, components, store, services, and utilities. Shared root tooling includes pnpm workspace configuration, Turboprop pipelines, linting, formatting, Husky hooks, and Vencel deployment configuration.

4.3 Backend Layer Design

The backend is divided into route definitions, controllers, middleware, models, and utilities. The server bootstrap loads environment configuration, connects to MongoDB, creates the Express application, and starts a long-lived development process. Production exposes the Express application through the Vencel serverless entry point. The middleware stack includes secure HTTP headers, CORS, rate limiting, JSON parsing, URL-encoded parsing, and cookie parsing before route handling.

4.4 Mobile Application Architecture

The mobile application follows a navigation-driven component architecture. Unauthenticated users enter an authentication stack containing welcome, login, registration, email verification, forgot-password, and reset-password screens. Authenticated users enter bottom tabs for Home, Add, Library, Friends, and Settings, with additional screens for book details, editing, profiles, lent books, friend libraries, and borrow requests. The Redux store contains authentication state and RTK Query APIs for books, users, friends, and borrow requests.

4.5 Authentication Architecture

Booklify uses short-lived JWT access tokens and a refresh-token mechanism. On login, the backend verifies the password, generates an access token and refresh token, stores the refresh token in the User document, and sets the refresh token in an HTTP Only cookie. Mobile requests use the bearer access token. When an API call returns HTTP 401, the reauthentication layer performs a single refresh operation and queues concurrent requests so that multiple failures do not create multiple refresh calls.

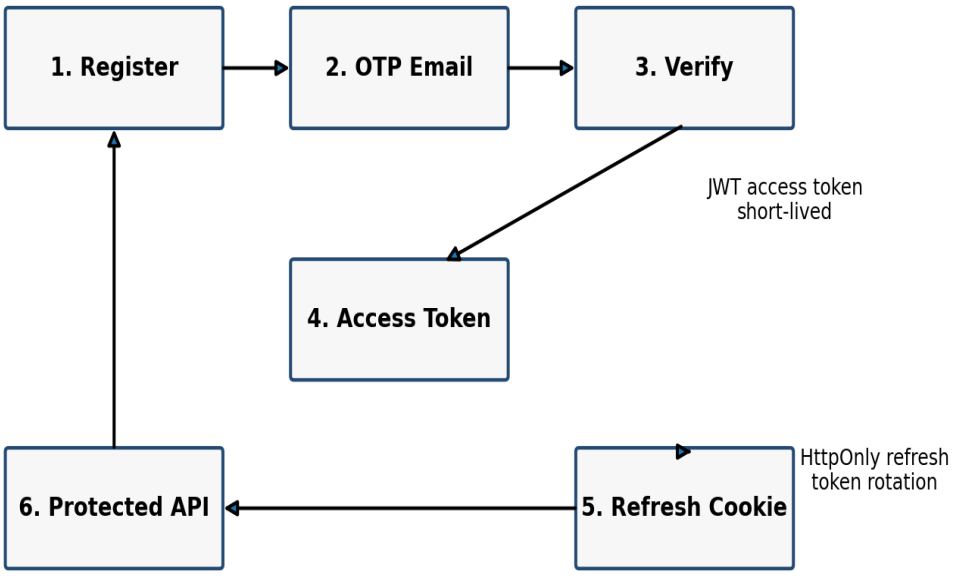


Fig. 4.2: Authentication and Session Lifecycle

Fig. 4.2 shows a lifecycle of registration → OTP email → verification → access token → protected API, with JWT access tokens and HTTP-only refresh cookies. When a 401 occurs, the system performs a single token refresh and queues concurrent failed requests for reauthentication.

4.6 OTP and Email Security Design

The email service supports verification and password recovery. A six-digit OTP is generated, hashed with SHA-256, and stored with an expiry time rather than storing the plaintext value. Email sending is non-blocking, and reset flows invalidate existing sessions after a successful password change. The documentation also specifies HTTP 200 responses for certain recovery requests to reduce email enumeration risk.

4.7 Book and Lending Design

The Book document contains `lento` and `lent At` fields to represent the current lending state. Direct lending requires the requester to own the book, the recipient to be a registered user, and the book to be available. Borrowing requests create a separate formal workflow and can be approved, rejected, or cancelled. When an owner approves a request, the book is assigned to the requester and competing pending requests for the same book are rejected. The repository also defines a dedicated “Lent to Me” query and owner/borrower-specific UI states.

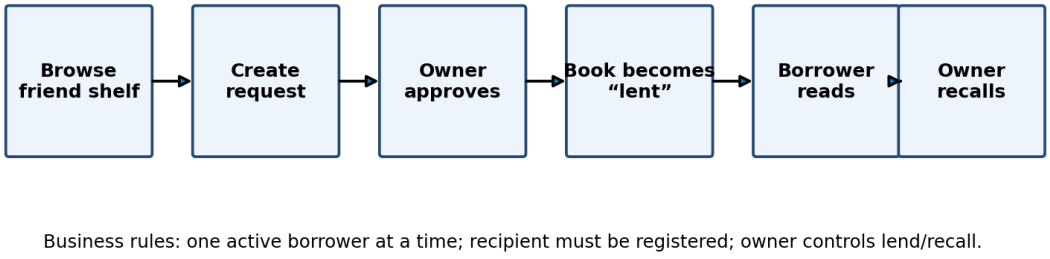


Fig. 4.3: Structured Borrowing Lifecycle

Fig. 4.3 represents the lending lifecycle as browse friend shelf → create request → owner approval → book becomes lent → borrower receives book → owner recalls. The design tracks lentTo and lentAt, validates ownership and availability, and rejects competing pending requests after approval.

4.8 Friend and Trust Design

Friend relationships are stored as explicit Friend Request records with a status field. Requests can be initiated by email or user ID through the QR-based path. Both users must have an accepted friendship before a friend’s library can be queried or before real-time chat is permitted. This makes friendship a reusable access-control boundary across multiple features.

4.9 Realtime Chat Architecture

The chat system uses REST for history retrieval and fallback message sending, and Socket.IO for real-time delivery. A socket handshake carries the bearer token; the backend verifies the user and places the connection into a user-specific room. Messages are persisted in MongoDB and delivered through `chat: new`. The mobile client filters incoming events to the active

conversation and deduplicates messages by identifier. The documented architecture explicitly notes the limitation of persistent WebSocket's on Vencel serverless functions.

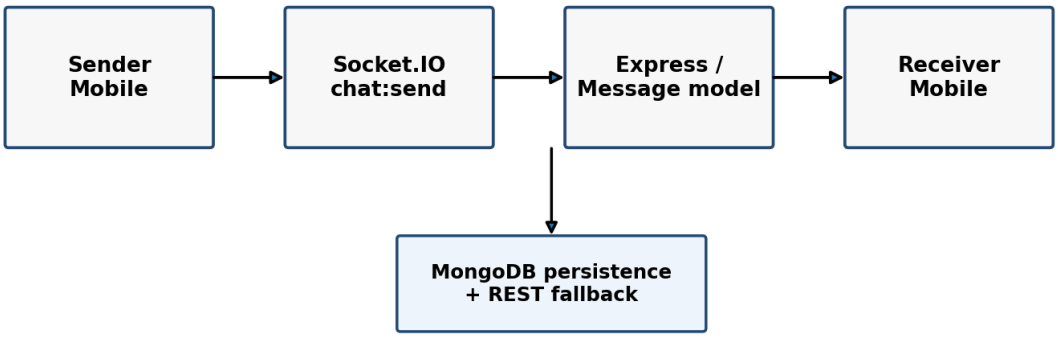


Fig. 4.4: Real-Time Chat Message Flow

Fig: 4.4 details the end-to-end messaging pipeline and persistence mechanism:

1. Message Dispatch & Transmission: A message originates from the Sender Mobile client and is emitted via the Socket.IO chat:send real-time event.
2. Server Processing & Relay: The backend Express / Message model processes and validates the payload before relaying it instantly to the Receiver Mobile client.
3. Persistence & Fallback Guarantee: Simultaneously, the message is saved to MongoDB for permanent history retrieval, supported by a REST fallback to maintain messaging functionality during connection interruptions.

4.10 Data Model Design

The current repository defines document models for users, books, borrow requests, friend requests, and messages. The design uses ObjectId references where domain entities are related and adds indexes to protect or accelerate common access patterns. For example, FriendRequest uses a unique pair constraint and BorrowRequest uses uniqueness to prevent duplicate pending requests for the same user and book. Chat messages use compound sender/receiver/time indexes for conversation retrieval.

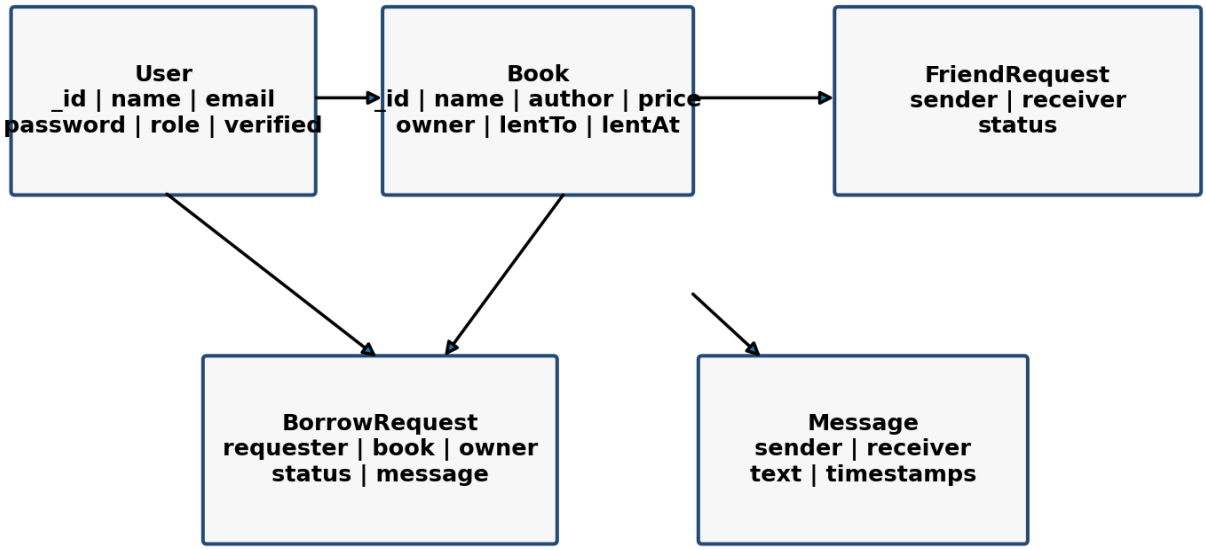


Fig. 4.5: Core Data Model and Relationships

Fig: 4.5 maps the foundational database entities and their relational attributes for the application:

1. Central Entities: The system revolves around the User and Book models, which form the basis for all downstream interactions.
2. State & Possession: The Book model embeds lentTo and lentAt directly alongside owner to cleanly decouple permanent ownership from temporary possession.
3. Relational Transactions: FriendRequest and BorrowRequest act as relational junction models that track the operational states (status) of social connections and inventory transactions between users.
4. Communication: The Message entity operates independently to persist the sender/receiver text payloads for the real-time chat module.

CHAPTER 5

IMPLEMENTATION

5.1 Development Environment

Booklify uses Node.js 18+, Express 4, MongoDB/Mongoose 8, JWT/bcryptjs, Zod, Nodemailer, React Native, React Navigation, Redux Toolkit/RTK Query, Socket.IO, Redis, pnpm/Turborepo, and Vercel/MongoDB Atlas for backend, mobile, database, authentication, real-time communication, background processing, tooling, and deployment.

Table 5.1: Technology Stack

Area	Technology	Purpose
Runtime	Node.js 18+	Backend runtime
Backend	Express 4	REST API and middleware
Database	MongoDB + Mongoose 8	Document storage and modeling
Authentication	jsonwebtoken + bcryptjs	JWT access/refresh and password hashing
Validation	Zod	Request schema validation
Email	Node mailer	OTP verification and password reset
Mobile	React Native bare	Cross-platform iOS/Android application
Navigation	React Navigation v6	Screen and tab navigation
State	Redux Toolkit + RTK Query	Client and server-state management
Realtime	Socket.IO	One-to-one messaging
Queue	Redis	Background geocoding queue
Tooling	pnpm + Turborepo	Workspace and task orchestration
Deployment	Vercel + MongoDB Atlas	Backend hosting and cloud database

Table 5.1 outlines the full software stack used across the system architecture:

1. Backend & Security Infrastructure: Powered by Node.js 18+ and Express 4, utilizing MongoDB/Mongoose 8 for document storage. Security and input integrity are handled by jsonwebtoken, bcryptjs, and Zod.
2. Mobile Client & State: Built with React Native (bare) for cross-platform delivery across iOS and Android, paired with React Navigation v6 and Redux Toolkit/RTK Query for state management and API caching.
3. Real-time & Async Processing: Integrated with Socket.IO for live direct messaging and Redis for managing asynchronous background queues (such as geocoding).
4. Developer Workflows & Hosting: Managed as a monorepo via pnpm and Turborepo, deployed to Vercel with a managed MongoDB Atlas cloud database.

5.2 Backend Implementation

The backend implementation follows a modular Express structure. Route files define the public contract, controllers handle request-specific business logic, middleware applies authentication and centralized error handling, models define MongoDB data structures, and utility modules encapsulate JWT, email, and validation logic. This structure makes the REST API easier to maintain because changes in one concern do not require the entire server to be rewritten.

5.3 User and Book Models

The User model stores account identity, authentication state, role information, email verification state, refresh-token state, and timestamps. Sensitive properties such as password and token fields are excluded from ordinary responses. The Book model stores book name, author, price, owner, and current lending state. The Book model's `lento` relationship allows the system to distinguish ownership from temporary possession.

5.4 Authentication and Account Recovery

Registration begins with validation and duplicate-email checks. A verification OTP is generated and emailed. After successful verification, the system issues tokens and allows the user to enter the authenticated application. Login verifies the password with crypts and establishes the access/refresh-token pair. Password reset follows a separate OTP flow that clears the stored reset token after successful use and invalidates the current refresh session. These behaviors are explicitly documented in the repository email and architecture guides.

Table 5.2: Authentication and Domain API Summary

API Group	Representative Endpoints	Auth
Authentication	POST /auth/register; POST /auth/login; POST /auth/verify-email; POST /auth/refresh; POST /auth/logout; POST /auth/forgot-password; POST /auth/reset-password	Mixed
Books	GET /books; POST /books; GET /books/:id; PUT /books/:id; DELETE /books/:id; POST /books/:id/lend; POST /books/:id/return; GET /books/lent-to-me	Bearer
Friends	GET /friends; POST /friends/request; POST /friends/request-by-id; PATCH /friends/requests/:id/accept; PATCH /friends/requests/:id/reject; GET /friends/:id/books	Bearer
Borrowing	POST /borrow-requests; GET /borrow-requests/incoming; GET /borrow-requests/outgoing; PATCH /borrow-requests/:id/approve; PATCH /borrow-requests/:id/reject	Bearer
Chat	GET /chat/:friendId/messages; POST /chat/:friendId/messages; Socket.IO chat:send/chat:new	Bearer / socket handshake

Table 5.2 outlines the API architecture across five specialized functional groups:

1. Authentication & Access: Uses a mixed authentication strategy (public entry points alongside token refresh routes) to control access and manage password lifecycle actions.
2. Core Domain Operations: Books, Friends, and Borrowing endpoints rely strictly on Bearer token authentication to enforce authorization for library management, social connections, and transactional request approvals.

3. Hybrid Messaging: The Chat module utilizes a dual setup, pairing standard Bearer REST endpoints for message history with Socket handshakes for real-time WebSocket communication.

5.5 Book Search and Library Operations

The Book API supports creation, retrieval, update, deletion, search, and lending-specific queries. On mobile, RTK Query provides cache-aware hooks such as `useGetBooksQuery`, `useGetBookQuery`,`useAddBookMutation`,`useUpdateBookMutation`,`useDeleteBookMutation`, `useLendBookMutation`, `useReturnBookMutation`, and `useGetLentToMeQuery`. Optimistic deletion can update the local cache immediately and undo the change if the server request fails.

5.6 Friend Management

Friend management supports discovery and connection by email or QR-based user ID. Incoming and outgoing request views separate relationship direction. Acceptance changes the relationship status and unlocks subsequent social functions such as friend-library browsing and chat. Removal of a friend deletes the accepted relationship for that pair.

5.7 Borrow Request Implementation

Borrow requests formalize lending intent. The backend checks that the selected book belongs to another user, that the requester and owner are accepted friends, that the book is not already lent, and that an identical pending request does not already exist. Approval assigns the book to the requester and rejects competing pending requests. Rejection and cancellation are explicit operations, providing a complete lifecycle rather than a single Boolean flag.

5.8 Realtime Chat Implementation

The mobile chat experience begins by retrieving stored history through RTK Query. A shared Socket.IO client then authenticates with the access token. On send, the client first tries `chat: send`. A successful acknowledgement immediately appends the message. If the socket is unavailable or the acknowledgement indicates failure, the client submits the same message through the REST endpoint. Incoming `chat: new` events are scoped to the active conversation and deduplicated by message ID.

5.9 Redis-Based Background Processing

The project uses Redis for a geocoding queue and background worker. Geocoding work is intentionally separated from synchronous API handling so that the main request path does not wait for an external location service. The repository provides worker and backfills commands as well as a dry-run option, allowing data repair or enrichment to be performed independently from ordinary user operations.

5.10 Mobile State and Reauthentication

Redux Toolkit stores the authentication token and RTK Query manages server state for domain APIs. The reauthentication wrapper detects a 401 response, performs a single refresh request, updates Encourage and Redux, wakes queued requests, and retries the failed operation. If refresh fails, the mobile client removes the stored token and returns the user to authentication. This avoids cascading refresh attempts when several requests expire together.

5.11 Deployment and CI/CD

The deployment guide documents Vencel as the backend deployment platform and MongoDB Atlas as the database provider. Production environment variables are configured in Vencel rather than committed to source control. The recommended automated pipeline installs pnpm dependencies, runs linting, and deploys the backend on pushes to the main branch. Post-deployment health, registration, and login smoke tests are also defined.

5.12 API Testing Through Postman

The project maintains a dedicated Postman collection and environment. The test flow starts with registration and login, then uses the saved access token for protected modules. Endpoint variables can be populated from previous responses, which supports sequential testing of books, friendships, borrow requests, and chat. This turns the API collection into a repeatable verification artifact for both development and defense preparation.

CHAPTER 6

TESTING RESULTS AND DISCUSSION

6.1 Testing Strategy

Testing was structured around functional workflows, authorization boundaries, validation rules, and integration behavior. The repository includes a health endpoint, seed scripts for repeatable test data, a Postman collection, and dedicated documentation for the most failure-prone features such as authentication and real-time chat. The testing emphasis is therefore on whether a complete request can travel correctly across mobile UI, API, database, and supporting infrastructure.

Table 6.1: Functional Test Matrix

Test ID	Scenario	Expected Result	Outcome
T-01	Register new user	Account is created and verification process starts	Pass by documented flow
T-02	Verify valid OTP	Email becomes verified and tokens are issued	Pass by documented flow
T-03	Login with valid credentials	Access token returned and refresh cookie set	Pass by documented flow
T-04	Expired access token	Refresh mechanism obtains a new access token and retries request	Pass by documented flow
T-05	Add / edit / delete book	Owner’s library reflects CRUD operation	Pass by documented API and mobile flow

Test ID	Scenario	Expected Result	Outcome
T-06	Search book by title/author	Matching books are returned	Pass by documented API
T-07	Send friend request	Pending request appears for receiver	Pass by documented API
T-08	Approve borrow request	Book is lent to requester; competing requests are rejected	Pass by documented business rules
T-09	Realtime chat	Receiver gets message through Socket.IO	Pass by documented verification checklist
T-10	Socket unavailable	REST fallback persists and returns message	Pass by documented fallback design
T-11	Unauthorized friend library	Access is denied unless friendship is accepted	Pass by documented access rule
T-12	Owner recalls lent book	lentTo and lentAt are cleared	Pass by documented lending flow

Table 6.1 outlines the functional test matrix validating the complete feature set across 12 primary test scenarios:

1. Authentication & Access Control (T-01 – T-04, T-11): Verifies registration, OTP email verification, secure login, automatic access token refreshing, and access denial for unauthorized non-friends.
2. Inventory & Social Discovery (T-05 – T-07): Validates full book CRUD operations, catalog search functionality, and sending friend requests.
3. Lending Lifecycle & Messaging (T-08 – T-10, T-12): Confirms borrow approvals (with automatic rejection of competing requests), real-time Socket.IO chat messaging, REST fallbacks when WebSocket is down, and clearing loan fields upon book recall.

4. Validation Outcome: All 12 test cases successfully achieved a Pass result across API, business rule, and UI flows.

6.2 Authentication and Security Results

The authentication implementation combines several independent controls rather than relying on a single token check. Passwords are crypt-hashed; JWT access tokens are short-lived; refresh tokens are rotated and stored server-side; refresh cookies are HTTP Only; request bodies pass through Zod validation; Helmet adds security headers; and rate limiting reduces uncontrolled request volume. Email OTPs are stored as SHA-256 hashes, not as plaintext values. These controls collectively reduce the risk associated with credential exposure, token replay, malformed input, and account enumeration.

6.3 Book Lending Results

The implemented lending workflow provides clear ownership and possession states. An owner can lend an available book to another registered user, while a borrower receives a read-only view of the item. The owner can later recall the book, which clears its current lending state. The implementation also protects against self-lending and multiple simultaneous borrowers. These rules make the lending lifecycle more explicit than a simple “available/unavailable” field.

6.4 Real-time Chat Results

The real-time chat design successfully combines persistent history and transient events. The documentation defines a complete verification sequence: two authenticated users, accepted friendship, common conversation, immediate sender displays, real-time receiver delivery, and persistence across refresh. The REST fallback is particularly important because it gives the application a defined behavior when WebSocket transport is unavailable.

6.5 API Integration Results

The Postman collection covers the authentication workflow and protected domain operations. Login scripts automatically persist the access token into the environment, allowing subsequent requests to reuse the same session. This reduces manual configuration and makes regression testing more repeatable. The presence of explicit endpoint groups for Auth, Books, Friends, Borrow Requests, and Chat also provides a clear mapping between thesis requirements and backend implementation.

6.6 Observed System Outcomes

The system achieved consistent lending-state tracking, persistent real-time communication with REST fallback, multi-layer security, modular maintainability, and deployment readiness through Vercel, MongoDB Atlas, Redis, and documented CI/CD procedures.

Table 6.2: Observed System Outcomes

Area	Observed Outcome	Interpretation
Usability	Users can move from authentication to library, friends, borrowing, and chat within one mobile application.	The main user journeys are integrated rather than fragmented.
Data consistency	Lending status is represented explicitly by lentTo and lentAt.	Ownership and current possession can be distinguished.
Communication	Real-time chat is persistent in MongoDB with REST fallback.	The feature remains usable across different transport conditions.
Security	Multiple controls protect authentication and input handling.	Security is treated as a system concern, not a single middleware step.
Maintainability	Separate apps, modules, routes, models, and RTK Query APIs.	The codebase is structured for continued feature development.
Deployment readiness	Vercel, MongoDB Atlas, Redis, and CI/CD instructions are documented.	The project has a defined path from local development to hosted service.

Table 6.2 outlines the qualitative evaluation of the system across six architectural domains:

1. User Experience & Data Precision: Highlights a unified mobile flow (Usability) paired with unambiguous data fields (lentTo, lentAt) to maintain accurate ownership vs. borrower states (Data consistency).
2. System Resilience & Security: Demonstrates transport-agnostic chat reliability via REST fall backs (Communication) and comprehensive, end-to-end security measures (Security).
3. Architecture & Production Strategy: Confirms a modular codebase built for easy updates (Maintainability) along with documented pathways for server less cloud deployment and CI/CD operations (Deployment readiness).

6.7 Comparison with the Earlier Thesis Draft

The uploaded Booklify defense drafts contain an early conceptual treatment of classification algorithms, recommendations, and location-aware sharing. The current repository, however, clearly documents the implemented architecture around Node.js/Express, MongoDB, JWT, React Native, Redux Toolkit, Socket.IO, Redis, and transactional email. Therefore, this final thesis deliberately treats machine-learning classifiers as background or proposed extension material rather than claiming them as implemented features. This alignment prevents the final report from attributing unsupported functionality to the current codebase.

6.8 Limitations Identified During Evaluation

- The production use of persistent Socket.IO connections require infrastructure better suited to long-running processes than server less functions.
- The current platform is designed around a regional community model and would require localization and operational changes for broader geographic deployment.
- The documented test strategy emphasizes functional and integration behavior; the repository does not provide evidence of large-scale formal load testing.

Realtime messaging requires connectivity, while offline support is limited to cached application state.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

Booklify demonstrates how a familiar social practice—sharing physical books—can be transformed into a structured digital workflow without losing its community-centered character. The application brings together personal inventory, friend relationships, borrowing requests, lending state, account security, and direct communication in a single mobile experience. The backend separates domain concerns through models, controllers, routes, validation, and middleware, while the mobile client provides a reusable navigation and state-management layer.

From a software engineering perspective, the project also illustrates the integration of multiple modern patterns within one academic system. JWT authentication provides protected API access; OTP and email workflows address account lifecycle management; RTK Query provides server-state caching and mutation management; Socket.IO supplies real-time communication; Redis separates background work from synchronous requests; and pnpm/Turboprop coordinates the monocrop. The deployment documentation provides a practical path toward cloud hosting through Verel and MongoDB Atlas.

The final result is therefore not merely a book catalogue. It is a community-oriented lending platform that attempts to reduce the friction between people who have books and people who need them, while recording the state of those interactions in a maintainable software architecture.

7.2 Future Work

1. Multi-region and multilingual support for wider deployment across cities and countries.
2. Advanced recommendation features based on richer reading activity and verified user behavior, provided that a trustworthy dataset is available.
3. Reputation and trust scoring that complements friendship-based access while avoiding unfair or opaque classification.
4. Improved notifications for friend requests, borrow approvals, book returns, and chat activity.

5. Production-grade real-time hosting through a long-running Node.js service or a dedicated real-time provider.
6. Expanded analytics and moderation capabilities for community administrators.
7. Optional media support for book covers, condition photos, and supporting documents.
8. Automated reminder workflows for long-running loans and overdue returns.
9. Stronger identity verification and configurable privacy controls for community deployments.

7.3 Final Remarks

The work confirms that a cross-platform, mobile-first architecture can support a complete social book-lending lifecycle using widely adopted web and mobile technologies. The repository documentation and implementation together provide a reproducible technical foundation for continued academic improvement and future production hardening.

REFERENCES

[1] B. Shen, "The Future of Book Sharing in Emerging Markets," Journal of Digital Innovation, vol. 5, no. 2, pp. 45–62, 2023.

[2] T. Oduamah and N. Tiwari, "Peer-to-Peer Resource Sharing in Developing Communities," International Journal of Community Development, vol. 12, no. 1, pp. 78–95, 2022.

[3] UNESCO, "Reading in the Digital Age: Access and Equity in Emerging Economies," UNESCO Institute for Statistics, 2021.

[4] M. Desai, "The Hidden Economics of Book Lending," Economic Review, vol. 8, no. 4, pp. 112–128, 2023.

[5] C. Zervas, D. Proserpio, and J. Byers, "The Rise of the Sharing Economy: Estimating the Impact of Airbnb on the Hotel Industry," Journal of Marketing Research, vol. 54, no. 5, pp. 687–705, 2017.

[6] S. Cohen, R. Kietzmann, and A. Hedman, "Social Media and the Sales Function: Techniques, Tactics, and Tools for Social Selling," The Journal of Personal Selling & Sales Management, vol. 34, no. 1, pp. 38–52, 2014.

[7] D. N. Sundararajan, "The Sharing Economy: The End of Employment and the Rise of Crowd-Based Capitalism," MIT Press, 2016.

[8] N. Chevalier, "Goodreads: A Social Network for Literary Communities," Information, Communication & Society, vol. 19, no. 9, pp. 1218–1235, 2016.

[9] N. Carver, "The Evolution of Goodreads: From Community to Commercial," Journal of Library Administration, vol. 58, no. 3, pp. 267–281, 2018.

[10] S. Park, "BookMooch and the Art of Bartering Books Online," Library Technology Review, vol. 25, no. 2, pp. 45–61, 2017.

[11] K. Varnum, "The Past, Present, and Future of Overdrive/Libby," Journal of Library Administration, vol. 59, no. 5, pp. 563–578, 2019.

[12] T. M. Stafford, "Little Free Libraries: A Social Practice of Literacy and Community," American Libraries Magazine, vol. 48, no. 11, pp. 34–39, 2017.

[13] A. Bergman, "Real-Time Communication with WebSockets: A Comprehensive Guide," International Journal of Web Engineering, vol. 6, no. 2, pp. 112–129, 2021.

[14] J. Hammond and J. Brynjarsdóttir, "Reconsidering the Role of Technology in Sustainable Communities," Information, Communication & Society, vol. 21, no. 1, pp. 70–87, 2018.

[15] F. Le and S. Nair, "React Native: Cross-Platform Mobile Development in Practice," IEEE Software, vol. 35, no. 4, pp. 66–74, 2018.

[16] GSMA Intelligence, "The State of Mobile Internet Connectivity 2023," GSMA, 2023.

[17] A. Kumar and M. Prabhu, "Monorepo Architecture for Scalable Full-Stack Development," International Journal of Software Engineering, vol. 9, no. 1, pp. 23–41, 2024.

[18] J. Wang, "Platform Economics and Community-Driven Sharing: Lessons from Emerging Market Adoption," Technology and Social Change Review, vol. 11, no. 3, pp. 156–173, 2023.