* ALHATE DIPAK BAPU .
* Software Engineering
* M.Tech BITS PILANI
* Roll No. 2023TM93604
* Full Stack Application

BookExchange Platform

Contents

[1. Project Overview 3](#_Toc182521886)

[1.1 Purpose & Objectives 3](#_Toc182521887)

[1.2 Scope 3](#_Toc182521888)

[1.3 Target Audience 3](#_Toc182521889)

[2. Functional Requirements 4](#_Toc182521890)

[2.1 User Stories 4](#_Toc182521891)

[2.1.1 User Story 1: User Authentication 4](#_Toc182521892)

[2.1.2 User Story 2: Book Listing 4](#_Toc182521893)

[2.1.3 User Story 3: Book Search 4](#_Toc182521894)

[2.1.4 User Story 4: Exchange Requests 4](#_Toc182521895)

[2.1.5 User Story 5: Transaction Management 5](#_Toc182521896)

[2.2 Features & Functionality 5](#_Toc182521897)

[2.3 User Roles & Permissions 5](#_Toc182521898)

[3. Non-Functional Requirements 6](#_Toc182521899)

[3.1 Performance 6](#_Toc182521900)

[3.2 Security 6](#_Toc182521901)

[3.3 Scalability 6](#_Toc182521902)

[3.4 Usability 6](#_Toc182521903)

[3.5 Reliability & Availability 6](#_Toc182521904)

[3.6 Maintainability 6](#_Toc182521905)

[3.7 Compliance 6](#_Toc182521906)

[4. 4. Architecture Overview 7](#_Toc182521907)

[4.1 System Architecture 7](#_Toc182521908)

[4.2 Frontend Architecture 7](#_Toc182521909)

[4.3 Backend Architecture 7](#_Toc182521910)

[4.4 Database Design 7](#_Toc182521911)

[4.5 Data Flow 8](#_Toc182521912)

[5. UI/UX Design 9](#_Toc182521913)

[5.1 Wireframes/Mockups 9](#_Toc182521914)

[5.2 Navigation Flow 9](#_Toc182521915)

[5.3 Design Guidelines 9](#_Toc182521916)

[5.4 Responsive Design 9](#_Toc182521917)

[6. API Design 0](#_Toc182521918)

[6.1 API Endpoints 0](#_Toc182521919)

[6.2 Authentication & Authorization 0](#_Toc182521920)

[6.3 Rate Limiting & Throttling 0](#_Toc182521921)

[6.4 Error Handling 0](#_Toc182521922)

[6.5 Versioning 0](#_Toc182521923)

[7. Implementation Demo Deck 1](#_Toc182521924)

[9. Deployment & Hosting 2](#_Toc182521925)

[7.1 Environment Setup 2](#_Toc182521926)

[7.2 Hosting Infrastructure 2](#_Toc182521927)

[7.3 Containerization & Orchestration 2](#_Toc182521928)

[7.4 CI/CD Pipeline 2](#_Toc182521929)

[7.5 Load Balancing & Caching 2](#_Toc182521930)

[7.6 Monitoring & Logging 2](#_Toc182521931)

[7.7 Backup & Recovery 2](#_Toc182521932)

[10. Testing Plan 3](#_Toc182521933)

[8.1 Unit Testing 3](#_Toc182521934)

[8.2 Integration Testing 3](#_Toc182521935)

[8.3 End-to-End Testing 3](#_Toc182521936)

[8.4 Performance Testing 3](#_Toc182521937)

[8.5 Security Testing 3](#_Toc182521938)

[11. Security Considerations 4](#_Toc182521939)

[9.1 Authentication & Authorization 4](#_Toc182521940)

[9.2 Data Protection 4](#_Toc182521941)

[9.3 Secure Data Storage 4](#_Toc182521942)

[9.4 Common Vulnerabilities 4](#_Toc182521943)

[9.5 Logging & Monitoring 4](#_Toc182521944)

[12. Project Timeline & Milestones 5](#_Toc182521945)

[10.1 Development Timeline 5](#_Toc182521946)

[10.2 Key Milestones 5](#_Toc182521947)

[10.3 Resource Allocation 5](#_Toc182521948)

[13. Risk Management 6](#_Toc182521949)

[11.1 Risk Identification 6](#_Toc182521950)

[11.2 Mitigation Strategies 6](#_Toc182521951)

[11.3 Contingency Plans 6](#_Toc182521952)

[14. 12. Future Enhancements 7](#_Toc182521953)

[12.1 Roadmap for New Features 7](#_Toc182521954)

[12.2 Scalability Considerations 7](#_Toc182521955)

**Web Application Design Document**

# Project Overview

## 1.1 Purpose & Objectives

Scope of this document is design and detail documentation on architectural design of Book Exchange Application Platform.

This web application is designed as a **Book Exchange Application** where users can:

- sign up,

- log in, and

- access various book-related services.

The goal is to create a secure, scalable platform that allows users to exchange books and build connections/networking with people from your region.

## 1.2 Scope

* Core Features:
  + User Authentication
    - (Sign up, Login, Forgot Password. Token Authentication & Authorisation)
  + Book Exchange Functionality
  + REST API Integration (Flask API for database operations)
  + UI Pages for front end
* Excluded Features (for now):
  + Social media integrations
  + Advanced search and filtering features for books
  + REST API Authorisation using token
  + Modularity of code directory structure
  + Messaging and transaction management

## 1.3 Target Audience

* General users looking to exchange or purchase books from nearest locations

# Functional **Requirements**

## 2.1 User Stories

### 2.1.1 User Story 1: User Authentication

As a user, I want to securely register, log in, and manage my account, So that I can access and use the book exchange platform.

**Acceptance Criteria:**

* The platform must allow users to register with a valid email and password.
* Passwords must be stored securely using encryption.
* Users should be able to reset their password via a password recovery system.
* Users should be able to log out from their account.

### 2.1.2 User Story 2: Book Listing

As a user, I want to list books that I want to exchange or lend, So that others can browse and request the books I offer.

**Acceptance Criteria:**

* Users should be able to add a book to their list by providing details such as title, author, genre, condition, and availability status.
* Each book listing must have a unique ID associated with a user’s profile.
* Users should be able to edit or delete book listings at any time.
* The book listing must be displayed in the user's profile and searchable by others.

### 2.1.3 User Story 3: Book Search

As a user, I want to search for books based on criteria such as title, author, genre, and location,

So that I can easily find books that interest me.

**Acceptance Criteria:**

* The platform must provide a search bar where users can enter keywords like title, author, or genre.
* The platform should allow users to filter search results by availability status, genre, and location.
* Users must be able to view detailed information about a book (title, author, condition, etc.) when clicking on a search result.
* The search results should be paginated or load incrementally to handle large datasets.

### 2.1.4 User Story 4: Exchange Requests

As a user, I want to send and receive book exchange requests, So that I can initiate a transaction to exchange books with others.

**Acceptance Criteria:**

* Users must be able to send an exchange request to another user for a specific book.
* The request must include the option to negotiate terms, such as delivery method and exchange duration.
* The recipient of the request should be able to accept, reject, or modify the request.
* Both parties should receive notifications about the status of the exchange request (pending, accepted, rejected, modified).
* The platform should track ongoing exchanges in the user's transaction history.

### 2.1.5 User Story 5: Transaction Management

As a user, I want to manage my book exchanges, so that I can track the status of all my exchange transactions.

**Acceptance Criteria:**

* Users must be able to view a history of their exchange requests, including pending, accepted, and completed exchanges.
* The transaction management interface should allow users to cancel pending exchanges.
* Users should receive notifications when a transaction status changes (e.g., request accepted, book delivered).
* Transaction history should be available to users on their profile page.

## 2.2 Features & Functionality

* **Authentication:** User sign-up, login, forgot password, token-based authentication (JWT).
* **Book Management:** CRUD operations for book listings & book searching, user profile details, and auth system database.
* **Dashboard:** User-specific dashboard showing their books and exchange status.
* **API Integration:** Flask API endpoints to perform database operations for books and users.

## 2.3 User Roles & Permissions

* **Admin:** Full access to manage users and books. (Not implemented)
* **Registered User:** Access to exchange books, view books.

# Non-Functional Requirements

## 3.1 Performance

* The application should support up to 500 concurrent users with minimal load times.

Note: It is kept at 500 for concurrent users as it’s a new platform.

## 3.2 Security

* JWT for secure authentication.
* Password hashing.
* Two-factor authentication for password reset.

## 3.3 Scalability

* Horizontal scaling for increased user load, with database read replicas for efficient data access.

## 3.4 Usability

* Simple, intuitive UI designed for ease of use across devices.

## 3.5 Reliability & Availability

* 99.9% uptime expected, with automatic failover for server downtime.

## 3.6 Maintainability

* Well-documented code following Python and React best practices.

## 3.7 Compliance

* GDPR (General Data Protection Regulation ) -compliant data handling, especially in user authentication.

# 4. Architecture Overview

## 4.1 System Architecture

* Microservice architecture with containerized services for frontend (React) and backend (Flask).

A diagram of a software company

Description automatically generated with medium confidence

## 4.2 Frontend Architecture

* **Technology Stack:** React, React Router (for navigation), Axios for client side request to backend.
* **Component Structure:** Reusable components for book listings, user profile, etc.
* **Routing:** React Router for navigation between pages.

A blue and white logo

Description automatically generatedA white number on a blue background

Description automatically generated

## 4.3 Backend Architecture

* **Technology Stack:** Flask API with PostgreSQL for database management.
* **API Structure:** RESTful API exposing endpoints for user authentication and book management.
* **Authentication:** JWT tokens for secure access control.

A black and white image of a moon

Description automatically generatedA blue and yellow snake logo

Description automatically generatedA white text on a black background

Description automatically generated

## 4.4 Database Design

* **Schema for each data model:**
  + Users table: (id, name, email, password\_hash).
  + Token table: (id, user\_id, token, refresh\_token, otp, otp\_expires, created\_by, expires\_at).
  + Books table: (id, title, author, genre, location, listed\_by, borrow\_period, request\_status, request\_by).
  + User Profile table: (id, username, email, age, location, profilepic)
* **Relationships:**
  + Users has a one-to-many relationship with Books.
  + Books has a one-to-one relationship with Users.
  + Token has one-to-many relationship with Users.

## 4.5 Data Flow

* User actions (e.g., login, book exchange requests) trigger Flask API calls, which interact with the Flask SQL database.

# UI/UX Design

## Wireframes/Mockups

Refer slide deck for mock up.

## 5.2 Navigation Flow

* Home → Login/Sign-up → Dashboard → Browse Books → Book Exchange

## 5.3 Design Guidelines

* Colour Scheme: Soft and dark blue and white tones for a clean interface.
* Typography: Sans-serif fonts for readability.

## 5.4 Responsive Design

* Design optimized for desktop, tablet, and mobile.

# API Design

## 6.1 API Endpoints

Flask API is used for backend. Following are the details of APIs.

User Profile Data Model & Microservice:

| Route | Method | Description | Parameters | Request Body | Response Codes | Response Example |
| --- | --- | --- | --- | --- | --- | --- |
| /user-profile | GET | Get user profile by email | Query Parameter: email (string, required) | N/A | 200: Success, 400: Missing email, 404: User not found | { "username": "JohnDoe",  "email": "john.doe@example.com",  "location": "New York" } |
| /user-profile | POST | Create or update user profile | Request Body: username (string), email (string), age (integer), location (string), profilePic (string, optional) | { "username":  "JaneDoe", "email":  "jane.doe@example.com", "age": 28,  "location": "Los Angeles",  "profilePic":  "https://example.com/profile-pic.jpg" } | 201: Created/Updated, 400: Invalid input, 500: Server error | {  "message": "User profile created/updated successfully",  "user": { "username": "JaneDoe",  "email": "jane.doe@example.com",  "location": "Los Angeles",  "profilePic": "https://example.com/profile-pic.jpg" } } |
| /user-profile | DELETE | Delete user profile by email | Query Parameter: email (string, required) | N/A | 200: Success, 400: Missing email, 404: User not found | {  "message": "User with email john.doe@example.com deleted successfully" } |
| /user-profiles | GET | Get all user profiles | N/A | N/A | 200: Success, 404: No profiles, 500: Server error | [ { "email": "john.doe@example.com",  "location": "New York", "profilePic":  "https://via.placeholder.com/50" }, {  "email": "jane.doe@example.com",  "location": "Los Angeles", "profilePic":  "https://via.placeholder.com/50" } ] |
| /user-profiles | DELETE | Delete all user profiles | N/A | N/A | 200: Success, 500: Server error | { "message": "Deleted 3 users from the database" } |

Users & Token Data Model & Microservice:

| Route | Method | Description | Parameters | Request Body | Response Codes | Response Example |
| --- | --- | --- | --- | --- | --- | --- |
| /signup | POST | User signup | None | {"email": "string",  "password":  "string",  "name": "string"} | 201: User created 400: Missing fields 409: User already exists | {"message": "User created successfully"} |
| /login | POST | User login | None | {"email": "string",  "password":  "string"} | 200: Login successful 401: Invalid credentials | {"token": "jwt\_token", "refresh\_token": "jwt\_token"} |
| /send-otp | POST | Send OTP for password reset | None | {"email": "string"} | 200: OTP sent 404: User not found 500: Failed to send OTP | {"message": "OTP sent successfully", "otp": "123456"} |
| /reset-password | POST | Reset user password | None | {"email": "string",  "otp": "string",  "password":  "string"} | 200: Password reset successful 400: Invalid OTP 404: User not found | {"message": "Password has been reset successfully"} |
| /user/<int:user\_id> | DELETE | Delete user by ID | user\_id: int (in path) | None | 200: User deleted 404: User not found | {"message": "User and associated tokens deleted successfully"} |
| /logout | POST | User logout | None | None | 200: Logout successful 400: Token not found 500: Server error | {"message": "Logout successful"} |

Book Listing Data Model & Microservice:

| Route | Method | Description | Parameters | Request Body | Response Codes | Response Example |
| --- | --- | --- | --- | --- | --- | --- |
| /books | POST | Add a new book | None | JSON: { "title": "book1", "author": "author1", ... } | 201 Created, 400 Bad Req. | {"message": "Book added successfully"} or {"message": "Missing fields"} |
| /books | GET | Get all books | None | None | 200 OK | [{"title": "book1", "author": "author1", ...}] |
| /books/search | GET | Get books by listed\_by | listed\_by as query param | None | 200 OK, 404 Not Found | [{"title": "book1", "author": "author1", ...}] or {"message": "No books found matching the provided criteria"} |
| /raise-request | POST | Send email for raising request | None | JSON: { "to": "recipient@example.com", "subject": "subject", "message": "content" } | 200 OK, 400 Bad Req. | {"success": True, "message": "Request email sent successfully!"} or {"error": "Invalid data received"} |
| /add-book | POST | Add a book and return its details | None | JSON: { "title": "book1", "author": "author1", ... } | 201 Created, 400 Bad Req. | {"message": "Success", "id": 1, "title": "book1", ...} |
| /update-book | POST | Update the request\_by field of a book | None | JSON: { "listed\_by": "email@example.com", "title": "book1", "request\_by": "requester1" } | 200 OK, 404 Not Found | {"message": "Book request has been updated successfully!"} or {"message": "Book not found!"} |
| /books/request-update | PATCH | Update the request\_status field of a book | None | JSON: { "listed\_by": "email@example.com", "title": "book1", "request\_status": "requested" } | 200 OK, 400 Bad Req., 404 Not Found | {"message": "Updated status to requested"} or {"message": "Book not found"} |
| /delete-book | DELETE | Delete a book by listed\_by and title | listed\_by, title as query params | None | 200 OK, 400 Bad Req., 404 Not Found | {"message": "Deleted successfully"} or {"message": "Book not found"} |

## 6.2 Authentication & Authorization

* JWT tokens are used to authenticate users across login endpoints.

Note: Due to time constraints, its not implemented on all endpoints.

## 6.3 Rate Limiting & Throttling

* Rate limiting set to 100 requests per minute per IP. It should be implemented on server side.

Note: This is not yet implemented.

## 6.4 Error Handling

* Standard 400, 401, 403, and 500 error codes with detailed error messages.

## 6.5 Versioning

* API versioning using /v1/ in URLs.

# Implementation Demo Deck



# Deployment & Hosting

## 7.1 Environment Setup

* Dev, Staging, and Production environments configured using Docker.

## 7.2 Hosting Infrastructure

* AWS EC2 instances running Docker containers.

## 7.3 Containerization & Orchestration

* Docker containers for both the React frontend and Flask backend micorservices.

## 7.4 CI/CD Pipeline

* GitHub Actions for continuous integration and deployment.

## 7.5 Load Balancing & Caching

* AWS Elastic Load Balancer (ELB) for distributing traffic.
* Redis for caching frequently accessed data.

## 7.6 Monitoring & Logging

* AWS CloudWatch for logging and monitoring performance.

## 7.7 Backup & Recovery

* Daily backups of the flask SQL database to AWS S3.

# Testing Plan

## 8.1 Unit Testing

* PyTest for testing Flask API endpoints.

## 8.2 Integration Testing

* Postman integration testing for API endpoints.

## 8.3 End-to-End Testing

* Selenium tests for full user journeys (from sign-up to book exchange).

## 8.4 Performance Testing

* Apache JMeter for load testing under different user loads.

## 8.5 Security Testing

* Regular penetration testing and vulnerability scans using OWASP ZAP.

# Security Considerations

## 9.1 Authentication & Authorization

* JWT-based authentication with token expiration and refresh strategies.

## 9.2 Data Protection

* SSL/TLS for all communications.
* AES-256 encryption for sensitive user data stored in the database.

## 9.3 Secure Data Storage

* Secure cookies for session management.

## 9.4 Common Vulnerabilities

* Protection against SQL injection, CSRF, and XSS vulnerabilities using Flask extensions and best practices.

## 9.5 Logging & Monitoring

* Login attempts and error logs are securely stored and monitored via CloudWatch.

# Project Timeline & Milestones

## 10.1 Development Timeline

* Phase 1: MVP Development (2 months)
  + 3 User stories implementation: Done with basic implementation. It needs further enhancement tasks.
* Phase 2: Beta Testing (1 month)
* Phase 3: Full Launch (1 month)

## 10.2 Key Milestones

* MVP Completion: End of Month 2
* Beta Testing: End of Month 3
* Full Launch: End of Month 4

## 10.3 Resource Allocation

* Developer Team: 2 Full Stack Developers, 1 UI/UX Designer, 1 DevOps Engineer.

# Risk Management

## 11.1 Risk Identification

* Scaling challenges under high user load.

## 11.2 Mitigation Strategies

* Test scalability with load testing.

## 11.3 Contingency Plans

* Implement horizontal scaling using AWS auto-scaling groups.

# 12. Future Enhancements

## 12.1 Roadmap for New Features

* Social media integration for sharing book lists.
* Advanced filtering options for book search.

## 12.2 Scalability Considerations

* Plan for implementing global CDN and multi-region hosting to support international users.