Book Bazaar Documentation

Version 1.0

1. Project Overview

1.1 Introduction

Book Bazaar is a comprehensive library management system that integrates both SQL and NoSQL databases. The system provides RESTful APIs for managing books, authors, sales, and reviews. This project demonstrates the implementation of a full-stack application using modern web technologies and database systems.

1.2 Features

- **Complete Book Management System**
  - Create, read, update, and delete books
  - Track book inventory
  - Manage book sales
  - Handle book reviews
- **Author Management**
  - Author profiles
  - Author-book relationships
  - Author search functionality
- **Sales and Inventory**
  - Real-time stock tracking
  - Sales recording
  - Sales history
  - Inventory management

- **Review System**
    - User reviews
    - Rating system
    - Review moderation

- **API Integration**
    - RESTful API endpoints
    - JSON response format
    - Error handling
    - Status codes

1.3 Technologies Used

Backend Framework:

- Python 3.9+
- Flask Web Framework

Databases:

- SQLite for structured data
    - Books
    - Authors
    - Sales
    - Stock

- MongoDB for unstructured data
    - Reviews
    - User comments
    - Ratings

Web Server:

- Apache 2.4
- mod_wsgi for Python-Apache integration

Development Tools:

- Postman for API testing
- SQLite Browser for database management
- MongoDB Compass for NoSQL database management

Version Control:

- Git for source code management

2. System Architecture

2.1 Component Overview

The system follows a modular architecture with the following components:

1. Database Layer

- SQLite Database
  - Primary data storage
  - Structured data management
  - ACID compliance

- MongoDB Database
  - Review storage
  - Flexible schema
  - Scalable document storage

2. Application Layer

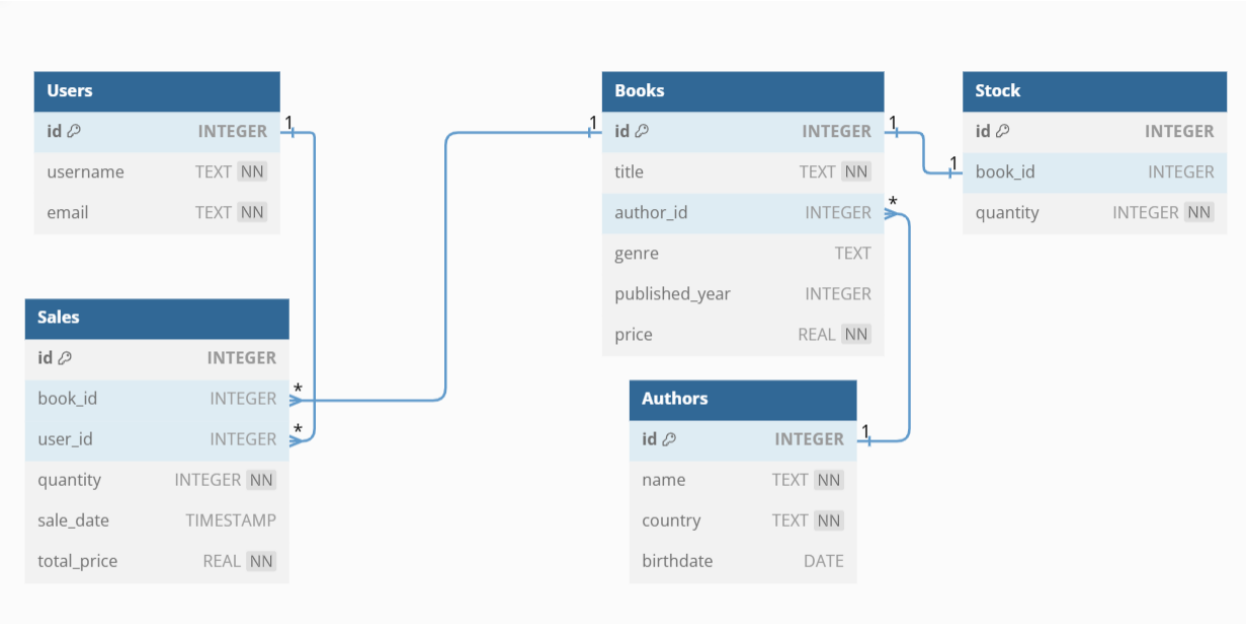
- Core Services
  - Database connections
  - Business logic
  - Data validation

- API Services
  - REST endpoints
  - Request handling
  - Response formatting

3. Web Server Layer

- Apache configuration
- WSGI integration
- Static file serving

2.2 Database Schema



(Figure 2.1: BookBazaar Database Schema)

SQLite Tables:

1. Users

- id (INTEGER PRIMARY KEY)
- username (TEXT NOT NULL)
- email (TEXT NOT NULL UNIQUE)

2. Authors

- id (INTEGER PRIMARY KEY)
- name (TEXT NOT NULL)
- country (TEXT NOT NULL)
- birthdate (DATE)

3. Books

- id (INTEGER PRIMARY KEY)
- title (TEXT NOT NULL)
- author_id (INTEGER FOREIGN KEY)
- genre (TEXT)
- published_year (INTEGER)
- price (REAL NOT NULL)

4. Stock

- id (INTEGER PRIMARY KEY)
- book_id (INTEGER FOREIGN KEY)
- quantity (INTEGER NOT NULL)

5. Sales

- id (INTEGER PRIMARY KEY)
- book_id (INTEGER FOREIGN KEY)
- user_id (INTEGER FOREIGN KEY)
- quantity (INTEGER NOT NULL)
- sale_date (TIMESTAMP)
- total_price (REAL NOT NULL)

MongoDB Collections:

1. Reviews

```
{
  _id: ObjectId,
  book_id: Integer,
  user_id: Integer,
  rating: Integer,
  comment: String,
  date: DateTime
}
```

2.3 API Structure

```
/api
 /books
  GET    /       - List all books
  POST   /       - Create new book
  GET    /<id>    - Get book details
  PUT    /<id>    - Update book
  DELETE /<id>    - Delete book

 /authors
  GET    /       - List all authors
  POST   /       - Create new author
```

GET /<id> - Get author details

PUT /<id> - Update author

DELETE /<id> - Delete author

/stock

GET /<book_id> - Get stock level

PUT /<book_id> - Update stock

/sales

POST / - Create new sale

GET /user/<id> - Get user sales

/reviews

GET /book/<id> - Get book reviews

POST /book/<id> - Add review

PUT /<id> - Update review

DELETE /<id> - Delete review

Hint: you can see all in postman we can add swagger but it isn't required

3. Installation Guide

3.1 Prerequisites

Required Software:

- Python 3.9 or higher
- SQLite3
- MongoDB
- Apache Web Server
- Git (for version control)

Python Packages:

- Flask
- Flask-RESTx
- PyMongo
- SQLite3
- mod_wsgi

Development Tools:

- Postman
- Visual Studio Code or PyCharm
- DB Browser for SQLite
- MongoDB Compass

3.2 SQLite Setup

1. Install SQLite:

Windows:

- Download SQLite from https://sqlite.org/download.html
- Add to system PATH

Linux:

- sudo apt-get update
- sudo apt-get install sqlite3

2.Create Database:

- cd /path/to/project
- sqlite3 database/bookbazaar.db

3.Initialize Schema:

- sqlite3 database/bookbazaar.db < database/schema.sql

3.3 MongoDB Setup

1. Install MongoDB:

Windows:

- Download MongoDB Community Server
- Run installer
- Add to system PATH

Linux:

- sudo apt-get install mongodb
- sudo systemctl start mongodb

2. Create Database:

- mongosh
- use bookbazaar_reviews

3.4 Python Environment Setup

1. Create Virtual Environment:

- python -m venv venv
- venv\Scripts\activate # Windows
- source venv/bin/activate # Linux/MacOS

2. Install Dependencies:

- pip install -r requirements.txt

3.5 Apache Configuration

1. Install Apache:
   Windows:
   - Download Apache from Apache Lounge
   - Extract to C:\Apache24

2. Install mod_wsgi:
    - pip install mod_wsgi (required c++ lib make sure you installed it)
3. Configure Virtual Host:

```
<VirtualHost *:80>
    # Admin email and domain name for the Flask app
    ServerAdmin www.bookbazaar.test

    # The ServerName directive specifies the domain name of your Flask
application
    # This is the URL that users will use to access the Flask app
    ServerName bookbazaar.test

    # Document root is the directory containing your Flask app
    DocumentRoot
"D:/ACC_Sprints_AI_ML_BootCamp/Capstone_projects/Cap2_BookBazaar"

    # Default log file locations (for error and access logs)
    ErrorLog "C:/Apache24/logs/error.log"
    CustomLog "C:/Apache24/logs/access.log" combined


    #WSGIScriptAlias is used to define the location of the WSGI application
    WSGIScriptAlias /
"D:/ACC_Sprints_AI_ML_BootCamp/Capstone_projects/Cap2_BookBazaar/wsgi.py"


    #  allowing access to the directory where the Flask app resides
    # 'Require all granted' means that all requests are allowed to access this
directory
    <Directory "D:/ACC_Sprints_AI_ML_BootCamp/Capstone_projects/Cap2_BookBazaar">

        Require all granted
    </Directory>
</VirtualHost>
```

4.API Documentation

Base URL: http://bookbazaar.test/api (we configure it with apachi)

4.1 Books API

1. Get All Books

GET /books

Description: Retrieve all books

Response 200:


```
[
  {
    "id": 1,
    "title": "1984",
    "author_id": 1,
    "genre": "Dystopian",
    "published_year": 1949,
    "price": 9.99
  }
]
```


2. **Get Single Book**

   GET /books/

   Description: Retrieve a specific book

   Response 200:


```
{
  "id": 1,
  "title": "1984",
  "author_id": 1,
  "genre": "Dystopian",
  "published_year": 1949,
  "price": 9.99
}
```


3. **Create Book**

   POST /books

Description: Create a new book
Request:

```
{
  "title": "New Book",
  "author_id": 1,
  "genre": "Fiction",
  "published_year": 2023,
  "price": 19.99
   }
```

Response 201:

```
{
   "id": 2,
   "message": "Book created successfully"
}
```

4. **Update Book**
   PUT /books/
   Description: Update an existing book
   Request:

```
{
  "title": "Updated Book",
  "genre": "Non-Fiction",
  "price": 24.99
}
```

Response 200:

```
{
  "message": "Book updated successfully"
}
```

5. Delete Book
   DELETE /books/<id>
   Description: Delete a book
   Response 200:

```
{
  "message": "Book deleted successfully"
```

}

4.2 Authors API

1. Get All Authors
   GET /authors
   Response 200:


```
[
  {
    "id": 1,
    "name": "George Orwell",
    "country": "United Kingdom",
    "birthdate": "1903-06-25"
  }
]
```


2. Get Single Author
   GET /authors/
   Response 200:

   JSON


```
{
  "id": 1,
  "name": "George Orwell",
  "country": "United Kingdom",
  "birthdate": "1903-06-25"
}
```


3. Create Author
   POST /authors
   Request:


```
{
  "name": "New Author",
  "country": "USA",
  "birthdate": "1990-01-01"
}
```


   Response 201:


```
{
```

"id": 2,

"message": "Author created successfully"

}

4. Update Author
   PUT /authors/<id>
   Request:

   {

   "country": "Canada",

   "birthdate": "1990-01-01"

   }

   Response 200:

   {

   "message": "Author updated successfully"

   }

5. Delete Author
   DELETE /authors/<id>
   Response 200:

   {

   "message": "Author deleted successfully"

   }

4.3 Stock API

1. Get Stock Level
   GET /stock/<book_id>
   Response 200:

   {

   "id": 1,

   "book_id": 1,

   "quantity": 10

   }

2. Update Stock
   PUT /stock/<book_id>
   Request:

   {

"quantity": 15

}

Response 200:

{

"message": "Stock updated successfully"

}

4.4 Sales API

1. Create Sale
   POST /sales
   Request:

   {

   "book_id": 1,

   "user_id": 1,

   "quantity": 2

   }

   Response 201:

   {

   "id": 1,

   "message": "Sale created successfully",

   "total_price": 19.98

   }

2. Get User Sales
   GET /sales/user/<user_id>
   Response 200:

   [

   {

   "id": 1,

   "book_id": 1,

   "user_id": 1,

   "quantity": 2,

   "sale_date": "2023-11-15T10:30:00Z",

```
    "total_price": 19.98
  }
]
```


3. Get Sale Details
   GET /sales/
   Response 200:


```
{
  "id": 1,
  "book_id": 1,
  "user_id": 1,
  "quantity": 2,
  "sale_date": "2023-11-15T10:30:00Z",
  "total_price": 19.98
}
```


4.5 Reviews API (MongoDB)

1. Get Book Reviews
   GET /books//reviews
   Response 200:


```
[
  {
    "id": "507f1f77bcf86cd799439011",
    "book_id": 1,
    "user_id": 1,
    "rating": 5,
    "comment": "Great book!",
    "date": "2023-11-15T10:30:00Z"
  }
]
```

2. Add Review
   POST /books//reviews
   Request:

   {

   "user_id": 1,

   "rating": 5,

   "comment": "Excellent read!"

   }

   Response 201:

   {

   "id": "507f1f77bcf86cd799439011",

   "message": "Review added successfully"

   }

3. Update Review
   PUT /reviews/<review_id>
   Request:

   {

   "rating": 4,

   "comment": "Updated review comment"

   }

   Response 200:

   {

   "message": "Review updated successfully"

   }

4. Delete Review
   DELETE /reviews/<review_id>
   Response 200:

   {

   "message": "Review deleted successfully"

   }

4.6 Users API

1. Get All Users
   GET /users
   Response 200:

```
[
  {
    "id": 1,
    "username": "john_doe",
    "email": "john@example.com"
  }
]
```

2. Get Single User
   GET /users/<id>
   Response 200:

```
{
  "id": 1,
  "username": "john_doe",
  "email": "john@example.com"
}
```

3. Create User
   POST /users
   Request:

```
{
  "username": "new_user",
  "email": "new@example.com"
}
```

   Response 201:

```
{
  "id": 2,
  "message": "User created successfully"
}
```

4. Update User
   PUT /users/
   Request:

{

"email": "updated@example.com"

}

Response 200:

{

"message": "User updated successfully"

}

5. Delete User
   DELETE /users/
   Response 200:

{

"message": "User deleted successfully"

}

Error Responses (global)

400 Bad Request:

{

"error": "Missing required fields"

}

404 Not Found:

{

"error": "Resource not found"

}

500 Server Error:

{

"error": "Internal server error"

}

Status Codes

- 200: Success
- 201: Created
- 400: Bad Request

• 401: Unauthorized (we didn't add middle ware as it was out of scope)
• 403: Forbidden
• 404: Not Found
• 500: Server Error

Request Headers

Content-Type: application/json

Response Headers

Content-Type: application/json

5. Database Operations

5.1 SQLite Operations

1. Database Connection

```python
import sqlite3
from contextlib import contextmanager
from config.config import Config

class SQLiteService:
    @staticmethod
    @contextmanager
    def get_connection():
        conn = sqlite3.connect(Config.SQLITE_DB_PATH)
        conn.row_factory = sqlite3.Row
        try:
            yield conn
        finally:
            conn.close()

    @staticmethod
```

```
    def init_db():
        with open('database/schema.sql', 'r') as f:
            schema = f.read()
        with SQLiteService.get_connection() as conn:
            conn.executescript(schema)
            conn.commit()
```

5.2 MongoDB Operations

1. Database Connection:

```
from pymongo import MongoClient

def get_mongo_connection():
    try:
        client = MongoClient('mongodb://localhost:27017/')
        db = client.bookbazaar_reviews
        return db
    except Exception as e:
        print(f"Error connecting to MongoDB: {e}")
        return None
```

6. Testing Guide

6.1 Postman Setup

1. Install Postman:
    - Download from: https://www.postman.com/downloads/
    - Install and create an account
2. Import Collection:
    - Open Postman
    - Click "Import"
    - Select BookBazaar.postman_collection.json
3. Set Environment:
    - Create new environment
    - Set variable 'baseUrl' to 'http://bookbazaar.test'

6.2 API Testing Procedures

1. Basic Test Flow:
    1. Create Author
    2. Create Book
    3. Update Stock
    4. Create Sale
    5. Add Review

7. Deployment Guide

7.1 Apache Setup

1-Install Apache

2-Enable Required Modules

7.2 WSGI Configuration

1. **Create wsgi.py:**

```
import sys
import os

# Add the path to your Flask app
sys.path.insert(0,
'D:/ACC_Sprints_AI_ML_BootCamp/Capstone_projects/Cap2_BookBazaar')


# Import the Flask app from cap_flask_api.py
from run import create_app

application = create_app()
```

2. **Apache Virtual Host Configuration:**

```
<VirtualHost *:80>
    # Admin email and domain name for the Flask app
    ServerAdmin www.bookbazaar.test

    # The ServerName directive specifies the domain name of your Flask
application
    # This is the URL that users will use to access the Flask app
    ServerName bookbazaar.test

    # Document root is the directory containing your Flask app
    DocumentRoot
"D:/ACC_Sprints_AI_ML_BootCamp/Capstone_projects/Cap2_BookBazaar"

    # Default log file locations (for error and access logs)
    ErrorLog "C:/Apache24/logs/error.log"
    CustomLog "C:/Apache24/logs/access.log" combined


    #WSGIScriptAlias is used to define the location of the WSGI application
    WSGIScriptAlias /
"D:/ACC_Sprints_AI_ML_BootCamp/Capstone_projects/Cap2_BookBazaar/wsgi.py"


    #  allowing access to the directory where the Flask app resides
    # 'Require all granted' means that all requests are allowed to access this
directory
    <Directory "D:/ACC_Sprints_AI_ML_BootCamp/Capstone_projects/Cap2_BookBazaar">

        Require all granted
    </Directory>
</VirtualHost>
```

7.3 Production Deployment

1. Security Considerations:
    - Disable debug mode
    - Set secure permissions

- Use HTTPS
- Implement rate limiting

2. Environment Variables:
    - export FLASK_ENV=production
    - export FLASK_DEBUG=0

8. Troubleshooting

8.1 Common Issues

1. Database Connection Issues:
    - Check database file permissions
    - Verify connection strings
    - Ensure proper user rights
2. API Errors:
    - Check request format
    - Verify endpoint URLs
    - Validate input data

8.2 Error Codes

1. HTTP Status Codes:
    - 400: Bad Request
    - 401: Unauthorized
    - 403: Forbidden
    - 404: Not Found
    - 500: Server Error
2. Custom Error Codes:
    - DB001: Database Connection Error
    - DB002: Query Execution Error
    - API001: Invalid Input
    - API002: Resource Not Found

8.3 Solutions

1. Database Issues:
   # Check SQLite database
   sqlite3 database/bookbazaar.db ".tables"

   # Check MongoDB
   mongo bookbazaar_reviews --eval "db.stats()"
2. Permission Issues:
   # Fix file permissions
   chmod 644 database/bookbazaar.db
   chmod 755 /path/to/BookBazaar

9.Appendices

9.1 Sample Data

1. Sample Books:
   INSERT INTO Books (title, author_id, genre, published_year, price)
   VALUES
   ('1984', 1, 'Dystopian', 1949, 9.99),
   ('Harry Potter', 2, 'Fantasy', 1997, 14.99);

2. Sample Authors:
   INSERT INTO Authors (name, country, birthdate)
   VALUES
   ('George Orwell', 'United Kingdom', '1903-06-25'),
   ('J.K. Rowling', 'United Kingdom', '1965-07-31');

9.2 Configuration Files

1-Flask Configuration:

```
class Config:
  SQLITE_DATABASE_URI = 'database/bookbazaar.db'
  MONGO_URI = 'mongodb://localhost:27017/'
```

MONGO_DBNAME = 'bookbazaar_reviews'

2. Apache Configuration:
   # Include in httpd.conf
   Include conf/extra/bookbazaar.conf