**Project General Description:**

BookBazaar is a comprehensive library management system that integrates both a relational database (SQLite) and a non-relational database (MongoDB). The project involves designing databases, connecting them to a Python application, developing RESTful APIs with Flask, and deploying these APIs using an Apache web server. Through these tasks, learners will gain hands-on experience in full-stack development, including database design, Python integration, API creation, testing with Postman, and server deployment.

**Task 1:**

Set Up the Relational Database with SQLite  
Description: You will create and set up a SQLite database file to store structured data for books, authors, and user information. SQLite is a file-based database that does not require a separate server  
Requirements:  
1.    Install SQLite and ensure it is accessible from your command line or terminal.  
2.    Create a new SQLite database file (e.g., bookbazaar.db).  
3.    Ensure that your application or Python scripts will have permission to read and write to this database file

**Task 2:**

Design the Relational Database Schema  
Description:  
You will design the database schema for your relational database. This includes deciding on table structures and relationships."  
"Requirements:  
1.    Identify the entities you need (e.g., Users, Authors, Books).  
2.    Determine the fields for each table (e.g., username, email for Users; name, country for Authors; title, author\_id, genre, published\_year for Books).  
3.    Define primary keys and any foreign keys (e.g., Books.author\_id references Authors.id).  
4.    Consider constraints such as NOT NULL and UNIQUE where appropriate.  
5.    Create an ER diagram to visualize your schema.  
6.    Prepare SQL commands to create the tables and insert some sample data.

**Task 3:**

Connect Python to SQLite  
Description:  
You will establish a connection from Python to your SQLite database. This connection will be used in subsequent tasks to run queries, insert data, and retrieve results  
Requirements:  
1.    Use Python’s built-in sqlite3 module to connect to the bookbazaar.db file.  
2.    Implement proper error handling in case the connection fails.  
3.    Confirm that you can successfully connect and close the connection.  
**Task 4:**

Title: Implement CRUD Operations on SQLite via Python  
Description:  
You will create functions in Python to perform Create, Read, Update, and Delete (CRUD) operations on your SQLite database.  
Requirements:  
1.    Write functions to insert a new book into the Books table.  
2.    Write functions to retrieve book details by ID or to list all books.  
3.    Write functions to update a book’s information (e.g., genre, title).  
4.    Write functions to delete a book by its ID.  
5.    Ensure your functions handle errors gracefully (e.g., if a book ID does not exist).

**Task 5:**

 Develop RESTful APIs with Python  
Description:  
You will build RESTful API endpoints so external clients can interact with your Books data stored in SQLite.  
Requirements:  
1.    Set up a basic Flask application.  
2.    Create routes for:   
o    GET /books to retrieve all books.  
o    POST /books to add a new book.  
o    PUT /books/<id> to update a book.  
o    DELETE /books/<id> to delete a book.  
3.    Return JSON responses and use proper HTTP status codes.  
4.    Handle invalid input and database errors with appropriate error responses.

**Task 6:**

Test APIs Using Postman  
Description:  
You will use Postman to test your API endpoints. Postman is a tool that allows you to send HTTP requests and inspect responses easily.  
"Requirements:  
1.    Install Postman.  
2.    Create a Postman collection for your APIs.  
3.    Test each endpoint:   
o    Add a new book.  
o    Retrieve the list of books.  
o    Update a book’s details.  
o    Delete a book.  
4.    Verify that the responses are as expected and handle errors correctly.

**Task 7:**

Host APIs on Apache Web Server  
Description:  
You will deploy your Flask application behind an Apache web server using mod\_wsgi. This simulates a more production-like environment.  
Requirements:  
1.    Install Apache and mod\_wsgi.  
2.    Configure Apache to serve your Flask application.  
3.    Ensure that when you access your server’s URL in a browser or via Postman, you can reach your API endpoints.  
4.    Troubleshoot any server configuration issues (e.g., permission issues, missing modules).

**Task 8:**

Title: Set Up the Non-Relational Database with MongoDB  
Description:  
You will install and initialize a MongoDB database to store reviews. MongoDB is a NoSQL database that stores data in JSON-like documents.  
Requirements:  
1.    Install MongoDB and ensure it is running.  
2.    Create a new database (e.g., bookbazaar\_reviews).  
3.    Set up user authentication for MongoDB if desired.  
4.    Confirm that you can connect to and interact with the database using the Mongo shell or a MongoDB client.

**Task 9:**

Connect Python to MongoDB Using PyMongo  
Description:  
You will use PyMongo, a Python driver for MongoDB, to connect your Python application to the MongoDB database. "  
Requirements:  
1.    Install pymongo.  
2.    Write a Python script or function to establish a connection to bookbazaar\_reviews.  
3.    Handle connection errors gracefully.  
4.    Confirm that you can perform simple operations (like db.command('ping')) to validate the connection.

**Task 10:**

Implement CRUD Operations on MongoDB via Python  
Description:  
You will create functions in Python to perform CRUD operations on the Reviews collection stored in MongoDB.  
Requirements:  
1.    Implement a function to add a new review.  
2.    Implement a function to retrieve reviews for a specific book.  
3.    Implement functions to update and delete reviews by their unique identifiers (e.g., review\_id).  
4.    Ensure proper error handling (e.g., handle cases where a review does not exist).

**Task 11:**

Integrate MongoDB Operations into the APIs  
Description:  
You will extend the Flask APIs to include endpoints that interact with the MongoDB database, allowing clients to manage reviews. "  
Requirements:  
1.    Create endpoints for:   
o    GET /books/<id>/reviews to list all reviews for a given book.  
o    POST /books/<id>/reviews to add a new review to a book.  
o    PUT /reviews/<review\_id> to update a review.  
o    DELETE /reviews/<review\_id> to delete a review.  
2.    Ensure that when adding a review, you verify the book’s existence in the SQLite database.  
3.    Return JSON responses and appropriate HTTP status codes.  
4.    Handle errors, such as invalid book IDs or review IDs that do not exist

**Task 12:**

Document the Entire Project  
Description:  
You will compile comprehensive documentation that covers installation, setup, usage, and API details for the entire project. "  
Requirements:  
1.    Create a word file (documentation file)  
2.    Explain the project goals, technologies used, and how to run the application.  
3.    Document all API endpoints, including request and response formats, and example calls.  
4.    Include setup instructions for SQLite, MongoDB, Python dependencies, Apache configuration, and Postman testing.  
5.    Make the documentation clear enough that a new developer can set up and understand the project without additional guidance.

**Final Notes and Tips for Learners:**

• Work incrementally. Complete one task at a time before moving on.  
• Keep thorough notes on what you do for each task, as these will help you when writing your documentation.  
•Troubleshoot issues by checking logs, reading error messages, and verifying your database connections.  
• Test frequently. Make use of Postman, and consider adding print statements or logging in your code to confirm that each part works as expected.  
• By the end of this project, you should have a working library management system with integrated relational and non-relational databases, hosted APIs, and extensive documentation—equipping you with valuable full-stack development experience.

**Deliverable** :

Database Files:

bookbazaar.db: SQLite database file with tables for books, authors, and users.  
MongoDB database setup with reviews collection (bookbazaar\_reviews).

Python Scripts:

Script to set up SQLite database schema.  
Script for MongoDB connection and operations using PyMongo.  
Functions for CRUD operations on both SQLite and MongoDB.

API Code:

Flask application with endpoints for managing books and reviews.  
Modularized API code integrating SQLite and MongoDB functionality.

Postman Collection:

Postman collection file for testing all API endpoints, including sample requests and responses.

Sample Data:

SQLite data: Books, authors, and user information in .sql or .csv format.  
MongoDB data: Reviews in JSON format.

Deployment Configuration:

Apache configuration files for hosting Flask APIs with mod\_wsgi.  
Deployed and accessible API endpoints tested on an Apache server.

Documentation:

Comprehensive project documentation (BookBazaar\_Documentation.docx or .pdf), including:  
Project overview.  
Setup and installation instructions for SQLite, MongoDB, Flask, and Apache.  
API endpoint details with request/response examples.  
Troubleshooting tips.

Final Integrated System

Fully operational library management system hosted on Apache, with working APIs and integrated SQLite and MongoDB databases.