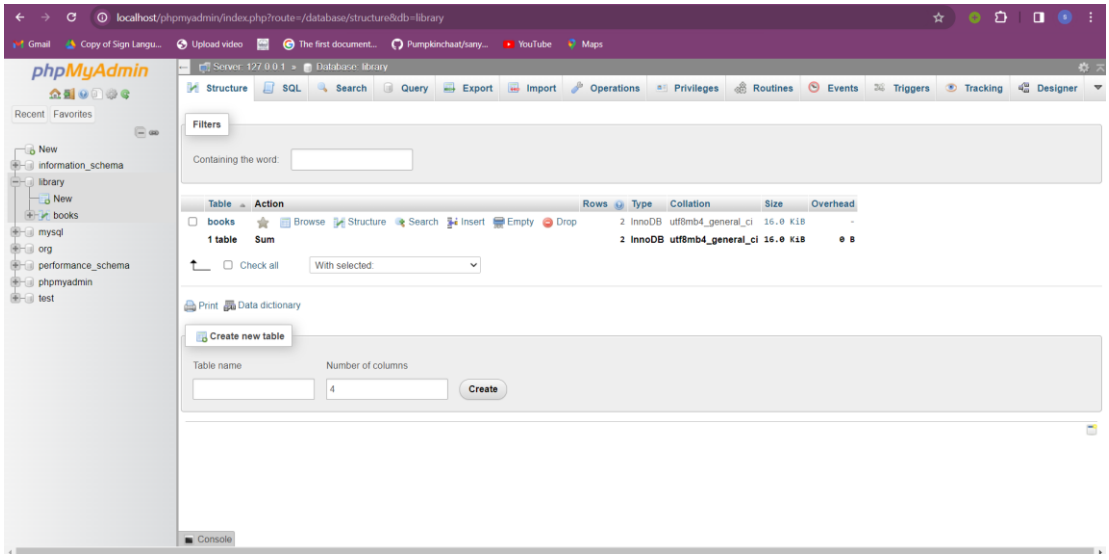
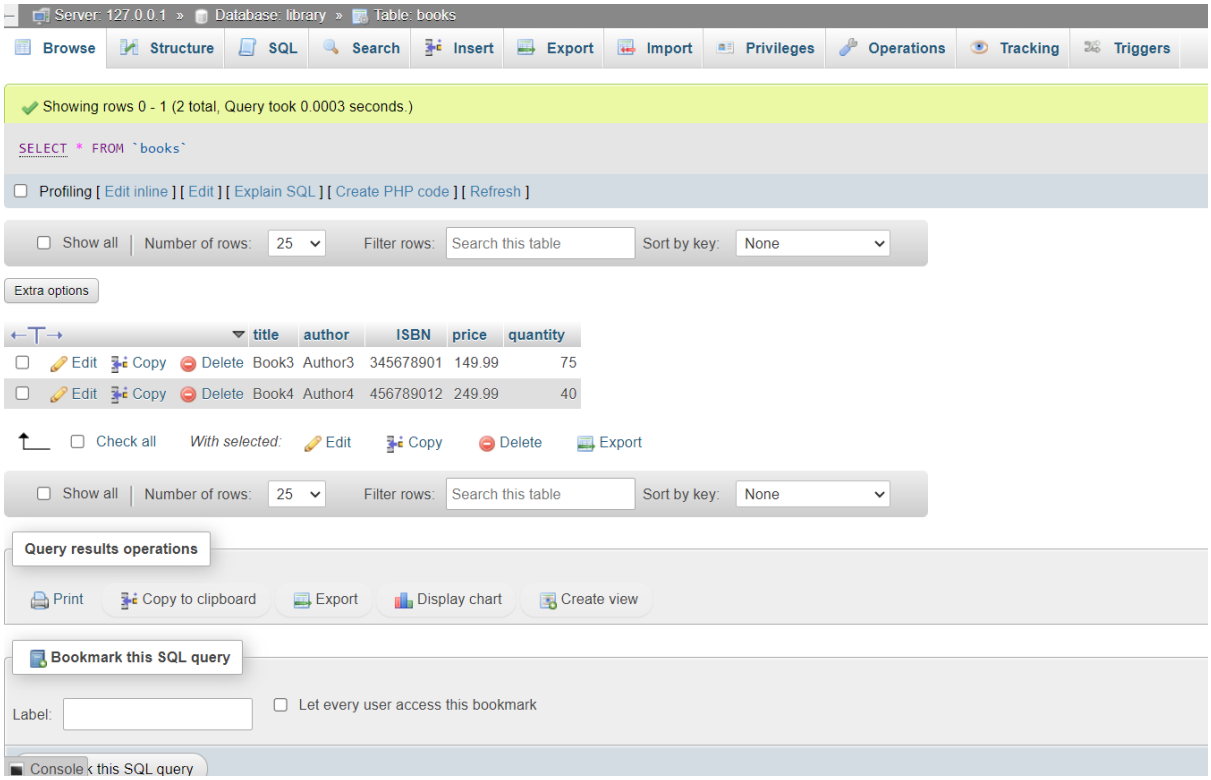
Introduction

This Flask application is a simple RESTful API for managing a library of books. It allows users to add, retrieve, update, and delete books from a MySQL database using HTTP requests. The application uses basic HTTP authentication to restrict access to certain endpoints.

Database Creation and Connection:

1. Here we are using a local server to host our API using XAMPP. XAMPP provides easy and quick MYSQL database setup for development and debugging.





2. For connecting our Flask application to the database we use mysql.connector

```python
import mysql.connector

app = Flask(__name__)
auth = HTTPBasicAuth()

# Connecting Flask App to MYSQL database
db_config = {
    'user': 'root',
    'password': '',
    'host': 'localhost',
    'database': 'library'
}

# Checking the connection
def connect_to_db():
    """Connect to the MySQL database using the provided configuration.
    Returns a connection object if successful, or None if there is an error.
    """
    try:
        conn = mysql.connector.connect(**db_config)
        return conn
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None
```

Retrieve all/specific Book (GET):

We retrieve all the available books in the table by querying out the information form the database. Once the query returns the result it is returned from the app in json format. Necessary error handling is included in case of exception.

```python
@app.route('/books')
def get_books():
    """Get all books from the database.
    Returns a list of books if successful, or a failure message if there is an error.
    """
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM books")
        rows = cursor.fetchall()

        books = []
        for row in rows:
            book = {
                'title': row[0],
                'author': row[1],
                'isbn': row[2],
                'price': row[3],
                'quantity': row[4]
            }
            books.append(book)

        return jsonify(books)
    else:
        return "Failed to connect to the database.", 500
```

To retrieve information of a specific book according to the ISBN we include the corresponding MYSQL query.

```python
@app.route('/books/<string:isbn>')
def get_book_by_isbn(isbn):
    """Get a book by its ISBN from the database.
    Returns the book if found, or an error message if not found or there is an error.
    """
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM books WHERE isbn = %s", (isbn,))
        row = cursor.fetchone()

        if row:
            book = {
                'title': row[0],
                'author': row[1],
                'isbn': row[2],
                'price': row[3],
                'quantity': row[4]
            }
            return jsonify(book)
        else:
            return "Book not found.", 404
    else:
        return "Failed to connect to the database.", 500
```

Update Specific Book (POST):

The data of the book is extracted from the POST requested API and in included in the database using necessary MYSQL query.

```python
@app.route('/books', methods=['POST'])
@auth.login_required
def add_book():
    """Add a new book to the database.
    Returns a success message if successful, or a failure message if there is an error.
    """
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        data = request.get_json()
        title = data.get('title')
        author = data.get('author')
        isbn = data.get('isbn')
        price = data.get('price')
        quantity = data.get('quantity')

        query = "INSERT INTO books (title, author, isbn, price, quantity) VALUES (%s, %s, %s, %s, %s)"
        cursor.execute(query, (title, author, isbn, price, quantity))
        conn.commit()

        return jsonify({'message': 'Book added successfully.'}), 201
    else:
        return "Failed to connect to the database.", 500
```

Add New Book (PUT):

For adding a new book, similar to update method we extract the information of the new Book and include it using MYSQL query.

```python
@app.route('/books/<string:isbn>', methods=['PUT'])
@auth.login_required
def update_book(isbn):
    """Update a book in the database.
    Returns a success message if successful, or a failure message if there is an error.
    """
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        data = request.get_json()
        title = data.get('title')
        author = data.get('author')
        price = data.get('price')
        quantity = data.get('quantity')

        query = "UPDATE books SET title = %s, author = %s, price = %s, quantity = %s WHERE isbn = %s"
        cursor.execute(query, (title, author, price, quantity, isbn))
        conn.commit()

        return jsonify({'message': 'Book updated successfully.'})
    else:
        return "Failed to connect to the database.", 500
```

Delete Specific Book (DELETE):

For adding a new book according to the given ISBN, we extract the ISBN of the Book from the API and include it using MYSQL query.

```python
@app.route('/books/<string:isbn>', methods=['DELETE'])
@auth.login_required
def delete_book(isbn):
    """Delete a book from the database.
    Returns a success message if successful, or a failure message if there is an error.
    """
    conn = connect_to_db()
    if conn:
        cursor = conn.cursor()
        query = "DELETE FROM books WHERE isbn = %s"
        cursor.execute(query, (isbn,))
        conn.commit()

        return jsonify({'message': 'Book deleted successfully.'})
    else:
        return "Failed to connect to the database.", 500


if __name__ == '__main__':
    app.run(debug=True)
```

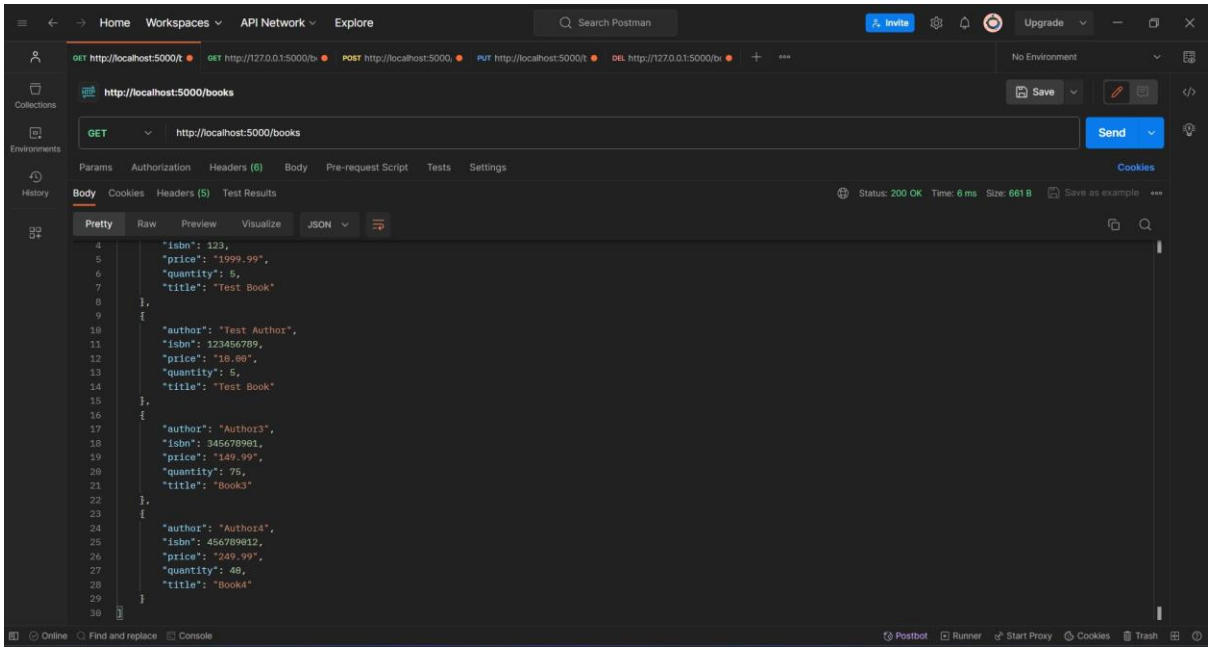
Basic Authorization:

I have used HTTPBasicAuth library which provide functions such as get_password and login_required to provide authorization. Basic authorization includes comparing the given username and password, if the given username is ‘admin’ the get_password() function return ‘password’, which can be used as authorization for update, delete and add functions.

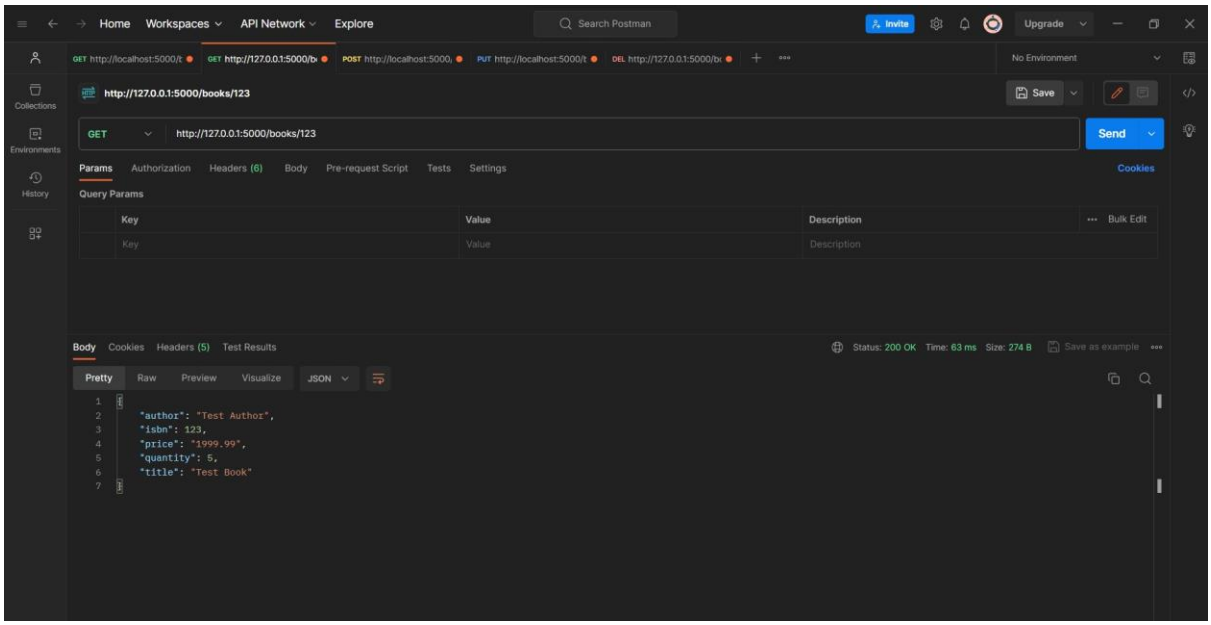
```python
@auth.get_password
def get_password(username):
    """Get the password for the given username.
    Returns the password if the username is 'admin', or None otherwise.
    """
    if username == 'admin':
        return 'password'
    return None


@auth.error_handler
def unauthorized():
    """Return an unauthorized error message.
    Returns a JSON response with a 401 status code and an error message.
    """
    return jsonify({'message': 'Unauthorized access'}), 401
```

Test (Using Postman):

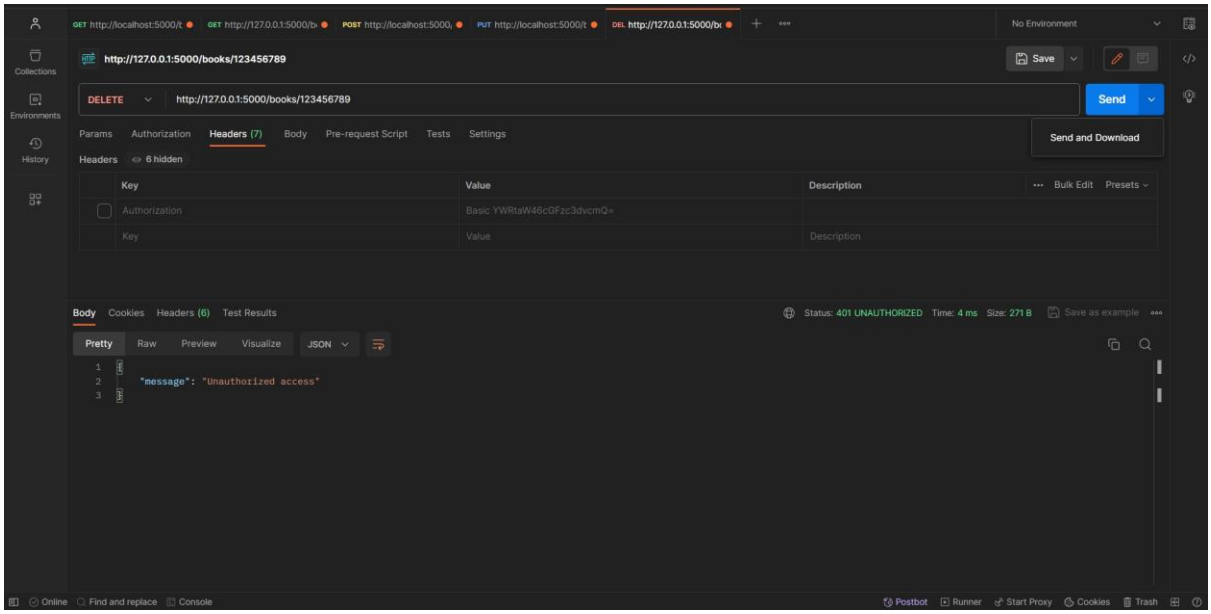
1. Retrieve all the books:

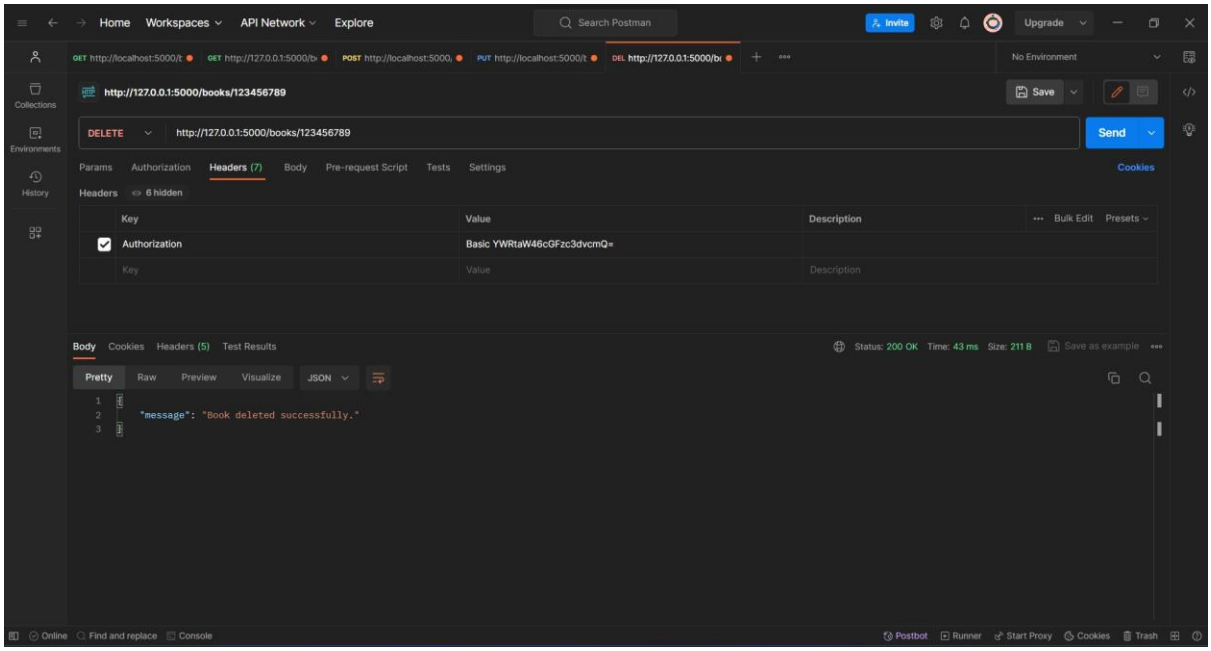


2. Retrieve Specific book



3. Delete Operation with and without Authorization:





4. PUT and POST method (including authorization check)