Python API Exercise

Case Study Title: "BookBuddy: A Book Collection Manager"

Objective:

Develop a Flask-based RESTful API for managing a personal book collection. The application will allow users to perform CRUD (Create, Read, Update, Delete) operations on a SQLite database while implementing proper error handling.

Scenario:

You have been hired by a small startup, *BookBuddy*, to create a backend system for their book collection management application. The system needs to provide a RESTful API that allows users to manage their collection of books efficiently. Each book should have attributes such as id, title, author, published_year, and genre.

The backend must be robust, scalable, and handle common errors gracefully, such as missing resources, invalid inputs, and database errors.

Requirements:

1. **Flask API**:
    - Create routes to perform CRUD operations:
        - **POST**: Add a new book.
        - **GET**: Retrieve all books or a specific book by its id.
        - **PUT**: Update an existing book's details.
        - **DELETE**: Remove a book from the collection.
2. **Database**:
    - Use SQLite to store the book data.
    - Create a table named books with the following schema:

```
CREATE TABLE books (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    author TEXT NOT NULL,
```

```
    published_year INTEGER NOT NULL,
    genre TEXT NOT NULL
);
```

3. **Error Handling:**

- Return appropriate HTTP status codes and error messages for:
    - Invalid book data (e.g., missing required fields).
    - Book not found for GET, PUT, or DELETE requests.
    - Database connection or query issues.
- Ensure all errors are returned in JSON format, such as:

```
{
    "error": "Book not found",
    "message": "No book exists with the provided ID"
}
```

4. **Bonus Requirements:**

- Implement input validation for fields like published_year (must be a valid year) and genre (must be one of predefined genres such as Fiction, Non-Fiction, Mystery, Sci-Fi, etc.).
- Implement a filter feature to retrieve books by genre or author.

Tasks:

1. **Database Setup**:
    - Write a Python script to initialize the SQLite database and create the books table.
    - Populate the table with sample data for testing.
2. **API Endpoints**:
    - Design and implement the following endpoints:
        - **POST** /books: Add a new book.
        - **GET** /books: Retrieve all books.

- **GET** /books/<id>: Retrieve a specific book by id.
- **PUT** /books/<id>: Update an existing book by id.
- **DELETE** /books/<id>: Delete a book by id.

3. **Error Handling**:
    - Ensure proper error responses are returned for:
        - Invalid data in the POST or PUT requests.
        - Nonexistent book IDs in GET, PUT, or DELETE requests.
        - Database connection failures.
4. **Testing**:
    - Test the API using a tool like Postman or curl.
    - Write Python unit tests to verify the functionality of each endpoint and error handling.

Example API Interactions:

1. **Add a New Book**:
    - Request:

```
POST /books
Content-Type: application/json
{
     "title": "The Great Gatsby",
     "author": "F. Scott Fitzgerald",
     "published_year": 1925,
     "genre": "Fiction"
}
```

    - Response:

```
{
     "message": "Book added successfully",
     "book_id": 1
}
```

2. **Retrieve All Books**:
    - Request:

      ```
      GET /books
      ```

    - Response:

      ```
      [
          {
              "id": 1,
              "title": "The Great Gatsby",
              "author": "F. Scott Fitzgerald",
              "published_year": 1925,
              "genre": "Fiction"
          },
          {
              "id": 2,
              "title": "To Kill a Mockingbird",
              "author": "Harper Lee",
              "published_year": 1960,
              "genre": "Fiction"
          }
      ]
      ```

3. **Error: Book Not Found**:
    - Request:

      ```
      GET /books/100
      ```

    - Response:

      ```
      {
          "error": "Book not found",
          "message": "No book exists with the provided
          ID"
      }
      ```