**Import Required Modules**

from flask import Flask, jsonify, request

* Flask: Imports the Flask framework, which is used to create the web application.
* jsonify: Converts objects (like dictionaries and lists) into JSON format, which is returned as a response.
* request: Allows handling incoming HTTP requests, such as **POST** and **PUT**, to extract data from the client.

**Initialize the Flask App**

app = Flask(\_\_name\_\_)

* Creates an instance of the Flask application.
* The \_\_name\_\_ argument helps Flask determine the root path of the application.

**Define Sample Data (Books)**

books = [

{"id": 1, "title": " Basics", "author": "John Doe"},

{"id": 2, "title": "Flask for Beginners", "author": "Jane Smith"}

]

* This is a **list of dictionaries**, where each dictionary represents a book with an id, title, and author.
* Acts as a temporary database.

**API Endpoints**

**1. Get All Books (GET Request)**

@app.route('/books', methods=['GET'])

def get\_books():

return jsonify(books)

* @app.route('/books', methods=['GET']): Defines a route /books that listens for **GET** requests.
* def get\_books(): Function that executes when the /books endpoint is accessed.
* jsonify(books): Converts the books list into a JSON response and returns it.

**Test in browser/Postman:**

nginx

GET http://127.0.0.1:5000/books

**Response:**

json

[

{"id": 1, "title": " Basics", "author": "John Doe"},

{"id": 2, "title": "Flask for Beginners", "author": "Jane Smith"}

]

**2. Get a Single Book by ID (GET Request)**

@app.route('/books/<int:book\_id>', methods=['GET'])

def get\_book(book\_id):

book = next((book for book in books if book["id"] == book\_id), None)

if book:

return jsonify(book)

return jsonify({"message": "Book not found"}), 404

* @app.route('/books/<int:book\_id>', methods=['GET']): Defines a route where <int:book\_id> is a dynamic parameter (expects an integer).
* book\_id: The ID extracted from the request URL.
* next((book for book in books if book["id"] == book\_id), None):
  + Loops through the books list.
  + If a book with the matching id is found, it is returned; otherwise, None is returned.
* If the book exists, it is returned as JSON.
* If not found, it returns a JSON response with {"message": "Book not found"} and **HTTP status code 404**.

**Test in browser/Postman:**

nginx

GET http://127.0.0.1:5000/books/1

**Response:**

json

{"id": 1, "title": " Basics", "author": "John Doe"}

**3. Add a New Book (POST Request)**

@app.route('/books', methods=['POST'])

def add\_book():

new\_book = request.get\_json()

books.append(new\_book)

return jsonify(new\_book), 201

* @app.route('/books', methods=['POST']): Defines a **POST** route at /books to add a new book.
* request.get\_json(): Extracts JSON data sent from the client.
* books.append(new\_book): Adds the new book to the books list.
* Returns the newly added book with **HTTP status code 201 (Created)**.

**Test in Postman/cURL:**

bash

POST http://127.0.0.1:5000/books

Content-Type: application/json

Body:

{

"id": 3,

"title": "New Book",

"author": "Alice"

}

**Response:**

json

{"id": 3, "title": "New Book", "author": "Alice"}

**4. Update an Existing Book (PUT Request)**

@app.route('/books/<int:book\_id>', methods=['PUT'])

def update\_book(book\_id):

updated\_data = request.get\_json()

for book in books:

if book["id"] == book\_id:

book.update(updated\_data)

return jsonify(book)

return jsonify({"message": "Book not found"}), 404

* @app.route('/books/<int:book\_id>', methods=['PUT']): Defines a **PUT** route at /books/<book\_id> to update an existing book.
* request.get\_json(): Extracts the updated book details from the request body.
* for book in books:: Loops through the books list to find the book with the matching ID.
* book.update(updated\_data): Updates the existing book dictionary with new values.
* If no book is found, returns **404 Not Found**.

**Test in Postman/cURL:**

bash

PUT http://127.0.0.1:5000/books/2

Content-Type: application/json

Body:

{

"title": "Flask Advanced Guide",

"author": "Jane Smith"

}

**Response:**

json

{"id": 2, "title": "Flask Advanced Guide", "author": "Jane Smith"}

**5. Delete a Book (DELETE Request)**

@app.route('/books/<int:book\_id>', methods=['DELETE'])

def delete\_book(book\_id):

global books

books = [book for book in books if book["id"] != book\_id]

return jsonify({"message": "Book deleted"}), 200

* @app.route('/books/<int:book\_id>', methods=['DELETE']): Defines a **DELETE** route at /books/<book\_id>.
* global books: Allows modifying the books list (since it’s a global variable).
* books = [book for book in books if book["id"] != book\_id]:
  + Creates a new list with all books **except** the one with the matching ID.
* Returns **200 OK** if the book is deleted.

**Test in Postman/cURL:**

nginx

DELETE http://127.0.0.1:5000/books/1

**Response:**

json

{"message": "Book deleted"}

**Run the Flask Application**

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

* if \_\_name\_\_ == '\_\_main\_\_':: Ensures that the script runs **only when executed directly**, not when imported as a module.
* app.run(debug=True): Starts the Flask development server with **debug mode enabled**, which:
  + Auto-restarts the server when code changes.
  + Shows detailed error messages in case of exceptions.

**Summary of Routes**

| **HTTP Method** | **Endpoint** | **Functionality** |
| --- | --- | --- |
| **GET** | /books | Retrieve all books |
| **GET** | /books/<book\_id> | Retrieve a single book by ID |
| **POST** | /books | Add a new book |
| **PUT** | /books/<book\_id> | Update an existing book |
| **DELETE** | /books/<book\_id> | Delete a book |

**Final Thoughts**

* This is a simple **REST API using Flask** with **CRUD operations**.
* Uses a **list** as a temporary database (can be replaced with **MySQL, PostgreSQL, or MongoDB**).
* Can be improved by **adding validation, authentication, and error handling**.