

## Book Store REST API — Steps (Flask, HTML files + JSON store)

### Files:

- app.py — the Flask app (REST API + simple HTML UI)
- books.json — optional JSON file used to load/save books (makes it easier to demo persistence)
- templates/ — separate HTML files (Bootstrap-light markup) for UI pages

You can use **Postman** to test the API endpoints (POST/GET/DELETE) and you can also use the browser UI pages to add/list/view books.

### 1 — Folder structure

```
bookstore_project/
|
├── app.py
├── books.json
├── requirements.txt
└── templates/
    ├── base.html
    ├── index.html    # home page — links to add/list UI
    ├── add_book.html # HTML form to add a book (client-side JS sends JSON)
    ├── list_books.html # list all books (renders from server)
    └── view_book.html # view a single book (renders from server)
```

### 2 — requirements.txt

Create requirements.txt:

Flask==2.3.2

(Adjust version as needed. Install with pip install -r requirements.txt.)

### 3 — Sample JSON file (optional)

Create books.json (optional initial data). If file absent, the app will start with an empty list.

books.json

```
{
  "books": [
    {
      "id": 1,
      "title": "Introduction to Python",
      "author": "A. Author",
      "price": 399.00
    },
    {
      "id": 2,
      "title": "Flask Web Development",
      "author": "B. Writer",
      "price": 499.50
    }
  ]
}
```

```
]  
}
```

#### 4 — app.py (complete code)

Create app.py with this content:

```
from flask import Flask, request, jsonify, render_template, redirect, url_for, abort
import json
import os

app = Flask(__name__)

JSON_FILE = "books.json"

# -----
# In-memory list + persistence helper
# -----
def load_books():
    """Load books from JSON_FILE if exists, otherwise return empty list."""
    if os.path.exists(JSON_FILE):
        with open(JSON_FILE, "r", encoding="utf-8") as f:
            try:
                data = json.load(f)
                return data.get("books", [])
            except json.JSONDecodeError:
                return []
    return []

def save_books(books):
    """Save books list to JSON_FILE."""
    with open(JSON_FILE, "w", encoding="utf-8") as f:
        json.dump({"books": books}, f, indent=4)

# Initialize in-memory list (loaded from JSON if present)
books = load_books()

# Maintain next id (simple incremental)
def next_id():
    if not books:
        return 1
    return max(book["id"] for book in books) + 1

# -----
# REST API Endpoints
# -----


@app.route("/add_book", methods=["POST"])
def add_book():
```

```

"""
Accepts JSON body with "title", "author", "price".
Returns JSON confirming addition.
"""

if not request.is_json:
    return jsonify({"error": "Request must be application/json"}), 415

data = request.get_json()
title = data.get("title")
author = data.get("author")
price = data.get("price")

# Basic validation
if not title or not isinstance(title, str):
    return jsonify({"error": "Field 'title' is required and must be a string"}), 400
if not author or not isinstance(author, str):
    return jsonify({"error": "Field 'author' is required and must be a string"}), 400
try:
    price_val = float(price)
    if price_val < 0:
        raise ValueError()
except Exception:
    return jsonify({"error": "Field 'price' is required and must be a non-negative number"}), 400

book = {"id": next_id(), "title": title.strip(), "author": author.strip(), "price": price_val}
books.append(book)
save_books(books) # persist to JSON file

return jsonify({"message": "Book added successfully", "book": book}), 201


@app.route("/books", methods=["GET"])
def get_books():
    """Return the complete list of books as JSON."""
    return jsonify({"books": books}), 200


@app.route("/book/<int:book_id>", methods=["GET"])
def get_book(book_id):
    """Return single book by id; 404 if not found."""
    book = next((b for b in books if b["id"] == book_id), None)
    if not book:
        return jsonify({"error": "Book not found"}), 404
    return jsonify(book), 200


@app.route("/book/<int:book_id>", methods=["DELETE"])
def delete_book(book_id):

```

```

"""Delete book by id. Returns 204 on success, 404 if id not found."""
global books
book = next((b for b in books if b["id"] == book_id), None)
if not book:
    return jsonify({"error": "Book not found"}), 404
books = [b for b in books if b["id"] != book_id]
save_books(books)
return "", 204

# -----
# Simple HTML UI (optional)
# -----
@app.route("/")
def index():
    return render_template("index.html")

@app.route("/ui/add", methods=["GET"])
def ui_add_book_form():
    # renders an HTML page with a form that uses fetch() to POST JSON to /add_book
    return render_template("add_book.html")

@app.route("/ui/books", methods=["GET"])
def ui_list_books():
    # Render server-side list for convenience
    return render_template("list_books.html", books=books)

@app.route("/ui/book/<int:book_id>", methods=["GET"])
def ui_view_book(book_id):
    book = next((b for b in books if b["id"] == book_id), None)
    if not book:
        abort(404)
    return render_template("view_book.html", book=book)

# -----
# Run app
# -----
if __name__ == "__main__":
    app.run(debug=True)

```

## 5 — HTML templates

Create templates/ folder and add the following files.

### templates/base.html

```

<!doctype html>
<html lang="en">

```

```

<head>
    <meta charset="utf-8" />
    <title>Bookstore Demo</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style>
        body { font-family: Arial, sans-serif; max-width: 800px; margin: 24px auto; }
        header { margin-bottom: 16px; }
        .card { border: 1px solid #ddd; padding: 12px; border-radius: 6px; margin-bottom: 12px; }
        a.button { display:inline-block; padding:6px 10px; background:#007bff; color:white; text-decoration:none; border-radius:4px;}
    </style>
</head>
<body>
    <header>
        <h1>Book Store Demo</h1>
        <p><a href="{{ url_for('index') }}">Home</a> | <a href="{{ url_for('ui_add_book_form') }}">Add Book (UI)</a> | <a href="{{ url_for('ui_list_books') }}">List Books (UI)</a> | <a href="{{ url_for('get_books') }}">API /books (JSON)</a></p>
        <hr>
    </header>

    {% block content %}{% endblock %}
</body>
</html>

```

### templates/index.html

```

{% extends "base.html" %}
{% block content %}
<div class="card">
    <h2>Welcome</h2>
    <p>This small app demonstrates a Book Store REST API with Flask.</p>
    <ul>
        <li>Add book via UI (sends JSON): <a href="{{ url_for('ui_add_book_form') }}">Add Book</a></li>
        <li>List books (HTML): <a href="{{ url_for('ui_list_books') }}">List Books</a></li>
        <li>API endpoints (JSON): <code>POST /add_book</code>, <code>GET /books</code>,
        <code>GET /book/&lt;id&gt;</code>, <code>DELETE /book/&lt;id&gt;</code></li>
    </ul>
</div>
{% endblock %}

```

### templates/add\_book.html

This page contains a simple HTML form and client-side JavaScript to send JSON to /add\_book.

```

{% extends "base.html" %}
{% block content %}

```

```

<div class="card">
  <h2>Add Book (UI)</h2>

  <form id="bookForm">
    <div>
      <label>Title:<br><input type="text" id="title" required style="width:100%"></label>
    </div>
    <div>
      <label>Author:<br><input type="text" id="author" required style="width:100%"></label>
    </div>
    <div>
      <label>Price:<br><input type="number" id="price" required step="0.01" style="width:100%"></label>
    </div>
    <div style="margin-top:8px;">
      <button type="submit">Add Book</button>
    </div>
  </form>

  <pre id="result" style="background:#f8f8f8; padding:8px; margin-top:8px;"></pre>
</div>

<script>
document.getElementById('bookForm').addEventListener('submit', async (e) => {
  e.preventDefault();
  const payload = {
    title: document.getElementById('title').value.trim(),
    author: document.getElementById('author').value.trim(),
    price: parseFloat(document.getElementById('price').value)
  };
  try {
    const res = await fetch('/add_book', {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify(payload)
    });
    const json = await res.json();
    document.getElementById('result').textContent = 'Status: ' + res.status + '\n' +
    JSON.stringify(json, null, 2);
    if (res.status === 201) {
      // clear form
      document.getElementById('bookForm').reset();
    }
  } catch (err) {
    document.getElementById('result').textContent = 'Error: ' + err;
  }
});
</script>

```

```
{% endblock %}
```

### templates/list\_books.html

```
{% extends "base.html" %}
{% block content %}
<div class="card">
  <h2>Books</h2>
  {% if books %}
    <ul>
      {% for b in books %}
        <li>
          <strong>{{ b.title }}</strong> by {{ b.author }} — ₹{{ '%.2f' | format(b.price) }}
          [ <a href="{{ url_for('ui_view_book', book_id=b.id) }}">view</a> ]
        </li>
      {% endfor %}
    </ul>
  {% else %}
    <p>No books yet.</p>
  {% endif %}
</div>
{% endblock %}
```

### templates/view\_book.html

```
{% extends "base.html" %}
{% block content %}
<div class="card">
  <h2>{{ book.title }}</h2>
  <p><strong>Author:</strong> {{ book.author }}</p>
  <p><strong>Price:</strong> ₹{{ '%.2f' | format(book.price) }}</p>
  <form id="deleteForm" method="post" action="#" onsubmit="return deleteBook();">
    <button type="submit" style="background:#c82333;color:white;padding:6px
10px;border:none;border-radius:4px;">Delete Book</button>
  </form>
  <p><a href="{{ url_for('ui_list_books') }}">Back to list</a></p>
</div>

<script>
async function deleteBook() {
  if (!confirm('Delete this book?')) return false;
  try {
    const res = await fetch('/book/{{ book.id }}', { method: 'DELETE' });
    if (res.status === 204) {
      alert('Deleted');
      window.location.href = "{{ url_for('ui_list_books') }}";
    } else {

```

```

    const data = await res.json();
    alert('Error: ' + JSON.stringify(data));
}
} catch (err) {
    alert('Error: ' + err);
}
return false;
}
</script>
{% endblock %}

```

## 6 — How to run (steps)

1. Open terminal and navigate into bookstore\_project folder.
2. (Optional) create virtualenv:
3. python -m venv venv
4. source venv/bin/activate # mac/linux
5. venv\Scripts\activate # windows
6. Install dependency:
7. pip install -r requirements.txt

or simply:

pip install Flask

8. Run the app:
9. python app.py

Default output: Running on http://127.0.0.1:5000/

10. Open browser:
  - o Home UI: <http://127.0.0.1:5000/>
  - o Add book UI: <http://127.0.0.1:5000/ui/add>
  - o List books UI: <http://127.0.0.1:5000/ui/books>
  - o API (JSON): <http://127.0.0.1:5000/books>

## 7 — How to test APIs using Postman / curl

### A. Add book (POST /add\_book)

- URL: [http://127.0.0.1:5000/add\\_book](http://127.0.0.1:5000/add_book)
- Method: POST
- Headers: Content-Type: application/json
- Body (raw JSON):

```
{
    "title": "Clean Code",
    "author": "Robert Martin",
    "price": 599.00
}
```

- Expected: HTTP 201 with JSON:

```
{
    "message": "Book added successfully",
    "book": {
        "id": 3,
```

```

    "title": "Clean Code",
    "author": "Robert Martin",
    "price": 599.0
}
}

curl -X POST http://127.0.0.1:5000/add_book \
-H "Content-Type: application/json" \
-d '{"title":"Clean Code","author":"Robert Martin","price":599}'

```

#### B. List all books (GET /books)

- URL: <http://127.0.0.1:5000/books>
- Method: GET
- Expected: HTTP 200 JSON with {"books": [...]}

cURL:

```
curl http://127.0.0.1:5000/books
```

#### C. Get single book (GET /book/1)

- URL: <http://127.0.0.1:5000/book/1>
- Method: GET
- If not found: returns 404 with {"error": "Book not found"}

cURL:

```
curl http://127.0.0.1:5000/book/1
```

#### D. Delete book (DELETE /book/1)

- URL: <http://127.0.0.1:5000/book/3>
- Method: DELETE
- Expected: HTTP 204 No Content (empty body)

cURL:

```
curl -X DELETE http://127.0.0.1:5000/book/3
```

## 8 — Notes

- **Why we used JSON file:** keeps things simple for beginners and demonstrates file persistence without requiring DB setup.
- **In-memory vs persistent:** the books list is in-memory but we read/write books.json so data survives restarts.
- **ID generation:** simple incremental integer. For production prefer UUIDs.
- **Validation:** basic checks included (title/author strings, price numeric). You can use libraries (Marshmallow/pydantic) for stronger validation.
- **Security:** This app is an educational demo — do not use debug mode in production and add authentication for protected actions.
- **Extensions for assignment:**
  - Add PUT/PATCH endpoint for update
  - Add search (query param ?author=...)
  - Add pagination for /books
  - Add unit tests using Flask test client

