

Introduction to REST API

REST (Representational State Transfer) is an Application Programming Interface that enables two systems to communicate seamlessly over the web using HTTP methods. It's the backbone of modern web services, allowing applications to send, receive, create, update, or delete data using standard web protocols that power today's interconnected digital world.

Core Principles of REST Architecture



Stateless

Each request is completely independent. The server doesn't store any client information between requests, ensuring reliability and scalability.



Client-Server Architecture

Clear separation between client and server promotes independent evolution and scalability of both systems.



Cacheable

Responses can be cached to dramatically improve performance and reduce server load for repeated requests.

Uniform Interface

Uses standard HTTP methods and clear, consistent resource URIs for predictable interactions.

Layered System

APIs can work through multiple architectural layers including load balancers and security systems.

HTTP Methods: The Language of REST

REST APIs communicate using four primary HTTP methods, each serving a specific purpose in data management.



GET

Retrieve data from the server without modifying anything

/api/books



POST

Create a new resource on the server

/api/books



PUT

Update an existing resource completely

/api/books/1



DELETE

Remove a resource from the server

/api/books/1

Anatomy of a REST API Request

Understanding the Components

Every REST API interaction involves four essential elements that work together to facilitate communication between systems.

01

Resource

The object of interest you're working with, such as a user profile, book record, or product listing in your database.

02

URI (Uniform Resource Identifier)

The specific path to access your resource, following a clear pattern like `/api/users/1` or `/api/products/42`.

03

Request

Sent by the client, containing the HTTP method, headers with metadata, and optional body with data to send.

04

Response

Returned by the server, including a status code indicating success or failure, plus the requested data or error information.

HTTP Status Codes: Understanding Server Responses

Status codes tell you exactly what happened with your API request. Learning these codes helps you debug issues and build robust applications.



200 OK

Success! The request was processed perfectly and data is returned.



201 Created

A new resource was successfully created on the server.



400 Bad Request

The server couldn't understand your request due to invalid syntax or data.

401 Unauthorized

Authentication credentials are missing or invalid. Login required.

404 Not Found

The requested resource doesn't exist at the specified URI.

500 Server Error

Something went wrong on the server side. Not your fault!

Advantages of REST API

- Platform-independent
- Easy to use with any programming language
- Supports multiple data formats (JSON, XML)
- Scalable and maintainable

Example REST API Request

GET Request:

```
GET /api/books/1  
Host: example.com
```

Response:

```
{  
  "bookid": 1,  
  "name": "Flask in Action",  
  "author": "John Doe"  
}
```

Flask-RESTful – Building REST APIs with Python

- A lightweight framework for developing RESTful web services.
- Extension of Flask that simplifies API creation.
- Supports easy mapping between URLs and Python classes.

What is REST?

- **REST** = *Representational State Transfer*
- Architectural style that uses **HTTP** for communication.
- Each resource is identified by a **URI (Uniform Resource Identifier)**.
- Common HTTP Methods:
 - **GET** – Retrieve data
 - **POST** – Create new data
 - **PUT** – Update data
 - **DELETE** – Remove data

What is Flask-RESTful?

- A Flask extension used to **build RESTful APIs** easily.
- Wraps Flask's functionality into a structured, class-based design.
- Provides helpful features like:
 - Resource routing
 - Request parsing
 - Error handling
 - JSON responses automatically

Installation

```
pip install flask  
pip install flask-restful
```

Basic Flask-RESTful Program

```
from flask import Flask
from flask_restful import Api, Resource

app = Flask(__name__)
api = Api(app)

class Hello(Resource):
    def get(self):
        return {"message": "Hello, Flask-RESTful!"}

api.add_resource(Hello, '/')

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

Example – CRUD on Book Resource

```
from flask import Flask, request
from flask_restful import Api, Resource

app = Flask(__name__)
api = Api(app)

books = []

class Book(Resource):
    def get(self):
        return books

    def post(self):
        data = request.get_json()
        books.append(data)
        return {"message": "Book added successfully"}, 201

api.add_resource(Book, '/books')

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

CRUD Operations

Method	Function	URL	Description
GET	get()	/books	Get all books
POST	post()	/books	Add a new book
PUT	put()	/books/<id>	Update a book
DELETE	delete()	/books/<id>	Delete a book

Adding PUT and DELETE

```
class BookById(Resource):
    def get(self, bookid):
        for book in books:
            if book['bookid'] == bookid:
                return book
        return {"message": "Book not found"}, 404

    def delete(self, bookid):
        global books
        books = [b for b in books if b['bookid'] != bookid]
        return {"message": "Book deleted successfully"}

api.add_resource(BookById, '/books/<int:bookid>')
```

JSON Request & Response Example

POST Request

```
POST /books
{
  "bookid": 1,
  "name": "Python Basics",
  "author": "John"
}
```

Response

```
{
  "message": "Book added successfully"
}
```

Key Features of Flask-RESTful

- Built-in request parsing using `reqparse`
- Automatic JSON conversion
- Custom error handling
- Supports resource nesting
- Easy integration with databases (SQLite, MySQL, MongoDB)

Advantages

- Easy to learn and use
- Lightweight and flexible
- Minimal boilerplate code
- Supports scalable, modular development

Real-World Use Cases

- Backend for web & mobile apps
- Connecting frontends (React, Angular, Vue) to Python APIs
- Microservices development
- IoT and data exchange platforms

Flask-REStful CRUD Example (Book Resource)

```
from flask import Flask, request from flask_restful import Api, Resource
```

```
app = Flask(name) api = Api(app)
```

```
books = [ {"bookid": 1, "name": "Python Basics", "author": "John"}, {"bookid": 2, "name": "Flask Guide", "author": "Mary"} ]
```

```
class BookList(Resource): def get(self): return books
```

```
def post(self):  
    new_book = request.get_json()  
    books.append(new_book)  
    return new_book, 201
```

```
api.add_resource(BookList, '/books')
```

```
if name == 'main': app.run(debug=True)
```

Handling Specific Book Operations (GET, PUT, DELETE)

```
class Book(Resource):  
    def get(self, bookid):  
        for book in books:  
            if book['bookid'] == bookid:  
                return book  
        return {"message": "Book not found"}, 404
```

```
    def put(self, bookid):  
        data = request.get_json()  
        for book in books:  
            if book['bookid'] == bookid:  
                book.update(data)  
                return {"message": "Book updated successfully"}  
        return {"message": "Book not found"}, 404  
  
    def delete(self, bookid):  
        global books  
        books = [book for book in books if book['bookid'] != bookid]  
        return {"message": "Book deleted successfully"}
```

```
api.add_resource(Book, '/books/int:bookid')
```

Testing Your API

GET Request

```
GET http://localhost:5000/books
```

POST Request

```
POST http://localhost:5000/books
```

```
Body (JSON):
```

```
{  
  "bookid": 3,  
  "name": "REST API with Flask",  
  "author": "Sara"  
}
```

Response

```
{"bookid": 3, "name": "REST API with Flask", "author": "Sara"}
```

Advantages of Flask-RESTful

- ✓ Lightweight and easy to learn
- ✓ Follows clear REST conventions
- ✓ Automatic JSON support
- ✓ Simple error handling and request parsing
- ✓ Easily integrates with databases (SQLite, MySQL, MongoDB)

Thank You