Lab6

NYCU Go Programming 2024

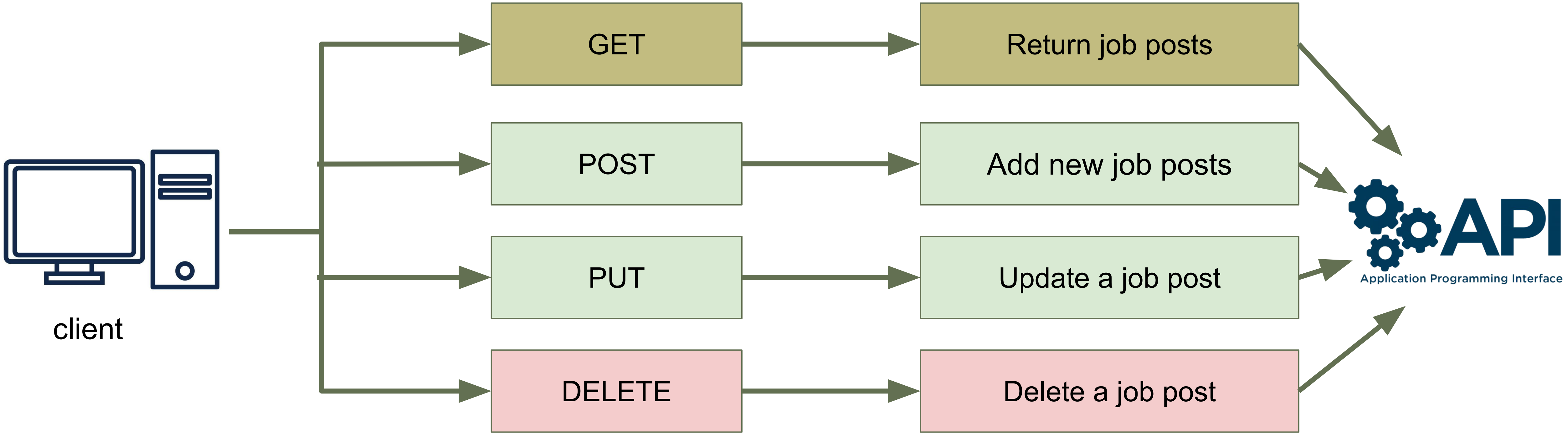
2024/11/12

RESTful API

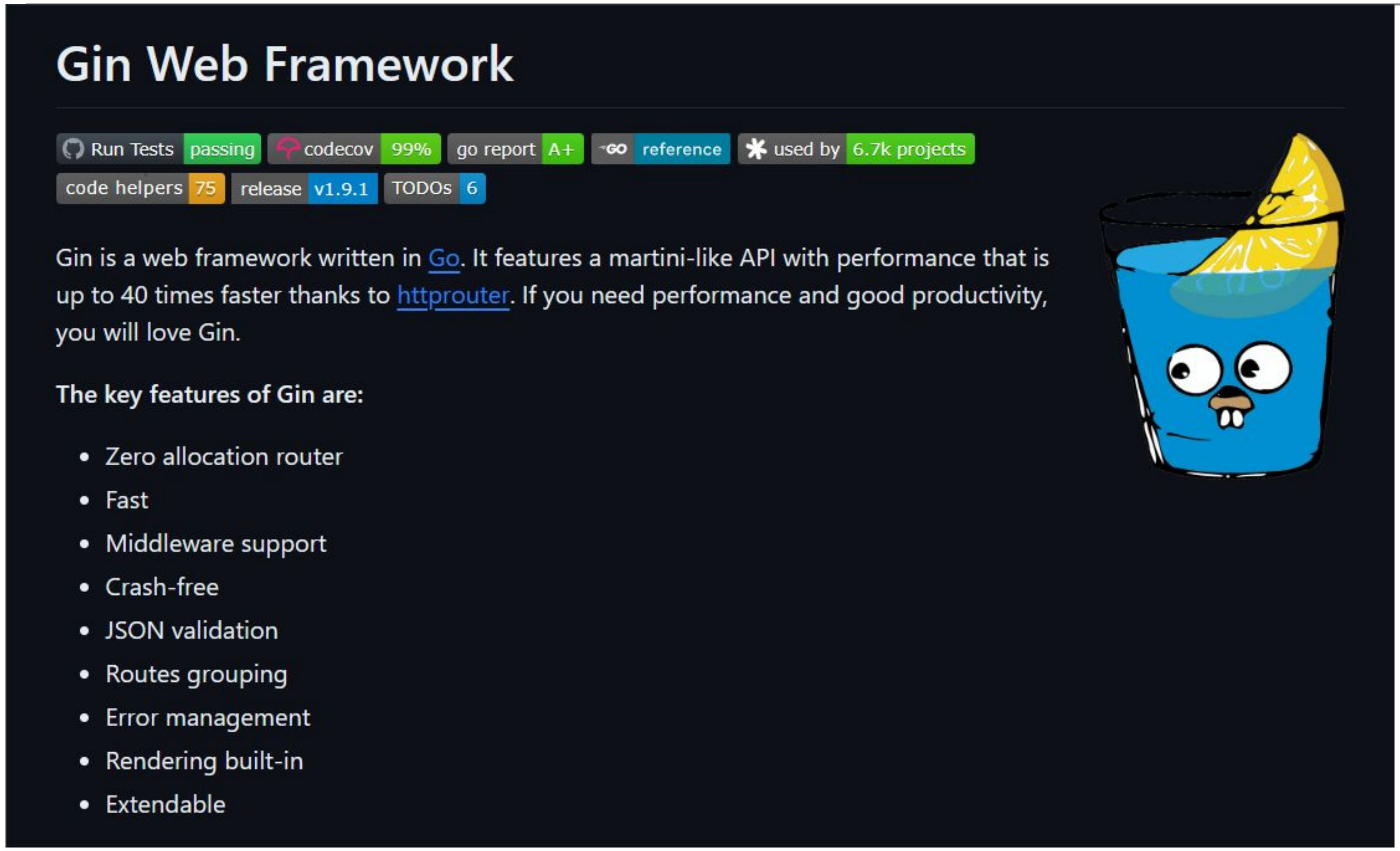
- REST (Representational state transfer) is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web.
- Web service APIs that adhere to the REST architectural constraints are called RESTful APIs.

RESTful API (Cont.)

Nouns 名詞	Verbs 動詞	Content Types
定義資源位置的 URL	對資源要做的動作	資源呈現方式:API 資源可以以多種方式表現
/book	Get、Post、Delete、Put	json、xml、yaml


client
GET
POST
PUT
DELETE
Return job posts
Add new job posts
Update a job post
Delete a job post
API
Application Programming Interface

Gin



Gin Web Framework

Run Tests passing | codecov 99% | go report A+ | GO reference | used by 6.7k projects | code helpers 75 | release v1.9.1 | TODOs 6

Gin is a web framework written in Go. It features a martini-like API with performance that is up to 40 times faster thanks to httprouter. If you need performance and good productivity, you will love Gin.

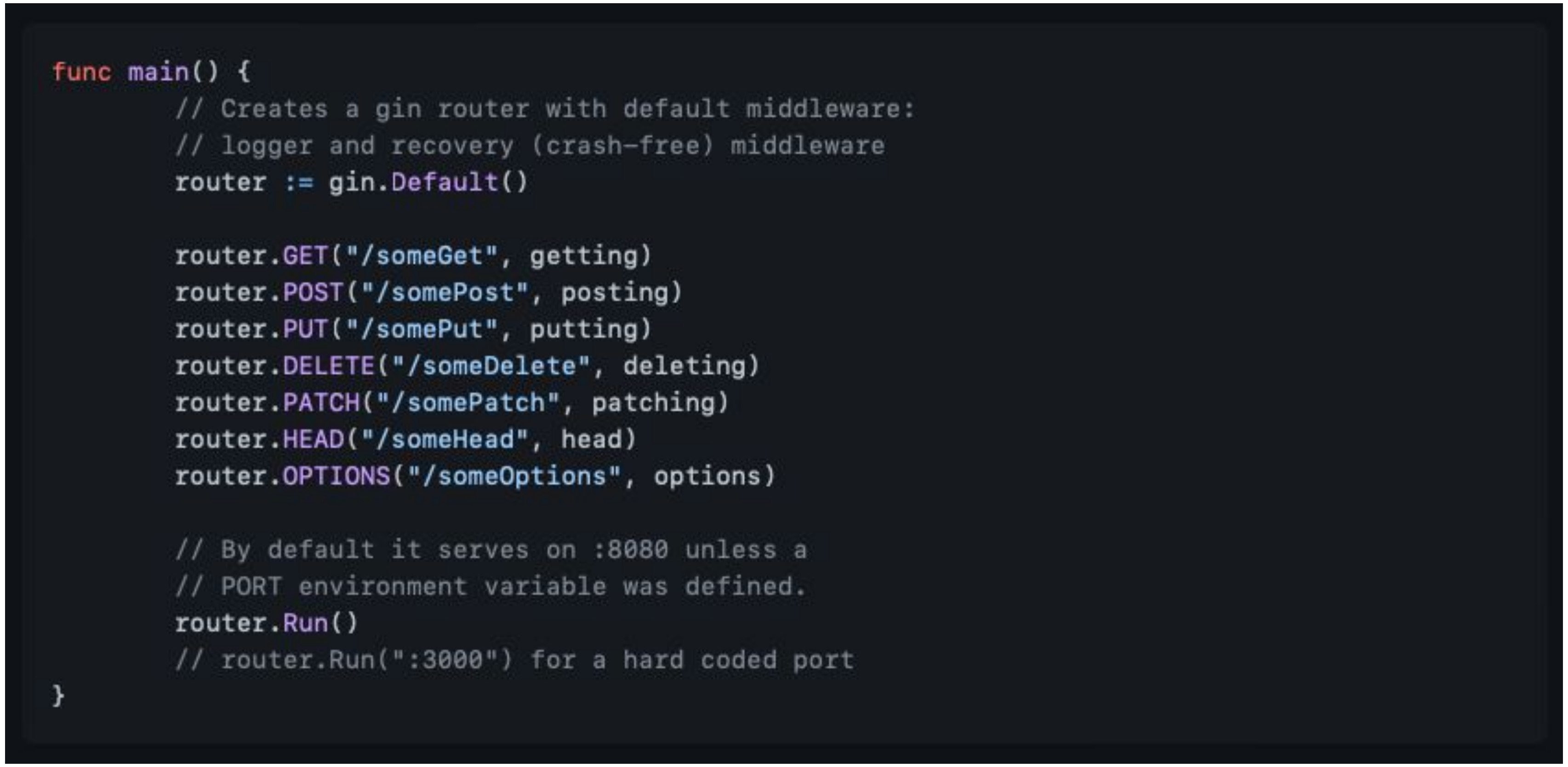
The key features of Gin are:

- Zero allocation router
- Fast
- Middleware support
- Crash-free
- JSON validation
- Routes grouping
- Error management
- Rendering built-in
- Extendable

- type Engine
  - Engine is the framework's instance, it contains the muxer, middleware and configuration settings. Create an instance of Engine, by using New() or Default()

- func Default() *Engine
  - Default returns an Engine instance with the Logger and Recovery middleware already attached.
- func (group *RouterGroup) Handle(httpMethod, relativePath string, handlers ...HandlerFunc) Iroutes
  - Handle registers a new request handle and middleware with the given path and method. The last handler should be the real handler, the other ones should be middleware that can and should be shared among different routes.
- func (group *RouterGroup) GET(relativePath string, handlers ...HandlerFunc) Iroutes
  - GET is a shortcut for router.Handle("GET", path, handlers).
- …

Examples



```
func main() {
        // Creates a gin router with default middleware:
        // logger and recovery (crash-free) middleware
        router := gin.Default()

        router.GET("/someGet", getting)
        router.POST("/somePost", posting)
        router.PUT("/somePut", putting)
        router.DELETE("/someDelete", deleting)
        router.PATCH("/somePatch", patching)
        router.HEAD("/someHead", head)
        router.OPTIONS("/someOptions", options)

        // By default it serves on :8080 unless a
        // PORT environment variable was defined.
        router.Run()
        // router.Run(":3000") for a hard coded port
}
```

Lab6: Bookshelf API

- 寫出一個電子書架的 API (資料格式使用 JSON)，含有五個功能

1. 列出所有書 (GET)
2. 列出單一本書 (GET)
3. 加入單一本書 (POST)
4. 刪除單一本書 (DELETE)
5. 更新單一本書 (PUT)

- 預設資料要包含：

  -
    ```
    {
        "id": "1",
        "name": "Blue Bird",
        "pages": "500"
    }
    ```
- 刪除一本書之後，之後再新增書本，它的 ID 繼續往上增加
  - （例如有 ID 1, 2, 3 的書本，將 ID 為 2 的書本刪除後，再新增時其 ID 為 4）

Lab6: Bookshelf API

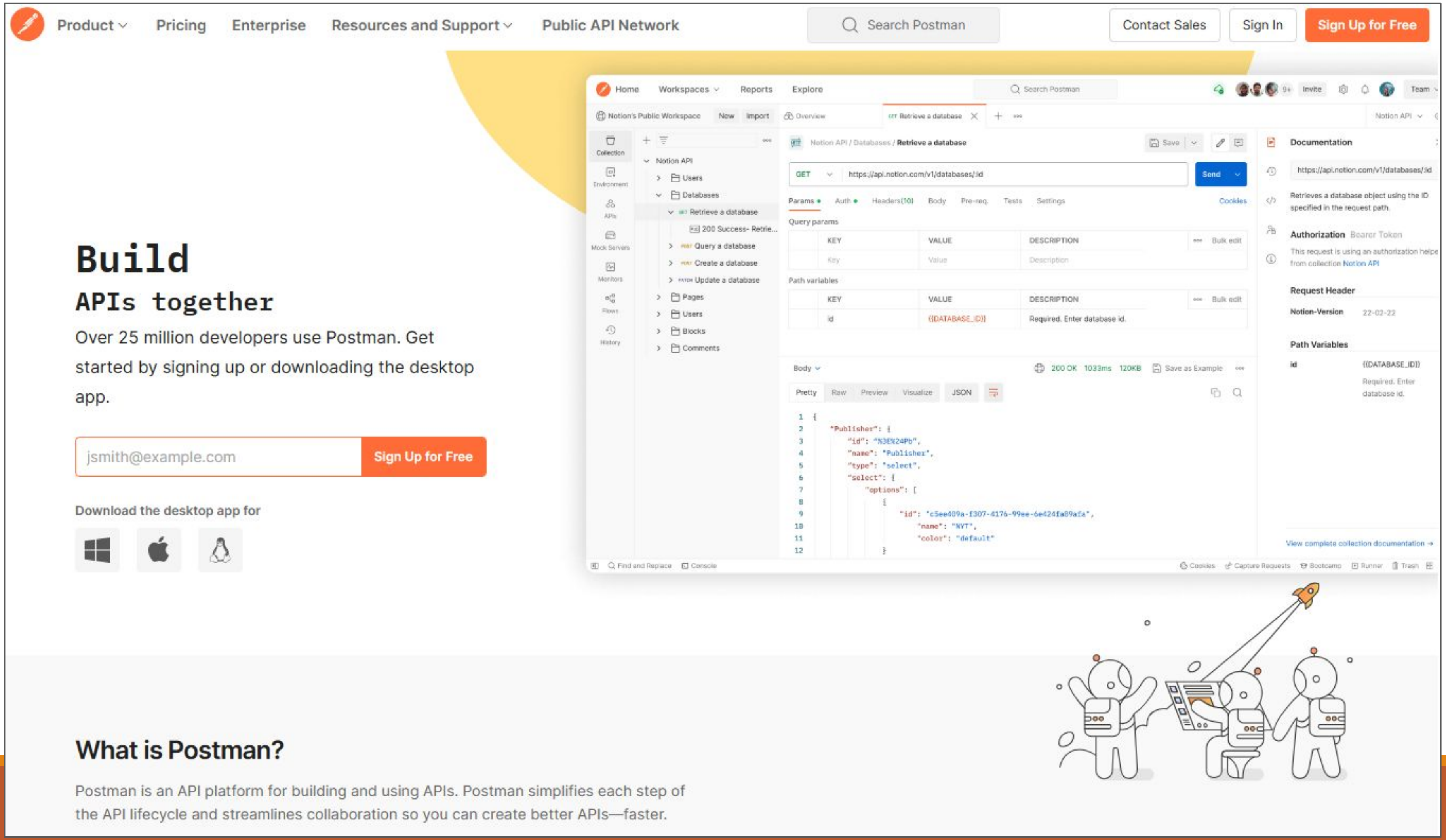
- .github
  - workflows
    - lab6.yml
- lab6
  - go.mod
  - go.sum
  - lab6.go
  - lab6_test.go

```
$ go test
```

How to Test API

- Postman
- curl
  - curl is a tool for transferring data from or to a server.

Postman



Notion API / Databases / **Retrieve a database**

GET https://api.notion.com/v1/databases/:id

Query params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Path variables

KEY	VALUE	DESCRIPTION
id	{{DATABASE_ID}}	Required. Enter database id.

Body 200 OK 1033ms 120KB

```
{
    "Publisher": {
        "id": "%3E%24Pb",
        "name": "Publisher",
        "type": "select",
        "select": {
            "options": [
                {
                    "id": "c5ee409a-f307-4176-99ee-6e424fa89afa",
                    "name": "NYT",
                    "color": "default"
                }
```

Documentation

https://api.notion.com/v1/databases/:id

Retrieves a database object using the ID specified in the request path.

Authorization Bearer Token

This request is using an authorization helpe from collection Notion API

Request Header

Notion-Version	22-02-22

Path Variables

id	{{DATABASE_ID}} Required. Enter database id.

View complete collection documentation →

Build
APIs together

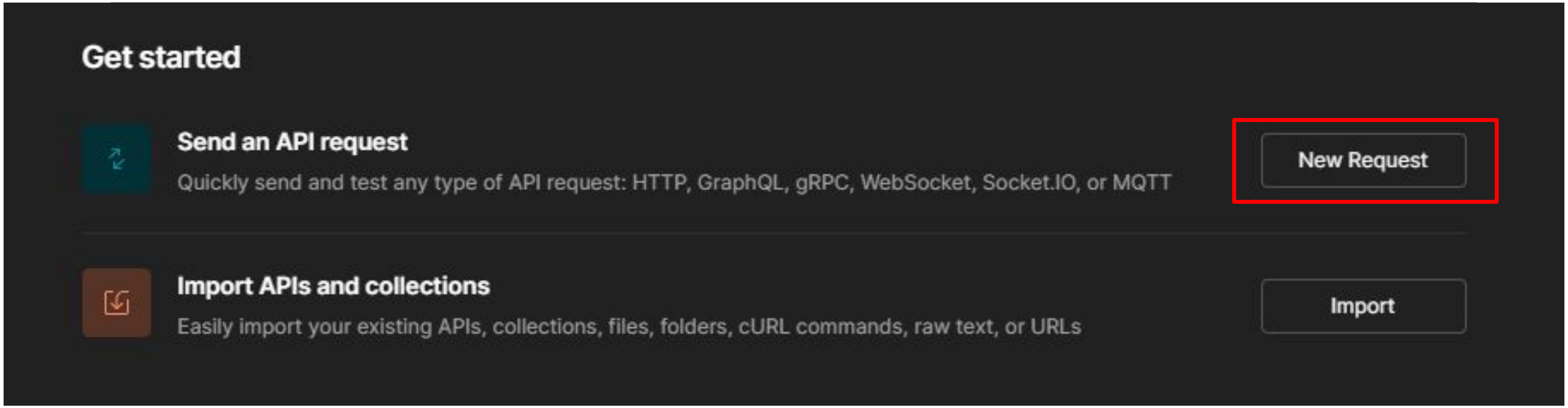
Over 25 million developers use Postman. Get started by signing up or downloading the desktop app.

jsmith@example.com Sign Up for Free

Download the desktop app for

What is Postman?

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.

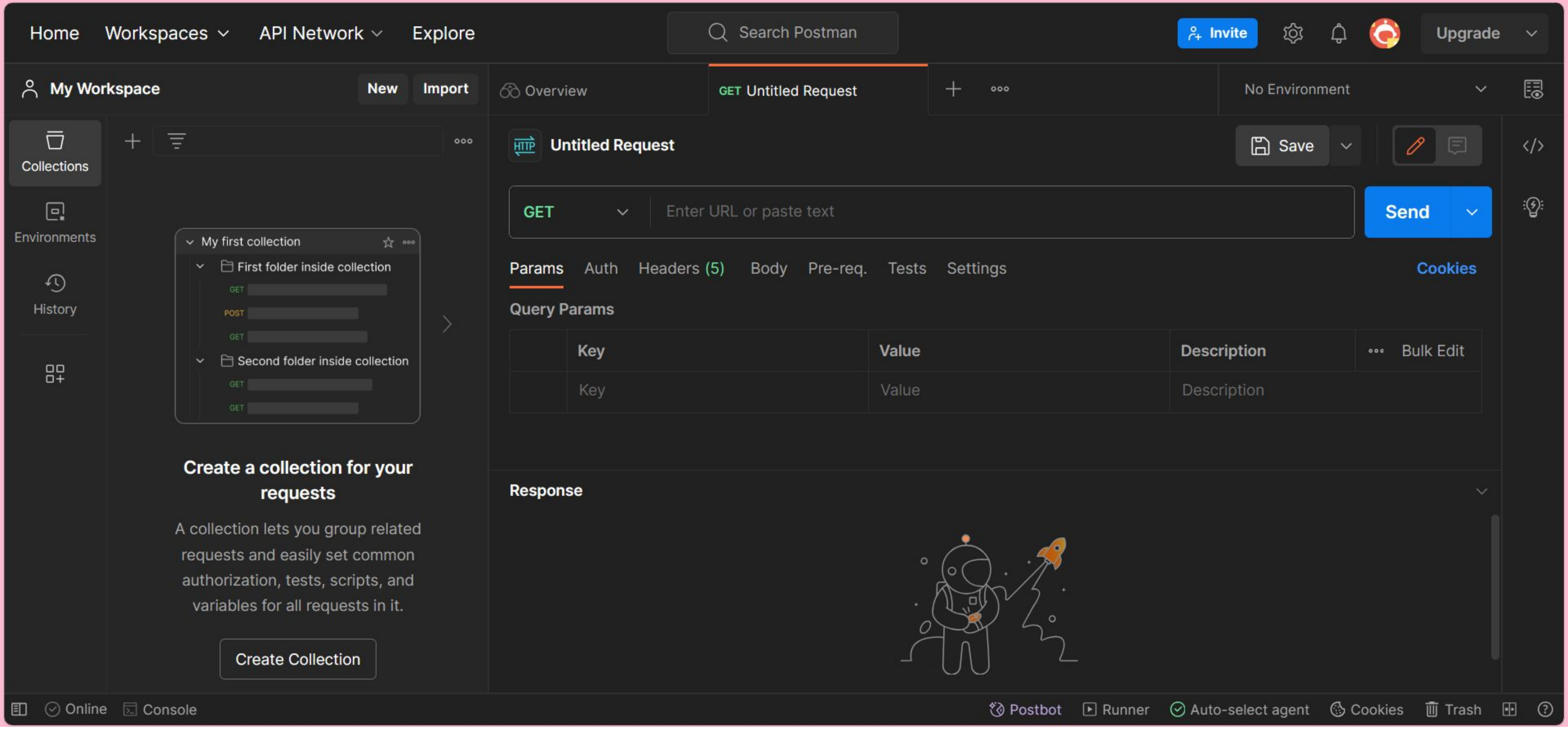

Get started

Send an API request

Quickly send and test any type of API request: HTTP, GraphQL, gRPC, WebSocket, Socket.IO, or MQTT

Import APIs and collections

Easily import your existing APIs, collections, files, folders, cURL commands, raw text, or URLs


Home Workspaces API Network Explore

Search Postman

Invite Upgrade

My Workspace New Import

Overview GET Untitled Request No Environment

Collections Environments History

My first collection
First folder inside collection
GET
POST
GET
Second folder inside collection
GET
GET

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create Collection

Untitled Request Save

GET Enter URL or paste text Send

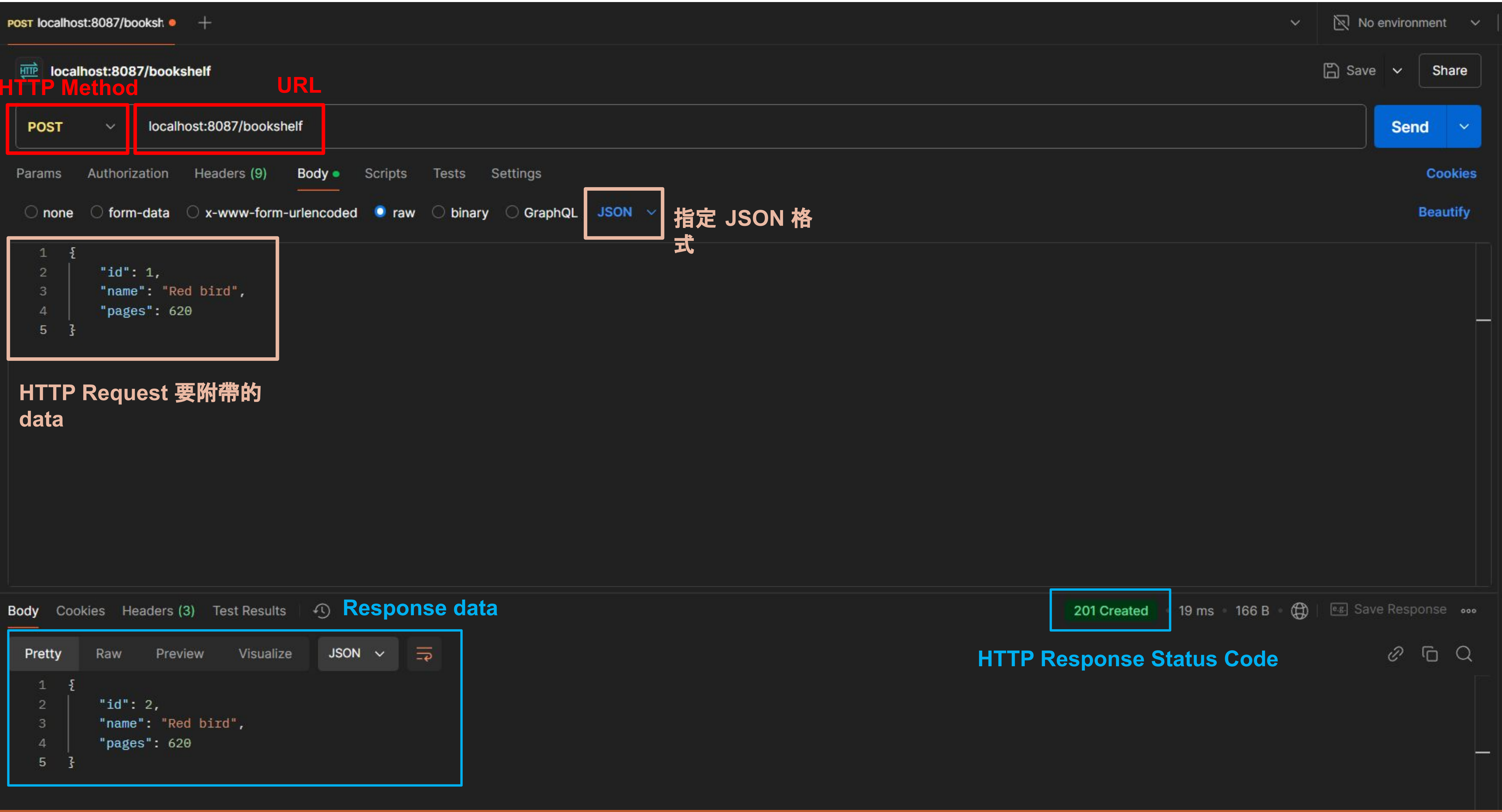
Params Auth Headers (5) Body Pre-req. Tests Settings Cookies

Query Params

	Key	Value	Description	••• Bulk Edit
	Key	Value	Description	

Response

Online Console Postbot Runner Auto-select agent Cookies Trash



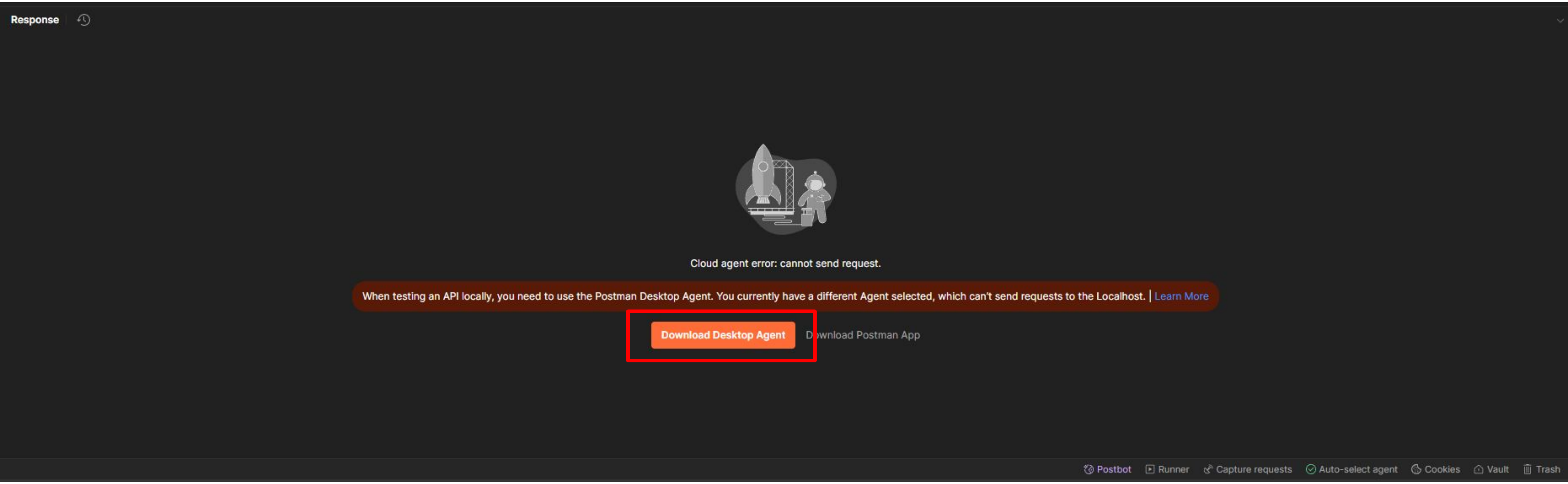
POST localhost:8087/bookshelf

Request body (raw, JSON):

```
{
    "id": 1,
    "name": "Red bird",
    "pages": 620
}
```

Response (201 Created):

```
{
    "id": 2,
    "name": "Red bird",
    "pages": 620
}
```

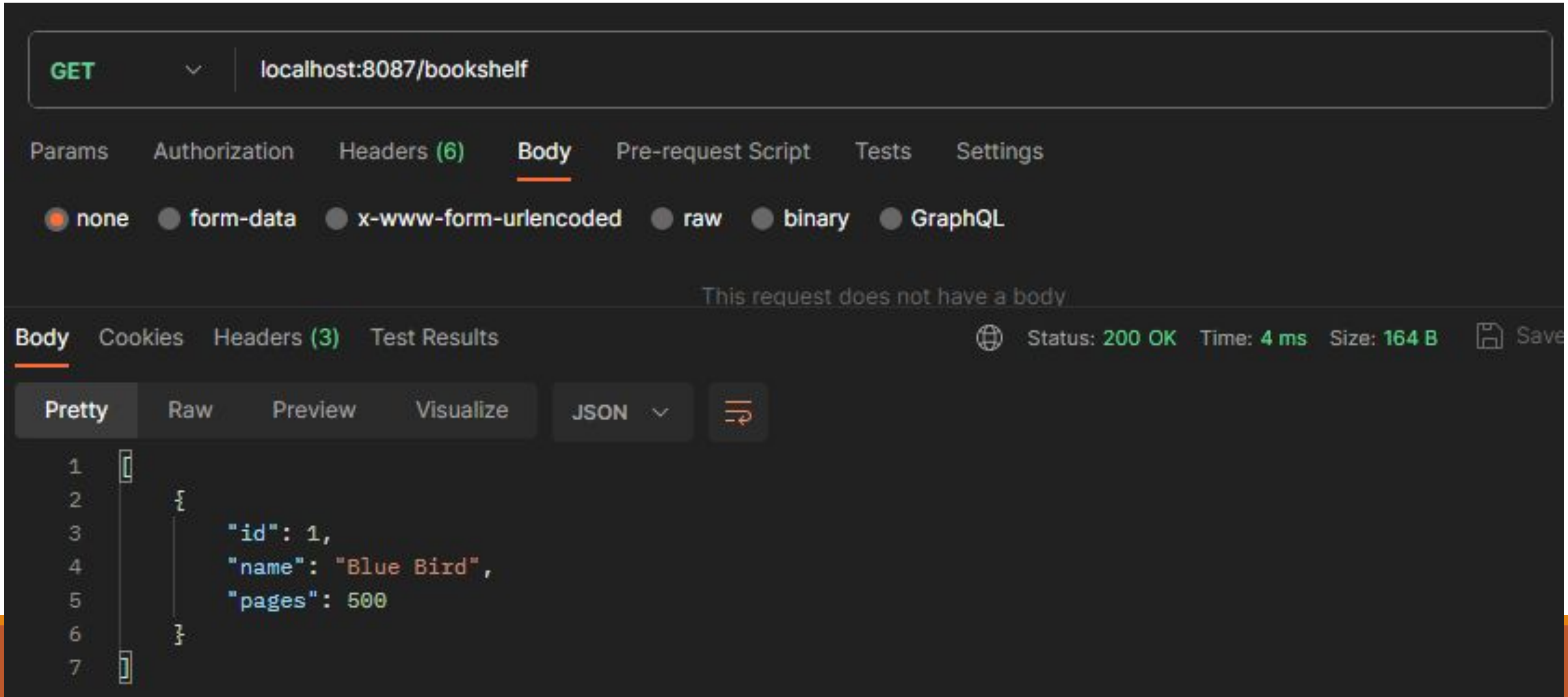

Response

Cloud agent error: cannot send request.

When testing an API locally, you need to use the Postman Desktop Agent. You currently have a different Agent selected, which can't send requests to the Localhost. | Learn More

GET example

- List all books



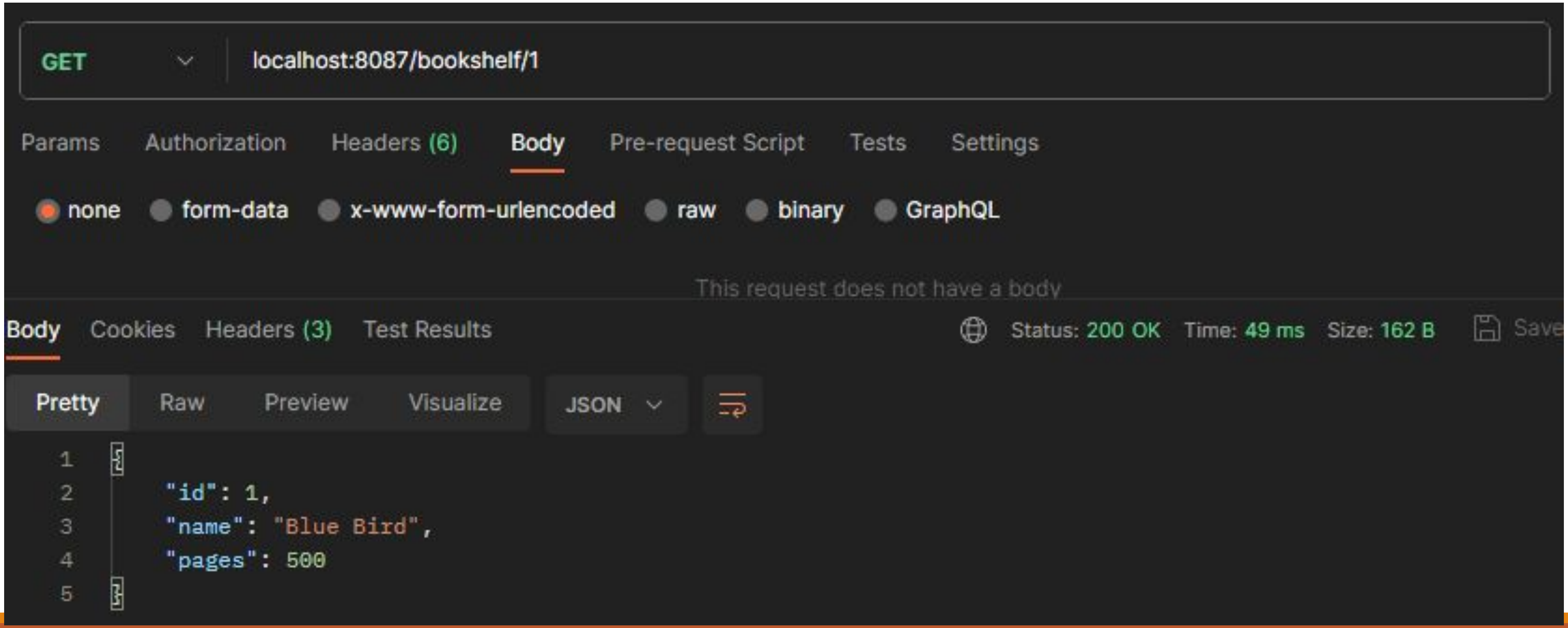
This request does not have a body

Status: 200 OK Time: 4 ms Size: 164 B

```
[
    {
        "id": 1,
        "name": "Blue Bird",
        "pages": 500
    }
]
```

GET example

- List a book with a specific ID



localhost:8087/bookshelf/1

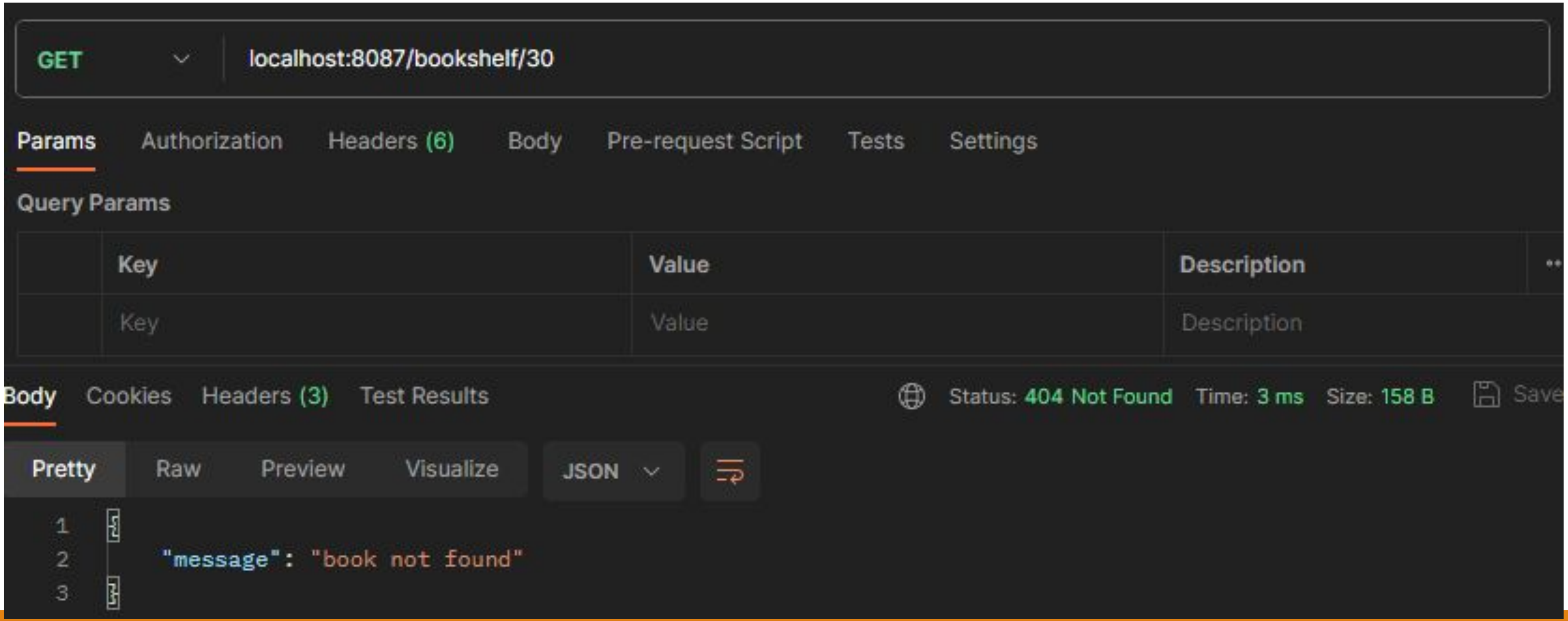
This request does not have a body

Status: 200 OK Time: 49 ms Size: 162 B

```
{
    "id": 1,
    "name": "Blue Bird",
    "pages": 500
}
```

GET example – Error handling

- Get 404 with error message when book ID is non-existent



GET localhost:8087/bookshelf/30

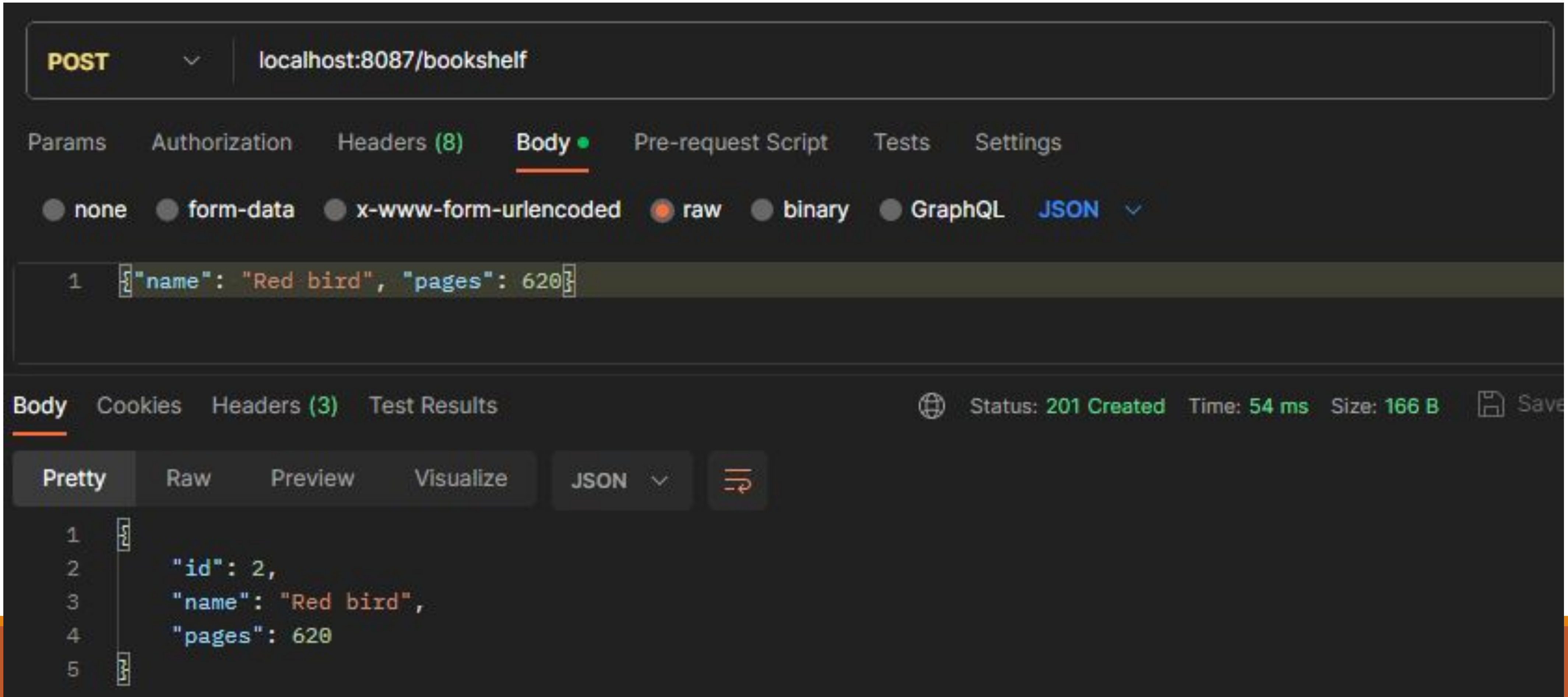
Key	Value	Description
Key	Value	Description

Status: 404 Not Found Time: 3 ms Size: 158 B

```
{
    "message": "book not found"
}
```

POST example

- Add a book. Notice that only "name" and "pages" are provided. The "id" should be generated by server.



POST localhost:8087/bookshelf

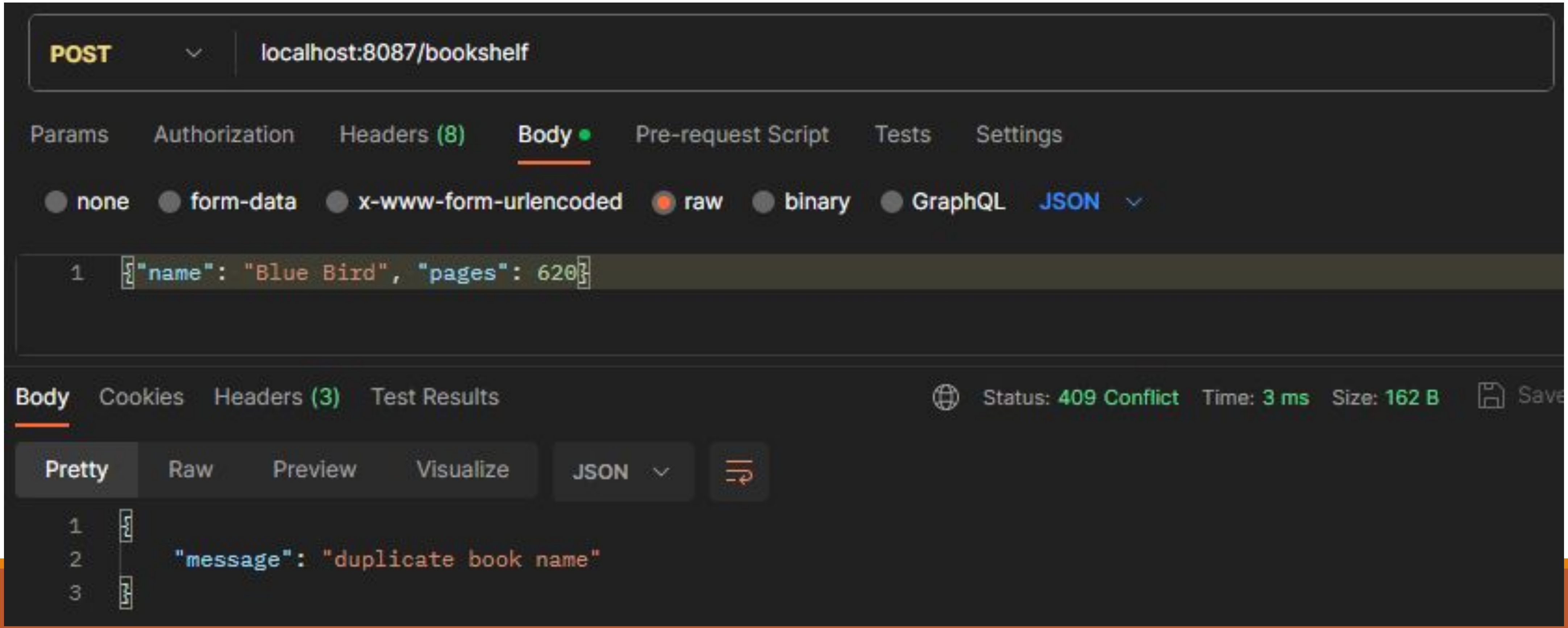
```
{"name": "Red bird", "pages": 620}
```

Status: 201 Created Time: 54 ms Size: 166 B

```
{
    "id": 2,
    "name": "Red bird",
    "pages": 620
}
```

POST example – Error handling

- Get 409 with error message when book name duplicates
- Book name is case sensitive


POST localhost:8087/bookshelf

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{"name": "Blue Bird", "pages": 620}
```

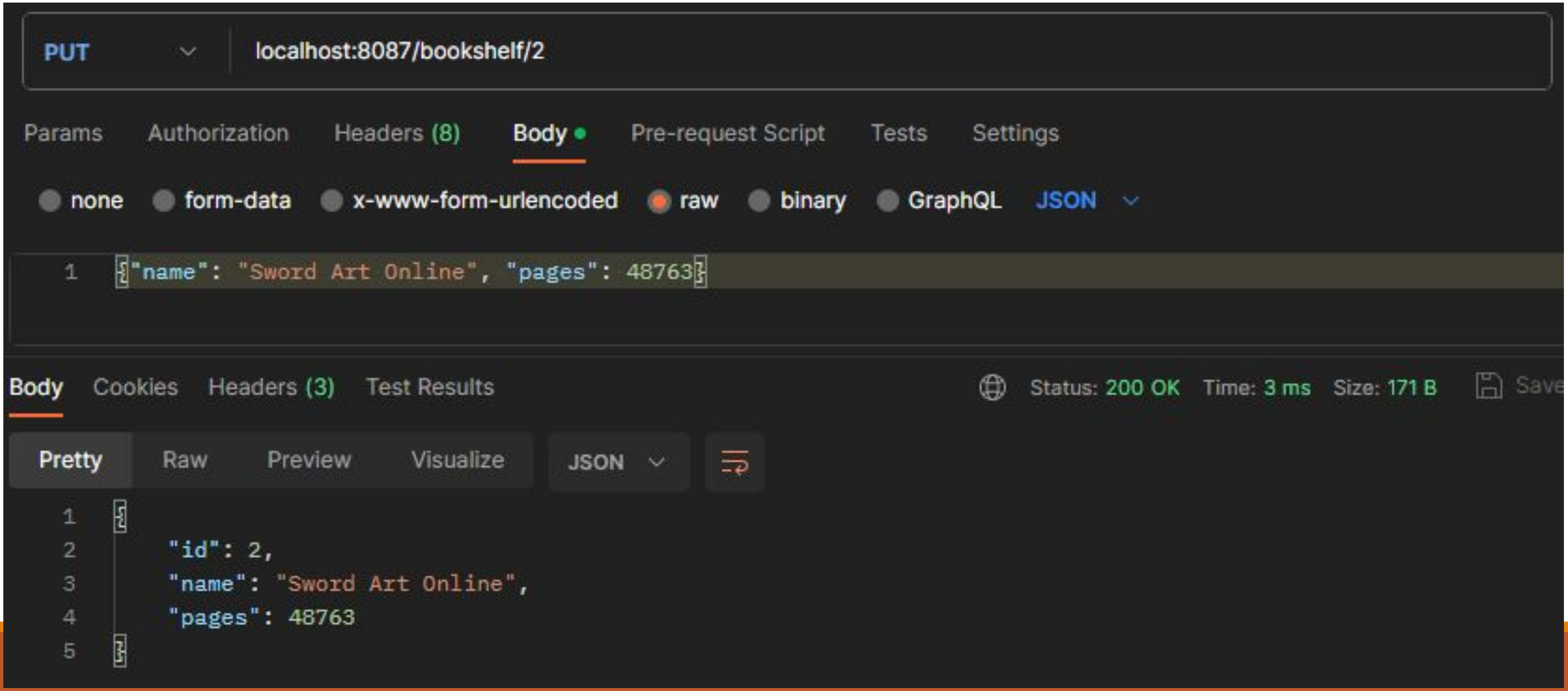
Body Cookies Headers (3) Test Results — Status: 409 Conflict Time: 3 ms Size: 162 B Save

Pretty Raw Preview Visualize JSON

```
{
    "message": "duplicate book name"
}
```

PUT example

- Update a book with a specific ID and book information



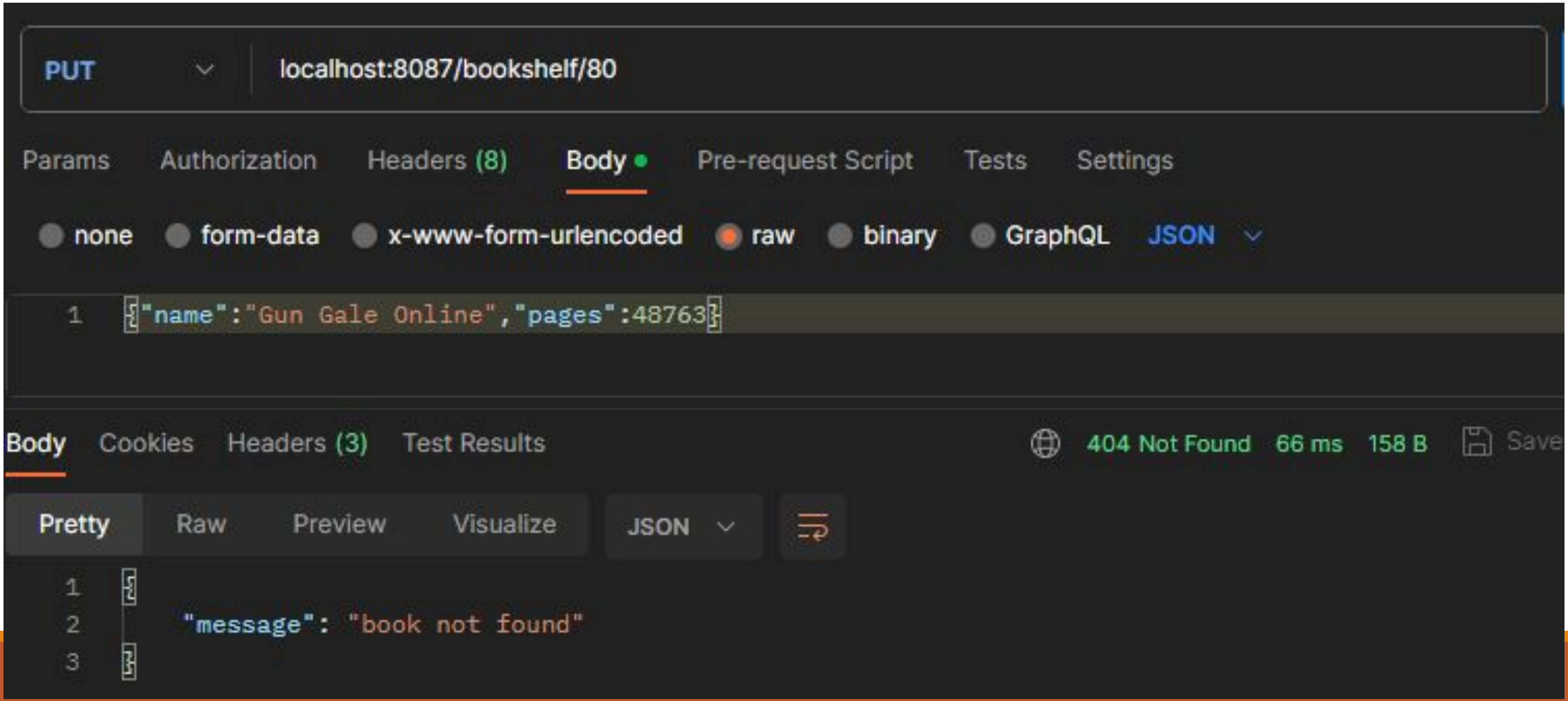
PUT localhost:8087/bookshelf/2

```
{"name": "Sword Art Online", "pages": 48763}
```

```
{
    "id": 2,
    "name": "Sword Art Online",
    "pages": 48763
}
```

PUT example – Error handling

- Get 404 with error message when ID is non-existent



```
localhost:8087/bookshelf/80
```

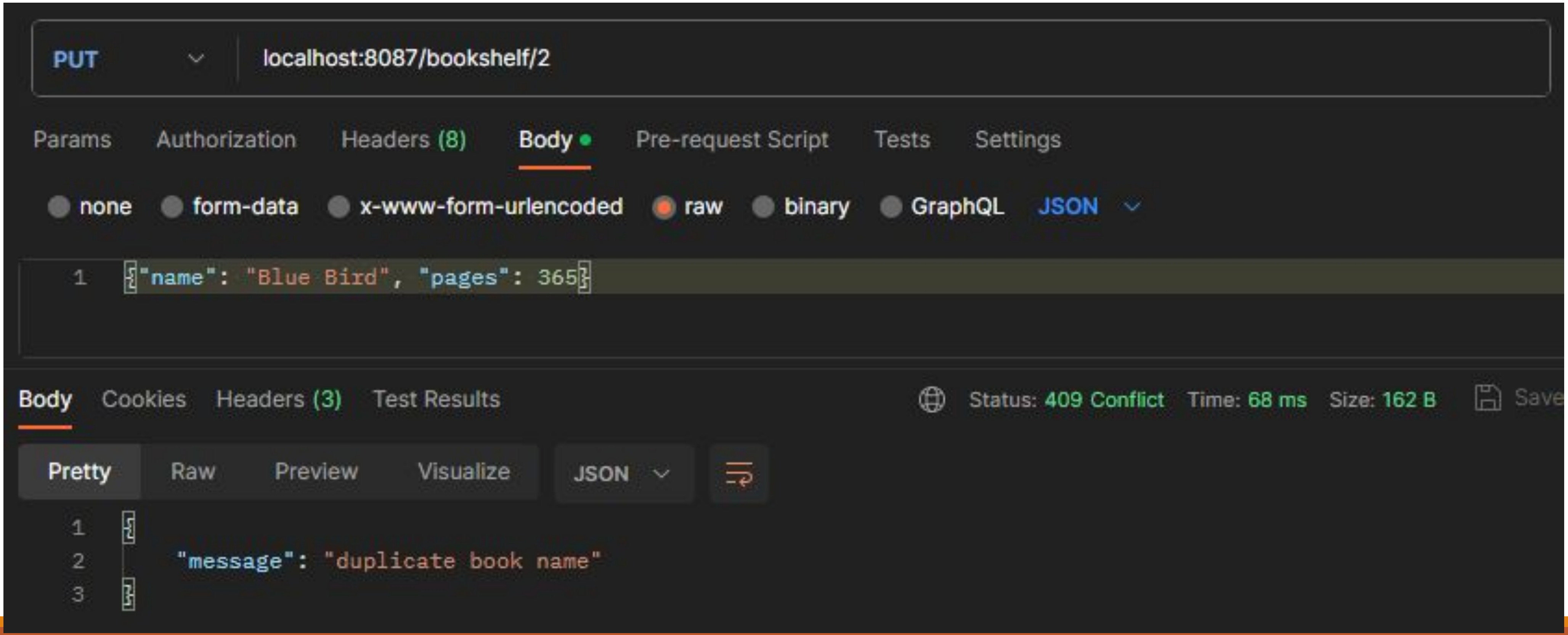
```
{"name":"Gun Gale Online","pages":48763}
```

404 Not Found 66 ms 158 B

```
{
    "message": "book not found"
}
```

PUT example – Error handling

- Get 409 with error message when book name duplicates



PUT localhost:8087/bookshelf/2

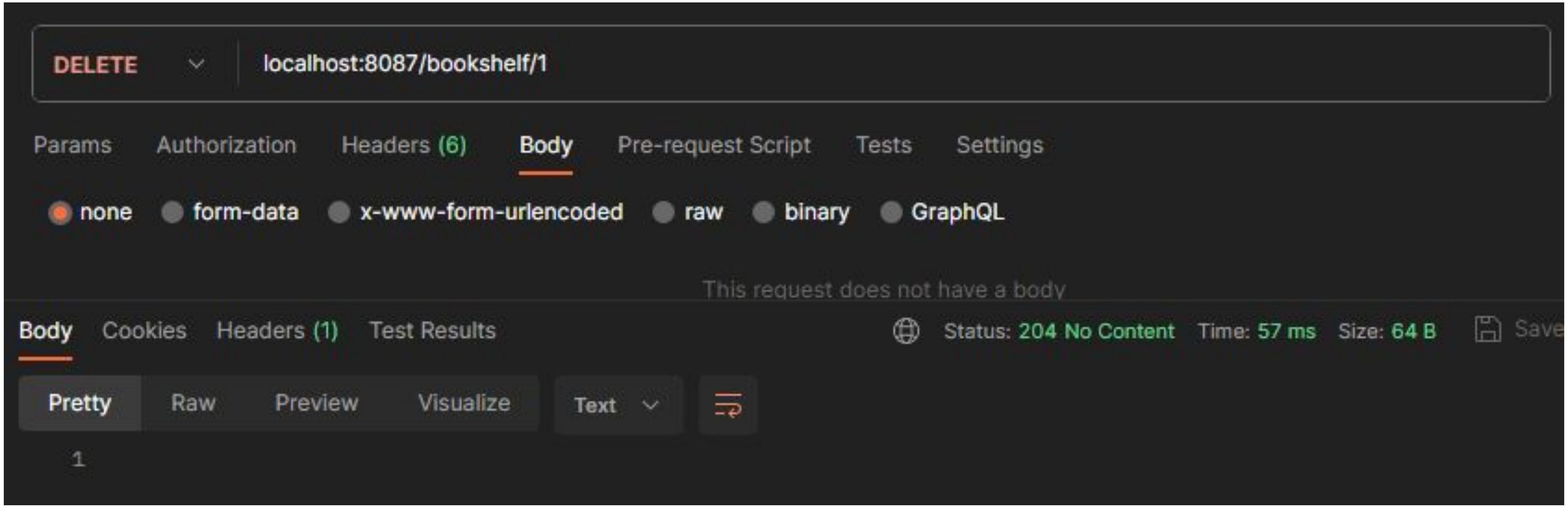
```
{"name": "Blue Bird", "pages": 365}
```

Status: 409 Conflict Time: 68 ms Size: 162 B

```
{
    "message": "duplicate book name"
}
```

DELETE example

- Delete a book with a specific ID



DELETE localhost:8087/bookshelf/1

Params Authorization Headers (6) Body Pre-request Script Tests Settings

● none ○ form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL

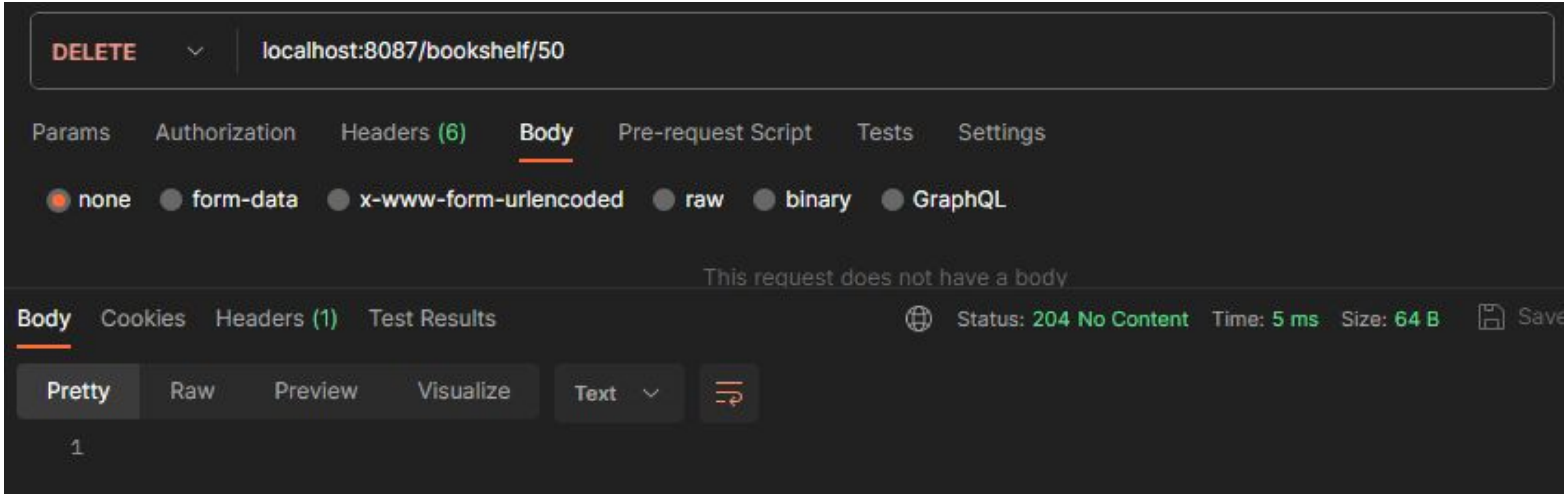
This request does not have a body

Body Cookies Headers (1) Test Results Status: 204 No Content Time: 57 ms Size: 64 B Save

Pretty Raw Preview Visualize Text

DELETE example

- Delete a non-existent ID → <u>also return 204</u>



DELETE localhost:8087/bookshelf/50

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

[✓ none] form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body | Cookies | Headers (1) | Test Results — Status: 204 No Content Time: 5 ms Size: 64 B

Pretty | Raw | Preview | Visualize | Text

```
1
```