

Documentation: Consists of Introduction, Routes, and Errors.

Introduction:

This API facilitates the storage of product info for a shop.

An item object is created and stored in the database. It contains the following structure:

ITEM

Property Name	Datatype	Comments
prodId	string	PRIMARY KEY, UNIQUE
name	string	
cost	float	
quantity	positive integer	

Routes:

- 1) '/', Method: GET

This route responds with "Welcome!". It is a test route to check if the server is online.

The screenshot shows a REST client interface with the following details:

- Header bar: Body, Cookies, Headers (7), Test Results.
- Status bar: 200 OK, 5 ms, 233 B, Save Response.
- Content area:
 - Pretty tab selected.
 - Raw, Preview, Visualize, HTML dropdown.
 - Response body: 1 Welcome

- 2) '/shop' Method: GET

This route responds with an array containing all of the items in the stock database.

The screenshot shows a REST client interface with the following details:

- Header bar: Body, Cookies, Headers (6), Test Results.
- Status bar: 200 OK, 7 ms, 378 B, Save Response.
- Content area:
 - Pretty tab selected.
 - Raw, Preview, Visualize, JSON dropdown.
 - Response body (JSON array):

```
1 [
2   {
3     "prodId": "a001",
4     "name": "bread",
5     "cost": "9.99",
6     "quantity": "40"
7   },
8   {
9     "prodId": "a002",
10    "name": "chees",
11    "cost": "19.99",
12    "quantity": "20"
13  },
14  {
15    "prodId": "a003",
16    "name": "pizza",
17    "cost": "14.99",
18    "quantity": "60"
19 }
```

3) '/shop/add' Method: POST

This route receives item data in the request body, checks for duplication, adds the item to the stock database, and responds with an array containing all of the items in stock.

POST localhost:3030/shop/add Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

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

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> prodId	a001			
<input checked="" type="checkbox"/> name	bread			
<input checked="" type="checkbox"/> cost	9.99			
<input checked="" type="checkbox"/> quantity	40			
Key	Value	Description		

Body Cookies Headers (6) Test Results 200 OK 4 ms 250 B Save Response

Pretty Raw Preview Visualize JSON JSON

```

1 [
2   {
3     "prodId": "a001",
4     "name": "bread",
5     "cost": "9.99",
6     "quantity": "40"
7   }
8 ]

```

4) '/shop/edit' Method: PUT

This route receives item data in the request body, checks if the item is found, edits the item info in the database, and responds with an array containing all of the items in stock.

PUT localhost:3030/shop/edit Send

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings Cookies

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

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> prodId	a002			
<input checked="" type="checkbox"/> name	cheese			
<input checked="" type="checkbox"/> cost				
<input checked="" type="checkbox"/> quantity				

Body Cookies Headers (6) Test Results 200 OK 5 ms 379 B Save Response

Pretty Raw Preview Visualize JSON JSON

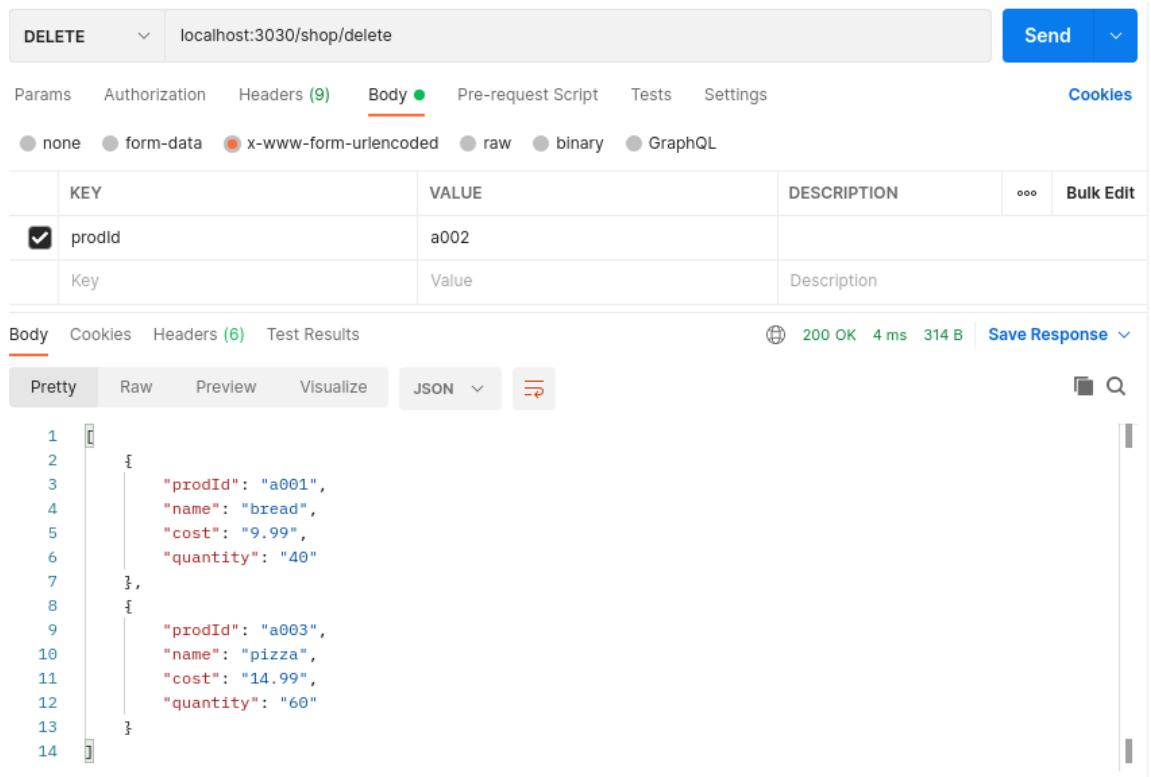
```

1 [
2   {
3     "prodId": "a001",
4     "name": "bread",
5     "cost": "9.99",
6     "quantity": "40"
7   },
8   {
9     "prodId": "a002",
10    "name": "cheese",
11    "cost": "19.99",
12    "quantity": "20"
13  },
14  {
15    "prodId": "a003",
16    "name": "milk",
17    "cost": "3.99",
18    "quantity": "100"
19  }
20 ]

```

5) '/shop/delete' Method: DELETE

This route receives item prodId in the request body, checks if the item is found, deletes the item, and responds with an array containing all of the items in stock.



The screenshot shows a Postman request configuration for a DELETE method. The URL is set to `localhost:3030/shop/delete`. The **Body** tab is selected, showing a JSON object with two properties: `prodId` (value: `a002`) and `Key` (value: `Description`). Below the body, the response status is shown as `200 OK` with `4 ms` and `314 B`.

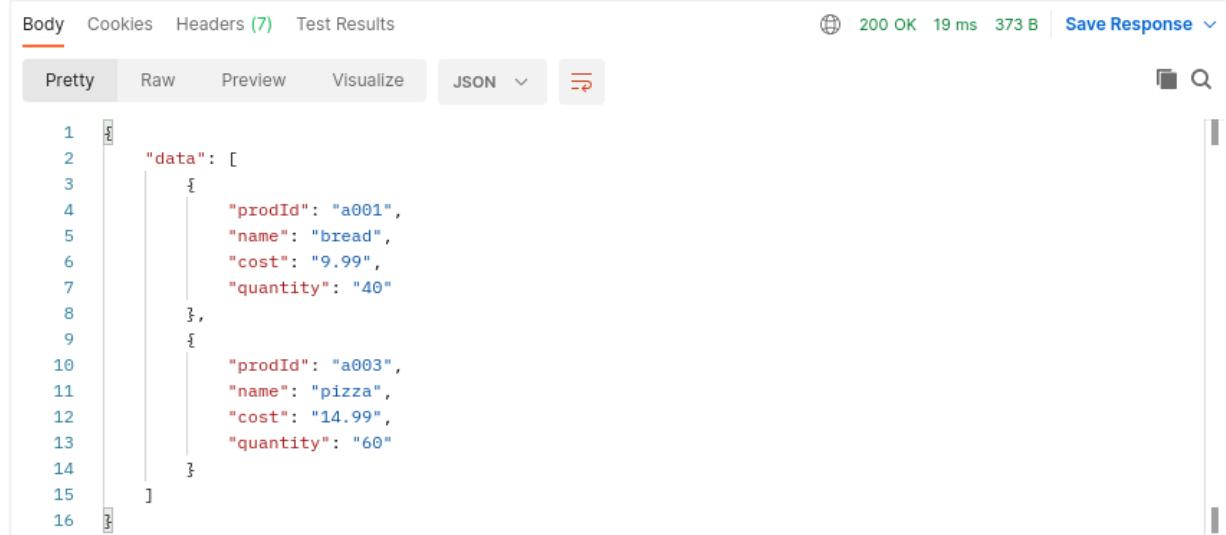
KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> prodId	a002			
Key	Value	Description		

Pretty Raw Preview Visualize JSON

```
1 {
2   "prodId": "a001",
3   "name": "bread",
4   "cost": "9.99",
5   "quantity": "40"
6 },
7 {
8   "prodId": "a003",
9   "name": "pizza",
10  "cost": "14.99",
11  "quantity": "60"
12 }
13 ]
14 }
```

6) '/shop/csv' Method: GET

This route generates a csv file, initiates a download request, and also responds with the generated csv.



The screenshot shows a Postman response for a GET method. The status is `200 OK` with `19 ms` and `373 B`. The response body is a JSON object with a single key `data`, which contains an array of two objects representing the shop items.

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 "data": [
2   {
3     "prodId": "a001",
4     "name": "bread",
5     "cost": "9.99",
6     "quantity": "40"
7   },
8   {
9     "prodId": "a003",
10    "name": "pizza",
11    "cost": "14.99",
12    "quantity": "60"
13  }
14 ]
15 ]
16 ]
```

Errors:

- 1) If prodId is null: (/shop/add, /shop/edit, /shop/delete)

The screenshot shows a Postman test results interface. At the top, it displays "Body", "Cookies", "Headers (5)", and "Test Results". On the right, it shows a status of "400 Bad Request" with "4 ms" and "190 B" response details, and a "Save Response" button. Below this, there are tabs for "Pretty", "Raw", "Preview", "Visualize", "Text", and a red "Copy" icon. The main content area contains the message "1 Bad Request: prodId cannot be empty".

- 2) If item already exists (/shop/add)

The screenshot shows a Postman test results interface. At the top, it displays "Body", "Cookies", "Headers (5)", and "Test Results". On the right, it shows a status of "406 Not Acceptable" with "5 ms" and "213 B" response details, and a "Save Response" button. Below this, there are tabs for "Pretty", "Raw", "Preview", "Visualize", "Text", and a red "Copy" icon. The main content area contains the message "1 Item already exists. Please use 'edit' instead of 'add'".

- 3) If item doesn't exist (/shop/edit, /shop/delete)

The screenshot shows two separate Postman test results interfaces. The top one is for the "/shop/edit" endpoint, displaying a "406 Not Acceptable" error with "5 ms" and "212 B" response details, and a "Save Response" button. It has tabs for "Pretty", "Raw", "Preview", "Visualize", "Text", and a red "Copy" icon. The message "1 Item doesn't exist. Please use 'add' instead of 'edit'" is shown. The bottom one is for the "/shop/delete" endpoint, also showing a "406 Not Acceptable" error with "5 ms" and "190 B" response details, and a "Save Response" button. It has similar tabs and the message "1 Bad Request: Item doesn't exist.".