

---

## FakeStoreAPI

FakeStoreAPI is a free online REST API that you can use whenever you need Pseudo-real data for your e-commerce or shopping website without running any server-side code. It's awesome for teaching purposes, sample codes, tests and etc.

You can visit in detail docs in FakeStoreAPI for more information.

### Why?

When I wanted to design a shopping website prototype and needed fake data, I had to use lorem ipsum data or create a JSON file from the base. I didn't find any online free web service to return semi-real shop data instead of lorem ipsum data. so I decided to create this simple web service with NodeJs(express) and MongoDB as a database.

### Resources

There are 4 main resources need in shopping prototypes:

- Products <https://fakestoreapi.com/products>
- Carts <https://fakestoreapi.com/carts>
- Users <https://fakestoreapi.com/users>
- Login Token <https://fakestoreapi.com/auth/login>

**New! "Rating" (includes rate and count) has been added to each product object!**

### How to

you can fetch data with any kind of methods you know(fetch API, Axios, jquery ajax,...)

#### Get all products

```
1 fetch("https://fakestoreapi.com/products")
2   .then((res) => res.json())
3   .then((json) => console.log(json));
```

---

## Get a single product

```
1 fetch("https://fakestoreapi.com/products/1")
2   .then((res) => res.json())
3   .then((json) => console.log(json));
```

## Add new product

```
1 fetch("https://fakestoreapi.com/products", {
2   method: "POST",
3   body: JSON.stringify({
4     title: "test product",
5     price: 13.5,
6     description: "lorem ipsum set",
7     image: "https://i.pravatar.cc",
8     category: "electronic",
9   }),
10 })
11   .then((res) => res.json())
12   .then((json) => console.log(json));
13
14 /* will return
15 {
16   id:31,
17   title:'...',
18   price:'...',
19   category:'...',
20   description:'...',
21   image:'...'
22 }
23 */
```

Note: Posted data will not really insert into the database and just return a fake id.

## Updating a product

```
1 fetch("https://fakestoreapi.com/products/7", {
2   method: "PUT",
3   body: JSON.stringify({
4     title: "test product",
5     price: 13.5,
6     description: "lorem ipsum set",
7     image: "https://i.pravatar.cc",
8     category: "electronic",
9   }),
10 })
```

---

```
11 .then((res) => res.json())
12 .then((json) => console.log(json));
13
14 /* will return
15 {
16     id:7,
17     title: 'test product',
18     price: 13.5,
19     description: 'lorem ipsum set',
20     image: 'https://i.pravatar.cc',
21     category: 'electronic'
22 }
23 */
```

```
1 fetch("https://fakestoreapi.com/products/8", {
2   method: "PATCH",
3   body: JSON.stringify({
4     title: "test product",
5     price: 13.5,
6     description: "lorem ipsum set",
7     image: "https://i.pravatar.cc",
8     category: "electronic",
9   }),
10 })
11 .then((res) => res.json())
12 .then((json) => console.log(json));
13
14 /* will return
15 {
16     id:8,
17     title: 'test product',
18     price: 13.5,
19     description: 'lorem ipsum set',
20     image: 'https://i.pravatar.cc',
21     category: 'electronic'
22 }
23 */
```

Note: Edited data will not really be updated into the database.

### Deleting a product

```
1 fetch("https://fakestoreapi.com/products/8", {
2   method: "DELETE",
3 });
```

Nothing will delete on the database.

---

## Sort and Limit

You can use query string to limit results or sort by asc|desc

```
1 // Will return all the posts that belong to the first user
2 fetch("https://fakestoreapi.com/products?limit=3&sort=desc")
3   .then((res) => res.json())
4   .then((json) => console.log(json));
```

## All available routes

### Products

```
1 fields:
2 {
3   id:Number,
4   title:String,
5   price:Number,
6   category:String,
7   description:String,
8   image:String
9 }
```

GET:

- /products (get all products)
- /products/1 (get specific product based on id)
- /products?limit=5 (limit return results )
- /products?sort=desc (asc|desc get products in ascending or descending orders (default to asc))
- /products/products/categories (get all categories)
- /products/category/jewelery (get all products in specific category)
- /products/category/jewelery?sort=desc (asc|desc get products in ascending or descending orders (default to asc))

POST:

- /products

-PUT,PATCH

- /products/1

-DELETE

- /products/1

---

## Carts

```
1 fields:
2 {
3     id:Number,
4     userId:Number,
5     date:Date,
6     products:[{productId:Number,quantity:Number}]
7 }
```

### GET:

- /carts (get all carts)
- /carts/1 (get specific cart based on id)
- /carts?startdate=2020-10-03&enddate=2020-12-12 (get carts in date range)
- /carts/user/1 (get a user cart)
- /carts/user/1?startdate=2020-10-03&enddate=2020-12-12 (get user carts in date range)
- /carts?limit=5 (limit return results )
- /carts?sort=desc (asc|desc get carts in ascending or descending orders (default to asc))

### POST:

- /carts

### PUT,PATCH:

- /carts/1

### DELETE:

- /carts/1

## Users

```
1 fields:
2 {
3     id:20,
4     email:String,
5     username:String,
6     password:String,
7     name:{
8         firstname:String,
9         lastname:String
10    },
11     address:{
12         city:String,
```

---

```
13     street:String,  
14     number:Number,  
15     zipcode:String,  
16     geolocation:{  
17         lat:String,  
18         long:String  
19     }  
20 },  
21     phone:String  
22 }
```

#### GET:

- /users (get all users)
- /users/1 (get specific user based on id)
- /users?limit=5 (limit return results )
- /users?sort=desc (asc|desc get users in ascending or descending orders (default to asc))

#### POST:

- /users

#### PUT,PATCH:

- /users/1

#### DELETE:

- /users/1

### Auth

```
1 fields:  
2 {  
3     username:String,  
4     password:String  
5 }
```

#### POST:

- /auth/login

### ToDo

- Add graphql support

- 
- Add pagination
  - Add another language support