Image - Docs

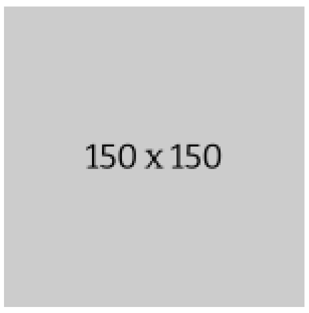
The **image** endpoint provides customizable placeholder images by specifying size in the URL, with options for background color, text color, and display text, ideal for use in websites and wireframes.

The base URL is: dummyjson.com/image

Generate square image

```
// https://dummyjson.com/image/SIZE
fetch('https://dummyjson.com/image/150')
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/png'}
```

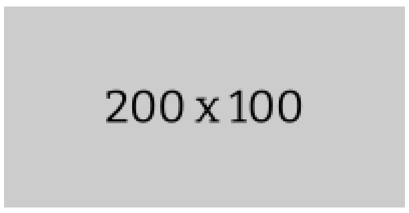
Output:



Generate custom size image

```
// https://dummyjson.com/image/WIDTHxHEIGHT
fetch('https://dummyjson.com/image/200x100')
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/png'}
```

Output:



Generate image with custom text

```
// https://dummyjson.com/image/SIZE/?text=TEXT
fetch('https://dummyjson.com/image/400x200/008080/ffffff?text=Hello+Peter')
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/png'}
```

Output:

Hello Peter

Generate image with custom colors

```
// https://dummyjson.com/image/SIZE/BACKGROUND/COLOR
fetch('https://dummyjson.com/image/400x200/282828')
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/png'}
```

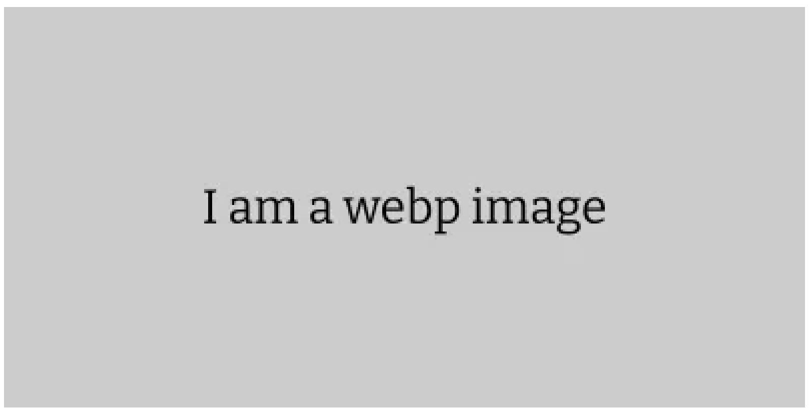
Output:

Generate image with different formats

Supported Formats: png, jpeg, webp

```
// https://dummyjson.com/image/SIZE/BACKGROUND/COLOR
fetch('https://dummyjson.com/image/400x200?type=webp&text=I+am+a+webp+image')
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/webp'}
```

Output:



Generate image with custom font family

Supported Fonts:
bitter, cairo, comfortaa, cookie, dosis, gotham, lobster, marhey, pacifico, poppins, quicksand, qwigley, satisfy,

ubuntu

```
// https://dummyjson.com/image/SIZE/BACKGROUND/COLOR
fetch('https://dummyjson.com/image/400x200/282828?fontFamily=pacifico&text=I+am+a+pa
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/png'}
```

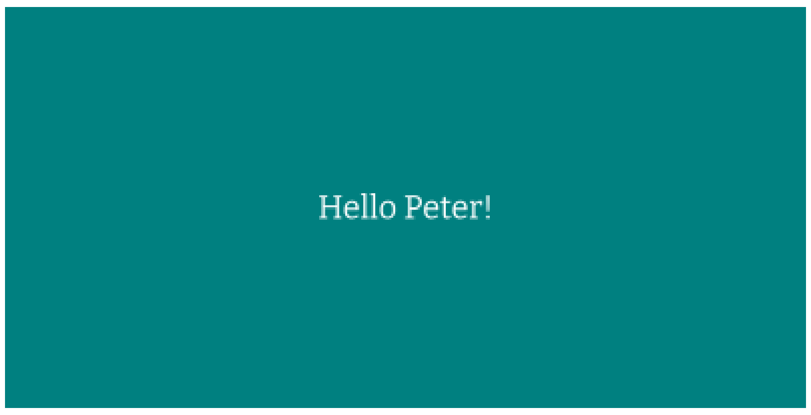
Output:

I am a pacifico font

Generate image with custom font size

```
// https://dummyjson.com/image/SIZE/?text=TEXT&fontSize=FONT_SIZE
fetch('https://dummyjson.com/image/400x200/008080/ffffff?text=Hello+Peter!&fontSize=
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/png'}
```

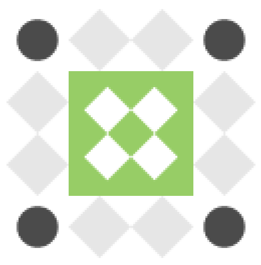
Output:



Generate identicon

```
// https://dummyjson.com/icon/HASH/SIZE/?type=png (or svg)
fetch('https://dummyjson.com/icon/abc123/150') // png is default
.then(response => response.blob()) // Convert response to blob
.then(blob => {
  console.log('Fetched image blob:', blob);
})
// Blob {size: SIZE, type: 'image/png'}
```

Output:



Auth - Docs

The **auth** endpoint provides details about the user authentication and authorization and refresh tokens.

Login user and get token

You can use any user's credentials from dummyjson.com/users

```
fetch('https://dummyjson.com/auth/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({

    username: 'emilys',
    password: 'emilyspass',
    expiresInMins: 30, // optional, defaults to 60
  })
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "id": 1,
  "username": "emilys",
  "email": "emily.johnson@x.dummyjson.com",
  "firstName": "Emily",
  "lastName": "Johnson",
  "gender": "female",
  "image": "https://dummyjson.com/icon/emilys/128",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiOiJtaWNoYWVsdyIsImVtYWlsIjoibWljaGFlbC
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiOiJtaWNoYWVsdyIsImVtYWlsIjoibWl
}
```

Get current auth user

```
/* providing token in bearer */
fetch('https://dummyjson.com/auth/me', {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer /* YOUR_TOKEN_HERE */',
  },
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "id": 1,
  "username": "emilys",
  "email": "emily.johnson@x.dummyjson.com",
  "firstName": "Emily",
  "lastName": "Johnson",
  "gender": "female",
  "image": "https://dummyjson.com/icon/emilys/128"
```

- Custom Response
- Intro
- Dynamic Image
- Auth
  - Login and get token
  - Get auth user
  - Refresh token
- Products
- Carts
- Recipes
- Users
- Posts
- Comments
- Todos
- Quotes
- Mock HTTP

```
  ... // other user fields
}
```

Refresh auth session

Extend the session and create a new access token without username and password

```
fetch('https://dummyjson.com/auth/refresh', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    refreshToken: '/* YOUR_REFRESH_TOKEN_HERE */',
    expiresInMins: 30, // optional, defaults to 60
  })
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiOiJtaWNoYWVsdyIsImVtYWlsIjoibWljaGFlbC
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiOiJtaWNoYWVsdyIsImVtYWlsIjoibWl
}
```

- Custom Response
- Intro
- Dynamic Image
- Auth
  - Login and get token
  - Get auth user
  - Refresh token
- Products
- Carts
- Recipes
- Users
- Posts
- Comments
- Todos
- Quotes
- Mock HTTP

Products - Docs

The **products** endpoint provides a comprehensive dataset of sample product information, including details like names, prices, descriptions, images, and categories, ideal for testing and prototyping e-commerce applications.

Get all products

By default you will get 30 items, use Limit and skip to paginate through all items.

```
fetch('https://dummyjson.com/products')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a single product

```
fetch('https://dummyjson.com/products/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Search products

```
fetch('https://dummyjson.com/products/search?q=phone')
.then(res => res.json())
.then(console.log);
```

Show Output

Limit and skip products

You can pass `limit` and `skip` params to limit and skip the results for pagination, and use `limit=0` to get all items.

You can pass `select` as query params with comma-separated values to select specific data

```
fetch('https://dummyjson.com/products?limit=10&skip=10&select=title,price')
.then(res => res.json())
.then(console.log);
```

Show Output

Sort products

You can pass `sortBy` and `order` params to sort the results, `sortBy` should be field name and `order` should be "asc" or "desc"

```
fetch('https://dummyjson.com/products?sortBy=title&order=asc')
.then(res => res.json())
.then(console.log);
```

Show Output

Get all products categories

```
fetch('https://dummyjson.com/products/categories')
.then(res => res.json())
.then(console.log);
```

Show Output

Get products category list

```
fetch('https://dummyjson.com/products/category-list')
.then(res => res.json())
.then(console.log);
```

Hide Output

```
[
  "beauty",
  "fragrances",
  "furniture",
  "groceries",
  "home-decoration",
  "kitchen-accessories",
  "laptops",
  "mens-shirts",
  "mens-shoes",
  "mens-watches",
  "mobile-accessories",
  "motorcycle",
  "skin-care",
  "smartphones",
  "sports-accessories",
  "sunglasses",
  "tablets",
  "tops",
  "vehicle",
  "womens-bags",
  "womens-dresses",
  "womens-jewellery",
  "womens-shoes",
  "womens-watches"
]
```

Get products by a category

```
fetch('https://dummyjson.com/products/category/smartphones')
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "products": [
    {
      "id": 122,
      "title": "iPhone 6",
      "category": "smartphones",
      ...
    },
    {...}
    // 16 items
  ],
  "total": 16,
  "skip": 0,
  "limit": 16
}
```

Add a new product

Adding a new product will not add it into the server.
It will simulate a POST request and will return the new created product with a new id

```
fetch('https://dummyjson.com/products/add', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    title: 'BMW Pencil',
    /* other product data */
  })
})
```

```
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "id": 195,
  "title": "BMW Pencil",
  /* other product data */
}
```

Update a product

Updating a product will not update it into the server.
It will simulate a PUT/PATCH request and will return updated product with modified data

```
/* updating title of product with id 1 */
fetch('https://dummyjson.com/products/1', {
  method: 'PUT', /* or PATCH */
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    title: 'iPhone Galaxy +1'
  })
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "id": 1,
  "title": "iPhone Galaxy +1", // only title was updated
  /* other product data */
}
```

Delete a product

Deleting a product will not delete it into the server.
It will simulate a DELETE request and will return deleted product with `isDeleted` & `deletedOn` keys

```
fetch('https://dummyjson.com/products/1', {
  method: 'DELETE',
})
.then(res => res.json())
.then(console.log);
```

Show Output

Carts - Docs

The **carts** endpoint offers a dataset of sample shopping cart data, including details like cart items, quantities, prices, and user IDs, useful for testing and prototyping e-commerce functionalities such as cart management and checkout processes.

Get all carts

```
fetch('https://dummyjson.com/carts')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a single cart

```
fetch('https://dummyjson.com/carts/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Get carts by a user

```
// getting carts by user with id 5
fetch('https://dummyjson.com/carts/user/5')
.then(res => res.json())
.then(console.log);
```

Show Output

Add a new cart

Adding a new cart will not add it into the server.
It will simulate a POST request and will return the new created cart with a new id

You can provide a `userId` and array of products as objects, containing `productId` & `quantity`

```
fetch('https://dummyjson.com/carts/add', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    userId: 1,
    products: [
      {
        id: 144,
        quantity: 4,
      },
      {
        id: 98,
        quantity: 1,
      },
    ]
  })
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
  {
    "id": 51,
    "products": [ // products were added by id
      {
        "id": 98,
        "title": "Rolex Submariner Watch",
        "price": 13999.99,
        "quantity": 4,
        "total": 55999.96,
        "discountPercentage": 0.82,
        "discountedPrice": 55541,
        "thumbnail": "https://cdn.dummyjson.com/products/images/mens-watches/Rolex%20Submariner%20Watch/thumbna
      },
      {
        "id": 144,
        "title": "Cricket Helmet",
        "price": 44.99,
        "quantity": 1,
        "total": 44.99,
        "discountPercentage": 10.75,
        "discountedPrice": 40,
        "thumbnail": "https://cdn.dummyjson.com/products/images/sports-accessories/Cricket%20Helmet/thumbnail.p
      }
    ],
    "total": 56044.95, // total was calculated with quantity
    "discountedTotal": 55581,
    "userId": 1, // user id is 1
    "totalProducts": 2,
    "totalQuantity": 5 // total quantity of items
}
```

Update a cart

Updating a cart will not update it into the server.
It will simulate a PUT/PATCH request and will return updated cart with modified data

Pass `"merge: true"` to include old products when updating

You can provide a `userId` and array of products as objects, containing `productId` & `quantity`

```
/* adding products in cart with id 1 */
fetch('https://dummyjson.com/carts/1', {
  method: 'PUT', /* or PATCH */
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    merge: true, // this will include existing products in the cart
    products: [
      {
        id: 1,
        quantity: 1,
      },
    ]
  })
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "id": 1,
  "products": [
    {
      "id": 1,
      "title": "Essence Mascara Lash Princess",
      "price": 9.99,
      "quantity": 1,
      "total": 9.99,
      "discountPercentage": 7.17,
      "discountedPrice": 9,
      "thumbnail": "https://cdn.dummyjson.com/products/images/beauty/Essence%20Mascara%20Lash%20Princess/thumbn
    }
    {...}
    // other old products
  ],
  "total": 103784.84, // total was updated
  "discountedTotal": 89695, // discounted total was updated
  "userId": 33,
  "totalProducts": 5, // total products were updated
  "totalQuantity": 16 // total quantity was updated
}
```

Delete a cart

Deleting a cart will not delete it into the server.
It will simulate a DELETE request and will return deleted cart with `isDeleted` & `deletedOn` keys

```
fetch('https://dummyjson.com/carts/1', {
  method: 'DELETE',
})
.then(res => res.json())
.then(console.log);
```

Show Output

- Custom Response
- Intro
- Dynamic Image
- Auth
- Products
- Carts
  - Get all carts
  - Get a single cart
  - Get carts by user
  - Add a cart
  - Update a cart
  - Delete a cart
- Recipes
- Users
- Posts
- Comments
- Todos
- Quotes
- Mock HTTP

Custom Response

Intro

Dynamic Image

Auth

Products

Carts

- Get all carts
- Get a single cart
- Get carts by user
- Add a cart
- Update a cart
- Delete a cart

Recipes

Users

Posts

Comments

Todos

Quotes

Mock HTTP

Recipes - Docs

The **recipes** endpoint offers a dataset of sample recipe data, including details like recipe names, ingredients, instructions, and images, useful for testing and prototyping cooking and food-related applications.

Get all recipes

By default you will get 30 items, use Limit and skip to paginate through all items.

```
fetch('https://dummyjson.com/recipes')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a single recipe

```
fetch('https://dummyjson.com/recipes/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Search recipes

```
fetch('https://dummyjson.com/recipes/search?q=Margherita')
.then(res => res.json())
.then(console.log);
```

Show Output

Limit and skip recipes

You can pass `limit` and `skip` params to limit and skip the results for pagination, and use `limit=0` to get all items.

You can pass `select` as query params with comma-separated values to select specific data

```
fetch('https://dummyjson.com/recipes?limit=10&skip=10&select=name,image')
.then(res => res.json())
.then(console.log);
```

Show Output

Sort recipes

You can pass `sortBy` and `order` params to sort the results, `sortBy` should be field name and `order` should be "asc" or "desc"

```
fetch('https://dummyjson.com/recipes?sortBy=name&order=asc')
.then(res => res.json())
.then(console.log);
```

Show Output

Get all recipes tags

```
fetch('https://dummyjson.com/recipes/tags')
.then(res => res.json())
.then(console.log);
```

Show Output

Get recipes by a tag

```
fetch('https://dummyjson.com/recipes/tag/Pakistani')
.then(res => res.json())
.then(console.log);
```

Show Output

Get recipes by a meal

```
fetch('https://dummyjson.com/recipes/meal-type/snack')
.then(res => res.json())
.then(console.log);
```

Show Output

Users - Docs

The **users** endpoint provides a versatile dataset of sample user information and related data like carts, posts, and todos, making it ideal for testing and prototyping user management functionalities in web applications.

Get all users

By default you will get 30 items, use Limit and skip to paginate through all items.

```
fetch('https://dummyjson.com/users')
.then(res => res.json())
.then(console.log);
```

Show Output

Login user and get token

You can use any user's credentials from dummyjson.com/users

```
fetch('https://dummyjson.com/user/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({

    username: 'emilys',
    password: 'emilyspass',
    expiresInMins: 30, // optional, defaults to 60
  })
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "id": 1,
  "username": "emilys",
  "email": "emily.johnson@x.dummyjson.com",
  "firstName": "Emily",
  "lastName": "Johnson",
  "gender": "female",
  "image": "https://dummyjson.com/icon/emilys/128",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiOiJtaWNoYWVsdyIsImVtYWlsIjoibWljaGFlbC
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwidXNlcm5hbWUiOiJtaWNoYWVsdyIsImVtYWlsIjoibWl
}
```

Get current authenticated user

```
/* providing token in bearer */
fetch('https://dummyjson.com/user/me', {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer /* YOUR_TOKEN_HERE */',
  },
})
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "id": 1,
  "firstName": "Emily",
  "lastName": "Johnson",
  "maidenName": "Smith",
  "age": 28,
  "gender": "female",
  "email": "emily.johnson@x.dummyjson.com",
  "image": "...",
  /* rest user data */
}
```

Get a single user

```
fetch('https://dummyjson.com/users/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Search users

```
fetch('https://dummyjson.com/users/search?q=John')
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "users": [
    {
      "id": 50,
      "firstName": "Emily",
      "lastName": "Johnson", // name matched the search query
      /* rest user data */
    },
    {...},
    {...}
    // 3 items
  ],
  "total": 3,
  "skip": 0,
  "limit": 3
}
```

Filter users

You can pass key (nested keys with `.`) and value as params to filter users. (key and value are case-sensitive)

`limit`, `skip` and `select` works too.

```
fetch('https://dummyjson.com/users/filter?key=hair.color&value=Brown')
.then(res => res.json())
.then(console.log);
```

Hide Output

```
{
  "users": [
    {
      "firstName": "Emily",
      "hair": {
        "color": "Brown", // filter matched hair color
        type: "Curly"
      },
      "id": 1,
      "lastName": "Johnson"
      /* rest user data */
    },
    {...},
    {...}
    // 23 items
  ],
  "total": 23,
  "skip": 0,
  "limit": 23
}
```

Limit and skip users

You can pass `limit` and `skip` params to limit and skip the results for pagination, and use `limit=0` to get all items.

You can pass `select` as query params with comma-separated values to select specific data

```
fetch('https://dummyjson.com/users?limit=5&skip=10&select=firstName,age')
.then(res => res.json())
.then(console.log);
```

Show Output

Limit and skip users

You can pass `sortBy` and `order` params to sort the results, `sortBy` should be field name and `order` should be "asc" or "desc"

```
fetch('https://dummyjson.com/users?sortBy=firstName&order=asc')
.then(res => res.json())
.then(console.log);
```

Show Output

Get user's carts by user id

```
/* getting carts of user with id 6 */
fetch('https://dummyjson.com/users/6/carts')
.then(res => res.json())
.then(console.log);
```

Show Output

Get user's posts by user id

```
/* getting posts of user with id 5 */
fetch('https://dummyjson.com/users/5/posts')
.then(res => res.json())
.then(console.log);
```

Show Output

Get user's todos by user id

```
/* getting todos of user with id 5 */
fetch('https://dummyjson.com/users/5/todos')
.then(res => res.json())
.then(console.log);
```

Show Output

Add a new user

Adding a new user will not add it into the server.
It will simulate a POST request and will return the new created user with a new id

```
fetch('https://dummyjson.com/users/add', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    firstName: 'Muhammad',
    lastName: 'Ovi',
    age: 250,
    /* other user data */
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Update a user

Updating a user will not update it into the server.
It will simulate a PUT/PATCH request and will return updated user with modified data

```
/* updating lastName of user with id 2 */
fetch('https://dummyjson.com/users/2', {
  method: 'PUT', /* or PATCH */
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    lastName: 'Owais'
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Delete a user

Deleting a user will not delete it into the server.
It will simulate a DELETE request and will return deleted user with `isDeleted` & `deletedOn` keys

```
fetch('https://dummyjson.com/users/1', {
  method: 'DELETE',
})
.then(res => res.json())
.then(console.log);
```

Show Output

Posts - Docs

The **posts** endpoint offers a dataset of sample blog post data, including details like titles, body content, user IDs, and tags, useful for testing and prototyping content management and social media features in web applications.

Get all posts

By default you will get 30 items, use Limit and skip to paginate through all items.

```
fetch('https://dummyjson.com/posts')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a single post

```
fetch('https://dummyjson.com/posts/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Search posts

```
fetch('https://dummyjson.com/posts/search?q=love')
.then(res => res.json())
.then(console.log);
```

Show Output

Limit and skip posts

You can pass `limit` and `skip` params to limit and skip the results for pagination, and use `limit=0` to get all items.

You can pass `select` as query params with comma-separated values to select specific data

```
fetch('https://dummyjson.com/posts?limit=10&skip=10&select=title,reactions,userId')
.then(res => res.json())
.then(console.log);
```

Show Output

Sort posts

You can pass `sortBy` and `order` params to sort the results, `sortBy` should be field name and `order` should be "asc" or "desc"

```
fetch('https://dummyjson.com/posts?sortBy=title&order=asc')
.then(res => res.json())
.then(console.log);
```

Show Output

Get all posts tags

```
fetch('https://dummyjson.com/posts/tags')
.then(res => res.json())
.then(console.log);
```

Show Output

Get posts tag list

```
fetch('https://dummyjson.com/posts/tag-list')
.then(res => res.json())
.then(console.log);
```

Show Output

Get posts by a tag

```
fetch('https://dummyjson.com/posts/tag/life')
.then(res => res.json())
.then(console.log);
```

Show Output

Get all posts by user id

```
/* getting posts by user with id 5 */
fetch('https://dummyjson.com/posts/user/5')
.then(res => res.json())
.then(console.log);
```

Show Output

Get post's comments

```
/* getting posts of comments with id 1 */
fetch('https://dummyjson.com/posts/1/comments')
.then(res => res.json())
.then(console.log);
```

Show Output

Add a new post

Adding a new post will not add it into the server.
It will simulate a POST request and will return the new created post with a new id

```
fetch('https://dummyjson.com/posts/add', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    title: 'I am in love with someone.',
    userId: 5,
    /* other post data */
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Update a post

Updating a post will not update it into the server.
It will simulate a PUT/PATCH request and will return updated post with modified data

```
/* updating title of post with id 1 */
fetch('https://dummyjson.com/posts/1', {
  method: 'PUT', /* or PATCH */
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    title: 'I think I should shift to the moon',
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Delete a post

Deleting a post will not delete it into the server.
It will simulate a DELETE request and will return deleted post with `isDeleted` & `deletedOn` keys

```
fetch('https://dummyjson.com/posts/1', {
  method: 'DELETE',
})
.then(res => res.json())
.then(console.log);
```

Show Output

Comments - Docs

The **comments** endpoint provides a dataset of sample user comments, including details like usernames, post IDs, and comment texts, ideal for testing and prototyping social interactions and feedback features in web applications.

Get all comments

By default you will get 30 items, use Limit and skip to paginate through all items.

```
fetch('https://dummyjson.com/comments')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a single comment

```
fetch('https://dummyjson.com/comments/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Limit and skip comments

You can pass `limit` and `skip` params to limit and skip the results for pagination, and use `limit=0` to get all items.

You can pass `select` as query params with comma-separated values to select specific data

```
fetch('https://dummyjson.com/comments?limit=10&skip=10&select=body,postId')
.then(res => res.json())
.then(console.log);
```

Show Output

Get all comments by post id

```
fetch('https://dummyjson.com/comments/post/6')
.then(res => res.json())
.then(console.log);
```

Show Output

Add a new comment

Adding a new comment will not add it into the server.
It will simulate a POST request and will return the new created comment with a new id

```
fetch('https://dummyjson.com/comments/add', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    body: 'This makes all sense to me!',
    postId: 3,
    userId: 5,
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Update a comment

Updating a comment will not update it into the server.
It will simulate a PUT/PATCH request and will return updated comment with modified data

```
/* updating body of comment with id 1 */
fetch('https://dummyjson.com/comments/1', {
  method: 'PUT', /* or PATCH */
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    body: 'I think I should shift to the moon',
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Delete a comment

Deleting a comment will not delete it into the server.
It will simulate a DELETE request and will return deleted comment with `isDeleted` & `deletedOn` keys

```
fetch('https://dummyjson.com/comments/1', {
  method: 'DELETE',
})
.then(res => res.json())
.then(console.log);
```

Show Output

Todos - Docs

The **todos** endpoint provides a dataset of sample to-do items, including details like task descriptions, statuses, and user IDs, ideal for testing and prototyping task management and productivity applications.

Get all todos

By default you will get 30 items, use Limit and skip to paginate through all items.

```
fetch('https://dummyjson.com/todos')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a single todo

```
fetch('https://dummyjson.com/todos/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a random todo

The random data will change on every call to `/random`.
You can optionally pass a length of max 10 as `/random/10`.

```
fetch('https://dummyjson.com/todos/random')
.then(res => res.json())
.then(console.log);
```

Show Output

Limit and skip todos

You can pass `limit` and `skip` params to limit and skip the results for pagination, and use `limit=0` to get all items.

```
fetch('https://dummyjson.com/todos?limit=3&skip=10')
.then(res => res.json())
.then(console.log);
```

Show Output

Get all todos by user id

```
/* getting todos of user with id 5 */
fetch('https://dummyjson.com/todos/user/5')
.then(res => res.json())
.then(console.log);
```

Show Output

Add a new todo

Adding a new todo will not add it into the server.
It will simulate a POST request and will return the new created todo with a new id

```
fetch('https://dummyjson.com/todos/add', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    todo: 'Use DummyJSON in the project',
    completed: false,
    userId: 5,
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Update a todo

Updating a todo will not update it into the server.
It will simulate a PUT/PATCH request and will return updated todo with modified data

```
/* updating completed status of todo with id 1 */
fetch('https://dummyjson.com/todos/1', {
  method: 'PUT', /* or PATCH */
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    completed: false,
  })
})
.then(res => res.json())
.then(console.log);
```

Show Output

Delete a todo

Deleting a todo will not delete it into the server.
It will simulate a DELETE request and will return deleted todo with `isDeleted` & `deletedOn` keys

```
fetch('https://dummyjson.com/todos/1', {
  method: 'DELETE',
})
.then(res => res.json())
.then(console.log);
```

Show Output

Custom Response

Intro

Dynamic Image

Auth

Products

Carts

Recipes

Users

Posts

Comments

Todos

Get all todos

Get a single todo

Get a random todo

Limit & Skip todos

Add a todo

Update a todo

Delete a todo

Quotes

Mock HTTP

Quotes - Docs

The **quotes** endpoint provides a dataset of sample quotes, including details like the quote text and author information, ideal for testing and prototyping features in applications that require motivational or inspirational content.

Get all quotes

By default you will get 30 items, use Limit and skip to paginate through all items.

```
fetch('https://dummyjson.com/quotes')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a single quote

```
fetch('https://dummyjson.com/quotes/1')
.then(res => res.json())
.then(console.log);
```

Show Output

Get a random quote

The random data will change on every call to `/random`.
You can optionally pass a length of max 10 as `/random/10`.

```
fetch('https://dummyjson.com/quotes/random')
.then(res => res.json())
.then(console.log);
```

Show Output

Limit and skip quotes

You can pass `limit` and `skip` params to limit and skip the results for pagination, and use `limit=0` to get all items.

```
fetch('https://dummyjson.com/quotes?limit=3&skip=10')
.then(res => res.json())
.then(console.log);
```

Show Output