Documentation Spree API

1 SUMMARY

Spree supports RESTful access to the resources that will be mentioned in this guide. The API was built using the Rabl gem.

To communicate with the API developers use JSON data format. Requests are communicated using the HTTP protocol. The API follows certain rules which will be mentioned as we proceed.

1.1 MAKING AN API CALL

To make an API call to your Spree store, you will need an authentication token. These keys can be generated on the user edit screen within the admin interface. Request to the API can be made in the following two ways-

1. Pass the X-Spree-Token header along with request:

```
$ curl --header "X-Spree-Token: YOUR_KEY_HERE" http://example.com/api/products.json
```

2. Pass the token as a parameter in the request.

```
$ curl http://example.com/api/products.json?token=YOUR_KEY_HERE
```

1.2 ERROR MESSAGES

One may encounter the following error messages when communicating with API:

- 404 NOT FOUND (Resource couldn't be found)
- 401 UNAUTHORZED (Not authorized to perform the action)
- 401 UNAUTHORZED (Invalid API key specified)

1.3 RULES

As mentioned earlier, the Spree API endpoints comply with some simple rules:

1. Successful requests for the API are returned with a status of 200.
2. Successful create and update requests result in a status of 201 and 200 respectively.
3. Both create and update requests will return Spree's representation of the data upon success.
4. If any create or update request fails, a status code of 422 will be returned, with a hash containing an "error" key, and an "errors" key. The errors value will contain all ActiveRecord validation errors encountered when saving this record.

5. Delete requests will return status of 200, and no content.
6. Requests that list collections, such as /api/products will return a limited result set back.
7. Requests that list collections can be paginated through by passing a page parameter that is a number greater than 0.
8. If a resource cannot be found, the API will return a status of 404.
9. Unauthorized requests will be met with a 401 response.

1.4 AUTHENTICATION

To make requests to Spree API, one needs the key or API token which can be generated at the Admin panel. However to retrieve the user details and token from the API itself , it is possible using the Spree Api Auth Gem. To sign in a particular user, POST the user's email and password to /api/users/sign_in using JSON as shown in the following sample request:

```
$ curl -v -H "Accept: application/json" -H "Content-type: application/json" -X POST -d '
{"user":{"email":"camelmasa@gmail.com", "password":"camelmasa"}}'
http://localhost:3000/api/users/sign_in
```

Similarly, to create a new user we make a similar POST request using JSON data format containing the user email, password and password_confirmation.

```
$ curl -v -H "Accept: application/json" -H "Content-type: application/json" -X POST -d '
{"user":{"email":"camelmasa@gmail.com", "password":"camelmasa",
"password_confirmation":"camelmasa"}}' http://localhost:3000/api/users/sign_up
```

To achieve above functionality add spree_api_auth gem to your Gemfile as follows:

```
gem 'spree_api_auth', github: 'anoopkanyan/spree_api_auth'
```

Next, bundle your dependencies and run the installation generator:

```
bundle

bundle exec rails g spree_api_extend:install
```

2 PRODUCTS

Product records track unique products within the store. These differ from Variants, which track the unique variations of a product. Together, products and variants describe what is for sale in a particular store.

The API provides us with functionality to work with products and perform actions such as creation, deletion, fetching and updating depending upon your permissions.

2.1 INDEX

Products are listed to an authenticated user. All the products in the store are listed to the admin, but only those products which have an available_on date in the past will be listed to other users. To get the products send a GET request as follows:

```
GET /api/products

$ curl http://example.com/api/products.json?token=YOUR_KEY_HERE
```

Products can also be paginated and iterated through by passing the page parameter, such a GET request looks like:

```
GET /api/products?page=2

$ curl http://example.com/api/products.json?token=YOUR_KEY_HERE&page=2
```

Parameters:

- show_deleted : Boolean – true to show deleted products, else false. By default, it is set to false. This option is only available to the users with an admin role and other users will not be able to list the products which have been deleted.
- page : Specify the page number of the products.
- per_page : The number of products to return per page.

2.2 SEARCH

For searching a particular product in your store make an API request as follows:

```
GET /api/products?q[name_cont]=Spree

$ curl https://example.com/api/products.json?token=YOUR_KEY_HERE&q[name_cont]=Spree
```

The search results are paginated. The search API is provided through the Ransack gem which Spree depends upon.

2.3 SORTING RESULTS

Results can be sorted in a particular order depending upon your needs. This is done by specifying which field to sort when making a request. The following request sorts results according to the id of products:

```
GET /api/products?q[s]=id%20desc

$ curl https://example.com/api/products.json?token=YOUR_KEY_HERE&q[s]=id%20desc
```

The above request sorts the results in descending order of ids associated with products. It is also possible to sort the results using an associated object's filed using request like:

```
GET /api/products?q[s]=shipping_category_name%20asc
```

2.4 SHOW

The details of a particular product can be retrieved in two ways. We can make a request using the product's permalink or alternatively make a request using the id attribute.

```
GET /api/products/a-product
```

In the above request replace a-product with a particular products permalink. Alternatively, send a request using the id of the product which looks like this

```
GET /api/products/1

$ curl https://example.com/api/products/1?token=YOUR_TOKEN_HERE
```

2.5 CREATE

First, to learn about potential attributes (required and non-required) before creating a product send a request like this:

```
GET /api/products/new
```

The above request will respond with product attributes and also distinguish between compulsory attributes and non-compulsory.

After this, you would want to create product. However, this action is only accessible by an admin user. By using the above request we now have the list of necessary parameters with us which are required to create a new product, which is possible by making a POST request which has the necessary parameters. Such a request looks like:

```
POST /api/products?product[name]=Jeans&product[price]=100&product[shipping_category_id]=7

$ curl -X POST -d 'product[name]=Jeans&product[price]=100&product[shipping_category_id]=7'
https://example.com/api/products?token=YOUR_TOKEN_HERE
```

The above request creates a product named "Jeans" with a price of $100 and shipping_category_id set to 7. The parameters used in making the request above are the necessary parameters, which means that at least they must be specified when creating a product.

2.6 UPDATE

The update action is only accessible by an admin user. The update action is performed by making a PUT request, to update a particular product's details. The request is made using the product's permalink or the id. The attributes associated with the product which are to be altered are passed in as parameters. So, the request looks like:

```
PUT /api/products/a-product
```

Suppose we want to change the name of a particular product to Headphones, then such a request would look like:

```
PUT /api/products/a-product?product[name]=Headphones
```

```
$ curl -X PUT -d 'product[name]=Shirt' https://example.com/api/products/1?token=YOUR_KEY_HERE
```

The above request would change the product's name to Shirt whose id is 1.

2.7 DELETE

Like create and update, the delete action is only accessible by an admin user. To delete a product make the following request:

```
DELETE /api/products/a-product

$ curl -X DELETE  https://example.com/api/products/7?token=YOUR_KEY_HERE
```

In the request above we delete the product whose id is 7. The same action can also be achieved using product's permalink instead of using product id.

The request works in a similar fashion to delete request using the admin interface, i.e. it does not actually remove the record from the database. It simply sets deleted_at field to current time on the product, as well as for all the variants of that particular product which was deleted.

3 ORDERS

Orders in your store may be made by the customers through the frontend of your website or may be manually entered by you. The actions that can be performed on orders using the API are mentioned in the text that follows:

3.1 INDEX

This action can only be performed by an admin user. To get a list of orders we make GET request and by passing the page parameter, the Orders can be paginated. Such a request looks like:

```
GET /api/orders
```

There are two parameters which can be passed:

- page : the page number of the Orders which is to be displayed.
- per_page : number of orders to return per page.

```
$ curl GET https://example.com/api/orders?token=YOUR_KEY_HERE
```

The above request lists the Orders for your store. Adding a parameter to the above request would make it look like:

```
$ curl GET https://example.com/api/orders?token=YOUR_KEY_HERE&page=2
```

Such a request would list all the Orders on the second page.

3.2 SEARCH

Searching for a particular Order follows a very similar one that we used for searching a particular product. Here too the searching API is provided by the Ransack Gem. To search for a particular product make a request as follows:

```
GET /api/orders?q[email_cont]=bob

$ curl GET https://example.com/api/orders?token=YOUR_TOKEN_HERE&[email_cont]=spree
```

3.3 SORT

To return the Orders in a specific order make a request as follows:

```
GET /api/orders?q[s]=number%20desc
```

The above request sorts the orders in descending order. To sort in ascending order just replace desc with asc. The request is analogous to the one we used to sort the products of our store.

3.4 SHOW

To get the details of any single product we send a GET request using the order's number to the API as follows:

```
GET /api/orders/R123456789

$ curl GET https://example.com/api/orders/R122150479?token=YOUR_TOKEN_HERE
```

Such a request would list all the details related to the product which has R122150479 as order number. The above request however, will only work out for an admin or a user whose has ordered that product. If a user attempts to access an order which does not belong to him, then an unauthorized error occurs. Users may pass the order's token when sending a request to the API to be authorized to view an order.

On successful response all the details relating to that order id are returned. When an order is in the deliver state, additional shipments information is also returned in the API.

3.5 CREATE

An order can be created through in the following manner:

POST /api/orders

Suppose we want to create a line item matching to a variant whose ID is "2" and the quantity is 5, we make the request as follows:

```
curl -v -H "Accept: application/json" -H "Content-type: application/json" -X POST -d ' { "order":
{ "line_items": [  { "variant_id": 2, "quantity": 5 } ] } }'
https://example.com/api/orders?token=YOUR_KEY_HERE
```

Here we POST JSON to our API which is of the form:

```
{
 "order": {
  "line_items": [
   { "variant_id": 2, "quantity": 5 }
  ]
 }
}
```

When the order creation turns out to be successful, a 201 STATUS is returned else we receive 422 which relates unprocessable entity. The request made above creates an order, but that would not visible in the admin panel unless passes through further stages.

3.6 EMPTY

To empty an order's cart, make a request like:

```
PUT /api/orders/R1234567/empty
```

All line items will be removed from the cart and information related to the order will be cleared. Also the inventory that gets depleted by the order gets restored to original.