Kairion Integration Guide

Manual

Author: Sebastian Hösel

Version: 2.4.2
Issue date: 2021-11-19

Contents

1. Shop prerequisites

Other than integrating the Kairion tag into the website, a few prerequisites are needed on the side of the shop:

1.1 Landing Page

The shop needs to provide an URL that accepts multiple product IDs and display the provided products in the order provided via the url. To achieve this, the URL structure and a separator between the products must be defined.

If products that are out of stock are provided via the url, these products should not be displayed on the landing page.

If no products are shown (e.g. none are provided or none are available) this page still should work and just not show any products.

Example

- https://example.com/landing/ is defined as base landing page url
- "_" is defined as separator
- https://example.com/landing/123_234_345 should display products "123","234","345"

1.2 Product Feed

The shop needs to provide a feed containing all of its products to Kairion. This feed can be provided **either** as a CSV or as a Google product feed.
The data provided in the feed is updated three times a day (every 8 hours).

1.2.1 Google product feed

For information about how to provide a valid Google product feed please refer to https://support.google.com/merchants/answer/7052112?hl=en .

1.2.1 CSV feed

The CSV can be provided either on a server of the shop or be placed & updated on a FTP space provided by Kairion.

1.2.1.1 Required fields

Following fields are required:

Field name	Description	Example
product_id	Internal product ID	1234
global_product_id	e.g. PZN, GTIN	4150002660408
product_name	Name of the product	IBU-ratiopharm 400 akut Schmerztabletten
link	Link to the product	https://www.testshop.de/products/1234
price	Price of the product	3.49 EUR
brand_name	Name of the brand	IBU-ratiopharm
availability	Availability of product. Possible values: ● in_stock ● out_of_stock ● preorder	in_stock

1.2.1.2 Sponsored product fields

For use of Kairion sponsored products every element shown in the native product display must be provided in the feed. The following fields are currently available for this:

Field name	Description	Example
description	Product description	IBU-ratiopharm® 400 mg ist ein entzündungshemmendes, fiebersenkendes und schmerzstillendes Arzneimittel (nicht-steroidales Antiphlogistikum/Analgetikum).
image_link	Link to image	https://www.testshop.de/public/img/1234_small.png
category	Main category of the product	Schmerzen>Schmerzmittel nach Marken>Ratiopharm®
sale_price	Discounted price of the product	3.49 EUR

manufacturer_name	Name of the product manufacturer	ratiopharm GmbH
sku	SKU	00266040
mobile_link	Differing link on mobile devices	https://m.testshop.de/products/1234
display_title	Different title to display in native ads	IBU-ratiopharm 400 akut Schmerztabl... (20 stk)
pzn	PZN	00266040
shipping_price	Shipping cost information	4,00 Euro, ab 25 Euro versandkostenfrei
status_text	Text describing status of the product	Reduziert!
herbal	Information about herbal ingredients	pflanzlich
homeopathic	Information about homeopathic products	homöopathisch
base_price	base packaging price	€ 9.65/100 g
shipping_info	Info text about shipping	<span style="color:008000">Sofort verfügbar</span>
quantity	Quantity of elements in package	20 Stück
form	Form of the product	Tabletten
price_type	Type of the price	UVP

If the product listing item contains some visible (or hidden, i.e. some data required for “Add to cart” button functionality) information that does not logically fit to any field within the currently available fields then please let us know.

1.3 Category Feed

The shop needs to provide a feed containing all categories that exist including their ID, title and hierarchical structure. The tree structure should be mapped by using “;” as separator and depicting the max. depth of the category tree.

Example

The tree

- all categories (1)
    - healthcare (2)
        - cough medicine (4)
        - hey fever (5)
            - pills (6)
            - nasal spray (7)
    - wellness (3)
        - self tanning (8)

should be provided as

```
1;all categories;;;;;;
1;all categories;2;healthcare;;;;
1;all categories;2;healthcare;4;cough medicine;;
1;all categories;2;healthcare;5;hay fever;;
1;all categories;2;healthcare;5;hay fever;6;pills
1;all categories;2;healthcare;5;hay fever;7;nasal spray
1;all categories;3;wellness;;;;
1;all categories;3;wellness;8;self tanning;;
```

A category ID should be either a numerical value or a string containing only small letters without special characters ([a-z]+).

1.4 Slots

The shops decide together with the Partner Management of Kairion on where ads should be displayed on a website (in which HTML element) and what the name of these spaces (slots) should be. The shop then needs to insert the appropriate elements into the HTML in the form

```
<div id="slotname"></div>
```

The content of this element will then be replaced by an ad delivered by Kairion.

1.5 Sponsored products

To use sponsored products (banners looking like native product listings) a slot needs to be placed inside product listings, similar to other banners. Every product field visible in the native listing needs to be provided to Kairion via product feed (see 1.2 Product Feed).

Since sponsored products banners use volatile product data like price and availability the product data available to Kairion has to be kept up to date. The product feed is by default parsed every eight hours, to update product data even more often Kairion offers the possibility to update product data in real time (see 4. Realtime product feed parser).

2. Integration of the Kairion tag into the shop

2.1 Main Structure

The minimal structure of the Kairion tag is as follows:

```
<script type="text/javascript">
  var pagetype = '';
  var products = [ ];
  var tags = [ ];
  var slots = { };
  var consent = true;
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: pagetype,
    products: products,
    tags: tags,
    consent: consent
  });
  window.kias.push({
    cmd: "setPageSlots",
    slots: slots
  });
</script>
<script src="https://js.kctag.net/kias-example.js" type="text/javascript" async></script>
```

with "https://js.kctag.net/kias-example.js" to be replaced with an url provided to the shop by Kairion. This script must be included in every page of the shop and the variables "products", "tags" and "slots" filled.

2.2 Page Calls

The Kairion tag needs to be implemented on every page of the shop and always provide the following information:

- current pagetype
- products currently displayed on the page
- information about given consent
- slots

If applicable, the call must also be implemented on all pages of a pagination.

2.2.1 Page Types

The shop agrees together with Kairion Partner Management on a list of all available page types. As default Kairion defines the following page types:

Page type	Description
front	The frontpage of the shop
category_page	A main category page
category_subpage	A sub category page
search_result	The result page of a search
product_detail	The detail page of one product
landing_page	The landing page, as defined in 1.1
thankyou_page	The page a user sees after a checkout

An unlimited number of page types can be defined additional to these defaults.

Example

```
var pagetype = 'front';
```

2.2.2 Products

All products that are visible on the current page must be provided to Kairion in the following form:

Key	Description
gtin	Internal product ID of the shop, same as in product feed (see 1.2) If available the global product identification number should be used.
categories	Array of categories the product belongs to, at least one. ID must be the same as in category feed (see 1.3)
price	The price of the product, as displayed, in Cents.

Example
For a page displaying three products:
- ID 111 with categories “cat1”,”cat2” and price 15,99€
- ID 222 with category “cat1” and price 9,99€
- ID 333 with category “cat4”, “cat5”, “cat6” and price 4,49€

The products variable must be filled like this:

```
var products = [
  {gtin: "111", categories: ["cat1","cat2"], price: 1599 },
  {gtin: "222", categories: ["cat1"], price: 999 },
  {gtin: "333", categories: ["cat4","cat5","cat6"], price: 449 }
];
```

2.2.3 Slots

The slots that were agreed upon (see 1.4) need to be provided in each page. The shop needs to provide the slot name as well as the identifier of the HTML element it should be displayed in.

Example
For a page containing:
- a slot “high” that should be displayed in HTML element “ad_above_content”
- a slot “sky” that should be displayed in HTML element “ad_sidebar”

the slots variable must be filled like this:

```
var slots = {
  high: "ad_above_content",
  sky: "ad_sidebar"
};
```

2.2.3.1 Testing the Slots

The Kairion tag offers a function to test if the slots provided are valid. To trigger this simply add the GET parameter “kti=est” to any url with a valid Kairion integration, this will cause all correctly configured slots to display a test banner.

Example

https://example.com/someurl/?kti=est

2.2.4 Consent

The shop must provide the Kairion tag with information, if consent was given by the user to the Kairion service. This has to be done via the “consent” field of the “setPageSettings” command (see “2.1 Main Structure”).
The CMP in the shop should be configured, so that, if consent was given to Kairion, “true” (boolean) should be provided to the tag, otherwise “false” (boolean).

Example
If the user has given consent:

```
var consent = true;
```

If the user has denied consent, or did not give explicit consent yet:

```
var consent = false;
```

2.2.4.1 Update consent

It may be necessary to update the given consent - especially in the case of the first page a user opens in the shop. To do this, the shop can simply call the “setPageSettings” again, with the updated consent.

Example

At the initial page load the kairion tag was called before the user even saw the CMP:

```
<script type="text/javascript">
  var pagetype = 'front';
  var products = [ ];
  var tags = [ ];
  var slots = { };
  var consent = false;
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: pagetype,
    products: products,
    tags: tags,
    consent: consent
  });
  window.kias.push({
    cmd: "setPageSlots",
    slots: slots
  });
</script>
<script src="https://js.kctag.net/kias-example.js" type="text/javascript" async></script>
```

After the user gave consent, the shop should update the consent information by pushing the “setPageSettings” command again:

```
<script type="text/javascript">
  var pagetype = 'front';
  var products = [ ];
  var tags = [ ];
  var slots = { };
  var consent = true;
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: pagetype,
    products: products,
    tags: tags,
    consent: consent
  });
</script>
```

2.2.4.2 Optional: separate consent for data extension

If the Kairion data extension is activated for the shop, the shop may ask for two separate consent-entities in the CMP, once for Kairion instore and once for Kairion data extension. The information that consent is given (consent: true) should in that case only be provided, if **both consent entities** were consented to by the user.
Please note that this only applies if two different consents are asked for in the CMP, and then consult the following logic table:

Consent given	**Provide to Kairion tag**
Instore: no / data extension: no	consent: false
Instore: no / data extension: yes	consent: false
Instore: yes / data extension: no	consent: false
Instore: yes / data extension: yes	consent: true

2.2.5 Search results

On search result pages the shop must provide the complete search term the visitor has searched for.

Example

On a search result page where the visitor searched for the term “Cough syrup” the “tags” variable must be filled like this:

```
var tags = ['kw_search:Cough syrup'];
```

2.2.6 Complete example: Mandatory Fields

A complete example for one page call (defining the variables inline) with the example data explained above could look like:

```
<script type="text/javascript">
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: "search_result",
    products: [
      {gtin: "111", categories: ["cat1","cat2"], price: 1599 },
      {gtin: "222", categories: ["cat1"], price: 999 },
      {gtin: "333", categories: ["cat4","cat5","cat6"], price: 449 }
    ],
    tags:  ['kw_search:Cough syrup'],
    consent: true
  });
  window.kias.push({
    cmd: "setPageSlots",
    slots: {
      high: "ad_above_content",
      sky: "ad_sidebar"
    }
  });
</script>
<script src="https://js.kctag.net/kias-example.js" type="text/javascript" async></script>
```

2.2.7 Optional: Additional Tags

Kairion can process additional tags to provide better targetings and exclusions, the name of these tags is freely definable. As a default Kairion defines the following tags:

Tag	Description
kw_brand	A page of the shop dedicated to one brand
kw_topic	A page of the shop about a certain topic

These tags can be provided to Kairion similar to the search term.

Example
On a page dedicated to the brand “Aspirin” the tags variable should be filled like:

```
var tags = ['kw_brand:Aspirin'];
```

2.2.8 Optional: Ajax

When the shops implements the Kairion tag on pages that are not completely reloaded (e.g. paginations implemented with ajax-calls) the Kairion tag has to be executed every time the contents of the page are updated. If the shop uses a mechanism like this the option “ajax: true” must be provided in the “setPageSettings” section.

Example
The following is a part of the complete example used in 2.2.5, extended by the ajax option.

```
...
window.kias.push({
  cmd: "setPageSettings",
  pageType: "search_result",
  products: [
    {gtin: "111", categories: ["cat1","cat2"], price: 1599 },
    {gtin: "222", categories: ["cat1"], price: 999 },
    {gtin: "333", categories: ["cat4","cat5","cat6"], price: 449 }
  ],
  tags:  ['kw_search:Cough syrup'],
  consent: true,
  ajax: true
});
...
```

2.2.9 Optional: Callback

Shops can provide a custom callback function to the Kairion tag that is called as soon as the Kairion adserver answers. This callback accepts one argument which contains the response from the adserver. It can be provided in the key “callback” in the “setPageSlots” section.

Example

The following is a part of the complete example used in 2.2.5, extended by the callback

```
...
window.kias.push({
  cmd: "setPageSlots",
  slots: {
    high: "ad_above_content",
    sky: "ad_sidebar"
  },
  callback: function(ads) {
    console.log(ads);
  }
});
...
```

2.2.10 Optional: Integration on blog pages

The Kairion tag can also be integrated on normal content pages inside the shop, like e.g. a blog. To do this, the topic of the content page has to be provided like a search term on a search result page. The provided term should make clear which topic the search page covers, therefore the title of the page often can be used.
The pagetype that is used for these pages should be decided upon with the Kairion partner management.
If no products are shown the products array can be left empty.

Example

```
var tags = ['kw_search:Die besten Tipps gegen Halsschmerzen'];
```

A full example for a blog page integration could look like:

```
<script type="text/javascript">
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: "blog_page",
    products: [],
    tags:  ['kw_search:Die besten Tipps gegen Halsschmerzen''],
    consent: true
  });
  window.kias.push({
    cmd: "setPageSlots",
    slots: {
      high: "ad_above_content",
      sky: "ad_sidebar"
    }
  });
</script>
<script src="https://js.kctag.net/kias-example.js" type="text/javascript" async></script>
```

2.2.11 Optional: Custom landing pages

Custom landing pages are special banners with a fixed width. These are available beneath a specified url prefix that has to be provided by the shop.

Example

- https://example.com/clp/ is defined as base custom landing page url
- https://example.com/clp/someurl/ is available and contains the discussed slot
- https://example.com/clp/someotherurl/ (as well as everything else starting with "https://example.com/clp/") is available and contains the discussed slot

A full example for a custom landing page integration, where the HTML element in which the banner would have the ID "clp_element", could look like:

```
<div id="clp_element"></div> <!-- div that will be filled -->
<script type="text/javascript">
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: "custom_landing_page",
    products: [],
    consent: true
  });
  window.kias.push({
    cmd: "setPageSlots",
    slots: {
      clp: "clp_element"
    }
  });
</script>
<script src="https://js.kctag.net/kias-example.js" type="text/javascript" async></script>
```

2.3 Checkout Call

Every checkout that happens in the shop must be send to Kairion. In this call the shop must provide all products that were bought, as well as the amount of each product bought. This should be accomplished by adding the following call **once** to the first page the visitor sees after his checkout:

```
var checkout_products = [ ];
...
window.kias.push({
  cmd: "checkout",
  products: checkout_products
});
...
```

2.3.1 Checkout Products

All products that were bought must be provided to Kairion in the following form:

Key	Description
gtin	Internal product ID of the shop, same as in product feed (see 1.2) If available the global product identification number should be used.
categories	Array of categories the product belongs to, at least one. ID must be the same as in category feed (see 1.3)
price	The price of the product, as displayed, in Cents.
amount	Amount of products that were bought

Example
For a checkout of three products:

- ID 111 with categories “cat1”,”cat2” and price 15,99€, bought one time
- ID 222 with category “cat1” and price 9,99€, bought 5 times
- ID 333 with category “cat4”, “cat5”, “cat6” and price 4,49€, bought 2 times

The checkout_products variable must be filled like this:

```
var checkout_products = [
  { gtin: "111", categories: ["cat1","cat2"], price: 1599, amount: 1 },
  { gtin: "222", categories: ["cat1"], price: 999, amount: 5 },
  { gtin: "333", categories: ["cat4","cat5","cat6"], price: 449, amount: 2 }
];
```

2.3.2 Complete example: Checkout

A complete example for the call on the page the visitor first sees after his checkout, with the example data explained above could look like:

```
<script type="text/javascript">
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: 'thankyou_page',
    products: [],
    tags:  [],
    consent: true
  });
  window.kias.push({
    cmd: "checkout",
    products:  [
      { gtin: "111", categories: ["cat1","cat2"], price: 1599, amount: 1 },
      { gtin: "222", categories: ["cat1"], price: 999, amount: 5 },
      { gtin: "333", categories: ["cat4","cat5","cat6"], price: 449, amount: 2 }
    ]
  });
</script>
<script src="https://js.kctag.net/kias-example.js" type="text/javascript" async></script>
```

2.4 Cart Call

Kairion needs to be notified if visitors currently have items in their cart. This should be accomplished by adding the following call **to every page call** as long as users have items in their cart:

```
var cart_products = [ ];
...
window.kias.push({
  cmd: "setCartProducts",
  products: cart_products
});
...
```

2.4.1 Cart Products

All products that are in the cart must be provided to Kairion in the following form (same as checkout products structure):

Key	Description
gtin	Internal product ID of the shop, same as in product feed (see 1.2) If available the global product identification number should be used.
categories	Array of categories the product belongs to, at least one. ID must be the same as in category feed (see 1.3)
price	The price of the product, as displayed, in Cents.
amount	Amount of products that were bought

Example

For a cart containing three products:

- ID 111 with categories “cat1”,”cat2” and price 15,99€, bought one time
- ID 222 with category “cat1” and price 9,99€, bought 5 times
- ID 333 with category “cat4”, “cat5”, “cat6” and price 4,49€, bought 2 times

The cart_products variable must be filled like this:

```
var cart_products = [
  { gtin: "111", categories: ["cat1","cat2"], price: 1599, amount: 1 },
  { gtin: "222", categories: ["cat1"], price: 999, amount: 5 },
  { gtin: "333", categories: ["cat4","cat5","cat6"], price: 449, amount: 2 }
];
```

2.4.2 Complete example: Cart Call

A complete example for the call on every page the visitor visits while he has products in his cart, in combination with the example data explained in 2.2.5 could look like:

```
<script type="text/javascript">
  window.kias = window.kias || [];
  window.kias.push({
    cmd: "setPageSettings",
    pageType: search_result,
    products: [
      { gtin: "111", categories: ["cat1","cat2"], price: 1599 },
      { gtin: "222", categories: ["cat1"], price: 999 },
      { gtin: "333", categories: ["cat4","cat5","cat6"], price: 449 }
    ],
    tags:  ['kw_search:Cough syrup'],
    consent: true
  });
  window.kias.push({
    cmd: "setPageSlots",
    slots: {
      high: "ad_above_content",
      sky: "ad_sidebar"
    }
  });
  window.kias.push({
    cmd: "setCartProducts",
    products:  [
      { gtin: "111", categories: ["cat1","cat2"], price: 1599, amount: 1 },
      { gtin: "222", categories: ["cat1"], price: 999, amount: 5 },
      { gtin: "333", categories: ["cat4","cat5","cat6"], price: 449, amount: 2 }
    ]
  });
</script>
<script src="https://js.kctag.net/kias-example.js" type="text/javascript" async></script>
```

3. Medi-Deal

Medi-Deal allows manufacturers to create campaigns for discounted products in shops. To implement this, the shops must provide a product detail page with a discounted price, and a feed containing all discounted sales that were generated.
For the sake of a transparent shopping experience for the user, it should be made clear on the product detail page that the discount granted is the Medi-Deal discount. (Asterisk text, product description, buybox, discount disruptor, etc.)."

3.1 Discounted product detail page

The shops need to provide Kairion for each medi-deal campaign with an url for a product detail page with a discounted price. This landing page should:

- be in the same layout and provide same functionality as regular product detail pages
- provide the product with a discounted agreed upon price
- show the amount of discount applied to the regular price

3.2 Discounted sales feed

Kairion has to be informed about every discounted sale via a csv feed. The feed needs to contain **one line per discounted product for each checkout**.
The feed must be in the following format:

Time
- time of checkout, format YYYY-MM-DD HH:MM:SS

Product-ID
- ID (PZN) of discounted product, eight digits

Discounted sales
- amount of discounted products sold

Discounted revenue
- Sum of revenue of that product in that checkout (format cents)

The feed must **always** contain **all** discounted sales (no lines are ever removed from the feed).

Example

Given:

- a discounted product p1 with pzn 01111111, costing normally 10,00 Euro, via discounted link 9,00 Euro
- a discounted product p2 with pzn 02222222, costing normally 5,00 Euro, via discounted link 3,99 Euro
- a non discounted product p3 with pzn 03333333, costing 6,99 Euro
- a checkout c1 at 05.01.2021 13:00:00 containing 1xp1
- a checkout c2 at 06.01.2021 13:30:00 containing 2xp1, 1xp3
- a checkout c3 at 10.01.2021 13:34:30 containing 1xp1, 4xp2, 2xp3

Should result in the following CSV:

```
Time;Product-ID;Discounted sales;Discounted revenue
05.01.2021 13:00:00;01111111;1;900;
06.01.2021 13:30:00;01111111;2;1800;
10.01.2021 13:34:30;01111111;1;900;
10.01.2021 13:34:30;02222222;4;15.96
```

4. Realtime product feed parser

4.1 General

The automatic product feed parser in the Kairion System is run three times a day. But for Sponsored products and banners including prices, sometimes information must be kept up to date even more often. The Realtime product feed parser feature was created to give the possibility to shop users to update information about products in the System more often.. To run all queries the ID of a shop will be needed, this information can be found in the URL link of a shop form. For available fields see 1.2.1 CSV feed.

4.2 API keys

To get access to run the product feed parser you need an API-key. For this you should either take one of your previously generated keys, or generate a new one in the Kairion management interface.

4.3 Products update

There are two possible ways of updating information about products in the system: triggering the parsing or updating one product directly via query.

4.3.1 Parsing trigger

In case multiple changes are needed, shop users can change all this information in their product feed, and after all changes are saved, users can trigger a product feed parsing on Kairion side. To do this use this query with your previously generated API-key and shop-ID:

```
curl --location --request POST 'https://products.kairion.de/parse?access_token=API-key' \
--header 'Content-Type: application/json' \
--data-raw '["SHOP_ID"]'
```

As a response you may get the following:

1. Parsing feed task is in queue and products will be updated soon (Status: 202)
2. Parsing for these shop(s) has finished recently and you need to wait XX minutes before you can trigger another parsing (Status: 429)

Parsing can be only run if there are no currently running parsing jobs of your shop and the last update was made more than two hours ago.

4.3.2 Single product update

In case if only one product needs to be changed you can use the following queries below. There is no need to run a product feed parsing after sending these requests.

4.3.2.1 Create a new product

```
curl --location --request POST
'https://products.kairion.de/products/SHOP_ID?access_token=API-key' \
--header 'Content-Type: application/json' \
--data-raw '{
    "product_id": "000000000",
    "global_product_id": "00",
    "product_name": "test product",
    "description": "test description",
    "link": "https://test.com",
    "availability": "in_stock",
    "price": 650,
    "manufacturer_name": "test manufacturer",
    "brand_name": "test brand"
}'
```

As a response you may get one of the following:

1. New product was created, you'll see that the "status": "active" was set by default if there wasn't any value (Status: 201)
2. Product with such product_id was created already, so old product was updated (Status: 200)
3. Product is missing required fields or fields have invalid values (Status: 400)

4.3.2.2 Update a product

```
curl --location --request POST
'https://products.kairion.de/products/SHOP_ID?access_token=API-key' \
--header 'Content-Type: application/json' \
--data-raw '{
    "product_id": "000000000",
    "status": "active",
    "global_product_id": "00",
    "product_name": "test product",
    "link": "https://test.com",
    "availability": "preorder",
    "price": 800,
    "manufacturer_name": "test manufacturer",
    "brand_name": "test brand"
}'
```

This request is the same as the one before (creating a product) so not existing fields will be removed.

As a response you may get one of the following:

1. New product was created, because product with product_id wasn’t found (Status: 201)
2. Product was updated (Status: 200)
3. Product is missing required fields or fields have invalid values (Status: 400)

4.3.2.3 Update only selected fields of a product (e.g. update price)

```
curl --location --request PATCH
'https://products.kairion.de/products/SHOP_ID/PRODUCT_ID?access_token=API_key \
--header 'Content-Type: application/json' \
--data-raw '{
    "description": "some other description",
    "price": 968
}'
```

As a response you may get one of the following:

1. Product was updated (Status: 200)
2. Product fields have invalid values. (Status: 400)
3. Product not found (Status: 404)

4.3.2.4 Delete a product

```
curl --location --request DELETE
'https://products.kairion.de/products/SHOP_ID/PRODUCT_ID?access_token=API-key'
```

As a response you may get one of the following:

1. Product status set to “inactive“, which means it was marked as deleted (Status: 204)
2. Product not found (Status: 404)

4.4 Open API description

To see the “human readable” api description paste the following code to https://editor.swagger.io

```
openapi: 3.0.3
info:
  title: Product feed
  description: >-
    Public API for shops
  version: 1.0.0
servers:
  - url: 'https://products.kairion.de/'
paths:
  '/parse':
    post:
      tags:
      - parsing
      summary: Add parsing to queue
      description: Add your product feed to parsing queue
      requestBody:
        content:
          'application/json':
            schema:
              $ref: '#/components/schemas/ParseRequest'
      responses:
        202:
          description: New parsing task was added to queue
          content: {}
        400:
          description: Invalid request
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ValidationError'
        401:
          description: No API access token in request or invalid
          content: {}
        403:
          description: No rights for shop
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ValidationError'
        429:
          description: Shop product feed has been parsed recently and you need to wait
          headers:
            Retry-After:
              schema:
                type: integer
              description: Timeout in seconds when new requset will be available
          content: {}
      security:
        - access_token: []
  '/products/{shopID}':
    post:
      tags:
      - product
      summary: Create or update product
      parameters:
      - name: shopID
        description: Your shop id in Kairion system
        in: path
        required: true
        schema:
          type: string
          pattern: '/^[a-f\d]{24}$/'
        example: '507f191e810c19729de860ea'
      requestBody:
        content:
          'application/json':
            schema:
              $ref: '#/components/schemas/ProductWithID'
      responses:
        201:
          description: New product was created
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ProductWithID'
        200:
          description: Product was updated
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ProductWithID'
        400:
          description: Invalid request
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ValidationError'
        401:
          description: No API access token in request or invalid
          content: {}
        403:
          description: No rights for shop or
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ValidationError'
      security:
        - access_token: []
  '/products/{shopID}/{productID}':
    parameters:
      - name: shopID
        description: Your shop id in Kairion system
        in: path
        required: true
        schema:
          type: string
          pattern: '/^[a-f\d]{24}$/'
        example: '507f191e810c19729de860ea'
      - name: productID
        in: path
        required: true
        schema:
          type: string
    patch:
      tags:
      - product
      summary: update one or more product fields
      requestBody:
        content:
          'application/json':
            schema:
              $ref: '#/components/schemas/ProductPatchRequest'
      responses:
        200:
          description: Product was updated
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ProductWithID'
        400:
          description: Invalid request
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ValidationError'
        401:
          description: No API access token in request or invalid
          content: {}
        403:
          description: No rights for shop
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ValidationError'
        404:
          description: Product or shop not found
          content: {}
      security:
        - access_token: []
    delete:
      tags:
      - product
      summary: Set product as inactive
      responses:
        204:
          description: Product was deleted
          content: {}
        401:
          description: No API access token in request or invalid
          content: {}
        403:
          description: No rights for shop
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ValidationError'
        404:
          description: Product or shop not found
          content: {}
      security:
        - access_token: []
components:
  securitySchemes:
    access_token:
      description: unique API access token for each shop
      type: apiKey
      name: access_token
      in: query
  schemas:
    ProductWithID:
      type: object
      additionalProperties: false
      properties:
        product_id:
          description: unique product id for your shop
          type: string
        product_name:
          type: string
        link:
          type: string
          format: uri
        availability:
          type: string
          enum:
          - in_stock
          - out_of_stock
          - preorder
          default: in_stock
        price:
          type: integer
          format: int32
          description: in cents
        brand_name:
          type: string

        global_product_id:
          type: string
        manufacture_name:
          type: string
        description:
          type: string
        image_link:
          type: string
          format: uri
        mobile_link:
          type: string
          format: uri
        category:
          type: string
        sku:
          type: string
      required:
      - product_id
      - product_name
      - link
      - price
      - brand_name
    ProductPatchRequest:
      type: object
      additionalProperties: false
      properties:
        product_name:
          type: string
        link:
          type: string
          format: uri
        availability:
          type: string
          enum:
          - in_stock
          - out_of_stock
          - preorder
        price:
          type: integer
          format: int32
          description: in cents
        brand_name:
          type: string
        global_product_id:
          type: string
        manufacture_name:
          type: string
        description:
          type: string
        image_link:
          type: string
          format: uri
        mobile_link:
          type: string
          format: uri
        category:
          type: string
        sku:
          type: string
```

```
  ParseRequest:
    type: array
    items:
      type: string
      pattern: '/^[a-f\d]{24}$/'
      example: '507f191e810c19729de860ea'
      description: Your shop id in Kairion system
    minItems: 1
  ValidationError:
    type: object
    properties:
      status:
        type: number
      error:
        type: string
```