
# HOW TO IMPLEMENT THE QUARTICON SYSTEM USING API

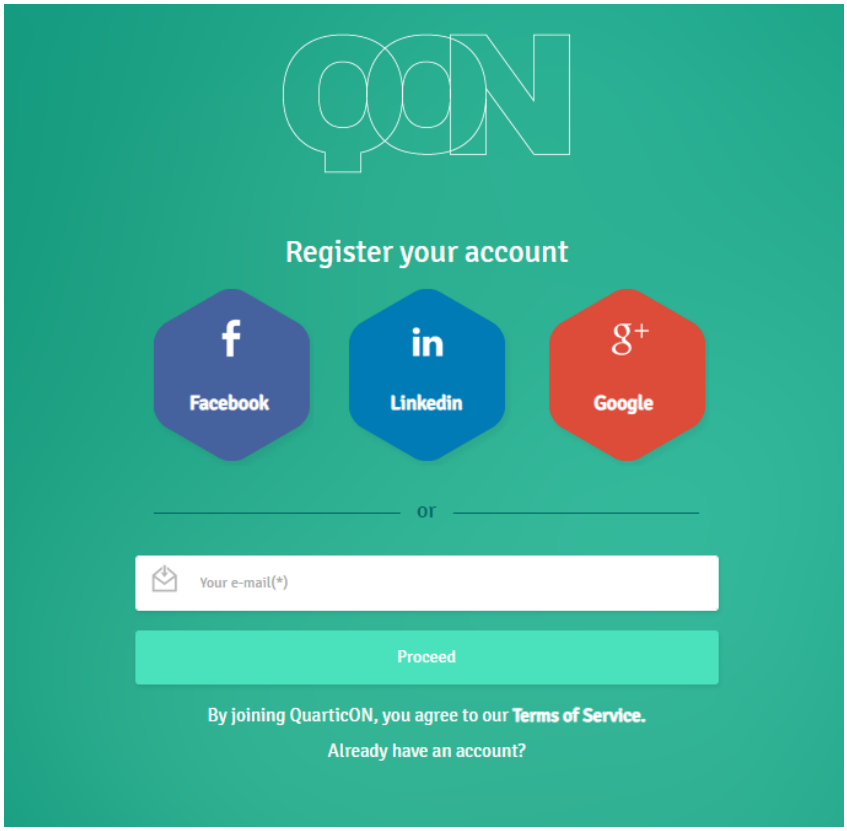
**IMPLEMENTATION INSTRUCTIONS BY API**

# Table of contents

## Index

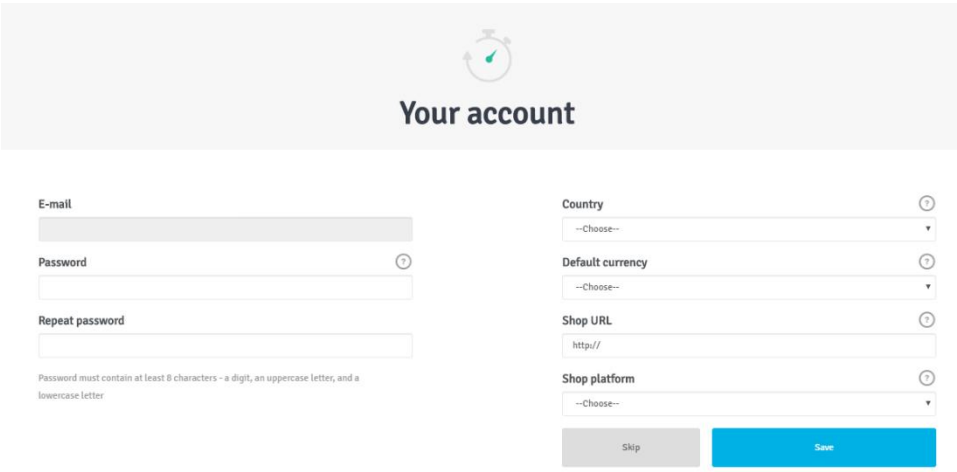
## Creating an account in the Panel

1. Create an account in the Customer Panel
-> https://cp.quarticon.com/register/stepone

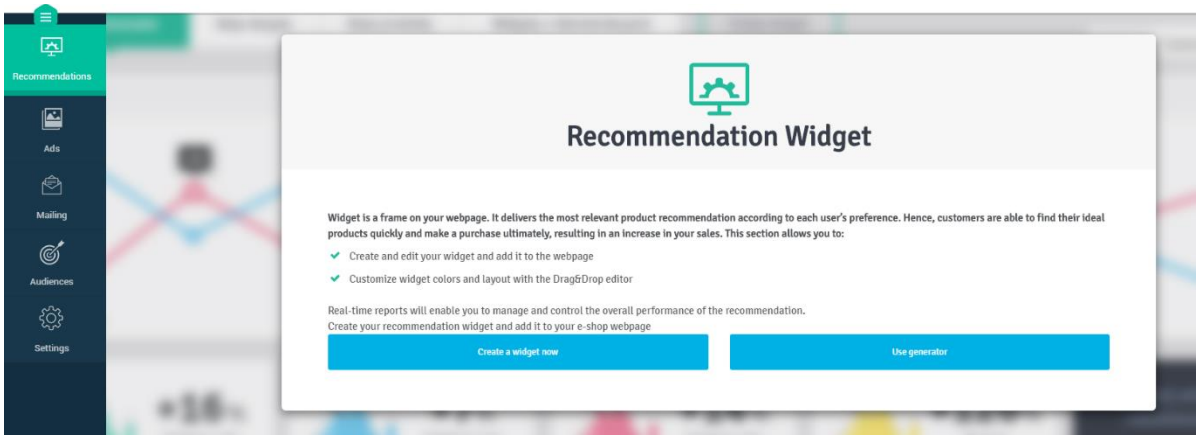



2. After logging in, you can use the tutorial which will guide you through the basic panel functions in a few simple steps.

3. After going through the tutorial (or skipping this step) you need to fill in your account details.



4. A recommendation frame creation window will appear. If you do not have a clear vision of how recommendation frames should look like on your site, we recommend that you create recommendation frames based on best practices.




5. After creating an account in the panel, contact your **Customer Account Manager** who will give you access keys to the API.

## Authorisation for API

Before you start working with the API, you must clearly confirm (authorise) your identity. The method of its confirmation and acquired rights depend on where a given query will be sent.

### In order to send catalogue data

To properly authorise to submit catalogue data, you must include the x-api-key header containing the API user key for each query.

**ATTENTION:**

This key allows you to freely manipulate the product catalogue, so it should remain **confidential.**

### To send events (GET and POST type)

In the case when we have to execute a given event, we should authenticate ourselves by adding the customer parameter to the query with a value specifying the customer symbol.

This type of authorisation should be used e.g. in order to send an event of clicking on a product or displaying a page.

# Synchronisation of the product catalogue

## Where should we do the integration?

Before you start to program the QuarticOn recommendation engine catalogue synchroniser, you should locate the places where you want to store:

- a new product **has been added** to the shop
- the product **has been updated** in the shop
- the product may have become **unavailable**
- the product has been **removed**

These are the places where we want our synchroniser to start up and send updated product information.

## What should a synchronisation query look like?

The synchronising query should remain:

- sent via POST to the following address https://restapi.quartic.pl/store/data/product
- have an authentication header (to send catalogue data)
- have a headline Content-Type: application/json
- have a correct structure (Request Body):

```
{
  "id": " productIdentifierIntheShop",
  "title": "producttitle",
  "image": "http://images.com/ProduktImage.png",
  "description": "test product description",
  "url": "http://sklep.dev/produkt/111-test-product",
  "price": "10.99",
  "priceOld": "12.99",
  "custom1": " any content that will be returned with the product in the recommendation",
  "custom2": " it will then be possible to draw it out in the code receiving the answer",
  "custom3": "and used to generate a frame ",
  "status": true,
```

```
"categories": [
  {
"id": "1",
"name": "Test products"
  }
],
"catalogSymbol": ""
}
```

where:

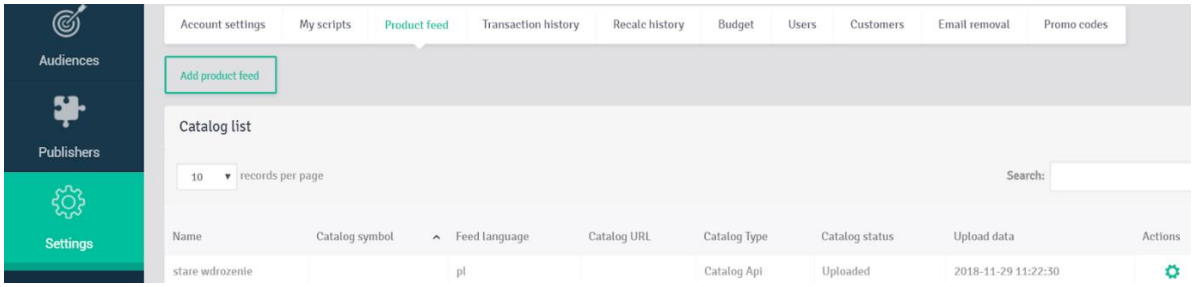
- **id** – product identifier composed of large and/or small letters and/or numbers linked by a hyphen or underline mark, e.g. ZaB-99_PL. The ID cannot contain spaces;
- **title** – text displayed as the name / title of the product;
- **image** – a direct link to the image (already properly scaled), which must comply with the scheme Uniform Resource Locator;
- **description** – product description (up to 20 000 characters);
- **url** – a direct link to the product, which must comply with the scheme Uniform Resource Locator;
- **price** – current product price (float type);
- **priceOld** – price of the product before discount;
- **custom** (from 1 to 3) – field for any value (up to 20 000 characters), which will be sent as generated frames. Most commonly used for marking:
    - promotions,
    - novelties,
    - last pieces of products;
- **status** – informs us if the product is currently available. Boolean type value:
    - true – yes,
    - false – no;
- **categories** – contains categories to which the product should be added (must be in line with the categories of the store);
- **catalogueSymbol –** in most cases we leave them as an empty string:

```
"catalogueSymbol": ""
```

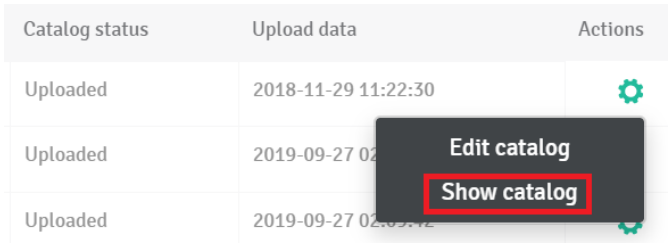
It is used only in shops with multiple catalogues for multiple language versions - thanks to this parameter we are able to recommend products for different catalogues.

## Synchronisation validation

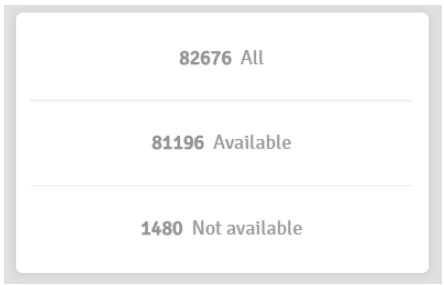
The recommended method of checking whether the first synchronisation has ended correctly is to go to *Account Settings*, and then select *Product Feed*:



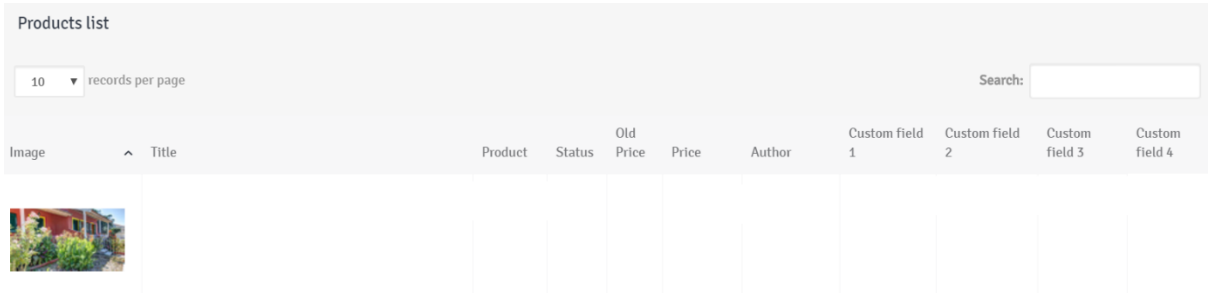

Then click on the settings field next to the catalogue named *Qonizer* (with blank catalogue symbol) and select *Show Catalogue*:




On the right side there will be a section showing the number of products in the catalogue:




and below the statistics we will be able to see examples of products from the catalogue:




# Tracking - monitoring customer activity

## What is tracking for?

In order to correctly personalise recommendations for each shop user, the recommendation engine needs behavioural data. For this purpose, tracking is carried out, i.e. monitoring the activity of customers. The QuarticOn system collects data on the behaviour of every visitor to the website - it tracks their movements (what products they watch, what categories they watch, where the cursor is

directed on the website), what links or buttons they use and the time spent on the website and in its individual places.

We also collect data on what stage of the purchase path the customers are at - whether they are looking for a product, whether they have already added a product to their shopping cart, whether they have already made a payment or completed the purchase. The system also gathers data on whether a visitor leaves the website. Data concerning each user is collected, stored and used for analysis from the moment of first customer identification. There is no time limit for storing data of the visitor's behaviour on the website.

Thanks to these input data, the engine is able to estimate the probability of clicking / purchasing recommended products to the customer, and on this basis select the best products to display in the recommendation frame (using all previous interactions as "learning" material).

## What products did the user see? - eventProductView

- sent via POST or GET
- on each product displayed by the user
- the query should have the correct structure (different depending on the method we send it)

POST:

- remember to change the value of the field customer on the correct customer symbol
- send to following address: https://restapi.quartic.pl/store/track/event

```
{
"productId": "productId",
"eventType": "eventProductView",
"userId": "string",
"deviceId": "string",
"timestamp": "11",
"referrer": "string",
"cookie": "string",
"customer": "symbol"
}
```

where:

- the product ID should be the same as the one given in the synchronisation process;
- userId should be in line with the user ID in the shop. If the user is not logged in, the field should remain empty;

- it should be given either *cookie* or *deviceId*, where:
    - **cookie** is a changeable user ID (which can be deleted/regenerated);
    - **deviceId** is unchangeable (phone's IMEI number, TV serial number, etc.);
- referrer might be:
    - the correct URL of the page that caused the redirection to the product;
    - the name Activity, which was triggered e.g. on an Android phone before switching to the product;
- timestamp - time of occurrence of the event;

## What products did the user buy? – eventTransaction

- sent via POST or GET
- every time a user has bought a product (regardless of whether he bought it through the engine or not)
- the query should have a correct structure (different for the method we send it)

POST:
- remember to change the value of the customer field to the correct customer symbol

send to the address: https://restapi.quartic.pl/store/track/event

```
{
"transactionId": "string",
"basket": [
{
"productId": "string",
"price": "1.00",
"quantity": "1"
}
]
,
"eventType": "eventTransaction",
"userId": "string",
"deviceId": "string",
"timestamp": "21",
"referrer": "string",
"cookie": "string",
"customer": "symbol"
}
```

where:

- transactionId is a unique string, different for each transaction;
- basket is a table of objects describing products added to the basket, where:
    - each product is another object;
    - contains the product identifier matching the identifier in the catalogue;
    - price in float format (for example 10.00);
    - the number of products bought;
- eventType set as eventTransaction;
- userId should be compatible with the user ID in the shop. If the user is not logged in, the field should be left blank;
- It should be given either a *cookie* or a *deviceId* where:
    - **cookie** is a changeable user ID (which can be deleted/regenerated);
    - **deviceId** is unchangeable (phone's IMEI number, TV serial number etc.);
- timestamp - time of occurrence of the event in the Unix timestamp format
- referrer might be:
    - the correct URL of the page that caused the redirection to the product;
    - the name Activity, which was triggered e.g. on an Android phone before switching to the product;

### Historical transactions

In order to increase the effectiveness of the QuarticOn recommendation engine in the early period of use, the shop has the possibility to upload transaction history to the engine. Thanks to this, the engine learns the shop as if it has been operating in it for a long time.

In case of problems with downloading transaction history, e.g. due to incompatible data type, we encourage you to contact our technical support department: helpdesk@quarticon.com

## Download recommendations

### How does the engine generate recommendations?

Understanding the process of generating recommendations is essential in order to avoid a situation where the recommendation engine, despite its efforts, will not be able to recommend anything.

The recommendation engine has been designed in such a way that it will **always** want to display the products that have the greatest chance of purchase. In order to do so, the system must:

1. identify **the shop** for which it will generate recommendations
2. identify the **user** for whom it will recommend
3. find out **where** the recommendations will be displayed and only these parameters are required by default:
    - customer
    - cookie
    - placementId

In case of logic in which the engine is to recommend products to another product (e.g. cross-selling or upselling) the required information also is:

4. to which **product** I should generate recommendations and the parameter used for this purpose is filterProduct.

If we want to give the engine the possibility of recommending products for users already logged in the shop, we can use *userId*, and in case we want to clearly identify users, the *deviceId* is helpful.

By default, the logic of the recommendation engine can be inconsistent with the commercial policy of the online shop, which is created by the shop managers on an expert level. Example:

On the category page, the shop wants to display **only** toys, while the recommendation engine finds a hidden link between a particular toy and, for example, the glasses that people most often buy with it.

In order to adapt the logic of the recommendation engine, below we present several sales strategies and business rules that are able to change the functioning of the recommendation engine algorithms to achieve compliance with the online shop's sales policy.

## Filtering to category

The filterCategory field allows you to filter products into the appropriate category. Thanks to this, we can narrow down the results obtained from the engine to the appropriate category (the category must be sent to us in the process of catalogue synchronisation).

For example, if you have a pen product, you can narrow it down to it:

- pens
    - with a refill
        - blue colour
            - for him

In this case, the request for a frame would take the form: https://restapi.quartic.pl/store/recommendation?placementId=ID_WIDGETU&cookie=COOKIE&customer=CUSTOMER_SYMBOL&filterCategory=**for_him**

**ATTENTION:** Be sure to ensure that there will always be enough products available to fill the frame before placing the filter in the next "in category" space.

## Recommendations in the context of the product

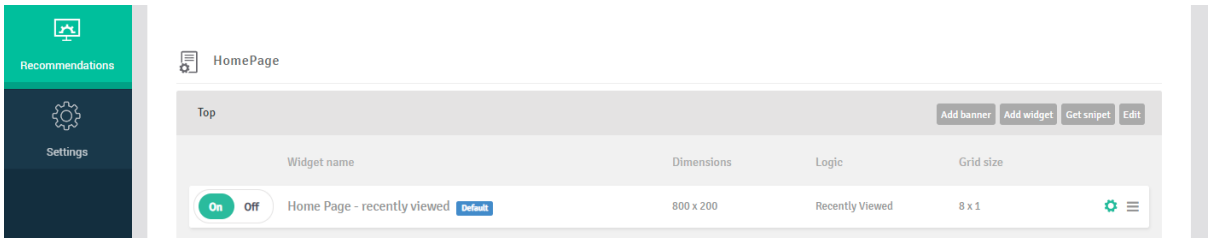
In some recommendation logics it will be necessary to transfer the product identifier to the system, for which we should look for recommendations. For example, on a product page to apply an up-selling or cross-selling strategy.

## Finding the required data

**placementId**

In order to find the recommendation identifier, you should:

1. Log in to the administration panel of the QuarticOn system;
2. Select the *Recommendations* tab from the menu on the left.;
3. Find the matching recommendation frame in the list of recommendation frames (if in step 4 - recommendation frame creation window we chose to create recommendation frames according to best practices, then the recommendation frame, which should appear at the top of the home page, will be in the HomePage directory in the Top placements);




4. From the menu on the right, select *Download snippet*;
5. From the opened modalbox select the diva identifier and remove the prefix_qS_ from it, i.e. for example:

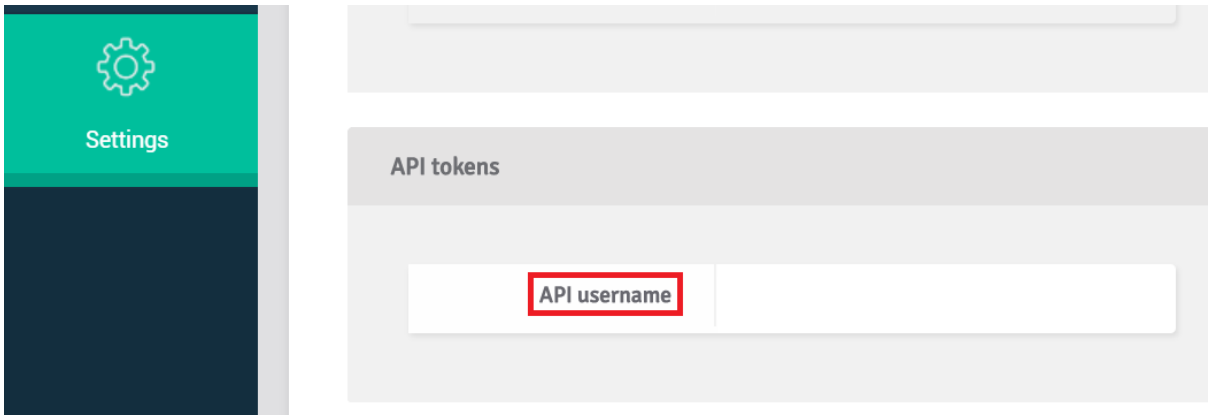
    a. copy the snippet and find the id:

    b. remove prefix:
    ~~_qS_~~1jseh
    c. and thanks to this simple procedure we found an identifier *placementu*.

**customerSymbol**

The client symbol should be delivered to us together with the API access data. However, if we do not receive it, we can get to know it in the following way:

1. Click on *Settings*;
2. Go below to *Shop details*;
3. Where we will find the tab *Tokeny API*
4. customerSymbol is placed in an *API user* section;




**Cookie**

A cookie is any identifier of the website visitor. It allows the user to identify in both cases, when he is not logged in and logged in from a given terminal device. The shop should be able to read such a cookie and send it to us both in BackEnd and FrontEnd.

## Downloading recommendations

**ATTENTION:**

Before downloading a recommendation, the option Start Trial / Start Test Period should be enabled in the user panel. Otherwise, instead of a recommendation, we will receive an error message.

Now let's test the operation of the recommendation in practice:

1. In any REST client let's call the address below:
   *https://restapi.quartic.pl/store/**recommendation**?*
   ***placementId**=IdentifierTheplaceOfthePanel&*
   ***cookie**=Any String&*
   ***customer**=YourCustomerSymbol*

2. What did the recommendation engine show?


```
{
 "status": "OK",
 "timestamp": 1516733783,
 "qrId": null,
 "duplicatesCount": 0,
 "data": [

{
"id": "ProductIdentifierInTheShop",
"title": " product title",
```

```
"image": "http://www.amautaspanish.com/blog/wp-
content/uploads/2015/10/translations-of-the-word-ok-2.jpg",

"description": " test title of product ",

"url": "https://rec-

ir.quartic.pl/c.php?qrid=ad_5a67855773d44&qdpi=1jseh&pid=4&ci=73422&c=testCustomer&p=prod
uctIdentifierinShop&ss=rtb_30428&url=http%3A%2F%2Fshop.dev%2Fp
roduct%2F111-test-product",

"price": "10,99",

"priceOld": "12,99",

"custom1": "any content that will be returned with the product in the recommendation”
"custom2": " it will then be possible to get it in the code receiving the answer",


"custom3": " and use to generate a frame",

"status": true,

"trackingString":
"eyJxcmlkIjoiYWRfNWE2Nzg1NTc3M2Q0NCIsInFkcGkiOiIxanNlaCIsInBpZCI6IjQiLCJjaSI6Ijcz
NDIyIiwiYyI6IjAyZjllMGVhZjA0Y2FlYTAiLCJwIjoiaWRlbnR5ZmlrYXRvclByb2R1a3R1V1NrbGV
waWUiLCJzcyI6InJ0Yl8zMDQyOCJ9"

}
],
}
```


Note that the engine has shown us:

a. information on the success of generating recommendations;
b. the generation date;
c. number of duplicates;
d. and:
    - previously entered product data;
    - together with a click (changed address to the product page, which informs the system about clicking on a frame);
    - and needed for the counting of statistics, trackingString;

3. Returned data can be provided in order to prepare recommendations and display on the website of the online shop.

## Identification of recommended transactions

Identification of transactions is possible thanks to the last event: eventClick which will combine clicks from a particular recommendation frame with Big Data solutions used in it. Thanks to this QuarticOn

will continue to learn how to best sell in an integrated shop, and those who have access to our panel will be able to see the impact of each individual frame on overall sales.

In order to implement eventClick, you need the trackingString parameter which you will get together with the generated frame.

**ATTENTION:**

Most of the integration can take place - thanks to a click - without sending an eventClick to the QuarticOn system.

## When is eventClick useful?

In cases when we want to use the recommendation engine, for example in a **mobile application** (making clicks is problematic here because of the application based on Activities), or AJAX shops that refresh only one container of the site, then it is better to use the product display without clicking on it and triggering eventClick at the same time.

## How to trigger eventClick?

- sent via POST or GET;
- each time the user clicks on the **recommended** product;
- the query should have a correct structure (different for the method by which we send it);

POST:

- remember to change the value of the customer field to the correct customer symbol
- send to address: https://restapi.quartic.pl/store/track/event

```
{
 "productId": "text",
"eventType":"eventClick",
 "userId": "text",
 "deviceId": "text",
 "timestamp": "121",
 "referrer": "text",
 "cookie": "text",
 "trackingString": "text returned together with the product with a recommendation”,
"customer": "symbol"
}
```

where:

- the product ID should be the same as in the synchronisation process;

- userId should be in line with the user ID of the shop. If the user is not logged in, the field should remain blank;
- It should be given either a *cookie* or a *deviceId* where:
    - **cookie** is a changeable user ID (which can be deleted/regenerated);
    - **deviceId** is unchangeable (phone's IMEI number, TV serial number etc.);
- referrer might be:
    - the correct URL of the page that trigger redirection to the product;
    - the name *Activity*, which was triggered e.g. on an Android phone before switching to the product;
- timestamp – time of occurrence of the action;
- trackingString – text returned together with product data from the recommendation. Note that each product has a **different** trackingString, and it changes **with each** recommendation.