Headless API

Author Kodmyran AB

Introduction

The headless API described in this document has a completely different use-case when compared to the other APIs in Kodmyran Commerce. It executes in the context of a user or session. This also means it runs in the Kodmyran Commerce framework user mode, not admin/supervisor mode as the classic APIs. This means it is much more restricted in the sense of what objects can be instantiated and accessed.

The sole purpose of this API is to allow any CMS or CMS-like system to be extended with shop functionality.

Kodmyran Commerce in headless mode will be a part of any request that hits your shop, the CMS system will create and draw the end-user pages based upon JSON data retrieved from Kodmyran Commerce. The session that exists between the CMS system and Kodmyran Commerce will keep track of which user is logged in to present that customers prices, their order history etc.

Kodmyran Commerce will never draw anything, it will only present the data in a JSON format; with one exception involving the cashier which can be presented as an embedded iframe to simplify deployment.

The headless API has an additional feature where it can execute a signed script, this can be used to for instance add products on one page and complete the checkout process in your regular store (different domains). See more about signed scripts here [].

If you want to integrate with an existing webshop that already includes cart functionality (e.g. Magento, Prestashop, WooCommerce etc) you should look at the JSON/REST integration API instead. This is not the API you are looking for.

Security

Each request sent to Kodmyran Commerce must be executed with a HTTP header describing an API key. The API key must never be sent directly from an end-user client; hence it can never ever be sent from a browser. In addition Kodmyran Commerce will restrict access to this API on an IP-address range basis to prevent invalid use.

The process of creating the API key is described in the API Overview

Overload protection

Unlike the other Kodmyran Commerce APIs there is no call restriction for this API, as it would directly impact end-users. Instead there is aggressive denial-of-service logic in place to counteract misuse of the system, both hardware and software solutions.

It is important to include the IP address of the end-user system requesting the call, this is used to detect bad behavior but also to tag orders with the proper remote IP. This is an absolute requirement for some payment providers.

Maintaining a session

The CMS system is responsible for keeping track of the session. The first request made on behalf of a new user will not have a session cookie, Kodmyran Commerce will return a new session ID cookie in the response. The new cookie is named "KMSESSIONID". The CMS system is responsible for storing this into the CMS session and provide this cookie in all subsequent requests.

Kodmyran Commerce will age out old sessions using standard rules, if an invalid session ID (does not exist in the session table) is received the session will be restarted and a new session ID may be returned. The CMS system is responsible for detecting this and update its local session.

Caching

Caching is very important for any high-performance system. Kodmyran Commerce deploys several layers of caching on its side and generally uses the same rules as a normal Kodmyran Frontend.

Caching within the CMS system can be trickier as there is no signal that some data is no longer valid. We recommend a conservative approach:

- Cache the basket until an operation that changes the count is used or you reach the cashier. The cache is per session.

- Cache the content of categories for 5 minutes, or until a user is logged in or out. This can be stored as one copy for non-logged in users, and variations for logged in users. If consistency requirements are high you may not be able to cache this at all.
- Cache products for 5 minutes or until an order is completed, take care to cache it differently for logged in users compared to anonymous users.
- In general, the user object need not be cached unless you need to display the customers name on each page. If so you should cache the name in the local CMS session.

The headless API is designed around a batched design, ideally there should only be one request for each end-user page load. You should do your very best to collect all operations within a single batch query. This will provide a much better experience for the end-user and also reduce the load on both the CMS and Kodmyran Commerce servers.

Cashier

The cashier part is the most complicated to integrate as it in turn interfaces with many different payment providers. Kodmyran Commerce takes a simple approach here and will by default return an iframe into which a standard cashier is integrated. This makes the logic in the CMS simple but limits what can be done design-wise.

You can also, optionally, retrieve a list of payment options and freight options and draw these yourself - but it is very important that you update the payment and freight options for each change made in the cashier. Enabling a particular payment option may turn some other freight option off, it may also add e.g. invoice fees to the cart meaning that the cart must be re-drawn too.

In general the cashier logic when not using an iframe is very complicated and requires substantial testing before deployment.

Images

When fetching products, you will receive links to several images, in their raw sizes. The sizes you receive are the default configured for that media slot. If you need more flexibility you can request images in a specific size by explicitly running the getproductimage command and using the path generated from it instead of the product defaults.

Usage

The API is quite simple and has one single HTTP endpoint to which you POST a command list, the response is JSON with a command list return array. Operation 1 in the received command list will have its output stored at position 1 in the command list response and so on.

Kodmyran Commerce will execute all operations in order, and if an error is detected will abort execution at that point and return a non-200 response.

The commands in the command list are essentially mapped 1:1 with the existing AJAX and low-level frameworks, this API is a small shim in front of the native APIs that extend their use to external CMS systems.

When a native command return a complex object it will be returned as a simplified JSON array, and prices etc will be adjusted to reflect the current audience (logged in user, discount schemes etc).

Each command block consists of a field named *verb*, and optionally, an argument block stored in the field *arguments*

```
{
    verb: "clearrow",
    arguments: {
        prodno: 1092
    }
}
```

Each corresponding response contains a *status* field and a *data* block. The data block will always exist, but may be empty if the specific command does not produce any result.

```
{
    status: "OK",
    data: {
    }
}
```

The endpoint for all calls into Kodmyran Commerce Headless API is:

`https://testaccount.shop4sale.se/headless/`

API groups

The API has been grouped into similar operations below to make integrating and understanding their use easier.

Cart

This group is used to add/remove products from the cart and to keep an inventory of the current contents.

clearcart

This command will empty the entire cart, apart from products that can not be removed (e.g. permanent freight or invoice costs)

Input:

None

Output:

None

clearrow

This command will remove a complete row from the cart matching the specified product number. You can optionally include a GUID reference to remove a specific row, and/or a manufacturing specifiction to apply the operation only to rows with the same manufacturing specification.

Input:

Argument	Format	Description
prodno	int	The product number (***required***)
guid	string	Apply to rows matching this GUID only (***optional***)
mspec	string	Apply to rows matching this manufacturing specification only (***optional***)

Output:

None

add

Add a product to the cart, will increase the quantity if an existing matching row already exists. If a row does not yet exist one will be created automatically.

Input:

Argument	Format	Description
prodno	int	The product number (***required***)
quantity	int	The amount to add, positive integer, if not specified 1 is assumed (***optional***)
guid	string	Apply to rows matching this GUID only (***optional***)
mspec	string	Apply to rows matching this manufacturing specification only (***optional***)

Output:

None

remove

Input:

Argument	Format	Description
prodno	int	The product number (***required***)
quantity	int	The amount to remove, positive integer, if not specified 1 is assumed (***optional***)
guid	string	Apply to rows matching this GUID only (***optional***)
mspec	string	Apply to rows matching this manufacturing specification only (***optional***)

Output:

None

getcart

Input:

None

Output:

A table of cart rows, each row containing the product name, the quantity, pricing, VAT rates and any discount applied to the row. The format of the product name is determined by the backend settings (include/exclude manufacturer name, include variation in title etc).

setquantity

Set the quantity on an existing row. If the product does not exist it will not be added to the cart.

Input:

Argument	Format	Description
prodno	int	The product number (***required***)
quantity	int	The new quantity, greater than zero (***required***)
guid	string	Apply to rows matching this GUID only (***optional***)
mspec	string	Apply to rows matching this manufacturing specification only (***optional***)

Output:

None

updatecart

Updatecart is called to inform backend payment systems to update their view too, it is used with third-party cashiers that manage the entire cashier process (e.g. Klarna Checkout). It enables solutions where the user can increase/decrease the quantity on the same page as the cashier.

Input:

None

Output:

None

Session

The session group contains generic information about the current user, such as their preferred language, default currency and so on.

clearlocale

Reset the current locale. This will remove any selection the user has made with respect to country, language and currency. They will all revert to their store default.

getlocale

Retrieve the current locale settings. You will receive the currently selected country, language and currency.

setlocale

Set the locale data, for Kodmyran Commerce this means the country, language and currency.

clearfilter

Reset the attribute filter completely

getfilter

Retrieve the current attribute filter

setfilter

Add a part to an attribute filter. To reset a filter partially you can use this call and specify a blank string for value.

login

Login the current user

logout

Logout the current user

getwarehouse

Retrieve the current warehouse ID.

getwarehouses

Retrieve a list of available warehouses

setwarehouse

Set the warehouse to use for any order created within this session. This command can be used to place an order against a physical store for pickup.

getvat

Returns the current VAT state.

setvat

Set the VAT state, turn on or off displaying prices with or without VAT.

getbrands

Get a list of available brands

getbrandmodel

Get the brand model ID currently selected. A return value of 0 means no filter is in effect.

getbrandmodels

Get a list of available models for a specific brand

setbrandmodel

Set the brand model ID to use when filtering.

clear

Clears the entire session, including the cart

User

This group is used to create a new user profile, update an existing profile and to login/out of Kodmyran Commerce.

getuser

Retrieve the address information of the currently logged in user

setuser

Update the address information of the currently logged in user

setpassword

Change the password of the currently logged in user

resetpassword

Send a password reset e-mail to the e-mail address of the specified account. The method used (generate password, password reset link etc) is determined by the settings inside Kodmyran Commerce.

makeuser

Create a new account.

You should not require customers to create an account! Kodmyran Commerce provides a cashier that can handle anonymous users correctly. The page housing this command should have a secondary position on your site.

addwish

Add a product to the currently logged in users wishlist

clearwish

Remove/clear the wishlist of the currently logged in user.

sendwish

Sends a wishlist from Kodmyran Commerce to a specified e-mail address. This operation can only be invoked if the user has been properly logged in.

Order

This group can be used to fetch the customers orders and present them on e.g. a “My Page” page.

getorder

Get a specific order, including the orderrows. This will also include any package references known for the order.

getorders

Get a list of orders for the currently logged in user, including the order timestamp and current status.

cancelorder

This command will cancel an order, this is only permitted if the order is in certain states.

Cashier

The cashier object is used to handle the final payment and selection of freight method.

cashier

Retrieve the iframe data to be used for an embedded cashier

getcoupon

Get the coupon currently selected

setcoupon

Attempt to select a discount coupon

setmodulefilter

Set a filter for the various modules, you can specify that modules must be tagged as either for private citizen use, for company use only or for both classes. Module lists will be filtered on this to present only valid freight and payment options.

If you have a field on your page where you select company or private citizen, that operation should trigger a call to this command.

getpayment

Retrieve the current payment ID. Note that Kodmyran Commerce always have a default payment method selected, even if the customer has never visited the cashier.

getpaymentmethods

Retrieve a list of payment methods. This list has been filtered based upon A/B testing rules as well as the modulefilter.

setpayment

Set the payment ID

getfreight

Retrieve the current freight ID. Note that Kodmyran Commerce always has a default freight method selected, even if the customer has never visited the cashier.

getfreightmethods

Retrieve a list of freight methods. This list has been filtered based upon A/B testing rules as well as the modulefilter.

setfreight

Set the freight ID

getaddresses

Retrieve a list of valid addresses for this customer, based upon their company registration number or personal ID. This method is available for customers in Sweden and Norway only, and only with specific payment options enabled.

setcomment

Attach a comment to the soon-to-be order

Category

The commands in the category group are used to retrieve the products to display in a particular category, in the order selected by the customer (or default if no selection has been made).

getcategories

Retrieve all category IDs and names belonging to a particular parent category. Use the special category ID 0 to request the top-level categories of the store.

getcategorycontent

Retrieve the products inside a particular category. The list will be returned in order and should not be further sorted on the CMS side. You can specify a starting offset and amount of products to return.

If an attribute filter and/or brand model filter is effective the content will reflect those filters.

getcategorysortorder

Return the current category sortorder

setcategorysortorder

Set the category sortorder

getcategorytopsellers

Return a list of products for the top-sellers of this category

Product

All product related commands are grouped into this section. You can retrieve product data and also invoke operations such as the wishlist functionality through this interface.

getproduct

Retrieve a product object, including prices and various other data. This call will typically be made on a product detail page.

getproductimage

Requests a specific product image or size

getproductreviews

Retrieve a list of reviews for this product

reviewproduct

Create a product review, with a vore, author and comment field

tipproduct

Sends a product tip e-mail message to the specified receiver. This call is by default disabled in Kodmyran Commerce and must be explicitly enabled. This is done to ensure the use of this command is protected by a CAPTCHA field or similar mechanism before enabling.

You are ***REQUIRED*** to build a captcha validation around this function

voteproduct

Similar to the reviewproduct command, but will only add a vote between 1 and 5. It does not add any author information or comment.

subscribestock

Add a subscriber to the watchlist for this product. Upon the product becoming available again an e-mail message will be dispatched to the subscriber which typically includes a direct link back to the product.

unsubscribestock

Remove a subscriber from the product watchlist.

Search

The search group is used to implement searching, the two commands are quite similar but return different amounts of data.

getsuggestions

The getsuggestions call will run the query and return a number of hits that it believes will match. The number of entries returned is limited. You will typically receive two separate lists, one for products and one for categories.

It is also possible that the output will vary somewhat as Kodmyran Commerce supports pluggable search engine providers. Depending upon the selected provider, categories may not be available, or you may get an additional section with spelling suggestions.

The purpose of this command is to provide functionality for a drop-down list beneath the search box as the user is typing, you should use a typeahead library to avoid sending to many requests.

There is a minimum length of the search term of 3 characters.

search

The search command will run your regular search, it supports paging by providing a starting offset and a limit of how many entries to return.

Signed script requests

The headless API can be used to create a script of commands to execute in one-go, this is achieved by a remote server registering a command script and getting a signature in response. The client is then redirected to a specific URL in the headless API that will execute the script, provided a correct signature, and redirect the user to a location specified in the script.

You send the script in a very similar way as a normal headless request, the difference is the URL used which is /headless/sign and that some commands accept additional parameters. Most specifically you can adjust the pricing of products added to the cart in the request (for dynamic pricing). However the product referenced must exist inside Kodmyran Commerce.

A signed request does not have to contain a meta section in the incoming request, and will typically not have one either. The request to sign must always come through your own server, never ever try to sign directly on the client as that would expose your API key directly. Hence there are two files in the below example, one for client side handling and one acting as a proxy script server side that creates the actual command script and signs it.

Even though this example script is written using PHP, any server language can be used that can communicate using JSON messages.

Example code:

index.html

```
<html>
<head>
    <title>Headless test</title>
</head>

<body>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>

<script type="text/javascript">

$(document).ready( function() {
    $('form').on('submit', function( ev ) {
        ev.preventDefault();

        // Fire away an AJAX request to build the JSON request
        var req = {
            "prodno": 13575
        };

        $.ajax({
            "url": "https://myexternalsite.se/code/server.php",
            "data": req,
            "method": "POST",
            "dataType": "json",
            "success": function( data, textStatus, xhr ) {
                // Ok, create a second AJAX request
                submitPurchase( data.signature, data.script );
            }
        });
    });
});

function submitPurchase( signature, scr )
```

```
{
    $.ajax({
        "url": "https://test.mydomain.se/headless",
        "headers": {
            "X-API-Signature": signature,
            "Content-Type": "application/json"
        },
        "data": JSON.stringify( scr ),
        "method": "POST",
        "dataType": "json",
        "xhrFields": {
            "withCredentials": true
        },
        "success": function( data, textStatus, xhr ) {
            if( data.response[1] )
            {
                if( data.response[1].location )
                    window.location.href = data.response[1].location;
            }
        }
    });
}

</script>

    Press button to add product to cashier and redirect to site X and the cashier.

    <form>
        <input type="submit" name="Send" value="Send me" />
    </form>
</body>
</html>
```

server.php

```
<?php

    $apikey = "29f038b6c333b8c29f6243a5a4f989f4939e884a";

    $req = array(
        'request' => array(
            'commandlist' => array(
                array(
                    "verb" => "add",
                    "arguments" => array(
                        "prodno" => 13575,
                        "price" => 154.50
                    ),
                ),
                array(
                    "verb" => "redirect",
                    "arguments" => array(
                        /* Redirect directly to cashier, can go somewhere else if desired */
                        "location" => "https://test.mydomain.se/qcashier.php",
                        "mode" => "none"
                    ),
                )
            )
        )
    );

    $nreq = json_encode( $req );

    $url = "https://test.mydomain.se/headless/sign";

    $headers = array(
        'Accept: application/json',
        'Content-Type: application/json',
        'X-Api-Key: '.$apikey,
    );

    // Initiate curl and set curl options
    $handle = curl_init();
    curl_setopt( $handle, CURLOPT_URL, $url);
    curl_setopt( $handle, CURLOPT_RETURNTRANSFER, true );
    curl_setopt( $handle, CURLOPT_SSL_VERIFYHOST, false );
    curl_setopt( $handle, CURLOPT_SSL_VERIFYPEER, false );
    curl_setopt( $handle, CURLOPT_HTTPHEADER, $headers );
    curl_setopt( $handle, CURLOPT_POST, true );
    curl_setopt( $handle, CURLOPT_POSTFIELDS, $nreq );

    $response = curl_exec( $handle );
    $code = curl_getinfo( $handle, CURLINFO_HTTP_CODE );
    curl_close( $handle );

    $rsp = json_decode( $response );

    if( isset( $rsp->signature ))
    {
        header( "Content-type: application/json" );
```

```
        $r = array(
            'signature' => $rsp->signature,
            'script' => $req,
        );

        print json_encode( $r );
        exit( 0 );
    }

    // Error handling code should be here
    die( "An error occurred" );

?>
```

Example request/response

Request sent to headless endpoint as a POST request:

```
{
    meta: {
        ip: "81.227.92.108"
    },
    request: {
        commandlist: [
            {
                verb: "getcurrency"
            },
            {
                verb: "getpayment"
            }
        ]
    }
}
```

Kodmyran Commerce responds with:

```
{
    meta: {
    },
    response: [
        {
            status: "OK",
            data: {
                currency: "SEK"
            }
        },
        {
            status: "OK",
            data: {
                paymentid: 36,
                type: "DIBS",
                displayname: "Kreditkortsbetalning"
            }
        }
    ]
}
```