

Please note: This article is OTYS only, and can only be read by users that have logged in.

Overview

The OTYS Site Builder (OTYS CMS) is an easy to use CMS system for our customers, and it is designed with one simple guideline: "Any recruiter should be able to create their own website".

Read more about the OTYS Site Builder and it's possibilities on the dedicated [documentation page](#)

Activation

An OTYS employee (consultant/support/csm) needs to enable the Site Builder for a client through a partner action in the Your!T CRM: "Enable Site Builder" this will guide you through the activation. But still read the information below, as it is useful to understand the dependencies.

Some of the requirements are configured/handled by the partner action, and some you will have to configure yourself.

Dependency	Explanation	Handled by partner action?
OTYS Client	You can only use the cms if you are authenticated as an user of a client of OTYS	No, if no client is present it will tell you
MyOTYS	The client will need to have an MyOTYS application admin which we can use to create an API key for the CMS	No, if no MyOTYS connection can be found, it will tell you.
CMS Partner	The CMS is an partner on it's own, and only clients that have this partner enabled will be able to use it.	Yes, the partner action will activate the CMS partner for the client
Slug system	The CMS can only list/expose jobs when the Slug system is enabled, this is done via setting: SE3452 but only after careful consideration as the slug system is not multisite compatible (as of 08-2025) and other websites of the client can be affected.	No, you will have to configure this after careful consideration of the clients situation.
Key or Super user present	The client will need at least 1 key or super user, for the Partner action to be able to add a "CMS Navigation item" to the navigation bar of the customer trough partner actions.	No, you will have to create one or give (temporary) rights to an existing user.
API Key Limit not reached	For the partner action to be able to create an API key, the clients limit must not be reached yet.	No, you will have to manage this yourself.
WebAPI partner enabled for client	The CMS is depending on a connection to the WebAPI and that can only be made when the WebAPI partner is enabled for the client	Yes, the partner action will activate the WebAPI partner for the client
WebAPI setting enabled for client	The WebAPI activation also depends on setting SE3587 so other WebAPI settings are configurable for the client	Yes, the partner action will activate the setting.
WebAPI Api Key	The CMS is depending on a connection to the WebAPI and that can only be made with such API Key	Yes, the partner action will trigger an creation request trough the CMS and MyOTYS
At least one user with permissions	To be able to actually see the navigation item in OTYS Go! and to use the CMS, the user should have PA212 enabled.	Yes, the partner action will enable/disable these rights for the selected users.

There are also some other things to keep in mind while activating the new Site Builder, that this partner action doesn't remind you about (yet).

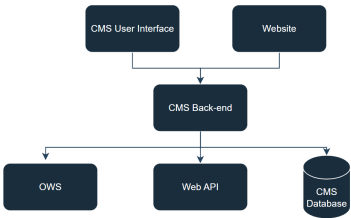
Dependency	Explanation
New Document Merge Fields	Some clients still use the old document merge fields, and those need to be transformed.
Google For Jobs	If Google For Jobs is enabled, The websites vacancy details will automatically contain the structured data
Hosted page	All websolutions still need an hosted page, to support GDPR, Newsletters and online CV/Job presentation

Technical components

Please note that this could be an complex topic, documented for our technical colleagues. We do not expect everyone to comprehend this.

Component	Technology	Internal project	Description, what it's used for
CMS User interface	Angular	OTYS Website FO Front-end	The place a user creates content
CMS Backend (api for interface)	Symfony + API Platform	OTYS Website FO Back-end	The interface talks to the back-end trough an REST API
Website	Symfony	OTYS Website FO Back-end	The website themselves
Web API	Symfony + API Platform	OTYS Web API	A REST API to communicate to OWS
CMS Database	PostgreSQL	OTYS Website FO Back-end	Here we store pages, settings and more
OTYS Web Services	PHP	OWS	Your!T Communication (settings etc)

Below a schematic on how these different components talk to each other



Development flow

Below the description of our development flow. With an example feature: "#123456 CMS: New widget for reviews"

1. A new task get's created in OTYS Go! (#123456)
2. Assuming this only requires work on OTYS FO Backend, the developer will make a new branch based upon the "stable" branch, the new branch will be called: "tasks/#123456-CMS-New-widget-for-reviews"
3. Working on their local machine they finish the feature, and merge their branch into "main" (a.k.a. Testing)

- 4. After the automatic build is done, they go to test their new feature on Testing, and after confirming it works, they mark the task to be tested.
- 5. If the testers are done and happy, the task/branch will get merged into "stable" (a.k.a. Staging)
- 6. And when staging is working with all it's new features/changes we will move on
- 7. Making sure if there are new features they are documented in Doc360 and if needed in the Guided tour
- 8. When we are ready to do a new live push we will replace the current Production environment with the Staging environment.
- 9. And now that we are live with new features/fixes we make sure to inform any stakeholders (customers) about it.

Technical overview of the CMS Interface (OTYS FO Front-end)

Please note that this could be an complex topic, documented for our technical colleagues. We do not expect everyone to comprehend this.

Below a list of different frameworks, plugins and tools we use for the CMS interface. We will monitor all these packages and make sure they are always up to date where possible.

Topic	Description
Front-end framework Angular	We chose for Angular because that's the most known front-end framework by OTYS Developers as of 2025 and it's capability of handling large complex applications
Styles using SCSS	SCSS also known as SASS is the preferred solution by OTYS Developers
WYSIWYG Editor by NGX-Editor based upon Prosemirror	The NGX-Editor does not need any licensing, is lightweight and easy to modify, like we did with links and AI Assistant
Dragging & Sorting by Sortable	Sortable is the most lightweight and dynamic library we we able to use.
Translations by Transloco	Transloco has is feature rich, and has the features we require
Icons by Font-Awesome	FontAwesome is already used in other applications of OTYS and we have a Pro license for it
Sliders by SwiperJs	SwiperJs is feature rich and updated frequently.
Colorpicker by NGX-Color-picker	NGX Color Picker is the only one as of 2025 that is lightweight and easy to use dynamically like we do
Guided tours by Angular Shepherd	Angular Shepherd is really basic, and allows quite some modifications like we require (think of styling and conditional tours)
Tracking by Matomo	Matomo is already used by many of our applications to track user behaviour and help us follow customer tracks when bugs occur
Debugging by Faro for Grafana	Faro reports timing metrics and debug information to Grafana, so we can identify issues before customers report them.
Package management by Renovate	Renovate automatically updates packages via merge requests.

Technical overview of the CMS Backend (OTYS FO Back-end)

Please note that this could be an complex topic, documented for our technical colleagues. We do not expect everyone to comprehend this.

Below a list of different frameworks, plugins and tools we use for the CMS back-end. We will monitor all these packages and make sure they are always up to date where possible.

Topic	Description
PHP version 8+	The latest version of PHP
Back-end framework Symfony	Symfony is chosen to be our preferred back-end framework for applications of this scale
REST API via API Platform	API Platform is a Symfony plug-in that enables you to create a REST API which is automatically documented
Database on PostgreSQL	A relational database system that is similar to MySQL but is more advanced and capable
Content translations trough Gedmo	A Symfony package allowing us to mark specific parts as translatable. And it will store the translations for us
View files via TWIG	A Symfony templating engine allowing us to create HTML based upon the CMS contents
Grid layout via Bootstrap	The go to system for grid layouts.
File storage and serving via Cloudflare	We use Cloudflare to serve images,videos and other files uploaded by users. We use smart tools to serve JPG'S & PNG'S as WebP optimizing performance
Firebase JWT Authentication	We are using JSON Web Token authenticating via HTTP ONLY cookies, meaning only the server can access it.
Sliders by SwiperJs	SwiperJs is feature rich and updated frequently.
Icons by Font-Awesome	FontAwesome is already used in other applications of OTYS and we have a Pro license for it
Styles using SCSS	SCSS also known as SASS is the preferred solution by OTYS Developers
Package management by Renovate	Renovate automatically updates packages via merge requests.

Authentication flow

Please note that this could be an complex topic, documented for our technical colleagues. We do not expect everyone to comprehend this.

Currently there is only 1 form of authentication, and that is trough OTYS Partner SSO, and that will allow you to access the site-builder and manage content.

It works as follows:

- The user clicks the "CMS" icon in their navigation bar in OTYS Go!
- OTYS Go! will redirect the user to our authentication endpoint with some POST data, containing a "sessionToken"
- Our application will now verify that sessionToken trough the OWS SsoService in combination with the Partner API Key of the CMS project.
- When the CMS partner is enabled and everything is correct, we will synchronize the OTYS data with the CMS data.
- When Synchronisation is done without any errors, we will authenticate the user
- We will create a HTTP Only cookie containing an JWT which represents user data that is only readable by our application because the JWT is encoded with secrets only the app knows
- We will now redirect the user to the Front-end. The cookie will be set, but is not accessible with javascript making it extra secure.
- We will invalidate this JWT when the user starts using another IP, User-agent etc. Making sure it can never be used when hijacked.
- The JWT is now valid for 14 hours, after that the user has to sign in again

Caching

Please note that this could be an complex topic, documented for our technical colleagues. We do not expect everyone to comprehend this.

To make an application efficient—both in using its own resources and in minimizing the load on other services—caching plays a crucial role. Caching refers to the practice of temporarily storing data in a quickly accessible location so that repeated requests for the same information can be served faster.

For example, instead of retrieving a list of 50 jobs with all their details from OWS every time (which takes about 1.5 seconds), the application can check if the exact same request was recently made. If so, it can return the cached result instead (taking around 0.0003 seconds).

This approach greatly reduces response times, saves processing effort, and avoids unnecessary calls to external or even internal systems.

For the sitebuilder we using various methods of caching on different application layers. Below each method, and an explanation on how it works.

Website caching: Web pages are served with Cache control headers

When you visit one of our clients websites (that are made with the Site-builder) the server will tell (for static pages) your browser may remember the webpage and its contents for at least 60 seconds. This means for example that when you open a websites homepage and go to the vacancies overview page, you will have asked the server twice for a webpage, which was freshly served to you. But now when you go back to the homepage within 60 seconds, the browser won't ask the server for the homepage contents anymore, as the server told your browser to keep it in mind for at least 60 seconds. Of Course for more dynamic pages like application forms etc. We wont send such "Cache-Control" headers. The current "Cache-Control" looks something like this:

Plain text
cache-controlmax-age=60, private, stale-while-revalidate=60

The "private" part is an important part of this "Cache-Control" header, as it will tell servers that are serving this content to never cache it themselves, and this content can only be cached by the end-user.

Website caching: Styles, fonts & scripts are served with Cache control headers

A web page consists of more then just "HTML"; It also requires stylesheets, scripts, fonts and images/videos. For this caching method we are focussing on those stylesheets, scripts & fonts. If any of those files get's served by our server it will also respond with an "Cache-Control" header, however a more efficient one, that will tell the browser to cache the contents for a whole year! That "Cache-Control" headers looks like this:

Plain text
cache-controlpublic, max-age=31536000, immutable

The "public" part is an important one for this "Cache-Control" header, as it will tell servers that they can cache it themselves as well, and serve everyone the same version.

Website caching: Images & videos are served with Cache Control headers

All websites have multimedia (like images and videos), for the Site-Builder we store that type of content not on our own servers, but on those of Cloud-flare a third party specialised in serving content quickly and efficiently. If a customer uploads for example an JPG for their header, we will (on the website it self) serve it as an WebP file, this is done through Cloud-flare, which on it's end also server-side caches these files, so even when it's not cached by the browser, they will be served rapidly from one of their web servers nearby the user. Cloudflare also sends one year Cache-Control headers with these images & videos and they look something like this:

Plain text
cache-controlmax-age=31536000

Server side caching: Requests to OTYS API's are cached on the server

Our back-end (Symfony) is also caching data, for example requests to the Web API or OWS are cached. If you visit a customers website and go to the vacancies overview chances are high that you were served cached content. Whenever the back-end needs to return data that relies on OWS or the WebAPI it first checks if it doesn't already have that specific data in its cache, and if so returns that. And if it does a request that wasn't stored in cache yet, it will store it in cache after retrieving the data "live". When storing those cache items in Symfony's file system cache, it gives it a name (key) based upon the request, but also smartly tags it, with for example the client_id, vacancy_id or even matchcriteria_id. So if we later want to remove all cache for a client or certain criteria we can do that. (In the Site-builder configuration you can click the "Remove cache" button.

Server side caching: Invalidating cache records with webhooks

When the Site-builder is activated for a customer for the first time, we will also register webhooks through the WebApi, these webhooks will tell the Site-builder if a client makes changes to vacancies, match criteria or even application forms. So if any website related content in OTYS Go! changes, we are notified immediately and invalidate (remove) cache related to that change, which is done by cache-tags you can read about above.

CMS User Interface caching: Cache semi-static data in localStorage

When a user is working in the CMS it (in the background) needs to have quite a bit of data, think of possible widgets, sections etc.. Instead of loading these from the server every time they need them, we cache such information in the browsers 'local storage' function. This way the CMS is quicker and more efficient with server resources.

Translations

Within this project we have multiple ways of translating content and interfaces. There are 4 different types of translations, below for each type explained how it is stored, translated and used.

Website content translations

When a client creates a multilingual website, they can translate their content in multiple languages (the languages they have in OTYS Go! defined in DB109). Content is stored in "control_values" these are then connected to widgets, footers etc.. Translated content is stored in "control_value_translations" which are connected to "control_values". This is automatically managed by Gedmo. Read more on translating content [here](#). Possible languages for this type of translation are: "Dutch", "English", "German", "French", "Czech", "Spanish", "American English", "Polish", "Slovak", "Romanian", "Turkish", "Russian", "Ukrainian" & "Danish"

Website static translations

Some words or sentences are not editable by the customer, and are managed in the Symfony project trough these files /translations/fo.[langcode].yaml. Things like: "404 - Job not found" or the labels of flags: "Dutch", "English". For now these are not editable by the client and are the same for all clients. We might change this in the future. For this we use the default Symfony translation interface. Possible languages for this type of translation are: "Dutch", "English", "German", "French", "Czech", "Spanish", "American English", "Polish", "Slovak", "Romanian", "Turkish", "Russian", "Ukrainian" & "Danish"

Control definition translations

In the backend we have defined which widgets, sections, and inputs there are. For example widgets have controls, like "Header title" or "Background color" to explain to the user in the CMS what they are editing we have given these controls translations as well. All controls consist of an "Label" and an "Explanation" (the info icon). These translations are stored and managed in the Symfony project: /translations/ui.[langcode].yaml. For this we use the default Symfony translation interface. Possible languages for this type of translation are: "Dutch", "English", "German", "French"

CMS Interface translations

The CMS interface also is translated and it's translations are stored in the /assets/i18n/[langcode].json files. For this we use the Angular Transloco plugin. Possible languages for this type of translation are: "Dutch", "English", "German", "French"

The different entities & the logic behind the data structure

Please note that this could be an complex topic, documented for our technical colleagues. We do not expect everyone to comprehend this.

The sitebuilder has a specific way of storing content & settings. Below explained how different entities are set up, what they consist of and how they are stored.

Client

The entity mirroring OTYS clients. This entity is persisted in the database.

Property	Data type	Explanation
id	UUID	An unique identifier
otysClientId	Integer	The uid of a client in OTYS.
otysApiKey	Varchar	The (openssl) encrypted Api key of a client used for WebApi communication
languages	Text	Comma separated language codes, in sync with DB109
lastSync	Timestamp	A timestamp representing the last sync. Syncs are done every authentication

Website

The entity mirroring OTYS websites. This entity is persisted in the database.

Property	Data type	Explanation
id	UUID	An unique identifier
otysSiteId	Integer	The id of the website in OTYS (1, 2, 3 etc..)
domain	Varchar	The Custom Domain of a website
domainVerified	Boolean	Whether we have verified ownership of the custom domain
domainPointingToOtys	Boolean	Whether we have seen the A-Record pointing to our servers at least once
theme	Varchar	Which theme is in use
finishedOnboarding	Boolean	Whether this websites onboarding was completed
otysBrandName	Varchar	The name of the website as it is called in OTYS
brandPrefix	Varchar	The prefix in front of .otys.website
defaultLanguage	Varchar	The default language of this website
active	Boolean	If an OTYS site is removed, we deactivate it with this
live	Boolean	Mark an website as live, (and we will register webhook)
clientId	UUID	Connection to the client entity
homePageId	UUID	Connection to the websites homepage (page entity)
vacanciesOverviewId	UUID	Connection to the websites vacancies overview (page entity)
footerId	UUID	Connection to the websites footer (footer entity)
redirectToWww	Boolean	If enabled custom domain websites will use www. in front of it
lastSync	Timestamp	A timestamp representing the last sync. Syncs are done every authentication

Website Setting

This entity is used for storing website related configuration, like: fonts, colors, brandname, logo etc..

Property	Data type	Explanation
id	UUID	An unique identifier
name	Varchar	The website setting ENUM reference (so the type of setting)
websiteId	UUID	Connection to website
controlValueId	UUID	Connection to a controlValue entity storing the setting value

Web Api Token

This entity is used to store the WebApi token (JWT) for a client, this is done by exchanging the Api key of a client.

Property	Data type	Explanation
id	UUID	An unique identifier
accessToken	Text	The JWT
expiresIn	Timestamp	The date this token will expire
clientId	UUID	Connection to the client it belongs to

Footer

This entity represents the footer of an website. Mostly used to define the type and connect controlValues to.

Property	Data type	Explanation
id	UUID	An unique identifier
type	Varchar	The Footer Enum

Page

One of the most important entities. Every website has pages.

Property	Data type	Explanation
id	UUID	An unique identifier
published	Boolean	Whether this page should be reachable.
hideInNavigation	Boolean	Whether to show this page in the navigation bar
title	Varchar	The page title, shows up in navigation and browser tab
position	Integer	A numeric value representing the order of the navigation
slug	Varchar	The URL of this page.
useExternalUrl	Boolean	A page can be marked as external with this

externalUrl	Varchar	When someone clicks this page in the navigation they get redirected here.
creationDate	Timestamp	The timestamp of when this page was created
lastUpdatedDate	Timestamp	The timestamp of when this page was last updated
metaDescription	Varchar	The SEO meta description of a page
metaKeywords	Varchar	The comma seperated keywords of a page
noIndex	Boolean	Whether to let this page be indexed by search engines
websiteId	UUID	The connection to the website this page belongs
parentId	UUID	Possible connection to a parent page, it will then show up as a child item
metaOgImageId	UUID	Reference to mediaObject for the OG (sharing) image

Page translation

An entity managed by the Gedmo plugin, storing certain field translations in different languages.

Property	Data type	Explanation
id	Integer	An unique identifier
locale	Varchar	The locale this translation is for
objectClass	Varchar	A PHP class reference
field	Varchar	Which field of the class (example: textValue)
foreignKey	UUID	The UUID of the translated controlValue
content	Text	The actual translation

Section

This entity is making up the horizontal building blocks of a website.

Property	Data type	Explanation
id	UUID	An unique identifier
padding	Varchar	How much space inside the section based upon enum(xs, s, m, l, xl)
margin	Varchar	How much space outside of the section based upon enum (xs, s, m, l, xl)
fullwidth	Boolean	If true the columns spread the entire page width instead of the container
verticalAlignColumnsCenter	Boolean	Whether to vertically align the columns center
backgroundColor	Varchar	A HEX Color code for the background of the section
textColor	Varchar	A Hex color code for the text color of the section
position	Integer	A numeric value representing the order of sections
pageId	UUID	The connection to the page this section is on

Bootstrap Column

This entity is making up the vertical building blocks of a website. (Column is a reserved table name, so picked Bootstrap Column)

Property	Data type	Explanation
id	UUID	An unique identifier
span	Integer	A number between 1 and 12 defining the relative width
bootstrapOffset	Integer	A number between 1 and 12 defining the offset (left) relative
position	Integer	A numeric value representing the order of columns
sectionId	UUID	The connection to the section this is in

Widget

This entity is representing all different kinds of content. Based upon a type enum. Widget definitions are defined by code.

Property	Data type	Explanation
id	UUID	An unique identifier
type	Varchar	One of the widget Enums
position	Integer	A numeric value representing the order of widgets inside a column
bootstrapColumnId	UUID	The connection to the bootstrapColumn this is in

Control value

This entity is representing all dynamically stored data based upon the abstract entity "Control" which is just an enum.

Property	Data type	Explanation
----------	-----------	-------------

id	UUID	An unique identifier
name	Varchar	The name based upon the abstract controls
type	Varchar	The control type
stringValue	Varchar	If the control type is defined to use stringValue as storage this field will contain the value
booleanValue	Boolean	If the control type is defined to use booleanValue as storage this field will contain the value
integerValue	Integer	If the control type is defined to use integerValue as storage this field will contain the value
textValue	Text	If the control type is defined to use textValue as storage this field will contain the value
position	Integer	A numeric value representing the order of a control (only when it has a parent, like a repeater)
mediaObjectId	UUID	If the type is mediaObject this represents the value as a connection
footerId	UUID	The possible connection to a widget, uniquely connected through id and name
widgetId	UUID	The possible connection to a widget, uniquely connected through id and name
parentId	UUID	The possible connection to a parent control (repeater controls us this)

Control value translation

An entity managed by the Gedmo plugin, storing translations in different languages.

Property	Data type	Explanation
id	Integer	An unique identifier
locale	Varchar	The locale this translation is for
objectClass	Varchar	A PHP class reference
field	Varchar	Which field of the class (example: textValue)
foreignKey	UUID	The UUID of the translated controlValue
content	Text	The actual translation

Media Object

A quite versatile entity, used for Folders and actual file references

Property	Data type	Explanation
id	UUID	An unique identifier
name	Varchar	Initially this represents the filename, but is updatable, or the foldername
type	Varchar	Either “folder” or “media”
alt	Varchar	Alternative text for when the resource is not loaded (yet)
creationDate	Timestamp	When this folder or file was created/uploaded
lastUpdatedDate	Timestamp	When this item was last updated
mimeType	Varchar	The mimeType of the file (or nothing in case of folders)
focalPointX	Integer	The position on the X axis where the subject should be according to the user
focalPointY	Integer	The position on the Y axis where the subject should be according to the user
size	Integer	The filesize defined in bytes
clientId	UUID	The connection to the client entity
parentId	UUID	The connection to a possible parent entity

Example: A written down explanation of how a “Header Advanced Widget” is set up in the code

In the project there is a Enum that defines the “HEADER_ADVANCED” case, it defines which Class represents it, so later on when a page is rendering we know based upon that string in the database “HEADER_ADVANCED” what Class defines it, and how to render it. This Class tells us the following:

- Widget name in this case: “Header Advanced” (used for picking it in the CMS interface)
- Widget category, in this case “Headers” (used for picking it in the CMS interface)
- Minimum columns, in this case: “12” (only shows up in the widget picker of a section that has a 12 column bootstrap configuration)
- Controls, in this case:
 - Wysiwyg Control (Keep in mind that these controls are classes as well, and in those classes is defined how they are setup to store data)
 - Name: “text” (The control name is stored in the database as a reference to this control)
 - Label: “Content” (The label is shown in the interface on top of the control)
 - Explain: “The content on top of the images” (The tooltip contents behind the question mark icon in the interface)
 - Category: “Content” (In the interface there are often 2 category tabs, content and configuration)
 - Translatable: “True” (Whether this value is translatable)
 - Repeater Control
 - Name: “slides”
 - Label: “Slides”
 - Explain: “This slides to show in the header”
 - Category: “Content”
 - Controls:
 - Image
 - Name: “image”
 - Label: “Image”
 - Explain: “The image to show in the header, it will be used as a background image”
 - Select control
 - Name: “aspect_ratio”
 - Label: “Aspect ratio”
 - Explain: “The aspect ratio of the header, ‘auto’ will use the content as a reference, while the others will set a fixed a width to height ratio”
 - Options: [.... The options....]
 - Default value: “4 / 1”
 - Device: “Desktop” (This controls value is only used for desktop, other controls are defined for the tablet and mobile value)
 - Category: “Configuration”

- And many more controls, like the other aspect_ratios and animation, speed, autoplay, pagination etc...
- Template: "/widgets/header_advanced_widget.html.twig" (The template where it's HTML is defined)
- Icon: "fa-regular fa-pager" (A FontAwesome class for the icon)

Deployment configuration

Please note that this could be an complex topic, documented for our technical colleagues. We do not expect everyone to comprehend this.

Some things are different for each deployment of the Site-builder, think of database, partner, settings, cloudflare. below an overview

Production

- URLs
 - URL UI: <https://cms.otys.app>
 - URL Websites: [brandname].otys.website
 - URL Websites: <https://customdomain.com>
 - URL API For UI: <https://cms.otys.app/api/>
- Partner
 - "OTYS CMS" - <https://otys.otysapp.com/nl/modular.html#/partners/158>
 - Setting to enable cms for an user: PA212
- Database
 - Host: pgdb01.otys
 - DB: otys_websites
 - Type: PostgreSQL (TCP/IP)
- Cloudflare
 - Bucket: otys-cms-public
 - CDN URL: <https://cms.otyscdn.com/>
 - OWS Connection
 - URL: <https://ows.otys.nl>
- WebApi
 - URL: <https://webapi.otys.app/api>

Staging

- URLs
 - URL UI: <https://cms-staging.otys.app/>
 - URL Websites: [brandname].otys.website (With Request Header: X-Otys-Cms-Env: staging)
 - URL Websites: <https://customdomain.com> (With Request Header: X-Otys-Cms-Env: staging)
 - URL API For UI: <https://cms-staging.otys.app/api>
- Partner
 - "OTYS CMS" - <https://otys.otysapp.com/nl/modular.html#/partners/158> (Same as production)
 - Setting to enable cms for an user: PA212 (Same as production)
- Database
 - Host: pgdb01.otys
 - DB: otys_websites_staging
 - Type: PostgreSQL (TCP/IP)
- Cloudflare
 - Bucket: otys-cms-public-staging
 - CDN URL: <https://cms-staging.otyscdn.com/>
 - OWS Connection
 - URL: <https://ows.otys.nl>
- WebApi
 - URL: <https://webapi-staging.otys.app/api>

Testing

- URLs
 - URL UI: <https://cms-testing.otys.app/>
 - URL Websites: [brandname].otys.website (With Request Header: X-Otys-Cms-Env: testing)
 - URL Websites: <https://customdomain.com> (With Request Header: X-Otys-Cms-Env: testing)
 - URL API For UI: <https://cms-testing.otys.app/api>
- Partner
 - "OTYS CMS Testing" - <https://otys.otysapp.com/nl/modular.html#/partners/152>
 - Setting to enable cms for an user: PA211
- Database
 - Host: pgdb01.otys
 - DB: otys_websites_testing
 - Type: PostgreSQL (TCP/IP)
- Cloudflare
 - Bucket: otys-cms-public-testing
 - CDN URL: <https://cms-testing.otyscdn.com/>
 - OWS Connection
 - URL: <https://ows.otys.nl>
- WebApi
 - URL: <https://webapi-test.otys.app/api>

Accessing non-production version of the Site builder

When testers, product managers & developers need to access the application on another environment than production they are able to do so. Below an explanation of how this works & what to do.

As you can see in the deployment configuration above, the database and partners of staging & testing differ from production, this means live website of clients are only on production, and they can't be accessed on testing or staging. You can however create similar websites on testing/staging and test new features or bug-fixes that way.

Only enable the staging or testing CMS on test clients, and never on production clients! If you doubt this statement, make sure to talk to developers first before taking any action.

Always make sure the production CMS is already enabled, and that this client meets all requirements for [Activation](#). You will otherwise run into errors and cause troubles for any future activation.

Accessing websites on staging or testing.

If you go to "brandname.otys.website" our servers will by default serve you the production version of that website. Sometimes this isn't what you want, in that case it's possible to tell our servers you want either the staging/testing version of that website. We do this by adding a custom header to the request called: "X-Otys-Cms-Env". Browser plugins like "[Simple Modify Headers](#)" for Google Chrome help you doing that.

In case you have installed "Simple Modify Headers" you can open it, and configure it to Add a new header "X-Otys-Cms-Env" and with an value of either: "staging" or "testing". Now be sure to click the "Start" button in this plugin, and you are ready to go.



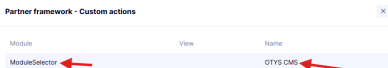
Any new requests to websites will contain this extra header. If our servers detect it, it will serve you either the staging or testing version of our application. However, make sure that you have configured such website with such domain on either staging or testing, otherwise it ofcourse wont show.

Accessing the CMS on staging

If you want to be able to use the Staging version of the CMS for a **TEST-client** where it is not already enabled, you can do this pretty easily by only adding an extra partner action to the navigation menu of this client. Do note, that the production version of the CMS should already be enabled for this client!

Before performing any of the steps below, check if the Staging version is not already activated for the client, you can do this by checking the clients navigation menu for a CMS icon, which is called: "OTYS CMS Staging"

1. Sign in as the desired TEST client with user OTYS Admin
2. Make sure only the Production CMS is enabled right now by checking the navigation menu ☐ OTYS CMS , if there is already a staging button, it was already activated for this client.
3. Open "Client setting" and search for: GE288
4. Open setting GE288
5. Here you will find the production partner action for the cms that is in the navigation menu of this client. You can identify it, by the Name & Module



6. Open this action, to reference it's contents, you will need to almost create an identical copy of it.

OTYS CMS

Name

OTYS CMS

Module

Module selector

Partner

OTYS CMS

Target

blank

Type

URL

URL

https://cms.otys.app/api/auth

Endpoint visibility check

https://cms.otys.app/api/auth/visibility-check-navigation


Post session token

☒

SSO Method

Post loginData

Image



Cancel

Delete

Save

7. Close the original, and click: "Add new" to create a new navigation button, for the staging CMS.

Fill in the navigation item like below, making sure the call it: "OTYS CMS Staging", and making sure to leave out the visibility check. Because this is always performed in an test client, we don't need the visibility check, and any user may see the icon.

OTYS CMS Staging

Name

OTYS CMS Staging

Module

Module selector

Partner

OTYS CMS

Target

blank

Type

URL

URL

https://cms-staging.otys.app/api/auth

Endpoint visibility check

☒


Post session token

☒

SSO Method

Post loginData

Image

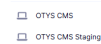


Cancel

Delete

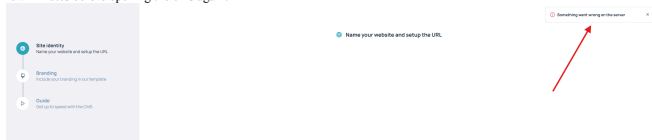
Save

8. Now refresh OTYS Go!, and see that your new navigation item has appeared



9. Now when you click this button, the activation of the Site-builder is triggered on staging. This means it will run lot's of scripts, one of which is creating the WebApi key trough MyOTYS.

This new API Key is not instantly ready for use, as the connected user is to "new" to be used right away, and you will have to wait for at least a few minutes before it will be usable. Your interface of the CMS most probably will throw some errors, just wait a few minutes before opening the cms again.



10. And that's it, you have enabled the CMS for the test client, and you can start creating a new website and try out features from staging.

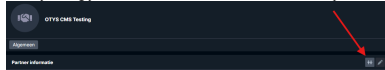
Accessing the CMS on testing

If you want to be able to use the Testing version of the CMS for a **TEST-client** where it is not already enabled, you can do this by configuring the partner, partner-setting & adding an extra partner action to the navigation menu of this client.

Do note, that the production version of the CMS should already be enabled for this client!

Before performing any of the steps below, check if the Testing version is not already activated for the client, you can do this by checking the clients navigation menu for a CMS icon, which is called: "OTYS CMS Testing"

1. Start by adding your TEST client to the activated clients of the partner "OTYS CMS Testing" - <https://otys.otysapp.com/nl/modular.html#/partners/152>



2. Sign in as the desired TEST client with user OTYS Admin

3. Make sure the Production CMS is enabled right now by checking the navigation menu  OTYS CMS , if there is already a testing button, it was already activated for this client.

4. Now go to user settings, and search for PA211


5. Enable this setting for all users you want to be able to use the CMS

6. Now go to client settings and search for: GE288

7. Open setting GE288

8. Here you will find the production partner action for the cms that is in the navigation menu of this client.

You can identify it, by the Name & Module

Partner framework - Custom actions		
Module	View	Name
ModuleSelector		OTYS CMS

9. Open this action, to reference it's contents, you will need to almost create an identical copy of it.

OTYS CMS

Name

OTYS CMS

Module

Module selector

Partner

OTYS CMS

Target

blank

Type

URL

URL

https://cms.otys.app/api/auth

Endpoint visibility check

https://cms.otys.app/api/auth/visibility-check-navigation


Post session token

☒

SSO Method

Post loginData

Image



Cancel

Delete

Save

10. Close the original, and click: "Add new" to create a new navigation button, for the staging CMS.

Fill in the navigation item like below, making sure the call it: "OTYS CMS Testing", and making sure to leave out the visibility check. Because this is always performed in an test client, we don't need the visibility check, and any user may see the icon.

OTYS CMS Testing

Name

OTYS CMS Testing

Module

Module selector

Partner

OTYS CMS Testing

Target

Nieuw venster

Type

URL

URL

https://cms-testing.otys.app/api/auth

Endpoint visibility check

☒


Post session token

☒

SSO Method

Post loginData

Image



Aanmaken

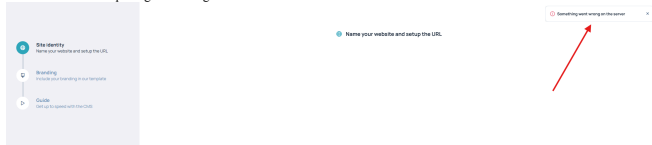
Verwijderen

Bewaren

11. Now refresh OTYS Go!, and see that your new navigation item has appeared

- OTYS CMS
- OTYS CMS Testing

12. Now when you click this button, the activation of the Site-builder is triggered on staging. This means it will run lot's of scripts, one of which is creating the WebApi key trough MyOTYS. This new API Key is not instantly ready for use, as the connected user is to "new" to be used right away, and you will have to wait for at least a few minutes before it will be usable. Your interface of the CMS most probably will throw some errors, just wait a few minutes before opening the cms again.



13. And that's it, you have enabled the CMS for the test client, and you can start creating a new website and try out features from testing.
