Roadiz Documentation

Version 2.1.0

Ambroise Maupate, Julien Blanchet

mars 06, 2023

Table des matières

Roadiz is a polymorphic CMS based on a node system that can handle many types of services. It is based on *Symfony* framework, *Doctrine ORM*, *API Platform* and *Twig* for maximum performances and security.

Le système de nœuds de Roadiz vous permet de créer une architecture de données sur-mesure et d'organiser vos contenus comme bon vous semble. Nous l'avons conçu pour supprimer les contraintes techniques présentes lors de la création d'arborescences et de mises en page de sites Web.

Imaginons que vous souhaitiez présenter votre portfolio et également… vendre des t-shirts. Avec Roadiz, vous serez en mesure de créer vos formulaires de contenu à partir de zéro et de choisir uniquement les champs dont vous avez besoin. Images et textes pour vos projets, tarifs et même géolocalisation pour vos produits. C'est en cela qu'il est *polymorphe*.

Note : This is documentation for Roadiz v2.x, browse v1.x documentation

---

CHAPITRE 1

Philosophie

When discovering Roadiz back-office interface, you will notice that there aren't any Rich text editor also called *WYSIWYG* editors. We chose to promote *Markdown* syntax in order to focus on content hierarchy and quality instead of content style. Our guideline is to preserve and respect the web-designers and graphic designers work.

You'll see that we built Roadiz as web-designers and for web-designers. It will allow you to create really quickly website prototypes using *Twig* templates or develop complex headless websites with the power of *API Platform*.

Nous avons pensé Roadiz comme un outil destiné à faire travailler concepteurs et développeurs ensemble, au sein d'expériences webs uniques. Et nous n'avons pas oublié les éditeurs : le thème back-office « Rozier » de Roadiz a été conçu pour offrir à tous les utilisateurs une meilleure expérience d'écriture et d'administration de leurs contenus.

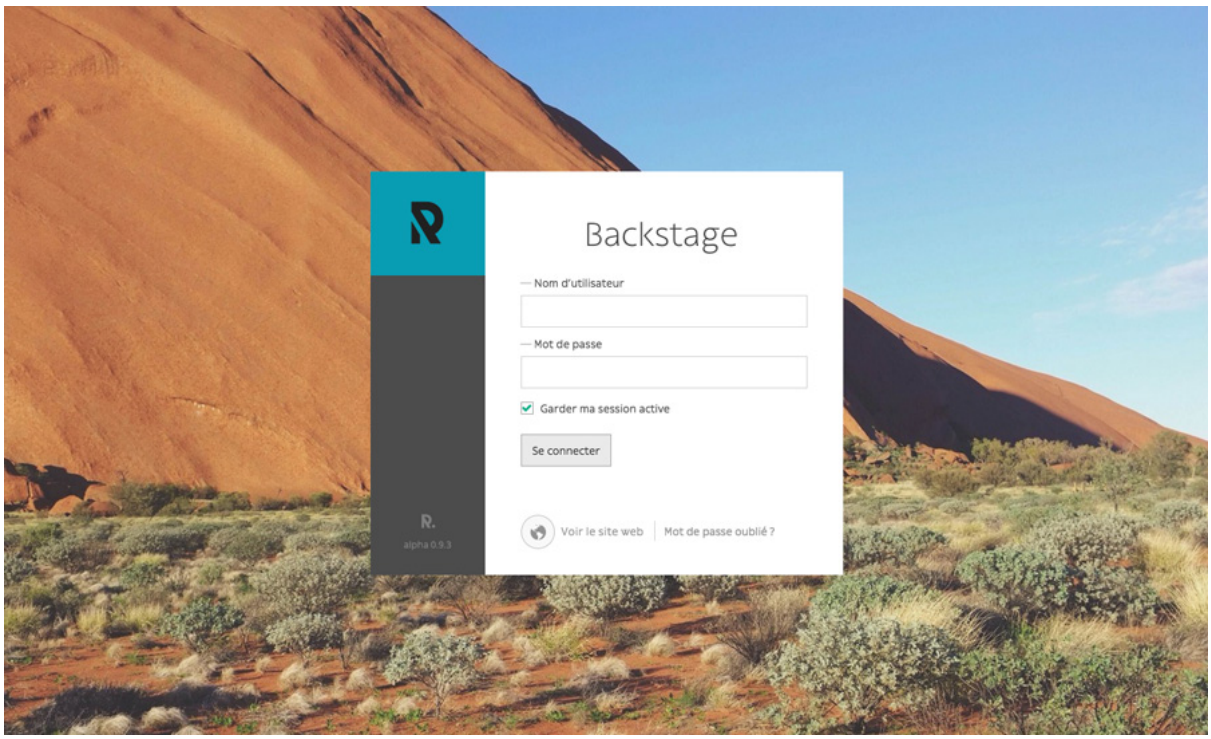
CHAPITRE 2

Utilisateur - Documentation

2.1 Utilisateur - Documentation

Note : La documentation utilisateur est en cours de rédaction. Nous vous invitons à envoyer vos questions sur notre compte Gitter ou bien de nous faire part de vos suggestions sur le dépôt Github.

Tout d'abord, vous allez avoir besoin de vous connecter au back-office de Roadiz pour effectuer n'importe quel changement sur votre site-web. Pour vous connecter, il vous suffit de taper `/rz-admin` à la suite du nom de domaine de votre site, dans la barre d'adresse de votre navigateur. Ensuite, un formulaire vous invitera à entrer votre nom d'utilisateur et votre mot de passe. Ceux-ci vous ont été transmis par email, ou bien vous les avez créé si vous avez installé Roadiz vous-même.



Ici vous pouvez choisir de garder votre connexion active pour quelques jours si vous n'avez pas bloqué les cookies de votre navigateur.

Si vous avez oublié vos identifiants, la section *Mot de passe oublié ?* vous permettra de le charger après avoir suivi un lien temporaire envoyé à votre adresse email.

Table des matières

2.1.1 Écrire en Markdown

> Markdown est un langage de balisage léger avec une syntaxe en texte brut conçue pour être facilement traduite en HTML et bien d'autres formats. [...] Le but de Markdown est de rester lisible tel quel sans donner l'impression qu'il a été marqué par des balises ou des instructions de formatage, contrairement au Rich Text Format (RTF) ou HTML qui utilisent des balises et instructions de formatage empêchant leur lecture par un(e) non-spécialiste.
>
> —Article sur Wikipédia — https://fr.wikipedia.org/wiki/Markdown

Titres

Ajouter deux dièses # ou plus en fonction de l'importance du titre souhaitée. Les boutons du back-office permettent d'insérer directement les dièses avant le texte sélectionné. Veuillez à bien laisser une ligne vide avant chaque nouveau titre.

```
## Architecture
### Modern architecture
```

Attention à ne pas utiliser une seule dièse pour créer un titre de niveau 1, car il est généralement réservé au titre principal de votre page.

Syntaxe alternative

Les titres principaux et secondaires peuvent être écrits en les soulignant avec les caractères = et −.

```
Architecture
============

Modern architecture
-------------------
```

Gras

Insérer deux étoiles * au début et à la fin du texte pour le mettre en gras. Le bouton du back-office permet d'insérer directement les 4 étoiles autour du texte sélectionné.

```
This is a **bold text.** And a normal one.
```

Attention à ne pas laisser d'espace à l'intérieur du groupe d'étoiles (de la même manière qu'avec les parenthèses) sinon le formatage ne s'appliquera pas.

Italique

Insérer une étoile * au début et à la fin du texte pour le mettre en italique. Le bouton du back-office permet d'insérer directement les 2 étoiles autour du texte sélectionné.

```
This is an *italic text.* And a normal one.
```

Les syntaxes gras et italiques peuvent bien sûr être combinées à l'aide de trois étoiles au début et à la fin du texte sélectionné.

Et si le caractère * est déjà utilisé dans mon texte ?

Les gras et italique peuvent être formatés de la même manière avec le caractère tiret-bas _ si votre texte contient le caractère *étoile*.

```
A __3* Bed & Breakfast__ has just opened its doors in middletown.
```

Barré

Insérer deux tildes ~ au début et à la fin de votre texte pour le barrer.

```
This is ~~striked-through text~~.
```

Listes ordonnées et non-ordonnées

Insérer une étoile * ou un tiret - suivi d'une espace et ce, pour chaque élément de la liste. Un élément par ligne. Laisser une ligne vide **avant et après** la liste. Pour les listes *ordonnées*, utiliser un chiffre suivi d'un point et d'une espace.

```
* A line
- An other line
* A unknown line

1. The first item
2. The second item
3. The third item
```

Si vous avez besoin de retourner à la ligne au sein d'un seul élément, vous devrez utiliser la syntaxe du retour-chariot.

Liste imbriquée

Vous pouvez insérer un deuxième/troisième/... niveau à votre liste en **laissant quatre espaces avant chaque nouvel élément de liste.**

```
- A list item
    - A sub-item
    - A second sub-item
        1. An ordered sub-sub-item
        2. The second sub-sub-item
```

Nouveau paragraphe et retour-chariot

Un simple retour à la ligne est toujours ignoré par Markdown car il fait la différence entre un **paragraphe** et un **retour-chariot** (retour à la ligne). Pour créer simplement un retour à la ligne forcé sans avoir à changer de paragraphe, **laisser au minimum 3 espaces à la fin de votre ligne de texte et aller à la ligne.**

```
Address:<space><space><space>
26 rue Burdeau<space><space><space>
69001 Lyon<space><space><space>
France
```

Pour créer un nouveau paragraphe, **laisser toujours une ligne vide entre vos blocs de texte.** Tout ligne vide en plus sera ignorée.

```
Nullam quis risus eget urna mollis ornare vel eu leo.
Cras justo odio, dapibus ac facilisis in, egestas eget quam.

Aenean eu leo quam. Pellentesque ornare sem lacinia
quam venenatis vestibulum.
```

En fonction du design (CSS) de votre site, les paragraphes peuvent ne posséder aucune marge visuelle entre eux. Insérer plus d'une ligne vide n'ajoutera pas plus d'espace car Markdown ignore les espaces et lignes superflues.

Liens hypertextes

Écrire le label du lien entre crochets suivis immédiatement de l'URL entre parenthèses. Pour les liens externes, attention à ne pas oublier le préfixe du protocole `http://` ou `https://`.

```
[My link](http://www.google.com)
```

Pour un lien interne, il faut utiliser la notation relative :

```
[Contact us](/page/contact-us)
```

Pour un lien d'email, préfixer l'URL avec `mailto:` :

```
[John Doe](mailto:jdoe@example.com)
```

Un titre de lien peut être ajouter en l'insérant avant la parenthèse fermante, entouré de guillemets.

```
[My link](http://www.google.com "Link to Google website")
```

Citations

Insérer le signe > **et une espace avant chaque nouveau paragraphe** pour inclure votre texte dans une citation. Vous pourrez alors utiliser les autres symboles Markdown à l'intérieur de votre citation.

```
> ### Donec ullamcorper nulla non metus auctor fringilla.
> Aenean lacinia **bibendum** nulla sed consectetur.
> Vestibulum id ligula porta felis euismod semper.
```

Images

Les images utilisent la même syntaxe que les liens en étant préfixées par un point d'exclamation `!`. Pour les images externes, n'oubliez pas d'écrire l'URL complet avec protocole `http://` ou `https://`.

```
![A cat](/files/cat.jpg)

![A cat from an other website](https://www.example.com/images/cat.jpg)
```

Attention, les images seront affichées à leur dimension et qualité natives, à moins que votre webdesigner n'ait prévu d'adapter les images provenant d'un champ Markdown via CSS. Comme pour les liens, les images externes peuvent casser si leur propriétaire supprime la ressource originale. Faites attention à toujours héberger vous-même les images importantes de votre site-web et d'utiliser des *liens relatifs*.

Notes de bas de page

Les notes de bas de page ne sont pas prises en charge par la version basique de Markdown mais par la version *MarkdownExtra*. Ainsi, **vérifiez que votre web-designer a bien utilisé le bon système Markdown dans votre thème.**

```
Praesent commodo cursus magna[^note], Sed posuere consectetur est at
lobortis. Vel scelerisque nisl consectetur et[^othernote].

[^note]: This a footnote
[^othernote]: This a second footnote
```

Markdown *générera automatiquement les liens d'ancre entre vos notes de bas de page et leur références.* Il **utilisera automatiquement des numéros pour les appels de notes**, vous n'avez du coup pas besoin de vous embêter à les numéroter manuellement, juste à les nommer de manière à bien les repérer.

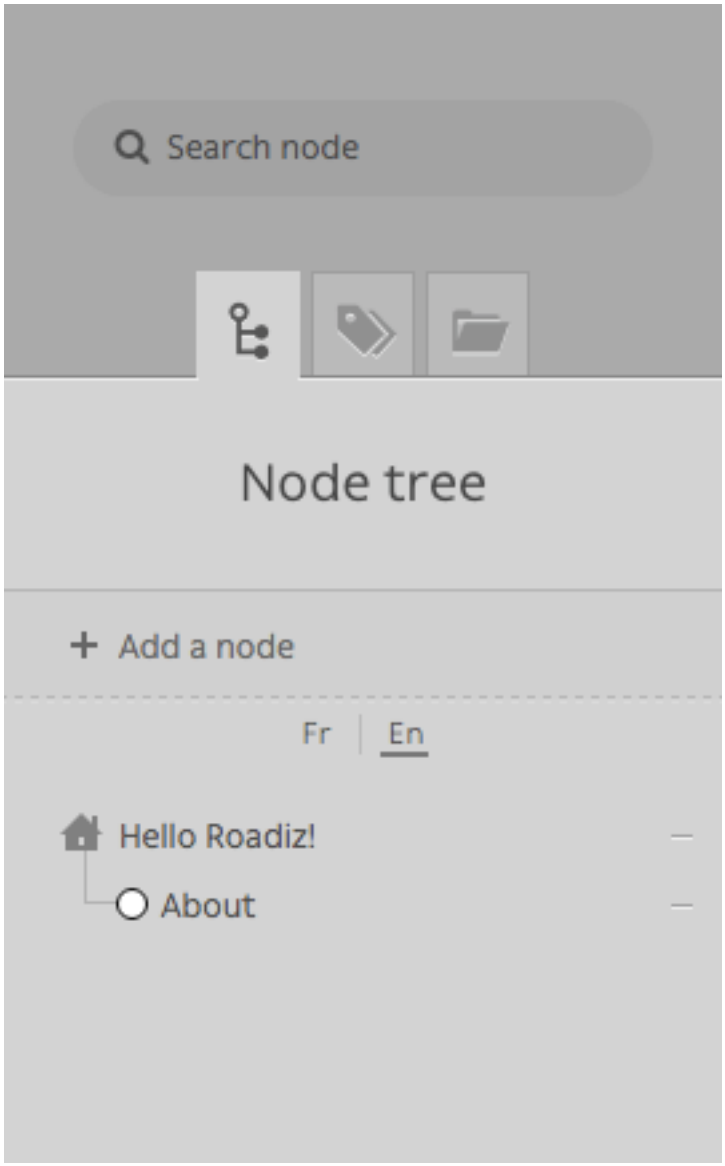
2.1.2 Gérer les nœuds

Les nœuds sont l'élément le plus important du CMS Roadiz. Ils permettent la mise en forme de votre contenu telle que vous le souhaitez en fonction de la définition de vos types de nœuds. Un nœud peut être un post de blog, une page de contenu, une galerie photo, ou même un produit de l'e-boutique. Voilà pourquoi nous l'avons appelé « Nœud », c'est une unité de données abstraite et interconnectée avec le reste de votre arborescence.

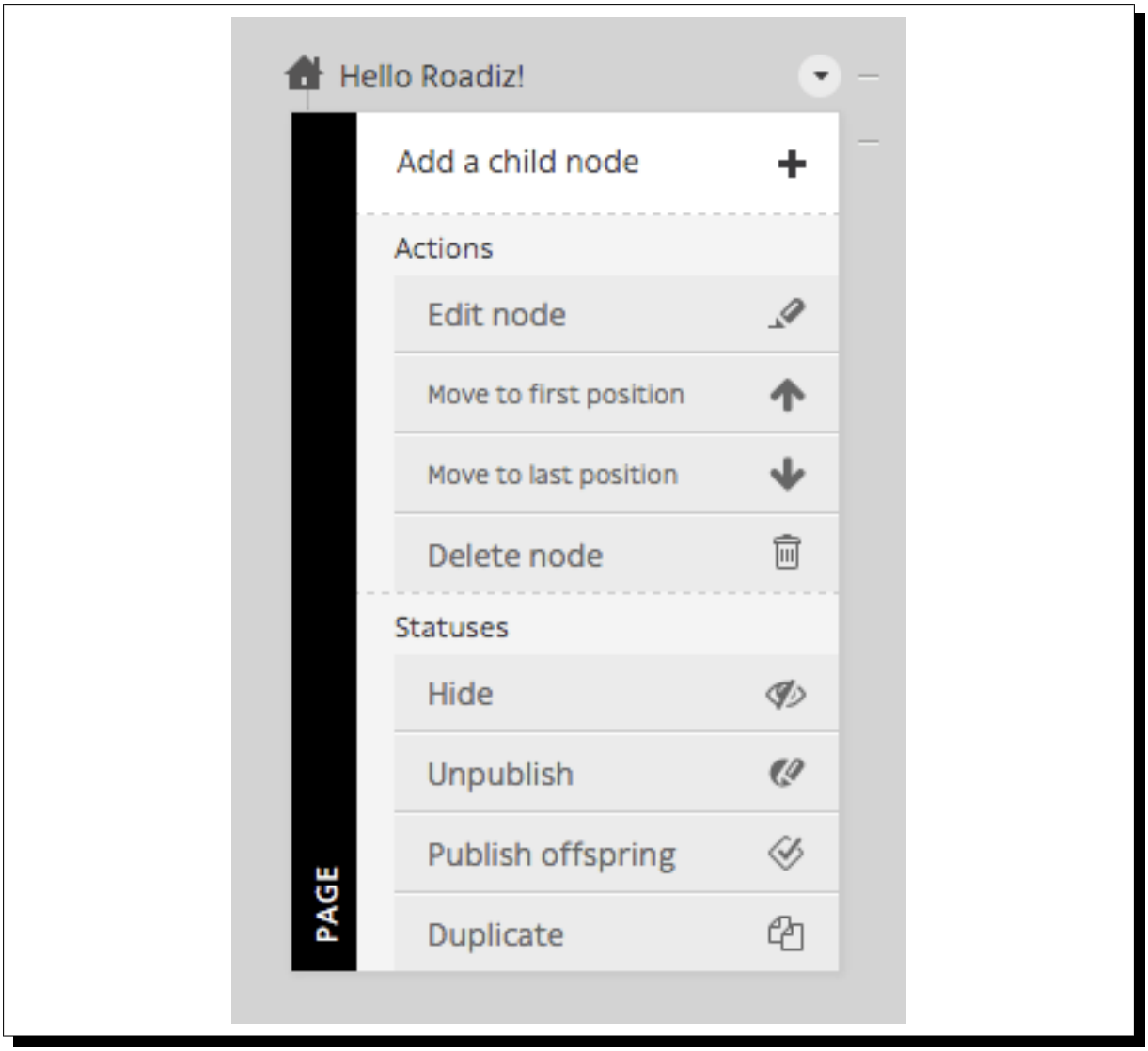
Navigation dans l'arborescence

Node tree

Arborescence simple après installation de Roadiz avec le *Default theme*.



Menu contextuel pour un nœud *Accueil*.



Chaque nœud a sa place dans votre site et c'est pourquoi nous avons choisi d'organiser votre contenu à travers une arborescence. C'est intuitif et cela fonctionne de la même manière que la gestion des fichiers de votre ordinateur.

— Pour éditer le contenu d'un nœud, cliquez simplement dessus.
— Pour déplacer un nœud dans votre arborescence, réalisez un « cliquer-déposer » grâce à la poignée (en forme de rond ou de losange). Vous pouvez déposer un nœud avant ou après un autre élément. Vous pouvez également le déposer à l'intérieur d'un autre nœud, en décalant légèrement votre souris vers la droite de ce dernier pour déplacer l'ombre du nœud à l'intérieur.
— Other actions are available from each node contextual menu. Right click on the node or click on the arrow at the right when you pass your mouse over.

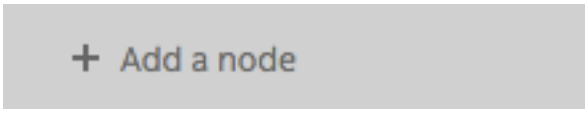
Actions des menus contextuels

— *Ajouter un nœud enfant* : créer une zone de contenu à l'intérieur d'un nœud existant.
— *Éditer* : renvoie à la page d'édition de contenu du nœud concerné.
— *Déplacer en première position* : déplacer un nœud à la première position au sein de l'arborescence du nœud parent.
— *Déplacer en dernière position* : déplacer en dernière position de l'arborescence du parent.
— *Supprimer* : placera le nœud actuel dans la corbeille. Une fenêtre de confirmation s'ouvrira afin de supprimer un nœud. Le nœud n'est pas supprimé définitivement, il se retrouve dans la corbeille.
— *Cacher/Afficher* : Changer la visibilité d'un nœud. Un nœud caché ne sera pas indexé dans votre site et n'aura pas d'url, même si vous êtes connecté en tant qu'administrateur.

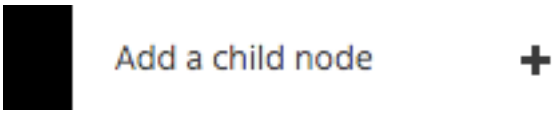
— *Publish/Unpublish :* Change a node' publication status. Unpublished nodes are not visible to anonymous visitors, but visible for back-office users using `?_preview=1` query parameter.
— *Publier la descendance* : Publier un nœud et tous ses noeuds-enfants rattachés.
— *Dupliquer* : Copier l'intégralité du contenu et des interactions du nœud actuel dans un nouveau nœud.

Création d'un nœud

Add buttons



Le bouton *Ajouter un nœud* est situé en haut de votre *arborescence*.



Le bouton « Ajouter un nœud *enfant* » est situé en haut de chaque menu contextuel d'un nœud.

Pour ajouter un nœud vide à votre arborescence, vous devrez choisir son emplacement. À l'intérieur de Roadiz, vous pouvez ajouter un contenu à la racine de votre arbre ou choisir un « nœud-parent ». Dans les deux cas, vous serez invité à choisir un *type* et un *nom* avant de créer votre nœud.

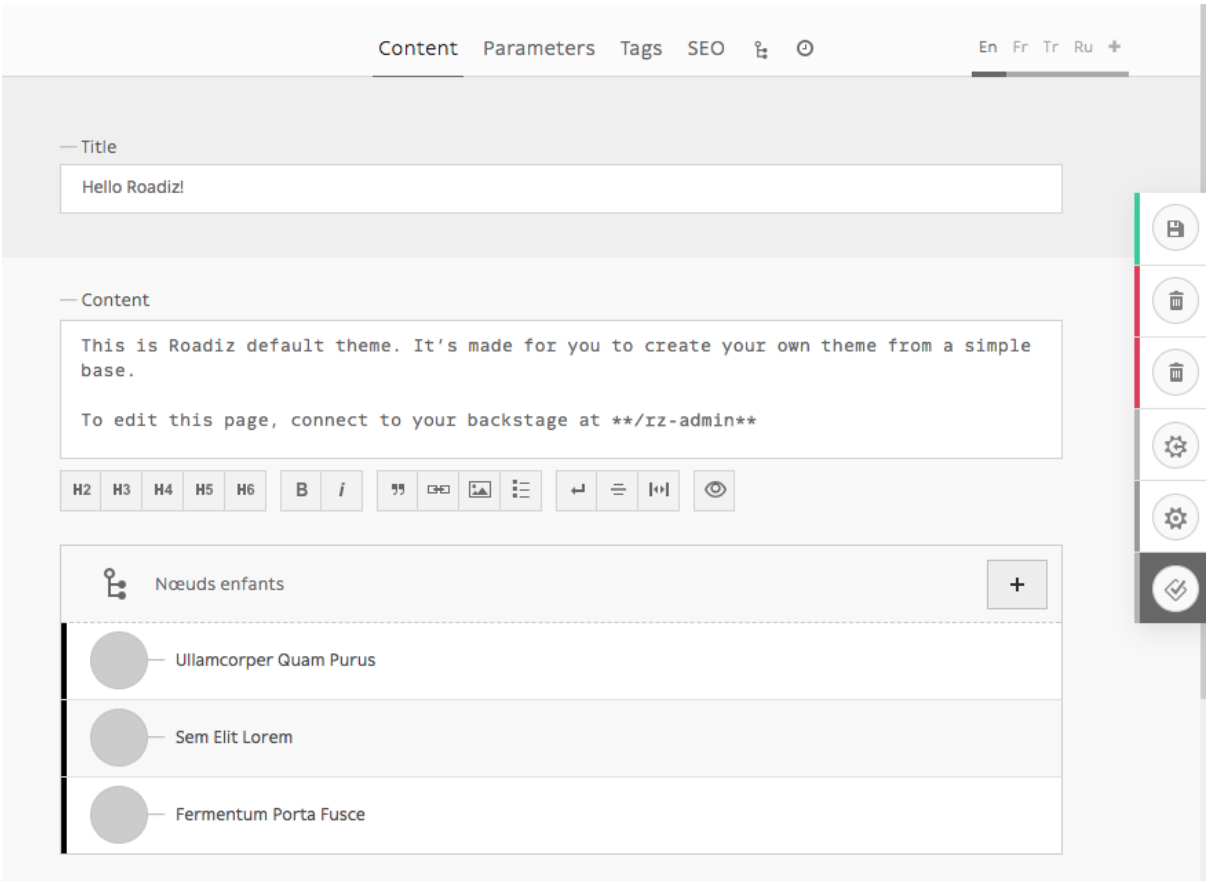
— Le *Nom du noeud* est l'identifiant global de votre nœud. Il doit être unique et ne changera pas d'une traduction à un autre. Il peut être modifié ultérieurement, sauf si votre développeur a verrouillé sa modification. Le *Nom du nœud* est utilisé pour construire les URL de vos pages, de manière générale.
— Le *Type de nœud* définit les champs disponibles de votre contenu. Choisissez le bien car la modification ne sera pas possible ultérieurement, il vous faudra supprimer le nœud et en créer un nouveau en cas d'erreur.

Éditer un nœud existant

La page d'édition du nœud est composée de plusieurs onglets.

— *Contenus du nœud*
— *Paramètres du nœud*
— *Étiquettes*
— *Référencement*
— *Arborescence*, si votre nœud est défini comme *conteneur (ou pile)*

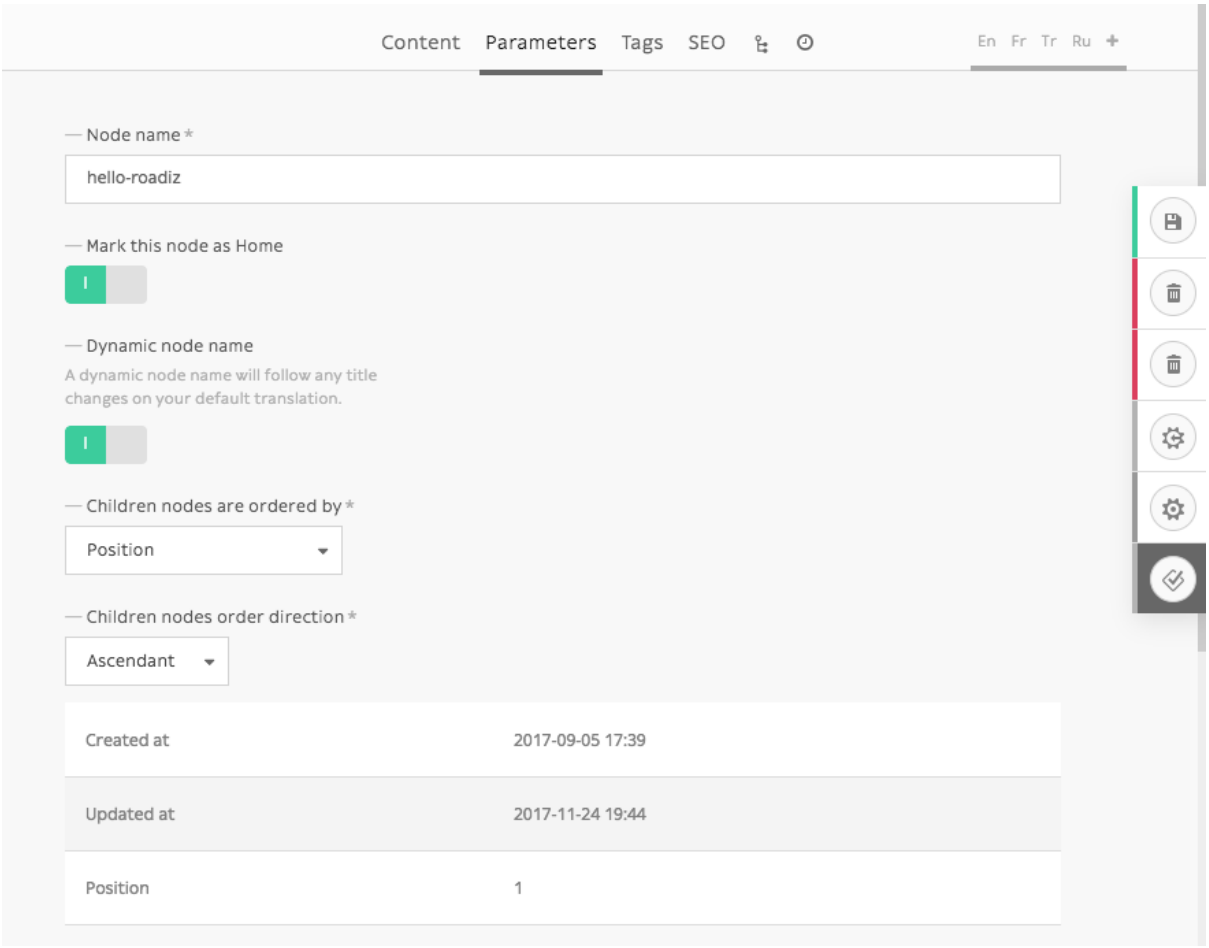
Contenu d'un nœud



L'onglet contenu vous permettra de modifier les données spécifiques de votre nœud, en utilisant des champs-types tels que *texte* ou *documents*, etc.

Cet onglet affichera les différents contenus traduits du nœud en fonction des champs marqués *Universel* ou non.

Paramètres du nœud

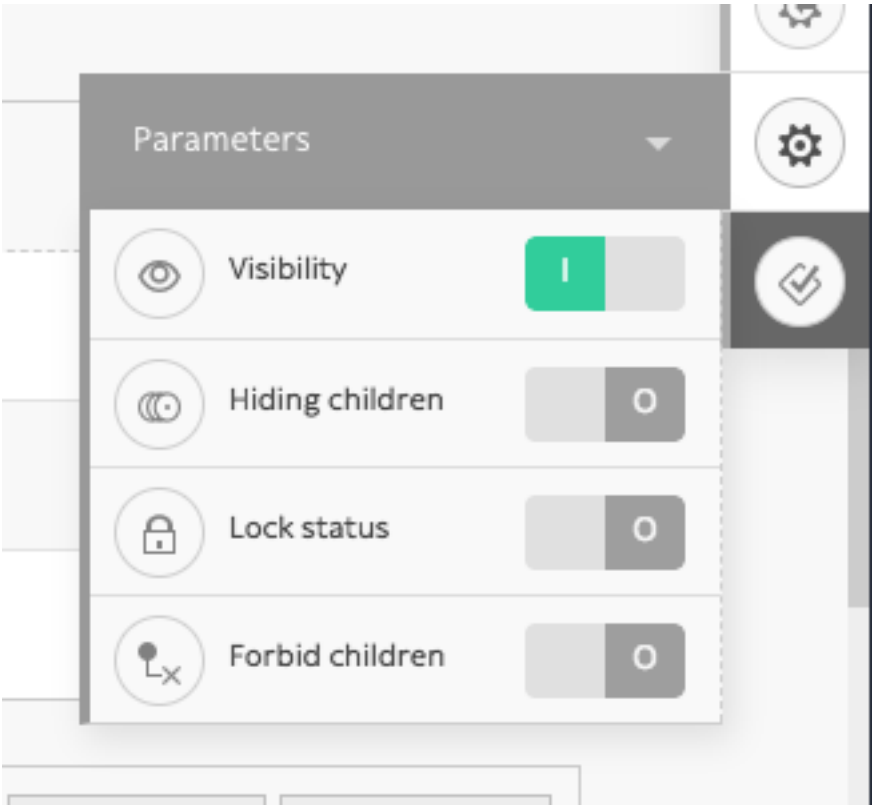


Les paramètres sont des données globales telles que votre *nom de nœud*. Ils sont utilisés pour gérer la visibilité de votre nœud en fonction des rôles de chaque utilisateur et des paramètres de chaque nœud. Cette section ne doit pas être utilisée de façon régulière puisque les paramètres sont fixés par votre développeur en amont pour correspondre à votre besoin.

Cet onglet affichera le même contenu sans importance de traduction.

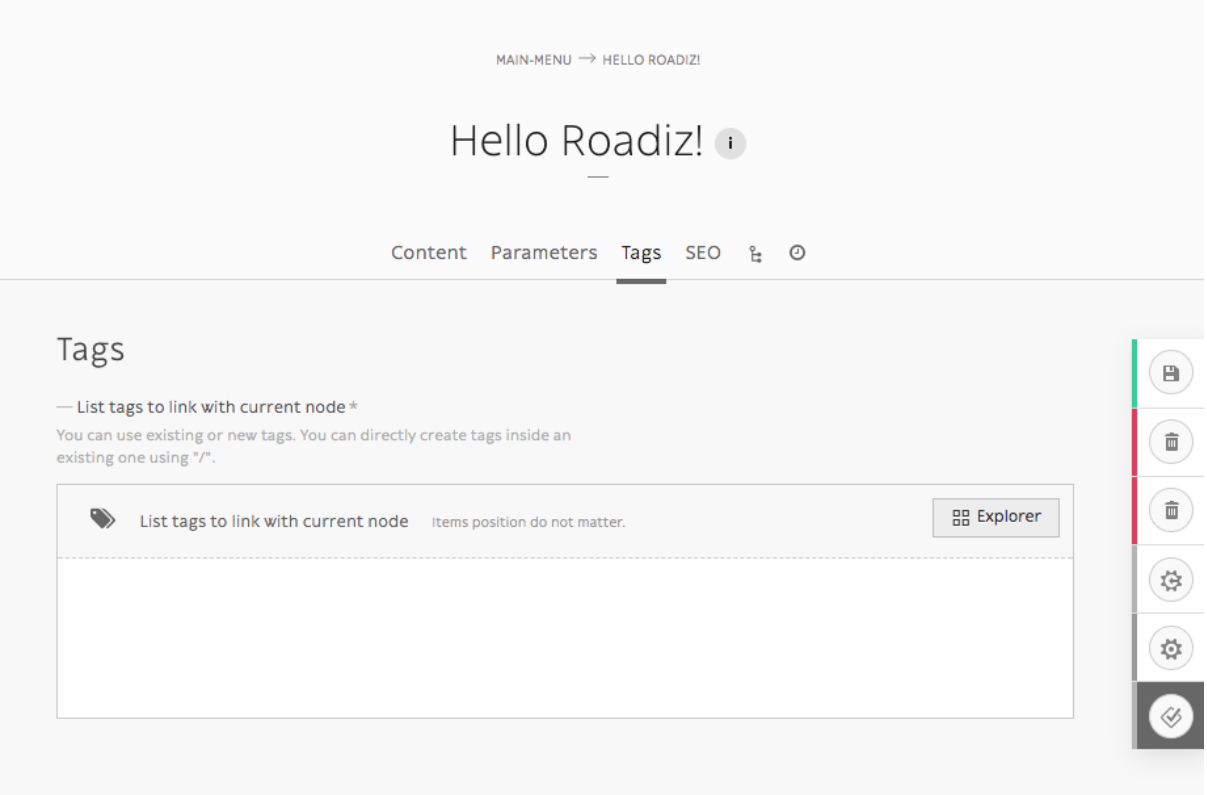
Actions et paramètres secondaires

Des paramètres supplémentaires sont disponibles sur le menu d'action à droite de votre écran. Ces paramètres s'applique au niveau du nœud et donc ils seront identiques pour chaque traduction.



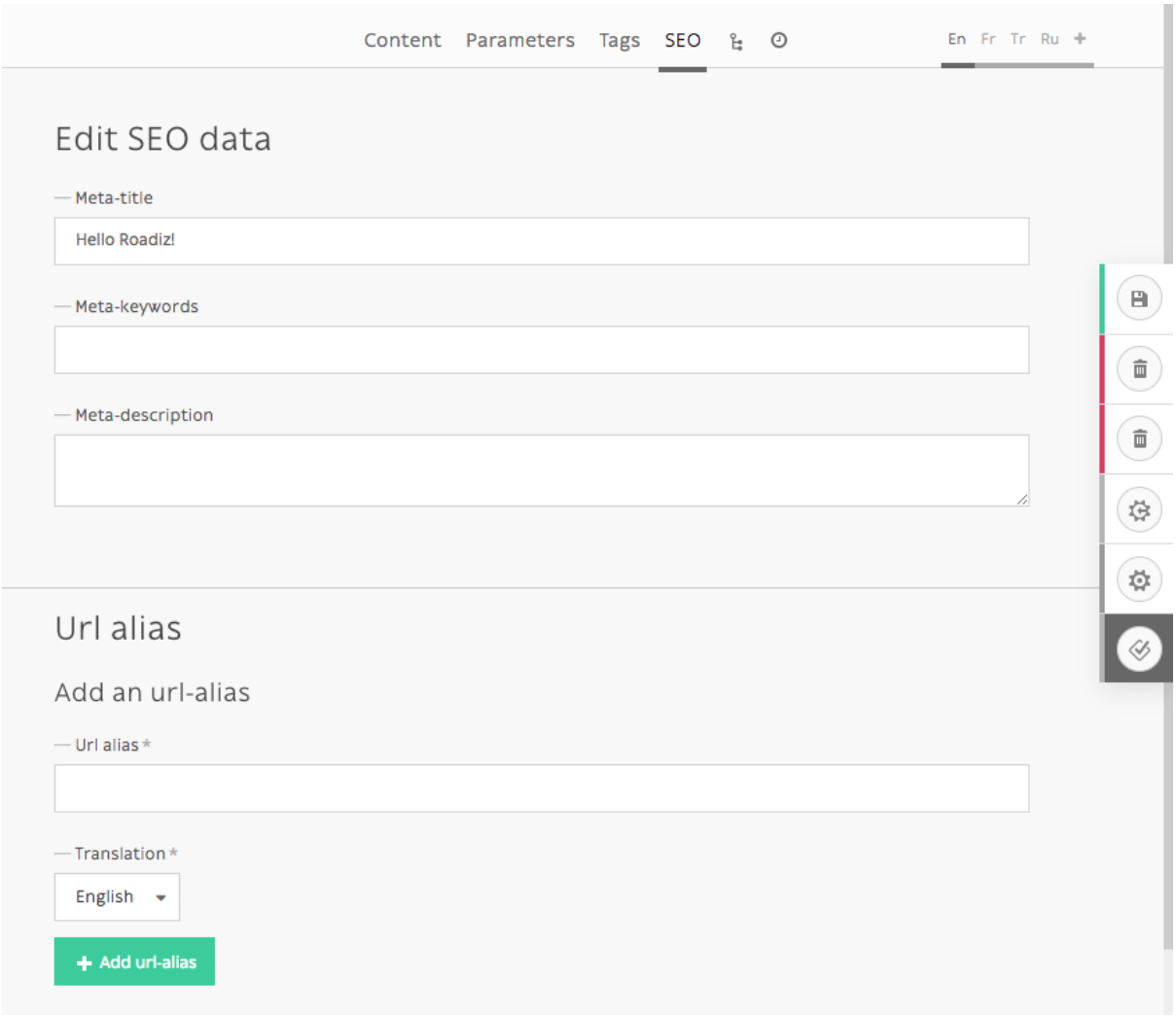
— **Visibilité** : Cache ou affiche le nœud actuel (en fonction du développement de votre thème)
— **Cacher les enfants** : Transforme le nœud actuel en **conteneur (ou pile)**, les nœuds enfants n'apparaitront plus dans le panneau global d'arborescence. Vous pourrez ajouter des boutons d'ajout de *types de pile* dans *l'onglet des paramètres du nœud*.
— **Verrouillage** : Empêche les utilisateurs de supprimer le nœud actuel ou de le renommer. Vous devriez activer ce mode si le nœud actuel est requis dans la logique de votre thème.
— **Nœud stéril** : Empêche les utilisateurs de créer des nœuds enfants.

Étiquettes



Cet onglet affichera le même contenu sans importance de traduction.

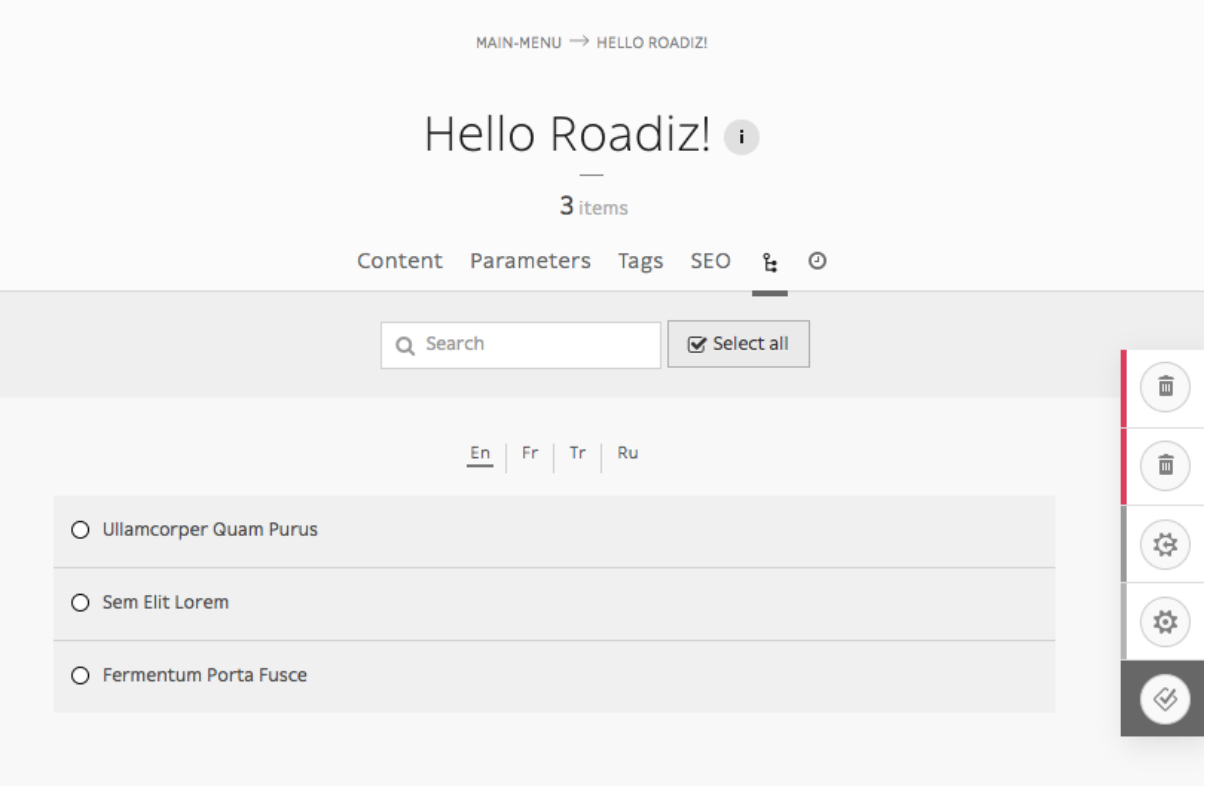
Référencement



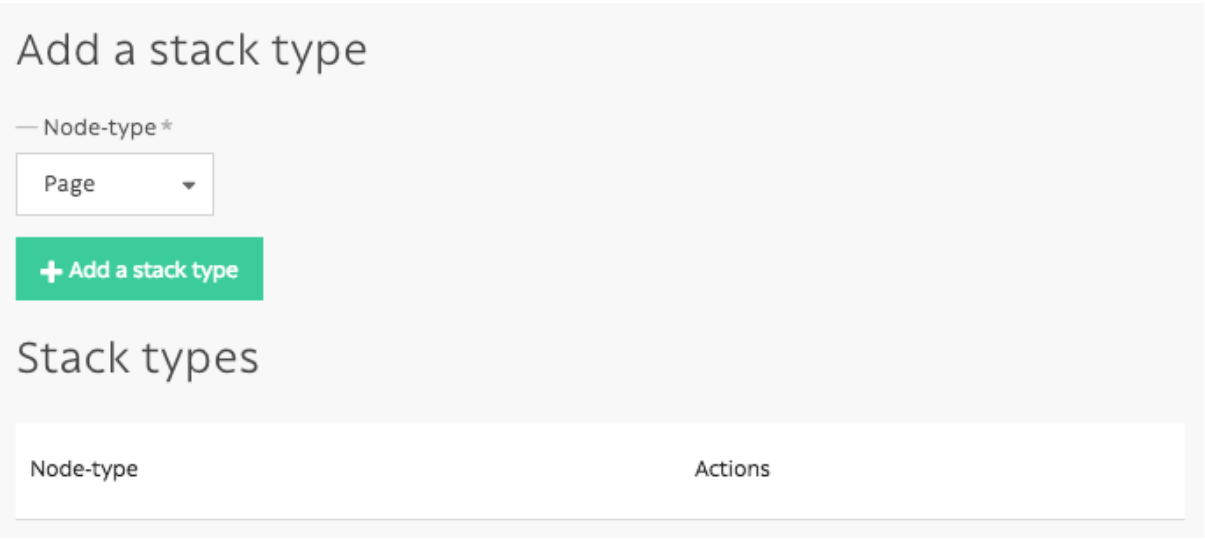
Cet onglet affichera les contenus en fonction de chaque traduction.

L'arborescence

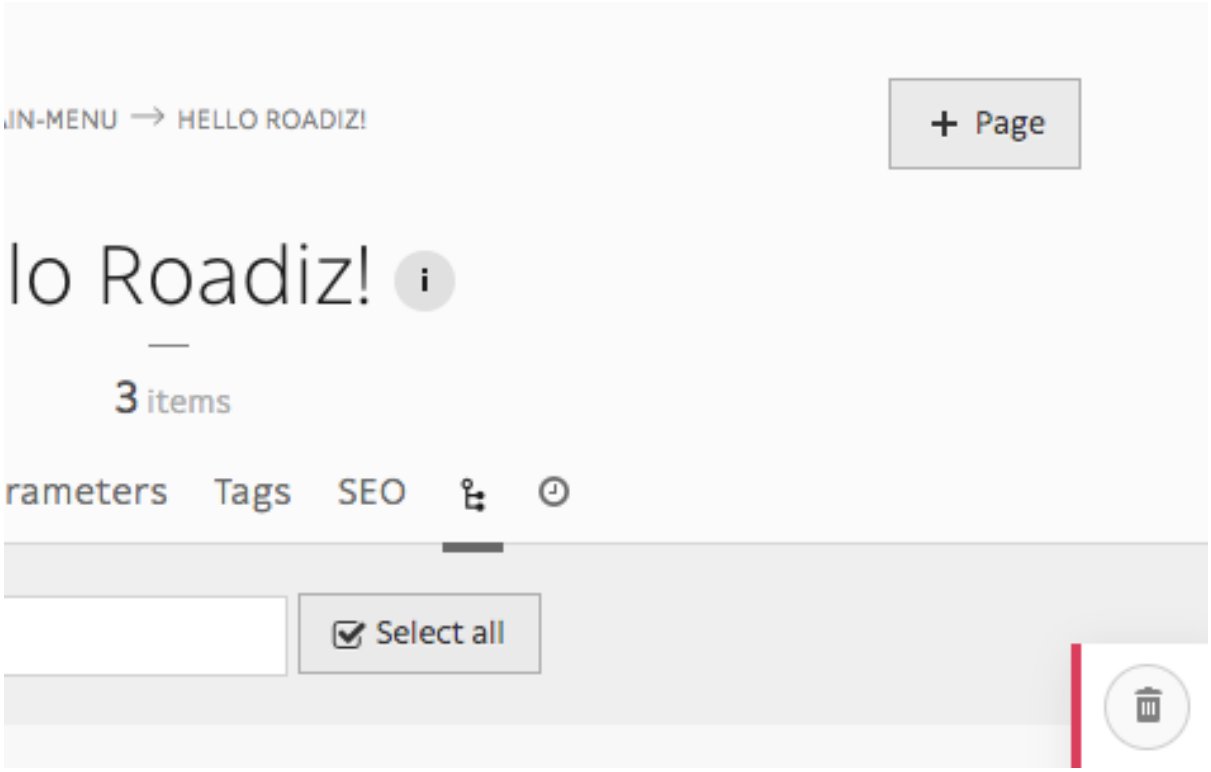
Quand un nœud est défini en tant que *conteneur (ou pile)*, son *arborescence* devient la vue (onglet) par défaut. Vous pouvez définir l'ordre par défaut d'affichage des nœuds enfants dans l'onglet *Paramètres*. De plus, si vous avez défini le *type de nœud* comme étant *publiable*, chaque nœud affichera sa date de publication avant son *titre*.



La vue de l'arborescence devient très intéressante si vous possédez un très grand nombre de nœuds enfants. C'est le cas, par exemple, si vous construisez un site listant des articles de blog ou des projets pour un portfolio.



Enfin, vous pouvez ajouter un *type de pile* dans l'onglet *Paramètres* pour créer des boutons d'ajout de nœud bien pratique dans l'onglet *Arborescence*.



Système de publication des nœuds

Pendant son cycle de vie, chaque nœud peut avoir différents statuts de publication. Lorsque vous créez un nouveau contenu, il sera automatiquement publié comme **Brouillon** par Roadiz afin de vous permettre de le modifier sans incidence sur vos visiteurs et sans rendre public un contenu en cours de réalisation.

Statuts disponibles :

- **Brouillon** : statut initial pour chaque nouveau nœud
- **En attente de validation** : un statut intermédiaire disponible pour les utilisateurs n'ayant pas les droits de publication
- **Publié** : il s'agit du statut le plus important, il rend votre contenu public aux visiteurs de votre site
- **Archivé** : lorsque vous ne souhaitez pas publier un nœud mais ne voulez pas non plus le supprimer de votre interface
- **Supprimé** : il s'agit du dernière statut disponible pour vos nœuds. Avant de vider votre corbeille, chaque nœud sera affiché avec cette mention.

Pour améliorer la visibilité des statuts, les nœuds au stade de *brouillon* et *en attente de validation* sont présentés par une forme de losange alors que les nœuds *publiés* ont une forme circulaire.

Prévisualiser les nœuds en attente de validation

Unpublished nodes are not viewable for anonymous visitors, Roadiz allows backend users to preview them adding a dedicated *query parameter* called `?_preview=1` in your website URLs. Using a different URL than your public website is very important as it could lead to errors or to expose unpublished content if your website is hosted behind a reverse proxy cache like *Varnish*.

For example, if your `my-news` page is not published yet, connecting to `http://mywebsite.com/my-news` will lead to a 404 page for your anonymous visitors, as well as you too. If you want to preview it, you'll have to connect to `http://mywebsite.com/my-news?_preview=1`. This URL will only allow authenticated backend users, other people will be denied.

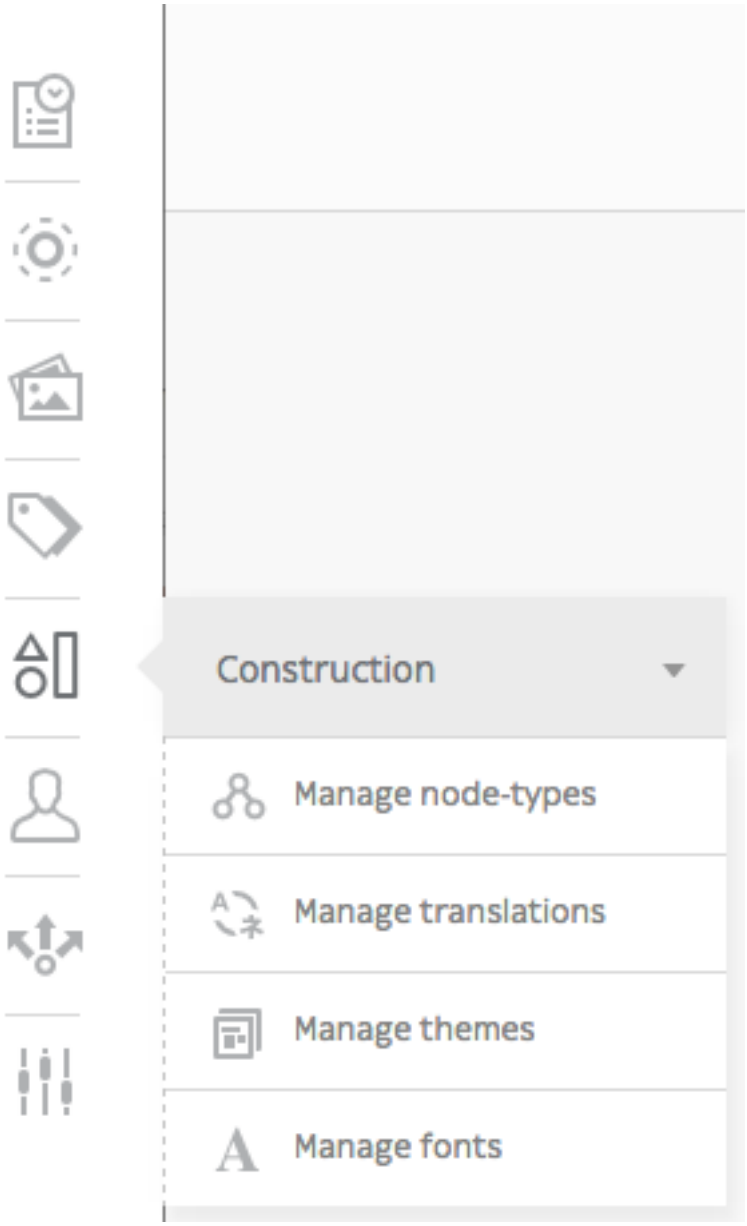
2.1.3 Gérer les types de nœuds

Ce guide vous permettra de comprendre comment créer et gérer des de nœuds en utilisant Roadiz, ajouter et supprimer des champs ou encore comment importer des nœuds.

Avant toute création de nœud, vous devez créer un nouveau « type » de nœud.

Si vous souhaitez en savoir plus sur les types de nœuds, merci de vous rendre à la section afférente de la documentation.

Lorsque vous travaillez dans le back-office de Roadiz, vous pouvez facilement gérer les types de nœuds via l'icône *Construction* de la barre d'outils.

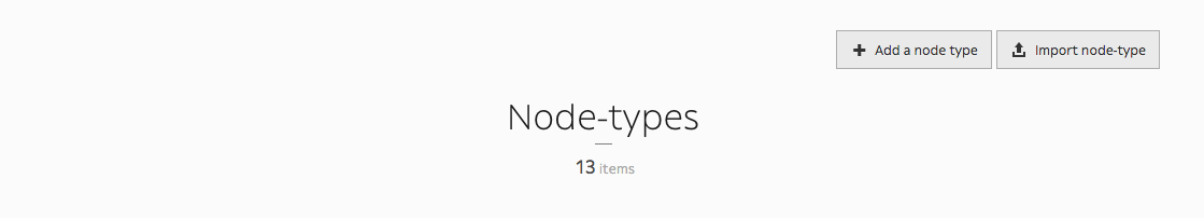


Ajouter un type de nœud

Une fois sur la page Types de nœuds (*https : //mywebsite.com/rz-admin/node-types*), cliquez sur * Ajouter un type de nœud* pour créer un nœud.

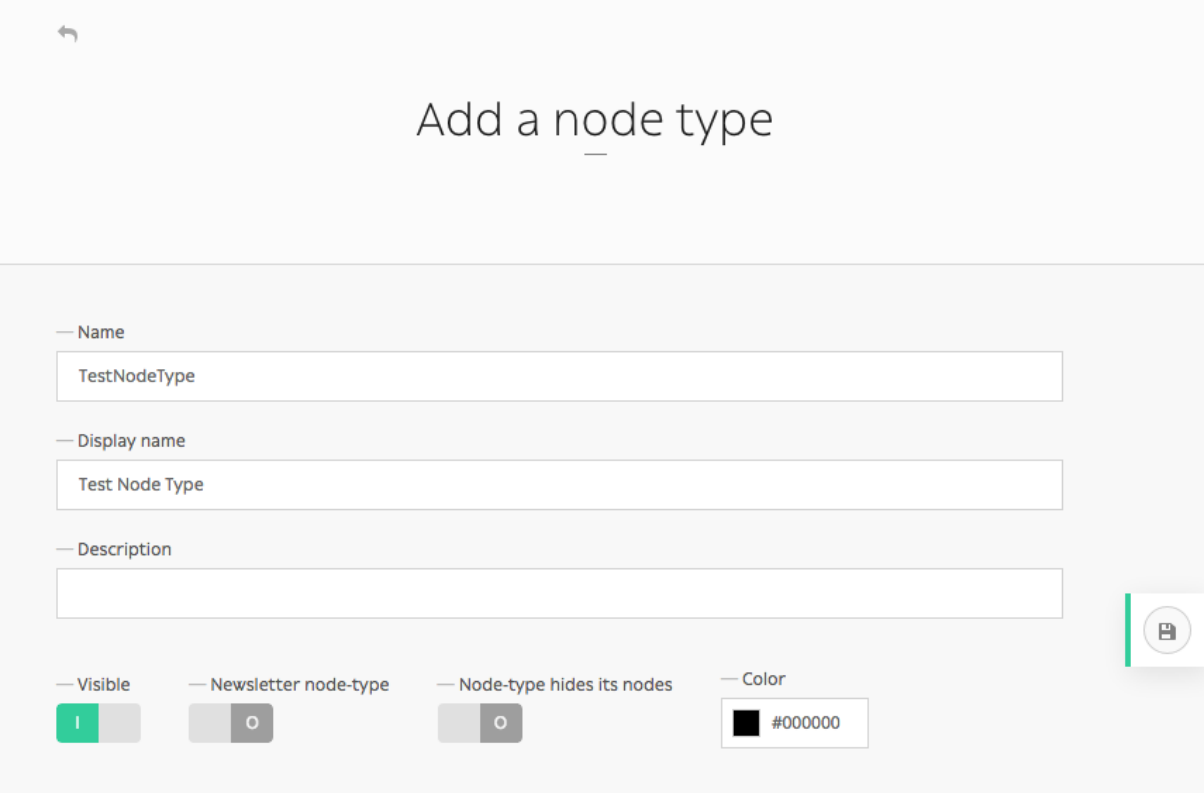
Note : À savoir, vous pouvez exporter et importer un type de nœud avec un fichier .json. Rendez-vous sur *Autre*

action pour plus d'informations.



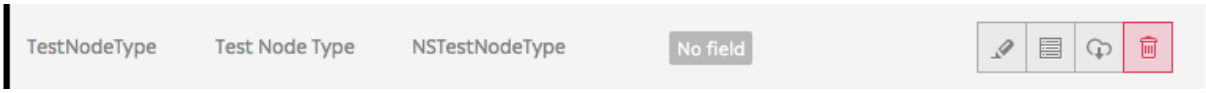
Upon filling the two mandatory settings *Name* (that developers will use) and *Display Name* (that back-office users will see), you are now ready to create your first node type.

Avertissement : Attention : une fois le nom de votre type de nœud défini et celui-ci créé, ce *Nom* ne pourra plus être modifié. En cas d'erreur, rendez-vous dans la section *Supprimer le type de nœud*.



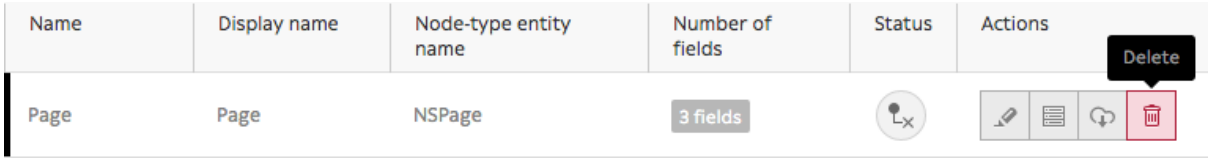
Les autres options (*Visible*, * Newsletter*, *Type de nœud caché* et les couleurs personnalisables) ne sont pas obligatoires et peuvent toujours être modifiées ultérieurement.

Vous avez à présent créé votre premier type de nœud ! Il apparaîtra dans la liste de vos types de nœuds disponibles et vous pouvez à présent gérer ses champs.



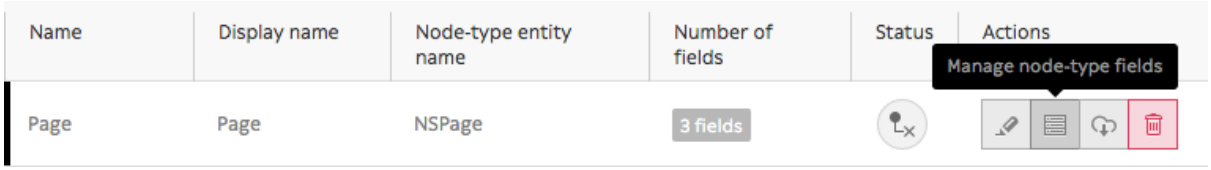
Supprimer un type de nœud

Vous avez fait une coquille en créant un type de nœud ? Ou vous n'avez tout simplement plus besoin de cette catégorie de nœuds ? Vous pouvez le supprimer en cliquant simplement sur l'icône corbeille de la page des Types de Nœuds (*https ://mywebsite.com/rz-admin/node-types*).



Ajouter un champ dans un type de nœud

Pour ajouter des champs à votre nouveau type de nœud, cliquez sur l'icône « Gestion des champs ».

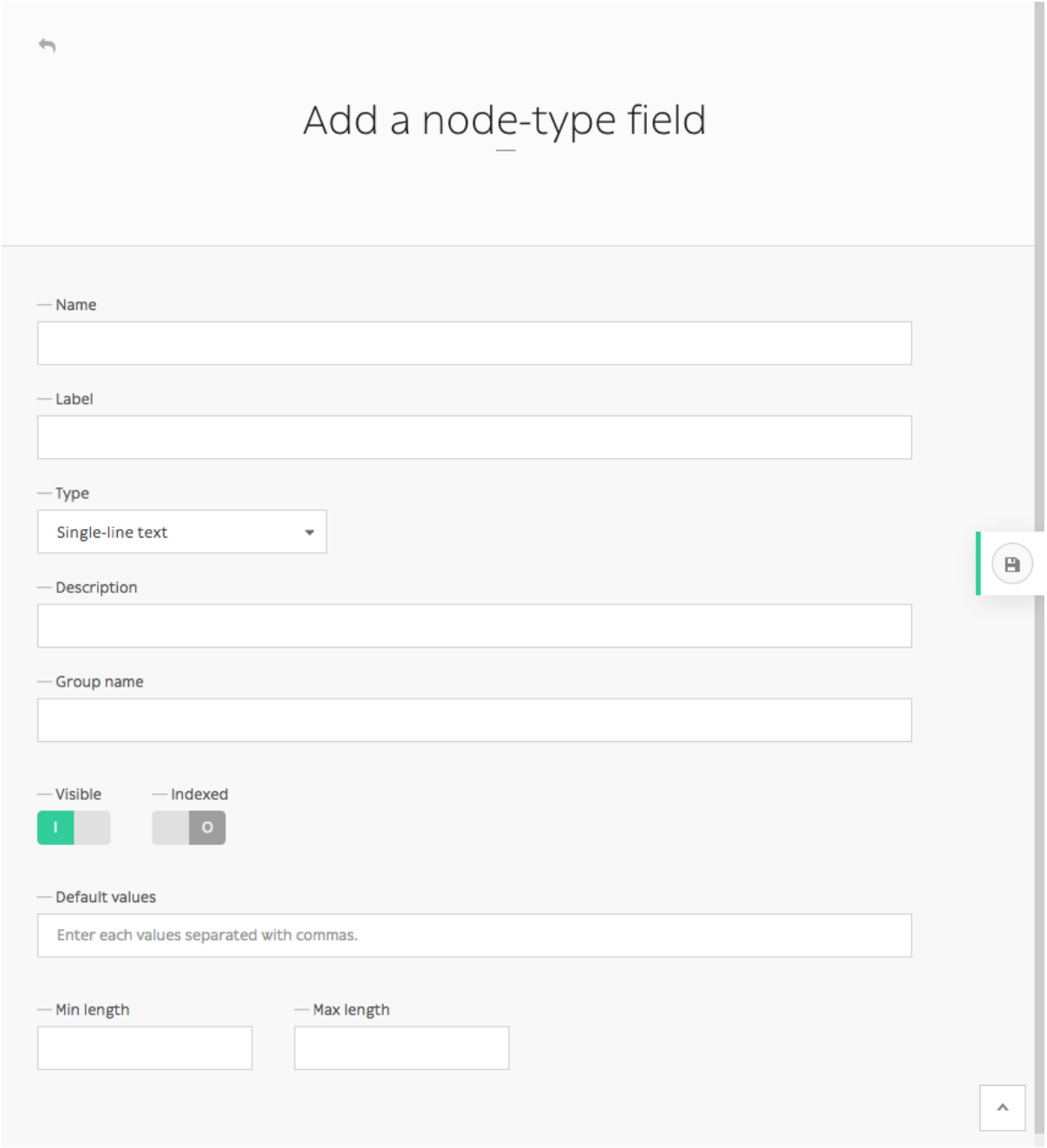


Puis cliquez sur le bouton "Ajouter un champ"

Remplissez le formulaire pour créer un nouveau champ :

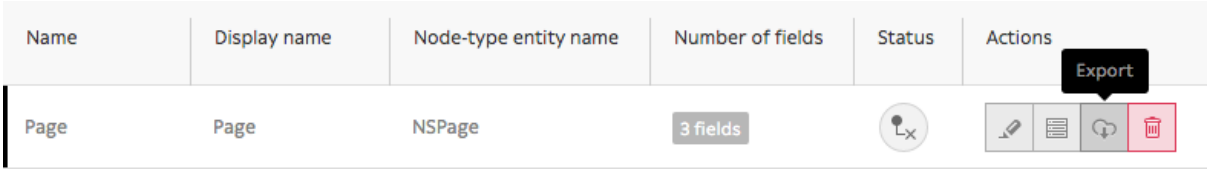
- *Nom* : utilisé par les développeurs
- *Étiquette* : ce que les utilisateurs du back-office verront
- *Type* : option de choix unique qui va définir le contenu (texte de base, texte markdown, documents, e-mail, numéro, choix simple ou multiple, nœuds enfants, etc.)
- *Description*, *Nom de groupe*, *Visible*, *Indexé*, *Valeur par défaut*, *Longueur minimum*, *Longueur maximum* (optionnel)

Note : Le champ *Valeurs par défaut » est important puisque vous pouvez y spécifier quel type de nœud vous souhaitez lui lier. Vous pouvez également l'utiliser comme un filtre dans la barre de recherche afin de ne voir que les les éléments contenant ces valeurs par défaut.



Autres actions

D'un site à un autre, vous pouvez exporter les types de nœuds au format JSON.



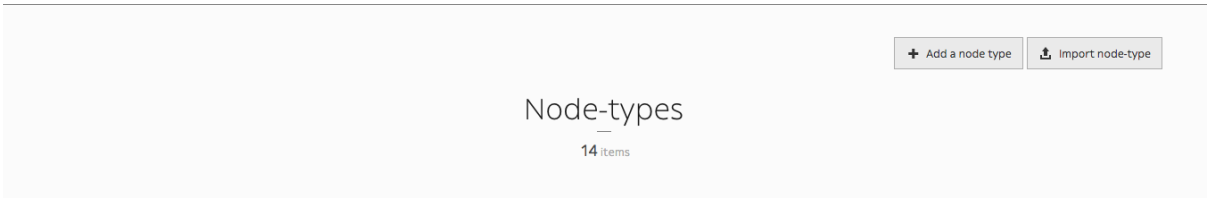
Un fichier .json devrait ressembler à cela lorsque vous l'ouvrez dans n'importe quel éditeur de code :

```
{
    "name": "Page",
    "displayName": "Page",
    "visible": true,
```

```
    "publishable": false,
    "reachable": true,
    "newsletterType": false,
    "hidingNodes": false,
    "color": "#000000",
    "fields": [
        {
            "position": 1.0,
            "name": "content",
            "label": "Content",
            "type": 4,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        },
        {
            "position": 2.0,
            "name": "children",
            "label": "N\u0153uds enfants",
            "type": 16,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        },
        {
            "position": 3.0,
            "name": "header_image",
            "label": "Header image",
            "type": 9,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        },
        {
            "position": 4.0,
            "name": "pictures",
            "label": "Pictures",
            "type": 9,
            "expanded": false,
            "nodeTypeName": "Page",
            "universal": false,
            "indexed": false,
            "visible": true
        }
    ],
    "defaultTtl": 0
}
```


Notez les quatre champs qui ont été ajoutés à ce type de nœud *Page*.

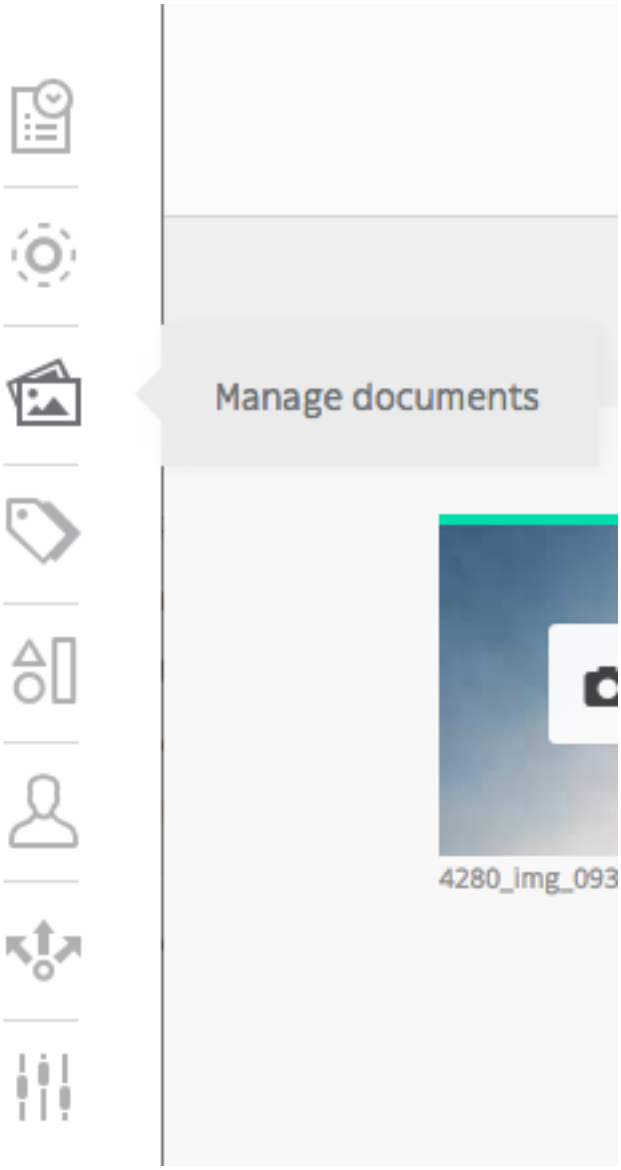
Vous pouvez écrire un fichier JSON vous-même, si vous en avez envie, mais il est probablement plus facile d'exporter simplement des types de noeuds à partir du back-office, puis d'importer sur votre nouveau site web. Vous pouvez le faire facilement en cliquant sur le bouton *Importer le type de nœud*, puis en le sélectionnant via votre explorateur de fichiers.



You are close to fully master Roadiz back-office powers ! Keep referring to this documentation if you have any problem, and let us know if any information goes missing.

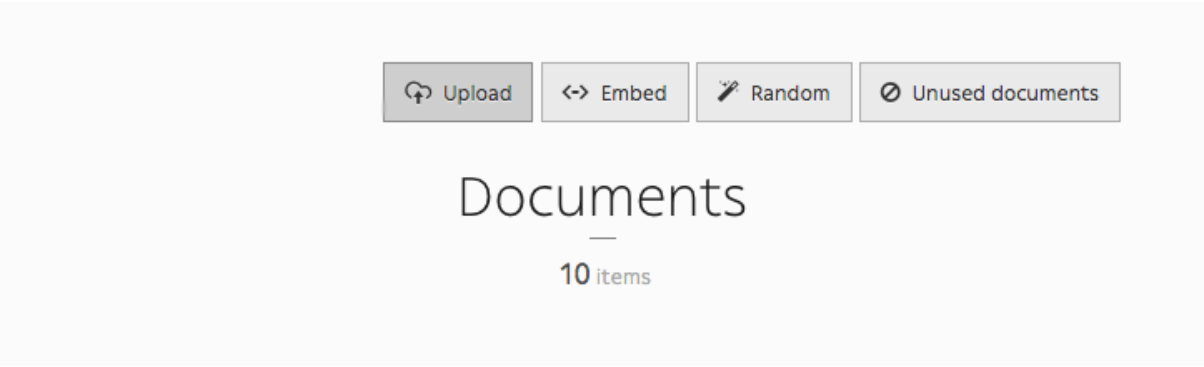
2.1.4 Gérer les documents

Vous pouvez gérer les documents via l'icône *Document* dans la barre d'outils. Vous pouvez soit téléverser, intégrer des documents externes, et même jeter un coup d'œil aux documents inutilisés de votre site Web.



Téléverser un document

Le téléversement d'un document n'a jamais été aussi simple : il suffit de glisser-déposer le document dans la zone désignée.



Lorsque vous téléchargez des fichiers, Roadiz essaie de réduire toute image de plus de 2500 pixels (largeur ou hauteur) afin de faciliter les transformations ultérieures. Si vous décidez d'augmenter cette limite au-delà de 2500 pixels, le CMS conservera toujours une version *naturelle* de votre document téléchargé. Gardez à l'esprit que Roadiz ne sera pas en mesure d'optimiser votre image dans un meilleur format : par exemple, si vous avez téléchargé une image PNG ne contenant pas de zone transparente, le CMS ne pourra pas la convertir au format JPEG et vous vous retrouverez avec un fichier lourd même si vous la redimensionnez.

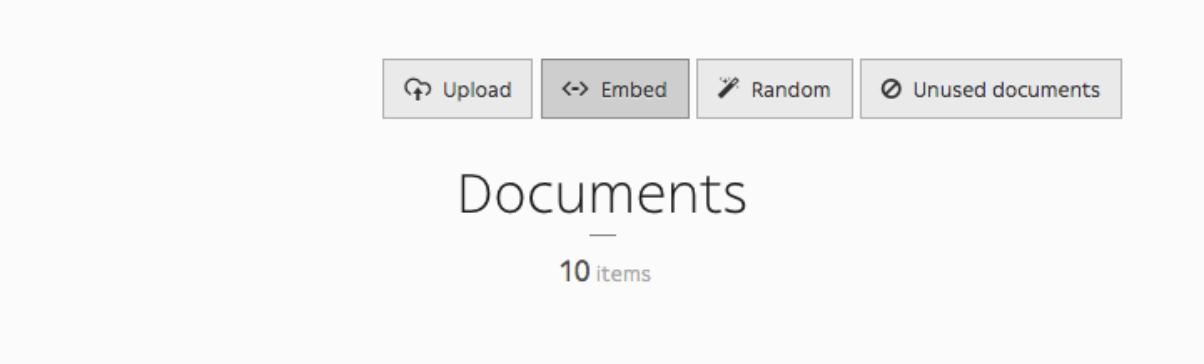
— Photographies, images complexes sans transparence : *utiliser JPEG*
— Logos, icônes, images complexes avec transparence : *utiliser PNG*

Utiliser un optimiseur d'image (comme *tinyjpg*, *kraken. o*, *jpgoptim*, ...) **avant** l'envoi sur Roadiz est inutile parce que, le CMS perdra ces optimisations lorsqu'il traitera les transformations (recadrage, redimensionnement, contraste et qualité).

D'un autre côté, si votre site web a été développé avec le support `picture`, Roadiz sera en mesure de servir toutes les images dans le nouveau format WEBP qui est 30% plus léger que le JPEG et supporte la transparence (canal alpha). Donc tout utilisateur navigant sur votre site web avec un Chrome, Firefox ou Edge ne verra que des images WEBP et économisera au moins 30% de la consommation de bande passante. Dans un avenir proche, le format d'image AVIF sera mieux supporté par les principaux navigateurs, et Roadiz pourra également générer une image optimisée avec ce format.

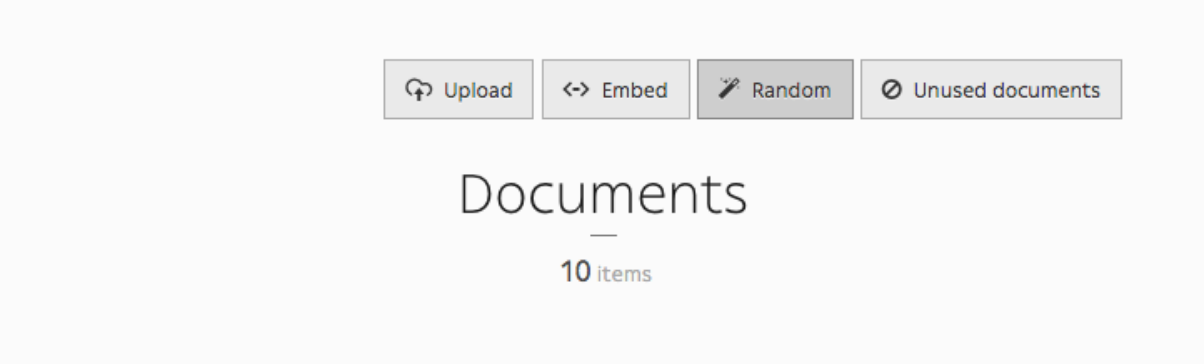
Intégrer un document

Voici la partie magique de la section Documents. Vous pouvez intégrer des médias depuis *YouTube*, *SoundCloud*, *Vimeo*, *Mixcloud*, *Spotify*, *Twitch*, *TED* et *Dailymotion*, puis utilisez-les comme s'il s'agissait de documents internes dans vos blocs de contenu.



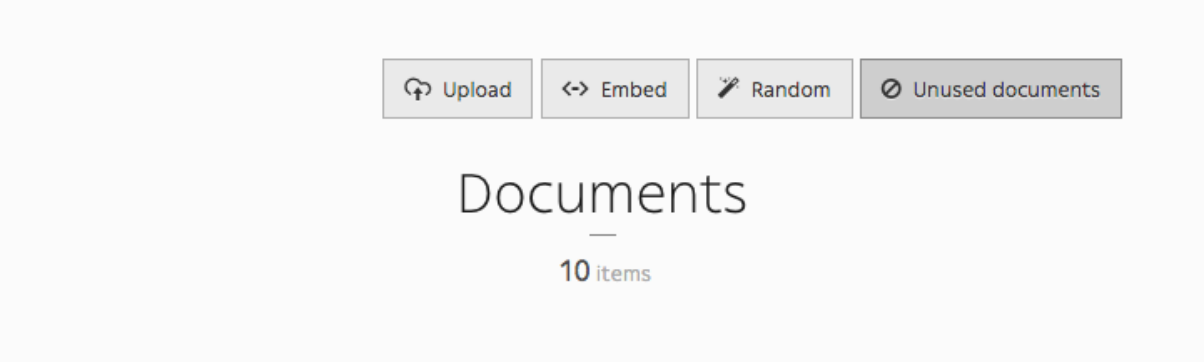
Document au hasard

J'ai de la chance est une fonctionnalité bien pratique qui permet de télécharger des documents au hasard sur votre site à partir de la base libre de *Splashbase*.



Document inutilisé

En cliquant sur *Document inutilisé*, vous pourrez vérifier la liste de toutes les sources non-référencées sur votre site, ce qui vous permettra par la suite de nettoyer votre base de données et de supprimer ainsi des fichiers inutiles.

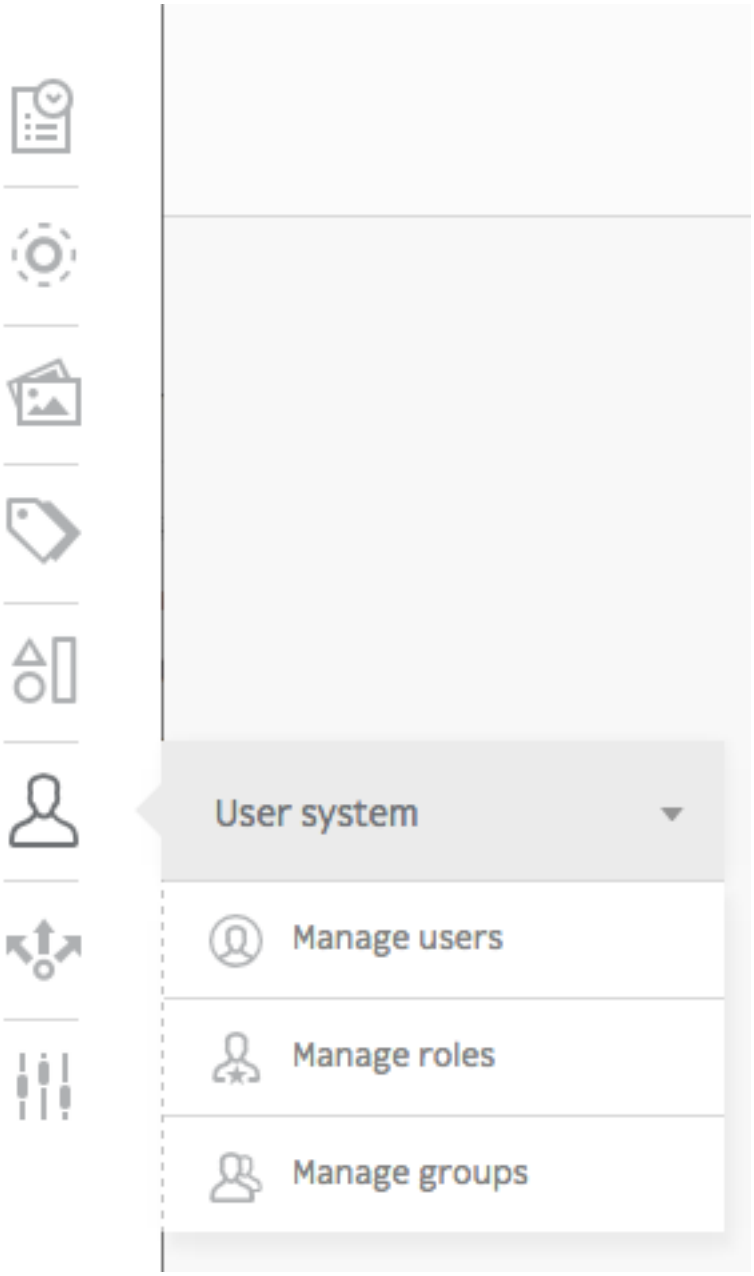


2.1.5 Gérer les utilisateurs

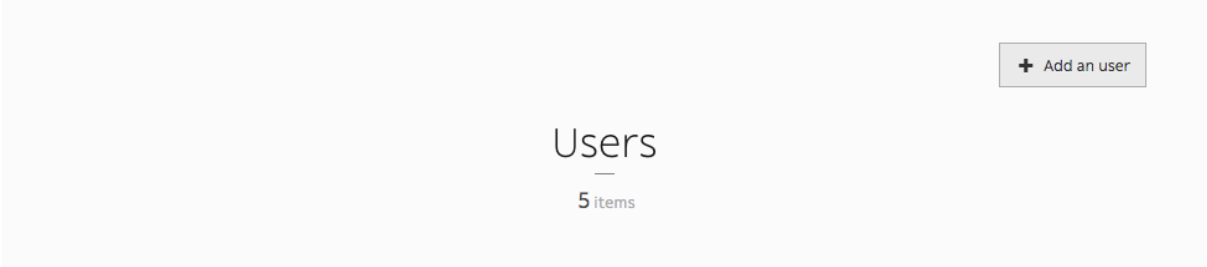
Il s'agit d'un guide simple sur la façon de créer et de gérer les utilisateurs en utilisant Roadiz CLI.

Il y a deux façons d'ajouter des utilisateurs, via le back-office et en ligne de commande.

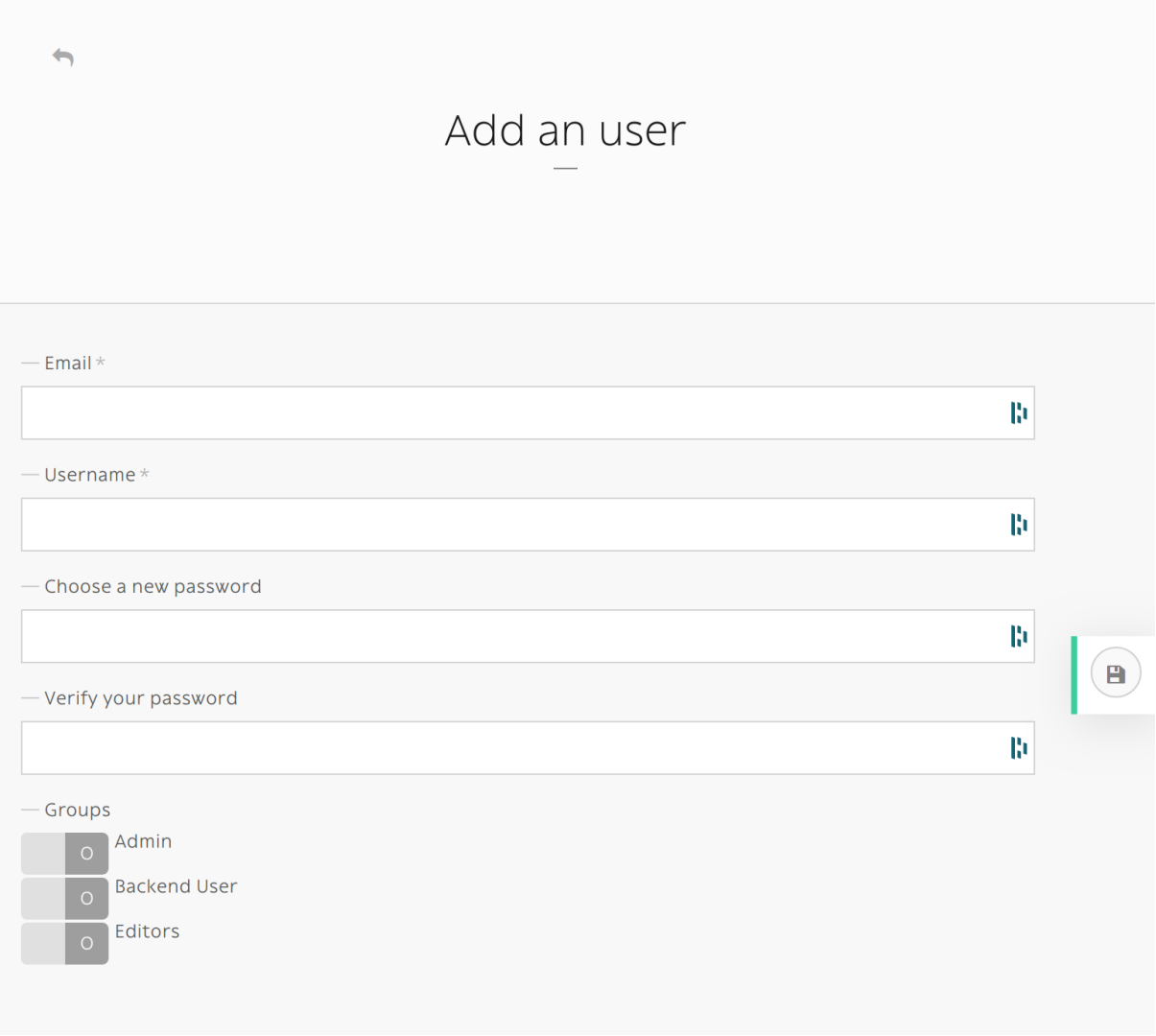
Lorsque vous travaillez dans le back-office de Roadiz, vous pouvez facilement gérer les utilisateurs via l'icône *Système utilisateur* dans la barre d'outils.



Créer un nouvel utilisateur



Vous pouvez ajouter des utilisateurs en cliquant sur le bouton *Ajouter un utilisateur*.



Si vous créez un utilisateur sans spécifier son *mot de passe*, un email avec un lien de réinitialisation du mot de passe sera envoyé. Assurez-vous que vous avez entré le bon email et que votre site web Roadiz a un *mailer* configuré. Le nouvel utilisateur sera verrouillé à moins qu'il ne réinitialise son mot de passe d'abord.

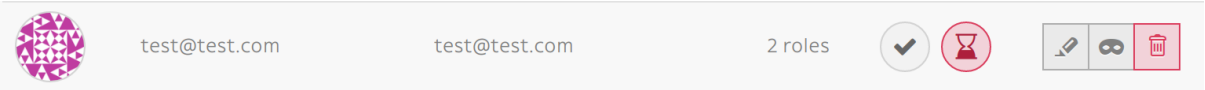
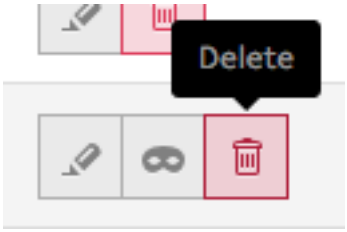


Fig. 2.1 – Un nouvel utilisateur sans mot de passe aura son compte verrouillé jusqu'à ce qu'il réinitialise son mot de passe.

La ligne de commande `bin/console users:create loginname` démarre une nouvelle session interactive de création d'un utilisateur. Vous allez créer un nouvel utilisateur avec son nom d'utilisateur et son email, vous pouvez aussi choisir si il s'agit d'un utilisateur backend ou bien super-admin.

Suppression d'un utilisateur

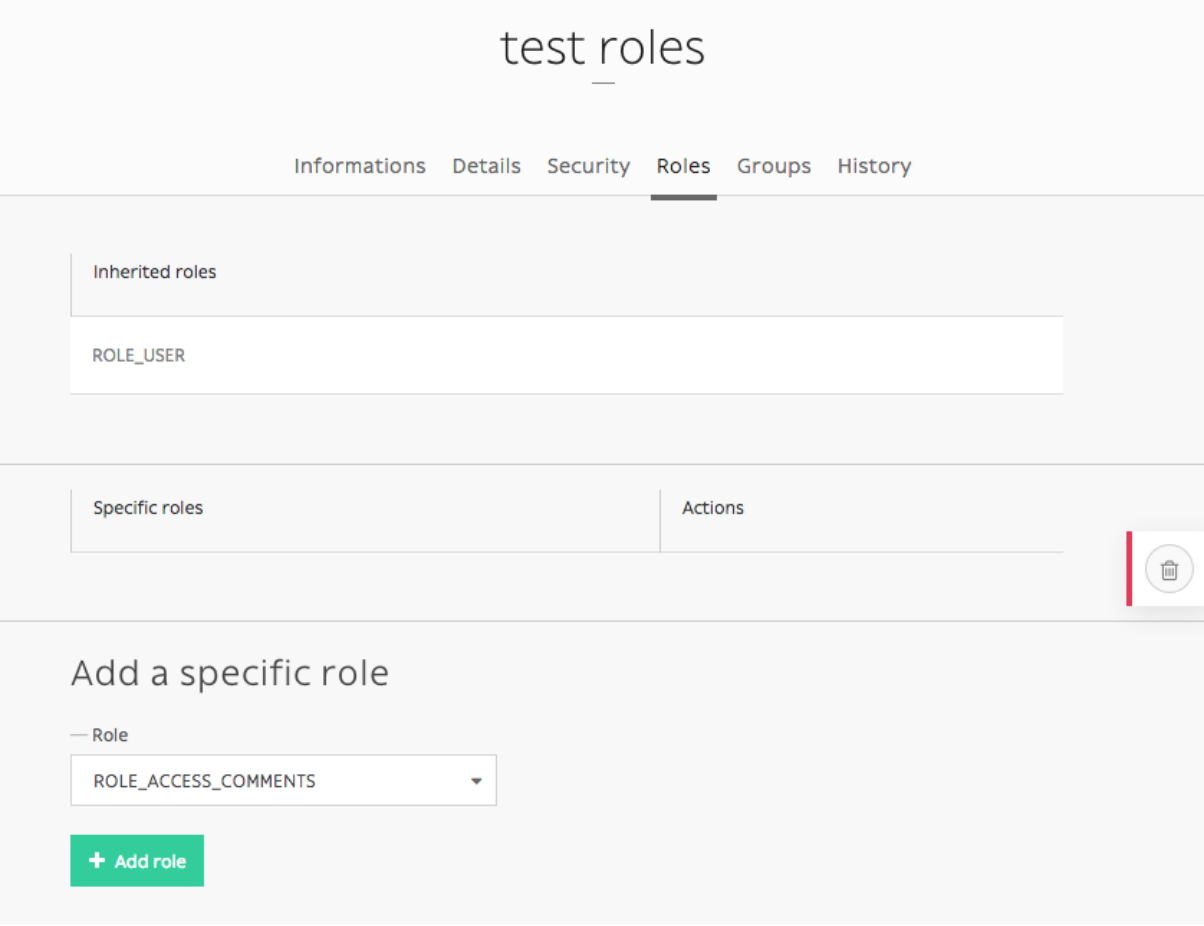
Vous pouvez supprimer des utilisateurs en cliquant sur l'icône de la corbeille.



La commande `bin/console users:delete loginname` supprime l'utilisateur « loginname ».

Ajouter un rôle

Vous pouvez modifier un profil utilisateur de la même manière que vous modifiez un type de nœud. Vous pouvez ajouter des rôles dans l'onglet *Roles*.



Si vous voulez ajouter le rôle `ROLE_SUPERADMIN` à l'utilisateur « test », il ressemblerait à cela en ligne de commande :

```
bin/console users:roles --add loginname
# You will be prompted to choose the ROLE with auto-complete feature.
```

Autres actions

It is possible to enable or disable users with `users:enable` or `users:disable` command. If a user doesn't remember his password, you can regenerate it with the `users:password` command. For more information and more actions, we invite you to check available commands with :

```
bin/console list users
```

Activer le SSO pour les utilisateurs du back-office

Roadiz est compatible avec la plupart des systèmes d'authentification centralisée (SSO) basés sur **OpenID** (Google, Exchange, ...). Vous pouvez configurer votre CMS pour que chaque membre de votre organisation puisse s'y connecter avec les `ROLES` que vous aurez choisi au préalable.

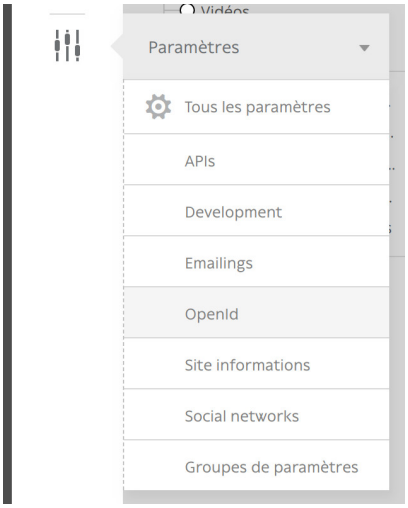
Avant d'aller plus loin, vérifiez que vous avez bien à votre dispositions les information suivantes depuis votre fournisseur d'identité *OpenID* :

— Client ID *OpenID*
— Client secret *OpenID*
— URL d'auto-découverte *OpenID* (ex : pour Google Suite https://accounts.google.com/.well-known/openid-configuration)

Puis vous devez déterminer :

— What *roles* (comma separated) you want to be automatically granted to users connected with your SSO.
— Quel nom de domaine utiliser pour restreindre les utilisateurs. C'est très important pour les utilisateurs de *Google Suite* car la configuration est la même pour tout les utilisateurs de ce service. Vous ne souhaitez pas autoriser tous les clients Google Suite à accéder à votre back-office .
— Un libellé de bouton pour la page de connexion de votre back-office.

```
# config/packages/roadiz_rozier.yaml
roadiz_rozier:
    open_id:
        # Verify User info in JWT at each login
        verify_user_info: false
        # Standard OpenID autodiscovery URL, required to enable OpenId login in␣
↪Roadiz CMS.
        discovery_url: '%env(string:OPEN_ID_DISCOVERY_URL)%'
        # For public identity providers (such as Google), restrict users emails by␣
↪their domain.
        hosted_domain: '%env(string:OPEN_ID_HOSTED_DOMAIN)%'
        # OpenID identity provider OAuth2 client ID
        oauth_client_id: '%env(string:OPEN_ID_CLIENT_ID)%'
        # OpenID identity provider OAuth2 client secret
        oauth_client_secret: '%env(string:OPEN_ID_CLIENT_SECRET)%'
        granted_roles:
            - ROLE_USER
            - ROLE_BACKEND_USER
            - ROLE_SUPERADMIN
```



Fill all your gathered information to the right *Roadiz* dotenv variables.

```
OPEN_ID_DISCOVERY_URL=https://accounts.google.com/.well-known/openid-configuration
OPEN_ID_HOSTED_DOMAIN=my-google-workspace-domain.com
OPEN_ID_CLIENT_ID=xxxxxxx
OPEN_ID_CLIENT_SECRET=xxxxxxx
```

Ensuite, si votre parametrage est correct, vous devriez voir votre bouton de connexion SSO sur la page de connexion du back-office *Roadiz*. Faites attention à ce que vos utilisateurs SSO aient les permissions suffisantes pour accéder au back-office, ils pourraient avoir une erreur 403 après avoir été redirigés depuis le flux SSO.

Note : L'implémentation OpenID de Google Suite est décrite sur https://developers.google.com/identity/protocols/oauth2/openid-connect#discovery

Premièrement, créez une nouvelle application OAuth2 sur https://console.cloud.google.com/ et suivez les instructions sur https://developers.google.com/identity/protocols/oauth2/openid-connect#getcredentials

CHAPITRE 3

Développeur - Documentation

3.1 Développeur - Documentation

3.1.1 Premiers pas

Requirements

Roadiz is a Symfony application running with PHP. You can follow regular Symfony requirements to optimize your local or production setup. Roadiz requires an HTTP server for static assets and **SSH access** with out/ingoing allowed connections. Here is a short summary of mandatory elements before installing Roadiz :

- Nginx or Apache, with a dedicated virtual host as described below.
- PHP 7.4+ **required**, 8.1 recommended
- Install theses PHP extensions (which are installed and enabled by default in most PHP installations) : JSON, Intl, cURL, MBString, Ctype, iconv, PCRE, Session, Zip, SimpleXML, and Tokenizer ;
- Your php.ini needs to have the `date.timezone` setting
- You need to have at least version 2.6.21 of libxml
- PHP *OPcache* + *APCu* (APC 3.0.17+ or another opcode cache needs to be installed)
- `php.ini` recommended settings
  - `short_open_tag = Off`
  - `magic_quotes_gpc = Off`
  - `register_globals = Off`
  - `session.auto_start = Off`
- MariaDB 10.5.2+ or MySQL 5.7+ database with *JSON_** functions support
- Install Composer, which is used to install PHP packages.
- Git

Avertissement : **Roadiz** and **Symfony** development and production environments heavily rely on Docker and docker-compose. We recommend you to learn these awesome tools if you're not using them yet.

You can use our official Docker image with *Nginx* and *PHP* already setup for you. We recommend that you create your own Docker image based on this official one.

Create a new Roadiz project

For new projects **Roadiz** can be easily setup using `create-project` command and our *Skeleton*.

```
# Create a new Roadiz project
composer create-project roadiz/skeleton my-website
cd my-website
# Create a local Dotenv to store your secrets
cp .env .env.local
# Edit your docker-compose parameter in .env to
# fit your development environment (OS, UID).
# .env file will be tracked by Git
#
# Initialize your Docker environment
docker-compose build
docker-compose up -d --force-recreate
```

Avertissement : **Roadiz** and **Symfony** development and production environments heavily rely on Docker and docker-compose. We recommend you to learn these awesome tools if you're not using them yet.

Composer will prompt you if you want to can versioning history. Choose the default answer `no` as we definitely want to replace *roadiz/skeleton Git* with our own versioning. Then you will be able to customize every files in your projects and save them using Git, not only your theme. Of course we added a default `.gitignore` file to prevent your configuration setting and entry points to be committed in your *Git* history. That way you can have different configuration on development and on your production server without bothering about merge conflicts.

Generate JWT private and public keys

```
# Generate a strong secret
openssl rand --base64 16;
# Fill JWT_PASSPHRASE env var.
openssl genpkey -out config/jwt/private.pem -aes256 -algorithm rsa -pkeyopt rsa_
↪keygen_bits:4096;
openssl pkey -in config/jwt/private.pem -out config/jwt/public.pem -pubout;
```

Install database

```
# Create and migrate Roadiz database schema
docker-compose exec -u www-data app bin/console doctrine:migrations:migrate
# Migrate any existing data types
docker-compose exec -u www-data app bin/console themes:migrate ./src/Resources/
↪config.yml
# Install base Roadiz fixtures, default translation, roles and settings
docker-compose exec -u www-data app bin/console install
# Clear cache
docker-compose exec -u www-data app bin/console cache:clear
# Create your admin account
docker-compose exec -u www-data app bin/console users:create -m username@roadiz.io␣
↪-b -s username
```

Then connect to `http://localhost:${YOUR_PORT}/rz-admin` to access your freshly-created Roadiz backoffice.

Note : If you setup Traefik on your local environment, you can reach your Roadiz app using your `domain.test` test domain name without specifying a non-default port. You have to change `HOSTNAME` dot-env variable and change your local DNS to point `domain.test` to `127.0.0.1`. The easiest way is to add `127.0.0.1 domain.test` to your `/etc/hosts` file.

Configuration manuelle

Roadiz is a full-stack Symfony application. It follows its configuration scheme as described in https://symfony.com/doc/5.4/configuration.html

Choisissez votre modèle d'héritage

Roadiz main feature is all about its polymorphic document model which is mapped on a relational database. This requires a challenging structure which can be lead to some performance bottlenecks when dealing with more than 20-30 node-types. So we made the data inheritance model configurable to allow switching to single_table scheme which will be more performant if you need lots of node-types. However *Single class* model will drop support for fields with the *same name but not the same type* because all node-type fields will be created in the **same SQL table**.

If you really need to mix field types, we advise you to keep the original joined table inheritance type which creates a dedicated SQL table for each node-type. *Joined table* inheritance can be very useful with a small number of node-type (max. 20) and very different fields. But its main drawback is that Roadiz needs to *LEFT JOIN* every node-type table for each node-source query, **unless you specify one node-type criteria**.

You can configure *Doctrine* strategy for NodesSources inheritance classes in `config/packages/roadiz_core.yaml` :

```
# config/packages/roadiz_core.yaml
roadiz_core:
    inheritance:
        # type: joined
        type: single_table
```

— Héritage par tables jointes : `joined`
— Héritage à l'aide d'une seule table : `single_table`

Avertissement : Si vous modifiez ce paramètre après avoir créé du contenu dans votre site Web, toutes les données des sources de nœud seront perdues.

Themes (compatibility with v1.x)

Themes are statically registered into Roadiz configuration for better performances and delaying database usage. You have to register any front-end theme in your `config/packages/roadiz_compat.yaml` file. Theme priority is not handled here but in each of your themes by overriding static `$priority` value;

```
# config/packages/roadiz_compat.yaml
roadiz_compat:
    themes:
        -
            classname: \Themes\DefaultTheme\DefaultThemeApp
            hostname: '*'
            routePrefix: ''
        -
            classname: \Themes\FooBarTheme\FooBarThemeApp
            hostname: 'foobar.test'
            routePrefix: ''
```

Vous pouvez définir des thèmes spécifiques au nom d'hôte et ajouter un préfixe de routage. Les valeurs par défaut sont `'*'` pour *hostname* et `''` (chaîne vide) pour le préfixe de route.

Avertissement : Si vous ne configurez aucun thème, cela mènera à une erreur 404 sur la page d'accueil de votre site. Mais vous aurez toujours accès au back-office qui est enregistré en dur dans la configuration de Roadiz.

Point d'entrée Solr

Roadiz can use an *Apache Solr* search-engine to index nodes-sources. Add this to your *config/packages/roadiz_core.yaml* to link your CMS to your *Solr* server :

```
# config/packages/roadiz_core.yaml
roadiz_core:
    solr:
        endpoint:
            localhost:
                host: "localhost"
                port: "8983"
                path: "/"
                core: "mycore"
                timeout: 3
                username: ""
                password: ""
```

Roadiz CLI command can easily handle Solr index. Just type `./bin/console solr:check` to get more information.

Invalidation du cache des reverse-proxies

Roadiz peut demander l'invalidation du cache à des reverse-proxies externes et internes tels que *Symfony* App-Cache ou bien une instance *Varnish*. Si configuré, Roadiz créera une requête `BAN` pour chaque proxy configuré **quand l'utilisateur efface les caches depuis le back-office**, et il va créer une requête `PURGE` **sur chaque node-source** *mis-à-jour* en utilisant la première URL de node-source accessible.

```
# config/packages/roadiz_core.yaml
roadiz_core:
    reverseProxyCache:
        frontend:
            default:
                host: '%env(string:VARNISH_HOST)%'
                domainName: '%env(string:VARNISH_DOMAIN)%'
```

Note : Assurez-vous que vous avez configuré votre reverse-proxy externe pour recevoir et gérer les requêtes HTTP `BAN` et `PURGE`.

Cache proxy Cloudflare

Si vous utilisez Cloudflare comme un reverse-proxy cache, vous pouvez configurer Roadiz pour envoyer des requêtes à Cloudflare pour purger tous les éléments ou fichiers (lors de l'édition d'un node-source). Vous devez renseignez les informations suivantes :

— Identifiant de la zone Cloudflare
— Identifiants de l'API Cloudflare (Bearer token ou email + clé d'identification)

Ensuite, vous pouvez configurer Roadiz avec le Bearer token :

```
# config/packages/roadiz_core.yaml
roadiz_core:
    reverseProxyCache:
        frontend: []
        cloudflare:
            zone: cloudflare-zone
            bearer: ~
```

Ou avec votre E-mail et votre AuthKey :

```
# config/packages/roadiz_core.yaml
roadiz_core:
    reverseProxyCache:
        frontend: []
        cloudflare:
            zone: cloudflare-zone
            email: ~
            key: ~
```

Note : Roadiz utilise les points d'entrée *Purge all files* et *Purge Files by URL* : https://api.cloudflare.com/#zone-purge-all-files qui sont disponibles sur tous les plans Cloudflare.

Chemins des entités

Roadiz uses *Doctrine* to map object entities to database tables.

```
# config/packages/doctrine.yaml
doctrine:
    orm:
        mappings:
            App:
                is_bundle: false
                type: attribute
                dir: '%kernel.project_dir%/src/Entity'
                prefix: 'App\Entity'
                alias: App
            RoadizCoreBundle:
                is_bundle: true
                type: attribute
                dir: 'src/Entity'
                prefix: 'RZ\Roadiz\CoreBundle\Entity'
                alias: RoadizCoreBundle
            RZ\Roadiz\Core:
                is_bundle: false
                type: attribute
                dir: '%kernel.project_dir%/vendor/roadiz/models/src/Roadiz/Core/
↪AbstractEntities'
                prefix: 'RZ\Roadiz\Core\AbstractEntities'
                alias: AbstractEntities
            App\GeneratedEntity:
                is_bundle: false
                type: attribute
                dir: '%kernel.project_dir%/src/GeneratedEntity'
                prefix: 'App\GeneratedEntity'
                alias: App\GeneratedEntity
```

Use `type: attribute` or `type: annotation` according to your Doctrine mapping type.

Configurer le mailer

Roadiz uses *Symfony Mailer* to send emails.

https://symfony.com/doc/5.4/mailer.html#transport-setup

Note : Faites attention au fait que de nombreux services SMTP externes (*Mandrill*, *Mailjet*...) n'acceptent que les emails provenant de domaines validés. Assurez-vous donc que votre application utilise un expéditeur connu d'email `From:` pour ne pas être blacklisté ou bloqué par ces services. Si vous avez besoin de répondre à vos e-mails à une adresse anonyme, utilisez plutôt l'en-tête `ReplyTo:`.

Traitement des images

Roadiz use Intervention Request Bundle to automatically create a lower quality version of your image if they are too big and offer on-the-fly image resizing and optimizing.

```
# config/packages/rz_intervention_request.yaml
parameters:
    env(IR_DEFAULT_QUALITY): '90'
    env(IR_MAX_PIXEL_SIZE): '1920'
    ir_default_quality: '%env(int:IR_DEFAULT_QUALITY)%'
    ir_max_pixel_size: '%env(int:IR_MAX_PIXEL_SIZE)%'

rz_intervention_request:
    driver: 'gd'
    default_quality: '%ir_default_quality%'
    max_pixel_size: '%ir_max_pixel_size%'
    cache_path: "%kernel.project_dir%/public/assets"
    files_path: "%kernel.project_dir%/public/files"
    jpegoptim_path: /usr/bin/jpegoptim
    pngquant_path: /usr/bin/pngquant
    subscribers: []
```

Extensions supplémentaires *Intervention Request*

N'importe quelle extension *Intervention Request* peut être ajoutée à la configuration avec l'aide de son `classname` et ses arguments constructeurs. Voici un exemple avec `WatermarkListener` qui imprimera du texte sur toutes vos images.

```
rz_intervention_request:
    # List additional Intervention Request subscribers
    subscribers:
        - class: "AM\\InterventionRequest\\Listener\\WatermarkListener"
          args:
               - 'Copyright 2017'
               - 3
               - 50
               - "#FF0000"
```

Utilisez kraken.io pour réduire considérablement la taille des images

Puisque vous pouvez ajouter des extension à *Intervention Request*, nous avons créé une qui envoie toutes vos images au service kraken.io pour les optimiser. Une fois que vous l'avez configuré, n'oubliez pas de vider vos caches pour voir les changements.

```
rz_intervention_request:
    # List additional Intervention Request subscribers
    subscribers:
        - class: "AM\\InterventionRequest\\Listener\\KrakenListener"
          args:
               - "your-api-key"
               - "your-api-secret"
               - true
```

Avertissement : Notez que chaque image générée est envoyée aux serveurs *kraken.io*. Cela peut prendre du temps pour la première génération d'image.

Commandes de console

Roadiz peut être exécuté comme un simple outil CLI en utilisant votre connexion SSH. Ceci est utile pour gérer les tâches d'administration de base sans avoir besoin d'une administration graphique.

```
./bin/console
```

Si votre système n'est pas configuré pour avoir *php* situé dans `/usr/bin/php` utilisez-le de cette façon :

```
php ./bin/console
```

La commande par défaut sans argument vous montrera la liste des commandes disponibles. Chaque commande a ses propres paramètres. Vous pouvez utiliser l'argument `--help` pour obtenir plus d'informations sur chaque outil :

```
./bin/console install --help
```

Nous avons même rendu les outils CLI *Doctrine* directement disponibles à partir de Roadiz Console. Attention, ce sont des commandes puissantes qui peuvent modifier votre base de données et vous faire perdre des données précieuses. Surtout lorsque vous aurez besoin de mettre à jour votre schéma de base de données auprès d'un thème ou suite à une mise à jour du noyau. **Faites toujours une sauvegarde de la base de données avant toute opération Doctrine**.

Upgrading

Note : Always do a database backup before upgrading. You can use the *mysqldump* or *pg_dump* tools to quickly export your database as a file.

— With a MySQL server : `mysqldump -u[user] -p[user_password] [database_name] > dumpfilename.sql`
— With a PostgreSQL server : `pg_dump -U [user] [database_name] -f dumpfilename.sql`

Use *Composer* to update dependencies or Roadiz itself with *Standard* or *Headless* editions, make sure that your Roadiz *version constraint* is set in your project `composer.json` file, then :

```
composer update -o;
```

Run database registered migrations (some migrations will be skipped according to your database type). Doctrine migrations are the default method to upgrade all none-node-type related entities :

```
bin/console doctrine:migrations:migrate;
```

In order to avoid losing sensible node-sources data. You should regenerate your node-source entities classes files :

```
bin/console generate:nsentities;
```

Then check if there is no pending SQL changes due to your Roadiz node-types :

```
bin/console doctrine:schema:update --dump-sql;
# Upgrade node-sources tables if necessary
bin/console doctrine:schema:update --dump-sql --force;
```

Then, clear your app caches :

```
# Clear cache for each environment
bin/console cache:clear -e dev
bin/console cache:clear -e prod
```

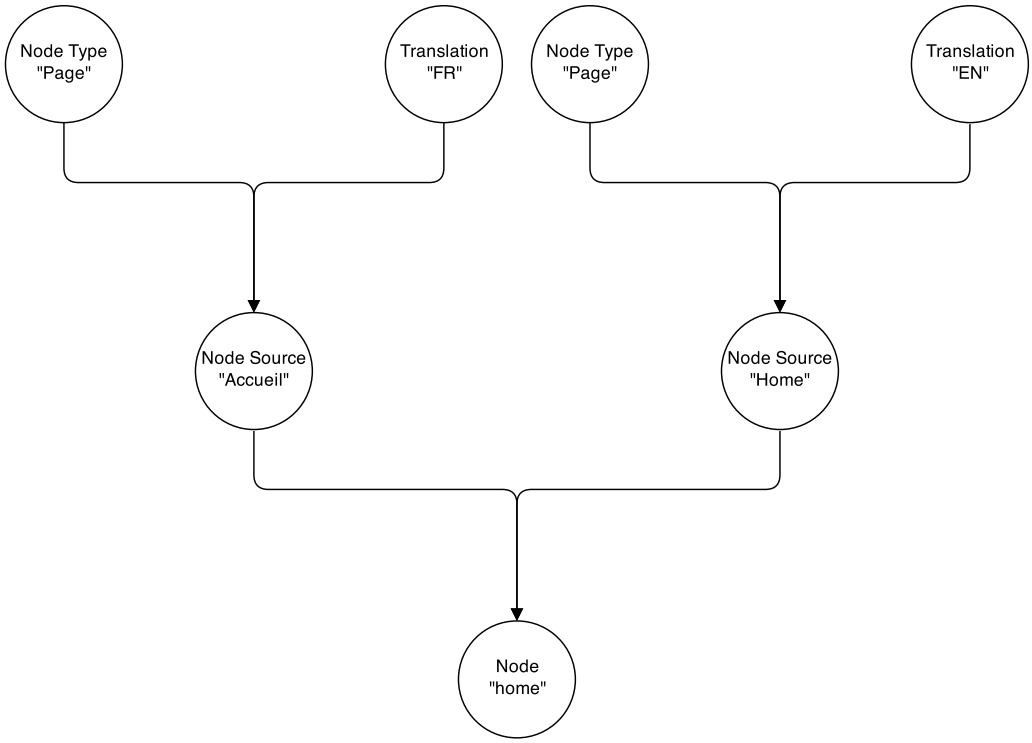
Note : If you are using a runtime cache like OPcache or APCu, you'll need to purge cache manually because it can't be done from a CLI interface as they are shared cache engines. As a last chance try, you can restart your `php-fpm` service.

3.1.2 Système de nœuds

Types de nœuds, nodes-sources et nœuds

Cette partie est la partie la plus importante de Roadiz. Presque tous les contenus de votre site seront créés sous la forme d'un nœud.

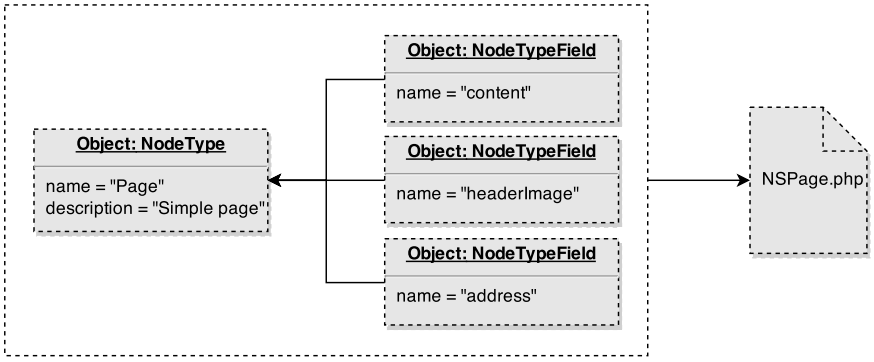
Regardons ce simple schéma de nœud avant de l'expliquer.



Maintenant, il est temps d'expliquer comment ça marche !

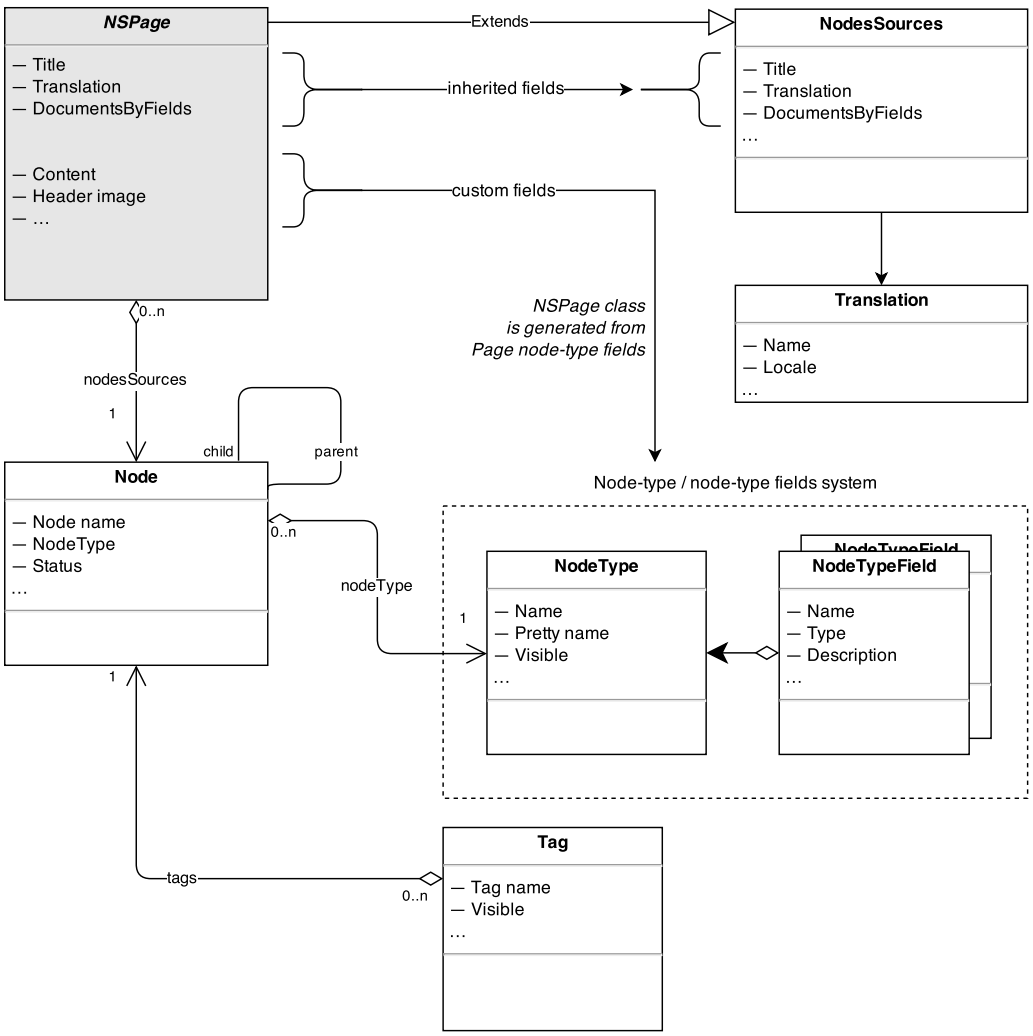
Qu'est-ce qu'un type de nœud

Un type de nœud est le gabarit de votre node-source. Il contiendra tous les champs que Roadiz utilisera pour générer une classe de node-source étendue.



Par exemple, un type de nœud « page » contiendra les champs « content » et « header image ». Le champ « title » est toujours disponible car il est codé en dur dans la classe `NodesSources`. Après avoir sauvegardé votre type de nœud, Roadiz génère une classe PHP `NSPage` qui étend la classe `NodesSources`. Vous le trouverez dans `gen-src/GeneratedNodeSources` (ou `app/gen-src/GeneratedNodeSources` avec *Roadiz Standard edition*). Roadiz appelle alors l'outil de mise à jour *Doctrine* pour migrer votre schéma de base de données. **Ne modifiez pas la classe générée.** Vous devrez la mettre à jour par l'interface d'administration.

Voici un schéma pour comprendre comment les types de noeuds peuvent définir des champs personnalisés dans les node-sources :



Le plus gros de la gestion des types de nœud sera effectués dans l'interface du back-office. Vous serez en mesure de créer, de mettre à jour les types de nœud et chacun de leurs champs de manière indépendante. Mais si vous préférez, vous pouvez utiliser les commandes CLI pour créer des types et des champs. Avec les commandes CLI de Roadiz, vous obtenez plusieurs outils pour gérer les types de nœuds. Nous vous encourageons vraiment à vérifier les commandes avec l'argument `--help`, comme suit :

```
bin/console nodetypes:add-fields
bin/console nodetypes:create
bin/console nodetypes:delete
bin/console nodetypes:list
```

Gardez à l'esprit que chaque opération de type de nœud ou de type de nœud nécessite une mise à jour de la base de données car Doctrine doit créer une table spécifique par type de noeud. N'oubliez pas d'exécuter les outils `bin/console doctrine:schema:update` pour effectuer des mises à jour. Il est très important de comprendre que *Doctrine* a besoin de voir les classes générées par vos types de nœuds **avant** la mise à jour du schéma de base de données. S'ils n'existent pas, il ne pourra pas créer vos tables de types personnalisés ou pire, il pourrait supprimer des données existantes, car *Doctrine* ne reconnaîtra pas ces tables spécifiques.

Jetons maintenant un œil sur les sources de nœud.

Sources de nœuds et traductions

Une fois votre type de nœud créé, sa définition est stockée dans la base de données dans les tables `node_types` et `node_type_fields`. Ces informations ne seront utilisées que pour construire vos formulaires d'édition de node-sources dans le back-office et pour construire une table de base de données personnalisée.

Héritage des données

Avec Roadiz, chaque donnée basée sur un type de nœud (appelée node-sources) est stockée dans une table différente préfixée par `ns_`. Lorsque vous créez un type de nœud *Page* avec 2 champs (*content* et *excerpt*), Roadiz dit à Doctrine de construire une table `ns_page` avec 2 colonnes et une clé primaire héritée de la table `nodes_sources`. Cela s'appelle : *Inheritance mapping*, votre table `ns_page` hérite de la table `nodes_sources` et lorsque vous interrogez une *Page* depuis la base de données, Doctrine combine les données provenant de ces 2 tables pour créer une source de nœud complète.

À la fin, votre node-source *Page* ne contiendra pas que 2 champs, mais bien plus, puisque l'entité `NodesSources` définit les `title`, `metaTitle`, `metaDescription`, `metaKeywords` et d'autres champs de données génériques qui peuvent être utilisés sur tous les types de nœuds.

Traductions

L'héritage des données des Node-sources est non seulement utilisé pour personnaliser les données, mais aussi pour les traduire. Comme vous l'avez vu dans la première image, chaque nœud peut possèder de nombreuses sources, à savoir une par langue.

Champs de type de nœud

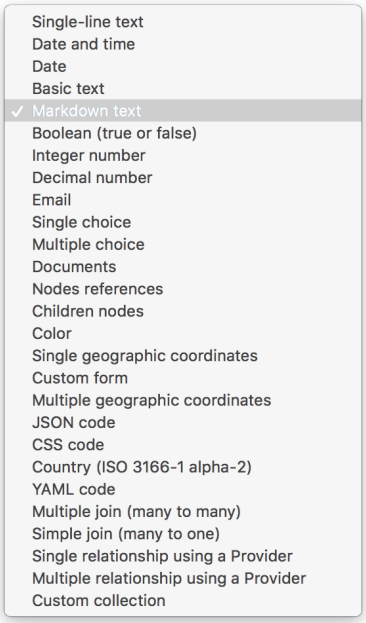
Roadiz peut gérer de nombreux champs de type de nœud. Voici une liste complète :

Note : *Titre*, *méta-title*, *méta-description* et *mots clés* sont toujours disponibles puisqu'ils sont stockés directement dans l'entité `NodesSources`. Vous serez sûr d'avoir toujours un *titre* quel que soit le type de nœud que vous utilisez.

Données simples

Les champs suivants stockent des données simples dans votre table de base de données de nœuds.

- Chaîne de caractères
- Date
- Date et heure
- Texte basic
- Texte Markdown
- Booléen
- Nombre entier
- Nombre décimal
- E-mail
- Couleur
- Coordonnées géographiques uniques
- Multiple geographic coordinates
- JSON code
- Code CSS
- Code pays (ISO 3166-1 alpha-2)
- Code YAML
- Jointure Many to many
- Jointure Many to One
- Relation unique en utilisant un Provider
- Relation multiple en utilisant un Provider
- Collection personnalisée



Single and multiple geographic coordinates

Geographic coordinates are stored in JSON format in your database using GeoJSON schema :

— A single point will be stored as a GeoJSON *feature* in order to hold additional properties such as *zoom*,
— Multiple points will be stored as a GeoJSON *feature collection*

By default, Roadiz back-office uses *Leaflet* library with *Open Street Map* for tiles rendering and basic geo-coding features.

Markdown options

You can restrict Markdown fields buttons using the following YAML configuration :

```
allow_h2: false
allow_h3: false
allow_h4: false
allow_h5: false
allow_h6: false
allow_bold: false
allow_italic: false
allow_blockquote: false
allow_list: false
allow_nbsp: false
allow_nb_hyphen: false
allow_image: false
allow_return: false
allow_link: false
allow_hr: false
allow_preview: false
```

Données virtuelles

Les types virtuels ne stockent pas vraiment les données dans la table node-source. Ils affichent des widgets personnalisés dans votre page d'édition pour lier des documents, des nœuds ou des formulaires personnalisés à votre node-source.

— Documents
— Références de nœuds
— Formulaires personnalisés

Données complexes

Ces types de champs doivent être créés avec des *valeurs par défaut* (séparées par des virgules) afin de les afficher sous la forme d'une "boîte de sélection" :

— Choix simple
— Choix multiple
— Nœuds enfants

Le type de champ *Nœuds enfant* est un champ virtuel spécial qui affichera une arborescence personnalisée à l'intérieur de votre page. Vous pouvez ajouter des boutons *création rapide* en listant les noms de vos types de nœuds dans les *valeurs par défaut*.

Champs universels

Si vous avez besoin que votre champ contienne exactement les mêmes données pour toutes les traductions, vous pouvez le définir comme *universel*. Par exemple, pour les documents, les données numéraires et booléennes qui ne changent pas d'une langue à une autre.

Cela va dupliquer les données à chaque fois depuis la traduction par défaut vers les autres pour vous faire gagnez du temps. Cela masquera également les champs d'édition sur les autres traductions pour éviter toute confusion lors du remplissage.

Champ YAML

Lorsque vous utilisez le type de champ YAML, vous obtenez une méthode supplémentaire pour retourner votre code déjà analysé. Si votre champ est nommé `data`, vos méthodes seront générées dans votre *NSEntity* en tant que `getData()` et `getDataAsObject()`.

— La méthode `getData()` retournera votre code YAML en tant que *string*.
— `getDataAsObject()` retournera des données mixtes, un tableau ou un `stdObject` selon le formatage de votre code. Cette méthode peut lancer une `\Symfony\Component\Yaml\Exception\ParseException` si votre code YAML n'est pas valide.

Jointures Many to Many et Many to One

Vous pouvez créer des relations personnalisées entre votre source et toutes les entités Doctrine disponible dans *Roadiz* ou dans votre thème.

Vous devez remplir le champ *valeurs par défaut* pour ces deux types.

```
# Entity class name
classname: Themes\MyTheme\Entities\City
# Displayable is the method used to display entity name
displayable: getName
# Same as Displayable but for a secondary information
alt_displayable: getZipCode
# Searchable entity fields
searchable:
    - name
    - slug
orderBy:
    - field: slug
      direction: ASC
```

You can use a custom proxy entity to support persisting `position` on your relation. Roadiz will generate a one-to-many relationship with proxy entity instead of a many-to-many. In this scenario you are responsible for creating and migrating `Themes\MyTheme\Entities\PositionedCity` entity. If you are migrating from a non-proxied many-to-many relation, you should keep the same table and field names to keep data intact.

```
# Entity class name
classname: Themes\MyTheme\Entities\City
# Displayable is the method used to display entity name
displayable: getName
# Same as Displayable but for a secondary information
alt_displayable: getZipCode
# Searchable entity fields
searchable:
    - name
    - slug
# This order will only be used for explorer
orderBy:
    - field: slug
      direction: ASC
# Use a proxy entity
proxy:
    classname: Themes\MyTheme\Entities\PositionedCity
    self: nodeSource
    relation: city
    # This order will preserve position
    orderBy:
        - field: position
          direction: ASC
```

Provider simple et multiple

Le type de Provider générique vous permet de récupérer toutes les données que vous voulez à travers une classe `Provider` depuis votre thème. Cela peut être très utile si vous avez besoin de récupérer des éléments depuis une API externe et de les référencer dans vos nodes-sources.

Imaginez que vous voulez lier votre page avec un post *Instagram*. Vous devrez créer une classe qui hérite de `Themes\Rozier\Explorer\AbstractExplorerProvider` et la configurer dans votre champ :

```
classname: Themes\MyTheme\Provider\ExternalApiProvider
```

Ce Provider va implémenter `getItems`, `getItemsById` et d'autres méthodes de `ExplorerProviderInterface` afin de pouvoir afficher vos posts *Instagram* dans le widget Roadiz et retrouver les éléments sélectionnés. Chaque post *Instagram* sera encapsulé dans un `Themes\Rozier\Explorer\AbstractExplorerItem` qui exposera vos données personnalisées de la bonne manière pour être affichées dans le back-office de Roadiz.

Vous trouverez un exemple d'implémentation dans Roadiz avec `Themes\Rozier\Explorer\SettingsProvider` et `Themes\Rozier\Explorer\SettingExplorerItem`. Ces classes ne récupèrent pas les données d'une API mais de votre base de données en utilisant `EntityListManager`.

Les types de Provider simple et multiple peuvent également accepter des options supplémentaires. Si vous voulez rendre votre provider configurable à l'exécution, vous pouvez passer des `options` dans la configuration de votre champ.

```
classname: Themes\MyTheme\Provider\ExternalApiProvider
options:
    - name: user
      value: me
    - name: access_token
      value: xxxxx
```

Ensuite, vous devez remplacer la méthode `configureOptions` de votre Provider pour définir quelles options sont autorisées.

```
use Symfony\Component\OptionsResolver\OptionsResolver;

/**
 * @param OptionsResolver $resolver
 */
public function configureOptions(OptionsResolver $resolver)
{
    $resolver->setDefaults([
        'page' => 1,
        'search' => null,
        'itemPerPage' => 30,
        // add more default options here
        'user' => 'me',
    ]);
    // You can required options
    $resolver->setRequired('access_token');
}
```

Collection personnalisée

Enfin, vous pouvez créer une collection de champs personnalisée pour y stocker des données en lecture seule en utilisant un `AbstractType` *Symfony* personnalisé.

Vous devez remplir le champ *valeurs par défaut* pour ce type.

```
# AbstractType class name
entry_type: Themes\MyTheme\Form\FooBarType
```

Veuillez noter que les données de *collection personnalisée* seront stockées en tant que tableau JSON dans votre base de données. Vous ne pourrez donc pas interroger votre source de noeud en utilisant ces données ou créer de filtres.

Dans votre `FooBarType`, vous pourrez utiliser des types de champs standards *Symfony* et des champs non-virtuels **Roadiz** tels que `MarkdownType`, `JsonType`, `YamlType`.

Handling nodes and their hierarchy

By default, if you use Entities API methods or traversing Twig filters, Roadiz will automatically handle security parameters such as `node.status` and `preview` mode.

```
// Secure method to get node-sources
// Implicitly check node.status
$this->nodeSourceApi->getBy([
    'node.nodeType' => $blogPostType,
    'translation' => $translation,
], [
    'publishedAt' => 'DESC'
]);
```

This first code snippet is using *Node-source API*. This will automatically check if current user is logged-in and if preview mode is *ON* to display or not *unpublished nodes*.

```
// Insecure method to get node-sources
// Doctrine raw method will get all node-sources
$this->managerRegistry->getRepository(NSBlogPost::class)->findBy([], [
    'publishedAt' => 'DESC',
```

```
    'translation' => $translation,
]);
```

This second code snippet uses standard Doctrine *Entity Manager* to directly grab node-sources by their entity class. This method does not check any security and will return every node-sources, **even unpublished, archived and deleted ones**.

Hierarchy

To traverse node-sources hierarchy, the easier method is to use *Twig* filters on your `nodeSource` entity. Filters will implicitly set `translation` from origin node-source.

```
{% set children = nodeSource|children %}
{% set nextSource = nodeSource|next %}
{% set prevSource = nodeSource|previous %}
{% set parent = nodeSource|parent %}

{% set children = nodeSource|children({
    'node.visible': true
}) %}
```

Avertissement : All these filters will take care of publication status and translation, **but not publication date-time neither visibility**.

```
{% set children = nodeSource|children({
    'node.visible': true,
    'publishedAt': ['>=', date()],
}, {
    'publishedAt': 'DESC'
}) %}

{% set nextVisible = nodeSource|next({
    'node.visible': true
}) %}
```

If you need to traverse node-source graph from your controllers you can use the *Entity API*. Moreover, Nodes-sources API allows you to filter using custom criteria if you choose a specific `NodeType`.

```
$children = $this->nodeSourceApi->getBy([
    'node.parent' => $nodeSource,
    'node.visible' => true,
    'publishedAt' => ['>=', new \DateTime()],
    'translation' => $nodeSource->getTranslation(),
],[
    'publishedAt' => 'DESC'
]);
```

Avertissement : Browsing your node graph (calling children or parents) could be very greedy and unoptimized if you have lots of node-types. Internally *Doctrine* will *inner-join* every nodes-sources tables to perform polymorphic hydration. So, make sure you filter your queries by one `NodeType` as much as possible with `nodeSourceApi` and `node.nodeType` criteria.

```
// Here Doctrine will only join NSPage table to NodesSources
$children =$this->nodeSourceApi->getBy([
    'node.nodeType' => $this->nodeTypesBag->get('Page'),
    'node.parent' => $nodeSource,
    'node.visible' => true,
    'publishedAt' => ['>=', new \DateTime()],
    'translation' => $nodeSource->getTranslation(),
],[
    'publishedAt' => 'DESC'
]);
```

Visibility

There are two parameters that you must take care of in your themes and your controllers, because they are not mandatory in all website cases :

— Visibility
— Publication date and time

For example, *publication date and time* won't be necessary in plain text pages and not timestampable contents. But we decided to add it directly in `NodesSources` entity to be able to filter and order with this field in Roadiz back-office. This was not possible if you manually create your own `publishedAt` as a node-type field.

Avertissement : Pay attention that *publication date and time* (`publishedAt`) and visibility (`node.visible`) **does not prevent** your node-source from being viewed if you did not explicitly forbid access to its controller. This field is not deeply set into Roadiz security mechanics.

If you need so, make sure that your node-type controller checks these two fields and throws a `ResourceNotFoundException` if they're not satisfied.

```
class BlogPostController extends MyAwesomeTheme
{
    public function indexAction(
        Request $request,
        Node $node = null,
        TranslationInterface $translation = null
    ) {
        $this->prepareThemeAssignation($node, $translation);

        $now = new DateTime("now");
        if (!$nodeSource->getNode()->isVisible() ||
            $nodeSource->getPublishedAt() < $now) {
            throw new ResourceNotFoundException();
        }

        return $this->render(
            'types/blogpost.html.twig',
            $this->assignation
        );
    }
}
```

Publication workflow

Each Node state is handled by a *Workflow* to switch between the following 5 states :

States

- `Node::DRAFT`
- `Node::PENDING`
- `Node::PUBLISHED`
- `Node::ARCHIVED`
- `Node::DELETED`

Transitions

- review
- reject
- publish
- archive
- unarchive
- delete
- undelete

You cannot changes a Node status directly using its *setter*, you must use Roadiz main *registry* to perform transition. This can prevent unwanted behaviours and you can track changes with events and guards :

```
$nodeWorkflow = $this->workflowRegistry->get($node);
if ($nodeWorkflow->can($node, 'publish')) {
    $nodeWorkflow->apply($node, 'publish');
}
```

Generating paths and url

You can use `generateUrl()` in your controllers to get a node-source' path or url. In your Twig template, you can use `path` method as described in Twig section : twig-generate-paths.

```
use Symfony\Cmf\Component\Routing\RouteObjectInterface;

class BlogPostController extends MyAwesomeTheme
{
    public function indexAction(
        Request $request,
        Node $node = null,
        TranslationInterface $translation = null
    ) {
        $this->prepareThemeAssignation($node, $translation);

        // Generate a path for current node-source
        $path = $this->generateUrl(
            RouteObjectInterface::OBJECT_BASED_ROUTE_NAME,
            [RouteObjectInterface::ROUTE_OBJECT => $this->nodeSource]
        );

        // Generate an absolute URL for current node-source
        $absoluteUrl =  $this->generateUrl(
            RouteObjectInterface::OBJECT_BASED_ROUTE_NAME,
            [RouteObjectInterface::ROUTE_OBJECT => $this->nodeSource],
            UrlGeneratorInterface::ABSOLUTE_URL
        );
    }
}
```

Overriding default node-source path generation

You can override default node-source path generation in order to use `{{ path() }}` method in your *Twig* templates but with a custom logic. For example, you have a `Link` node-type which purpose only is to link to an other node in your website. When you call *path* or *URL* generation on it, you should prefer getting its linked node path, so you can listen to `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPathGeneratingEvent:class` event and stop propagation to return your linked node path instead of your *link* node path.

```php
use GeneratedNodeSources\NSLink;
use Symfony\Component\EventDispatcher\EventDispatcherInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPathGeneratingEvent;

class LinkPathGeneratingEventListener implements EventSubscriberInterface
{
    public static function getSubscribedEvents(): array
    {
        return [
            NodesSourcesPathGeneratingEvent:class => ['onLinkPathGeneration']
        ];
    }

    /**
     * @param NodesSourcesPathGeneratingEvent $event
     * @param string                          $eventName
     * @param EventDispatcherInterface        $dispatcher
     */
    public function onLinkPathGeneration(
        NodesSourcesPathGeneratingEvent $event,
        $eventName,
        EventDispatcherInterface $dispatcher
    ) {
        $nodeSource = $event->getNodeSource();

        if ($nodeSource instanceof NSLink) {
            if (filter_var($nodeSource->getExternalUrl(), FILTER_VALIDATE_URL)) {
                /*
                 * If editor linked to an external link
                 */
                $event->stopPropagation();
                $event->setComplete(true);
                $event->setContainsScheme(true); // Tells router not to prepend
↪protocol and host to current URL
                $event->setPath($nodeSource->getExternalUrl());
            } elseif (count($nodeSource->getNodeReferenceSources()) > 0 &&
                null !== $linkedSource = $nodeSource->
↪getNodeReferenceSources()[0]) {
                /*
                 * If editor linked to an internal page through a node reference
                 */
                /** @var FilterNodeSourcePathEvent $subEvent */
                $subEvent = clone $event;
                $subEvent->setNodeSource($linkedSource);
                /*
                 * Dispatch a path generation again for linked node-source.
                 */
                $dispatcher->dispatch($subEvent);
                /*
                 * Fill main event with sub-event data
                 */
                $event->setPath($subEvent->getPath());
```

```
                $event->setComplete($subEvent->isComplete());
                $event->setParameters($subEvent->getParameters());
                $event->setContainsScheme($subEvent->containsScheme());
                // Stop propagation AFTER sub-event was dispatched not to prevent␣
↪it to perform.
                $event->stopPropagation();
            }
        }
    }
}
```

3.1.3 Building headless websites using API

Since Roadiz v2, headless development is default and is the most powerful way to build reactive websites and applications. Roadiz is built on API Platform, and it exposes all main entities as API Resources using DTO (data transfer objects)

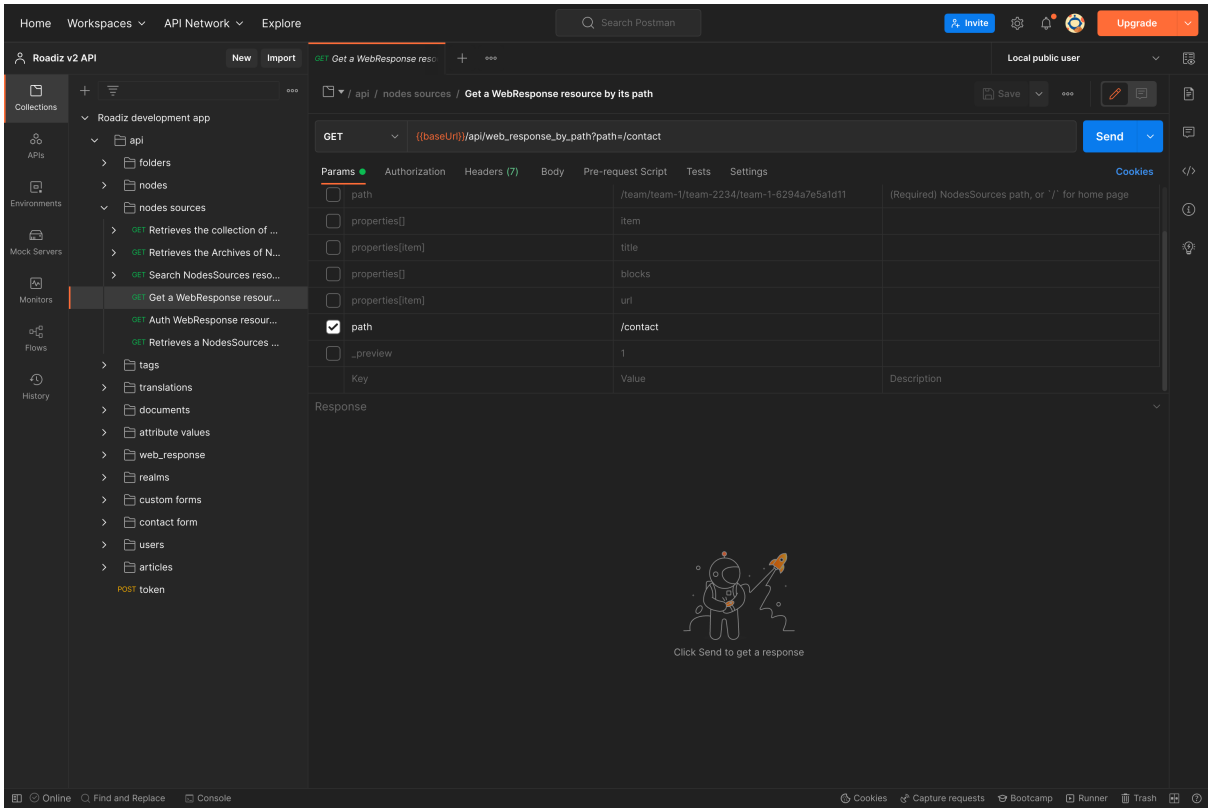


Fig. 3.1 – Consuming Roadiz API with Postman application is a great way to explore and test REST calls for your frontend app

WebResponse concept

A REST-ful API will expose collection and item entry-points for each resource. But in both case, you need to know your resource type or your resource identifier **before** executing your API call. Roadiz introduces a special resource named **WebResponse** which can be called using a `path` query param in order to reduce as much as possible API calls and address N+1 problem.

```
GET /api/web_response_by_path?path=/contact
```

API will expose a WebResponse single item containing :

— An item
— Item breadcrumbs
— Head object
— Item blocks tree-walker
— Item realms
— and if blocks are hidden by Realm configuration

Note : Roadiz *WebResponse* is used in Rezo Zero Nuxt Starter to populate all data during the `asyncData()` routine in `_.vue` page


```
{
    "@context": "/api/contexts/WebResponse",
    "@id": "/api/web_response_by_path?path=/contact",
    "@type": "WebResponse",
    "item": {
        "@id": "/api/pages/7",
        "@type": "Page",
        "content": "Magni deleniti ut eveniet. Aliquam aut et excepturi vitae␣
↪placeat molestiae. Molestiae asperiores nihil sed temporibus quibusdam. Non␣
↪magnam fuga at. sdf",
        "subTitle": null,
        "overTitle": null,
        "headerImage": [],
        "test": null,
        "pictures": [],
        "nodeReferences": [],
        "stickytest": false,
        "sticky": false,
        "customForm": [],
        "title": "Contact",
        "publishedAt": "2021-09-10T15:56:00+02:00",
        "metaTitle": "",
        "metaKeywords": "",
        "metaDescription": "",
        "users": [],
        "node": {
            "@type": "Node",
            "@id": "/api/nodes/7",
            "visible": true,
            "position": 3,
            "tags": []
        },
        "slug": "contact",
        "url": "/contact"
    },
    "breadcrumbs": {
        "@type": "Breadcrumbs",
        "@id": "_:14750",
        "items": []
    },
    "head": {
        "@type": "NodesSourcesHead",
        "@id": "_:14679",
        "googleAnalytics": null,
        "googleTagManager": null,
        "matomoUrl": null,
        "matomoSiteId": null,
        "siteName": "Roadiz dev website",
        "metaTitle": "Contact - Roadiz dev website",
        "metaDescription": "Contact, Roadiz dev website",
        "policyUrl": null,
```

```
        "mainColor": null,
        "facebookUrl": null,
        "instagramUrl": null,
        "twitterUrl": null,
        "youtubeUrl": null,
        "linkedinUrl": null,
        "homePageUrl": "/",
        "shareImage": null
    },
    "blocks": [],
    "realms": [],
    "hidingBlocks": false
}
```

Retrieve common content

Now that we can fetch each page data, we need to get all unique content for building Menus, Homepage reference, headers, footers, etc. We could extend our _WebResponse_ to inject theses common data to each request, but it would bloat HTTP responses, and affect API performances.

For these common content, you can create a `/api/common_content` API endpoint in your project which will fetched only once in your frontend application.

```
# config/api_resources/common_content.yml

App\Api\Model\CommonContent:
    collectionOperations: {}
    itemOperations:
        getCommonContent:
            method: 'GET'
            path: '/common_content'
            read: false
            controller: App\Controller\GetCommonContentController
            pagination_enabled: false
            normalization_context:
                pagination_enabled: false
                groups:
                    - get
                    - common_content
                    - web_response
                    - walker
                    - walker_level
                    - children
                    - children_count
                    - nodes_sources_base
                    - nodes_sources_default
                    - urls
                    - tag_base
                    - translation_base
                    - document_display
```

Then create you own custom resource to hold your menus tree-walkers and common content :

```
<?php

declare(strict_types=1);

namespace App\Controller;

use App\Model\CommonContent;
use App\TreeWalker\MenuNodeSourceWalker;
```

```php
use Doctrine\Persistence\ManagerRegistry;
use Psr\Cache\CacheItemPoolInterface;
use RZ\Roadiz\CoreBundle\Api\Model\NodesSourcesHeadFactory;
use RZ\Roadiz\Core\AbstractEntities\TranslationInterface;
use RZ\Roadiz\CoreBundle\Api\TreeWalker\AutoChildrenNodeSourceWalker;
use RZ\Roadiz\CoreBundle\Bag\Settings;
use RZ\Roadiz\CoreBundle\EntityApi\NodeSourceApi;
use RZ\Roadiz\CoreBundle\Preview\PreviewResolverInterface;
use RZ\Roadiz\CoreBundle\Repository\TranslationRepository;
use RZ\TreeWalker\WalkerContextInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\RequestStack;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;
use Symfony\Component\Routing\Exception\ResourceNotFoundException;

final class GetCommonContentController extends AbstractController
{
    private RequestStack $requestStack;
    private ManagerRegistry $managerRegistry;
    private WalkerContextInterface $walkerContext;
    private Settings $settingsBag;
    private NodeSourceApi $nodeSourceApi;
    private CacheItemPoolInterface $cacheItemPool;
    private NodesSourcesHeadFactory $nodesSourcesHeadFactory;
    private PreviewResolverInterface $previewResolver;

    public function __construct(
        RequestStack $requestStack,
        ManagerRegistry $managerRegistry,
        WalkerContextInterface $walkerContext,
        Settings $settingsBag,
        NodeSourceApi $nodeSourceApi,
        NodesSourcesHeadFactory $nodesSourcesHeadFactory,
        CacheItemPoolInterface $cacheItemPool,
        PreviewResolverInterface $previewResolver
    ) {
        $this->requestStack = $requestStack;
        $this->walkerContext = $walkerContext;
        $this->cacheItemPool = $cacheItemPool;
        $this->nodeSourceApi = $nodeSourceApi;
        $this->managerRegistry = $managerRegistry;
        $this->nodesSourcesHeadFactory = $nodesSourcesHeadFactory;
        $this->settingsBag = $settingsBag;
        $this->previewResolver = $previewResolver;
    }

    public function __invoke(): ?CommonContent
    {
        try {
            $request = $this->requestStack->getMainRequest();
            $translation = $this->getTranslationFromRequest($request);
            $home = $this->nodeSourceApi->getOneBy([
                'node.home' => true,
                'translation' => $translation
            ]);
            $mainMenu = $this->nodeSourceApi->getOneBy([
                'node.nodeName' => 'main-menu',
                'translation' => $translation
            ]);
            $footerMenu = $this->nodeSourceApi->getOneBy([
                'node.nodeName' => 'footer-menu',
                'translation' => $translation
```

```
            ]);
            $errorPage = $this->nodeSourceApi->getOneBy([
                'node.nodeName' => 'error-page',
                'translation' => $translation
            ]);

            $resource = new CommonContent();

            if (null !== $home) {
                $resource->home = $home;
            }
            if (null !== $mainMenu) {
                $resource->mainMenuWalker = MenuNodeSourceWalker::build(
                    $mainMenu,
                    $this->walkerContext,
                    3,
                    $this->cacheItemPool
                );
            }
            if (null !== $footerMenu) {
                $resource->footerMenuWalker = MenuNodeSourceWalker::build(
                    $footerMenu,
                    $this->walkerContext,
                    3,
                    $this->cacheItemPool
                );
            }
            if (null !== $footer) {
                $resource->footerWalker = AutoChildrenNodeSourceWalker::build(
                    $footer,
                    $this->walkerContext,
                    3,
                    $this->cacheItemPool
                );
            }
            if (null !== $errorPage) {
                $resource->errorPageWalker = AutoChildrenNodeSourceWalker::build(
                    $errorPage,
                    $this->walkerContext,
                    3,
                    $this->cacheItemPool
                );
            }
            if (null !== $request) {
                $request->attributes->set('data', $resource);
            }
            $resource->head = $this->nodesSourcesHeadFactory->createForTranslation(
↪$translation);
            return $resource;
        } catch (ResourceNotFoundException $exception) {
            throw new NotFoundHttpException($exception->getMessage(), $exception);
        }
    }

    protected function getTranslationFromRequest(?Request $request):␣
↪TranslationInterface
    {
        $locale = null;

        if (null !== $request) {
            $locale = $request->query->get('_locale');

            /*
```

```
             * If no _locale query param is defined check Accept-Language header
             */
            if (null === $locale) {
                $locale = $request->getPreferredLanguage($this->
↪getTranslationRepository()->getAllLocales());
            }
        }
        /*
         * Then fallback to default CMS locale
         */
        if (null === $locale) {
            $translation = $this->getTranslationRepository()->findDefault();
        } elseif ($this->previewResolver->isPreview()) {
            $translation = $this->getTranslationRepository()
                ->findOneByLocaleOrOverrideLocale((string) $locale);
        } else {
            $translation = $this->getTranslationRepository()
                ->findOneAvailableByLocaleOrOverrideLocale((string) $locale);
        }
        if (null === $translation) {
            throw new NotFoundHttpException('No translation for locale ' .
↪$locale);
        }
        return $translation;
    }

    protected function getTranslationRepository(): TranslationRepository
    {
        return $this->managerRegistry->getRepository(TranslationInterface::class);
    }
}
```

Then, the following resource will be exposed :

```
{
    "@context": "/api/contexts/CommonContent",
    "@id": "/api/common_content?id=unique",
    "@type": "CommonContent",
    "home": {
        "@id": "/api/pages/11",
        "@type": "Page",
        "content": null,
        "image": [],
        "title": "Accueil",
        "publishedAt": "2022-04-12T16:24:00+02:00",
        "node": {
            "@type": "Node",
            "@id": "/api/nodes/10",
            "visible": true,
            "tags": []
        },
        "slug": "accueil",
        "url": "/fr"
    },
    "mainMenuWalker": {
        "@type": "MenuNodeSourceWalker",
        "@id": "_:3341",
        "children": [],
        "childrenCount": 0,
        "item": {
            "@id": "/api/menus/2",
            "@type": "Menu",
            "title": "Menu principal",
```

```
            "publishedAt": "2022-04-12T00:39:00+02:00",
            "node": {
                "@type": "Node",
                "@id": "/api/nodes/1",
                "visible": false,
                "tags": []
            },
            "slug": "main-menu"
        },
        "level": 0,
        "maxLevel": 3
    },
    "footerMenuWalker": {
        "@type": "MenuNodeSourceWalker",
        "@id": "_:2381",
        "children": [],
        "childrenCount": 0,
        "item": {
            "@id": "/api/menus/3",
            "@type": "Menu",
            "linkInternalReference": [],
            "title": "Menu du pied de page",
            "publishedAt": "2022-04-12T11:18:12+02:00",
            "node": {
                "@type": "Node",
                "@id": "/api/nodes/2",
                "visible": false,
                "tags": []
            },
            "slug": "footer-menu"
        },
        "level": 0,
        "maxLevel": 3
    },
    "footerWalker": {
        "@type": "AutoChildrenNodeSourceWalker",
        "@id": "_:2377",
        "children": [],
        "childrenCount": 0,
        "item": {
            "@id": "/api/footers/16",
            "@type": "Footer",
            "content": "",
            "title": "Pied de page",
            "publishedAt": "2022-04-12T19:02:47+02:00",
            "node": {
                "@type": "Node",
                "@id": "/api/nodes/15",
                "visible": false,
                "tags": []
            },
            "slug": "footer"
        },
        "level": 0,
        "maxLevel": 3
    },
    "errorPageWalker": {
        "@type": "AutoChildrenNodeSourceWalker",
        "@id": "_:3465",
        "children": [],
        "childrenCount": 0,
        "item": {
            "@id": "/api/pages/153",
```

```
            "@type": "Page",
            "title": "Page d'erreur",
            "publishedAt": "2022-05-12T17:16:40+02:00",
            "node": {
                "@type": "Node",
                "@id": "/api/nodes/146",
                "visible": false,
                "tags": []
            },
            "slug": "error-page",
            "url": "/fr/error-page"
        },
        "level": 0,
        "maxLevel": 3
    },
    "head": {
        "@type": "NodesSourcesHead",
        "@id": "_:14679",
        "googleAnalytics": null,
        "googleTagManager": null,
        "matomoUrl": null,
        "matomoSiteId": null,
        "siteName": "Roadiz dev website",
        "metaTitle": "Contact - Roadiz dev website",
        "metaDescription": "Contact, Roadiz dev website",
        "policyUrl": null,
        "mainColor": null,
        "facebookUrl": null,
        "instagramUrl": null,
        "twitterUrl": null,
        "youtubeUrl": null,
        "linkedinUrl": null,
        "homePageUrl": "/",
        "shareImage": null
    }
}
```


Exposing node-types

All resources configuration files are located in `./config/api_resources` folder.

Here is an example of API resource for a `Page` node-type, you'll find default `collectionOperations` and `itemOperations` plus a special `getByPath` operation which allow overriding **WebResponse** serialization groups.

```
App\GeneratedEntity\NSPage:
    iri: Page
    shortName: Page
    collectionOperations:
        get:
            method: GET
            normalization_context:
                groups:
                    - nodes_sources_base
                    - nodes_sources_default
                    - urls
                    - tag_base
                    - translation_base
                    - document_display
    itemOperations:
        get:
```

```
            method: GET
            normalization_context:
                groups:
                    - nodes_sources
                    - urls
                    - tag_base
                    - translation_base
                    - document_display

        getByPath:
            method: GET
            normalization_context:
                groups:
                    - web_response
                    - position
                    - walker
                    - walker_level
                    - walker_metadata
                    - meta
                    - children
                    - children_count
                    - nodes_sources
                    - urls
                    - tag_base
                    - translation_base
                    - document_display
```

To automatically generate your resources YAML configuration files, execute the following CLI command :

```
bin/console generate:api-resources
```

Extending serialization

Imagine you created a *BlogPost* node-type, and you want to give users a preview of your latest BlogPosts in your website home page. You would create :

— A `BlogPost` node-type
— and a `BlogPostFeedBlock` node-type
— and your `Page` node-type would have a `blocks` field allowing `BlogPostFeedBlock` creation

When you hit `/api/web_response_by_path?path=/` you should expect your `WebResponse` blocks section to expose your `BlogPostFeedBlock`. But how do you expose automatically the latest 3 blog-posts ? You could add a node-reference field in your `BlogPostFeedBlock` and ask your editor to **manually** expose 3 blog-posts. But this won't be automatic at all.

Using a custom Normalizer

One solution is to extend `BlogPostFeedBlock` normalization stage. With this technique you will be able to inject additional data into your serialized object.

First, create a Normalizer which will **decorate** Roadiz and API Platform default item normalizers :

```
<?php

declare(strict_types=1);

namespace App\Serializer\Normalizer;

use App\GeneratedEntity\NSBlogPostFeedBlock;
use App\GeneratedEntity\NSBlogPost;
use Doctrine\ORM\Tools\Pagination\Paginator;
```

```php
use RZ\Roadiz\CoreBundle\EntityApi\NodeSourceApi;
use RZ\Roadiz\CoreBundle\Serializer\Normalizer\AbstractPathNormalizer;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
use Symfony\Component\Serializer\Normalizer\NormalizerInterface;

final class BlogPostFeedBlockNormalizer extends AbstractPathNormalizer
{
    public NodeSourceApi $nodeSourceApi;

    public function __construct(
        NodeSourceApi $nodeSourceApi,
        NormalizerInterface $decorated,
        UrlGeneratorInterface $urlGenerator
    ) {
        parent::__construct($decorated, $urlGenerator);
        $this->nodeSourceApi = $nodeSourceApi;
    }

    /**
     * @param mixed $object
     * @param string|null $format
     * @param array $context
     * @return array|\ArrayObject|bool|float|int|mixed|string|null
     * @throws \Symfony\Component\Serializer\Exception\ExceptionInterface
     */
    public function normalize($object, $format = null, array $context = [])
    {
        $data = $this->decorated->normalize($object, $format, $context);
        if (
            $object instanceof NSBlogPostFeedBlock &&
            is_array($data)
        ) {
            // fetch latest 3 blog-posts from DB
            $blogPosts = $this->nodeSourceApi->getBy(
                [
                    'node.nodeType.name' => 'BlogPost',
                    'node.visible' => true,
                    'publishedAt' => ['<=', new \DateTime()]
                ],
                [
                    'publishedAt' => 'DESC'
                ],
                3
            );

            if ($blogPosts instanceof Paginator) {
                $blogPosts = $blogPosts->getIterator()->getArrayCopy();
            }

            $data['blogPosts'] = array_map(
                function (NSBlogPost $blogPost) use ($format, $context) {
                    // Call decorated normalizer to normalize your BlogPosts
                    // Be careful to set properly your service decoration priority!
                    return $this->decorated->normalize($blogPost, $format,
↪$context);
                },
                $blogPosts
            );
        }
        return $data;
    }
}
```

Then register your custom normalizer in your app `services.yaml` with a very low decoration priority setting to be sure your normalizer will be decorating **after** Roadiz and API Platform normalizers.

```
# config/services.yaml
services:
    # ...
    App\Serializer\Normalizer\BlogPostFeedBlockNormalizer:
        decorates: 'api_platform.jsonld.normalizer.item'
        # normalizer must be decorating after Roadiz and API Platform normalizers
        decoration_priority: -1
    # Need a different name to avoid duplicate YAML key
    app.serializer.normalizer.blog_post_feed_block.json:
        class: 'App\Serializer\Normalizer\BlogPostFeedBlockNormalizer'
        decorates: 'api_platform.serializer.normalizer.item'
        # normalizer must be decorating after Roadiz and API Platform normalizers
        decoration_priority: -1
```

If your service is well registered, you should see your `BlogPostFeedBlock` exposing a new `blogPosts` virtual field with your latest 3 blog-posts :


```
{
    "@context": "/api/contexts/WebResponse",
    "@id": "/api/web_response_by_path?path=/",
    "@type": "WebResponse",
    "item": {},
    "breadcrumbs": {},
    "head": {},
    "blocks": [
        {
            "@type": "AutoChildrenNodeSourceWalker",
            "children": [],
            "item": {
                "@id": "/api/blog_posts_feed_blocks/23",
                "@type": "BlogPostFeedBlock",
                "title": "My latest blog-posts",
                "blogPosts": [
                    {
                        "@id": "/api/blog_posts/22",
                        "@type": "BlogPost",
                        "title": "My super blog post",
                        "publishedAt": "2022-10-14T11:41:15+02:00",
                        "slug": "super-blog-post"
                    },
                    {
                        "@id": "/api/blog_posts/21",
                        "@type": "BlogPost",
                        "title": "My super blog post, but older",
                        "publishedAt": "2022-10-14T11:41:13+02:00",
                        "slug": "super-blog-post-but-older"
                    },
                    {
                        "@id": "/api/blog_posts/20",
                        "@type": "BlogPost",
                        "title": "My super blog post, but older again",
                        "publishedAt": "2022-10-13T11:41:13+02:00",
                        "slug": "super-blog-post-but-older-again"
                    }
                ]
            }
        }
    ]
}
```

Serialization groups

Roadiz CMS uses `symfony/serializer` to perform JSON serialization over any objects, especially *Doctrine* entities.

id Serialize every entity `id`.

timestamps Serialize every date-timed entity `createdAt` and `updatedAt` fields.

position Serialize every entity `position` fields.

color Serialize every entity `color` fields.

nodes_sources Serialize entities in a `NodesSources` context (all fields).

nodes_sources_base Serialize entities in a `NodesSources` context, but with essential information.

nodes_sources_documents Serialize documents linked to a `NodesSources` for each virtual field.

nodes_sources_default Serialize `NodesSources` fields not contained in any **group**.

nodes_sources_‘‘group‘‘ Custom serialization groups are created according to your node-typ fields groups. For example, if you set a field to a `link` group, `nodes_sources_link` serialization group will be automatically generated for this field. *Be careful*, Roadiz will use groups *canonical names* to generate serialization groups, it can mix _ and -.

node Serialize entities in a `Node` context.

node_children Serialize Nodes with their `children`.

node_attributes Serialize Nodes with their `attribute values`.

tag Serialize Tag.

tag_base Serialize Tags with minimum information.

tag_parent Serialize Tags with their `parent`.

tag_children Serialize Tags with their `children`, do not use with `tag_parent` group.

node_type Serialize entities in a `NodeType` context.

attribute Serialize entities in a `Attribute` context.

attribute_documents Serialize documents linked to a `Attribute` for each virtual field.

custom_form Serialize entities in a `CustomForm` context.

document Serialize entities in a `Document` context.

document_display Serialize `Document` information required for displaying them.

document_private Serialize `Document` privacy information

document_display_sources Serialize `Document` information required for displaying alternative sources (audio, video).

document_thumbnails Serialize `Document` first available thumbnail.

document_folders Serialize `Document` information required for displaying attached **visible** folders.

document_folders_all Serialize **all** `Document` attached folders, even not visible ones.

folder Serialize entities in a `Folder` context.

translation Serialize entities in a `Translation` context.

translation_base Serialize `Translation` information required for displaying them.

setting Serialize entities in a `Setting` context.

setting_group Serialize entities in a `SettingGroup` context.

user Serialize entities in a `User` context.

user_group Serialize User entity with its groups.

user_role Serialize User entity with its roles.

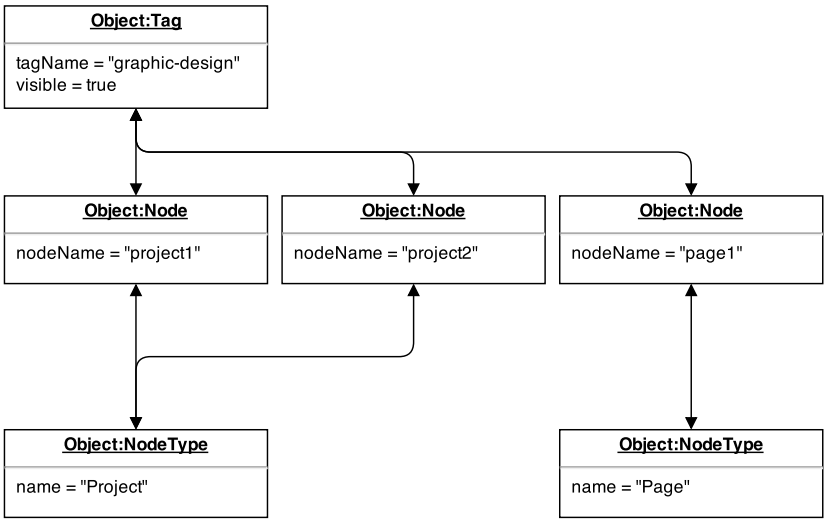
user_personal Serialize User entity with its personal information.

user_identifier Serialize User entity with its identifier (may be a personal information).

We recommend using Rezo Zero Nuxt starter to build your frontend applications. This starter is built to use Roadiz API and relies on dynamic routing and supports API redirections.

3.1.4 Tag system

Nodes are essentially hierarchical entities. So we created an entity to link nodes between them no matter where/what they are. Tags are meant as *tag* nodes, we couldn't be more explicit. But if you didn't understand here is a schema :

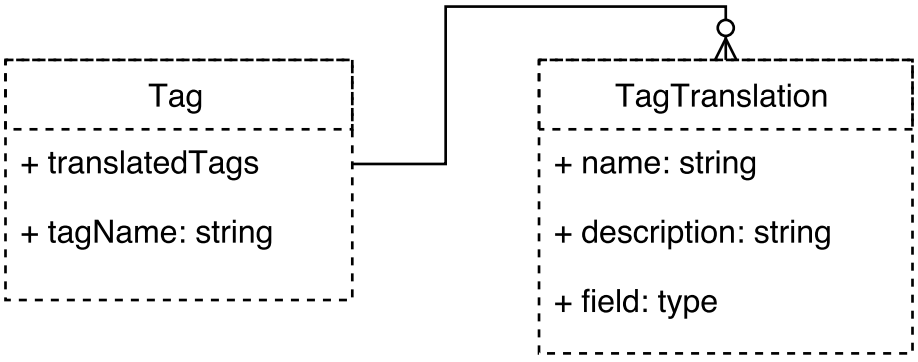


You can see that tags can gather heterogeneous nodes coming from different types (pages and projects). Tags can be used to display a category-navigation on your theme or to simply tidy your backoffice node database.

Did you notice that `Tags` are related to `Nodes` entities, not `NodesSources` ? We thought that it would be easier to manage that way not to forget to tag a specific node translation. It means that you won't be able to differentiate tag two `NodesSources`, if you absolutely need to, we encourage you to create two different nodes.

Translate tags

You will notice that tags work the same way as nodes do. By default, *tags names* can't contain special characters in order to be used in URLs. So we created `TagTranslation` entities which stand for Tag's sources :



In that way you will be able to translate your tags for each available languages and link documents to them.

Tag hierarchy

In the same way as *Nodes* work, tags can be nested to create *tag groups*.

Exposing tags in API

When using API Platform data transfer objects, Tags are ready-to-use with translations set-up on `name` and `description` fields :

```
{
    "@type": "Tag",
    "@id": "/api/tags/6",
    "slug": "event",
    "name": "Event",
    "description": null,
    "color": "#000000",
    "visible": true,
    "documents": [],
    "parent": {
        "@type": "Tag",
        "@id": "/api/tags/3",
        "slug": "type",
        "name": "Type",
        "description": null,
        "color": "#000000",
        "visible": true,
        "documents": [],
        "parent": null
    }
}
```

Displaying node-source tags with Twig

Tag translations are already set up to track your current locale if you fetched them using `|tags` *Twig* filter. Simply use `getTranslatedTags()->first()` Tag method to use them in your templates.

```
{% set tags = nodeSource|tags %}

<ul>
{% for tag in tags %}
    {% set tagTranslation = tag.translatedTags.first %}
    <li id="{{ tag.tagName }}">{{ tagTranslation.name }}</li>
{% endfor %}
</ul>
```

Tags translations documents

Documents can be linked to your tag translations. They will be different for each translation, so make sure to synchronize them manually if you want to use the same document for all translations.

They are available with `getDocuments()` method and will be ordered by **position only**.

Imagine, you want to link a PDF document for each of your tags, you can create a download link as described below :

```
{% set tags = nodeSource|tags %}
<ul>
{% for tag in tags %}
    {% set tagTranslation = tag.translatedTags.first %}
    <li id="{{ tag.tagName }}">
        <p>{{ tagTranslation.name }}</p>
        {% if tagTranslation.documents[0] %}
            <a href="{{ tagTranslation.documents[0]|url }}" class="tag-document">{
↪% trans %}download_tag_pdf{% endtrans %}</a>
        {% endif %}
    </li>
{% endfor %}
</ul>
```

3.1.5 Documents system

Exposing documents in API

When using API Platform data transfer objects, Documents are ready-to-use with translations set-up on `name` and `description` fields. Made sure to configure your API operations with at least `document_display` serialization group :


```
{
    "@type": "Document",
    "@id": "/api/documents/xxxxx",
    "relativePath": "xxxxxxx/my_image.jpg",
    "type": "image",
    "mimeType": "image/jpeg",
    "name": null,
    "description": null,
    "embedId": null,
    "embedPlatform": null,
    "imageAverageColor": "#141414",
    "imageWidth": 1000,
    "imageHeight": 750,
    "mediaDuration": 0,
    "copyright": "© John Doe",
    "externalUrl": null,
    "processable": true,
    "thumbnail": null,
    "alt": "This is an image"
}
```


Expose document thumbnails

Thumbnails are exposed by default for each document, it is useful when documents are not displayable : PDF, native video, ZIP, etc :


```
{
    "@type": "Document",
    "@id": "/api/documents/xxxxx",
    "relativePath": "xxxxxxx/img_2004_framed_1080p_2000.webm",
    "type": "video",
    "mimeType": "video/webm",
    "name": null,
    "description": null,
    "embedId": null,
    "embedPlatform": null,
    "imageAverageColor": null,
    "imageWidth": 1920,
    "imageHeight": 1080,
    "mediaDuration": 14,
    "copyright": null,
    "externalUrl": null,
    "processable": false,
    "thumbnail": {
        "@type": "Document",
        "@id": "/api/documents/xxxxx",
        "relativePath": "xxxxxxx/img_2004_framed_1080p_2000.png",
        "type": "image",
        "mimeType": "image/png",
        "name": null,
        "description": null,
        "embedId": null,
        "embedPlatform": null,
```

```
        "imageAverageColor": "#917357",
        "imageWidth": 2662,
        "imageHeight": 1504,
        "mediaDuration": 0,
        "copyright": null,
        "externalUrl": null,
        "processable": true,
        "thumbnail": null,
        "alt": "img_2004_framed_1080p_2000.png"
    },
    "alt": "img_2004_framed_1080p_2000.webm"
}
```


Expose document alternative sources

Alternative sources are not serialized by default for performance matters, but you can enable them in your project. Add `document_display_sources` serialization group to your resource configuration.


```
{
    "@type": "Document",
    "@id": "/api/documents/xxxxx",
    "relativePath": "xxxxxxxx/img_2004_framed_1080p_2000.webm",
    "type": "video",
    "mimeType": "video/webm",
    "name": null,
    "description": null,
    "embedId": null,
    "embedPlatform": null,
    "imageAverageColor": null,
    "imageWidth": 1920,
    "imageHeight": 1080,
    "mediaDuration": 14,
    "copyright": null,
    "externalUrl": null,
    "processable": false,
    "thumbnail": {
        "@type": "Document",
        "@id": "/api/documents/xxxxx",
        "relativePath": "xxxxxxxx/img_2004_framed_1080p_2000.png",
        "type": "image",
        "mimeType": "image/png",
        "name": null,
        "description": null,
        "embedId": null,
        "embedPlatform": null,
        "imageAverageColor": "#917357",
        "imageWidth": 2662,
        "imageHeight": 1504,
        "mediaDuration": 0,
        "copyright": null,
        "externalUrl": null,
        "processable": true,
        "thumbnail": null,
        "alt": "img_2004_framed_1080p_2000.png"
    },
    "altSources": [
        {
            "@type": "Document",
            "@id": "/api/documents/xxxxx",
            "relativePath": "xxxxxxxx/img_2004_framed_1080p_2000.mp4",
            "type": "video",
```

```
            "mimeType": "video/mp4",
            "name": null,
            "description": null,
            "embedId": null,
            "embedPlatform": null,
            "imageAverageColor": null,
            "imageWidth": 0,
            "imageHeight": 0,
            "mediaDuration": 0,
            "copyright": null,
            "externalUrl": null,
            "processable": false,
            "thumbnail": null,
            "alt": "img_2004_framed_1080p_2000.mp4"
        }
    ],
    "alt": "img_2004_framed_1080p_2000.webm"
}
```

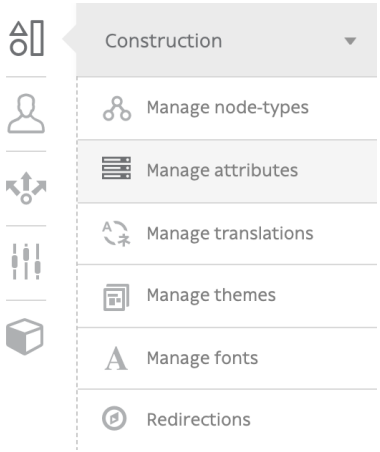

Expose document folders

Document folders are not serialized by default for performance matters, but you can enable them in your project. Add `document_folders` serialization group to your resource configuration.


```
{
    "@type": "Document",
    "@id": "/api/documents/3436",
    "relativePath": "xxxxxxxx/youtube_wplj0yxcnwk.jpg",
    "type": "image",
    "mimeType": "image/jpeg",
    "name": "Shirine - Bande annonce",
    "description": "",
    "embedId": "wPlj0YxCNwk",
    "embedPlatform": "youtube",
    "imageAverageColor": "#2d2426",
    "imageWidth": 200,
    "imageHeight": 113,
    "mediaDuration": 0,
    "copyright": "Opéra de Lyon (https://www.youtube.com/user/OperadeLyon)",
    "externalUrl": null,
    "processable": true,
    "thumbnail": null,
    "folders": [
        {
            "@type": "Folder",
            "@id": "/api/folders/20",
            "slug": "danse",
            "name": "Danse",
            "visible": true
        },
        {
            "@type": "Folder",
            "@id": "/api/folders/31",
            "slug": "opera-inside",
            "name": "Opera-inside",
            "visible": false
        }
    ],
    "alt": "Shirine - Bande annonce"
}
```

3.1.6 Attributs

Les attributs sont des entités destinées à qualifier des caractéristiques sur d'autres entités, telles que les *nœuds*. La différence principale entre les *tags* et les *attributs* est que vous pouvez définir une valeur pour chaque *noeud* attribué.



L'attribut « Color » peut être défini à « rouge » pour un noeud et défini à « vert » pour un autre.

Ok, mais maintenant quelle est la différence entre les champs *attributs* et node-type ? Il n'y en a pas tellement, parce que les champs de type de nœud décrivent également les caractéristiques de votre nœud, mais ils sont figés et définis par le développeur. Une fois vos champs de type de nœud créés, vous devez implémenter votre fonctionnalité dans vos gabarits *Twig* et le traduire dans vos fichiers XLF.

Les attributs sont destinés à être créés et ajoutés par les éditeurs afin qu'ils puissent les utiliser dans le cycle de vie de votre site sans avoir besoin de développement supplémentaire.

Du point de vue des développeurs, les attributs ne sont qu'une collection d'entités à afficher au sein d'une boucle. Ensuite, votre éditeur peut en créer de nouveaux et être sûr qu'ils seront affichés sans aucun développement supplémentaire.

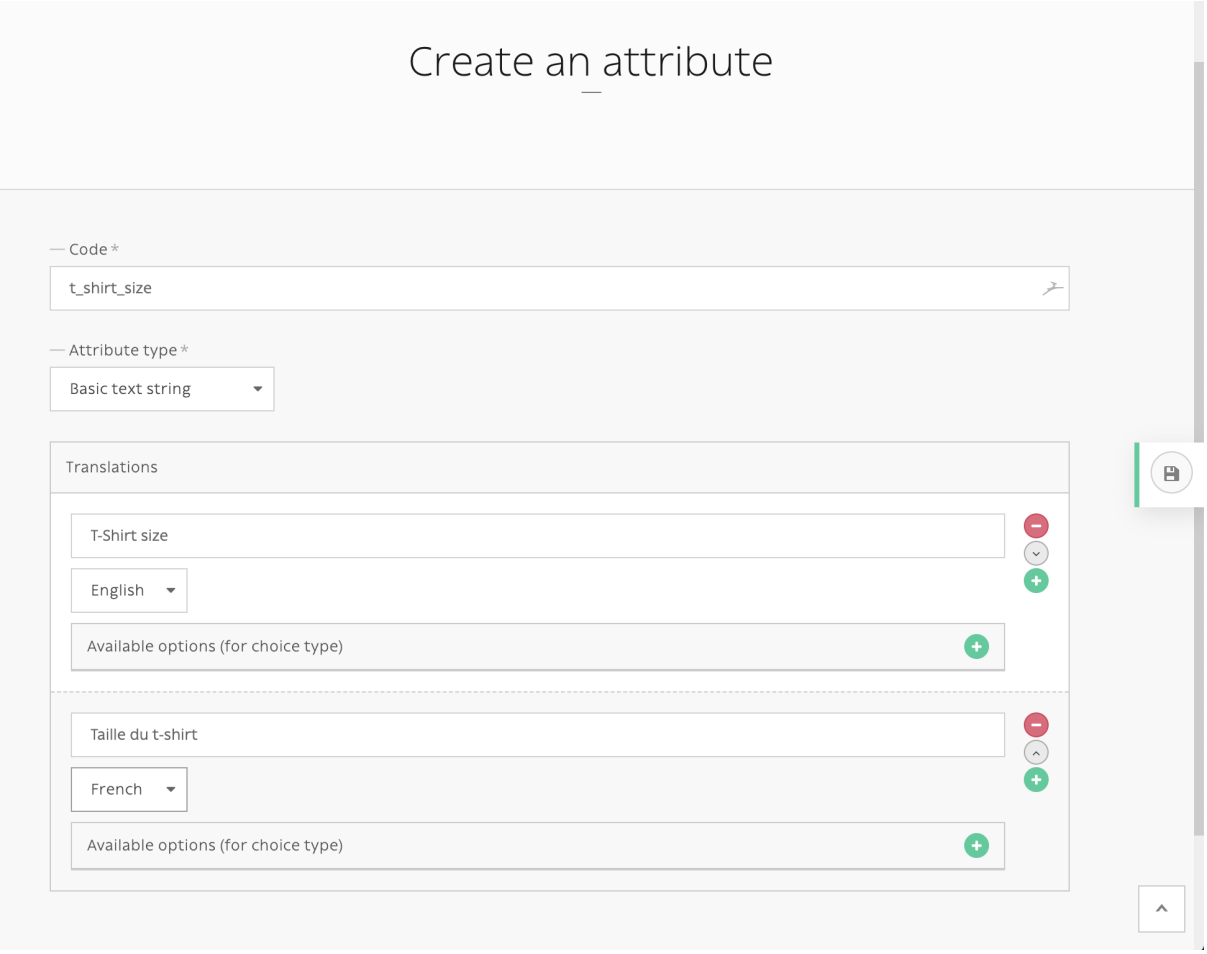
```
<ul class="block-attributes">
    {% for attributeValueTranslation in nodeSource|attributes %}
        <li>
            {% if attributeValueTranslation.attribute.documents|length %}
                {% for document in attributeValueTranslation.attribute.documents %}
                    <figure>{{ document|display }}</figure>
                {% endfor %}
            {% endif %}
            <strong>{{ attributeValueTranslation|attribute_label(translation) }}:</
↪strong>
            {% if attributeValueTranslation is datetime %}
                {{ attributeValueTranslation.value|format_datetime('medium', 'short
↪', locale=app.request.locale) }}
            {% elseif attributeValueTranslation is date %}
                {{ attributeValueTranslation.value|format_date('medium',␣
↪locale=app.request.locale) }}
            {% elseif attributeValueTranslation is country %}
                {{ attributeValueTranslation.value|country_name(request.locale) }}
            {% else %}
                {{ attributeValueTranslation.value }}
            {% endif%}
        </li>
    {% endfor %}
</ul>
```

Si vous avez regroupé vos attributs, vous pouvez utiliser le filtre `grouped_attributes` à la place :

```
<ul class="block-attributes">
    {% for item in nodeSource|grouped_attributes %}
        <li>
            {% if item.group %}
                <strong>{{ item.group|attribute_group_label(translation) }}</
↪strong>
            {% endif %}
            <ul>
                {% for attributeValueTranslation in item.attributeValues %}
                    <li>
                        {% if attributeValueTranslation.attribute.documents|length
↪%}
                            {% for document in attributeValueTranslation.attribute.
↪documents %}
                                <figure>{{ document|display }}</figure>
                            {% endfor %}
                        {% endif %}
                        <strong>{{ attributeValueTranslation|attribute_
↪label(translation) }}:</strong>
                        {% if attributeValueTranslation is datetime %}
                            {{ attributeValueTranslation.value|format_datetime(
↪'medium', 'short', locale=app.request.locale) }}
                        {% elseif attributeValueTranslation is date %}
                            {{ attributeValueTranslation.value|format_date('medium
↪', locale=app.request.locale) }}
                        {% elseif attributeValueTranslation is country %}
                            {{ attributeValueTranslation.value|country_
↪name(request.locale) }}
                        {% else %}
                            {{ attributeValueTranslation.value }}
                        {% endif%}
                    </li>
                {% endfor %}
            </ul>

        </li>
    {% endfor %}
</ul>
```

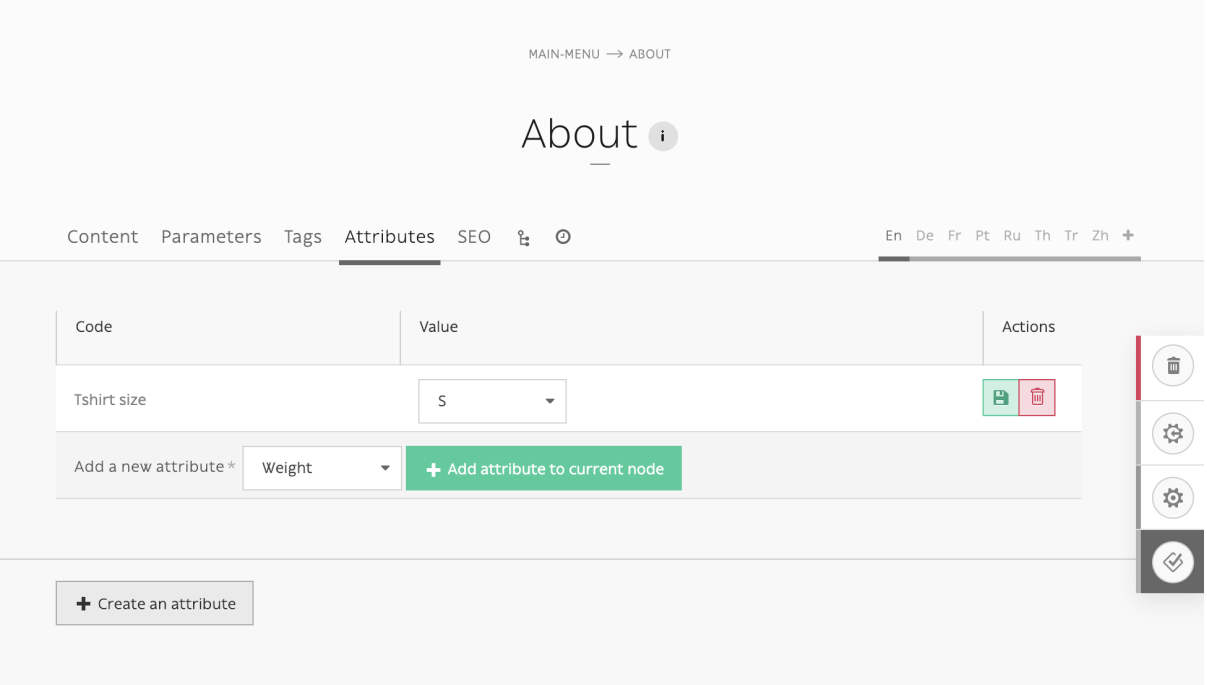
Types d'attributs



- *Chaîne de caractères*
- *Date*
- *Date et heure*
- *Choix simple* : choix parmi les options définies dans votre attribut
- *Booléen*
- *Nombre entier*
- *Nombre décimal*
- *Adresse e-mail*
- *Couleur*
- *Pays* : Code pays ISO 2 lettres

Ajouter des attributs aux nœuds

La section *Attribute* est disponible pour n'importe quel nœud pour chaque traduction.



Extension Twig

Plusieurs filtres et tests sont disponibles pour faciliter le templating avec des *attributs* :

Filtres

- `attributes` : identique à la méthode `node_source_attributes()`, récupère tous les attributs disponibles depuis une `NodesSources`.
- `grouped_attributes` : identique à la méthode `node_source_grouped_attributes()`, récupère tous les attributs disponibles depuis une `NodesSources` et les récupère dans leur **groupe**.
- `attribute_label(translation)` : récupère l'attribut traduit `label` ou `code` s'il n'est pas traduit.
- `attribute_group_label(translation)` : récupère le groupe d'attributs traduit `name` ou `canonicalName` s'il n'est pas traduit.

Tests

- `datetime`
- `date`
- `country`
- `boolean`
- `choice`
- `enum`

3.1.7 Forms

Roadiz uses *Symfony* forms logic and API. However, we made ready-made contact and custom forms builders to ease up your development and even make form-building available for your website editors.

Building contact forms

With Roadiz you can easily create simple contact forms with `ContactFormManager` class. Your controller has a convenient shortcut to create this manager with `$this->createContactFormManager()` method.

If you want to add your own fields, you can use the manager' form-builder with `$contactFormManager->getFormBuilder();`. Then add your field using standard *Symfony* form syntax. Do not forget to use *Constraints* to handle errors.

One contact-form for one action

Here is an example to create your contact form in your controller action.

```
use Symfony\Component\Validator\Constraints\File;
use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\Extension\Core\Type\SubmitType;

// ...
// Create contact-form manager and add 3 default fields.
$contactFormManager = $this->createContactFormManager()
                           ->withDefaultFields();
/*
 * (Optional) Add custom fields...
 */
$formBuilder = $contactFormManager->getFormBuilder();
$formBuilder->add('callMeBack', CheckboxType::class, [
        'label' => 'call.me.back',
        'required' => false,
    ])
    ->add('document', FileType::class, [
        'label' => 'document',
        'required' => false,
        'constraints' => [
            new File([
                'maxSize' => $contactFormManager->getMaxFileSize(),
                'mimeTypes' => $contactFormManager->getAllowedMimeTypes(),
            ]),
        ]
    ])
    ->add('send', SubmitType::class, [
        'label' => 'send.contact.form',
    ]);

/*
 * This is the most important point. handle method will perform form
 * validation and send email.
 *
 * Handle method should return a Response object if everything is OK.
 */
if (null !== $response = $contactFormManager->handle()) {
    return $response;
}

$form = $contactFormManager->getForm();
```

In this example, we used `withDefaultFields` method which add automatically `email`, `name` and `message` fields with right validation constraints. This method is optional and you can add any field you want manually, just keep in mind that you should always ask for an `email`.

Then in your contact page Twig template :

```twig
{#
 # Display contact errors
 #}
{% for label, messages in app.flashes(['warning', 'error']) %}
    {% for message in messages %}
        <p class="alert alert-{{ label }}">
            {{- message -}}
        </p>
    {% endfor %}
{% endfor %}
{#
 # Display contact form
 #}
{% form_theme contactForm '@MyTheme/forms.html.twig' %}
{{ form(contactForm) }}
```

Using contact-form in *block* controllers

If you want to use *contact-forms* in blocks instead of a full page, you will need to make your redirection response **bubble** through *Twig* render. The only way to stop Twig is to **throw an exception** and to pass your Redirect or Json response within your Exception.

Roadiz makes this possible with `RZ\Roadiz\CoreBundle\Exception\ForceResponseException`. For example, in a `App/Controllers/ContactBlockController`, instead of returning the `contactFormManager` response, you will have to throw a `ForceResponseException` with it as an argument.

```php
// ./src/Controllers/ContactBlockController.php

use RZ\Roadiz\CoreBundle\Exception\ForceResponseException;

...
// Create contact-form manager and add 3 default fields.
$contactFormManager = $this->createContactFormManager()
                           ->withDefaultFields();

if (null !== $response = $contactFormManager->handle()) {
    /*
     * Force response to bubble through Twig rendering process.
     */
    throw new ForceResponseException($response);
}

$form = $contactFormManager->getForm();
```

Then, in your *master* controller (i.e. `PageController`), `render` method will automatically catch your *ForceResponseException* exception in order to extract the forced response object. Then it will return your response instead of your page twig rendered output.

Alter email configuration after form submit

If you want to customize emails sent with form data, you can use Symfony form events to change the contact form manager options : The following example alters the email subject to add the user email and makes the subject unique for the receiver.

```php
$formBuilder->addEventListener(FormEvents::SUBMIT, function (FormEvent $event) use
↪($contactFormManager) {
   $contactFormManager->setSubject($contactFormManager->getSubject() . ': ' .
↪$event->getForm()->get('email')->getData());
```

```
});
```

You can also use this behaviour to change dynamically the contact form receiver after the user chose it in a select box input.

Note : You can read more about form events at https://symfony.com/doc/current/form/events.html

Securing your form with *Google reCAPTCHA*

Roadiz can seamlessly use *Google reCAPTCHA* to secure your contact form against robots. All you need to do is to register on https://www.google.com/recaptcha/ to ask for a *sitekey* and a *secret*. Once you've got these two keys, add them to your Roadiz settings.

Then, just use `withGoogleRecaptcha()` method on your contact-form manager.

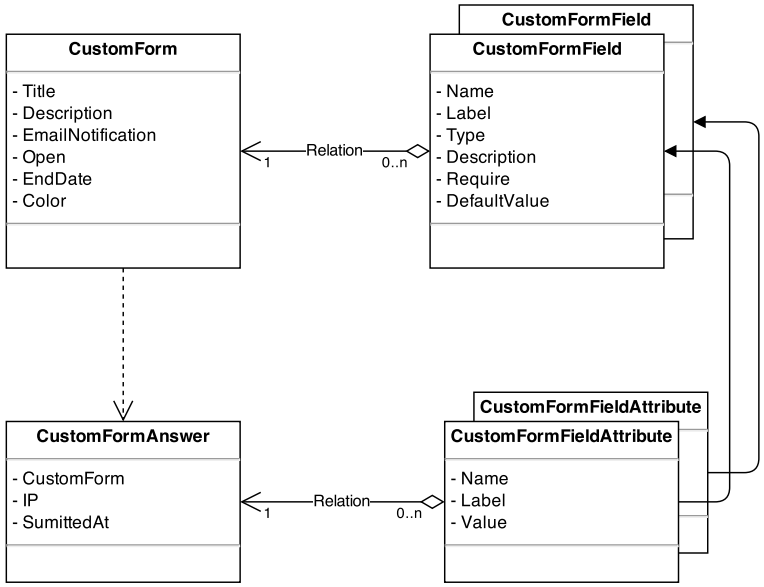
```
// Create contact-form manager, add 3 default fields and add a reCAPTCHA.
$contactFormManager = $this->createContactFormManager()
                           ->withDefaultFields()
                           ->withGoogleRecaptcha();
```

Do not forget to add recaptcha form-template and to embed google's javascript.

```
 {# In your theme' forms.html.twig file #}
 {% block recaptcha_widget -%}
    <input id="my-form-recaptcha" type="hidden" name="{{ form.vars.name }}" />
    <script src="https://www.google.com/recaptcha/api.js?render={{ configs.
↪publicKey }}"></script>
    <script>
        /*
         * Google Recaptcha v3
         * @see https://developers.google.com/recaptcha/docs/v3
         */
        (function() {
            if (!window.grecaptcha) {
                console.warn('Recaptcha is not loaded');
            }
            var form = document.getElementById('my-form');
            form.addEventListener('submit', function (event) {
                event.preventDefault();
                window.grecaptcha.ready(function() {
                    window.grecaptcha.execute('{{ configs.publicKey }}', {action:
↪'submit'}).then(function(token) {
                        var input = document.getElementById('my-form-recaptcha');
                        if (input) {
                            input.value = token;
                        }
                        form.submit()
                    });
                });
            });
        })();
    </script>
{%- endblock recaptcha_widget %}
```

Building custom forms

Building a custom form looks like building a node but it is a lot simpler ! Let's have a look at structure image.



After creating a custom form, you add some question. The questions are the CustomFormField type.

The answer is saved in two entities :

— in CustomFormAnswer
— in CustomFormFieldAttribute

The CustomFormAnswer will store the IP and the submitted time. While question answer will be in CustomFormFieldAttribute with the CustomFormAnswer id and the CustomFormField id.

Exposing a custom form in your API

Custom-form can be filled in a headless context, using _definition_ and _post_ endpoints :

```
GET {{baseUrl}}/api/custom_forms/:id/definition
```

Custom form definition is a **JSON form schema** meant to give your frontend application a recipe to build a HTML form :

```
{
    "title": "",
    "type": "object",
    "properties": {
        "subject": {
            "type": "string",
            "title": "Subject",
            "attr": {
                "data-group": null,
                "placeholder": null
            },
            "description": "Est aut quas eum error architecto.",
```

```
            "propertyOrder": 1
        },
        "email": {
            "type": "string",
            "title": "Email",
            "attr": {
                "data-group": null,
                "placeholder": null
            },
            "description": "Email address",
            "widget": "email",
            "propertyOrder": 2
        },
        "test": {
            "title": "TEST",
            "type": "object",
            "properties": {
                "message": {
                    "type": "string",
                    "title": "Message",
                    "attr": {
                        "data-group": "TEST",
                        "placeholder": null
                    },
                    "widget": "textarea",
                    "propertyOrder": 1
                },
                "fichier": {
                    "type": "string",
                    "title": "File",
                    "attr": {
                        "data-group": "TEST",
                        "placeholder": null
                    },
                    "widget": "file",
                    "propertyOrder": 2
                }
            },
            "required": [
                "fichier"
            ],
            "attr": {
                "data-group-wrapper": "test"
            },
            "propertyOrder": 3
        }
    },
    "required": [
        "subject",
        "email",
        "test"
    ]
}
```

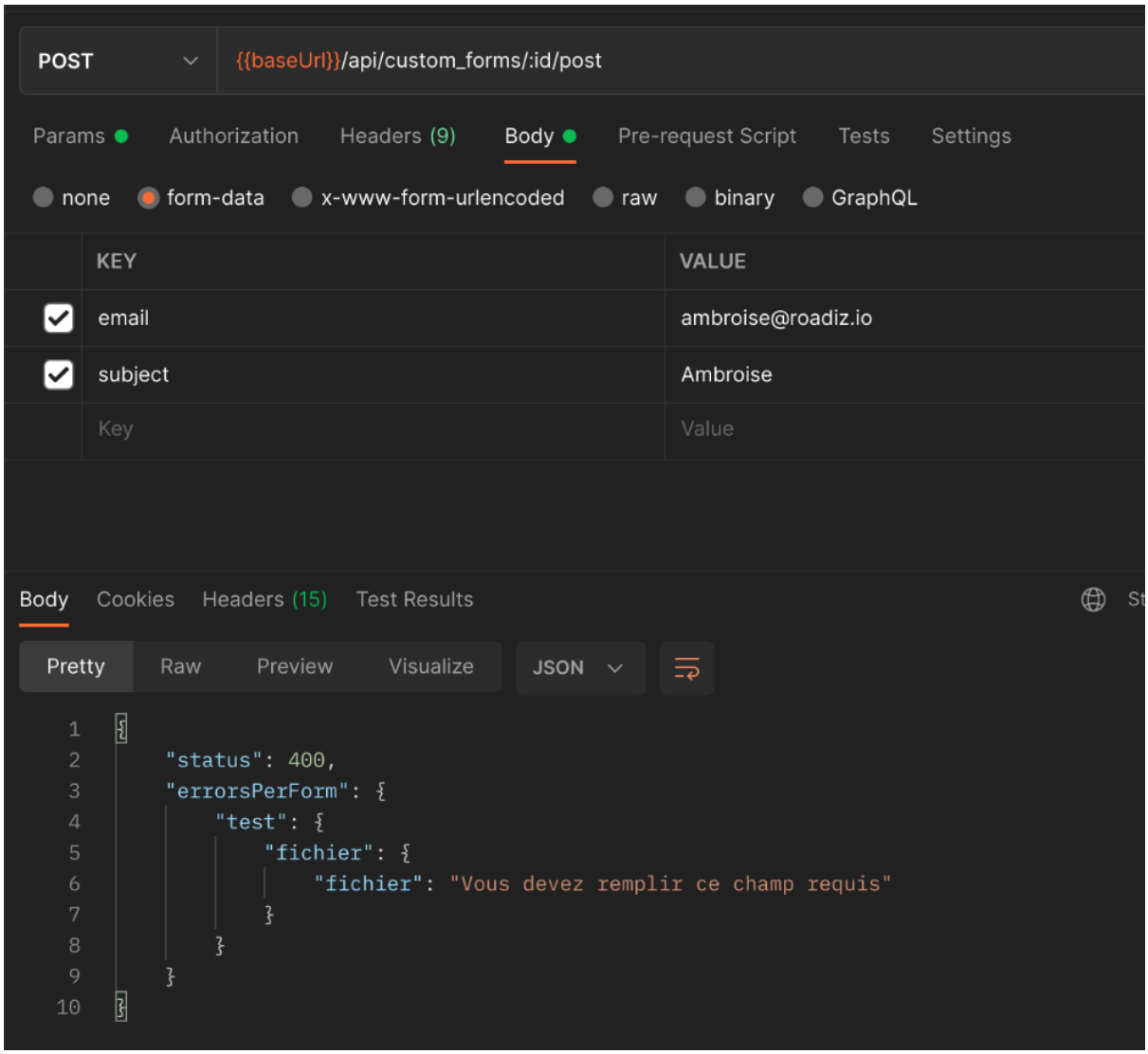
Then you can send your data to the **post** endpoint using *FormData* and respecting field hierarchy :

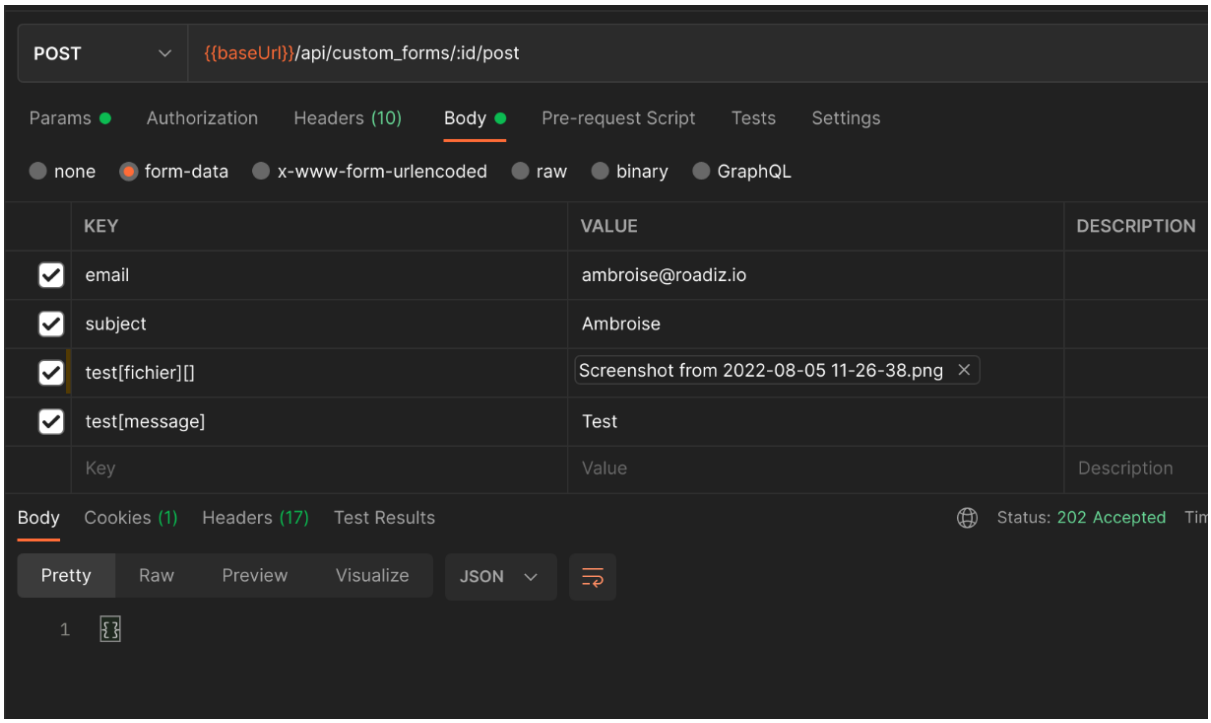
```
POST {{baseUrl}}/api/custom_forms/:id/post
```

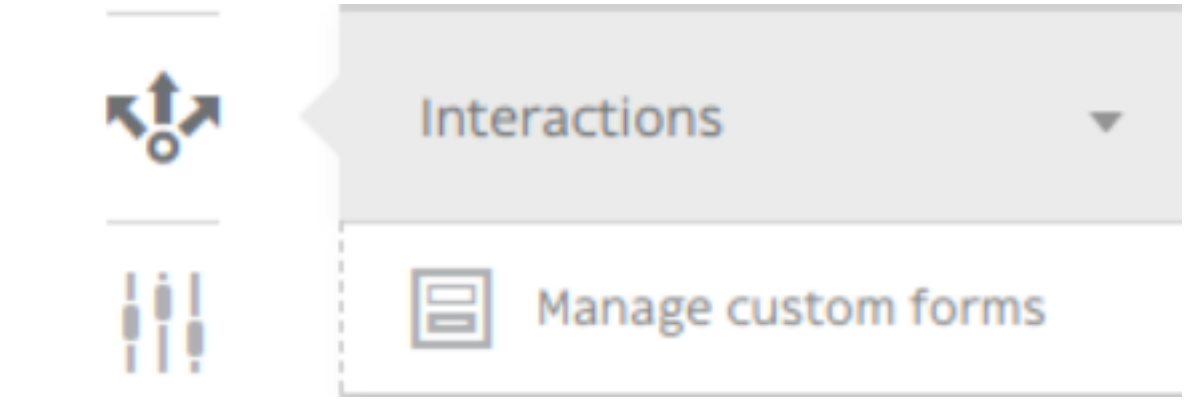
If there are any error, a *JSON* response will give you details fields-by-fields.

If post is successful, APi will respond an empty `202 Accepted` response

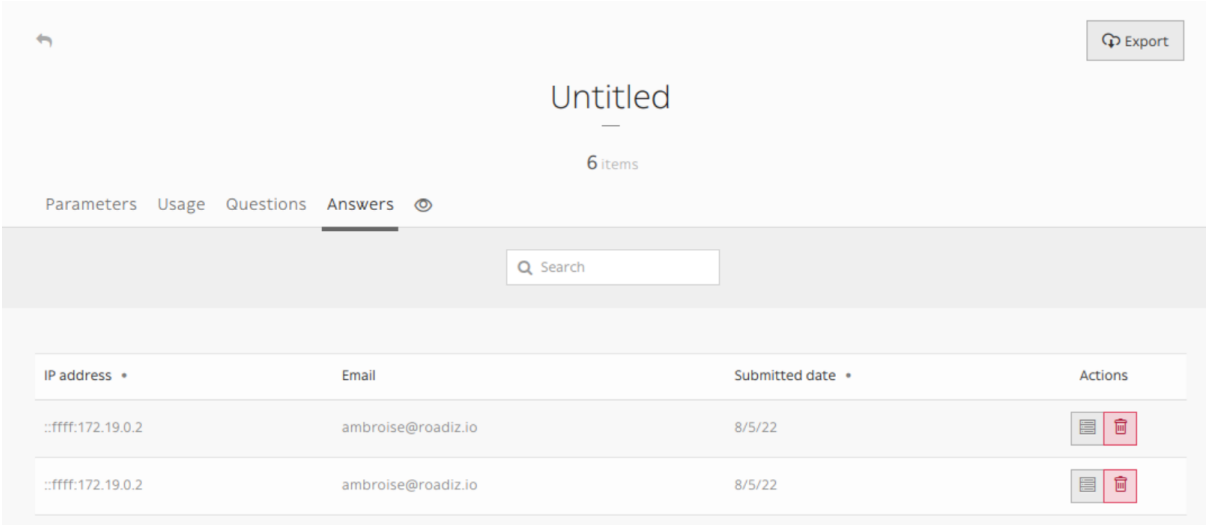
Then you will be able to see all your form submits in Roadiz backoffice :


POST
{{baseUrl}}/api/custom_forms/:id/post
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
KEY
VALUE
email
ambroise@roadiz.io
subject
Ambroise
Key
Value
Body
Cookies
Headers (15)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"status": 400,
"errorsPerForm": {
"test": {
"fichier": {
"fichier": "Vous devez remplir ce champ requis"
}
}
}
}

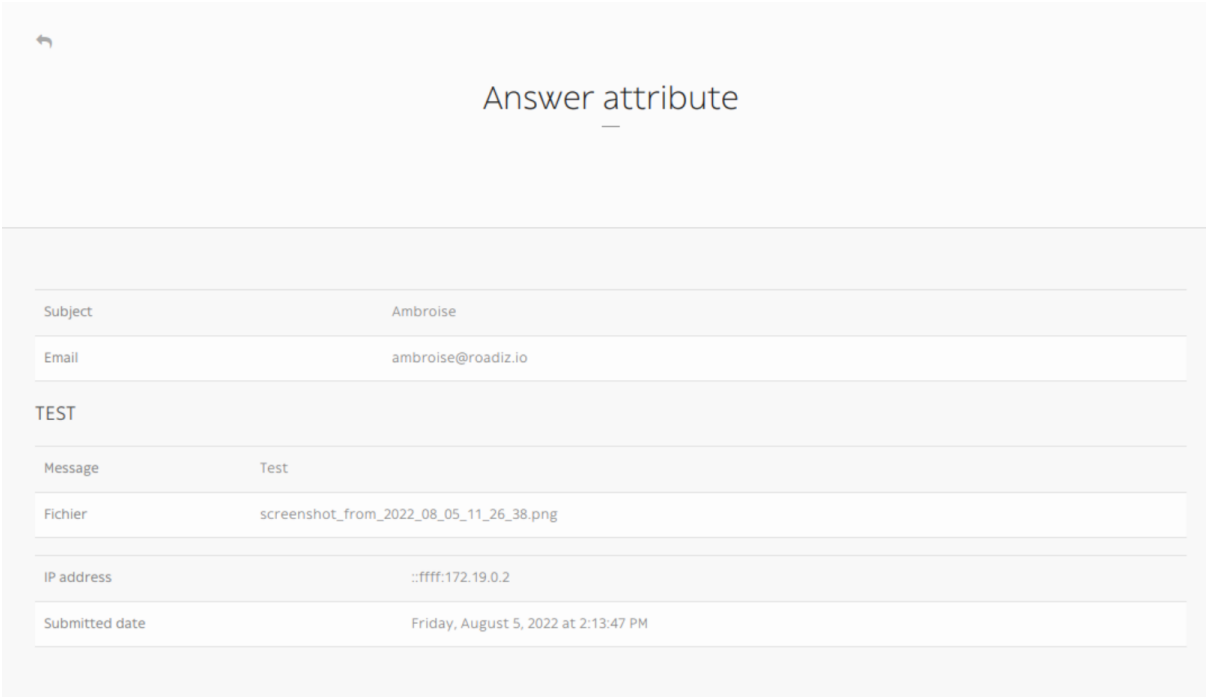

POST
{{baseUrl}}/api/custom_forms/:id/post
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
KEY
VALUE
DESCRIPTION
email
ambroise@roadiz.io
subject
Ambroise
test[fichier][]
Screenshot from 2022-08-05 11-26-38.png
test[message]
Test
Key
Value
Description
Body
Cookies (1)
Headers (17)
Test Results
Status: 202 Accepted
Pretty
Raw
Preview
Visualize
JSON
{}



In Manage custom forms section / Answers



Note : Any file attached to your custom-form answers will be uploaded as private documents.



Adding custom form to your theme

If you want to integrate your custom-forms into your theme, you can use Roadiz `CustomFormHelper` class to generate a standard `FormInterface` and to create a view into your theme templates.

First you must create a dedicated action for your node or your block if you used `{{ nodeSource|render(@AwesomeTheme) }}` Twig filter.

```php
use RZ\Roadiz\CoreBundle\Entity\CustomForm;
use RZ\Roadiz\CoreBundle\Exception\EntityAlreadyExistsException;
use RZ\Roadiz\CoreBundle\Exception\ForceResponseException;
use Symfony\Cmf\Component\Routing\RouteObjectInterface;
use RZ\Roadiz\CoreBundle\CustomForm\CustomFormHelper;
use Symfony\Component\Form\FormError;
use Symfony\Component\HttpFoundation\JsonResponse;

// ...

/*
 * Get your custom form instance from your node-source
 * only if you added a *custom_form reference field*.
 */
$customForms = $this->nodeSource->getCustomFormReference();
if (isset($customForms[0]) && $customForms[0] instanceof CustomForm) {
    /** @var CustomForm $customForm */
    $customForm = $customForms[0];

    /*
     * Verify if custom form is still open
     * for answers
     */
    if ($customForm->isFormStillOpen()) {
        /*
         * CustomFormHelper will generate Symfony form against
         * Roadiz custom form entity.
         * You can add a Google Recaptcha passing following options.
         */
        $helper = $this->customFormHelperFactory->createHelper($customForm);
        $form = $helper->getForm($request, false, true);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            try {
                $answer = $helper->parseAnswerFormData($form, null, $request->
↪getClientIp());

                if ($request->isXmlHttpRequest()) {
                    $response = new JsonResponse([
                        'message' => $this->getTranslator()->trans('form_has_been_
↪successfully_sent')
                    ]);
                } else {
                    $this->publishConfirmMessage(
                        $request,
                        $this->getTranslator()->trans('form_has_been_successfully_
↪sent')
                    );
                    $response = $this->redirect($this->generateUrl(
                        RouteObjectInterface::OBJECT_BASED_ROUTE_NAME,
                        [RouteObjectInterface::ROUTE_OBJECT => $this->nodeSource->
↪getParent()]
                    ));
                }
```

```
                /*
                 * If you are in a BlockController use ForceResponseException
                 */
                throw new ForceResponseException($response);
                /*
                 * Or directly return redirect response.
                 */
                //return $response;
            } catch (EntityAlreadyExistsException $e) {
                $form->addError(new FormError($e->getMessage()));
            }
        }

        $this->assignation['form'] = $form->createView();
    }
}
```

If you didn't do it yet, create a custom form theme in your `views/` folder :

```
{#
 # AwesomeTheme/Resources/views/form.html.twig
 #}
{% extends "bootstrap_3_layout.html.twig" %}

{% block form_row -%}
    <div class="form-group form-group-{{ form.vars.block_prefixes[1] }} form-group-{{ form.vars.name }}">
        {% if form.vars.block_prefixes[1] != 'separator' %}
            {{- form_label(form) -}}
        {% endif %}
        {{- form_errors(form) -}}
        {#
         # Render field description inside your form
         #}
        {% if form.vars.attr['data-description'] %}
            <div class="form-description">
                {{ form.vars.attr['data-description']|markdown }}
            </div>
        {% endif %}
        {{- form_widget(form) -}}
    </div>
{%- endblock form_row %}

{% block recaptcha_widget -%}
   <input id="my-form-recaptcha" type="hidden" name="{{ form.vars.name }}" />
   <script src="https://www.google.com/recaptcha/api.js?render={{ configs.publicKey }}"></script>
   <script>
       /*
        * Google Recaptcha v3
        * @see https://developers.google.com/recaptcha/docs/v3
        */
       (function() {
           if (!window.grecaptcha) {
               console.warn('Recaptcha is not loaded');
           }
           var form = document.getElementById('my-form');
           form.addEventListener('submit', function (event) {
               event.preventDefault();
               window.grecaptcha.ready(function() {
                   window.grecaptcha.execute('{{ configs.publicKey }}', {action: 'submit'}).then(function(token) {
                       var input = document.getElementById('my-form-recaptcha');
```

```
                        if (input) {
                            input.value = token;
                        }
                        form.submit()
                    });
                });
            });
        })();
    </script>
{%- endblock recaptcha_widget %}
```

In your main view, add your form and use your custom form theme :

```
{#
 # AwesomeTheme/Resources/views/form-blocks/customformblock.html.twig
 #}
{% if form %}
    {% form_theme form '@AwesomeTheme/form.html.twig' %}
    {{ form_start(form) }}
    {{ form_widget(form) }}
    <div class="form-group">
        <button class="btn btn-primary" type="submit">{% trans %}send_form{% endtrans %}</button>
    </div>
    {{ form_end(form) }}
{% else %}
    <p class="alert alert-warning">{% trans %}form_is_not_available{% endtrans %}</p>
{% endif %}
```

3.1.8 Contributing

If you want to contribute to Roadiz project by reporting issues or hacking code, let us thank you ! You are awesome !

Reporting issues

When you encounter an issue with Roadiz we would love to hear about it. Because thanks to you, we can make the most awesome and stable CMS ! If you submit a bug report please include all information available to you, here are some things you can do :

— Try to simplify the things you are doing until getting a minimal set of actions reproducing the problem.
— Do not forget to join a screenshot or a trace of your error.

Coding style

The code you contributed to the project should respect the guidelines defined in PHP *PSR2* standard. If you install the requirements for devs by the command `composer update --dev`, you can use *phpcs* to check your code. You can copy and paste the following command-lines to check easily :

```
php bin/phpcs --report=full --report-file=./report.txt -p ./
```

Or you can use *phpcbf* to automatically fix code style issues.

```
php bin/phpcbf --report=full --report-file=./report.txt -p ./
```

Please take those rules into account, we aim to have a clean codebase. A coherent code-style will contribute to Roadiz stability. Your code will be checked when we will be considering your pull requests.

Static analysis

Then we use `phpstan` as a static code analyzer to check bugs and misuses before they occur :

```
php bin/phpstan analyse -c phpstan.neon
```

3.1.9 Troubleshooting

Empty caches manually for an environment

If you experience errors only on a dedicated environment such as `prod``or ``dev`, it means that cache is not fresh for these environments. As a first try, you should always call `bin/console cache:clear;` (replace *prod* by your environment) in command line.

Problem with entities and Doctrine cache ?

After each Roadiz **upgrade** you should always upgrade your node-sources entity classes and upgrade database schema.

```
bin/console generate:nsentities;
bin/console doctrine:schema:update --dump-sql --force;
bin/console cache:clear;
```

CHAPITRE 4

Extensions - Documentation

4.1 Extensions - Documentation

4.1.1 Extending Roadiz

Add back-office entry

At first, create a controller into your theme folder, for example `src/Controller/Admin/AdminController`.

Example :

```
namespace App\Controller\Admin;

use Themes\Rozier\RozierApp;
use Symfony\Component\HttpFoundation\Request;

class AdminController extends RozierApp
{
    public function listAction(
        Request $request
    ) {
        return $this->render(
            'admin/test.html.twig',
            $this->assignation
        );
    }
}
```

If you look at this example you can see the class extends `RozierApp` class. This will enable you to "inject" your code into Rozier Back-stage DOM and Style.

Now let's have a look to your twig template file `templates/admin/test.html.twig`.

```
{% extends '@Rozier/layout.html.twig' %}

{% block customStyles %}
<style>
    /* Custom styles here */
```

```
</style>
{% endblock %}

{% block customScripts %}
<script>
    /* Custom Scripts here */
</script>
{% endblock %}

{% block content %}
<section class="content-global add-test">
    <header class="content-header header-test header-test-edit">
        <h1 class="content-title test-add-title">{% trans %}Test admin{% endtrans
↪%}</h1>
    </header>

    <article class="content content-test">
        <p>This page is created from MyTheme to show you how to extend backoffice␣
↪features.</p>
    </article>
</section>
{% endblock %}
```

The first line is for inheriting from Rozier base template, you can notice that we explicitly choose *@Rozier* namespace.

The two next blocks are made for you to add some CSS or Javascript. For CSS, the block `customStyle` can be use to link an external file with a `<link>` tag, the path must be something like that `{{ asset('static/css/customstyle.css', 'MyTheme') }}`, or add directly some CSS with « <style> » tag. For JS, the block `customScripts` work as is, just link an external JS file or write your `<script>` tag.

Then create your own content, do not hesitate to give a look at Rozier back-stage theme Twig files to use the right DOM structure. For simple features, you wouldn't have to extend JS nor CSS if you follow the same HTML coding style.

Linking things together

Add the route in the theme `config/routes.yaml` file.

In this case the route will be :

```
adminTestPage:
    # Setting your path behind rz-admin will activate Firewall
    path: /rz-admin/test
    defaults:
        _controller: App\Controller\Admin\AdminController::listAction
```

Inject your own entries in back-stage

The last thing to do is to add your new admin entry in the back-office menu.

Go to your `config/packages/roadiz_rozier.yaml` and add your own `entries` :

```
roadiz_rozier:
    entries:
        # ...
        customAdmin:
            name: customAdmin
            route: adminTestPage
            icon: uk-icon-cube
            roles: ~
```

If you want to have a category and sub-entries, just change the path at `null` value and create your `subentries` array as described in the next example :

```
roadiz_rozier:
    entries:
        # ...
        customAdmin:
            name: customAdmin
            route: ~
            icon: uk-icon-cube
            roles: ~
            subentries:
                customAdminPage:
                    name: 'customAdmin page'
                    route: adminTestPage
                    icon: uk-icon-cube
                    roles: ~
```

You can restrict buttons to users with specific roles. Just replace `roles: ~` with `roles: [ 'ROLE_ACCESS_NODES' ]`. You can even create your own roles to take full power of Roadiz extension system.

Avertissement : Adding roles in `roadiz_rozier.entries` service will only restrict buttons display in Rozier backstage interface. To really protect your controllers from unwanted users add `$this->validateAccessForRole('ROLE_ACCESS_MY_FEATURE');` at the first line of your back-ofice controller‘s actions. This will kick non-granted users from your custom back-office parts. Give a look at Rozier theme controllers to see how we use it.

4.1.2 Events

Roadiz node system implements several events. So you will be able to create and inject your own event subscribers inside *Roadiz* dispatcher.

To understand how the event dispatcher works, you should read the Symfony documentation at before.

Nodes events

- `RZ\Roadiz\CoreBundle\Event\Node\NodeCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeDeletedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeUndeletedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeDuplicatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodePathChangedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\Node\NodeTaggedEvent::class` : This event is triggered for tag and un-tag action.
- `RZ\Roadiz\CoreBundle\Event\Node\NodeVisibilityChangedEvent::class` : This event is triggered each time a node becomes visible or unvisible.
- `RZ\Roadiz\CoreBundle\Event\Node\NodeStatusChangedEvent::class` : This event is triggered each time a node status changes.

Each node event object contains the current `Node` entity. You will get it using `$event->getNode()`.

NodesSources events

`RZ\Roadiz\CoreBundle\Event\NodesSourcesEvents`
- `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesCreatedEvent::class`

— `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPreUpdatedEvent::class`: This event is dispatched BEFORE entity manager FLUSHED.
— `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesUpdatedEvent::class`: This event is dispatched AFTER entity manager FLUSHED.
— `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesDeletedEvent::class`

Each node-source event object contains the current `NodesSources` entity. You will get it using `$event->getNodeSource()`.

— `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesIndexingEvent::class`: This event type is dispatched during Solr indexation. Your event will be `\RZ\Roadiz\CoreBundle\Event\FilterSolariumNodeSourceEvent` and it will allow you to alter or improve your Solr index according to your node-source type.

Note : You will find a simple subscriber example in Roadiz back-office theme which is called `Themes\Rozier\Events\SolariumSubscriber`. This subscriber is useful to update or delete your *Solr* index documents against your node-source database.

— `RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesPathGeneratingEvent::class`: This event type is dispatched when the node-router generate a path for your node-source using `{{ path() }}` Twig method or `$this->urlGenerator->generate()` controller method. The default subscriber will generate the complete hierarchical path for any node-source using their identifier (available url-alias or node' name).

Tags events

— `RZ\Roadiz\CoreBundle\Event\Tag\TagCreatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\Tag\TagUpdatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\Tag\TagDeletedEvent::class`

Each tag event object contains the current `Tag` entity. You will get it using `$event->getTag()`.

Folders events

— `RZ\Roadiz\CoreBundle\Event\Folder\FolderCreatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\Folder\FolderUpdatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\Folder\FolderDeletedEvent::class`

Each folder event object contains the current `Folder` entity. You will get it using `$event->getFolder()`.

Translations events

— `RZ\Roadiz\CoreBundle\Event\Translation\TranslationCreatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\Translation\TranslationUpdatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\Translation\TranslationDeletedEvent::class`

Each folder event object contains the current `Translation` entity. You will get it using `$event->getTranslation()`.

UrlAlias events

— `RZ\Roadiz\CoreBundle\Event\UrlAlias\UrlAliasCreatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\UrlAlias\UrlAliasUpdatedEvent::class`
— `RZ\Roadiz\CoreBundle\Event\UrlAlias\UrlAliasDeletedEvent::class`

Each folder event object contains the current `UrlAlias` entity. You will get it using `$event->getUrlAlias()`.

User events

- `RZ\Roadiz\CoreBundle\Event\User\UserCreatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserUpdatedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserDeletedEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserDisabledEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserEnabledEvent::class`
- `RZ\Roadiz\CoreBundle\Event\User\UserPasswordChangedEvent::class`

Each folder event object contains the current `User` entity. You will get it using `$event->getUser()`.

4.1.3 Extending Solr indexation

How to index page blocks contents

If all your text content is written in *block nodes* instead of reachable *pages*, you should index them into your page Solr documents to improve your search engine relevancy.

You can use the `NodesSourcesIndexingEvent::class` event to enhance your node indexing data before it's persisted into *Solr* engine (especially `collection_txt` field) :

```php
<?php

declare(strict_types=1);

namespace App\EventSubscriber;

use RZ\Roadiz\CoreBundle\Api\TreeWalker\AutoChildrenNodeSourceWalker;
use RZ\Roadiz\CoreBundle\Entity\NodesSources;
use RZ\Roadiz\CoreBundle\Event\NodesSources\NodesSourcesIndexingEvent;
use RZ\Roadiz\CoreBundle\SearchEngine\SolariumFactoryInterface;
use RZ\TreeWalker\WalkerContextInterface;
use RZ\TreeWalker\WalkerInterface;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;

/**
 * Index sub nodes content into any reachable node-source.
 */
final class NodeSourceIndexingEventSubscriber implements EventSubscriberInterface
{
    private WalkerContextInterface $walkerContext;
    private SolariumFactoryInterface $solariumFactory;
    private int $maxLevel;

    /**
     * @param WalkerContextInterface $walkerContext
     * @param SolariumFactoryInterface $solariumFactory
     * @param int $maxLevel
     */
    public function __construct(
        WalkerContextInterface $walkerContext,
        SolariumFactoryInterface $solariumFactory,
        int $maxLevel = 5
    ) {
        $this->walkerContext = $walkerContext;
        $this->solariumFactory = $solariumFactory;
        $this->maxLevel = $maxLevel;
    }

    /**
     * @inheritDoc
```

```php
     */
    public static function getSubscribedEvents(): array
    {
        return [
            NodesSourcesIndexingEvent::class => ['onIndexing'],
        ];
    }

    public function onIndexing(NodesSourcesIndexingEvent $event): void
    {
        $nodeSource = $event->getNodeSource();

        if (null !== $nodeSource->getNode() && $nodeSource->isReachable() && !
↪$event->isSubResource()) {
            $assoc = $event->getAssociations();

            $blockWalker = AutoChildrenNodeSourceWalker::build(
                $nodeSource,
                $this->walkerContext,
                $this->maxLevel
            );

            // Need a locale field
            $locale = $nodeSource->getTranslation()->getLocale();
            $lang = \Locale::getPrimaryLanguage($locale) ?? 'fr';

            foreach ($blockWalker->getChildren() as $subWalker) {
                $this->walkAndIndex($subWalker, $assoc, $lang);
            }

            $event->setAssociations($assoc);
        }
    }

    /**
     * @param WalkerInterface $walker
     * @param array $assoc
     * @param string $locale
     * @throws \Exception
     */
    protected function walkAndIndex(WalkerInterface $walker, array &$assoc, string
↪$locale): void
    {
        $item = $walker->getItem();
        if ($item instanceof NodesSources) {
            $solarium = $this->solariumFactory->createWithNodesSources($walker->
↪getItem());
            // Fetch all fields array association AS sub-resources (i.e. do not␣
↪index their title)
            $childAssoc = $solarium->getFieldsAssoc(true);
            $assoc['collection_txt'] = array_filter(array_merge(
                $assoc['collection_txt'],
                $childAssoc['collection_txt']
            ));
            if (!empty($childAssoc['collection_txt_' . $locale])) {
                $assoc['collection_txt_' . $locale] .= PHP_EOL . $childAssoc[
↪'collection_txt_' . $locale];
            }
        }
        if ($walker->count() > 0) {
            foreach ($walker->getChildren() as $subWalker) {
                $this->walkAndIndex($subWalker, $assoc, $locale);
            }
```

```
            }
        }
}
```

Index