Roadiz Documentation

Version 1.7.0

Ambroise Maupate, Julien Blanchet

août 05, 2022

Table des matières

Roadiz est un système de gestion de contenus polymorphe construit sur un système de nœuds, lui permettant de proposer une grande variété de services. Il est basé sur les composants *Symfony*, *Doctrine ORM*, *Twig* et *Pimple* pour garantir un haut niveau de sécurité ainsi qu'un maximum de performances.

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*.

CHAPITRE 1

Philosophie

En découvrant l'interface back-office de Roadiz, vous remarquerez qu'il n'y a pas d'éditeurs de texte riche ou éditeurs *WYSIWYG*. Nous avons fait le choix de promouvoir la syntaxe *Markdown* afin de se concentrer sur la hiérarchie et la qualité de vos contenu, au lieu de leur mise en forme. Notre objectif est de préserver et de respecter le travail des designers et des développeurs.

Vous constaterez que nous avons construit Roadiz en tant que designers, pour des designers. Il vous permettra de créer très rapidement des prototypes de sites Web en utilisant des templates *Twig*. De même, vous pourrez utiliser la puissance des composants *Symfony* et *Doctrine* pour construire des applications complexes.

Nous avons également opté pour une gestion stricte des Plugins et autres modules add-ons. Nombreux sont ceux qui ne mettent pas à jour leur Wordpress en raison des dépendances de plug-in ! C'est la raison pour laquelle nous avons décidé de ne pas construire Roadiz autour d'un système « plug-in », mais d'utiliser plutôt un système de **thèmes**, obligeant ainsi toutes les extensions Roadiz à servir les caractéristiques d'un thème. Ces thèmes vous permettront de créer des layouts de site Web impressionnants, mais aussi de multiples back-offices personnalisés et spécifique aux besoins de vos clients. Vous serez en mesure de centraliser tout votre code dans un seul endroit, de sorte que vous pourrez utiliser un outil de versioning tel que Git.

Le système de thème de Roadiz vous permettra de connecter en chaîne des thèmes et des fonctionnalités à partir de plusieurs dépôts de code. Puisque notre CMS est construit sur le conteneur d'injection Pimple, Roadiz peut fusionner tous les thèmes disponibles sur un même site. Par exemple, vous serez en mesure de créer un thème de portfolio grâce au système d'URL des nœuds ainsi qu'aux thèmes statiques, qui utiliseront un système de routage statique, un forum ou un blog ou encore même les deux ! Le thème vous permettra même de créer des entités supplémentaires Doctrine et d'étendre votre back-office. En vous basant simplement sur votre code métier, vous pouvez étendre Roadiz pour créer un gestionnaire pour votre forum. Cerise sur le gâteau, vous pouvez aussi affecter chaque thème à un nom de domaine pour créer des versions spécifiques mobiles à destination de certains médias.

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.

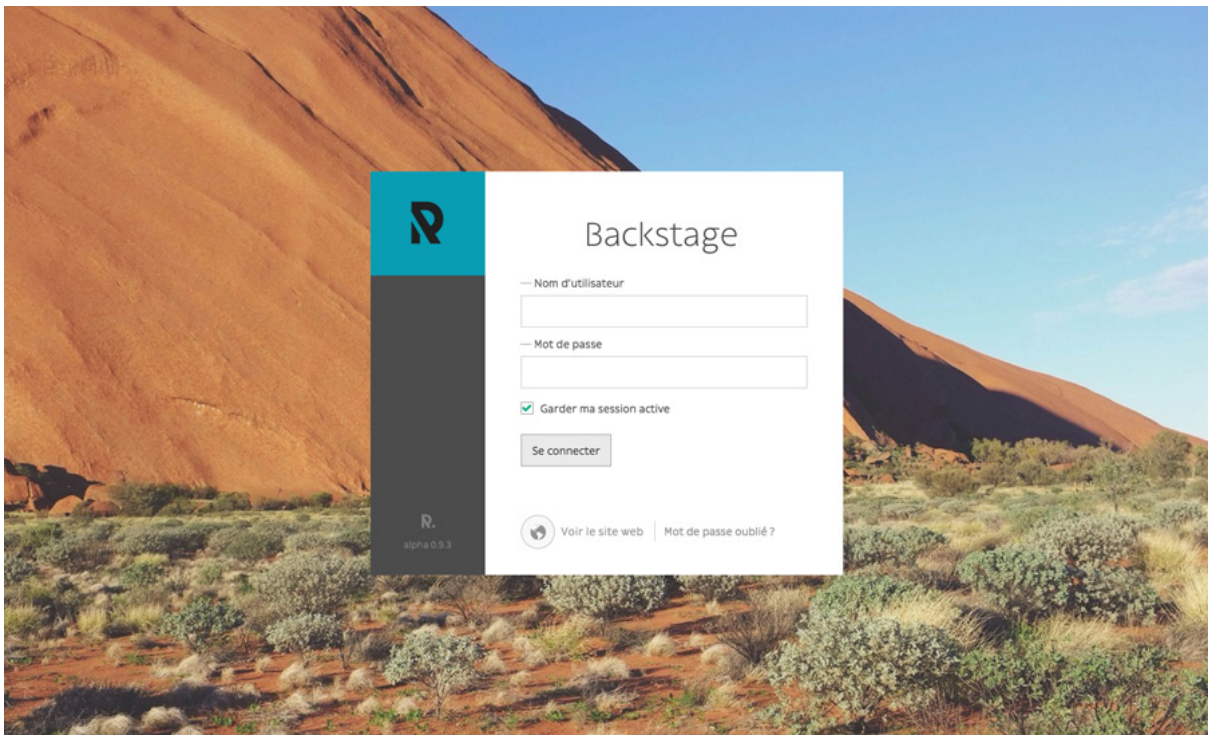
CHAPITRE 2

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.

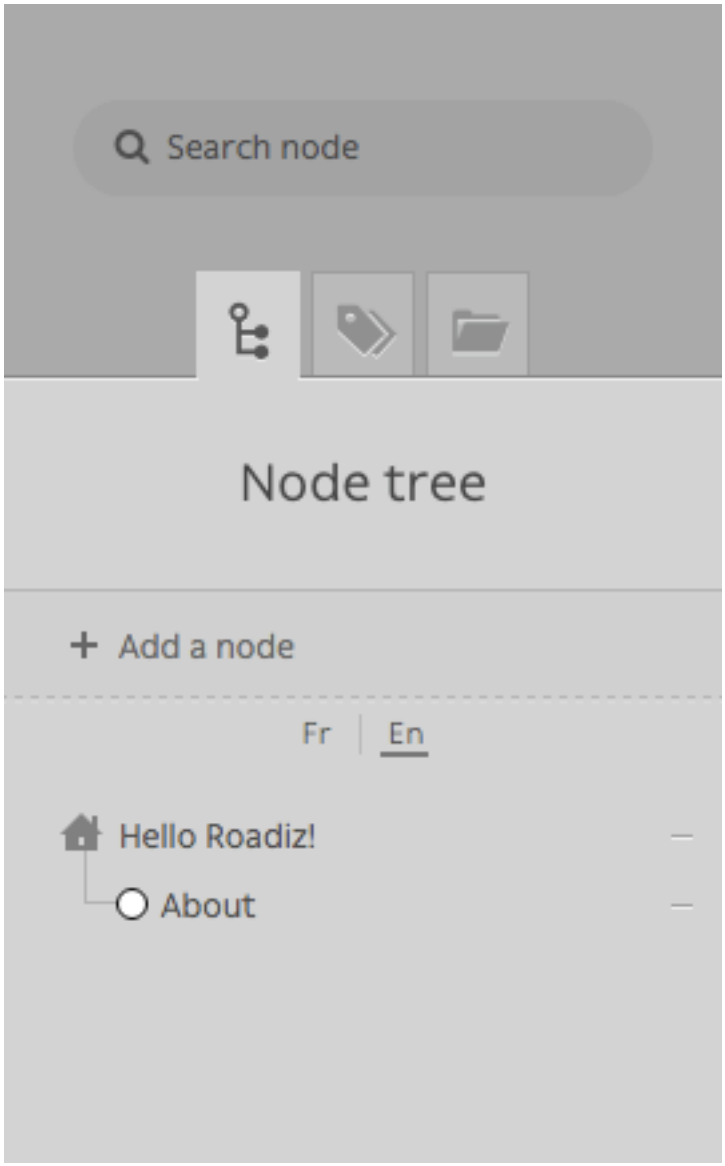
2.1.2 Gérer les nœuds

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.

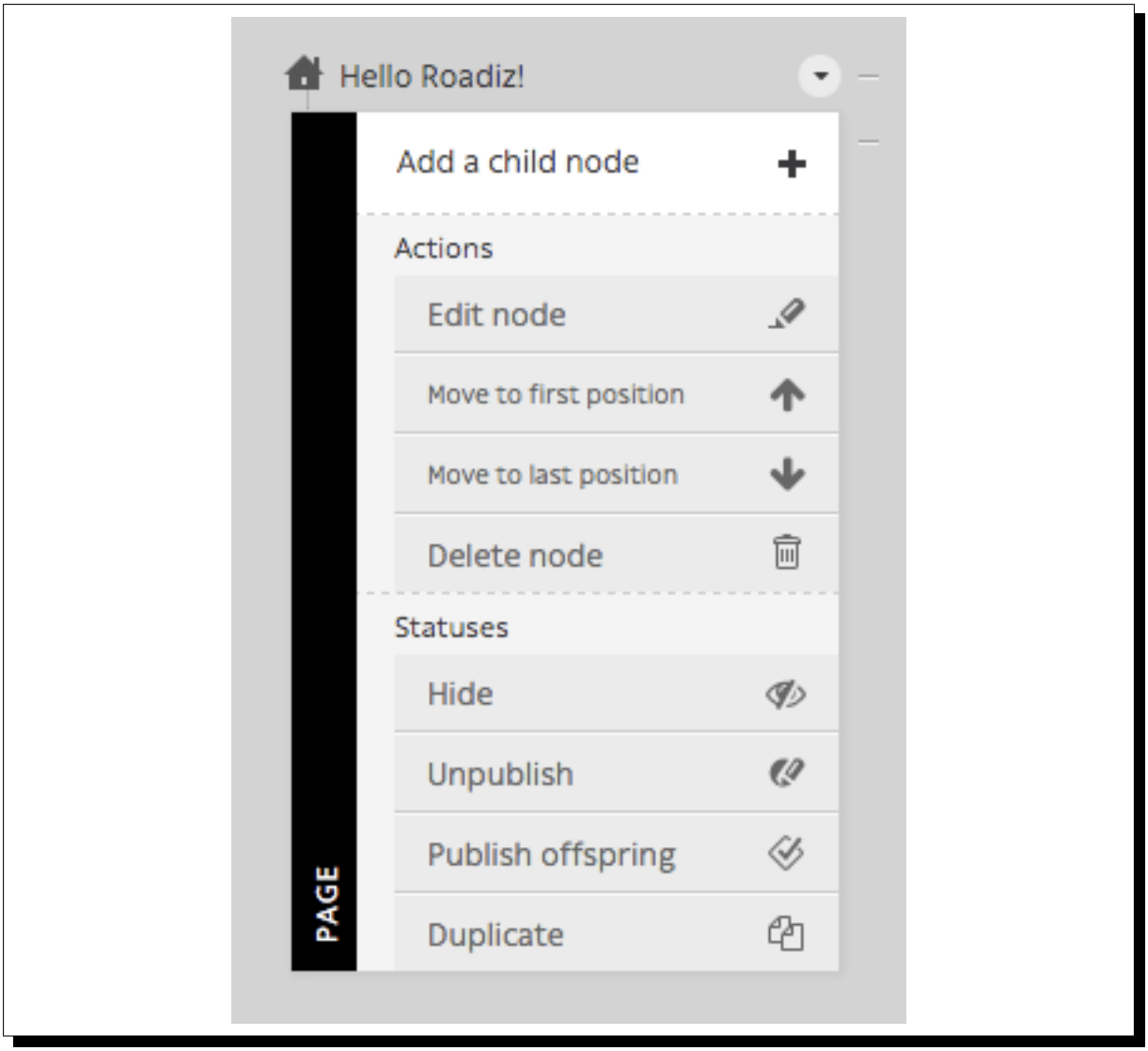
Navigation dans l'arborescence

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.
— D'autres options sont disponibles à partir du menu contextuel de chaque nœud. Faites un clic-droit sur le nœud ou cliquez sur la flèche droite qui apparaît au survol.

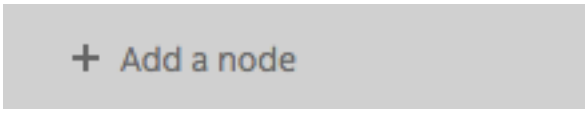
Actions des menus contextuels

— *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.

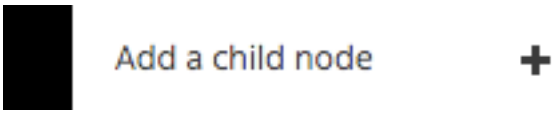
— *Publish/Unpublish :* Change a node' publication status. Unpublished nodes are not visible to anonymous visitors, but visible for back-office users using `?_preview=1` query parameter.
— *Publier la descendance* : Publier un nœud et tous ses noeuds-enfants rattachés.
— *Dupliquer* : Copier l'intégralité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

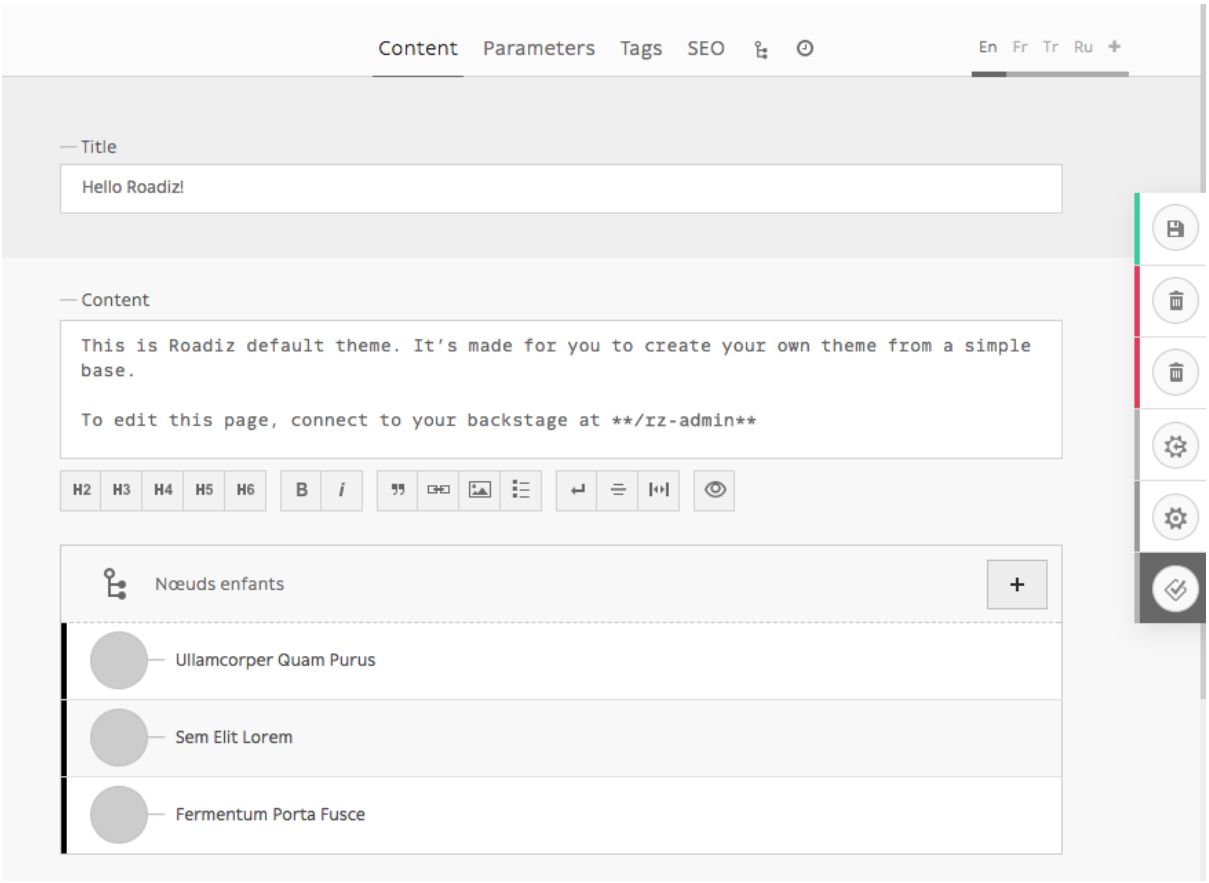
— Le *Nom du noeud* est l'identifiant global de votre nœud. Il doit être unique et ne changera pas d'une traduction à un autre. Il peut être modifié ultérieurement, sauf si votre développeur a verrouillé sa modification. Le *Nom du nœud* est utilisé pour construire les URL de vos pages, de manière générale.
— 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)*

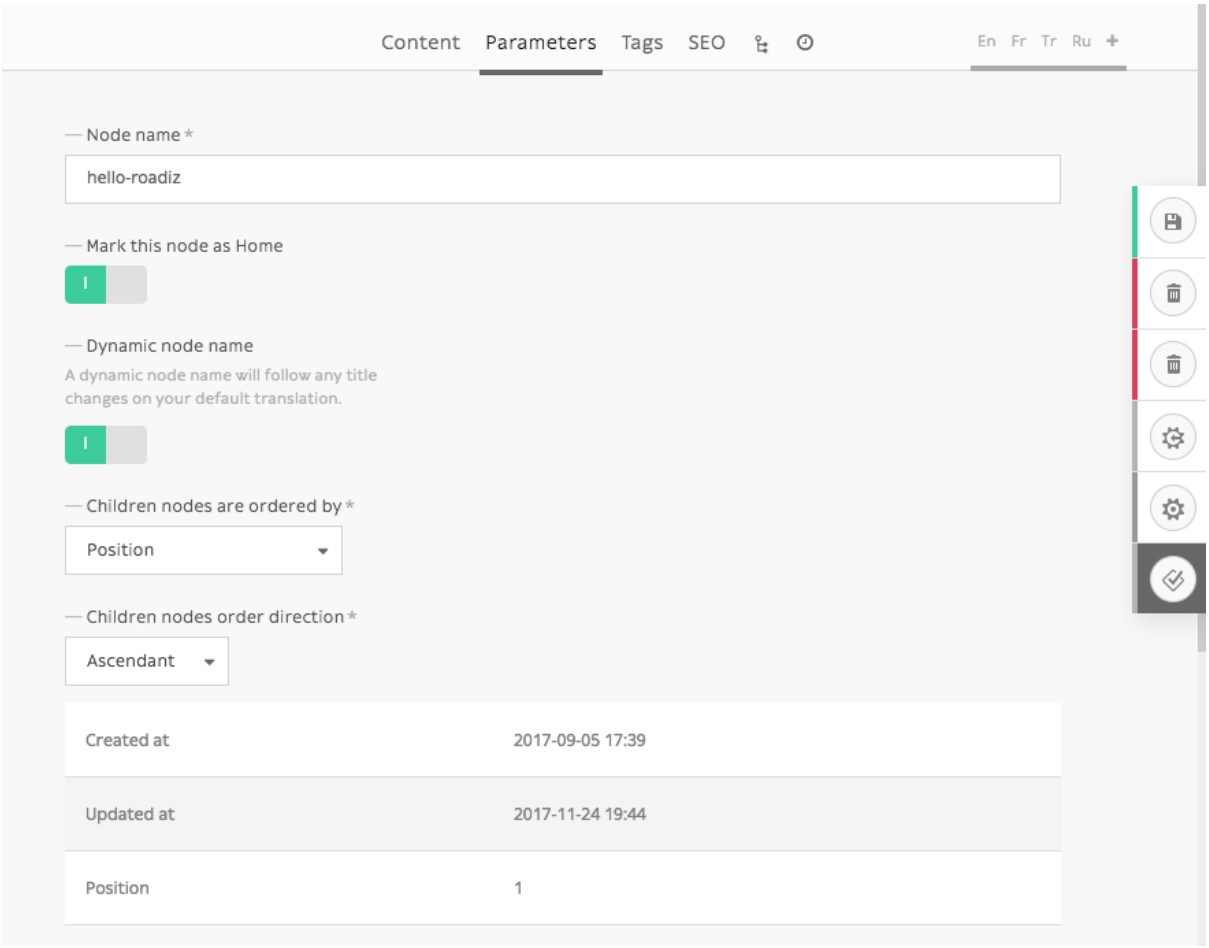
Contenu d'un nœud



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

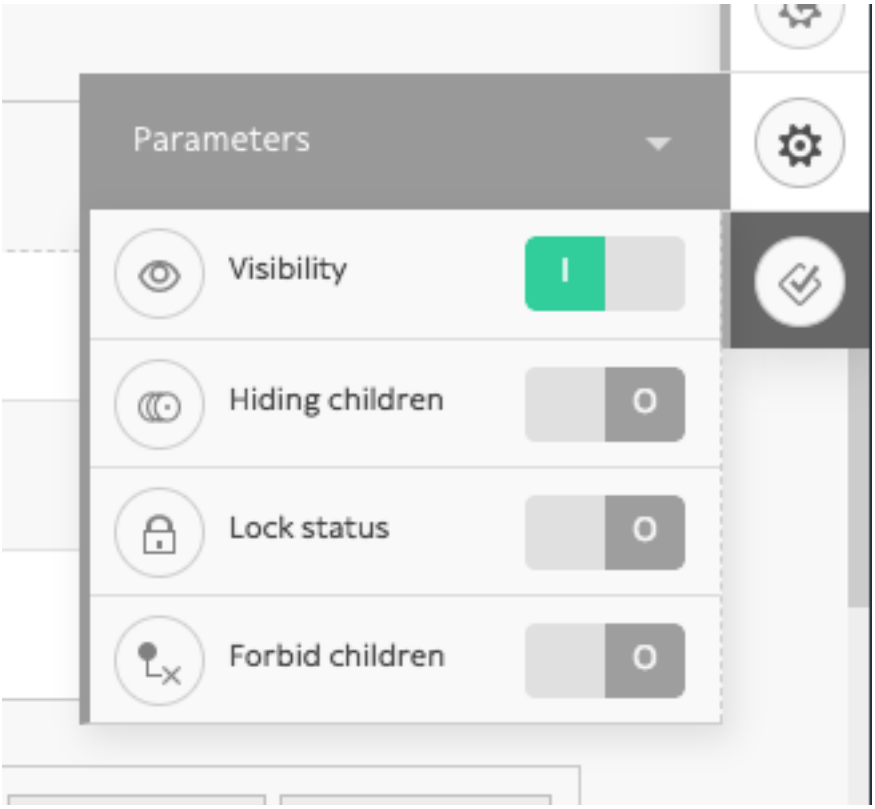


Les paramètres sont des données globales telles que votre *nom de nœud*. Ils sont utilisés pour gérer la visibilité de votre nœud en fonction des rôles de chaque utilisateur et des paramètres de chaque nœud. Cette section ne doit pas être utilisée de façon régulière puisque les paramètres sont fixés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

Cet onglet affichera le même contenu sans importance de traduction.

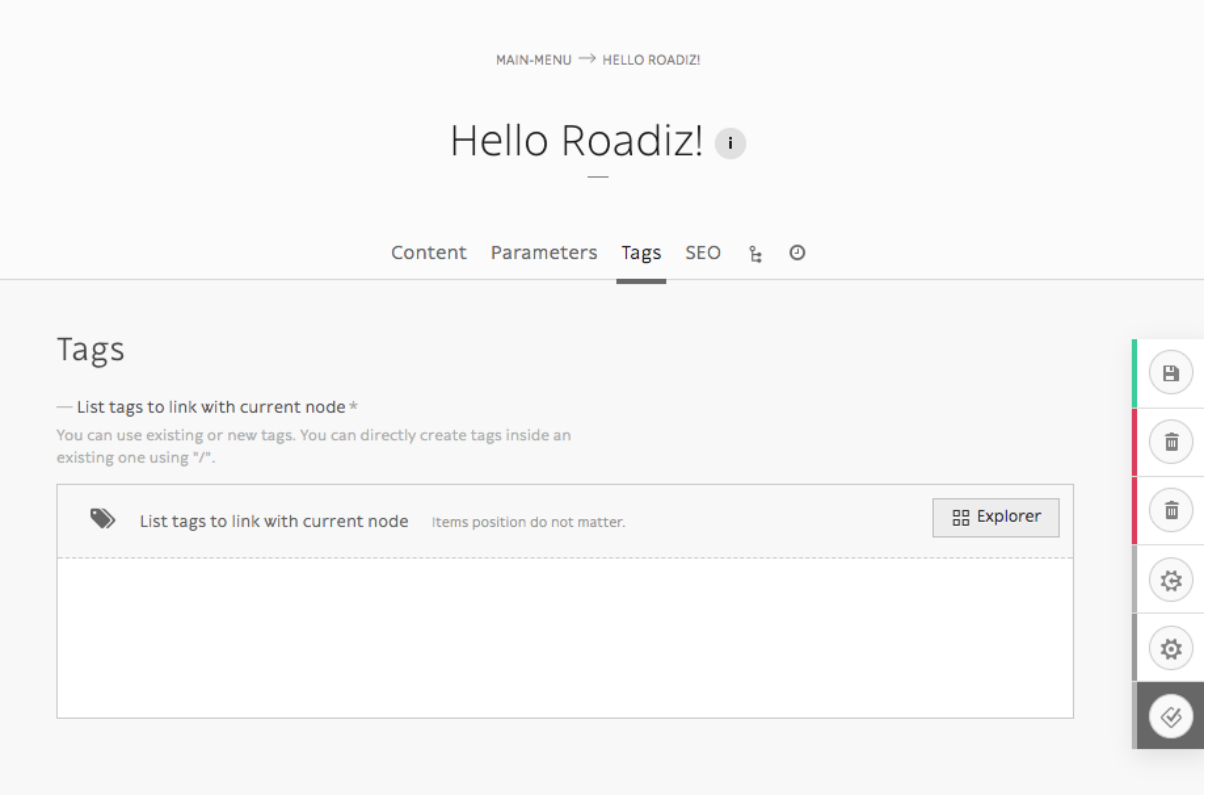
Actions et paramètres secondaires

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.



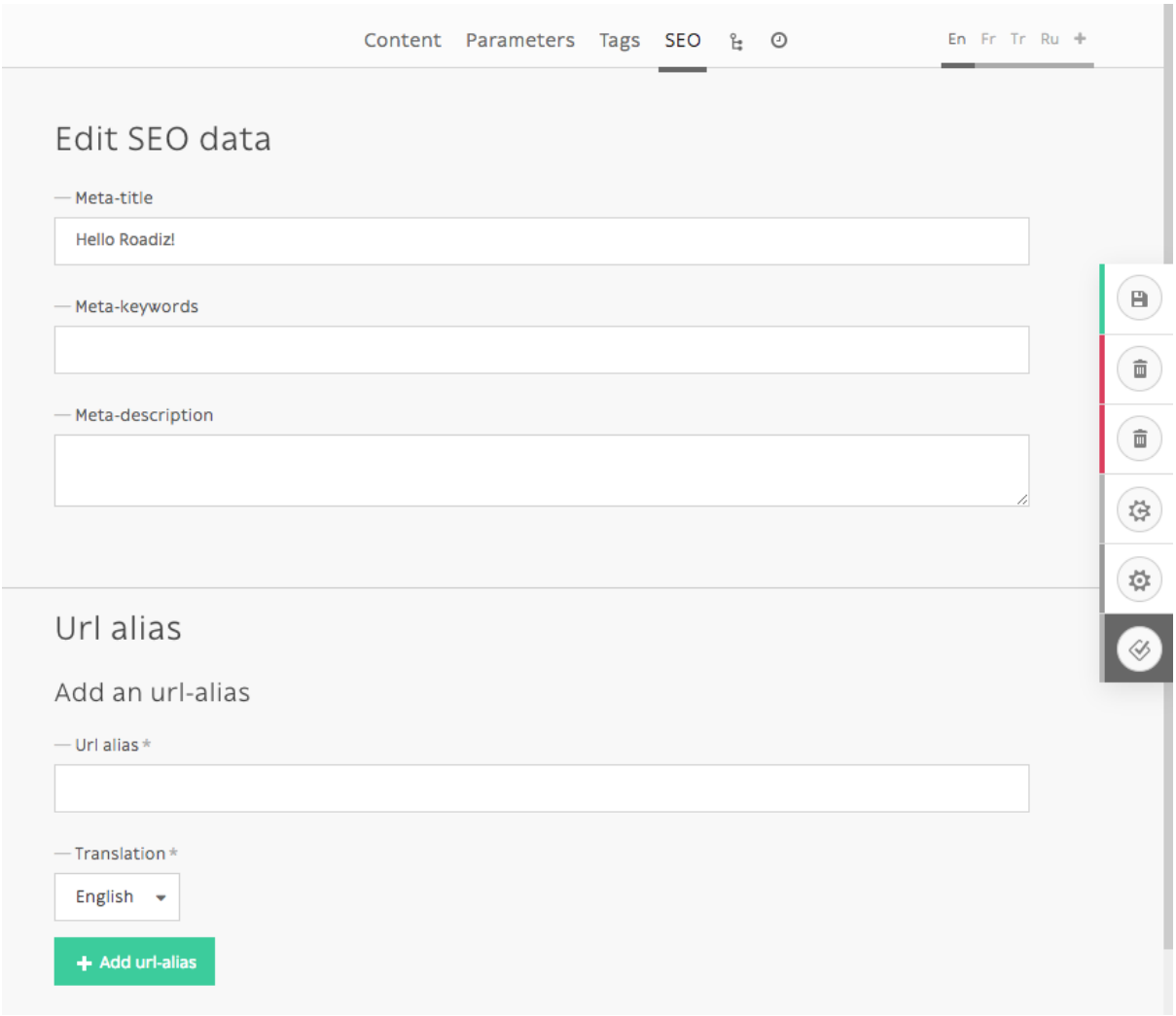
— **Visibilité** : Cache ou affiche le nœud actuel (en fonction du développement de votre thème)
— **Cacher les enfants** : Transforme le nœud actuel en **conteneur (ou pile)**, les nœuds enfants n'apparaitront plus dans le panneau global d'arborescence. Vous pourrez ajouter des boutons d'ajout de *types 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.

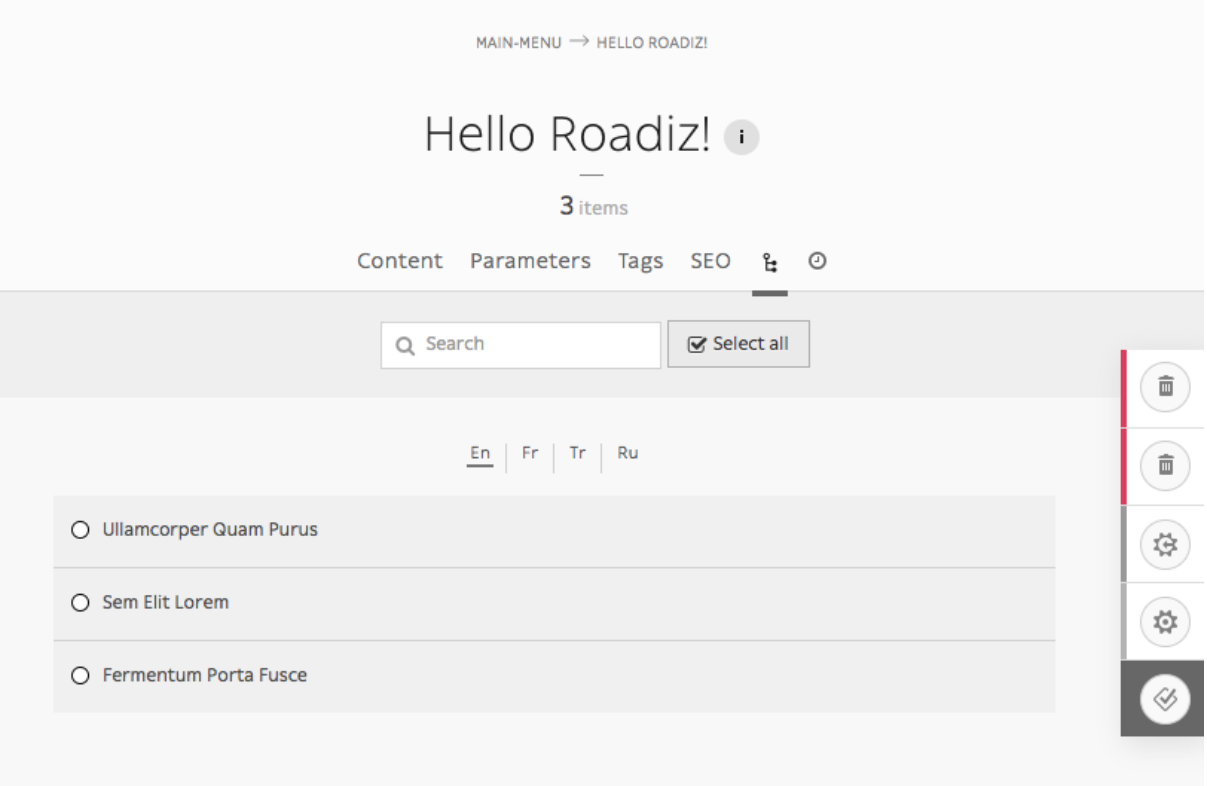
Référencement



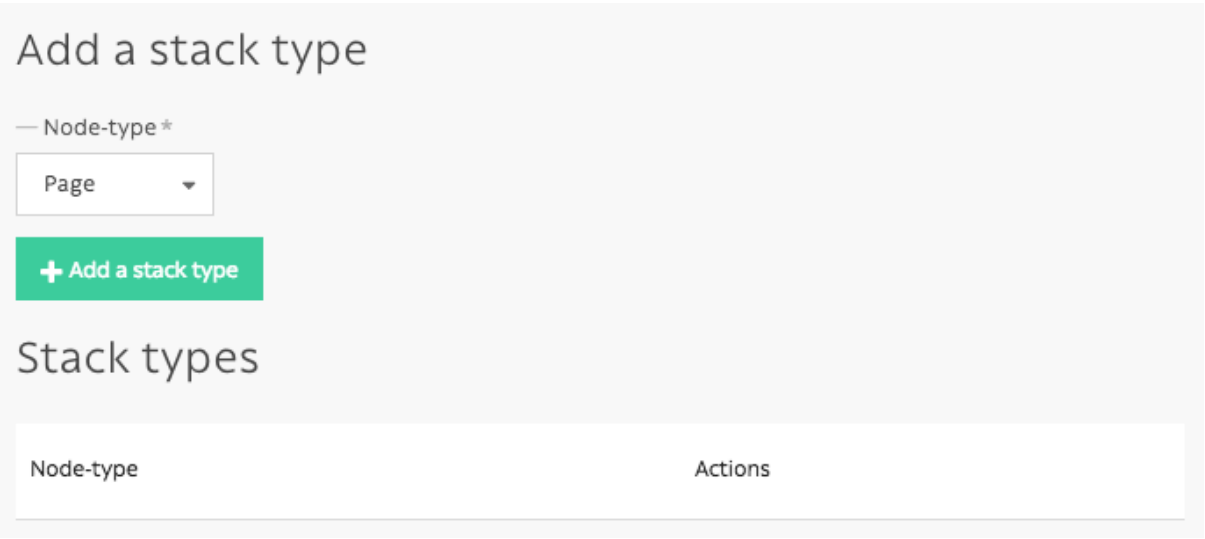
Cet onglet affichera les contenus en fonction de chaque traduction.

L'arborescence

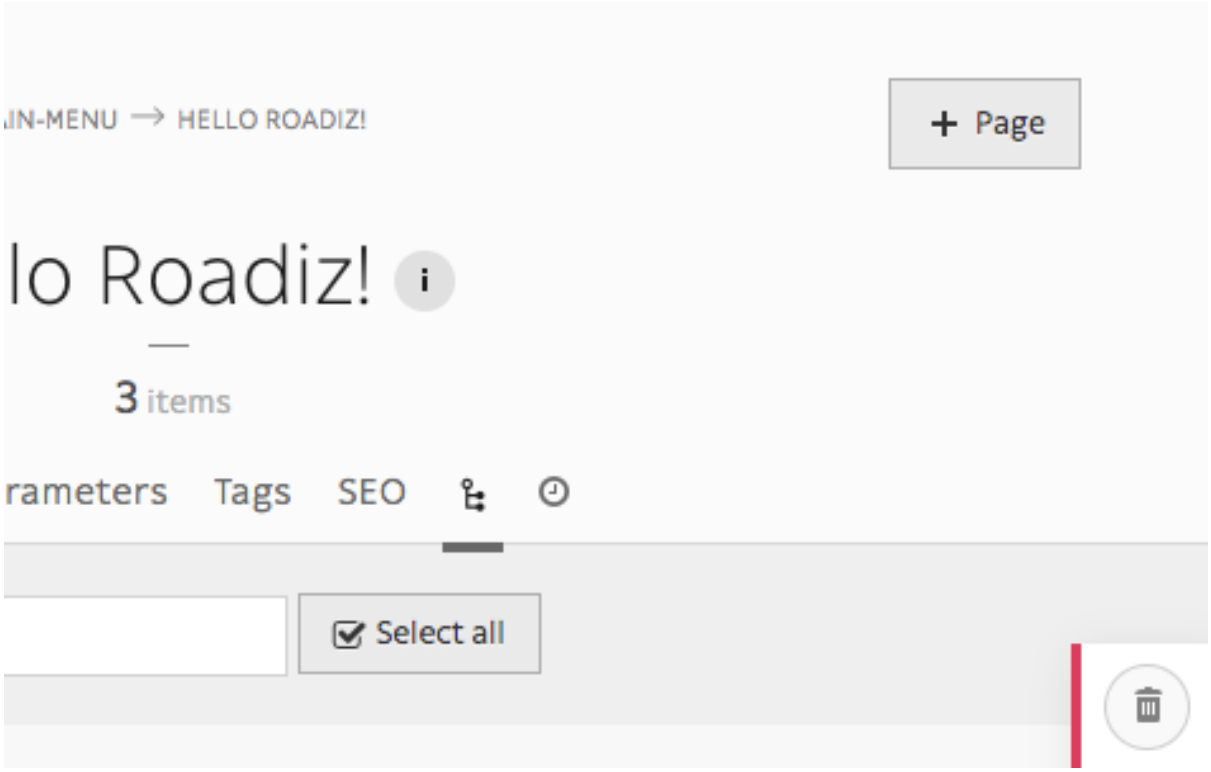
Quand un nœud est défini en tant que *conteneur (ou pile)*, son *arborescence* devient la vue (onglet) par défaut. Vous pouvez définir l'ordre par défaut d'affichage des nœuds enfants dans l'onglet *Paramètres*. De plus, si vous avez défini le *type de nœud* comme étant *publiable*, chaque nœud affichera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

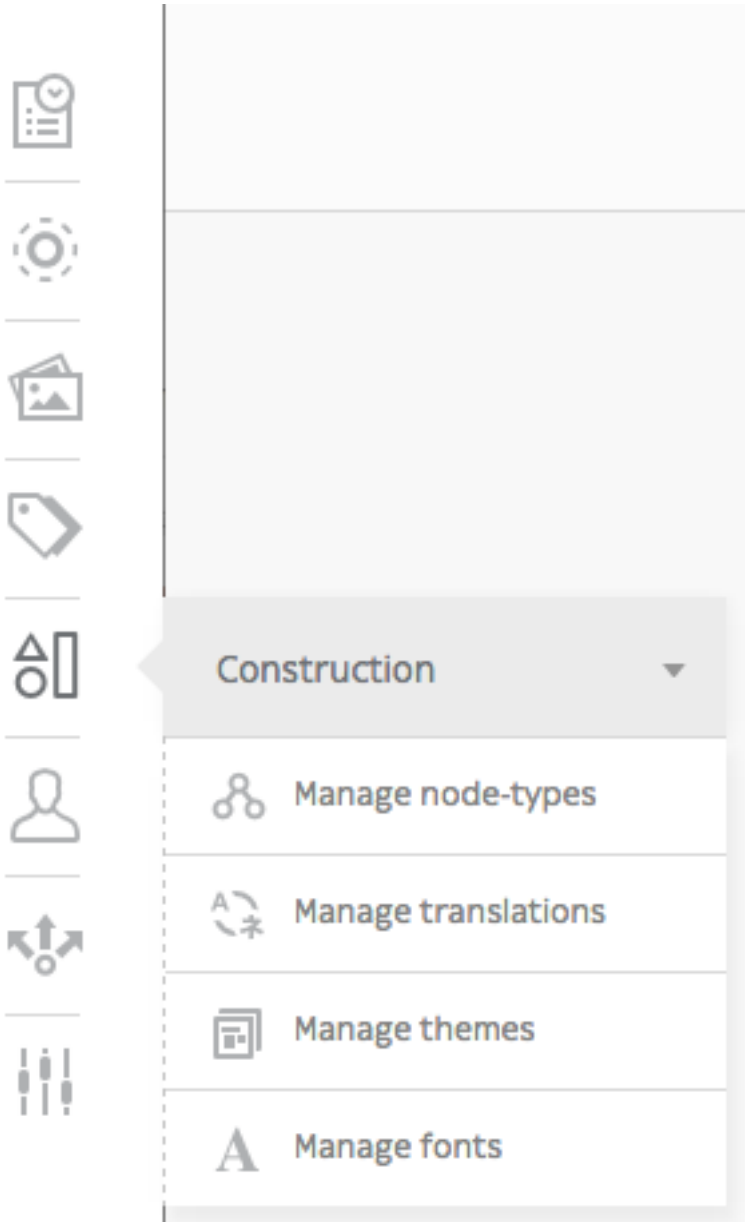
2.1.3 Gérer les types de nœuds

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.

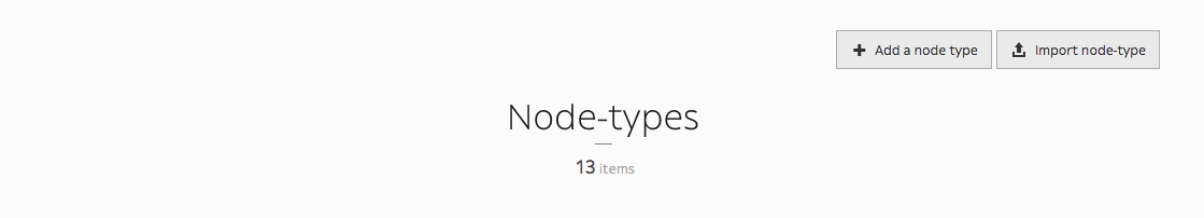


Ajouter un type de nœud

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.

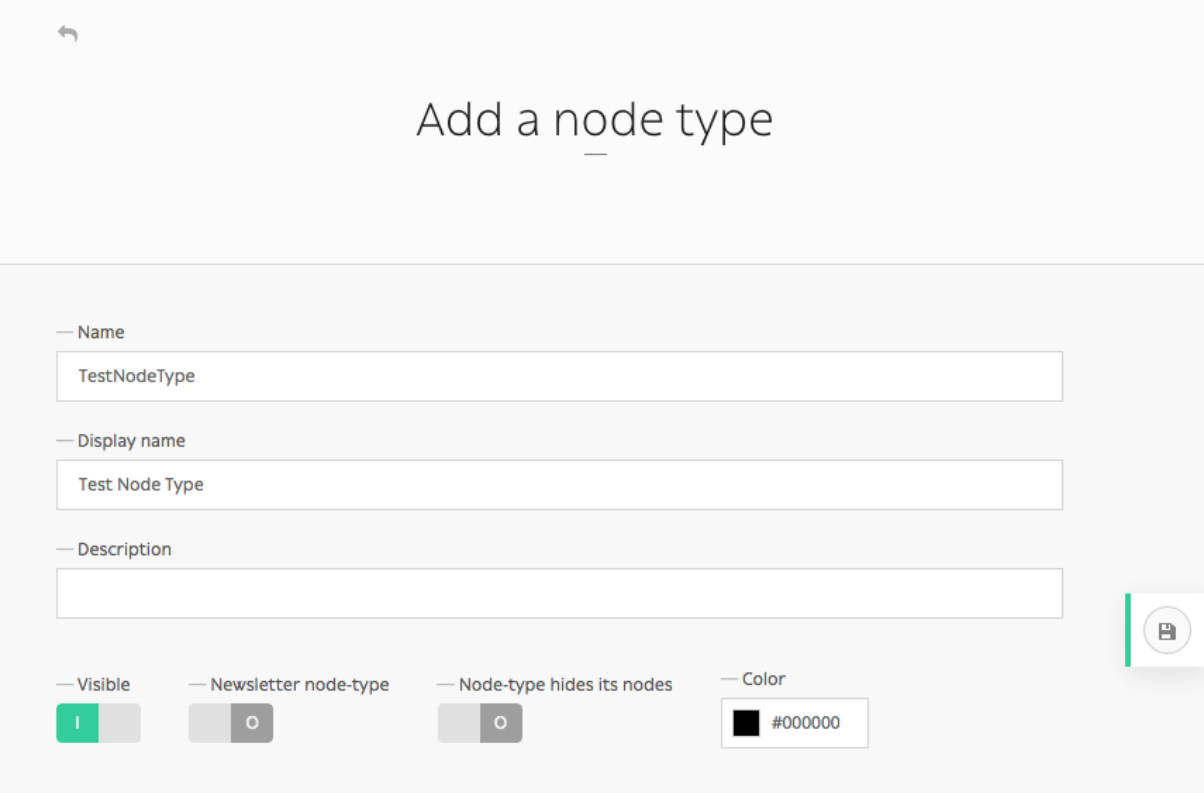
Note : À savoir, vous pouvez exporter et importer un type de nœud avec un fichier .json. Rendez-vous sur *Autre*

action pour plus d'informations.



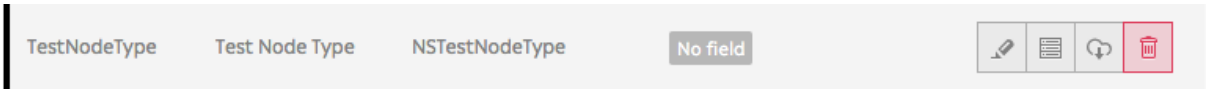
Une fois les deux paramètres obligatoires remplis - *Nom* (que les développeurs utiliseront) et *Nom d'affichage* (que les utilisateurs de back-office verront) - , vous pourrez créer votre premier type de nœud.

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*.



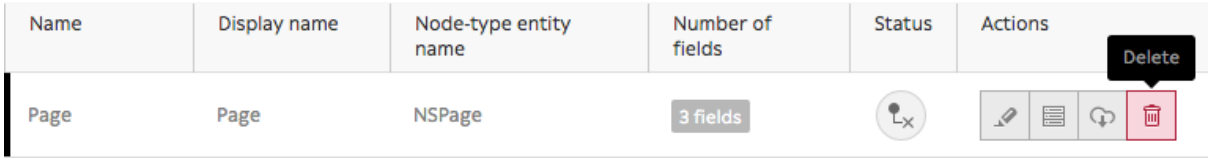
Les autres options (*Visible*, * Newsletter*, *Type de nœud caché* et les couleurs personnalisables) ne sont pas obligatoires et peuvent toujours être modifiées ultérieurement.

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.



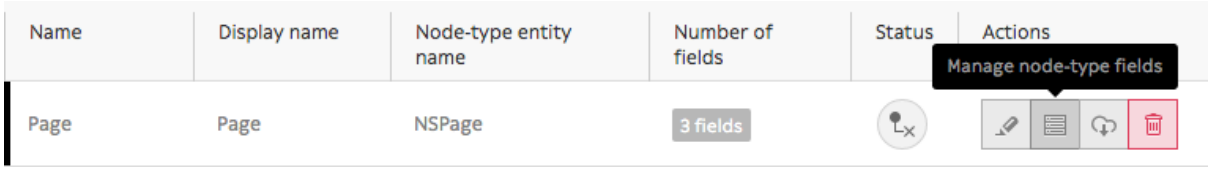
Supprimer un type de nœud

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 ».

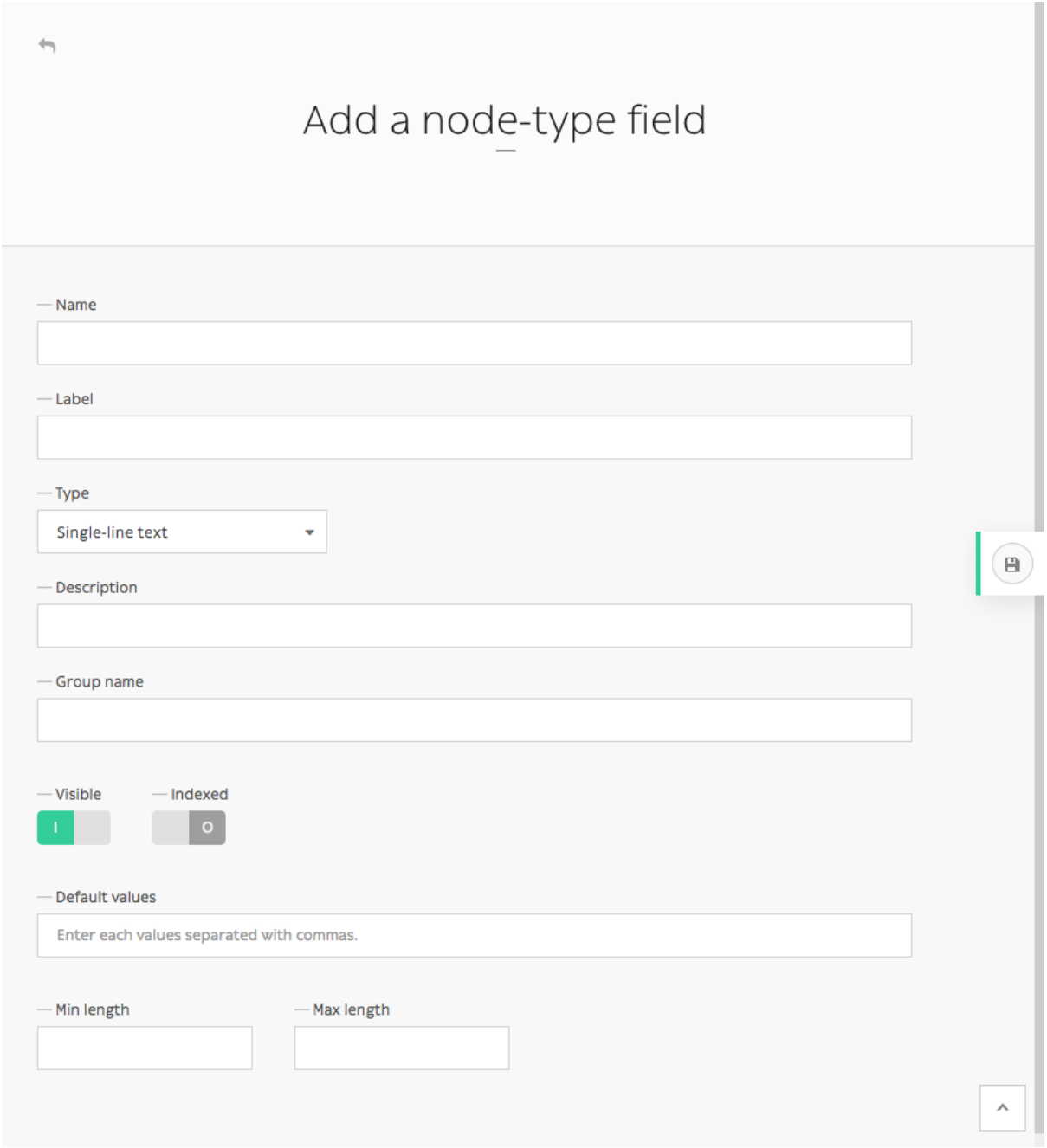


Puis cliquez sur le bouton "Ajouter un champ"

Remplissez le formulaire pour créer un nouveau champ :

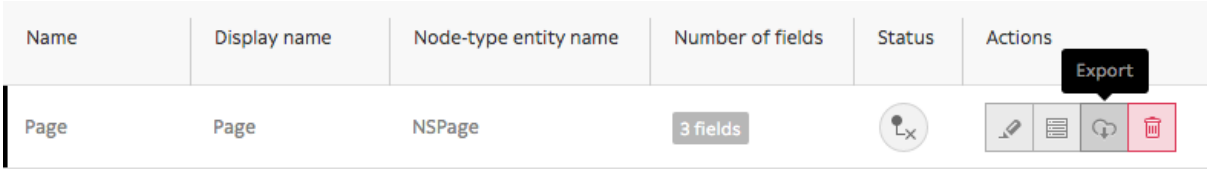
- *Nom* : utilisé par les développeurs
- *Étiquette* : ce que les utilisateurs du back-office verront
- *Type* : option de choix unique qui va définir le contenu (texte de base, texte markdown, documents, e-mail, numéro, choix simple ou multiple, nœuds enfants, etc.)
- *Description*, *Nom de groupe*,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



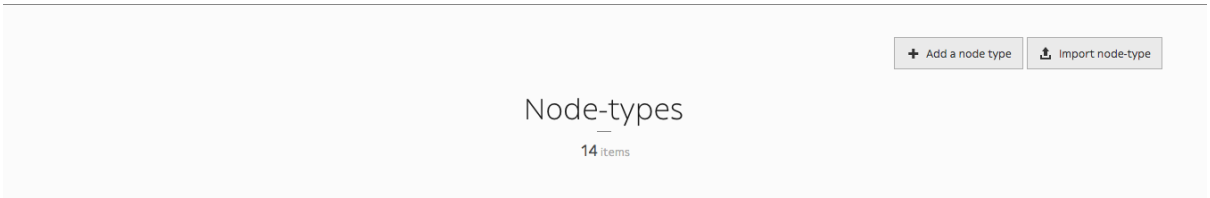
Un fichier .json devrait ressembler à cela lorsque vous l'ouvrez dans n'importe quel éditeur de code :

```
{
    "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*.

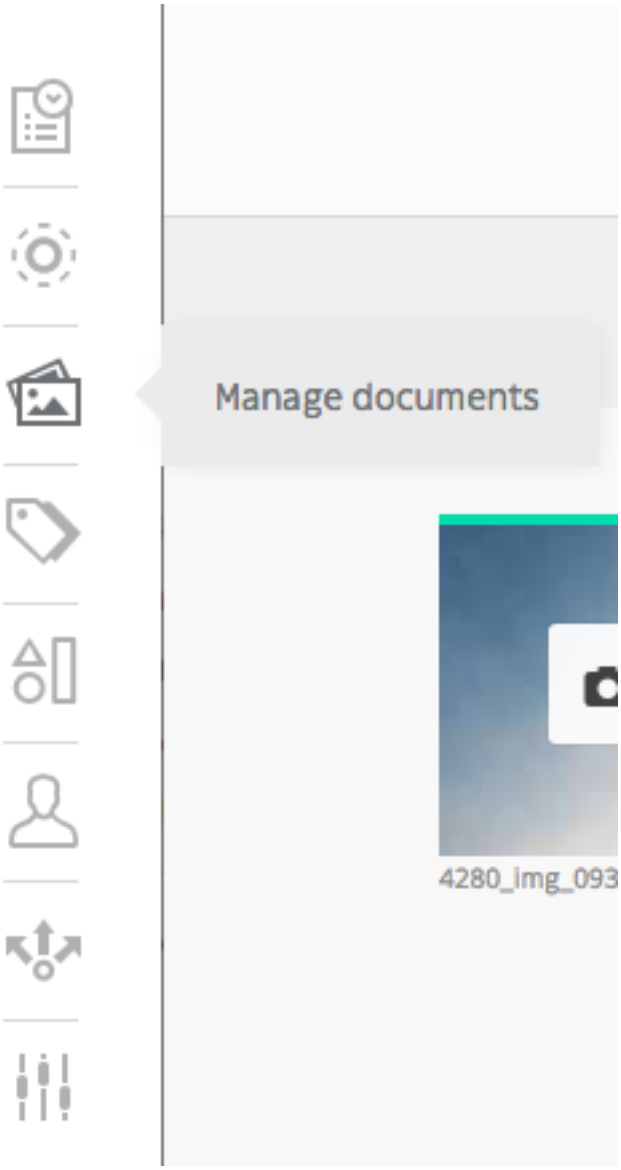
Vous pouvez écrire un fichier JSON vous-même, si vous en avez envie, mais il est probablement plus facile d'exporter simplement des types de noeuds à partir du back-office, puis d'importer sur votre nouveau site web. Vous pouvez le faire facilement en cliquant sur le bouton *Importer le type de nœud*,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



Vous êtes proche de la maîtrise complète des pouvoirs du back-office de Roadiz ! Continuez à vous référer à cette documentation si vous rencontrez des difficultés et faites-nous signe pour toute question ou complément d'information.

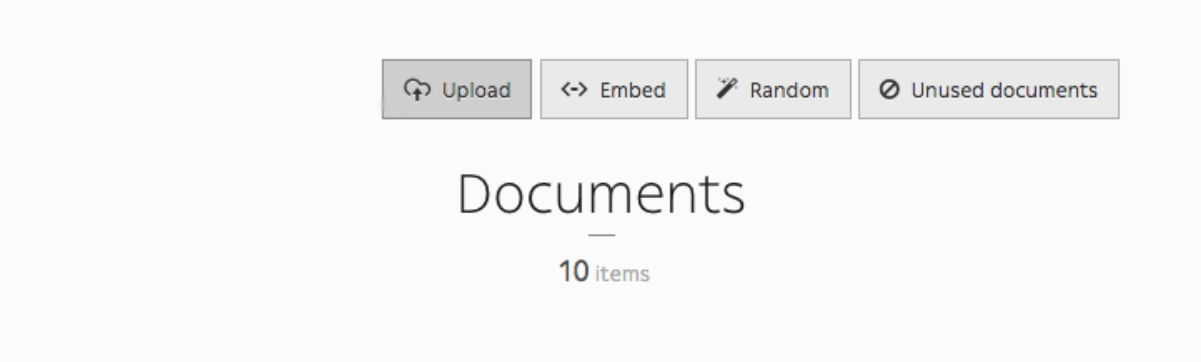
2.1.4 Gérer les documents

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.

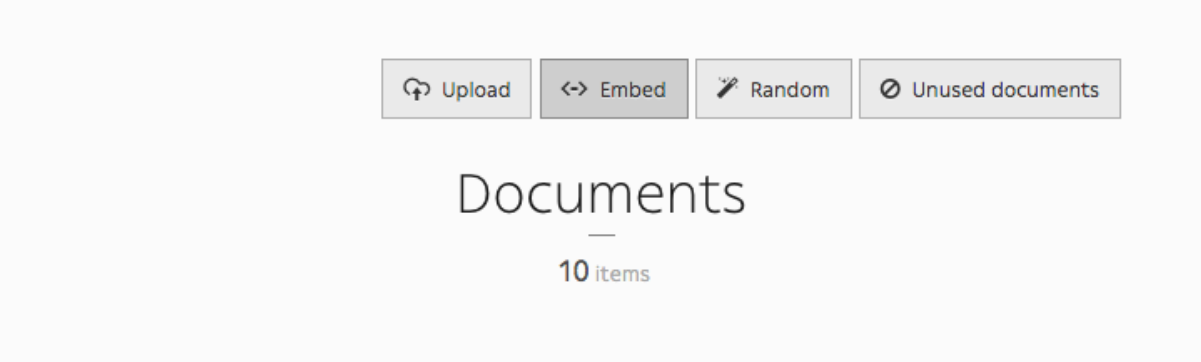
— Photographies, images complexes sans transparence : *utiliser JPEG*
— Logos, icônes, images complexes avec transparence : *utiliser PNG*

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.

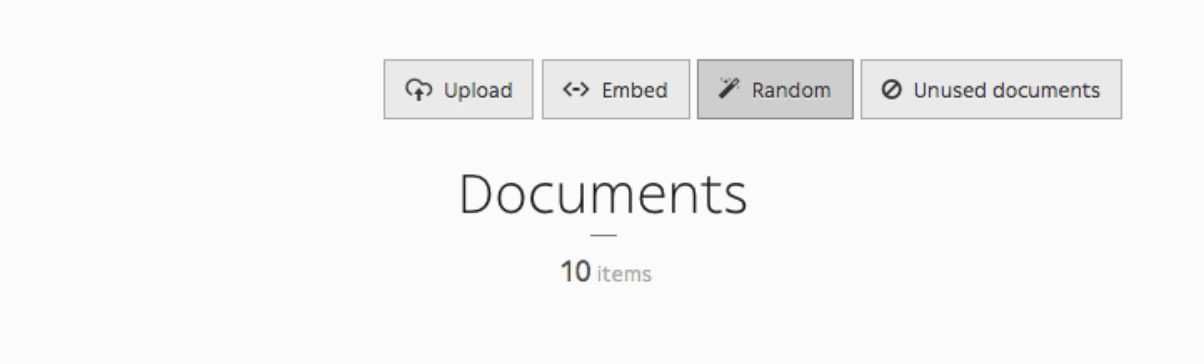
Intégrer un document

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.



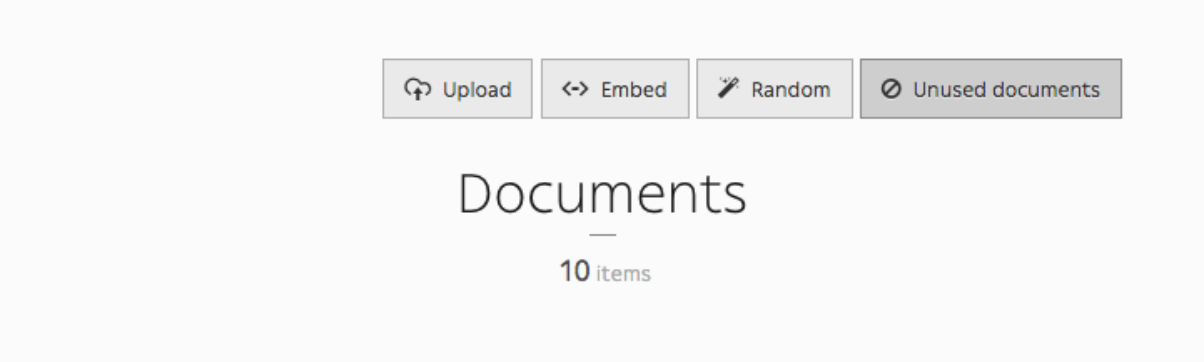
Document au hasard

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.

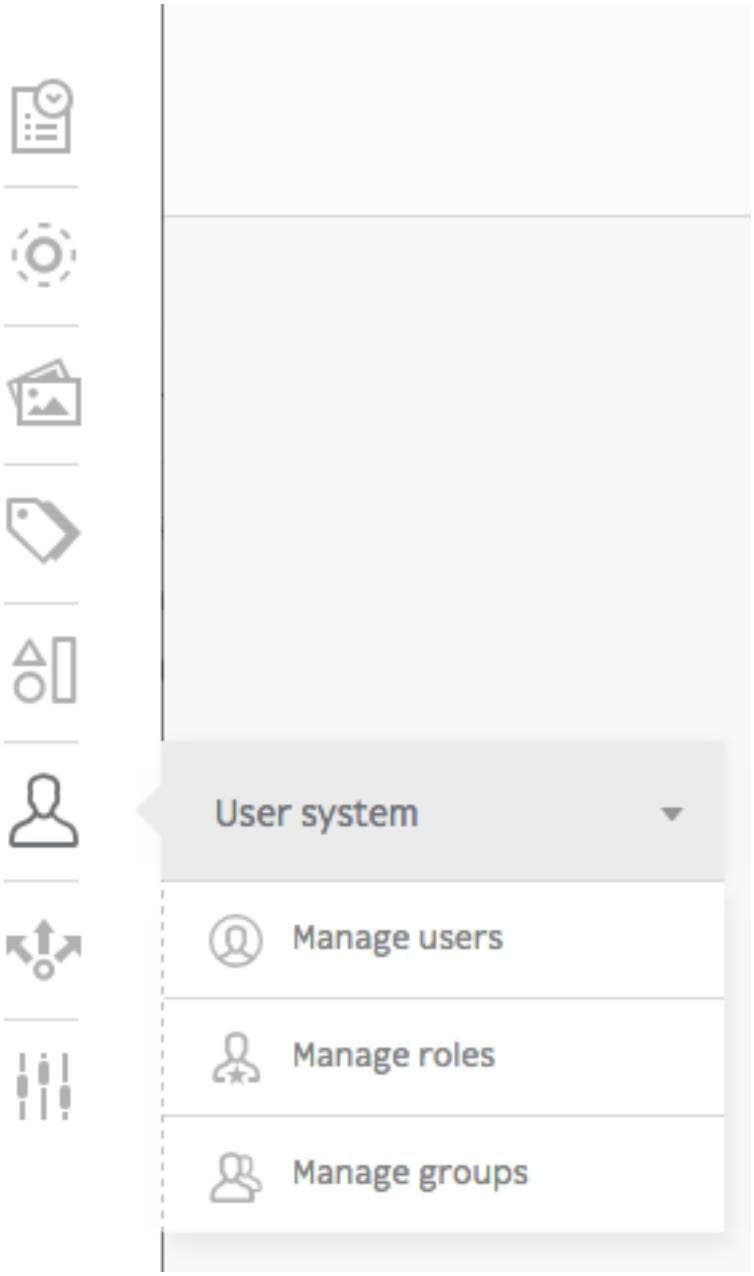


2.1.5 Gérer les utilisateurs

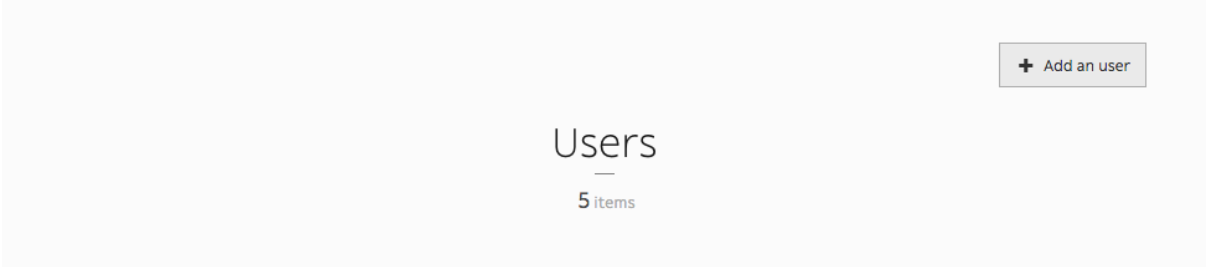
Il s'agit d'un guide simple sur la façon de créer et de gérer les utilisateurs en utilisant Roadiz CLI.

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

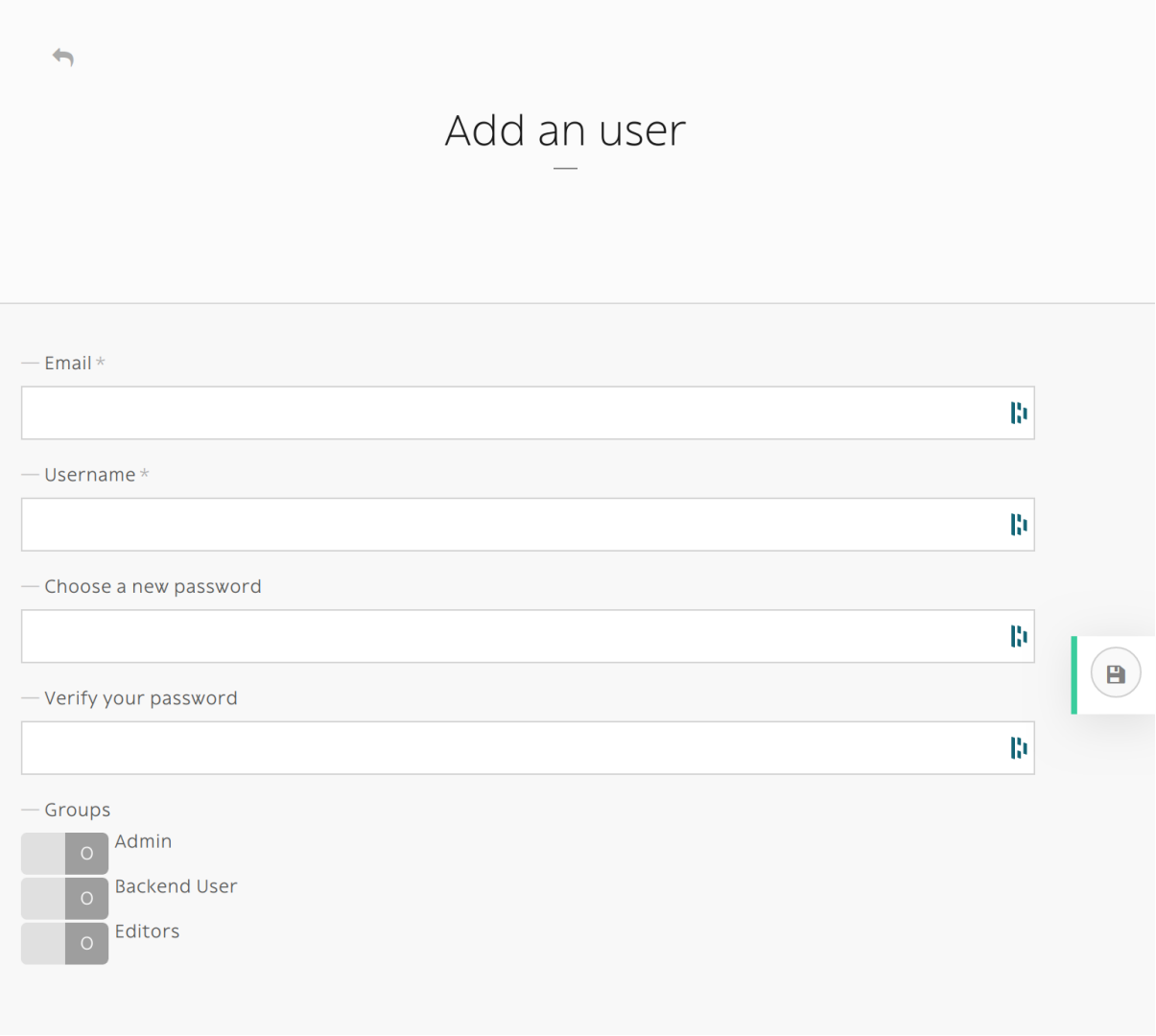
Lorsque vous travaillez dans le back-office de Roadiz, vous pouvez facilement gérer les utilisateurs via l'icône *Système utilisateur* dans la barre d'outils.



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.

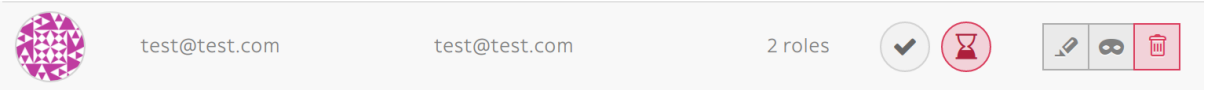
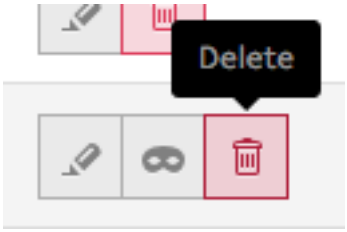


Fig. 2.1 – Un nouvel utilisateur sans mot de passe aura son compte verrouillé jusqu'à ce qu'il réinitialise son mot de passe.

La ligne de commande `bin/roadiz 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

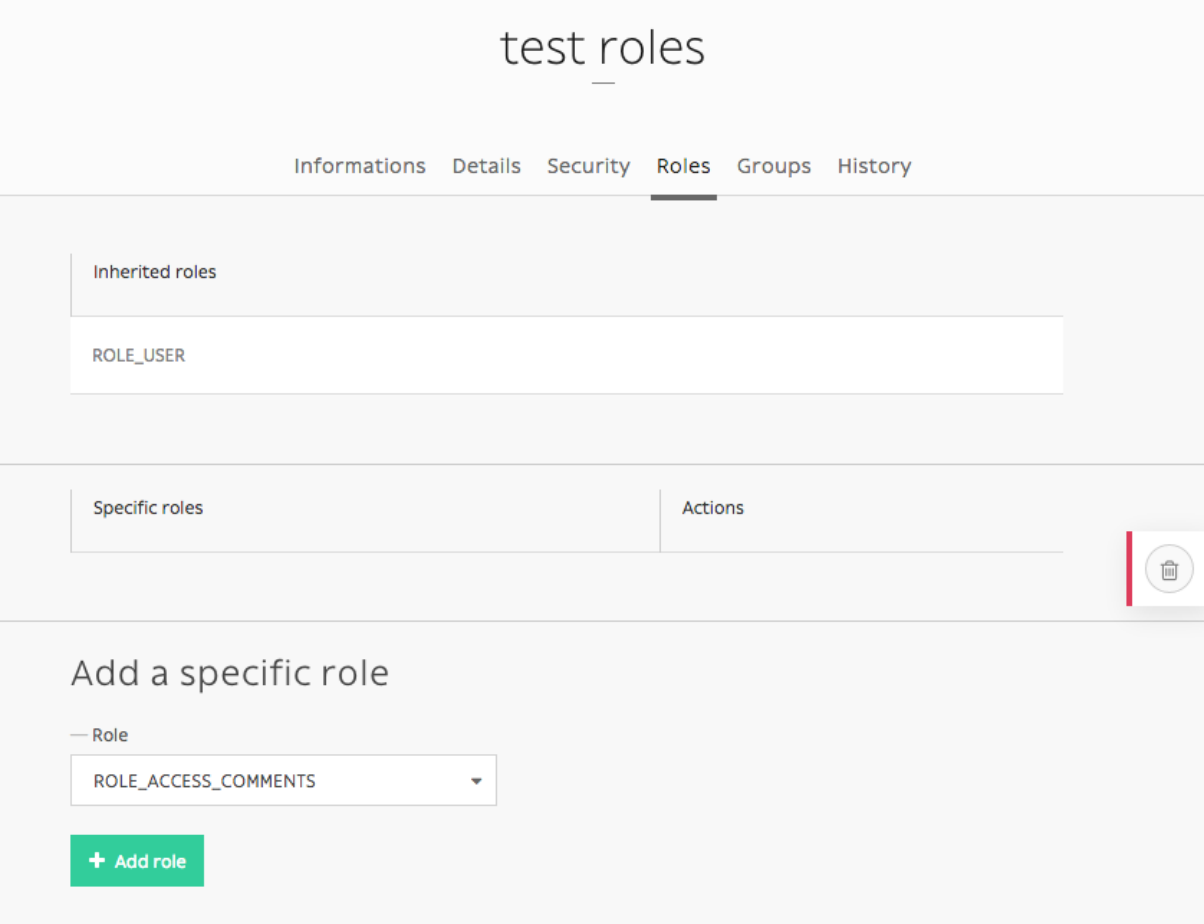
Vous pouvez supprimer des utilisateurs en cliquant sur l'icône de la corbeille.



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

Autres actions

Il est possible d'activer ou de désactiver les utilisateurs avec la commande `users:enable` ou `users:disable`. Si un utilisateur ne se souvient pas de son mot de passe, vous pouvez le régénérer avec la commande `users:password`. Pour plus d'informations et plus d'actions, nous vous invitons à vérifier les commandes disponibles avec :

```
bin/roadiz 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* :

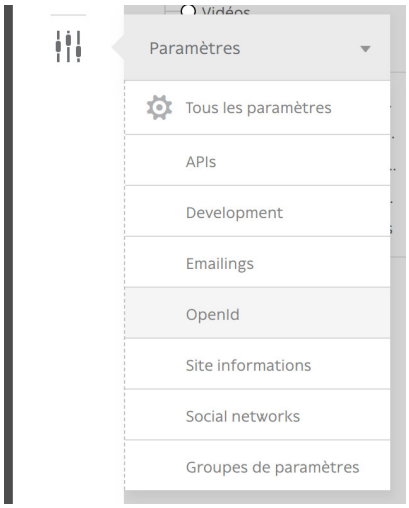
— Client ID *OpenID*
— Client secret *OpenID*
— URL d'auto-découverte *OpenID* (ex : pour Google Suite https://accounts.google.com/.well-known/openid-configuration)

Puis vous devez déterminer :

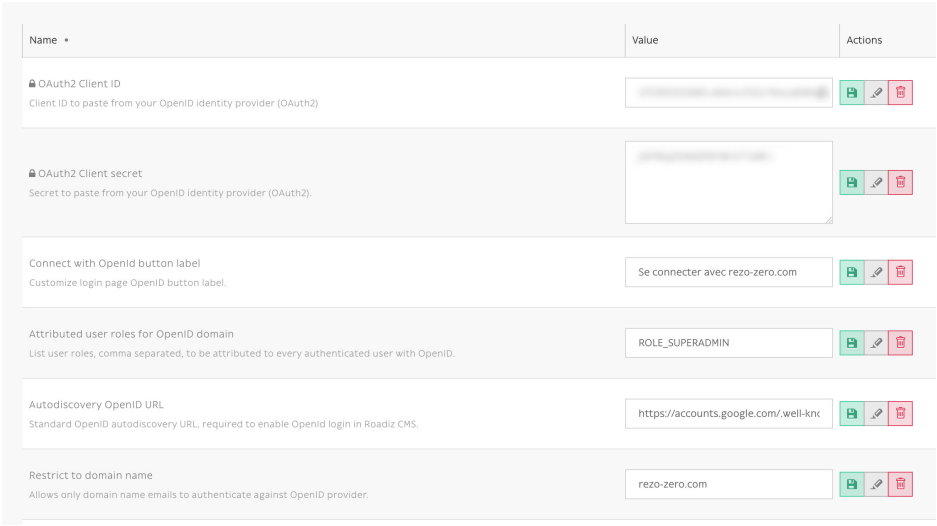
— Quels *rôles* (séparés par virgule) vous souhaitez automatiquement assigner à vos utilisateurs connectés par SSO. Vous pouvez toujours développer une stratégie plus complexe pour votre site web en implémentant `RZ\Roadiz\OpenId\Authentication\Provider\JwtRoleStrategy`.

```
$container->extend('jwtRoleStrategies', function (array $strategies, Container $c)
↪{
    return array_merge($strategies, [
        new MyWebsiteRoleStrategy($c['em'])
    ]);
});
```

— 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.



Remplissez toutes les informations collectées dans les paramètres de *Roadiz* correspondant.



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

Developer documentation deals with Roadiz' inside and how to create your own themes. It requires a minimum of *Twig* templating knowledge and oriented-object *PHP* skills.

3.1.1 Premiers pas

Requirements

Roadiz is a web application running with PHP. It 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**
- `php-gd` extension
- `php-intl` extension
- `php-xml` extension
- `php-curl` extension
- `php-mbstring` extension
- JSON needs to be enabled
- ctype needs to be enabled
- Your php.ini needs to have the `date.timezone` setting
- You need to have at least version 2.6.21 of libxml
- PHP tokenizer needs to be enabled
- 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
- Zip/Unzip
- cUrl
- Composer
- Git

Note : If you are using a *shared hosting plan*, make sure that your server's SSH connection allows external connections. You can verify with a simple `ping google.com`. If you get request timeouts, your hosting provider might be blocking your SSH connectivity. You should consider using at least a VPS-like hosting. If you really need to setup Roadiz on a simple shared-hosting plan, we encourage you to install it on your own computer and send it with SFTP/FTP (it might take a long time) or *rsync* it.

For Nginx users

If you are using Nginx, you don't have to enable any extensions. You only need to create your *virtual host* using our example file `/samples/nginx.conf`.

For Apache users

If you are using *Apache* do not forget to enable these mods :

- `mod_rewrite` : enabling Roadiz front-controller system.
- `mod_expires` : enabling http cache headers on static assets.

And do not use built-in `mod_php`, prefer *PHP-FPM* !

Then use `/samples/apache.conf` template to create your *virtual host* configuration file. It shows how to set rewrite and secure private folders from being viewed from public visitors.

If you do not have access to your Apache virtual host configuration, you can use the built-in *htaccess* generator :

```
bin/roadiz generate:htaccess
```

This command will generate `.htaccess` files in each critical folder to enable PHP scripts or deny public access to forbidden folders.

Standard Edition

`bin/roadiz generate:htaccess` is not needed anymore with *Roadiz Standard edition* as you will configure your *Apache/Nginx* root to `web/` folder only. No source or configuration files will be exposed anymore.

CMS Structure

- `bin/` : Contains the Roadiz CLI executable
- `docker/` : Tools for creating development and production *Docker* image for your project
- `app/` : Contains every runtime resources from configuration to app cache and nodes-sources entities
  - `cache/` : Every cache file for *Twig* templates and Intervention Request images (this folder must be writable for PHP)
  - `conf/` : Your setup configuration file(s) (this folder must be writable for PHP)
  - `gen-src/` : Generated PHP code for Doctrine and your Node-types entities (this folder must be writable for PHP)
  - `logs/` : *Monolog* logs folder
- `files/` : Private documents and font files root (this folder must be writable for PHP)
- `samples/` : This folder contains useful configuration and example files for Apache or Nginx webservers
- `web/` : Your website root, it contains your application entry-points and your public assets
  - `files/` : Public documents (this folder must be writable for PHP)
  - `themes/` : public assets mirror for each theme, this folder contains symlinks to your `themes/YourTheme/static` folder
- `themes/` : Contains your own themes and all your website logic
- `vendor/` : Dependencies folder managed by *Composer*

Installer Roadiz Standard Edition

Pour les nouveaux projets **Roadiz** peut être facilement configuré en utilisant la commande `create-project`.

```
# Create a new Roadiz project
composer create-project roadiz/standard-edition my-website
cd my-website
# Create a new theme for your project and link assets
# path in public folder
bin/roadiz themes:generate --symlink --relative FooBar
# Go to your theme
cd themes/FooBarTheme
# Install JS dependencies with npm or Yarn
yarn # or npm install
# Build your new theme assets
yarn build # or npm run build
```

Composer will prompt you if you want to can versioning history. Choose the default answer `no` as we definitely want to replace *standard-edition 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.

Do not forget to add the following lines to your `app/AppKernel.php` file once you generated a new theme :

```
// app/AppKernel.php
public function register(\Pimple\Container $container)
{
    parent::register($container);
    /*
     * Add your own service providers.
     */
    $container->register(new␣
↪\Themes\FooBarTheme\Services\FooBarThemeServiceProvider());
}
```

Note : Pour les utilisateurs de *Windows*, la commande `bin/roadiz themes:generate --symlink --relative FooBar` peut être utilisée sans l'option `--relative` pour créer des **liens symboliques absolus**. Vous pouvez même installer les ressources de votre thème en tant que *hard copy* sans option `--symlink`. Assurez-vous d'exécuter régulièrement `bin/roadiz themes:assets:install FooBar` lorsque vous utilisez le mode *hard copy* pour mettre à jour vos ressources. Vous devriez utiliser des *liens symboliques* dès que possible pour éviter les problèmes de mise à jour.

Travailler avec les environnements Roadiz

Environnement d'installation

Une fois que vous avez réussi à télécharger Roadiz et ses dépendances, vous devrez configurer sa base de données et toutes les informations nécessaires pour démarrer votre site Web.

Comme toutes les applications *Symfony*, *Roadiz* fonctionne en utilisant des environnements. Par défaut, il y a un environnement *production* qui est géré par le point d'entrée `index.php`. En ce moment, si vous essayez de vous connecter à votre nouveau site web Roadiz, vous obtiendrez une erreur car nous n'avons pas installé sa base de données et ses données essentielles.

Pour pouvoir utiliser l'interface d'installation de Roadiz *installation*, vous devrez appeler le point d'entrée `install.php`. Ce fichier a été généré lorsque vous avez exécuté la commande `composer install`. Cet environnement sera accessible à l'adresse `http://mywebsite.com/install.php`.

Pour des raisons de sécurité, nous avons activé un filtrage d'IP pour ce point d'entrée, vous pouvez ajouter votre propre adresse IP dans le tableau suivant : `array('127. .0.1', 'fe80::1', '::1', ':ffff:127. .0.1', '::ffff:127.0.0.1')`. Ce filtrage IP est très important si vous travaillez sur un serveur public, personne à part vous ne devrait pouvoir accéder au point d'entrée *install*.

À la fin du processus d'installation, vous serez invité à supprimer le fichier `install.php` et à vous connecter à l'URL finale de votre site.

Environnement de développement

L'environnement de *production* de Roadiz n'est pas conçu pour développer vos propres thèmes et étendre les fonctionnalités du back-office. De la même manière que l'environnement *install*, nous avons préparé un environnement *dev* pour désactiver la mise en cache des ressources et activer les fonctionnalités de débogage. Vous trouverez un fichier `dev.php` à la racine de votre site web qui a été généré à la commande `composer install`. De la même manière que pour *install.php*, vous devrez ajouter votre propre adresse IP pour filtrer qui peut accéder à votre environnement *dev*.

Environnement de prévisualisation

L'environnement *preview* n'est pas un vrai car il ajoute seulement un drapeau au noyau de Roadiz pour permettre aux utilisateurs du back-office de voir les nœuds non publiés. Par défaut, il est disponible en utilisant le point d'entrée `preview.php`, sauf si vous décidez de le supprimer.

Note : Since Roadiz v1.6, preview environment is deprecated in favor of `?_preview=1` query param which allows previewing in any environment, especially for API responses (JSON).

Environnement de production

C'est le point d'entrée `index.php` par défaut qui sera appelé par tous vos visiteurs. Il n'y a pas de restriction à ce sujet et il lancera l'application Roadiz en utilisant les politiques de mise en cache les plus fortes. Il n'est donc pas recommandé d'utiliser pour le développement (vous devrez vider les caches chaque fois que vous changez quelque chose dans le code).

Environnement pour le vidage du cache

L'environnement *clear_cache* est uniquement destiné à vider le cache Roadiz sans exécuter toute l'application. Il peut être utile si vous utilisez un cache op-code comme *APC* ou PHP natif *OPcache*. Ces caches spécials ne peuvent pas être purgées depuis utilitaires en ligne de commande, donc vous devez appeler un script PHP depuis votre navigateur ou via `curl` pour les vider. Comme l'environnement *install* et *dev*, `clear_cache.php` est restreint par IP pour ne pas permettre à tout le monde de vider vos caches applicatifs. Vous devrez ajouter votre propre adresse IP pour filtrer qui peut y accéder.

Utiliser Docker pour le développement

La Standard Edition de Roadiz est fournie avec un exemple d'environnement `docker-compose` prêt à être utilisé pour le développement. *Docker* sur Linux fournira des performances natives et un environnement quasiment identique à la production sans avoir d'installer des paquets supplémentaires votre machine de développement. Les performances ne seront pas aussi bonnes sur les hôtes *macOS* ou *Windows*, mais cela évitera l'installation de PHP et de MySQL directement sur votre ordinateur.

Premièrement, éditez le fichier `.env` et configurez-le selon votre machine hôte (vous pouvez le copier depuis `.env.dist` s'il n'existe pas).

```
# Build PHP image
docker-compose build;

# Create and start containers
docker-compose up -d;
```

Ensuite, votre site web sera disponible sur `http://localhost:${APP_PORT}`.

Pour les utilisateurs de Linux, où *Docker* fonctionne nativement (sans virtualisation sous-jacente), faites attention au fait que *PHP* fonctionne avec l'utilisateur *www-data*. Vous devez mettre à jour votre fichier `.env` pour refléter votre utilisateur local **UID** pendant la compilation d'image.

```
# Type id command in your favorite terminal app
id
# It should output something like
# uid=1000(toto)
```

Utilisez donc le même uid dans votre fichier *.env* **avant** de commencer et de construire votre image Docker.

```
USER_UID=1000
```

Using Vagrant for development

Roadiz comes with a dedicated `Vagrantfile` which is configured to run the official `roadiz/standard-edition` box with a *LEMP* stack (Nginx + PHP7.0-FPM + MariaDB), a *phpMyAdmin*, a *Mailcatcher* and an *Apache Solr server*. This will be useful to develop your website on your local computer.

Note : *Git*, *Composer*, *Virtual Box* and *Vagrant* must be setup on your local computer before going further into Vagrant development.

- https://getcomposer.org/download/
- https://www.virtualbox.org/
- https://www.vagrantup.com/

Once you've created your Roadiz project, *Composer* should has copied `samples/Vagrantfile.sample` file as `Vagrantfile` at your project root. Then do a `vagrant up` in Roadiz' folder. Then Vagrant will run your code in `/var/www` and you will be able to completely use `bin/roadiz` commands without bloating your computer with lots of binaries.

Once vagrant VM has provisioned you will be able to use :

- `http://192.168.33.10/install.php` to proceed to install.
- `http://192.168.33.10:8983/solr` to use *Apache Solr* admin.
- `http://192.168.33.10/phpmyadmin` for your *MySQL* db admin.
- `http://192.168.33.10:1080` for your *Mailcatcher* tool.

Do not hesitate to add an entry in your `/etc/hosts` file to use a local *domain name* instead of using the private IP address (eg. http://site1.dev). And for each Vagrant website, **do not forget to increment your private IP**.

```
# /etc/hosts
# Vagrant hosts
192.168.33.10    site1.dev
192.168.33.11    site2.dev
# ...
```

Note : Be careful, **Windows users**, this `Vagrantfile` is configured to use a *NFS* fileshare. Disable it if you did not setup a *NFS* emulator. For *OS X* and *Linux* user this is built-in your system, so have fun !

Access entry-points

`web/install.php`, `web/clear_cache.php` and `web/dev.php` entry points are IP restricted to *localhost*. To be able to use them with a *Vagrant* setup, you'll need to add your host machine IP to the `$allowedIp` array. We already set two IP for you that should work for *forwarded* and *private* requests. Just uncomment the following lines in these files and edit them if necessary.

```
$allowedIp = [
    '10.0.2.2',     // vagrant host (forwarded)
    '192.168.33.1', // vagrant host (private)
    '127.0.0.1', 'fe80::1', '::1' // localhost
];
```

Database and Solr credentials

Roadiz *Vagrant* box provides standard *MariaDB* and *Apache Solr* servers which run automatically at launch. Here are their default credentials :

Database credentials

— Host : `localhost`
— User : `roadiz`
— Password : `roadiz`
— Database : `roadiz` or `roadiz_test` (for executing unit tests)

Solr credentials

— Host : `localhost`
— Core : `roadiz` or `roadiz_test` (for executing unit tests)
— User : *none*
— Password : *none*

Avertissement : Of course, this *Vagrant* virtual machine should not be used for any *production* environment. You can find provisioning scripts on our Github repository, feel free to make enhancement sugggestions about them.

Full config.yml example for Vagrant

```
---
appNamespace: "my-roadiz-project"
timezone: "Europe/Paris"
doctrine:
    driver: "pdo_mysql"
    host: "localhost"
    user: "roadiz"
    password: "roadiz"
    dbname: "roadiz"
    charset: utf8mb4
    default_table_options:
        charset: utf8mb4
        collate: utf8mb4_unicode_ci
cacheDriver:
    type: ~
```

```
    host: ~
    port: ~
security:
    secret: "my-roadiz-project"
mailer:
    type: ~
    host: "localhost"
    port: 25
    encryption: false
    username: ""
    password: ""
entities:
    - ../vendor/roadiz/roadiz/src/Roadiz/Core/Entities
    - ../vendor/roadiz/models/src/Roadiz/Core/AbstractEntities
    - gen-src/GeneratedNodeSources
rememberMeLifetime: 2592000
additionalServiceProviders: []
additionalCommands: []
assetsProcessing:
    driver: gd
    defaultQuality: 90
    maxPixelSize: 1920
    jpegoptimPath: /usr/bin/jpegoptim
    pngquantPath: /usr/bin/pngquant
solr:
    endpoint:
        localhost:
            host: "localhost"
            port: "8983"
            path: "/solr"
            core: "roadiz"
            timeout: 3
            username: ""
            password: ""
```

Using PHP server for development

If *Vagrant* is too heavy for your purpose, you can simply use *PHP built-in server*.

Following command will launch a web-server listening on all IP addresses on port 8080. We also use a dedicated *router* file to serve static and generated resources :

```
# Create a new Roadiz project
composer create-project roadiz/standard-edition
# Create a new theme for your project
cd standard-edition
bin/roadiz themes:generate --symlink --relative FooBar

# Launch PHP server with web folder as root
php -S 0.0.0.0:8080 -t web vendor/roadiz/roadiz/conf/router.php
# OR use Makefile recipe
make dev-server
```

Standard Edition has a `Makefile` recipe for launching internal PHP server with a chosen port and IP : `make dev-server`.

If you want to use PHP internal server, make sure you have installed all required PHP extensions and that you have a database server :

— You can use a local MySQL/MariaDB server
— Or use a SQLite3 database, just use `../app/conf/database.db3` path during install.

> PHP web server was designed to aid application development. It may also be useful for testing purposes or for application demonstrations that are run in controlled environments. It is not intended to be a full-featured web server. It should not be used on a public network.

Use Mailhog to catch outgoing emails

Contrary to *Vagrant*, using PHP built-in server does not provide any additional tool such as *Mailcatcher* or *Apache Solr*. You can setup Mailhog to catch outgoing emails in a clean web interface.

On *macOS*, use *HomeBrew* : `brew update && brew install mailhog`, then configure your PHP `sendmail_path` to use it.

Configuration manuelle

Cette section explique comment le fichier de configuration principal `app/conf/config.yml` fonctionne. C'est beaucoup plus pratique que de lancer le thème d'installation pour chaque mise à jour de configuration.

Votre fichier `app/conf/config.yml` est construit en utilisant la syntaxe YAML. Chaque partie correspond à la configuration d'un *service* de Roadiz.

Note : Par défaut, chaque environnement Roadiz lit le fichier de configuration `app/conf/config.yml`. Mais vous pouvez spécifier différents fichiers pour les environnements `dev` et `test`. Il suffit de créer un fichier `app/conf/config_dev.yml` ou `app/conf/config_test.yml` pour remplacer les paramètres par défaut. Vous pourrez utiliser une base de données différente, un mailer ou une instance *Solr* pour ne pas polluer votre environnement de production.

Source Edition

Roadiz Source edition stocke les fichiers de configuration dans le dossier `conf/`.

DotEnv

`app/conf/config.yml` can resolve `.env` variables if you want to add this file to your Git repository or Docker images. For example :

```
doctrine:
    driver: "pdo_mysql"
    host: '%env(string:MYSQL_HOST)%'
    user: '%env(string:MYSQL_USER)%'
    password: '%env(string:MYSQL_PASSWORD)%'
    dbname: '%env(string:MYSQL_DATABASE)%'
    server_version: '%env(string:MYSQL_VERSION)%'
    # "utf8mb4" charset requires at least mysql 5.7
    # due to large index requirement.
    # otherwise change it to "utf8"
    charset: utf8mb4
    default_table_options:
        charset: utf8mb4
        collate: utf8mb4_unicode_ci
```

This configuration will be resolved against `.env` :

```
# MySQL
MYSQL_ROOT_PASSWORD=root
MYSQL_HOST=db
```

```
MYSQL_DATABASE=roadiz
MYSQL_USER=roadiz
MYSQL_PASSWORD=roadiz
MYSQL_VERSION=8.0
```

Doctrine

La section de configuration la plus importante traite de la connexion à la base de données qui est gérée par *Doctrine* :

```
doctrine:
    driver: "pdo_mysql"
    host: "localhost"
    user: ""
    password: ""
    dbname: ""
```

Roadiz utilise *Doctrine ORM* pour stocker vos données. Il passera directement cette configuration YAML à *Doctrine* afin que vous puissiez utiliser tous les pilotes et options disponibles que vous trouverez dans la documentation officielle : http://doctrine-dbal.readthedocs.org/en/latest/reference/configuration.html

Choisissez votre modèle d'héritage

La caractéristique principale de *Roadiz* concerne son modèle de données polymorphique qui est stocké sur une base de données relationnelle. Cela nécessite une structure complexe qui peut conduire à des soucis de performance lorsque vous créez plus de 20-30 types de nœuds. Nous avons donc rendu le modèle d'héritage de données configurable pour permettre le passage au schéma single_table qui sera plus performant si vous avez besoin de beaucoup de types de nœuds. Cependant ce modèle de *classe unique* désactivera le support des champs indexables et vous ne pourrez pas créer de champs avec le *même nom mais pas le même type* car tous les champs de type de nœud seront créés dans la **même table SQL**.

Si vous avez vraiment besoin de créer des champs indexables et de mélanger les types de champs, nous vous conseillons de conserver le type d'héritage original joined table qui crée une table SQL dédiée pour chaque type de nœud. L'héritage basé sur des *tables jointes* peut être très utile avec un petit nombre de types de nœud (max. 20) et des champs très différents. Mais son principal inconvénient est que Roadiz a besoin de *LEFT JOIN* chaque table de type de nœud pour chaque requête générique avec les node-source, **sauf si vous spécifiez un critère de type de nœud**.

Vous pouvez configurer la stratégie *Doctrine* pour les classes d'héritage NodesSources dans `app/conf/config.yml` :

```
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.

Thèmes

Depuis *Roadiz v1.0*, les thèmes sont statiquement enregistrés dans la configuration de Roadiz pour de meilleures performances et retarder au maximum la connexion à la base de données. Vous pouvez enregistrer n'importe quel

thème front-end dans votre fichier `app/conf/config.yml`. La priorité du thème n'est pas gérée ici mais dans chacun de vos thèmes en écrasant la valeur statique `$priority;`

```
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.

Pilotes de cache

Lorsqu'il est défini en tant que *null*, les pilotes de cache seront automatiquement choisis par Roadiz en fonction de votre configuration PHP et des extensions disponibles.

Parfois, si une extension de cache est disponible mais que vous ne voulez pas l'utiliser, vous devrez spécifier un autre type de pilote de cache (utilisez `array` pour désactiver les caches). C'est un cas connu lorsque vous utilisez des plans d'hébergement mutualisé *OVH* qui fournissent une extension PHP *memcached* mais ne vous permet pas de vous y connecter.

```
cacheDriver:
    type: null
    host: null
    port: null
```

Les types de cache disponibles sont :
- *apc*
- *xcache*
- *memcache* (nécessite la configuration `host` et `port`)
- *memcached* (nécessite la configuration `host` et `port`)
- *redis* (nécessite la configuration `host` et `port`)
- *array*

Gestionnaires Monolog

Par défaut, Roadiz écrit ses logs dans le dossier `app/logs/` dans un fichier nommé d'après votre environnement d'exécution (par exemple. `roadiz_prod.log`). Mais vous pouvez aussi personnaliser *Monolog* pour utiliser trois gestionnaires différents. Attention : l'utilisation de gestionnaires de logs personnalisés désactivera la journalisation Roadiz par défaut (sauf pour la journalisation via *Doctrine*), il est donc préférable d'utiliser toujours le gestionnaire *default* en plus de votre gestionnaire personnalisé.

Types de gestionnaire disponibles :
- `default` : gestionnaire par défaut de Roadiz qui écrit dans le dossier `app/logs/` dans un fichier nommé d'après votre environnement en cours d'exécution
- `stream` : Définit un flux de fichiers journaux sur votre système. **Votre chemin doit être accessible en écriture !**

— `rotating_file` : Defines a log file stream on your local system which will be rotated to avoid large files. **Your path must be writable !**
— `syslog` : Écrit dans le système *syslog*.
— `gelf` : Envoie des messages formatés GELF à un point d'entrée externe défini par la valeur *url*. Roadiz utilise un gestionnaire qui **ne déclenchera aucune erreur** si votre point d'entrée externe n'est pas joignable, assurez-vous qu'il est correct. C'est une bonne idée de combiner le gestionnaire *gelf* avec un système de journalisation local si votre point d'entrée externe est en panne.
— `sentry` : Envoyer les logs à votre instance *Sentry*. **Nécessite une bibliothèque PHP sentry/sentry** : `composer require sentry/sentry php-http/curl-client guzzlehttp/psr7`. C'est une bonne idée de combiner un gestionnaire *sentry* avec un système de journalisation local si votre point d'entrée externe est en panne.

Les valeurs `type` et `level` sont obligatoires pour chaque gestionnaire.

Voici un exemple de configuration :

```
monolog:
    handlers:
        default:
            type: default
            level: INFO
        file:
            type: stream
            # Be careful path must be writable by PHP
            path: '%kernel.log_dir%/roadiz.log'
            level: INFO
            level: INFO
        rotate:
            type: rotating_file
            path: '%kernel.log_dir%/roadiz.log'
            level: DEBUG
        syslog:
            type: syslog
            # Use a custom identifier
            ident: my_roadiz
            level: WARNING
        graylog:
            type: gelf
            # Gelf HTTP entry point url (with optional user:passwd authentication)
            url: http://graylog.local:12202/gelf
            level: WARNING
        sentry:
            type: sentry
            level: WARNING
            url: https://xxxxxx:xxxxxx@sentry.io/1
```

Point d'entrée Solr

Roadiz peut utiliser un moteur de recherche *Apache Solr* pour indexer les nodes-sources. Ajoutez-le à votre *config.yml* pour lier votre CMS à votre serveur *Solr* :

```
solr:
    endpoint:
        localhost:
            host: "localhost"
            port: "8983"
            path: "/"
            core: "mycore"
            timeout: 3
            username: ""
            password: ""
```

La commande CLI Roadiz peut facilement gérer l'index Solr. Il suffit de taper `bin/roadiz solr:check` pour obtenir plus d'informations.

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.

```
reverseProxyCache:
    frontend:
        localhost:
            host: localhost
            domainName: myapp.test
        external:
            host: varnish
            domainName: myapp.test
```

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 :

```
reverseProxyCache:
    frontend: []
    cloudflare:
        zone: cloudflare-zone
        bearer: ~
```

Ou avec votre E-mail et votre AuthKey :

```
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 utilise *Doctrine* pour synchroniser les entités aux tables de votre base de données. Afin de rendre Roadiz extensible, vous pouvez ajouter vos propres chemins à la section `entities`.

```
entities:
    - "../vendor/roadiz/roadiz/src/Roadiz/Core/Entities"
    - "../vendor/roadiz/models/src/Roadiz/Core/AbstractEntities"
    - "gen-src/GeneratedNodeSources"
```

Configurer le mailer

Roadiz utilise *Swift Mailer* pour envoyer des emails. Cette bibliothèque géniale est construite pour gérer différents types de transports de courrier et de protocoles. Par défaut, Roadiz utilise votre configuration locale avec `sendmail` mais il est fortement recommandé d'utiliser un autre transport (comme un serveur SMTP externe) dans votre fichier `app/conf/config.yml`.

Vous pouvez utiliser *SSL*, *TLS* ou aucun chiffrement.

```
mailer:
    type: "smtp"
    host: "localhost"
    port: 25
    encryption: false
    username: ""
    password: ""
```

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 utilise la bibliothèque Image Intervention pour créer automatiquement une version optimisée de votre image si elle est trop grande. Vous pouvez définir les seuils dans la section `assetsProcessing`. `driver` et `defaultQuality` seront également utilisés pour le traitement d'images à la volée avec la bibliothèque Intervention Request.

```
assetsProcessing:
    # gd or imagick (gd does not support TIFF and PSD formats)
    driver: gd
    defaultQuality: 90
    # pixel size limit () after roadiz
    # should create a smaller copy.
    maxPixelSize: 1280
    # Path to jpegoptim binary to enable jpeg optimization
    jpegoptimPath: ~
    # Path to pngquant binary to enable png optimization (3x less space)
    pngquantPath: ~
    # List additionnal Intervention Request subscribers
    subcribers: []
```

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.

```
assetsProcessing:
    # List additionnal Intervention Request subscribers
    subcribers:
        - 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.

```
assetsProcessing:
    # List additionnal Intervention Request subscribers
    subcribers:
        - 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/roadiz
```

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

```
php ./bin/roadiz
```

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/roadiz 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**.

Commandes supplémentaires

Si vous développez votre propre thème, vous devrez peut-être créer des commandes CLI personnalisées. Roadiz peut gérer des commandes supplémentaires si vous les ajoutez dans votre `app/conf/config.yml` comme

vous le feriez pour n'importe quelle *entités* additionnelle. Assurez-vous que toutes les commandes supplémentaires étendent la classe `Symfony\Component\Console\Command\Command`.

```
additionalCommands:
    - \Themes\DefaultTheme\Commands\DefaultThemeCommand
```

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 Roadiz command (MySQL/MariaDB only) : `bin/roadiz database:dump -c` will generate a SQL file in `app/` folder
— 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/roadiz migrations:migrate;
```

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

```
bin/roadiz generate:nsentities;
```

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

```
bin/roadiz orm:schema-tool:update --dump-sql;
# Upgrade node-sources tables if necessary
bin/roadiz orm:schema-tool:update --dump-sql --force;
```

Then, clear your app caches :

```
# Clear cache for each environment
bin/roadiz cache:clear -e dev
bin/roadiz cache:clear -e prod
bin/roadiz cache:clear -e prod --preview
bin/roadiz cache:clear-fpm -e dev
bin/roadiz cache:clear-fpm -e prod
bin/roadiz cache:clear-fpm -e prod --preview
```

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.

Moving a website to another server

Before moving your website, make sure you have backed up your data :

— Dump your database, using classic `mysqldump` or `pg_dump` tools. If you're using *MySQL* `bin/roadiz database:dump -c` command can speed-up the process by naming automatically your file against your *app-namespace*.
— Archive your files using `bin/roadiz files:export`, Roadiz will create a ZIP file with your public/private documents and fonts.

Moving to a SSH+Git hosting plan or an other development machine

From this point you can install your new web-server, as described in *Install section*. Pay attention that if your theme needs some additional *composer* dependencies you should *clone/copy* it into your *themes/* folder **before** running `composer install --no-dev`.

Then import your dump and files into your new server.

Once you've imported your database, you must edit manually your *conf/config.yml*, you can reuse the former server's one and adapt its database credentials.

Avertissement : Do not perform any schema update if no app/gen-src/GeneratedNodeSources classes is available, it will erase your NodesSources data as their entities files haven't been generated yet.

When you have edited your `app/conf/config.yml` file, regenerate your *Doctrine* entities class files :

```
bin/roadiz generate:nsentities;
```

Now you can perform a schema update without losing your nodes data :

```
bin/roadiz orm:schema-tool:update --dump-sql;
bin/roadiz orm:schema-tool:update --force;
bin/roadiz cache:clear -e prod
bin/roadiz cache:clear -e prod --preview
bin/roadiz cache:clear-fpm -e prod
bin/roadiz cache:clear-fpm -e prod --preview
```

Note : If you are using an OPcode cache like XCache or APC, you'll need to purge cache manually because it can't be done from a CLI interface as they are shared cache engines. The most effective way is to restart your *PHP-FPM* service or *Apache* if your are using *mod_php*.

Install assets

By default, web theme assets are not tracked by *Git*. Make sure that you installed every theme assets using `bin/roadiz themes:assets:install` command. You should use `--symlink --relative` options when possible to prevent updating assets manually. For Windows users, remove `--relative` option to create *absolute* symlinks.

```
bin/roadiz themes:assets:install --symlink --relative Debug;
bin/roadiz themes:assets:install --symlink --relative Install;
bin/roadiz themes:assets:install --symlink --relative Rozier;
# Then your theme (FooBarTheme)
bin/roadiz themes:assets:install --symlink --relative FooBar;
```

composer.json file should execute Roadiz default themes asset install automatically after each update or install.

Synchronize documents and fonts

You can move your `files/` folder using SFTP but the best way is to use `rsync` command as it will upload only newer files and it is much faster.

```
# This will synchronize files on your production server from your local Roadiz␣
↪setup.
# Do not forget ending slash after each path!
rsync -avcz -e "ssh -p 22" /path/to/roadiz/files/ user@my-prod-server.com:/path/to/
↪roadiz/files/
rsync -avcz -e "ssh -p 22" /path/to/roadiz/web/files/ user@my-prod-server.com:/
↪path/to/roadiz/web/files/
```

It works in the other way too. If you want to work on your local copy with up to date files and fonts, you can download actual files from the production website :

```
# This will synchronize files on your local development server from your␣
↪production server.
# Do not forget ending slash after each path!
rsync -avcz -e "ssh -p 22" user@my-prod-server.com:/path/to/roadiz/files/ /path/to/
↪roadiz/files/
rsync -avcz -e "ssh -p 22" user@my-prod-server.com:/path/to/roadiz/web/files/ /
↪path/to/roadiz/web/files/
```

Moving to a non-SSH hosting plan

You have nearly finished your brand new website using Roadiz. You have been working on your own server using Git and Composer, up to this point everything went well.

Now you have to push to production, but your prod-server has no SSH connection. You are stuck with an SFTP connection or worst, an old FTP one. Don't panic, it will take a little more time but it is still possible.

Avertissement : Many shared-plan hosting companies offer you only one or two databases. When moving a Roadiz website, make sure that your database is empty and do not contain orphan tables, you must respect the rule "One app = One database".

Note : If you can ZIP on your production server or if you are going to push your files via FTP, do not forget to exclude `.git` and `node_modules` folders ! These folders have **lots** of useless files for a production SSH-less environment. Here is a sample ZIP command to exclude them : `zip -r mywebsite.zip mywebsite/ -x "mywebsite/.git/*" "mywebsite/themes/**/static/node_modules/*"`.

— Before transferring your website, make sure you have `.htaccess` file in every sensitive folders. You can use the `bin/roadiz generate:htaccess` on your computer.
— If you have at least SFTP, you should have to rights to zip/unzip on your distant server. So zip the whole Roadiz folder.
— If you only have FTP, you must be prepared to transfer your Roadiz folder, file-by-file. Just get yourself a nice cup of coffee.
— Once everything is copied on your production server, verify than you have the same files as on your dev-server.
— Import your database dump with phpmyadmin or pgmyadmin.
— Edit your `conf/config.yml` to match your new database credentials.
— Verify that root `.htaccess` file contains every information to enable Apache url-rewriting.
— Try to connect to your website
— If it doesn't work or display anything, read your PHP log file to understand where the problem comes from. It might be your database credentials or an outdated PHP version. Check that your hosting manager has installed every needed PHP extensions, see *Requirements*.

Installer Standard Edition sur un hébergement mutualisé

. . . sans commandes SSH et FTP

Les CMS modernes basés sur les commandes *Composer* et *CLI* ne peuvent pas être déployés facilement sur des environnements d'hébergement mutualisés sur lesquels seul FTP est disponible. Voici quelques outils pratiques pour déployer un *Roadiz* avec FTP.

La première condition est que vous devrez configurer un environnement local qui sera un miroir des fichiers de votre FTP le dossier, `vendor/` inclus. . . oui. Prenez une très longue tasse de café lorsque vous initiez le premier envoi FTP, ce sera long, très long. Les envois suivants ne feront que pousser les nouveaux fichiers.

La seconde condition est que vous devez créer toutes vos entités de type de nœud sur votre environnement local d'abord pour pouvoir faire un miroir de toutes les classes `GeneratedNodeSources\*` car vous ne serez pas en mesure de les générer sur votre hébergement mutualisé.

Préparez votre environnement local avec Makefile

Standard-edition est fourni avec un exemple de `Makefile` afin que vous puissiez écrire une recette `push-prod` pour automatiser tous les processus en utilisant le paquet `lftp`. Assurez-vous qu'il est bien installé sur votre ordinateur :

```
# macOS
brew install lftp
# Ubuntu/Debian
apt-get install lftp
# ...
```

Cette recette effacera vos fichiers de cache, elle générera les fichiers Apache `.htaccess`, copiera les ressources de votre thème en tant que vrais fichiers, miroir de tous les fichiers nécessaires sans toutes les exceptions (la liste peut être améliorée) puis copiera les ressources de votre thème à l'aide de liens symboliques.

```
push-prod:
    make cache
    bin/roadiz generate:htaccess
    bin/roadiz themes:assets:install ${THEME_PREFIX}
    lftp -e "mirror --only-newer --parallel=3 -R \
        --exclude '/\..+/$$' \
        -x 'app/conf/config\.yml' \
        -x '\.env' \
        -x '(README\.md|Makefile|Dockerfile|Vagrantfile)' \
        -x '(bin|docker|samples|tmp|\.git|\.idea|files)/' \
        -x 'app/(cache|logs|sessions|tmp)/' \
        -x 'web/files/' \
        -x 'node_modules/' \
        -x 'bower_components/' \
        -x 'themes/${THEME}/(app|node_modules|webpack)/' \
        -x '\.(psd|rev|log|cmd|bat|pif|scr|exe|c?sh|reg|vb?|ws?|sql|db)$$' \
        ./ ${FTP_REMOTE_PATH}" -u ${FTP_USER},${FTP_PASS} ${FTP_HOST}
    bin/roadiz themes:assets:install --relative --symlink ${THEME_PREFIX}
```

Assurez-vous que votre configuration correspond à votre plan d'hébergement partagé, par exemple, ajustez votre driver de cache sur `file` ou `php` pour obtenir des performances décentes.

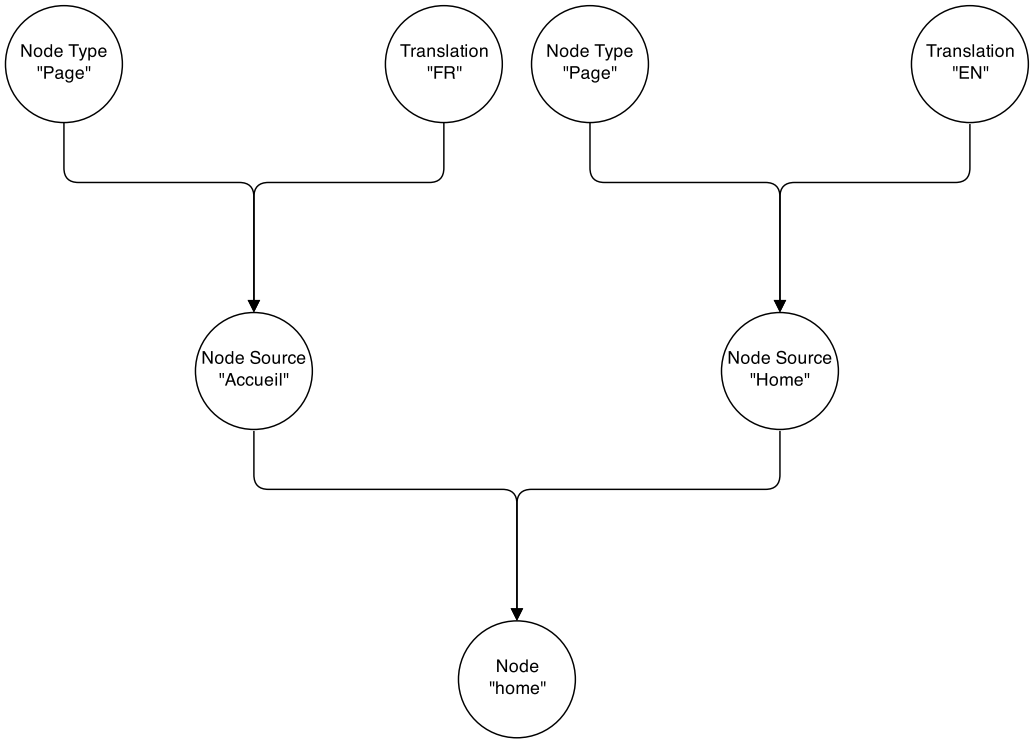
```
cacheDriver:
    type: file
    host: null
    port: null
```

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.

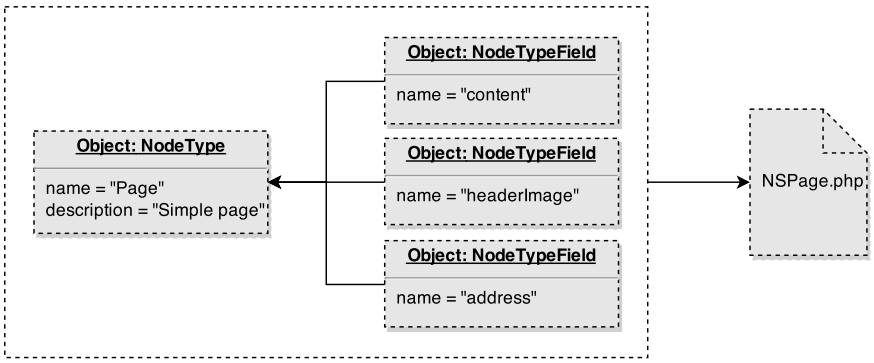
Regardons ce simple schéma de nœud avant de l'expliquer.



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

Qu'est-ce qu'un type de nœud

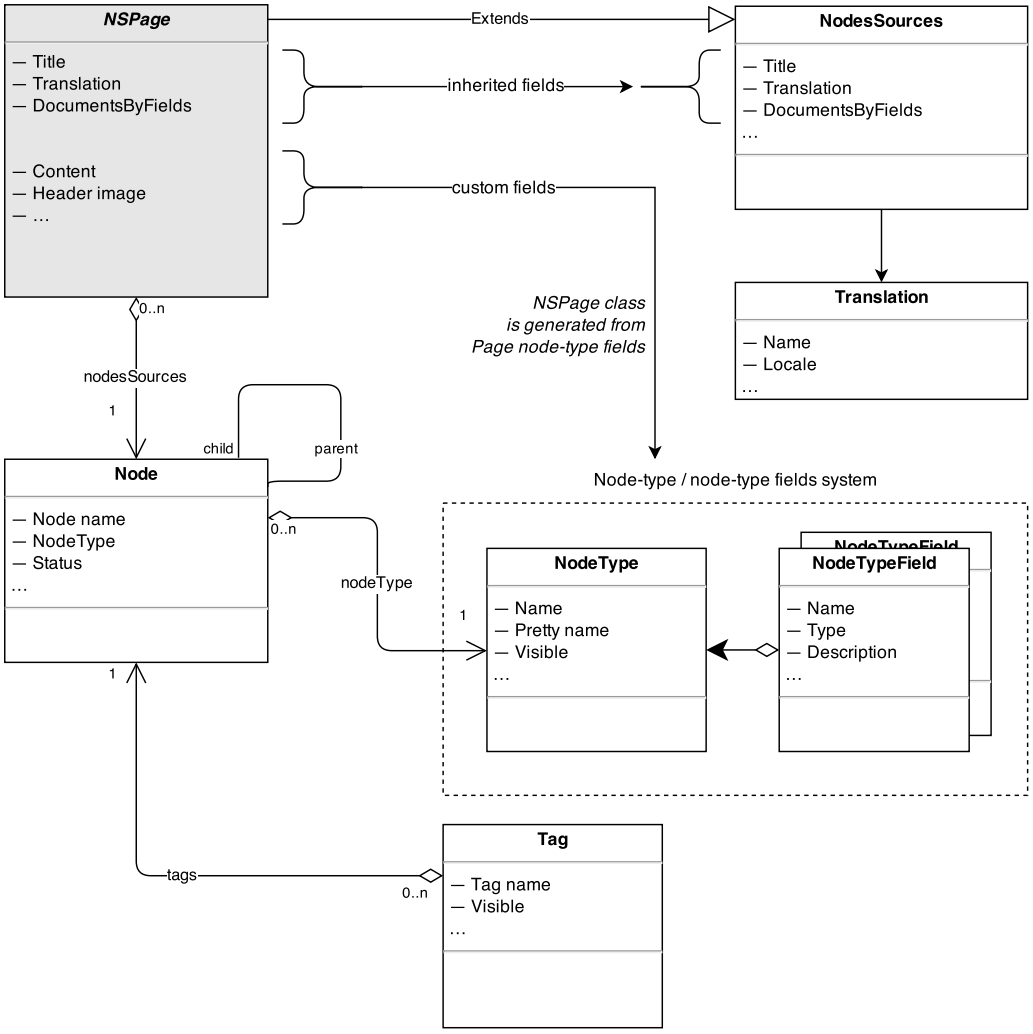
Un type de nœud est le gabarit de votre node-source. Il contiendra tous les champs que Roadiz utilisera pour générer une classe de node-source étendue.



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/roadiz nodetypes:add-fields
bin/roadiz nodetypes:create
bin/roadiz nodetypes:delete
bin/roadiz 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/roadiz orm:schema-tool: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

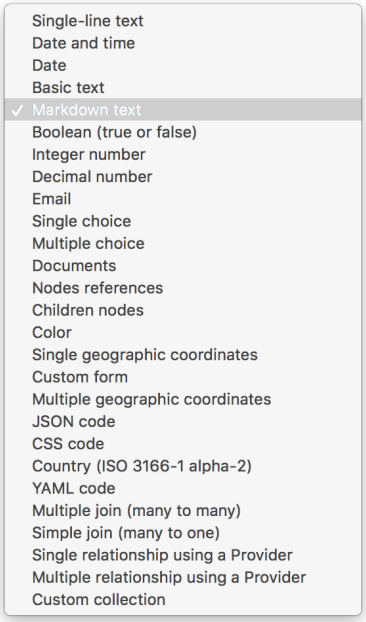
Les champs suivants stockent des données simples dans votre table de base de données de nœuds.

— Chaîne de caractères
— Date
— Date et heure
— Texte basic
— Texte Markdown
— Booléen
— Nombre entier
— Nombre décimal
— E-mail
— Couleur
— Coordonnées géographiques uniques
— 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

Note : Le champ *coordonnées géographiques uniques* stocke ses données au format JSON. Assurez-vous que vous n'avez pas de données déjà inscrite à la main dans la base de donnée.

Avertissement : Pour utiliser les *coordonnées géographiques uniques*, vous devez créer un compte *Google API Console* avec *Maps API v3* activé. Ensuite, créez une *clé de navigateur* et collez-la dans le paramètre "Google Client ID" dans les paramètres Roadiz pour activer les champs de type de nœud *géographique*. Si vous ne l'avez pas fait, une simple entrée de texte sera affichée au lieu de *Widget Roadiz*.



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->get('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->get(ManagerRegistry::class)->getRepository('GeneratedNodeSources\NSBlogPost
↪')->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->get('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 hydratation. 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->get('nodeSourceApi')->getBy([
    'node.nodeType' => $this->get('nodeTypesBag')->get('Page'),
    'node.parent' => $nodeSource,
    'node.visible' => true,
    'publishedAt' => ['>=', new \DateTime()],
    'translation' => $nodeSource->getTranslation(),
],[
    'publishedAt' => 'DESC'
]);
```

Visibility

There are two parametres 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 :

```
/** @var Registry $registry */
$registry = $this->get('workflow.registry');
if ($registry->get($node)->can($node, 'publish')) {
    $registry->get($node)->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 : *Generating paths and url*.

```
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\Core\Events\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\Core\Events\NodesSources\NodesSourcesPathGeneratingEvent;

class LinkPathGeneratingEventListener implements EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        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(NodesSourcesEvents::NODE_SOURCE_PATH_
↪GENERATING, $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();
            }
        }
    }
}
```

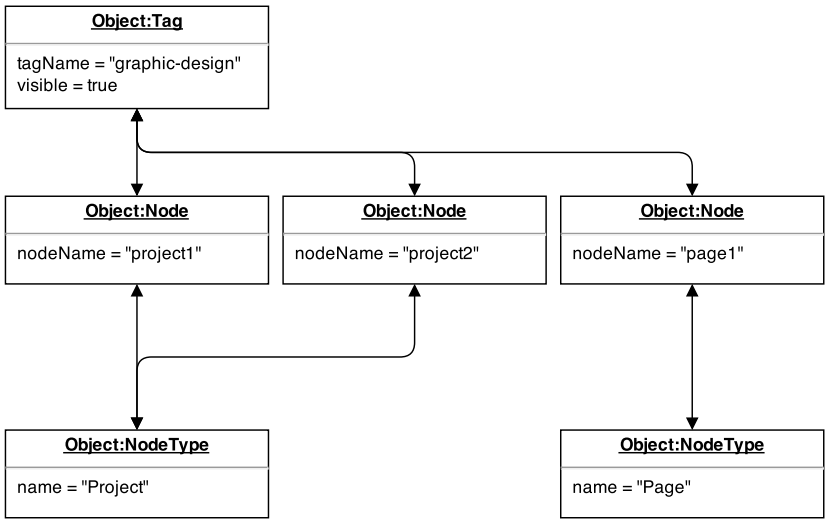
Then register your subscriber to the Roadiz event dispatcher in your theme `setupDependencyInjection` :

```
/** @var EventDispatcher $dispatcher */
$dispatcher = $container['dispatcher'];
$dispatcher->addSubscriber(new LinkPathGeneratingEventListener());
```

This method has an other great benefit : it allows your path logic to be cached inside node-source url' cache provider, instead of generating your custom URL inside your Twig templates or PHP controllers.

3.1.3 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 :

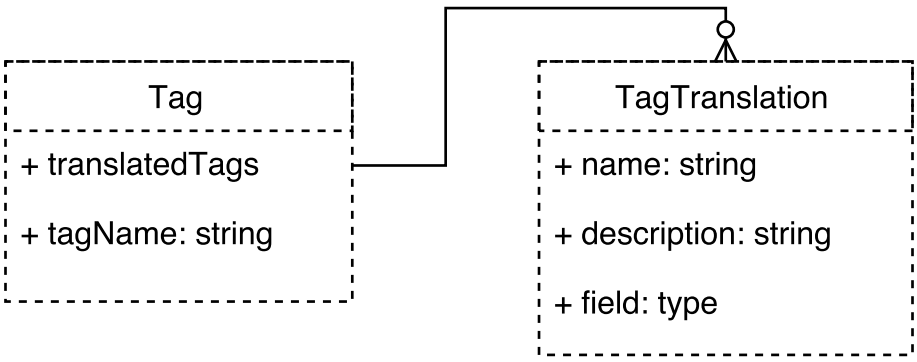


You can see that tags can gather heterogeneous nodes coming from different types (pages and projects). Tags can be used to display a category-navigation on your theme or to simply tidy your backoffice node database.

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*.

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.

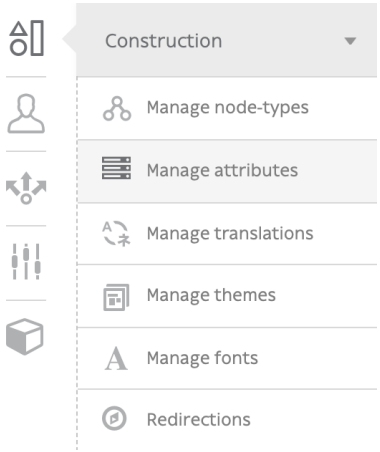
They are available with `getDocuments()` method and will be ordered by **position only**.

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>
```

3.1.4 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.

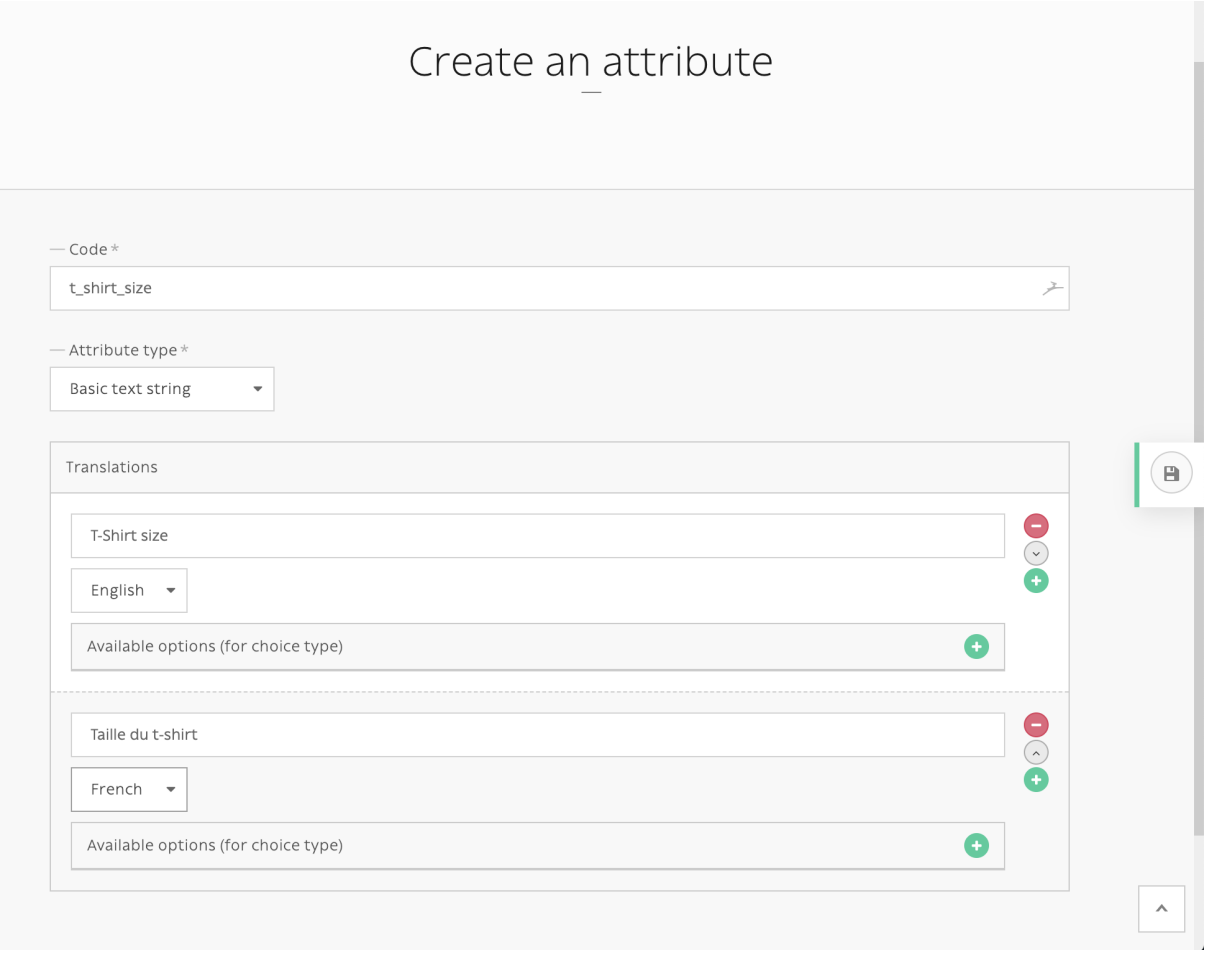
```
<ul class="block-attributes">
    {% for attributeValueTranslation in nodeSource|attributes %}
        <li>
            {% if attributeValueTranslation.attribute.documents|length %}
                {% for document in attributeValueTranslation.attribute.documents %}
                    <figure>{{ 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>
```

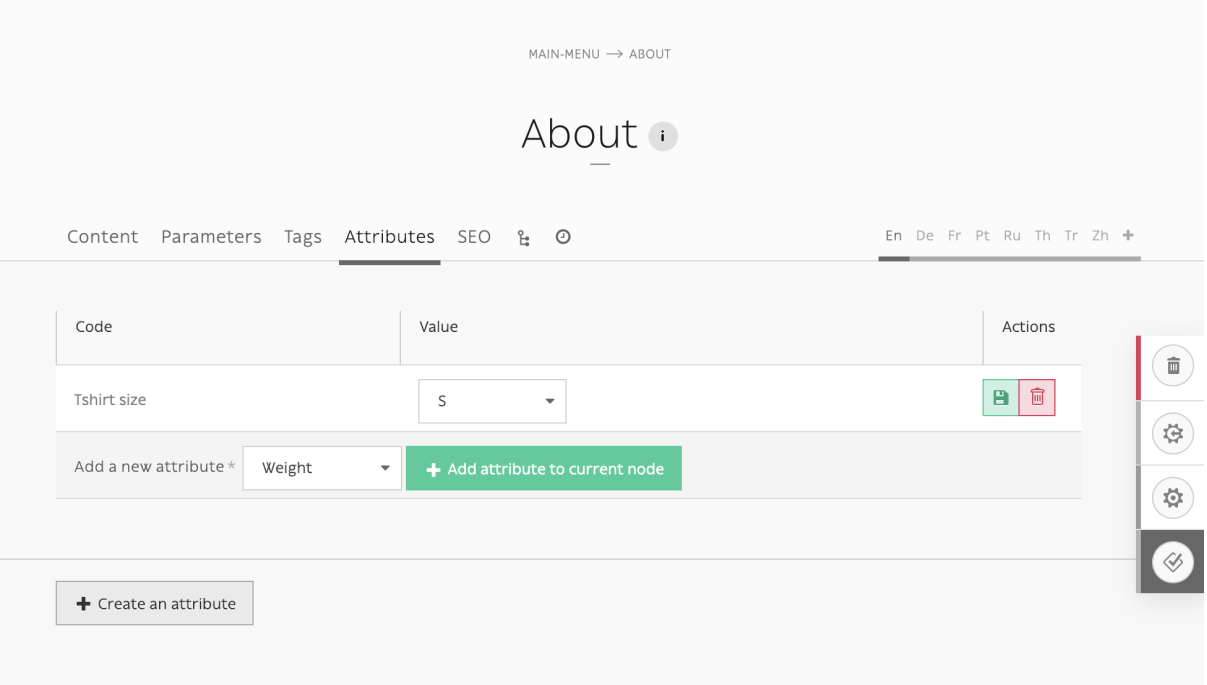
Types d'attributs



- *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`

3.1.5 Go headless

Since Roadiz v1.6, you can create a 100% *headless* project using :

```
# Create a new Roadiz project
composer create-project roadiz/headless-edition myheadless_project
```

No more theme, just business logic

Headless edition get rid of themes logic in favor of a simple `src/` folder to add custom business logic. All API features are handled by *AbstractApiTheme* which is registered as a *composer* dependency and in your `src/AppKernel.php` file.

```
# src/AppKernel.php
public function register(Container $container): void
{
    parent::register($container);
    // Headless edition: do not remove API services
    $container->register(new␣
↪\Themes\AbstractApiTheme\Services\AbstractApiServiceProvider());
    $container->register(new \App\AppServiceProvider());

    /*
     * Add your own service providers.
     */
}
```

Headless structure

- `bin/` : Contains the Roadiz CLI executable
- `docker/` : Tools for creating development and production *Docker* image for your project
- `app/` : Contains every runtime resources from configuration to app cache and nodes-sources entities
  - `cache/` : Every cache file for *Twig* templates and Intervention Request images (this folder must be writable for PHP)
  - `conf/` : Your setup configuration file(s) (this folder must be writable for PHP)
  - `gen-src/` : Generated PHP code for Doctrine and your Node-types entities (this folder must be writable for PHP)
  - `logs/` : *Monolog* logs folder
- `files/` : Private documents and font files root (this folder must be writable for PHP)
- `samples/` : This folder contains useful configuration and example files for Apache or Nginx webservers
- `web/` : Your website root, it contains your application entry-points and your public assets
  - `files/` : Public documents (this folder must be writable for PHP)
  - `themes/` : public assets mirror for each theme, this folder contains symlinks to your `themes/YourTheme/static` folder
- `src/` : Contains all your website logic
- `vendor/` : Dependencies folder managed by *Composer*

Configure CORS

```
# src/AppServiceProvider.php
/**
 * @return array
 */
$container['api.cors_options'] = [
    'allow_credentials' => true,
    // Allow all origin or defines some regex domains
    'allow_origin' => ['*'],
    'allow_headers' => true,
    'origin_regex' => false,
    'allow_methods' => ['GET'],
    // Expose Link header for NuxtJS to resolve other translations
    'expose_headers' => ['Link'],
    'max_age' => 60*60*24
];
```

API usage and authentication

API endpoints are described in detail in AbstractApiTheme repository README. You'll get automatic :
- Collection listing for all nodes-sources
- Collection listing per node-type
- Single item endpoint
- User endpoint
- ...

All with query-string parameters for searching and filtering JSON output.

API can be accessed using :
- Simple `X-Api-Key` access per application for non-user related content
- Or **OAuth2** applications with `client_credentials` or `authorization_code` grant types.

3.1.6 Themes

Creating a theme

Roadiz themes are one of the main parts of the CMS. They allow you to create your really personal website. You can duplicate an existing theme to customize stylesheets and images. Or you can start from ground and build your very own theme using our API. Every visible part of Roadiz is a theme. Even backoffice interface is a theme, and it's called *Rozier* according to the street name where REZO ZERO created it.

Each theme is a folder which must be placed in `themes/` folder. Roadiz *Sources* comes with 3 default themes :
- *Install* : It's the first page theme you see when you launch Roadiz in your browser for the first time.
- *Rozier* : Here is the REZO ZERO designed backoffice for Roadiz, it's available from `rz-admin/` url and protected by firewall.
- *DefaultTheme* : It's a demo theme which is mainly used to demonstrate basic Roadiz features and to try the back-office editing capabilities.

As these 3 themes come bundled with Roadiz, you can't edit or update their files. Your changes would be overrode the next time you update Roadiz via Git or direct download. If you want to create your own Backoffice, you can. Just name it differently and hook it in backoffice or using CLI commands.

Source Edition

If you are using Roadiz *Source edition*, we configured *Git* versioning tool to ignore every additional theme you create in `/themes` folder. So you can **initialize your a new git repository per custom theme you create.** That way you can use code versioning independently from Roadiz updates.

Preparing your own frontend theme

To start from a fresh and clean foundation, we built a BaseTheme to fit our needs with many starter node-types and a front-end framework using *ES6* and *Webpack*.

```
# Use Roadiz command to pull and rename BaseTheme after your own project
bin/roadiz themes:generate --relative --symlink MyAwesome
```

Your theme will be generated as `/themes/MyAwesomeTheme` with `/themes/MyAwesomeTheme/MyAwesomeThemeApp.php` class.

Standard Edition

Roadiz *Standard edition* will create a symbolic link into `web/` folder to publish your new theme public assets as `/web/themes/MyAwesomeTheme/static`. Make sure that your system supports *symbolic links*.

Edit your main class information (`MyAwesomeThemeApp.php`)

```
namespace Themes\MyAwesomeTheme;

use RZ\Roadiz\CMS\Controllers\FrontendController;
use RZ\Roadiz\Core\Entities\Node;
use RZ\Roadiz\Core\Entities\Translation;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

/**
 * MyAwesomeThemeApp class
 */
class MyAwesomeThemeApp extends FrontendController
{
    protected static string $themeName =      'My awesome theme';
    protected static string $themeAuthor =    'Ambroise Maupate';
    protected static string $themeCopyright = 'REZO ZERO';
    protected static string $themeDir =       'MyAwesomeTheme';
    protected static bool $backendTheme =    false;

    //...
}
```

Then you will be able to add your fresh new theme into Roadiz backoffice or through Roadiz install.

Static routing

Before searching for a node's Url (Dynamic routing), Roadiz will parse your theme `route.yml` to find static controllers and actions to execute. Static actions just have to comply with the `Request` / `Response` scheme. It is advised to add `$_locale` and `$_route` optional arguments to better handle multilingual pages.

```
foo:
    path: /foo
    defaults:
        _controller: Themes\MyAwesomeTheme\Controllers\FooBarController::fooAction
bar:
    path: /{_locale}/bar
    defaults:
        _controller: Themes\MyAwesomeTheme\Controllers\FooBarController::barAction
    requirements:
        # Use every 2 letter codes
        _locale: "[a-z]{2}"
```

```
public function fooAction(Request $request)
{
    $translation = $this->bindLocaleFromRoute($request, 'en');
    $this->prepareThemeAssignation(null, $translation);

    return $this->render('foo.html.twig', $this->assignation);
}

public function barAction(
    Request $request,
    $_locale = null,
    $_route = null
) {
    $translation = $this->bindLocaleFromRoute($request, $_locale);
    $this->prepareThemeAssignation(null, $translation);

    return $this->render('bar.html.twig', $this->assignation);
```

```
}
```

Dynamic routing

Note : Every node-types will be handled by a specific `Controller`. If your created a "Page" type, Roadiz will search for a `...\Controllers\PageController` class and it will try to execute the `indexAction` method.

An indexAction method must comply with the following signature. It will take the HttpFoundation's Request as first then a `Node` and a `Translation` instances. These two last arguments will be useful to generate your page information and to render your current node.

```
/**
 * Default action for any Page node.
 *
 * @param Symfony\Component\HttpFoundation\Request $request
 * @param RZ\Roadiz\Core\Entities\Node              $node
 * @param RZ\Roadiz\Core\Entities\Translation       $translation
 *
 * @return Symfony\Component\HttpFoundation\Response
 */
public function indexAction(
    Request $request,
    Node $node = null,
    TranslationInterface $translation = null
) {
    $this->prepareThemeAssignation($node, $translation);

    return $this->render(
        'types/page.html.twig',  // Twig template path
        $this->assignation       // Assignation array to fill template placeholders
    );
}
```

As *Symfony* controllers do, every Roadiz controllers actions have to return a valid `Response` object. This is the render method purpose which will generate a standard *html* response using a *Twig* template and an assignation array.

Note : It's very easy to create *JSON* responses for your API with Roadiz. You just have to replace `$this->render($template, $assignation);` method with `$this->renderJson($data);`. This method is a shortcut for `new JsonResponse($data);`.

Home page case

Homepage is always a special page to handle. With Roadiz you have the choice to handle it as a static page or as a dynamic page. In both case you'll need to setup a static route in your theme `Resources/routes.yml` file.

```
homePage:
    path: /
    defaults:
        _controller: Themes\MyAwesomeTheme\MyAwesomeThemeApp::homeAction
homePageLocale:
    path: /{_locale}
    defaults:
        _controller: Themes\MyAwesomeTheme\MyAwesomeThemeApp::homeAction
```

```
    requirements:
        # Use every 2 letter codes
        _locale: "[a-z]{2}"
```

Now you can code your `homeAction` method in `MyAwesomeThemeApp` class. It will need 2 arguments :
— A `Request` object : `$request`
— An optional locale string variable `$_locale = null`

Dynamic home

If your home page is built with a node. You can tell Roadiz to handle home request as a *Page* request (if your home is a *page* type node) using `$this->handle($request);` method. This method will use the `PageController` class and `page.html.twig` template to render your home. This can be useful when you need to switch your home page to an other page, there is no need to make special ajustments.

```
/**
 * {@inheritdoc}
 */
public function homeAction(
    Request $request,
    $_locale = null
) {
    /*
     * Get language from static route
     */
    $translation = $this->bindLocaleFromRoute($request, $_locale);
    $home = $this->getHome($translation);

    /*
     * Render Homepage according to its node-type controller
     */
    return $this->handle($request, $home, $translation);
}
```

Static home

Imagine now that your home page has a totally different look than other pages. Instead of letting `handle()` method returning your Response object, you can create it directly and use a dedicated `home.html.twig` template. The fourth argument *static : :getThemeDir()* is optional, it explicits the namespace to look into. It becames useful when you mix several themes with the same templates names.

```
/**
 * {@inheritdoc}
 */
public function homeAction(
    Request $request,
    $_locale = null
) {
    /*
     * Get language from static route
     */
    $translation = $this->bindLocaleFromRoute($request, $_locale);
    $home = $this->getHome($translation);

    /*
     * Render Homepage manually
     */
    $this->prepareThemeAssignation($home, $translation);
```

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

Keep in ming that `prepareThemeAssignation` method will assign for you some useful variables no matter you choice a dynamic or a static home handling :

- `node`
- `nodeSource`
- `translation`

Using Twig

Note : Twig is the default rendering engine for *Roadiz* CMS. You'll find its documentation at http://twig.sensiolabs.org/doc/templates.html

---

When you use *Dynamic routing* within your theme, Roadiz will automatically assign some variables for you.

- **cms_version** — [string]
- **cms_prefix** — [string]
- **help_external_url** — [string] Back-office help URL (this can be overriden in your theme if your wrote a dedicated documentation for your customers)
- **is_debug** - [boolean]
- **is_preview** - [boolean]
- **is_dev_mode** - [boolean]
- **is_prod_mode** - [boolean]
- **head**
  - **devMode** — [boolean]
  - **universalAnalyticsId** — [string]
  - **useCdn** - [boolean]
  - **baseUrl** — [string] Server base Url. Basically your domain name, port and folder if you didn't setup Roadiz at you server root
  - **filesUrl** — [string]
- **bags**
  - **settings** — [SettingsBag]
  - **nodeTypes** — [NodeTypesBag]
  - **roles** — [RolesBag]
- **app**
  - **session** — [Session]
  - **user** — [User]
  - **request** — [object] Symfony request object which contains useful data such as current URI or GET parameters

There are some more content only available from *FrontendControllers*.

- **_default_locale** — [string]
- **meta**
  - **siteName** — [string]
  - **siteCopyright** — [string]
  - **siteDescription** — [string]

Then, in each dynamic routing *actions* you will need this line `$this->storeNodeAndTranslation($node, $translation);` in order to make page content available from your Twig template.

- **node** — [object]
- **nodeSource** — [object]
- **translation** — [object]
- **pageMeta**
  - **title** — [string]
  - **description** — [string]
  - **keywords** — [string]

All these data will be available in your Twig template using `{{ }}` syntax. For example use `{{ pageMeta.title }}` inside your head's `<title>` tag. You can of course call objects members within Twig using the *dot* separator.

```
<article>
    <h1><a href="{{ path(nodeSource) }}">{{ nodeSource.title }}</a></h1>
    <div>{{ nodeSource.content|markdown }}</div>

    {% set images = nodeSource.images %}

    {% for image in images %}
        {% set imageMetas = image.documentTranslations.first %}
        <figure>
            {{ image|display({'width':200 }) }}
            <figcaption>{{ imageMetas.name }} -- {{ imageMetas.copyright }}</
↪figcaption>
        </figure>
    {% endfor %}
</article>
```

Checking role permissions

You can use standard Twig `is_granted` method to check if current authenticated user can access resources according to Roadiz *ROLES*.

```
{% if is_granted('ROLE_ACCESS_TAGS') %}
<div class="tags">
    User can access tags
</div>
{% endif %}
```

Checking node-sources type

You can use every node-type names as simple *Twig test* against your `nodeSource` variables.

```
{% set parentNodeSource = nodeSource|parent %}

{% if parentNodeSource is Page %}
    <p>I'm in a page</p>
{% endif %}
{% if parentNodeSource is BlogPost %}
    <p>I'm in a blog article</p>
{% endif %}
```

You can use real node-type names, like `Page`, or their PHP classname, like `NSPage`.

Generating paths and url

Standard Twig `path` and `url` methods are both working for *static* and *dynamic* routing. In Roadiz, these methods can take either a `string` identifier or a `NodesSources` instance. Of course optional parameters are available for both, they will automatically create an *http query string* when using a node-source.

```
{# Path generation with a Symfony route  #}
{# Eg. /fr  #}
{{ path('homePageLocale', {_locale: 'fr'}) }}

{# Path generation with a node-source  #}
{# Eg. /en/about-us  #}
```

```
{{ path(nodeSource) }}

{# Url generation with a node-source  #}
{# Eg. http://localhost:8080/en/about-us  #}
{{ url(nodeSource) }}

{# Path generation with a node-source and parameters  #}
{# Eg. /en/about-us?page=2  #}
{{ path(nodeSource, {'page': 2}) }}
```

Generating assets paths and url

You also can use standard `{{ asset('file.jpg', 'FooBarTheme') }}` to generate a path to a public asset stored in your theme. This will generate `/themes/FooBarTheme/static/file.jpg` path, **do not** add a leading slash to let Symfony package generate path according to request context.

If you need this path to converted to absolute url, use `{{ absolute_url(asset('file.jpg', 'FooBarTheme')) }}`.

`asset` method second argument is the *package* to use for resolving assets. Roadiz automatically registers a package using your theme name.

Note : Make sure you are **not** using a leading slash in your asset paths. If you begin path with a slash, the assets package won't resolve it and it will assume that your path is already absolute.

Handling node-sources with Twig

Most of your front-end work will consist in editing *Twig* templating, *Twig* assignations and... *Twig* filters. Roadiz core entities are already linked together so you don't have to prepare your data before rendering it. Basically, you can access *nodes* or *node-sources* data directly in *Twig* using the "dot" separator.

There is even some magic about *Twig* when accessing private or protected fields : just write the field-name and it will use the getter method instead : `{{ nodeSource.content|markdown }}` will be interpreted as `{{ nodeSource.getContent|markdown }}` by *Twig*.

Note : Roadiz will transform your node-type fields names to *camel-case* to create getters and setters into you NS class. So if you created a `header_image` field, getter will be named `getHeaderImage()`. However, if you called it `headerimage`, getter will be `getHeaderimage()`

You can access methods too ! You will certainly need to get node-sources' documents to display them. Instead of declaring each document in your PHP controller before, you can directly use them in *Twig* :

```
{% set images = nodeSource.images %}
{% for image in images %}
    {% set imageMetas = image.documentTranslations.first %}
    <figure>
        {{ image|display({ 'width':200 }) }}
        <figcaption>{{ imageMetas.name }} -- {{ imageMetas.copyright }}</
↪figcaption>
    </figure>
{% endfor %}
```

Use node references

If you added a *node reference* field to your node-source type you will able to grab them directly from the proxy methods. For example, add a `artist_references` field which links some `Artist` nodes to your `Page` node-type. Then you will be able to grab them using :

```
{% set artists = nodeSource.artistReferencesSources %}
{% for artist in artists %}
    <a href="{{ path(artist) }}">{{ artist.title }}</a>
{% endfor %}
```

Note the `Sources` suffix after field getter name. It allows you to directly fetch `NodesSources` objects instead of `Node`.

We encourage you to set only one Node-Type to your node-reference field to optimize *Doctrine* queries.

Loop over node-source children

With Roadiz you will be able to grab each node-source children using custom `children` Twig filter.

```
{% set childrenBlocks = nodeSource|children %}
{% for childBlock in childrenBlocks %}
<div class="block">
    <h2>{{ childBlock.title }}</h2>
    <div>{{ childBlock.content|markdown }}</div>
</div>
{% endfor %}
```

```
{#
 # This statement will only grab *visible* "Page" children node-sources and
 # will order them ascendant according to their *title*.
 #}
{% set childrenBlocks = nodeSource|children({
        'node.visible': true,
        'node.nodeType': bags.nodeTypes.get('Page'),
    },
    {'title': 'ASC'}
) %}
```

Note : Calling `getChildren()` from a node-source *handler* or `|children` filter will **always** return `NodesSources` objects from the same translation as their parent.

Add previous and next links

In this example, we want to create links to jump to *next* and *previous* pages. We will use node-source handler methods `getPrevious()` and `getNext()` which work the same as `getChildren()` method. `|previous` and `|next` Twig filters are also available.

```
{% set prev = nodeSource|previous %}
{% set next = nodeSource|next %}

{% if (prev or next) %}
<nav class="contextual-menu">
    {% if prev %}
    <a class="previous" href="{{ path(prev) }}"><i class="uk-icon-arrow-left"></i>
↪{{ prev.title }}</a>
    {% endif %}
```

```
    {% if next %}
    <a class="next" href="{{ path(next) }}">{{ next.title }} <i class="uk-icon-
↪arrow-right"></i></a>
    {% endif %}
</nav>
{% endif %}
```

Note : Calling `getPrevious` and `getNext` from a node-source *handler* will **always** return `NodesSources` objects from the same translation as their sibling.

Additional filters

Roadiz's Twig environment implements some useful filters, such as :

- `markdown` : Convert a markdown text to HTML
- `inlineMarkdown` : Convert a markdown text to HTML without parsing *block* elements (useful for just italics and bolds)
- `markdownExtra` : Convert a markdown-extra text to HTML (footnotes, simpler tables, abbreviations)
- `centralTruncate(length, offset, ellipsis)` : Generate an ellipsis at the middle of your text (useful for filenames). You can decenter the ellipsis position using `offset` parameter, and even change your ellipsis character with `ellipsis` parameter.
- `handler` : Get an entity handler using Roadiz *HandlerFactory*.

NodesSources filters

These following Twig filters will only work with `NodesSources` entities... not `Nodes`. Use them with the *pipe* syntax, eg. `nodeSource|next`.

- `children` : shortcut for `NodesSourcesHandler::getChildren()`
- `next` : shortcut for `NodesSourcesHandler::getNext()`
- `previous` : shortcut for `NodesSourcesHandler::getPrevious()`
- `firstSibling` : shortcut for `NodesSourcesHandler::getFirstSibling()`
- `lastSibling` : shortcut for `NodesSourcesHandler::getLastSibling()`
- `parent` : shortcut for `$source->getParent()`
- `parents` : shortcut for `NodesSourcesHandler::getParents(array $options)`
- `tags` : shortcut for `NodesSourcesHandler::getTags()`
- `render(themeName)` : initiate a sub-request for rendering a given block *NodesSources*

Documents filters

These following Twig filters will only work with `Document` entities. Use them with the *pipe* syntax, eg. `document|display`.

- `url` : returns document public URL as *string*. See *document URL options*.
- `display` : generates an HTML tag to display your document. See *document display options*.
- `imageRatio` : return image size ratio as *float*.
- `imageSize` : returns image size as *array* with `width` and `height`.
- `imageOrientation` : get image orientation as *string*, returns `landscape` or `portrait`.
- `path` : shortcut for document real path on server.
- `exists` : shortcut to test if document file exists on server. Returns `boolean`.
- `embedFinder` : return the embed finder to handle external documents sources such as *Youtube* or *Vimeo*

Translations filters

These following Twig filters will only work with `Translation` entities. Use them with the *pipe* syntax, eg. `translation|menu`.

— `menu` : shortcut for `TranslationViewer::getTranslationMenuAssignation()`.
This filter returns some useful informations about current page available languages and their urls. See getTranslationMenuAssignation method definition. You do not have to pass it the current request object as the filter will grab it for you. But you can specify if you want *absolute* urls or not.

Standard filters and extensions are also available :

— `{{ path('myRoute') }}` : for generating static routes Url.
— `u.truncate` and `u.wordwrap` which are parts of the UnicodeString instance .
— `format_date`, `format_datetime`, `format_currency`, `format_number`, `country_name` which are parts of the IntlExtension extension .

Create your own Twig filters

Imagine now that your are rendering some dynamic CSS stylesheets with Twig. Your are listing your website projects which all have a distinct color. So you've created a CSS route and a `dynamic-colors.css.twig`.

```
{% for project in projects %}
.{{ project.node.nodeName }} h1 {
    color: {{ project.color }};
}
{% endfor %}
```

This code should output a CSS like that :

```
.my-super-project h1 {
    color: #FF0000;
}
.my-second-project h1 {
    color: #00FF00;
}
```

Then you should see your "super project" title in red on your website. OK, that's great. But what should I do if I need to use a RGBA color to control the Alpha channel value ? For example, I want to set project color to a `<div class="date">` background like this :

```
.my-super-project .date {
    background-color: rgba(255, 0, 0, 0.5);
}
.my-second-project .date {
    background-color: rgba(0, 255, 0, 0.5);
}
```

Great... I already see coming guys complaining that "rgba" is only supported since IE9... We don't give a shit!...

Hum, hum. So you need a super filter to extract decimal values from our backoffice stored hexadecimal color. Roadiz enables us to extend Twig environment filters thanks to *dependency injection!*

You just have to extend `setupDependencyInjection` static method in your main theme class. Create it if it does not exist yet.

```
// In your SuperThemeApp.php
public static function setupDependencyInjection(\Pimple\Container $container)
{
    parent::setupDependencyInjection($container);

    // We extend twig filters
```

```
    $container->extend('twig.filters', function ($filters, $c) {

        // The first filter will extract red value
        $red = new \Twig_SimpleFilter('red', function ($hex) {
            if ($hex[0] == '#' && strlen($hex) == 7) {
                return hexdec(substr($hex, 1, 2));
            } else {
                return 0;
            }
        });
        $filters->add($red);

        // The second filter will extract green value
        $green = new \Twig_SimpleFilter('green', function ($hex) {
            if ($hex[0] == '#' && strlen($hex) == 7) {
                return hexdec(substr($hex, 3, 2));
            } else {
                return 0;
            }
        });
        $filters->add($green);

        // The third filter will extract blue value
        $blue = new \Twig_SimpleFilter('blue', function ($hex) {
            if ($hex[0] == '#' && strlen($hex) == 7) {
                return hexdec(substr($hex, 5, 2));
            } else {
                return 0;
            }
        });
        $filters->add($blue);

        // Then we return our extended filters collection
        return $filters;
    });
}
```

And... Voilà ! You can use `red`, `green` and `blue` filters in your Twig template.

```
{% for project in projects %}
.{{ project.node.nodeName }} .date {
    background-color: rgba({{ project.color|red }}, {{ project.color|green }}, {{␣
↪project.color|blue }}, 0.5);
}
{% endfor %}
```

Use custom Twig extensions

Just like you did to add your own *Twig* filters, you can add your own *Twig* extensions. Instead of extending `twig.filters` service, just extend `twig.extensions` service.

```
// In your SuperThemeApp.php
public static function setupDependencyInjection(\Pimple\Container $container)
{
    parent::setupDependencyInjection($container);

    // We extend twig extensions
    $container->extend('twig.extensions', function ($extensions, $c) {
        $extensions->add(new MySuperThemeTwigExtension());
        return $extensions;
```

```
    });
}
```

Creating a custom *Twig* extension is better if you want to add many new `filters`, `methods` and `globals` at the same time.

Displaying documents

Did you noticed that *images* relation is available directly in nodeSource object? That's a little shortcut to `(nodeSource|handler).documentFromFieldName('images')`. Cool, isn't it? When you create your *documents* field in your node-type, Roadiz generate a shortcut method for each document relation in your `GeneratedNodesSources/NSxxxx` class.

Now, you can use the `DocumentViewer` service to generate HTML view for your documents no matter they are *images*, *videos* or *embed*. Two *Twig* filters are available with `Documents` :

- `|display` generates an HTML tag to display your document.
- `|url` generates a public URL to reach your document.
- `|embedFinder` gets the EmbedFinder for current document according to the embed-platform type (Youtube, Vimeo, Soundcloud...).

```
{# Grab only first document from "images" field #}
{% set image = nodeSource.images[0] %}

{# Always test if document exists #}
{% if image %}
{{ image|display({
    'width':200,
    'crop':"1:1",
    'quality':75,
    'embed':true
}) }}
{% endif %}
```

HTML output options

- **absolute** (true|false, default : `false`), generates an *absolute* URL with protocol, domain-name and base-url. This must be used for social network images.
- **embed** (true|false, default : `false`), display an embed as iframe instead of its thumbnail
- **identifier**
- **class**
- **alt** : If not filled, it will get the document name, then the document filename
- **lazyload** (true|false, default : `false`), fill image src in a `data-src` attribute instead of `src` to prevent it from loading. It will add automatically `lazyload_class` class to your HTML image.
- **lazyload_class** (default : `lazyload`) Class name to be added when enabling lazyloading.
- **fallback** (URL|data-uri) Defines a custom fallback image URL or *data-uri* when using `lazyload` option in order to fill `src` attribute and validate against W3C
- **blurredFallback** (false|true, default : `false`) Generated a very low quality image version for lazyload fallback to better control image size and better experience.
- **picture** (false|true, default : `false`), use `<picture>` element instead of image and allow serving WebP image to compatibles browsers. **Only use if your server support WebP**.
- **inline** (true|false, default : `true`), **for SVG**, display SVG inline code in html instead of using an `<object>` tag. Default `true`.
- **loading** (auto|lazy|eager|null, default : `null`), for next-gen browser only that will support native lazy-loading. This will be applied only on *img*, *picture* and *iframe* elements. *This can fail W3C validation*.

Images resampling options

- **width**
- **height**
- **crop** (ratio : `{w}:{h}`, for example : `16:9`), defines a ratio for cropping images. **You must combine ‘‘crop‘‘ option with ‘‘width‘‘ or ‘‘height‘‘ option**, otherwise you'll end with a cyclic dependency to compute output image size. We recommend you to use `fit` option with *final image size* to better control output : `crop` may lose precision on width or height especially on small sizes.
- **fit** (fixed dimensions : `{w}x{h}`, for example : `100x200`), if you are using *fit* option, Roadiz will be able to add `width` and `height` attributes to your `<img>` tag.
- **align, to use along with `fit` parameter to choose which part of the picture to fit. Allowed options :**
  - top-left
  - top
  - top-right
  - left
  - center
  - right
  - bottom-left
  - bottom
  - bottom-right
- **grayscale** (boolean, default : `false`)
- **quality** (1-100, default : `90`)
- **flip** (`h` or `v`), mirror your image vertical or horizontal
- **blur** (1-100, default : `0`) *(can be really slow to process)*
- **sharpen** (1-100, default : `0`)
- **contrast** (1-100, default : `0`)
- **background** (hexadecimal color without #)
- **progressive** (boolean, default : `false`), it will interlace the image if it's a *PNG* file.
- **noProcess** (boolean, default : `false`) : Disable image processing, useful if you want to keep animated GIF

Audio / Video options

- **autoplay** (boolean, default : `false`)
- **controls** (boolean, default : `true`)
- **loop** (boolean, default : `false`)
- **muted** (boolean, default : `false`)
- **custom_poster** (string) : URL to a image to be used as video poster

For *Soundcloud* embeds
- **hide_related** (boolean, default : `false`)
- **show_comments** (boolean, default : `false`)
- **show_user** (boolean, default : `false`)
- **show_reposts** (boolean, default : `false`)
- **visual** (boolean, default : `false`)

For *Mixcloud* embeds
- **mini** (boolean, default : `false`)
- **light** (boolean, default : `true`)
- **hide_cover** (boolean, default : `true`)
- **hide_artwork** (boolean, default : `false`)

For *Vimeo* embeds
- **displayTitle** (boolean, default : `false`)
- **byline** (boolean, default : `false`)
- **portrait** (boolean, default : `false`)
- **color** (boolean)
- **api** (boolean, default : `true`)
- **automute** (boolean, default : `false`)

— **autopause** (boolean, default : `false`)

For *Youtube* embeds

— **modestbranding** (boolean, default : `true`)
— **rel** (boolean, default : `false`)
— **showinfo** (boolean, default : `false`)
— **start** (integer, default : `false`)
— **end** (integer, default : `false`)
— **enablejsapi** (boolean, default : `true`)
— **playlist** (boolean, default : `false`)
— **playsinline** (boolean, default : `false`) : Allow iframe to play inline on iOS

You can use **multiple source files** for one video document or audio document. Just upload a file using tge same filename name but with a different extension. Use this method to add a poster image to your video too. For example : for `my-video.mp4` file, upload `my-video.webm`, `my-video.ogg` and `my-video.jpeg` documents. *Roadiz* will automatically generate a `<video>` tag using all these files as *source* and *poster* attribute.

Using src-set attribute for responsive images

Roadiz can generate a `srcset` attribute to create a responsive image tag like the one you can find on these examples.

— **srcset** (Array) Define for each rule an Array of format. Specifications
— **media** (Array) Define one `srcset` for each media-query. You cannot use `media` without `picture` option.

```
{% set image = nodeSource.images[0] %}
{% if image %}
{{ image|display({
    'fit':'600x600',
    'quality':75,
    'srcset': [
        {
            'format': {
                'fit':'200x200',
                'quality':90
            },
            'rule': '780w',
        },
        {
            'format': {
                'fit':'600x600',
                'quality':75
            },
            'rule': '1200w',
        }
    ],
    'sizes': [
        '(max-width: 780px) 200px',
        '(max-width: 1200px) 600px',
    ],
}) }}
{% endif %}
```

This will output an `img` tag like the following one :

```
<img src="/assets/f600x600-q75/image.jpg"
     srcset="/assets/f600x600-q75/image.jpg 1200w, /assets/f200x200-q90/image.jpg␣
↪780w"
     sizes="(max-width: 780px) 200px, (max-width: 1200px) 600px"
     data-ratio="1"
     width="600" height="600"
     alt="A responsive image">
```

Generate <picture> elements

If you want to combine `srcset` for media queries **and** device ratio, use `picture` element with `media` option :

```
{% set image = nodeSource.images[0] %}
{% if image %}
{{ image|display({
    'fit':'640x400',
    'quality':75,
    'picture': true,
    'media': [
        {
            'srcset': [
                {
                    'format': {
                        'fit':'320x200',
                        'quality':90
                    },
                    'rule': '1x',
                },
                {
                    'format': {
                        'fit':'640x400',
                        'quality':75
                    },
                    'rule': '2x',
                }
            ],
            'rule': '(max-width: 767px)'
        },
        {
            'srcset': [
                {
                    'format': {
                        'fit':'800x600',
                        'quality':80
                    },
                    'rule': '1x',
                },
                {
                    'format': {
                        'fit':'1600x1200',
                        'quality':70
                    },
                    'rule': '2x',
                }
            ],
            'rule': '(min-width: 768px)'
        }
    ]
}) }}
{% endif %}
```

This will output a `picture` element supporting :

- *WebP* image format (*Roadiz* will automatically generate a `.webp` image if your PHP is compiled with *webp* support)
- *Media query* attributes
- *Device ratio* src-set rules
- A fallback `img` element for older browsers

```
<picture>
    <source media="(max-width: 767px)"
```

```
            srcset="/assets/f320x200-q90/folder/file.jpg.webp 1x, /assets/f640x400-
↪q75/folder/file.jpg.webp 2x"
            type="image/webp">
    <source media="(max-width: 767px)"
            srcset="/assets/f320x200-q90/folder/file.jpg 1x, /assets/f640x400-q75/
↪folder/file.jpg 2x"
            type="image/jpeg">

    <source media="(min-width: 768px)"
            srcset="/assets/f800x600-q80/folder/file.jpg.webp 1x, /assets/
↪f1600x1200-q70/folder/file.jpg.webp 2x"
            type="image/webp">
    <source media="(min-width: 768px)"
            srcset="/assets/f800x600-q80/folder/file.jpg 1x, /assets/f1600x1200-
↪q70/folder/file.jpg 2x"
            type="image/jpeg">

    <img alt="file.jpg"
         src="/assets/f640x400-q75/folder/file.jpg"
         data-ratio="1.6"
         width="640" height="400" />
</picture>
```

More document details

You can find more details in our API documentation.

— If document is an **image** : `getDocumentByArray` method will generate an `<img />` tag with a `src` and `alt` attributes.
— If it's a **video**, it will generate a `<video />` tag with as many sources as available in your document database. Roadiz will look for same filename with each HTML5 video extensions (filename.mp4, filename.ogv, filename.webm).
— Then if document is an external media **and** if you set the `embed` flag to `true`, it will generate an iframe according to its platform implementation (*Youtube*, *Vimeo*, *Soundcloud*).
— Get the external document URI (the one used for creating iframe for example) with `(document|embedFinder).source(options...)` twig command.

Displaying document metas

Documents can have *name*, *description* and *copyright* (which can be translated), just access them using `documentTranslations` multiple relation (`documentTranslations.first` should always contain current context' translation) :

```
{% for document in nodeSource.documents %}
    <div class="document-item">
        {{ document|display }}

        {% set metas = document.documentTranslations.first %}
        <h3 class="document-item-name">{{ metas.name }}</h3>
        <div class="document-item-description">{{ metas.description|markdown }}</
↪div>
        <em class="document-item-copyright">{{ metas.copyright }}</em>
    </div>
{% endfor %}
```

Displaying document thumbnails

Embed and non-HTML documents will not display automatically their thumbnails, even if they got one. Native videos and audios will always try to display `<video>` or `<audio>` elements, so if you need to force display their thumbnail image you'll need to write it manually :

```
{% for document in nodeSource.documents %}
    <div class="document-item">
        {% if document.hasThumbnails %}
            {{ document.thumbnails[0]|display }}
        {% else %}
            {{ document|display({
                'controls': true,
                'autoplay': false
            }) }}
        {% endif %}
    </div>
{% endfor %}
```

Non-viewable document types, such as *PDF*, *Word*, *Excel*, *Archives*..., will always use their thumbnail image, if there is one, when you call `{{ document|display }}`.

Manage global documents

You can store documents inside *settings* for global images such as header images or website logo. Simply create a new *setting* in Roadiz back-office choosing *Document* type, then a file selector will appear in settings list to upload your picture.

To use this document setting in your theme, you can assign it globally in your `MyThemeApp::extendAssignation` method. Use `getDocument` method instead of `get` to fetch a `Document` object that you'll be able to display in your Twig templates :

```
$document = $this->get('settingsBag')->getDocument('site_logo');
```

Or in a Twig template :

```
<figure id="site-logo">{{ bags.settings.getDocument('site_logo')|display }}</
↪figure>
```

This way is the easiest to fetch a global document, but it needs you to upload it once in *Settings* section. If this does not suit you, you can always fetch a *Document* manually using its *Doctrine* repository and a hard-coded `filename`.

```
$this->assignation['head']['site_logo'] = $this->get('em')->
↪getRepository(Document::class)->findOneByFilename('logo.svg');
```

Extending your Twig assignation

For a simple website theme, base assignation will work for almost every cases. Using `node` or `nodeSource` data from your Twig template, you will be able to render all your page fields.

Now imagine you need to load data from another node than the one being requested. Or imagine that you want to create a complex homepage which displays a summary of your latest news. You will need to extend existing assignated variables.

For example, create a simple node-type called *Page*. Add several basic fields inside it such as *content* and *images*. If you well-understood *how to create a theme* section you will create a *PageController.php* which look like this :

```php
<?php
namespace Themes\MyTheme\Controllers;

use Themes\MyTheme\MyThemeApp;
use RZ\Roadiz\Core\Entities\Node;
use RZ\Roadiz\Core\AbstractEntities\TranslationInterface;
use Symfony\Component\HttpFoundation\Request;

/**
 * Frontend controller to handle Page node-type request.
 */
class PageController extends MyThemeApp
{
    /**
     * Default action for any Page node.
     *
     * @param Request $request
     * @param Node|null $node
     * @param TranslationInterface $translation
     *
     * @return Symfony\Component\HttpFoundation\Response
     */
    public function indexAction(
        Request $request,
        Node $node = null,
        TranslationInterface $translation = null
    ) {
        $this->prepareThemeAssignation($node, $translation);

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

You will be able to render your page using `themes/MyTheme/Resources/views/types/page.html.twig` template file :

```twig
{% extends '@MyTheme/base.html.twig' %}

{% block content %}

<h1>{{ nodeSource.title }}</h1>
<div class="content">{{ nodeSource.content|markdown }}</div>
<div class="images">
    {% for image in nodeSource.images %}
        <figure>
            {{ image|display }}
        </figure>
    {% endfor %}
</div>
{% endblock %}
```

Use theme-wide assignation

Custom assignations are great but what can I do if I have to use the same variables in several controllers ? We added a special `extendAssignation` method which is called at the end of your theme preparation process (`prepareThemeAssignation` and `prepareNodeSourceAssignation`). Just override it in your `MyThemeApp` main class, then every theme controllers and templates will be able to use these variables.

For example, you can use this method to make `<head>` variables available for each of your website pages.

```php
/**
 * {@inheritdoc}
 */
protected function extendAssignation()
{
    parent::extendAssignation();

    $this->assignation['head']['facebookUrl'] = $this->get('settingsBag')->get(
↪'facebook_url');
    $this->assignation['head']['facebookClientId'] = $this->get('settingsBag')->
↪get('facebook_client_id');
    $this->assignation['head']['instagramUrl'] = $this->get('settingsBag')->get(
↪'instagram_url');
    $this->assignation['head']['twitterUrl'] = $this->get('settingsBag')->get(
↪'twitter_url');
    $this->assignation['head']['googleplusUrl'] = $this->get('settingsBag')->get(
↪'googleplus_url');
    $this->assignation['head']['googleClientId'] = $this->get('settingsBag')->get(
↪'google_client_id');
    $this->assignation['head']['maps_style'] = $this->get('settingsBag')->get(
↪'maps_style');
    $this->assignation['head']['themeName'] = static::$themeName;
    $this->assignation['head']['themeVersion'] = static::VERSION;
}
```

Use *Page / Block* data pattern

At REZO ZERO, we often use complex page design which need removable and movable parts. At first we used to create long node-types with a lot of fields, and when editors needed to move content to an other position, they had to cut and paste text to another field. It was long and not very sexy.

So we thought about a modular way to build pages. We decided to use one master node-type and several slave node-types instead of a single big type. Here is what we call **Page/Block pattern**.

This pattern takes advantage of Roadiz node hierarchy. We create a very light *Page* node-type, with an *excerpt* and a *thumbnail* fields, then we create an other node-type that we will call *BasicBlock*. This block node-type will have a *content* and *image* fields.

The magic comes when we add a last field into *Page* master node-type called *children_nodes*. This special field will display a node-tree inside your edit page. In this field parameter, we add *BasicBlock* name as a default value to tell Roadiz that each *Page* nodes will be able to contain *BasicBlock* nodes.

So you understood that all your page data will be allocated in several *BasicBlock* nodes. Then your editor will just have to change block order to re-arrange your page content. That's not all ! With this pattern you can join images to each block so that each paragraph can be pictured with a *Document* field. No need to insert image tags right into your Markdown text as you would do in a Wordpress article.

How to template *Page / Block* pattern

Now that you've structured your data with a *Page* node-type and a *BasicBlock*, how do render your data in only one page and only one URL request ? We will use custom assignations !

You can directly assign your children blocks at the beginning of your *Twig* template. Make sure the global `bags` service is available and reachable.

```twig
{# Fetch only BasicBlock nodes inside #}
{% set blocks = nodeSource|children({
    node.nodeType : bags.nodeTypes.get('BasicBlock'),
}) %}
```

```
{# Fetch ALL non-reachable nodes inside #}
{% set blocks = nodeSource|children({
    node.nodeType.reachable : false,
}) %}
```

Note : You can use different *block* types in the same *page*. Just create as many node-types as you need and add their name to your *Page* `children_node` default values. Then add each node-type into `children` criteria using an array instead of a single value : `node.nodeType : [bags.nodeTypes.get('BasicBlock'), bags.nodeTypes.get('AnotherBlock')]`. That way, you will be able to create awesome pages with different looks but with the same template (basic blocks, gallery blocks, etc).

Now we can update your `types/page.html.twig` template to use your assignated blocks.

```
{% if blocks %}
<section class="page-blocks">
{% for pageBlock in blocks %}
    {% include '@MyTheme/blocks/' ~ pageBlock.nodeTypeName|u.snake ~ '.html.twig'␣
↪with {
        'nodeSource': pageBlock,
        'parentNodeSource': nodeSource,
        'themeServices': themeServices,
        'bags': bags,
        'head': head,
        'node': pageBlock.node,
        'nodeType': pageBlock.node.nodeType,
        'loop': loop,
        'blocksLength':blocks|length
    } only %}
{% endfor %}
</section>
{% endif %}
```

Whaaat ? What is that include ? This trick will save you a lot of time ! We ask Twig to include a sub-template according to each block type name. Eg. for a *BasicBlock* node, Twig will include a `blocks/basicblock.html.twig` file. It's even more powerful when you are using multiple block types because Twig will automatically choose the right template to render each part of your page.

Then create each of your blocks templates files in `blocks` folder :

```
{# This is file: blocks/basicblock.html.twig #}

<div class="basicblock {% if loop.index0 is even %}even{% else %}odd{% endif %}">
    {#
     # Did you notice that 'pageBlock' became 'nodeSource' as
     # we passed it during include for a better compatibility
     #}
    <h3>{{ nodeSource.title }}</h3>
    <div class="content">{{ nodeSource.content|markdown }}</div>

    <div class="images">
    {% for image in nodeSource.images %}
        <figure>
            {{ image|display({'width':200}) }}
        </figure>
    {% endfor %}
    </div>
</div>
```

Voilà ! This is the simplest example to demonstrate you the power of *Page / Block* pattern. If you managed to reproduce this example you can now try it using multiple *block* node-types, combining multiple sub-templates.

Use a TreeWalker to control your node hierarchy

Page/Block pattern is really powerful and is the foundation for almost every Rezo Zero websites. But this approach can lead to performance issues if developers do not specify each available node-types for each child. Thus, we wanted to remove this ORM logic from your Twig templates, in order to comply with MVC pattern, but more important, in order to expose node hierarchy into a REST JSON API.

Rezo Zero developed a third-party library : `rezozero/tree-walker` which aims to abstract node hierarchy from the context and the CMS where it is used.

```
composer require rezozero/tree-walker
```

A `TreeWalker` is a traversable object you will be able to loop on in your Twig template, but also to serialize into a JSON object. This TreeWalker object can be configured with *definitions* in order to fetch next-level objects from your *database*, your CMS, or even an external API. That way you instantiate a new TreeWalker with a *root* object and by simply traversing it, it will trigger a fetch operation (`getChildren`) which will look for the right definition for the *root* object `class`. Then « tree walking » operation goes on for each of your *root* object children until your definitions list is empty or when you reached the max-level limit.

Here is an example of what the *Page/Block* pattern looks like using a *block tree-walker* :

```
{% if blockWalker %}
    <div class="page-blocks">
        {% for subWalker in blockWalker %}
            {% include '@MyTheme/blocks/' ~ subWalker.item.nodeTypeName|u.snake ~
↪'.html.twig' ignore missing with {
                'nodeSource': subWalker.item,
                'parentNodeSource': nodeSource,
                'themeServices': themeServices,
                'head': head,
                'node': subWalker.item.node,
                'nodeType': subWalker.item.node.nodeType,
                'loop': loop,
                'blockWalker': subWalker,
                'blocksLength': blockWalker|length
            } only %}
        {% endfor %}
    </div>
{% endif %}
```

Frontend developers do not need to know how to fetch children blocks anymore, they just need to loop over the *tree-walker* at each template level.

Use block rendering

A few times, using *Page / Block* pattern won't be enough to display your page blocks. For example, you will occasionally need to create a form inside a block, or you will need to process some data before using them in your Twig template.

For this we added a `render` filter which basically create a sub-request to render your block. This new request make possible to create a dedicated `Controller` for your block.

Let's take the previous example about a page with several *basic blocks* inside. Imagine you have a new *contact block* to insert in your page, then how would you create your form ? The following code shows how to "embed" a sub-request inside your block template.

```
{#
 # This is file: blocks/contactblock.html.twig
 #}
<div class="contactblock {% if loop.index0 is even %}even{% else %}odd{% endif %}">
```

```
    <h3>{{ nodeSource.title }}</h3>
    <div class="content">{{ nodeSource.content|markdown }}</div>

    {#
     # We created a display_form node-type field to enable/disable form
     # but this is optional
     #}
    {% if nodeSource.displayForm %}
        {#
         # "render" twig filter initiate a new Roadiz request
         # using *nodeSource* as primary content. It takes one
         # argument to locate your block controller
         #}
        {{ nodeSource|render('MyTheme') }}
    {% endif %}
</div>
```

Then Roadiz will look for a `Themes\MyTheme\Controllers\Blocks\ContactBlockController.php` file and a `blockAction` method inside.

```
namespace Themes\MyTheme\Controllers\Blocks;

use RZ\Roadiz\Core\Entities\NodesSources;
use RZ\Roadiz\Core\Exceptions\ForceResponseException;
use Symfony\Component\HttpFoundation\Request;
use Themes\MyTheme\MyThemeApp;

class ContactBlockController extends MyThemeApp
{
    public function blockAction(Request $request, NodesSources $source,
↪$assignation)
    {
        $this->prepareNodeSourceAssignation($source, $source->getTranslation());

        $this->assignation = array_merge($this->assignation, $assignation);

        // If you assign session messages here, do not assign it in your
        // MyThemeApp::extendAssignation() method before.
        $this->assignation['session']['messages'] = $this->get('session')->
↪getFlashBag()->all();

        /*
         * Add your form code here, for example
         */
        $form = $this->createFormBuilder()->add('name', 'text')
                                          ->add('send_name', 'submit')
                                          ->getForm();
        $form->handleRequest($request);
        if ($form->isSubmitted() && $form->isValid()) {
            // some stuff
            throw new ForceResponseException($this->redirect($request->getUri()));
        }

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

        return $this->render('form-blocks/contactblock.html.twig', $this->
↪assignation);
    }
}
```

Then create your template `form-blocks/contactblock.html.twig`:

```
<div class="contact-form">
    {% for messages in session.messages %}
        {% for message in messages %}
            <p class="alert alert-success">{{ message }}</p>
        {% endfor %}
    {% endfor %}

    {{ form(contactForm) }}
</div>
```

Use controller rendering

Roadiz implements the standard *Symfony* fragment rendering too. Use `render()` Twig function with `controller()` function to initiate a Roadiz sub-request and embed complex contents into your templates.

```
{# views/base.html.twig #}

{# ... #}
<div id="sidebar">
    {{ render(controller(
        'Themes\\MyTheme\\Controllers\\ArticleController::recentArticlesAction',
        { 'max': 3 }
    )) }}
</div>
```

Then use regular Roadiz controllers and actions to handle your sub-request :

```
// themes/MyTheme/Controllers/ArticleController.php
namespace Themes\MyTheme\Controllers;

// ...

class ArticleController extends MyThemeApp
{
    public function recentArticlesAction(Request $request, int $max = 3, string $_
↪locale = 'en')
    {
        $translation = $this->bindLocaleFromRoute($request, $_locale);
        $this->prepareThemeAssignation(null, $translation);

        // make a database call or other logic
        // to get the "$max" most recent articles
        $articles = ...;

        return $this->render(
            'article/recent_list.html.twig',
            ['articles' => $articles]
        );
    }
}
```

See https://symfony.com/doc/current/templating/embedding_controllers.html for more details about *Symfony* render extension.

Paginate entities using EntityListManager

Roadiz implements a powerful tool to display lists and paginate them. Each `Controller` class allows developer to use `createEntityListManager` method.

In `FrontendController` inheriting classes, such as your theme ones, this method is overridden to automatically use the current `authorizationChecker` to filter entities by status when entities are *nodes*.

`createEntityListManager` method takes 3 arguments :

— **Entity classname**, i.e. `RZ\Roadiz\Core\Entities\Nodes` or `GeneratedNodeSources\NSArticle`. The great thing is that you can use it on a precise `NodesSources` class instead of using *Nodes* or *NodesSources* then filtering on *node-type*. Using a `NS` entity allows you to filter on your own custom fields too.
— **Criteria array**, (optional)
— **Ordering array**, (optional)

EntityListManager will automatically grab the current page looking for your Request parameters. If `?page=2` is set or `?search=foo`, it will use them to filter your list and choose the right page.

If you want to handle pagination manually, you always can set it with `setPage(page)` method, which must be called **after** handling *EntityListManager*. It is useful to bind page parameter in your *routing* configuration.

```
projectPage:
    path: /articles/{page}
    defaults:
        _controller:␣
↪Themes\MyAwesomeTheme\Controllers\ArticleController::listAction
        page: 1
    requirements:
        page: "[0-9]+"
```

Then, build your `listAction` method.

```
public function listAction(
    Request $request,
    $page,
    $_locale = 'en'
) {
    $translation = $this->bindLocaleFromRoute($request, $_locale);
    $this->prepareThemeAssignation(null, $translation);

    $listManager = $this->createEntityListManager(
        NSArticle::class,
        ['sticky' => false], //sticky is a custom field from Article node-type
        ['node.createdAt' => 'DESC']
    );
    /*
     * First, set item per page
     */
    $listManager->setItemPerPage(20);
    /*
     * Second, handle the manager
     */
    $listManager->handle();
    /*
     * Third, set current page manually
     * AFTER handling entityListManager
     */
    if ($page > 1) {
        $listManager->setPage($page);
    }

    $this->assignation['articles'] = $listManager->getEntities();
    $this->assignation['filters'] = $listManager->getAssignation();

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

Then create your `articles-feed.html.twig` template to display each entity paginated.

```
{# Listing #}
<ul class="article-list">
    {% for article in articles %}
        <li class="article-item">
            <a class="article-link" href="{{ path(article) }}">
                <h2>{{ article.title }}</h2>
            </a>
        </li>
    {% endfor %}
</ul>

{# Pagination #}
{% if filters.pageCount > 1 %}
    <nav class="pagination">
        {% if filters.currentPage > 1 %}
            <a class="prev-link" href="{{ path('projectPage', {page: filters.
↪currentPage - 1}) }}">
                {% trans %}prev.page{% endtrans %}
            </a>
        {% endif %}
        {% if filters.currentPage < filters.pageCount %}
            <a class="next-link" href="{{ path('projectPage', {page: filters.
↪currentPage + 1}) }}">
                {% trans %}next.page{% endtrans %}
            </a>
        {% endif %}
    </nav>
{% endif %}
```

Alter your Roadiz queries with events

The `FilterQueryBuilderEvent` can be used when *EntityListManager* criteria or *API* services won't offer enough parameters to select your entities. This event will be dispatched when just before *Doctrine* QueryBuilder will execute the DQL query so that you can add more DQL statements. This can be very powerful if you need, for example, to force an `INNER JOIN` or to use complexe DQL commands.

```
// Prepare a Closure listener to filter every NodesSources
// which are not called "About"
$callable = function(FilterQueryBuilderEvent $event) {
    // Specify the repository on which your filter will be applied
    // Try to be the more precise you can

    // This will be applied to all nodes-sources (greedy)
    if ($event->supports(NodesSources::class)) {
        $qb = $event->getQueryBuilder();
        $qb->andWhere($qb->expr()->neq($qb->expr()->lower('ns.title'), ':neq'));
        $qb->setParameter('neq', 'about');
    }
    // This will be applied only on your Page nodes-sources (safer)
    if ($event->supports(NSPage::class)) {
        $qb = $event->getQueryBuilder();
        $qb->andWhere($qb->expr()->neq($qb->expr()->lower('ns.title'), ':neq'));
        $qb->setParameter('neq', 'about');
    }
};

// Register your listener in Roadiz event dispatcher
/** @var EventDispatcher $eventDispatcher */
$eventDispatcher = $this->get('dispatcher');
$eventDispatcher->addListener(
```

```
    QueryBuilderEvents::QUERY_BUILDER_SELECT,
    $callable
);

// Do some queries or use Roadiz EntityListManager

// Do not forget to remove your listener not to alter EVERY
// queries on NodesSources in your following code.
$eventDispatcher->removeListener(
    QueryBuilderEvents::QUERY_BUILDER_SELECT,
    $callable
);
```

Avertissement : QueryBuilder events are a powerful tool to alter **all Roadiz entities pipeline**. Make sure to remove your listener from the dispatcher **before rendering your Twig templates** or to only support the `entityClass` you need. This could alter every queries such as `|children` Twig filters or your main navigation loop.

Add a firewall in your theme

You may need to add a secured area in your website or application, even for none-backend users. Roadiz uses *Symfony* security components to handle firewalled requests. You will be able to extend the *firewall map* in your Theme `addDefaultFirewallEntry` method.

Before create your firewall map entry, you must understand that Roadiz already has 2 firewall areas :

- `^/rz-admin` area, which naturally matches every back-office sections
- `^/` area which is required for previewing unpublished node and get user information across the whole website

The last firewall request matcher can be tricky to deal with, especially if you want to add another secured area as it *listen* to every requests. When you'll add new firewall map entry, you may call `parent::addDefaultFirewallEntry($container);` **before** your custom configuration to be sure that `^/` request matcher has **the lowest priority**. However, if you want to override `^/` request matcher configuration you have to omit the parent method call.

```
/**
 * {@inheritdoc}
 */
public static function addDefaultFirewallEntry(Container $container)
{
    /*
     * Your custom firewall map entry configuration
     * goes here
     */

    /*
     * Call parent ONLY if you don't want to create
     * a firewall map at website root level. And call it after
     * your own firewall entry.
     */
    parent::addDefaultFirewallEntry($container);
}
```

Configuring a non-root firewall map entry with FirewallEntry class

Before copy and pasting the following lines, think about it a little time… A firewall map entry defines several mandatory routes :

- A *base path* for your firewall to be triggered

— A *login* path, which **can** be **outside** or **inside** of your firewall map
— A *login_check* path, which **must** be **inside** of your firewall map
— A *logout* path, which **must** be **inside** of your firewall map
— A new role describing your secured area purpose (i.e. *ROLE_ACCESS_PRESS* for a private press kit area), you should create this role in Roadiz backoffice before.

If this example I will use :

— `/press` as my base path for secured area
— `/signin` for my login page, notice that it's not in my firewall
— `/press/login_check`
— `/press/logout`
— *ROLE_ACCESS_PRESS*

Here is the code to add in your theme' addDefaultFirewallEntry method. Do not forget to add the matching *use* statement in your file header.

```
use RZ\Roadiz\Utils\Security\FirewallEntry;
use Pimple\Container;

/**
 * {@inheritdoc}
 */
public static function addDefaultFirewallEntry(Container $container)
{
    /*
     * Call parent ONLY if you don't want to create
     * a firewall map at website root level.
     */
    parent::addDefaultFirewallEntry($container);

    $firewallBasePattern = '^/press';
    $firewallBasePath = '/press';
    $firewallLogin = '/signin';
    $firewallLogout = '/press/logout';
    $firewallLoginCheck = '/press/login_check';
    $firewallBaseRole = 'ROLE_ACCESS_PRESS';

    $firewallEntry = new FirewallEntry(
        $container,
        $firewallBasePattern,
        $firewallBasePath,
        $firewallLogin,
        $firewallLogout,
        $firewallLoginCheck,
        $firewallBaseRole
        // You can add a special AuthenticationSuccessHandler
        // if you need to do some stuff for your theme at visitor login
        //'Themes\YourTheme\Authentication\AuthenticationSuccessHandler'
    );
    // Allow anonymous authentication
    $firewallEntry->withAnonymousAuthenticationListener();
    // Allow switch user feature
    $firewallEntry->withSwitchUserListener();

    /*
     * Finally add this entry to the Roadiz
     * firewall map.
     */
    $container['firewallMap']->add(
        $firewallEntry->getRequestMatcher(),
        $firewallEntry->getListeners(),
        $firewallEntry->getExceptionListener()
    );
}
```

Add login routes

After configuring your Firewall, you'll need to add your routes to your theme `routes.yml` file. *Logout* and *login_check* won't need any controller setup as they will be handled directly by Roadiz firewall event dispatcher. The only one you need to handle is the *login* page.

```
themeLogout:
    path:     /press/logout
themeLoginCheck:
    path:     /press/login_check
themeLoginPage:
    path:     /signin
    defaults:
        _controller: Themes\MySuperTheme\Controllers\LoginController::loginAction
```

Avertissement : If your **login** route is *inside* your firewall **and** your access map require an other role than `IS_AUTHENTICATED_ANONYMOUSLY` you must add a special access map entry to enable your public visitor to access your login page.

```
$this->container['accessMap']->add(new RequestMatcher('^/press/signin'),
['IS_AUTHENTICATED_ANONYMOUSLY']);
```

Add this line with your login page pattern **before** adding your firewall entry. Access map entries order is important !

In your `LoginController`, just add error handling from the `securityAuthenticationUtils` service to display a feedback on your login form :

```
/**
 * {@inheritdoc}
 */
public function loginAction(
    Request $request,
    $_locale = 'en'
) {
    $translation = $this->bindLocaleFromRoute($request, $_locale);
    $this->prepareThemeAssignation(null, $translation);
    $helper = $this->get('securityAuthenticationUtils');
    $this->assignation['last_username'] = $helper->getLastUsername();
    $this->assignation['error'] = $helper->getLastAuthenticationError();

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

Then, you can create your *login* form as you want. Just use the required fields :

- `_username`
- `_password`

And do not forget to set your form *action* to `{{ path('themeLoginCheck') }}` and to use *POST* method.

```
{% if error %}
    <div class="alert alert-danger"><i class="fa fa-warning"></i> {{ error.
↪message|trans }}</div>
{% endif %}
<form id="login-form" class="form" action="{{ path('themeLoginCheck') }}" method=
↪"post">
    <div class="form-group">
        <label class="control-label" for="_username">{% trans %}username{%␣
↪endtrans %}</label>
        <input class="form-control" type="text" name="_username" id="_username"␣
↪placeholder="{% trans %}username{% endtrans %}" value="" />
```

```
    </div>
    <div class="form-group">
        <label class="control-label" for="_password">{% trans %}password{%␣
↪endtrans %}</label>
        <input class="form-control" type="password" name="_password" id="_password
↪" placeholder="{% trans %}password{% endtrans %}" value="" />
    </div>
    <div class="form-group">
        <label class="control-label" for="_remember_me">{% trans %}keep_me_logged_
↪in{% endtrans %}</label>
        <input class="form-control" type="checkbox" name="_remember_me" id="_
↪remember_me" value="1" />
    </div>
    <div class="form-group">
        <button class="btn btn-primary" type="submit"><i class="fa fa-signin"></i>
↪{% trans %}login{% endtrans %}</button>
    </div>
</form>
```

Configuring a root firewall map entry with FirewallEntry class

You may want to offer authentication for every pages of your website and manage access control manually within your node-type controllers. In that case you need to override default front-end Firewall map entry with your own and defined login/logout paths.

```
use RZ\Roadiz\Utils\Security\FirewallEntry;
use Pimple\Container;

/**
 * {@inheritdoc}
 */
public static function addDefaultFirewallEntry(Container $container)
{
    /*
     * Do not call parent method
     */

    $firewallBasePattern = '^/';
    $firewallBasePath = '/';
    $firewallLogin = '/accounts';
    $firewallLogout = '/accounts/logout';
    $firewallLoginCheck = '/accounts/login_check';

    /*
     * You MUST use IS_AUTHENTICATED_ANONYMOUSLY base role not to prevent
     * users to access your website
     */
    $firewallBaseRole = 'IS_AUTHENTICATED_ANONYMOUSLY';

    $firewallEntry = new FirewallEntry(
        $container,
        $firewallBasePattern,
        $firewallBasePath,
        $firewallLogin,
        $firewallLogout,
        $firewallLoginCheck,
        $firewallBaseRole
    );
    // Allow anonymous authentication
    $firewallEntry->withAnonymousAuthenticationListener()
                  ->withSwitchUserListener()
```

```
                    // Automatically redirect to themeLoginPage route
                    // if AccessDeniedException is thrown
                    ->withAccessDeniedHandler('themeLoginPage')
                    ->withReferer();

    /*
     * Finally add this entry to the Roadiz
     * firewall map.
     */
    $container['firewallMap']->add(
        $firewallEntry->getRequestMatcher(),
        $firewallEntry->getListeners(),
        $firewallEntry->getExceptionListener()
    );
}
```

For the moment, every pages of your website will be public. You'll need to use `is_granted` *Twig* filter and `$this->denyAccessUnlessGranted($role)` method to manage access control to your contents.

Multi-theme website

If your website has more than one theme you must disable firewall entries on every non-main theme app class not to register duplicated firewall entries with the same access-map rules.

For example, if you registered a `MainTheme` and a `SecondaryTheme`, add the following lines to your `themes/SecondaryTheme/SecondaryThemeApp.php` class :

```
/**
 * {@inheritdoc}
 */
public static function addDefaultFirewallEntry(Container $container)
{
    /*
     * Do not register any firewall entry
     */
}
```

Sending emails from your website

`emailManager` factory service is available to send emails from your application. It's already configured to use simple CSS styles and you can use an HTML and plain-text templates.

```
$request = $this->get('request');
$receiver = 'dest@test.com';
$origin = $this->get('settingsBag')->get('email_sender');

$emailManager = $this->get('emailManager');
$title = $this->getTranslator()->trans('hello_world');
$emailManager->setEmailTitle($title);
$emailManager->setSubject($title);
$emailManager->setReceiver($receiver);
$emailManager->setOrigin($origin);

$emailManager->setAssignation([
    'content' => 'Bla bla bla',
    'title' => $title,
    'site' => $this->get('settingsBag')->get('site_name'),
    'disclaimer' => 'You are receiving this email because you rocks!',
]);
$emailManager->setEmailTemplate('@MySuperTheme/emails/email.html.twig');
```

```
$emailManager->setEmailPlainTextTemplate('@MySuperTheme/emails/email.txt.twig');
$emailManager->send();
```

Assignation works the same way as HTML template for your website, you must assign every content and information you'll need to print in your emails.

Note : Be careful, every image path or links **must be** an absolute URL, not a path as your receivers won't be able to resolve your full domain name. Make sure you are using `url()` instead of `path()` for links and wrap with `absolute_url()` method your static assets (like in the example before).

Your *emails/email.html.twig* template should inherits from Roadiz *base_email.html.twig* template.

```
{% extends 'base_email.html.twig' %}

{% block title %}<title>{{ title }}</title>{% endblock %}

{% block content_table %}
    <table width="100%" cellpadding="0" cellspacing="0">
        <tr>
            <td class="content-block">
                <h1>{{ title }}</h1>
            </td>
        </tr>
        <tr>
            <td class="content-block">{{ content|markdown }}</td>
        </tr>
    </table>
{% endblock %}
```

Your *emails/email.txt.twig* template should inherits from Roadiz *base_email.txt.twig* template.

```
{% extends 'base_email.txt.twig' %}

{% block title %}{{ title }}{% endblock %}
{% block content_table %}{{ content|markdown|strip_tags }}{% endblock %}
```

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.

```
1 use Symfony\Component\Validator\Constraints\File;
2 use Symfony\Component\Form\Extension\Core\Type\CheckboxType;
```

```php
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();

// Assign your form view to display it in Twig.
$this->assignation['contactForm'] = $form->createView();
```

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\Core\Exceptions\ForceResponseException`. For example, in a `Themes\MyAwesomeTheme\Controllers\Blocks\ContactBlockController`, instead of returning the `contactFormManager` response, you will have to throw a `ForceResponseException` with it as an argument.

```
// ./themes/MyAwesomeTheme/Controllers/Blocks/ContactBlockController.php

use RZ\Roadiz\Core\Exceptions\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();

// Assign your form view to display it in Twig.
$this->assignation['contactForm'] = $form->createView();

return $this->render('blocks/contactformblock.html.twig', $this->assignation);
```

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.

```
$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.

```
{# Recaptcha v2 #}
<script src='https://www.google.com/recaptcha/api.js'></script>
```

```
{# In your theme' forms.html.twig file #}
{% block recaptcha_widget -%}
    {# Recaptcha v2 #}
    <div class="g-recaptcha" data-sitekey="{{ configs.publicKey }}"></div>
{%- endblock recaptcha_widget %}
```

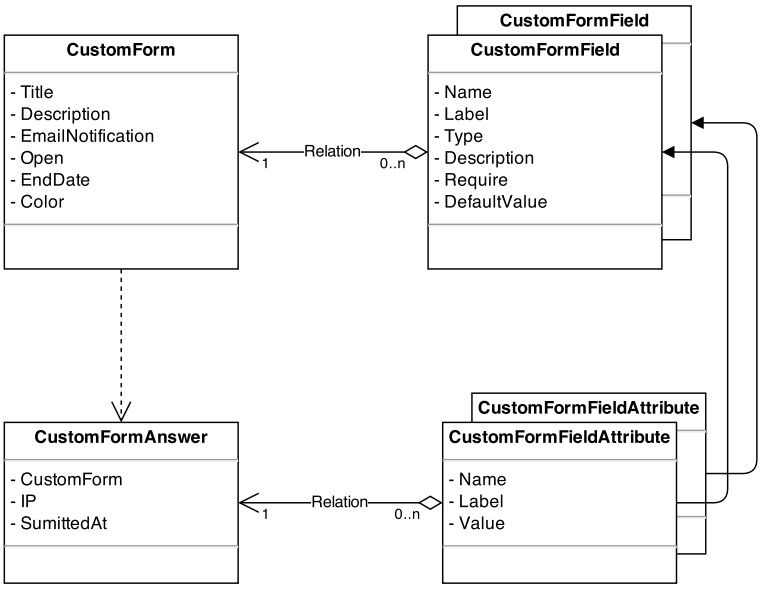
Note : Backend logic with Google Recaptcha is compatible with v2 **and** v3. You'll need to adapt your frontend logic.

Sending contact form and accept application/json

If you want to send your contact form using *window.fetch* and *window.FormData*, Roadiz will still generate an html-based *Response* or *RedirectResponse*. You need to add *Accept : application/json* header to your request so that Roadiz will respond as JSON. Roadiz *JsonResponse* will contain *success* message or *error* messages for each wrong fields.

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.

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.

```
use RZ\Roadiz\Core\Entities\CustomForm;
use RZ\Roadiz\Core\Exceptions\EntityAlreadyExistsException;
use RZ\Roadiz\Core\Exceptions\ForceResponseException;
use Symfony\Cmf\Component\Routing\RouteObjectInterface;
use RZ\Roadiz\Utils\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.
          */
         $helper = new CustomFormHelper($this->get('em'), $customForm);
         $form = $helper->getFormFromAnswer($this->get('formFactory'), null, true, [
             'recaptcha_public_key' => $this->get('settingsBag')->get('recaptcha_public_key'),
             'recaptcha_private_key' => $this->get('settingsBag')->get('recaptcha_private_key'),
             'request' => $request,
         ]);
         $form->handleRequest($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['session']['messages'] = $this->get('session')->getFlashBag()->all();
         $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 -%}
    <div class="g-recaptcha" data-sitekey="{{ configs.publicKey }}"></div>
{%- 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 Services

Roadiz is built upon Pimple dependency injection container. Thanks to this architecture, all Core and Backoffice services are available from any controller in your themes.

```
$this->get('nameOfService');
```

- Doctrine entity manager : `$this->get('em')`
- Twig rendering environment : `$this->get('twig.environment')`
- Translator : `$this->get('translator')`
- Url matcher : `$this->get('urlMatcher')`
- Url generator : `$this->get('urlGenerator')`
- Authorization checker : `$this->get('securityAuthorizationChecker')`

— User token storage : `$this->get('securityTokenStorage')`
— Firewall : `$this->get('firewall')`
— Assets packages : `$this->get('assetPackages')`
— Email manager : `$this->get('emailManager')`
— Contact form manager : `$this->get('contactFormManager')`
— Accessing global settings *ParameterBag* : `$this->get('settingsBag')`
— Accessing global node-types *ParameterBag* : `$this->get('nodeTypesBag')`
— Getting an entity handler : `$this->get('factory.handler')->getHandler($myEntity)`
— ...

Entity APIs

All these services are Doctrine repository wrappers meant to ease querying entities inside your themes and according to `AuthorizationChecker`. This will implicitely check if nodes or node-sources are published when you request them without bothering to insert the right criteria in your *findBy* calls.

Each of these implements `AbstractApi` methods `getBy` and `getOneBy`
— nodeApi
— nodeTypeApi
— nodeSourceApi
— tagApi

Using Solr API

Solr is a really powerful tool to search over your node database with a clever plain-text search engine and the ability to highlight your criteria in the search results. Before going further, make sure that a Solr server is available and that it is well configured in your `config.yml`. You can use the `bin/roadiz solr:check` command to verify and then `bin/roadiz solr:reindex` command to force synchronizing your node database with Solr index.

You can use the `solr.search.nodeSource` service and its two methods to get node-sources from a search query.

Simple search results

`$this->get('solr.search.nodeSource')->search()` method will return a `SolrSearchResults` *traversable* object listing `NodesSources` :

```
$criteria = [];
/** @var SolrSearchResults $results */
$results = $this->get('solr.search.nodeSource')
                ->search(
                    $request->get('q'), # Use ?q query parameter to search with
                    $criteria,          # a simple criteria array to filter search
↪results
                    10,                 # result count
                    true                # Search in tags too
                    10000               # Proximity (optional, default: 10000)
                    1                   # Page (optional, default: 1)
                );

foreach ($results as $nodeSource) {
    # NodesSources object
    echo $nodeSource->getTitle();
}
```

Search results with highlighting

`$this->get('solr.search.nodeSource')->searchWithHighlight()` method will return a `SolrSearchResults` *traversable* object with arrays with a simple structure : `nodeSource` for the NodesSources object and `highlighting` for the *html* data with highlighted text wrapped in `span.solr-highlight` html tag.

```
$criteria = [];
/** @var SolrSearchResults $results */
$results = $this->get('solr.search.nodeSource')
                ->searchWithHighlight(
                    $request->get('q'), # Use ?q query parameter to search with
                    $criteria,          # a simple criteria array to filter search
↪results
                    10,                 # result count
                    true                # Search in tags too
                    10000               # Proximity (optional, default: 10000)
                    1                   # Page (optional, default: 1)
                );

foreach ($results as $result) {
    # NodesSources object
    $nodeSource = $result['nodeSource'];
    # String object (HTML)
    $highlight = $result['highlighting'];
}
```

Count search results

`$this->get('solr.search.nodeSource')->search()` and `$this->get('solr.search.nodeSource')->searchWithHighlight()``methods will return a ``SolrSearchResults` *traversable* object. You can use `SolrSearchResults::getResultCount()` to get results count.

```
$criteria = [];
/** @var SolrSearchResults $results */
$results = $this->get('solr.search.nodeSource')
                ->search(
                    $request->get('q'), # Use ?q query parameter to search with
                    $criteria,          # a simple criteria array to filter search
↪results
                    10,                 # result count
                    true                # Search in tags too
                    10000               # Proximity (optional, default: 10000)
                    1                   # Page (optional, default: 1)
                );

$pageCount = ceil($results->getResultCount()/$this->getItemPerPage());
$itemPerPage = $this->getItemPerPage();
```

Search criteria

Solr will search in *NodesSources* index by default, but it will not filter by translation, type or visibility. You can add your own search criteria using similar filter names as `nodeSourceApi`.

- `visible` : boolean
- `translation` : a `Translation` object
- `_locale` : string
- `tags` : a `Tag` object or `Tag` array

— `nodeType` : a `NodeType` object
— `status` : defaults to `Node::PUBLISHED`
— `publishedAt` : `\DateTime` or array like `EntityRepository::findBy` method.

```
$criteria = [
    'visible' => true,
    'translation' => $translation,
    // Returns only published nodes-sources
    'publishedAt' => ['<=', new \DateTime()],
    'nodeType' => [
        $this->get('nodeTypesBag')->get('Page'),
        // ...
    ],
];
```

3.1.9 Services list

Here is the current list of Roadiz services registered into `Kernel` container. These can be overridden or used from any Controller with `$this->get()` method.

Kernel

stopwatch `Symfony\Component\Stopwatch\Stopwatch`
kernel `RZ\Roadiz\Core\Kernel`
dispatcher `Symfony\Component\EventDispatcher\EventDispatcher`

Assets

versionStrategy `Symfony\Component\Asset\VersionStrategy\EmptyVersionStrategy`
interventionRequestSupportsWebP `bool`
interventionRequestConfiguration `AM\InterventionRequest\Configuration`
interventionRequestSubscribers `array`
interventionRequestLogger `Monolog\Logger`
interventionRequest `AM\InterventionRequest\InterventionRequest`
assetPackages `RZ\Roadiz\Utils\Asset\Packages`

Back-office

backoffice.entries `array`

Bags

settingsBag `RZ\Roadiz\Core\Bags\Settings`
rolesBag `RZ\Roadiz\Core\Bags\Roles`
nodeTypesBag `RZ\Roadiz\Core\Bags\NodeTypes`

Console

console.commands `array`

Debug

messagescollector `DebugBar\DataCollector\MessagesCollector`

doctrine.debugstack `Doctrine\DBAL\Logging\DebugStack`

debugbar `RZ\Roadiz\Utils\DebugBar\RoadizDebugBar`

debugbar.renderer `DebugBar\JavascriptRenderer`

Doctrine

doctrine.relative_entities_paths `array`

doctrine.entities_paths `array`

em.config `Doctrine\ORM\Configuration`

DoctrinePersistenceManagerRegistry `Doctrine\Persistence\ManagerRegistry`, you should use this service instead of `em` whenever possible.

em `Doctrine\ORM\EntityManager`, you can access it using `$this->get(EntityManagerInterface::class)`.

em.eventSubscribers `array`

nodesSourcesUrlCacheProvider `Doctrine\Common\Cache\CacheProvider`

CacheProvider : :class Factory Creates a `CacheProvider::class` using Roadiz configuration, , you can access it using `$this->get(CacheProvider::class)`.

ResolveTargetEntityListener : :class `Doctrine\ORM\Tools\ResolveTargetEntityListener`, to resolve `interface` names into real entities `class`.

Embed documents

document.platforms `array`

embed_finder.youtube Factory `RZ\Roadiz\Utils\MediaFinders\YoutubeEmbedFinder`

embed_finder.vimeo Factory `RZ\Roadiz\Utils\MediaFinders\VimeoEmbedFinder`

embed_finder.dailymotion Factory `RZ\Roadiz\Utils\MediaFinders\DailymotionEmbedFinder`

embed_finder.soundcloud Factory `RZ\Roadiz\Utils\MediaFinders\SoundcloudEmbedFinder`

embed_finder.mixcloud Factory `RZ\Roadiz\Utils\MediaFinders\MixcloudEmbedFinder`

embed_finder.spotify Factory `RZ\Roadiz\Utils\MediaFinders\SpotifyEmbedFinder`

embed_finder.ted Factory `RZ\Roadiz\Utils\MediaFinders\TedEmbedFinder`

embed_finder.twitch Factory `RZ\Roadiz\Utils\MediaFinders\TwitchEmbedFinder`

Entity Api

nodeApi `RZ\Roadiz\CMS\Utils\NodeApi`

nodeTypeApi `RZ\Roadiz\CMS\Utils\NodeTypeApi`

nodeSourceApi `RZ\Roadiz\CMS\Utils\NodeSourceApi`

tagApi `RZ\Roadiz\CMS\Utils\TagApi`

Factories

emailManager Factory `RZ\Roadiz\Utils\EmailManager`

contactFormManager Factory `RZ\Roadiz\Utils\ContactFormManager`

NodeFactory : :class `RZ\Roadiz\Utils\Node\NodeFactory`. Factory to create new nodes from a title, a node-type and translation.

TagFactory : :class `RZ\Roadiz\Utils\Tag\TagFactory`. Factory to create new tags from a title, a parent tag and a translation.

factory.handler `RZ\Roadiz\Core\Handlers\HandlerFactory` Creates any Handler based on entity class.

node.handler Factory `RZ\Roadiz\Core\Handlers\NodeHandler`

nodes_sources.handler Factory `RZ\Roadiz\Core\Handlers\NodesSourcesHandler`

node_type.handler Factory `RZ\Roadiz\Core\Handlers\NodeTypeHandler`

node_type_field.handler Factory `RZ\Roadiz\Core\Handlers\NodeTypeFieldHandler`

document.handler Factory `RZ\Roadiz\Core\Handlers\DocumentHandler`

custom_form.handler Factory `RZ\Roadiz\Core\Handlers\CustomFormHandler`

custom_form_field.handler Factory `RZ\Roadiz\Core\Handlers\CustomFormFieldHandler`

folder.handler Factory `RZ\Roadiz\Core\Handlers\FolderHandler`

font.handler Factory `RZ\Roadiz\Core\Handlers\FontHandler`

group.handler Factory `RZ\Roadiz\Core\Handlers\GroupHandler`

newsletter.handler Factory `RZ\Roadiz\Core\Handlers\NewsletterHandler`

tag.handler Factory `RZ\Roadiz\Core\Handlers\TagHandler`

translation.handler Factory `RZ\Roadiz\Core\Handlers\TranslationHandler`

document.viewer Factory `RZ\Roadiz\Core\Viewers\DocumentViewer` **Deprecated** :

translation.viewer Factory `RZ\Roadiz\Core\Viewers\TranslationViewer`

user.viewer Factory `RZ\Roadiz\Core\Viewers\UserViewer`

document.url_generator Factory `RZ\Roadiz\Utils\UrlGenerators\DocumentUrlGenerator`

document.factory Factory `RZ\Roadiz\Utils\Document\DocumentFactory`

Forms

formValidator `Symfony\Component\Form\Validator\ValidatorInterface`

formFactory `Symfony\Component\Form\FormFactoryInterface`

form.extensions `array`

form.type.extensions `array`

Rollerworks\Component\PasswordStrength\Blacklist\BlacklistProviderInterface Blacklist chained provider for Password forms.

Rollerworks\Component\PasswordStrength\Validator\Constraints\BlacklistValidator Blacklist form validator to be used and instanciated by Symfony ContainerConstraintValidatorFactory.

Importers

RZ\Roadiz\CMS\Importers\ChainImporter Factory Creates an chained importer that will import serialized data based on required entity class.

RZ\Roadiz\CMS\Importers\GroupsImporter Factory Creates an importer for `Group`

RZ\Roadiz\CMS\Importers\NodesImporter Factory Creates an importer for `Node`

RZ\Roadiz\CMS\Importers\NodeTypesImporter Factory Creates an importer for `NodeType`

RZ\Roadiz\CMS\Importers\RolesImporter Factory Creates an importer for `Role`

RZ\Roadiz\CMS\Importers\SettingsImporter Factory Creates an importer for `Setting`

RZ\Roadiz\CMS\Importers\TagsImporter Factory Creates an importer for `Tag`

Logger

logger.handlers `array`

logger.path `string`

logger `Monolog\Logger`

Mailer

mailer.transport `\Swift_SmtpTransport` or `\Swift_SendmailTransport`
mailer `\Swift_Mailer`

Routing

httpKernel `Symfony\Component\HttpKernel\HttpKernel`
requestStack `Symfony\Component\HttpFoundation\RequestStack`
requestContext `Symfony\Component\Routing\RequestContext`
resolver `Symfony\Component\HttpKernel\Controller\ControllerResolver`
argumentResolver `Symfony\Component\HttpKernel\Controller\ArgumentResolver`
router `Symfony\Cmf\Component\Routing\ChainRouter`
staticRouter `RZ\Roadiz\Core\Routing\StaticRouter`
nodeRouter `RZ\Roadiz\Core\Routing\NodeRouter`
redirectionRouter `RZ\Roadiz\Core\Routing\RedirectionRouter`
urlGenerator Alias to `router`
httpUtils `Symfony\Component\Security\Http\HttpUtils`
routeListener `RZ\Roadiz\Core\Events\TimedRouteListener`
routeCollection `RZ\Roadiz\Core\Routing\RoadizRouteCollection`

Security

session.pdo `\PDO` or `null` if pdo session are not configured.
session.storage `Symfony\Component\HttpFoundation\Session\Storage\NativeSessionStorage`
session `Symfony\Component\HttpFoundation\Session\Session`
sessionTokenStorage `Symfony\Component\Security\Csrf\TokenStorage\SessionTokenStorage`
csrfTokenManager `Symfony\Component\Security\Csrf\CsrfTokenManager`
securityAuthenticationUtils `Symfony\Component\Security\Http\Authentication\AuthenticationUtil`
contextListener `Symfony\Component\Security\Http\Firewall\ContextListener`
accessMap `Symfony\Component\Security\Http\AccessMap`
SymfonyComponentSecurityCoreUserUserProviderInterface `Symfony\Component\Security\Core\User\Chai`
userProviders `array<Symfony\Component\Security\Core\User\UserProviderInterface>`
userChecker `Symfony\Component\Security\Core\User\UserChecker`
daoAuthenticationProvider `Symfony\Component\Security\Core\Authentication\Provider\DaoAuther`
rememberMeAuthenticationProvider `Symfony\Component\Security\Core\Authentication\Provider\Re`
rememberMeCookieName `string`
rememberMeCookieLifetime `integer`
cookieClearingLogoutHandler `Symfony\Component\Security\Http\Logout\CookieClearingLogoutHan`
tokenBasedRememberMeServices `Symfony\Component\Security\Http\RememberMe\TokenBasedRememb`
rememberMeListener `Symfony\Component\Security\Http\Firewall\RememberMeListener`
authenticationProviderList `array<Symfony\Component\Security\Core\Authentication\Provider\Aut`
authenticationManager `Symfony\Component\Security\Core\Authentication\AuthenticationProvid`
security.voters `array`
accessDecisionManager `Symfony\Component\Security\Core\Authorization\AccessDecisionManager`
securityAuthenticationTrustResolver `Symfony\Component\Security\Core\Authentication\Authenticat`
securityAuthorizationChecker `Symfony\Component\Security\Core\Authorization\AuthorizationChe`

securityTokenStorage `Symfony\Component\Security\Core\Authentication\Token\Storage\TokenSto`
securityAccessListener `Symfony\Component\Security\Http\Firewall\AccessListener`
roleHierarchy `RZ\Roadiz\Utils\Security\DoctrineRoleHierarchy`
roleHierarchyVoter `Symfony\Component\Security\Core\Authorization\Voter\RoleHierarchyVoter`
groupVoter `RZ\Roadiz\Core\Authorization\Voter\GroupVoter`
switchUser `Symfony\Component\Security\Http\Firewall\SwitchUserListener`
firewallMap `Symfony\Component\Security\Http\FirewallMap`
passwordEncoder `array`
userImplementations `array`
userEncoderFactory `Symfony\Component\Security\Core\Encoder\EncoderFactory`
firewall `RZ\Roadiz\Utils\Security\TimedFirewall`
accessDeniedHandler `RZ\Roadiz\Core\Authorization\AccessDeniedHandler`
nodeChrootResolvers `array<RZ\Roadiz\Core\Authorization\Chroot\NodeChrootResolver>`
RZRoadizCoreAuthorizationChrootNodeChrootResolver `RZ\Roadiz\Core\Authorization\Chroot\NodeChr`

Serialization

JMSSerializerSerializerBuilder `JMS\Serializer\SerializerBuilder`
serializer.subscribers `array`
serializer `JMS\Serializer\Serializer`

Solr

solr `Solarium\Client`
AdapterInterface `Solarium\Core\Client\Adapter\AdapterInterface`
SolariumFactoryInterface `RZ\Roadiz\Core\SearchEngine\SolariumFactory`
solr.ready `boolean` Return true if Solr server is reachable.
solr.search.nodeSource Factory `RZ\Roadiz\Core\SearchEngine\NodeSourceSearchHandler`
solr.search.document Factory `RZ\Roadiz\Core\SearchEngine\DocumentSearchHandler`

Themes

themeResolver `RZ\Roadiz\Utils\Theme\StaticThemeResolver`
ThemeGenerator `RZ\Roadiz\Utils\Theme\ThemeGenerator`
logger.themes `Monolog\Logger`

Translation

defaultTranslation `RZ\Roadiz\Core\Entities\Translation` or `null` if you don't have any default translation.
translator.locale `string` or `null`
translator `Symfony\Component\Translation\Translator`

Twig

twig.cacheFolder `string`
twig.loaderFileSystem `Twig\Loader\FilesystemLoader`
twig.environment_class Private `Twig\Environment`
twig.formRenderer `Symfony\Bridge\Twig\Form\TwigRendererEngine`
twig.environment `Twig\Environment`
twig.extensions `Doctrine\Common\Collections\ArrayCollection`
twig.filters `Doctrine\Common\Collections\ArrayCollection`
twig.fragmentHandler `Symfony\Component\HttpKernel\Fragment\FragmentHandler`
twig.profile `Twig\Profiler\Profile`
twig.routingExtension `Symfony\Bridge\Twig\Extension\RoutingExtension`
twig.centralTruncateExtension `Twig\TwigFilter`
twig.cacheExtension `Asm89\Twig\CacheExtension\Extension`

Configuration

config.path `string`
config.handler `RZ\Roadiz\Config\YamlConfigurationHandler`
config `array`

Workflow

workflow.registry `Symfony\Component\Workflow\Registry`
workflow.node_workflow `RZ\Roadiz\Workflow\NodeWorkflow`

Utils

utils.nodeNameChecker `RZ\Roadiz\Utils\Node\NodeNameChecker`
utils.uniqueNodeGenerator `RZ\Roadiz\Utils\Node\UniqueNodeGenerator`
utils.universalDataDuplicator `RZ\Roadiz\Utils\Node\UniversalDataDuplicator`

OpenId

RZRoadizOpenIdOAuth2LinkGenerator : `RZ\Roadiz\OpenId\OAuth2LinkGenerator`
jwtRoleStrategies `array<RZ\Roadiz\OpenId\Authentication\Provider\JwtRoleStrategy>`
RZRoadizOpenIdAuthenticationProviderJwtRoleStrategy `RZ\Roadiz\OpenId\Authentication\Provider\Ch`
RZRoadizOpenIdAuthenticationProviderOAuth2AuthenticationProvider
`RZ\Roadiz\OpenId\Authentication\Provider\OAuth2AuthenticationProvider`
RZRoadizOpenIdAuthenticationProviderOpenIdAccountProvider `RZ\Roadiz\OpenId\Authentication\Provi`
RZRoadizOpenIdDiscovery `RZ\Roadiz\OpenId\Discovery` or `null`
oauth2AuthenticationProvider.validators : `array<RZ\Roadiz\OpenId\Authentication\Validator\JwtVal`

Message

MessageBusInterface : :class `Symfony\Component\Messenger\RoutableMessageBus`
logger.messenger `Monolog\Logger`

3.1.10 Serialization

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

```
$response = new JsonResponse(
    $this->get('serializer')->serialize(
        $this->getNodeSource(),
        'json',
        SerializationContext::create()->setGroups(['nodes_sources', 'urls', 'walker
↪', 'children'])
    ),
    Response::HTTP_OK,
    [],
    true
);
```

Customize existing serialized entities

Serialize nodes-sources URL

Your can add data in your serialized data from your themes using `EventSubscriberInterface` listening on `serializer.post_serialize` event. For example, you may want to get your *reachable* `NodesSources` URL in their JSON response. Here is how to do :

```
<?php
declare(strict_types=1);

namespace Themes\MyAwesomeTheme\Serialization;

use JMS\Serializer\EventDispatcher\EventSubscriberInterface;
use JMS\Serializer\EventDispatcher\ObjectEvent;
use JMS\Serializer\Metadata\StaticPropertyMetadata;
use JMS\Serializer\Visitor\SerializationVisitorInterface;
use Pimple\Container;
use RZ\Roadiz\Core\ContainerAwareInterface;
use RZ\Roadiz\Core\ContainerAwareTrait;
use RZ\Roadiz\Core\Entities\NodesSources;
use Symfony\Cmf\Component\Routing\RouteObjectInterface;
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;

final class NodesSourcesUriSubscriber implements EventSubscriberInterface,␣
↪ContainerAwareInterface
{
    use ContainerAwareTrait;

    /**
     * @param Container $container
     */
    public function __construct(Container $container)
    {
        $this->container = $container;
    }

    public static function getSubscribedEvents()
    {
        return [[
            'event' => 'serializer.post_serialize',
            'method' => 'onPostSerialize',
        ]];
    }
```

```
    /**
     * @param ObjectEvent $event
     * @return void
     */
    public function onPostSerialize(ObjectEvent $event)
    {
        $nodeSource = $event->getObject();
        $visitor = $event->getVisitor();
        $context = $event->getContext();

        if ($context->hasAttribute('groups') &&
            in_array('urls', $context->getAttribute('groups'))) {
            /** @var UrlGeneratorInterface $urlGenerator */
            $urlGenerator = $this->get('urlGenerator');
            if ($nodeSource instanceof NodesSources &&
                null !== $nodeSource->getNode() &&
                null !== $nodeSource->getNode()->getNodeType() &&
                $visitor instanceof SerializationVisitorInterface &&
                $nodeSource->getNode()->isPublished() &&
                $nodeSource->getNode()->getNodeType()->isReachable()
            ) {
                $visitor->visitProperty(
                    new StaticPropertyMetadata('string', 'url', []),
                    $urlGenerator->generate(
                        RouteObjectInterface::OBJECT_BASED_ROUTE_NAME,
                        [
                            RouteObjectInterface::ROUTE_OBJECT => $nodeSource
                        ],
                        UrlGeneratorInterface::ABSOLUTE_URL
                    )
                );
            }
        }
    }
}
```

Then register your `NodesSourcesUriSubscriber` in your theme services :

```
$container->extend('serializer.subscribers', function (array $subscribers,␣
↪Container $c) {
    $subscribers[] = new NodesSourcesUriSubscriber($c);
    return $subscribers;
});
```

Then your JSON response will contain the additional key `url` :

```
{
    ...
    "url": "http://myawesomesite.test/about-us"
}
```

You can add every piece of data in your serialized responses, even node children by injecting a *tree-walker*.

Serialize documents URL

Same way for injecting your document URL (i.e. image path, or iframe source), we need to call Roadiz document URL generator for any Document serialized into our JSON response :

```
<?php
declare(strict_types=1);
```

```php
namespace Themes\MyAwesomeTheme\Serialization;

use JMS\Serializer\EventDispatcher\EventSubscriberInterface;
use JMS\Serializer\EventDispatcher\ObjectEvent;
use JMS\Serializer\Metadata\StaticPropertyMetadata;
use JMS\Serializer\Visitor\SerializationVisitorInterface;
use Pimple\Container;
use RZ\Roadiz\Core\ContainerAwareInterface;
use RZ\Roadiz\Core\ContainerAwareTrait;
use RZ\Roadiz\Core\Entities\Document;
use RZ\Roadiz\Core\Exceptions\InvalidEmbedId;
use RZ\Roadiz\Utils\MediaFinders\EmbedFinderFactory;
use RZ\Roadiz\Utils\UrlGenerators\DocumentUrlGenerator;

final class DocumentUriSubscriber implements EventSubscriberInterface,␣
↪ContainerAwareInterface
{
    use ContainerAwareTrait;

    /**
     * @param Container $container
     */
    public function __construct(Container $container)
    {
        $this->container = $container;
    }

    public static function getSubscribedEvents()
    {
        return [[
            'event' => 'serializer.post_serialize',
            'method' => 'onPostSerialize',
            'class' => Document::class,
        ]];
    }

    /**
     * @param ObjectEvent $event
     * @return void
     */
    public function onPostSerialize(ObjectEvent $event)
    {
        $document = $event->getObject();
        $visitor = $event->getVisitor();
        $context = $event->getContext();

        if (null !== $this->container &&
            $context->hasAttribute('groups') &&
            in_array('urls', $context->getAttribute('groups'))) {
            /** @var DocumentUrlGenerator $urlGenerator */
            $urlGenerator = $this->get('document.url_generator')->setDocument(
↪$document);

            if ($document instanceof Document &&
                $visitor instanceof SerializationVisitorInterface) {
                $urls = [];
                if ($document->isEmbed() && $document->getEmbedId()) {
                    try {
                        /** @var EmbedFinderFactory $embedFinderFactory */
                        $embedFinderFactory = $this->
↪get(EmbedFinderFactory::class);
                        if (null !== $document->getEmbedPlatform() &&
```

```
                                $embedFinderFactory->supports($document->
↪getEmbedPlatform())) {
                                $embedFinder = $embedFinderFactory->createForPlatform(
                                    $document->getEmbedPlatform(),
                                    $document->getEmbedId()
                                );
                                if (null !== $embedFinder) {
                                    $urls['embed'] = $embedFinder->getSource();
                                }
                            }
                        } catch (InvalidEmbedId $embedException) {
                        }
                    }
                    if ($document->isProcessable()) {
                        $visitor->visitProperty(
                            new StaticPropertyMetadata('array', 'urls', []),
                            array_merge($urls, [
                                'original' => $urlGenerator->setOptions([
                                    'noProcess' => true,
                                ])->getUrl(true)
                            ], $this->getSizes($urlGenerator))
                        );
                    } else {
                        if ($document->hasThumbnails()) {
                            $thumbnail = $document->getThumbnails()->first();
                            if ($thumbnail instanceof Document && $thumbnail->
↪isProcessable()) {
                                /** @var DocumentUrlGenerator $thumbUrlGenerator */
                                $thumbUrlGenerator = $this->get('document.url_generator
↪')->setDocument($thumbnail);
                                $urls = array_merge($urls, $this->getSizes(
↪$thumbUrlGenerator));
                            }
                        }
                        $visitor->visitProperty(
                            new StaticPropertyMetadata('array', 'urls', []),
                            array_merge($urls, [
                                'original' => $urlGenerator->setOptions([
                                    'noProcess' => true
                                ])->getUrl(true),
                            ])
                        );
                    }
                }
            }
        }

        protected function getSizes(DocumentUrlGenerator $generator): array
        {
            return [];
        }
}
```

Groups

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.

tag Serialize entities in a `Tag` context.

tag_base Serialize entities in a `Tag` context.

node_type Serialize entities in a `NodeType` context.

attribute Serialize entities in a `Attribute` context.

custom_form Serialize entities in a `CustomForm` context.

document Serialize entities in a `Document` context.

folder Serialize entities in a `Folder` context.

translation Serialize entities in a `Translation` context.

setting Serialize entities in a `Setting` context.

setting_group Serialize entities in a `SettingGroup` context.

3.1.11 Études de cas

Quelques guides pas à pas pour travailler avec Roadiz.

Download a website on my computer to work with Vagrant

This case study is meant to get a fresh development environment from an **existing** Roadiz **Source edition** website and theme. Following code snippets are using some variables data, in theses examples I'll use :

— `MYUSER` as the MySQL database user.
— `MYPASSWORD` as the MySQL database user password.
— `MYDATABASE` as the MySQL database name.
— `~/Documents/Websites` as the working directory on your own computer.
— `database-YYYY-mm-dd.sql` is the mysql dump file name, replace `YYYY-mm-dd` with the current date.
— `mysuperwebsite` is your website root folder.
— `git@github.com:johndoe/SuperTheme.git` is an example *Github* repository for your theme.
— `SuperTheme` is an example theme name and folder.

On the production server :

1. Generate a database dump on your production server.

```
mysqldump -uMYUSER -pMYPASSWORD MYDATABASE > database-YYYY-mm-dd.sql
```

Then download it on your computer. You can also use *phpmyadmin* web tool to export your database tables. Make sure to disable *foreign key verification* and add the *DROP IF EXISTS* directive on *phpmyadmin* export form.

On your computer :

1. Clone Roadiz on your favorite folder, choose well between master or develop branch if you want the stable version or the latest features.

```
cd ~/Documents/Websites;
# Here I choose the develop branch, because I'm warrior
git clone -b develop https://github.com/roadiz/roadiz.git mysuperwebsite;
```

2. Clone your website theme in Roadiz `themes/` folder, choose well your branch too. If you already have a *develop* branch, clone with `-b develop` option.

```
cd ~/Documents/Websites/mysuperwebsite/themes;
# My theme already has a develop branch so...
git clone -b develop git@github.com:johndoe/SuperTheme.git SuperTheme;
```

3. **[Optional]** Initialize *git-flow* on the theme. You should always work on *develop*. *Master* branch is only for releases. If you don't have *git-flow* on your computer, you can find some help on the official documentation.

```
cd ~/Documents/Websites/mysuperwebsite/themes/SuperTheme;
# You must fetch every available branches before initializing git flow
git checkout master;
git checkout develop;
git flow init;
# Follow instructions
# Git flow should checkout on develop branch for you
```

4. Install Roadiz' *Composer* dependencies (after cloning the theme to be sure that all *composer* dependencies are loaded)

```
cd ~/Documents/Websites/mysuperwebsite;
composer install --no-dev;
```

5. Launch your Vagrant environment. Do not to automatically provision your VM if you want to choose what tool to install.

```
vagrant up --no-provision;
# ... lots of lines, bla bla bla
```

Choose tools to install on your VM, `roadiz` provisioner is mandatory... obviously, `devtools` provisioner will install *Composer*, *Node.js*, *Grunt* and *Bower* commands. If you have lots of website on your computer, it's better to install these tools directly on your host machine, they will be more effective than on the VM. And you will be able to take advantage of *Composer* and *NPM* cache between your dev websites.

```
# Everything
vagrant provision --provision-with roadiz,phpmyadmin,mailcatcher,solr,devtools
# OR on a dev computer
vagrant provision --provision-with roadiz,phpmyadmin,mailcatcher,solr
```

6. Import your database dump. First, you'll need to copy it into your Roadiz website to make it available within your Vagrant VM. Then import it in your VM using the `mysql` tool.

```
mv ~/Downloads/database-YYYY-mm-dd.sql ~/Documents/Websites/mysuperwebsite/
↪database-YYYY-mm-dd.sql;
cd ~/Documents/Websites/mysuperwebsite;
# Enter your VM
vagrant ssh;
# Your website is located in /var/www folder
cd /var/www;
mysql -uroadiz -proadiz roadiz < database-YYYY-mm-dd.sql;
# Exit your VM
exit;
```

7. Update your conf/config.yml file to fill in your mysql credentials.

```
cd ~/Documents/Websites/mysuperwebsite;
# composer should have create a starter config file for you
subl conf/config.yml; # If you work SublimeText
```

8. Use the `bin/roadiz generate:nsentities` to regenerate *Doctrine* entities existing in database but not as files.

```
cd ~/Documents/Websites/mysuperwebsite;
vagrant ssh;
cd /var/www;
bin/roadiz generate:nsentities;
# You may have to check database schema if your production website is not up to
# date with latest Roadiz
bin/roadiz orm:schema-tool:update --dump-sql --force;
```

9. Download your production documents to your dev VM. You don't have to do this within your VM.

```
cd ~/Documents/Websites/mysuperwebsite/files;
rsync -avcz -e "ssh -p 22" myuser@superwebsite.com:~/path/to/roadiz/files/ ./
# do not forget ending slashes in both paths.
```

10. If you are using a Vagrant VM you have to add your IP address to the `dev.php` file to authorize your host computer to use the development environment.

11. Connect to `http://localhost:8080/dev.php` to begin. Every outgoing emails should be catched by *Mailcatcher*. You can see them at address `http://localhost:1080`.

3.1.12 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.

Running tests

If you developed a new feature or simply want to try out an installation of Roadiz you can run unit-tests. For this you will need to install the testing dependencies, this can easily be done using :

```
composer update --dev
```

You have to run unit-tests on a dedicated database not to lose any existing Roadiz website. You can create a `conf/config_test.yml` YAML configuration which will be read only for this environment. Then, wire this configuration to a blank database. Unit-tests can be launched by the following command :

```
php bin/phpunit -v --bootstrap=tests/bootstrap.php --whitelist ./src tests/
```

If your are writing a feature, don't forget to write a unit test for it. You can find some example in the folder `tests`. In Roadiz, there are 4 types of tests :

— Standard tests which must extend `\PHPUnit_Framework_TestCase`. These tests should only test simple logic methods and classes as they won't require Roadiz kernel to boot up.
— Kernel dependent tests which must extend *RZRoadizTestsKernelDependentCase*'. These tests should only test logic classes and methods inside Roadiz kernel without any database concern.
— Schema dependent tests which must extend `RZ\Roadiz\Tests\SchemaDependentCase`. These tests should only test low level database methods and classes without relying on node-types or translations. Use this type of testing if you want to test Roadiz entities and repositories methods except for Nodes and NodeTypes.
— DefaultTheme dependent tests which must extend `RZ\Roadiz\Tests\DefaultThemeDependentCase`. These tests rely on a complete Roadiz installation with existing node-types and translation. They are longer to prepare as PHPUnit must install a fresh Roadiz with DefaultTheme at each case.

Note : Each `SchemaDependentCase` and `DefaultThemeDependentCase` will provision a fresh Roadiz database then drop it. Make sure to use a dedicated database. If you execute unit-tests from an existing Roadiz website, you'll have to run `bin/roadiz generate:nsentities` at the end of your testing session to build your NodesSources classes again (every environment share the same `gen-src` folder).

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 -l 3 src
```

Standard Edition

Roadiz *Standard Edition* comes with a `Makefile` to make it easier to test your sources and your theme. Just execute `make test` and `phpcbf` and `phpstan` will test your theme sources. If you are using *Gitlab CI/CD* with our `.gitlab-ci.yml` file, those tests will be executed at each pipeline trigger.

3.1.13 Troubleshooting

Empty caches manually for an environment

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

If you still get errors from a specific env and you are using an *OPcode* cache or *var cache* (*APC*, *XCache*), call `clear_cache.php` entry point *from your browser* or execute `curl http://localhost/clear_cache.php` from your 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/roadiz generate:nsentities;
bin/roadiz orm:schema-tool:update --dump-sql --force;
bin/roadiz cache:clear -e prod;
```

If you are using a *OPCode var cache* like *APC*, *XCache*, you should purge it as Roadiz stores doctrine configuration there for better performances, call `clear_cache.php` entry point from your browser or `curl http://localhost/clear_cache.php` from your command line.

Running Roadiz behind a reverse proxy

If you are behind a reverse-proxy like *Varnish* or *Nginx proxy* on a *Docker* environment, IP addresses, domain name and proto (https/http) could not be correctly set. So you will have to tell Roadiz to trust your proxy in order to use `X_FORWARDED_*` env vars.

Add this line to your `index.php` and `preview.php` files after `$request = Request::createFromGlobals();` line.

```
$request = Request::createFromGlobals(); // Existing line to get request
// Trust incoming request IP as your reverse proxy for only X_FORWARDED... headers.
Request::setTrustedProxies(
    [$request->server->get('REMOTE_ADDR')],
    Request::HEADER_X_FORWARDED_ALL
);
```

Find help before posting an issue on Github

Join us on Gitter : https://gitter.im/roadiz/roadiz

CHAPITRE 4

Extensions - Documentation

4.1 Extensions - Documentation

4.1.1 Extending Roadiz

It is time to see how to extend Roadiz ! As you read in Roadiz *Philosophie* part, we won't ship « plugin » or « module » like others CMS. But you will be able to add of lot of features using the part that really matters : Themes !

Theme powered CMS

We coded the Theme system to be the core of your extending experience. You don't need to change something else than your theme. So you can use a versioning tool or backup easily your work which will be only at one place.

You can add new entities. If so, don't forget to add your `Entities` namespace in Roadiz config file. With theses additional entities, you maybe will need to create a back-office entry to manage them. It's easy ! Let's see how to.

Create your own database entities

You can create a theme with your own entities. Just add your *Entities* folder to the global configuration file (`app/conf/config.yml`).

```
entities:
    - "../vendor/roadiz/roadiz/src/Roadiz/Core/Entities"
    - "../vendor/roadiz/models/src/Roadiz/Core/AbstractEntities"
    - "gen-src/GeneratedNodeSources"
    - "../themes/MyTheme/Entities"
```

Verify if everything is OK by checking migrations :

```
bin/roadiz orm:schema-tool:update --dump-sql;
```

If you see your entities being created and no system database erased, just apply your migration with `--force`. If Doctrine send some error, you probably need to clear metadata cache :

```
bin/roadiz cache:clear -e prod;
```

Clearing cache from command line **will not empty op-code cache**. Be sure to call `clear_cache.php` entry point to actually clear *PHP-FPM* related caches. You can use an `curl` command if your website is accessible from *localhost* :

```
curl http://localhost/clear_cache.php;
```

Add back-office entry

At first, create a controller into your theme folder, for example `themes/MyTheme/AdminControllers/AdminController`.

Example :

```php
namespace Themes\MyTheme\AdminControllers;

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

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

If you look at this exemple you can see the class extends `RozierApp` not your `MyThemeApp` class ! This will enable you to "inject" your code into Rozier Back-stage DOM and Style. But be careful to use *MyThemeApp : :getThemeDir()* as your template namespace.

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

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

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

{% block customScripts %}
<script>
    /* Custom Stripts 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 `route.yml` file.

In this case the route will be :

```
adminTestPage:
    # Setting your path behind rz-admin will activate Firewall
    path: /rz-admin/test
    defaults:
        _controller: Themes\MyTheme\AdminControllers\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 `MyThemeApp.php` main class and override `setupDependencyInjection` method, or create it if it doesn't exist.

```
public static function setupDependencyInjection(Container $container)
{
    parent::setupDependencyInjection($container);

    $container->extend('backoffice.entries', function (array $entries, $c) {

        /*
         * Add a customAdmin entry in your Backoffice
         */
        $entries['customAdmin'] = [
            'name' => 'customAdmin',
            'path' => $c['urlGenerator']->generate('adminTestPage'),
            'icon' => 'uk-icon-cube',
            'roles' => null,
            'subentries' => null
        ];

        return $entries;
    });
}
```

Do not forget to add `use Pimple\Container;` in your file header.

`setupDependencyInjection` method is called statically at boot time when Roadiz's kernel is running all available Themes to setup services. In the code above, you will extend `backoffice.entries` service which define every buttons available in Rozier backstage main-menu.

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 :

```
$entries['customAdmin'] = [
    'name' => 'customAdmin',
    'path' => null,
    'icon' => 'uk-icon-cube',
    'roles' => null,
    'subentries' => [
        'customAdminPage' => [
            'name' => 'customAdmin page',
            'path' => $c['urlGenerator']->generate('adminTestPage'),
            'icon' => 'uk-icon-cube',
            'roles' => null
        ],
        // Add others if you want
    ]
];
```

You can restrict buttons to users with specific roles. Just replace `'roles' => null` with `'roles' => array('ROLE_ACCESS_NODES')`. You can even create your own roles to take full power of Roadiz extension system.

Avertissement : Adding roles in `backoffice.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\Core\Events\Node\NodeCreatedEvent::class`
- `RZ\Roadiz\Core\Events\Node\NodeUpdatedEvent::class`
- `RZ\Roadiz\Core\Events\Node\NodeDeletedEvent::class`
- `RZ\Roadiz\Core\Events\Node\NodeUndeletedEvent::class`
- `RZ\Roadiz\Core\Events\Node\NodeDuplicatedEvent::class`
- `RZ\Roadiz\Core\Events\Node\NodePathChangedEvent::class`
- `RZ\Roadiz\Core\Events\Node\NodeTaggedEvent::class` : This event is triggered for tag and un-tag action.
- `RZ\Roadiz\Core\Events\Node\NodeVisibilityChangedEvent::class` : This event is triggered each time a node becomes visible or unvisible.
- `RZ\Roadiz\Core\Events\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\Core\Events\NodesSourcesEvents`

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

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

— `RZ\Roadiz\Core\Events\NodesSources\NodesSourcesIndexingEvent::class` : This event type is dispatched during Solr indexation. Your event will be `\RZ\Roadiz\Core\Events\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\Core\Events\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->get('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\Core\Events\Tag\TagCreatedEvent::class`
— `RZ\Roadiz\Core\Events\Tag\TagUpdatedEvent::class`
— `RZ\Roadiz\Core\Events\Tag\TagDeletedEvent::class`

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

Folders events

— `RZ\Roadiz\Core\Events\Folder\FolderCreatedEvent::class`
— `RZ\Roadiz\Core\Events\Folder\FolderUpdatedEvent::class`
— `RZ\Roadiz\Core\Events\Folder\FolderDeletedEvent::class`

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

Translations events

— `RZ\Roadiz\Core\Events\Translation\TranslationCreatedEvent::class`
— `RZ\Roadiz\Core\Events\Translation\TranslationUpdatedEvent::class`
— `RZ\Roadiz\Core\Events\Translation\TranslationDeletedEvent::class`

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

UrlAlias events

— `RZ\Roadiz\Core\Events\UrlAlias\UrlAliasCreatedEvent::class`
— `RZ\Roadiz\Core\Events\UrlAlias\UrlAliasUpdatedEvent::class`
— `RZ\Roadiz\Core\Events\UrlAlias\UrlAliasDeletedEvent::class`

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

User events

- `RZ\Roadiz\Core\Events\User\UserCreatedEvent::class`
- `RZ\Roadiz\Core\Events\User\UserUpdatedEvent::class`
- `RZ\Roadiz\Core\Events\User\UserDeletedEvent::class`
- `RZ\Roadiz\Core\Events\User\UserDisabledEvent::class`
- `RZ\Roadiz\Core\Events\User\UserEnabledEvent::class`
- `RZ\Roadiz\Core\Events\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 Themes\MyTheme\Event;

use GeneratedNodeSources\NSGroupBlock;
use GeneratedNodeSources\NSPage;
use Pimple\Container;
use RZ\Roadiz\Core\ContainerAwareInterface;
use RZ\Roadiz\Core\ContainerAwareTrait;
use RZ\Roadiz\Core\Entities\NodesSources;
use RZ\Roadiz\Core\Events\NodesSources\NodesSourcesIndexingEvent;
use RZ\Roadiz\Core\SearchEngine\SolariumFactoryInterface;
use RZ\Roadiz\Core\SearchEngine\SolariumNodeSource;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;

final class PageIndexingEventSubscriber implements EventSubscriberInterface,␣
↪ContainerAwareInterface
{
    use ContainerAwareTrait;

    public function __construct(Container $container)
    {
        $this->container = $container;
    }

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

```
    public function onIndexing(NodesSourcesIndexingEvent $event)
    {
        $nodeSource = $event->getNodeSource();
        if ($nodeSource instanceof NSPage || $nodeSource instanceof NSGroupBlock) {
            $assoc = $event->getAssociations();

            /*
             * Fetch every non-reachable blocks
             * to gather their text content in master page document
             */
            $children = $this->container['nodeSourceApi']->getBy([
                'node.nodeType.reachable' => false,
                'node.visible' => true,
                'translation' => $nodeSource->getTranslation(),
                'node.parent' => $nodeSource->getNode(),
            ]);

            /** @var NodesSources $child */
            foreach ($children as $child) {
                /** @var SolariumNodeSource $solarium */
                $solarium = $this->container[SolariumFactoryInterface::class]->
↪createWithNodesSources($child);
                // Fetch all fields array association AS sub-resources (i.e. do␣
↪not index their title)
                $childAssoc = $solarium->getFieldsAssoc(true);
                $assoc['collection_txt'] = array_merge(
                    $assoc['collection_txt'],
                    $childAssoc['collection_txt']
                );
            }

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

Then register this subscriber to your event-dispatcher :

```
# In your theme ServiceProvider...
$container->extend('dispatcher', function (EventDispatcherInterface $dispatcher,␣
↪Container $c) {
    $dispatcher->addSubscriber(new PageIndexingEventSubscriber($c));
    return $dispatcher;
});
```

Index

T

U

V

W