
Roadiz Documentation

Version 1.4.0

Ambroise Maupate, Julien Blanchet

août 31, 2020

Table des matières

1 Philosophie	3
2 Utilisateur - Documentation	5
2.1 Utilisateur - Documentation	5
2.1.1 Écrire en Markdown	6
2.1.2 Gérer les nœuds	9
2.1.3 Gérer les types de nœuds	20
2.1.4 Gérer les documents	25
2.1.5 Gérer les utilisateurs	27
3 Développeur - Documentation	31
3.1 Développeur - Documentation	31
3.1.1 First steps	31
3.1.2 Node system	46
3.1.3 Tag system	57
3.1.4 Attributes	58
3.1.5 Themes	62
3.1.6 Forms	91
3.1.7 Services	96
3.1.8 Services list	99
3.1.9 Serialization	104
3.1.10 Case studies	105
3.1.11 Contributing	107
3.1.12 Troubleshooting	108
4 Extensions - Documentation	111
4.1 Extensions - Documentation	111
4.1.1 Extending Roadiz	111
4.1.2 Events	114
4.1.3 Extending Solr indexation	116
Index	119

Roadiz est un CMS polymorphe qui repose 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.

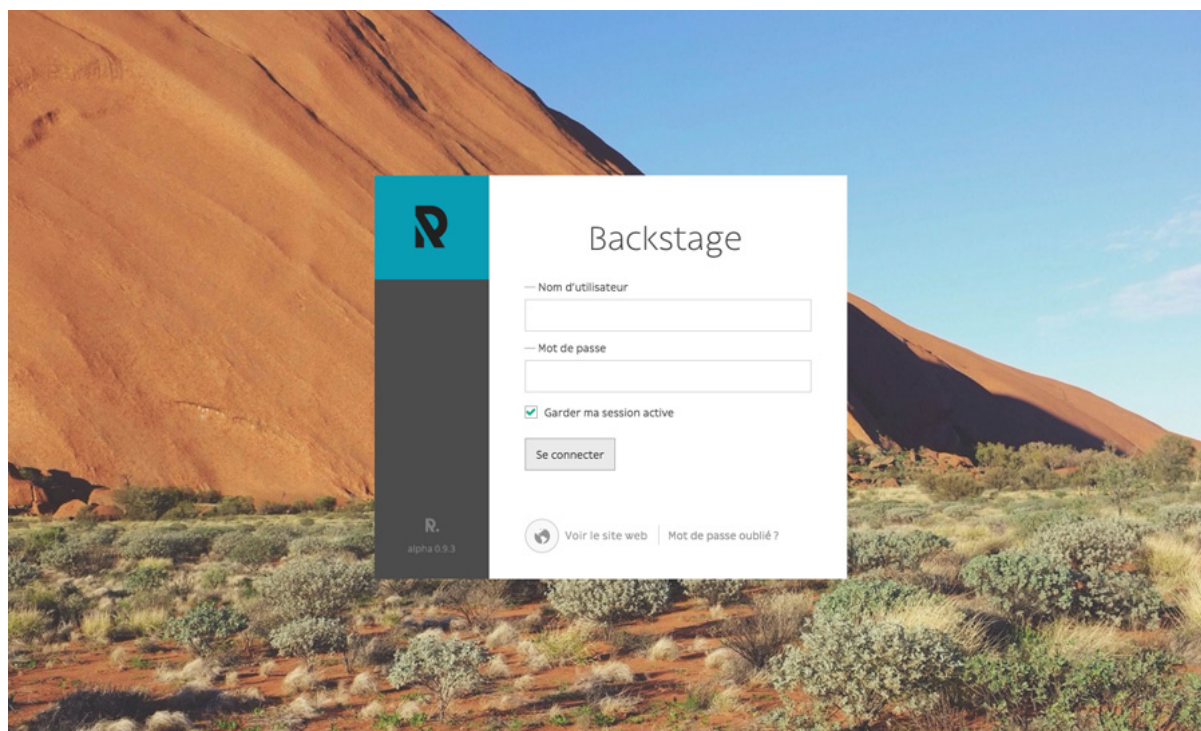
Le système de thème de Roadiz vous permettra de connecter en chaîne des thèmes et des fonctionnalités à partir de plusieurs dépôts de code. Puisque notre CMS est construit sur le conteneur d'injection Pimple, Roadiz peut fusionner tous les thèmes disponibles sur un même site. Par exemple, vous serez 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.

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

Add two hashtag # or more according to your title importance level. Backoffice shortcut buttons allow to directly insert your titles marks before your selected text. Make sure to leave a blank line before each new title you write.

```
## 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

Main title and second level titles can be written using = and - as underline characters.

```
Architecture
=====

Modern architecture
-----
```

Gras

Insert two stars * before and after your text to set in bold. Backoffice shortcut button allows to insert directly the 4 characters around your selected text.

```
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

Insert one star * before and after your text to set in italic. Backoffice shortcut button allows to insert directly the 2 characters around your selected text.

```
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 ?

Bold and italic markup can be performed using `_` (underscore) character too if you actually need to write a *star* character in your text.

```
A 3* Bed & Breakfast has just opened its doors in middletown.
```

Barré

Insert two tildes `~` before and after your text to strike-through.

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

Listes ordonnées et non-ordonnées

Insert a star `*` or a dash `-` followed by a single whitespace for each of your list item. One item per line. Leave a blank line before and after your list. For *ordered* list, use a digit followed by a dot and a whitespace instead.

```
* 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

You can insert a second/third/... level for your list, just by leaving four spaces before your new list-item mark.

```
- 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

A simple line-break is always ignored by Markdown language because it makes a difference between a **paragraph** and a **line-break**. To simply create a line-break without creating a new paragraph, **leave at least 3 spaces at the end of your text line then go to a new line.**

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

To create a new paragraph, **always leave a blank line between your text blocks**. Any additional blank line will be ignored.

```
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

Write link label between braces immediately followed by the URL between parenthesis. For external links do not forget protocol prefix `http://` or `https://`.

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

To create a internal link, just use relative notation :

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

Then, for an email link, use `mailto:` prefix :

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

A link title can be added by inserting it before *ending parenthesis* wrapped in quotes.

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

Citations

Insert a > sign before each new paragraph and a space to wrap your text in a quote block. You can then use all other Markdown symbols inside your quote.

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

Images

Images use the link syntax with an exclamation mark prefix !. For external images do not forget to write full URL with protocol `http://` or `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

Footnotes are not supported with *basic* Markdown syntax, but the *Markdown Extra* one. So before using them, **make sure your webdesigner used the right Markdown parser in your theme.**

```
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.

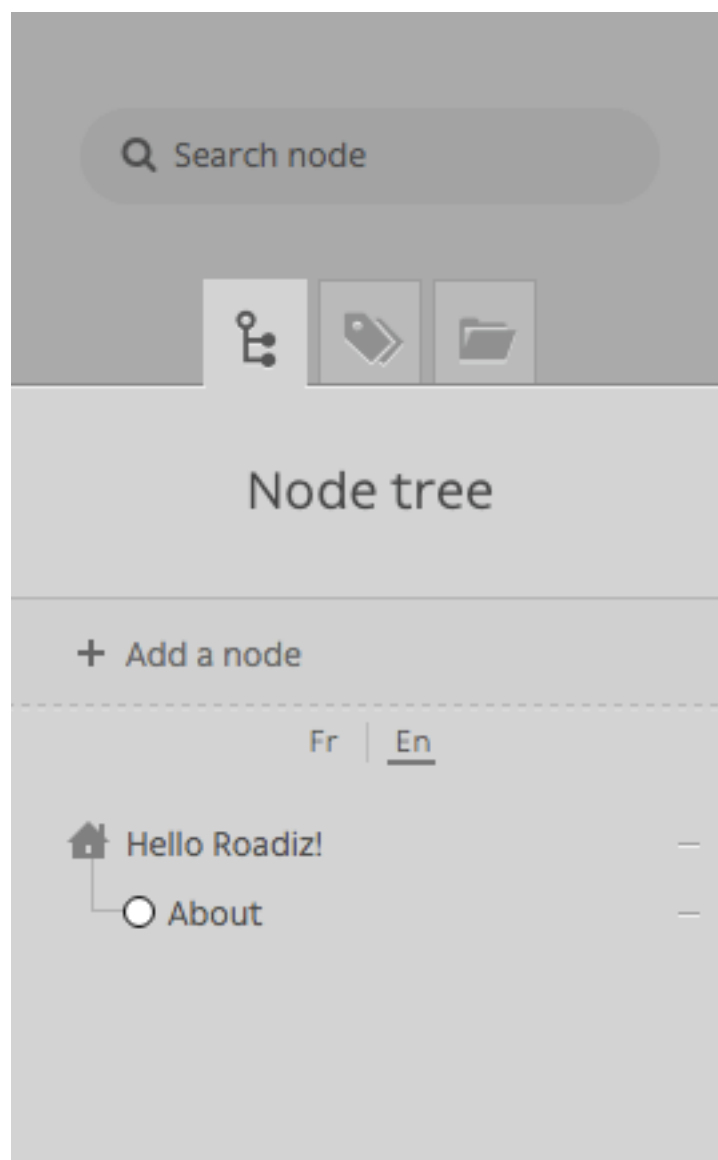
2.1.2 Gérer les nœuds

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.

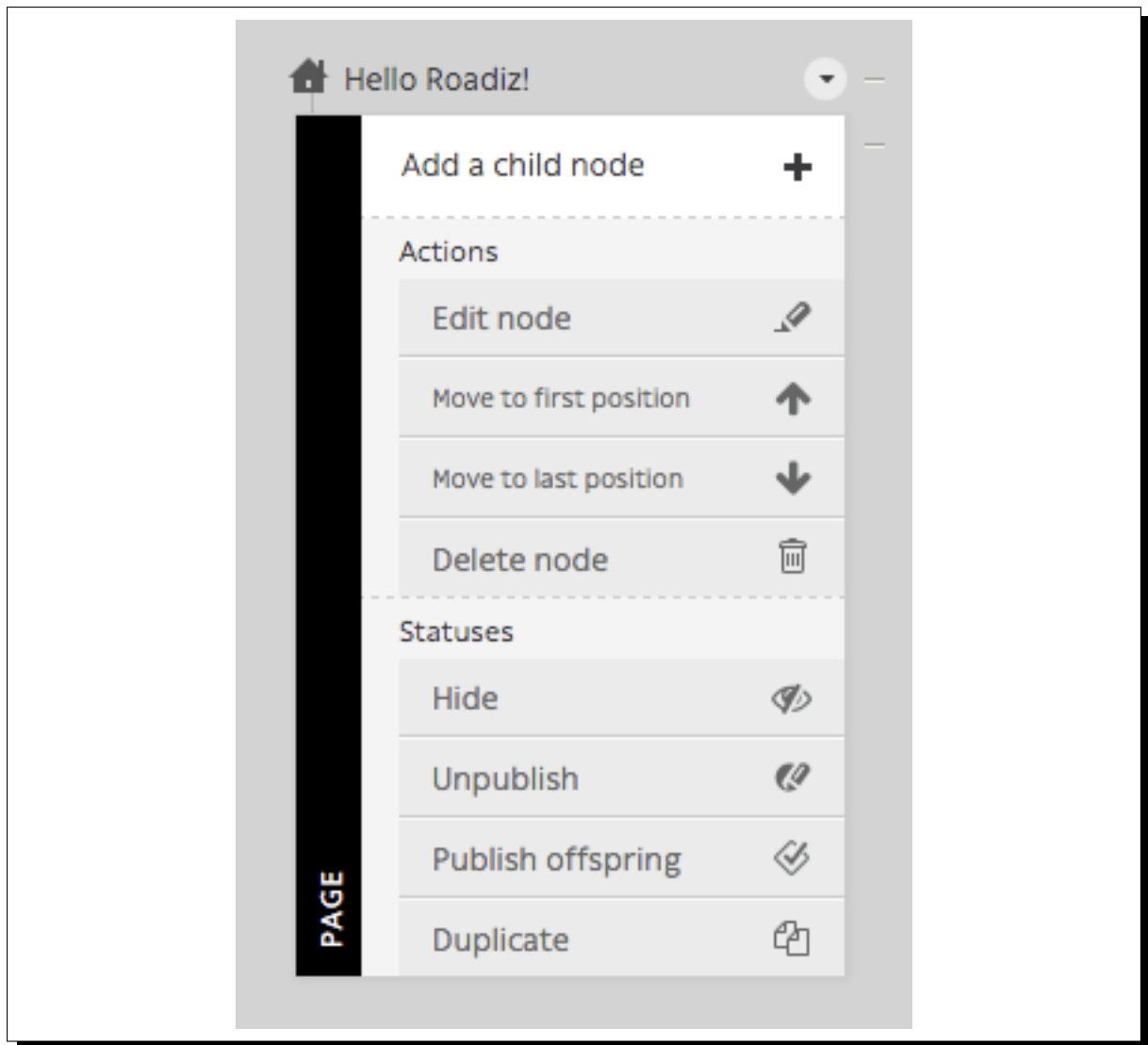
Navigation dans l'arborescence

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.

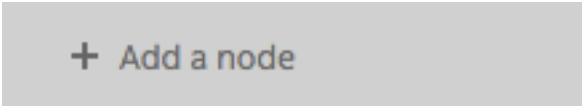
Actions des menus contextuels

- *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.

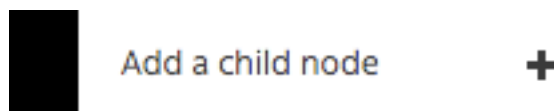
- *Publier/Dépublier* : Modifier le statut de publication d'un nœud. Les nœuds non publiés ne sont pas visibles par les visiteurs anonymes, mais sont accessibles aux utilisateurs du back-office utilisant le chemin d'accès `preview.php`.
- *Publier la descendance* : Publier un nœud et tous ses nœuds-enfants rattachés.
- *Dupliquer* : Copier l'intégralité du contenu et des interactions du nœud actuel dans un nouveau nœud.

Création d'un nœud

Add buttons

A screenshot of a light gray rectangular button with a dark gray plus sign icon on the left and the text 'Add a node' in a dark gray sans-serif font.

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.

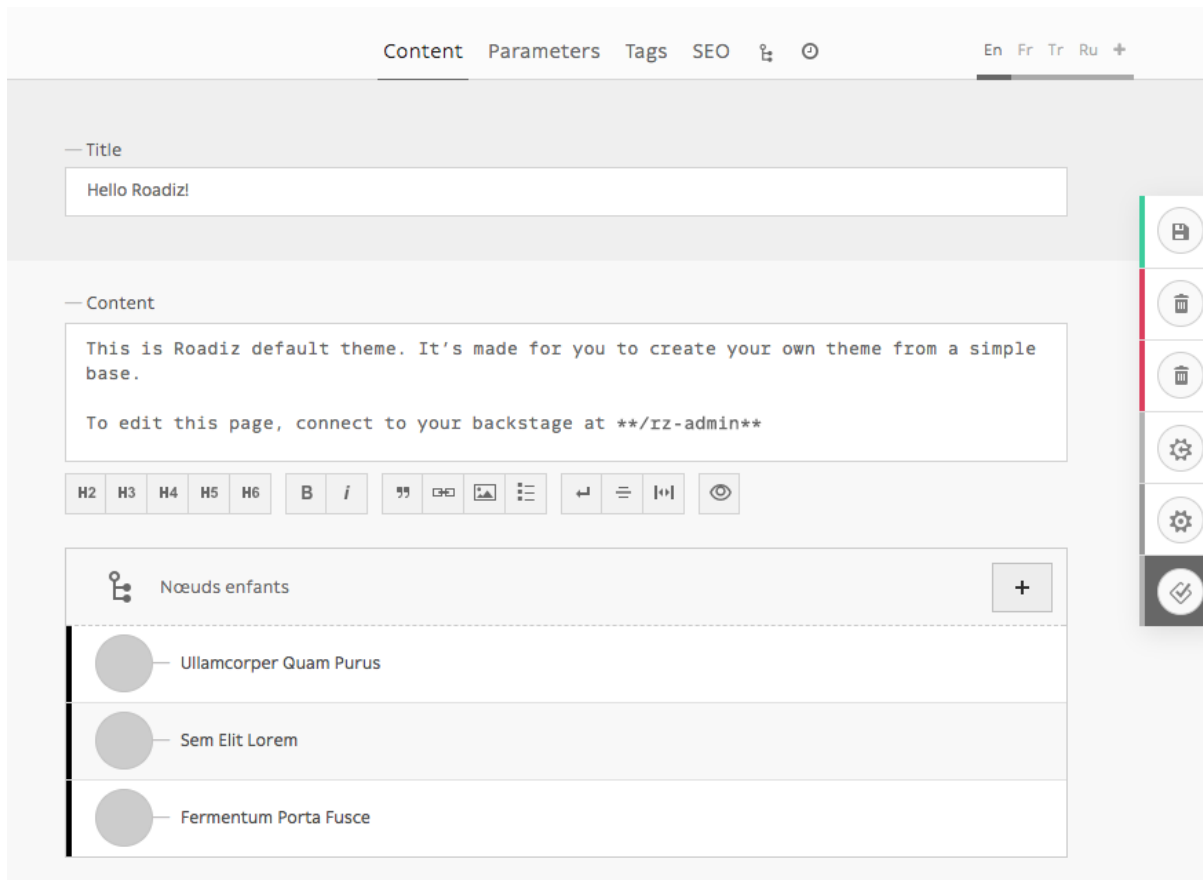
- Le *Nom du nœud* est l'identifiant global de votre nœud. Il doit être unique et ne changera pas d'une traduction à un autre. Il peut être modifié ultérieurement, sauf si votre développeur a verrouillé sa modification. Le *Nom du nœud* est utilisé pour construire les URL de vos pages, de manière générale.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

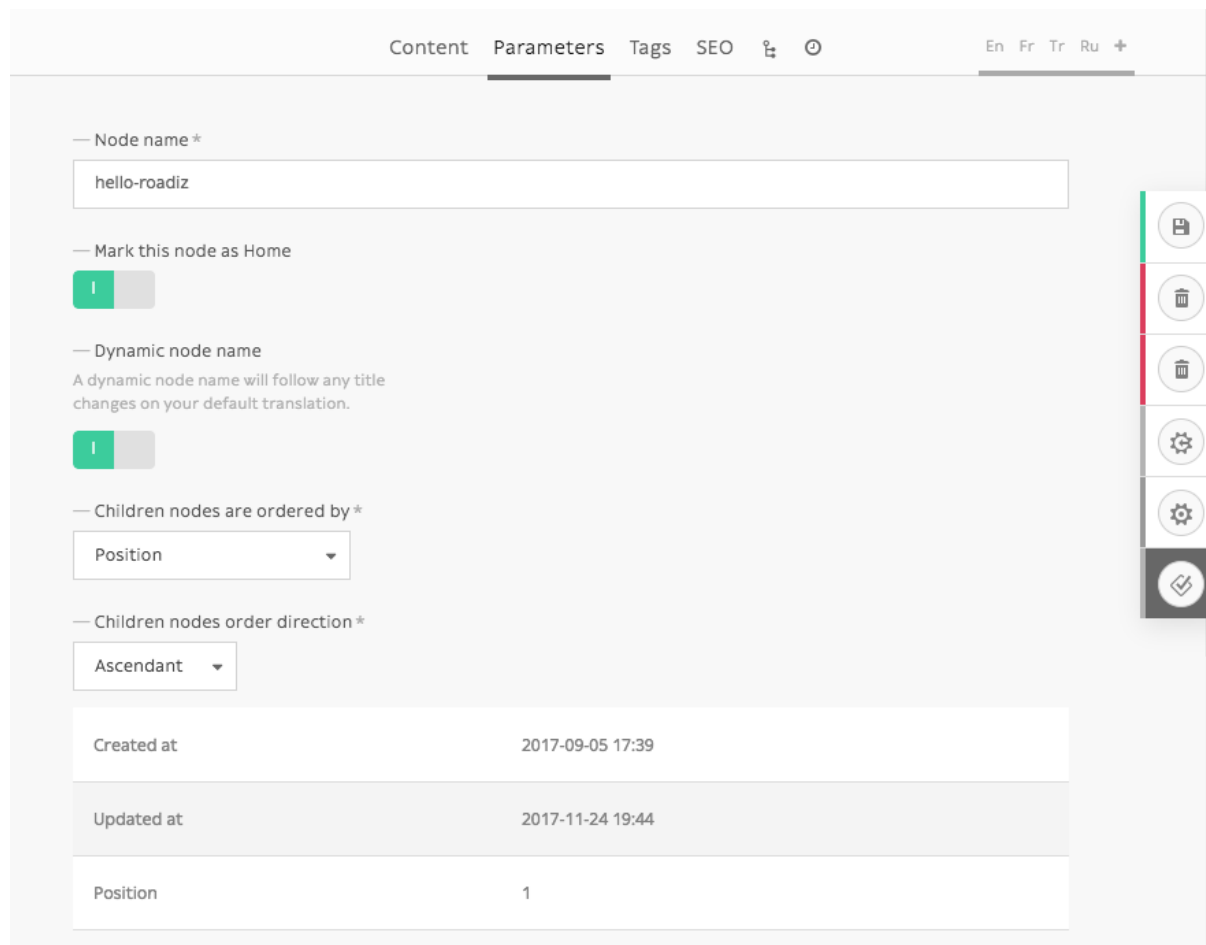

Contenu d'un nœud



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

Content Parameters Tags SEO   En Fr Tr Ru +

— Node name *

hello-roadiz

— Mark this node as Home

☒

— Dynamic node name

A dynamic node name will follow any title changes on your default translation.

☒

— Children nodes are ordered by *

Position

— Children nodes order direction *

Ascendant

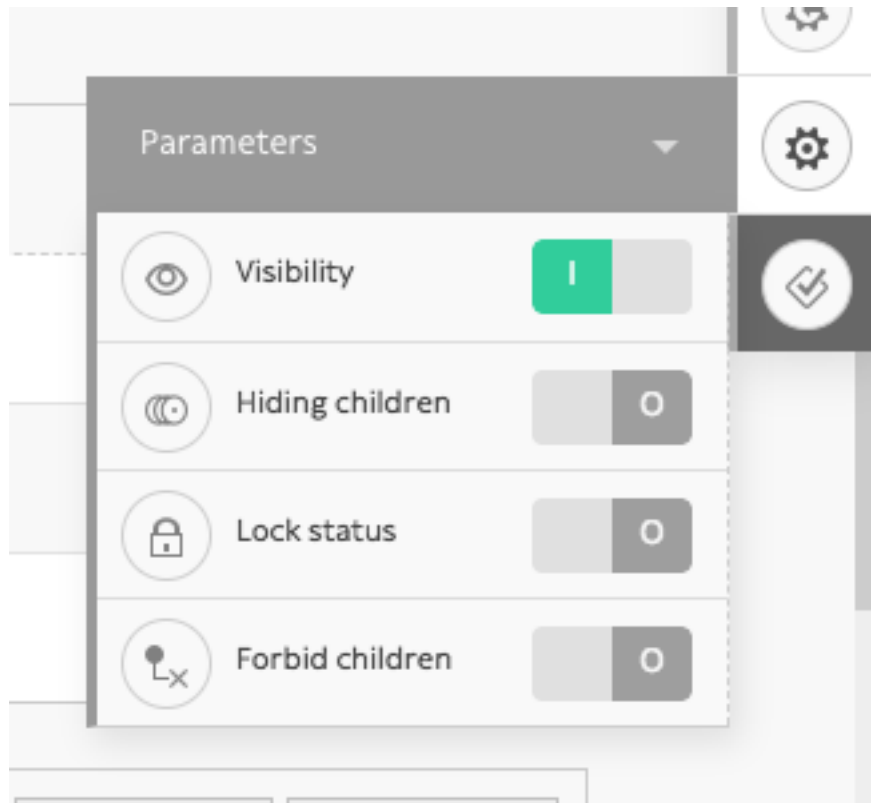
Created at	2017-09-05 17:39
Updated at	2017-11-24 19:44
Position	1

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.

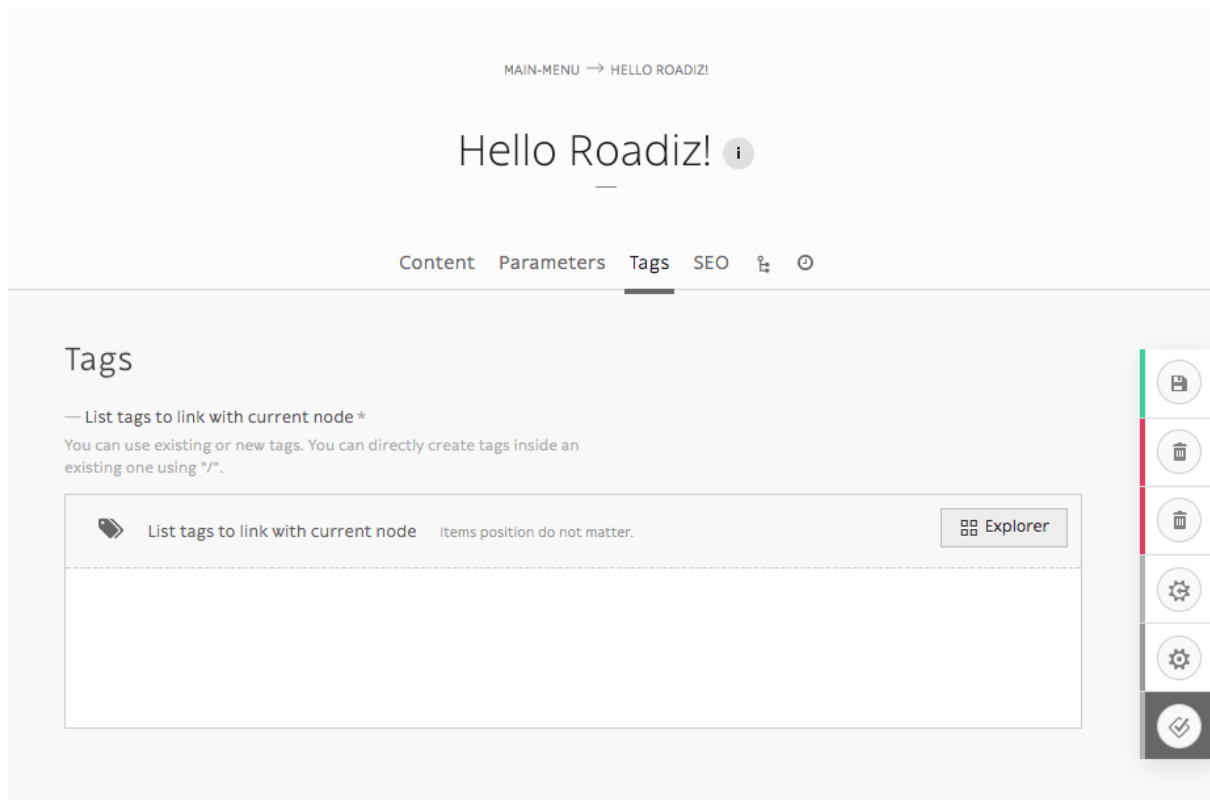
Actions et paramètres secondaires

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'apparaîtront 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

Content Parameters Tags **SEO**

En Fr Tr Ru +

Edit SEO data

— Meta-title
Hello Roadiz!

— Meta-keywords

— Meta-description

Url alias

Add an url-alias

— Url alias *

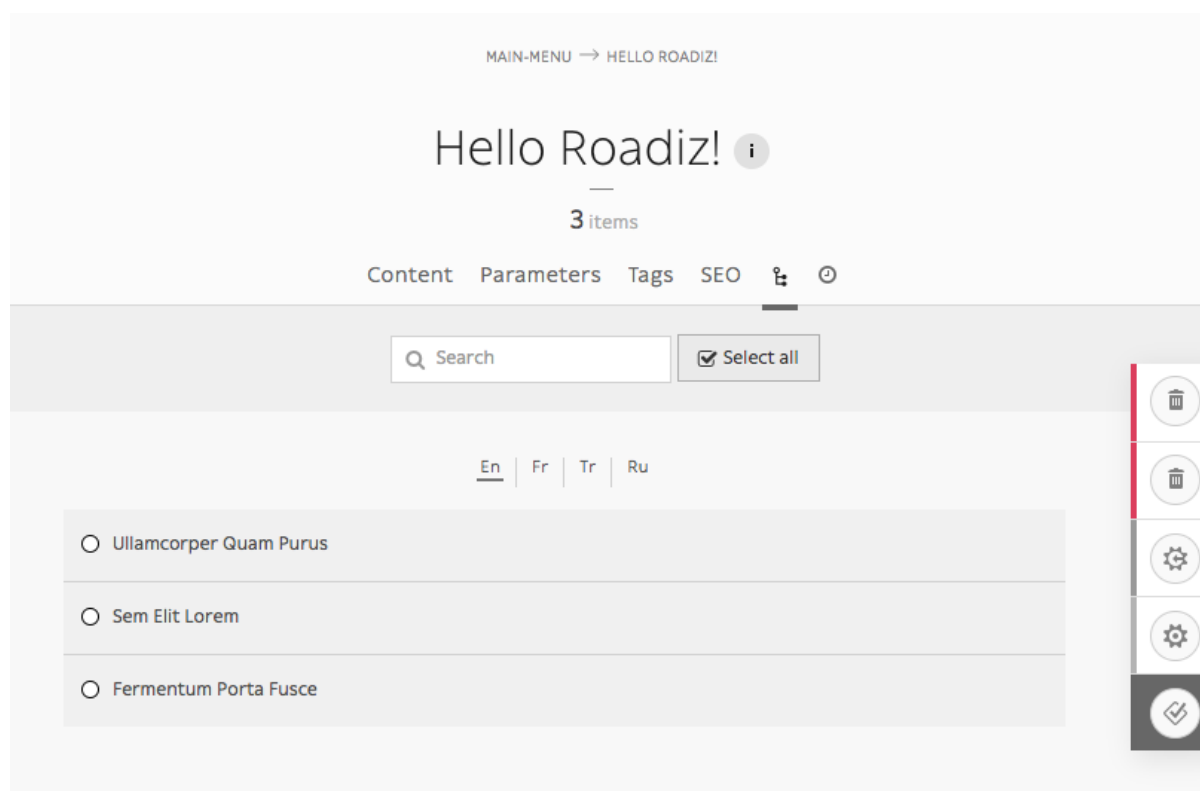
— Translation *
English

+ Add url-alias

Cet onglet affichera les contenus en fonction de chaque traduction.

L'arborescence

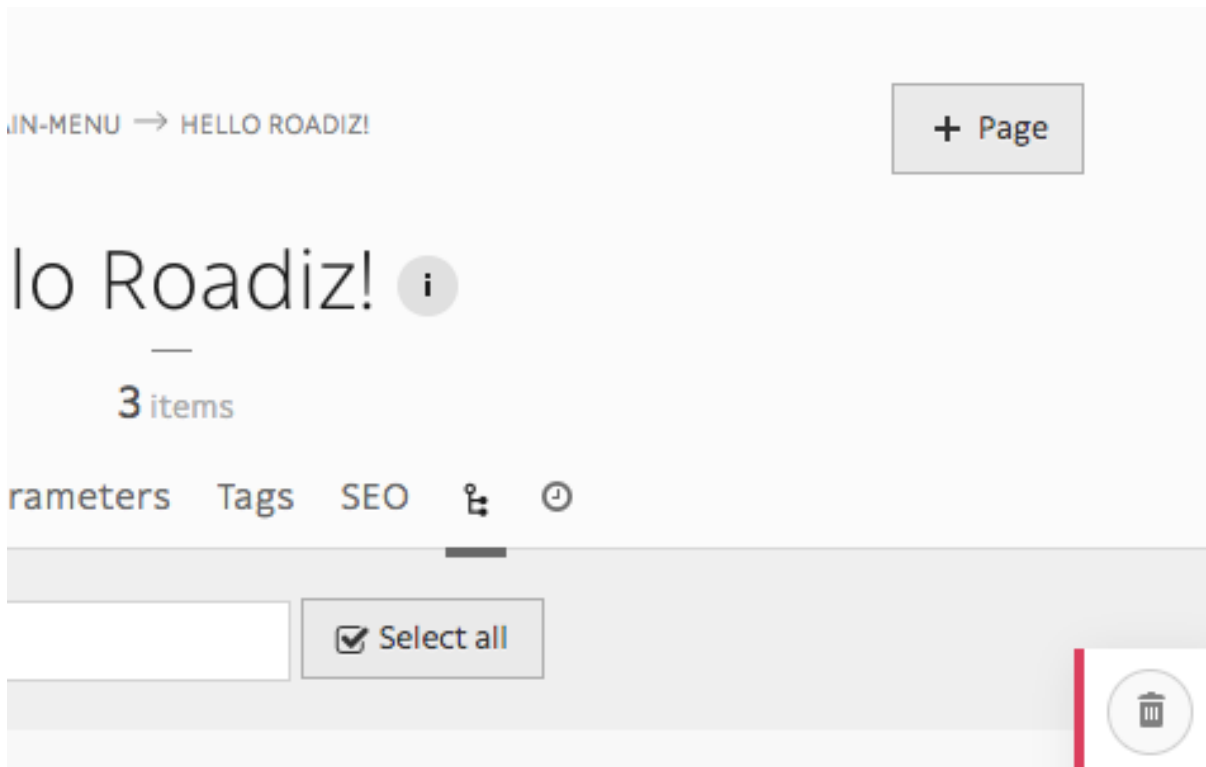
Quand un nœud est défini en tant que *conteneur* (ou *pile*), son *arborescence* devient la vue (onglet) par défaut. Vous pouvez définir l'ordre par défaut d'affichage des nœuds enfants dans l'onglet *Paramètres*. De plus, si vous avez défini le *type de nœud* comme étant *publiable*, chaque nœud affichera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

Puisqu'un nœud non-publié n'est pas visible par vos visiteurs anonymes, Roadiz permet aux utilisateurs du back-office de les prévisualiser en utilisant un *chemin d'accès* appelé `preview.php`. Nous avons décidé de créer un chemin d'accès spécifique afin de bien différencier son URL de votre site public et d'éviter ainsi erreurs et confusions si votre site est hébergé par un proxy inverse.

Par exemple, si votre page actualités n'est pas encore publiée, la relier à `http://mywebsite.com/actualites` aboutira à une page d'erreur 404 pour tous les visiteurs anonymes ou connectés (vous compris). Cependant, si vous souhaitez la prévisualiser, vous devrez vous connecter à `http://mywebsite.com/`

`preview.php/actualites`. Cette URL autorisera l'accès aux visiteurs connectés au back-office uniquement, bloquant la page aux internautes anonymes.

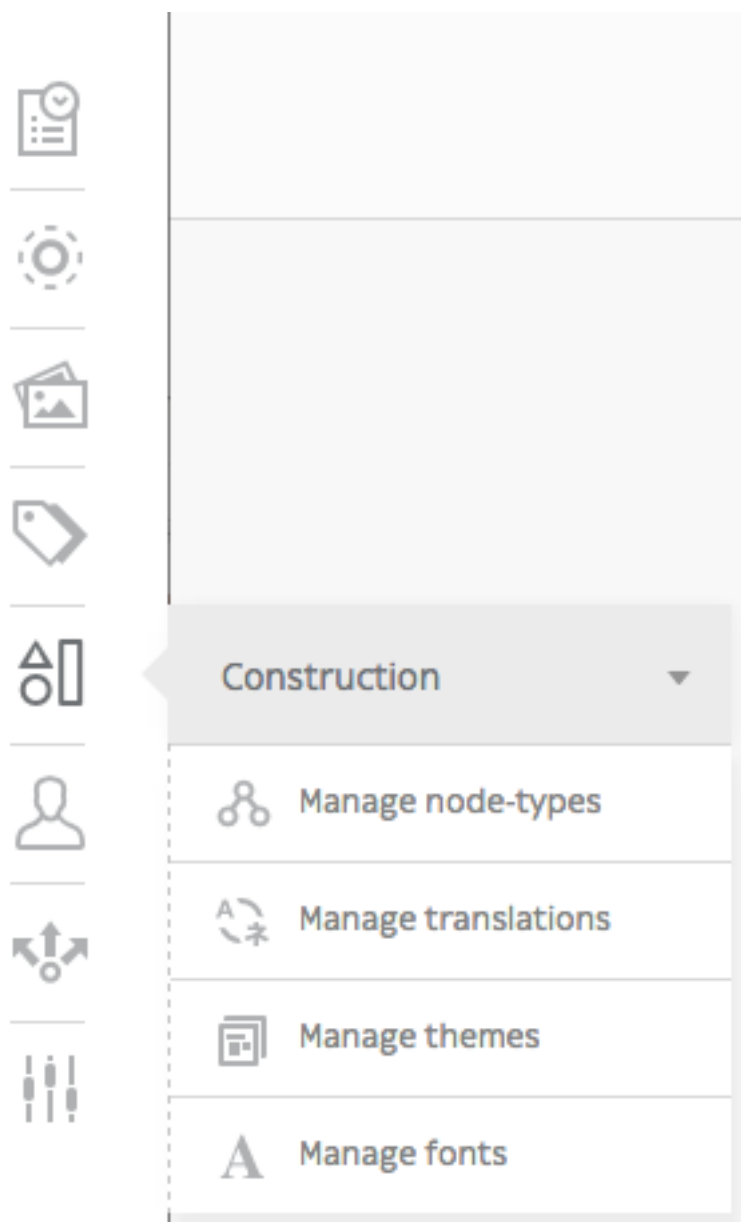
2.1.3 Gérer les types de nœuds

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.

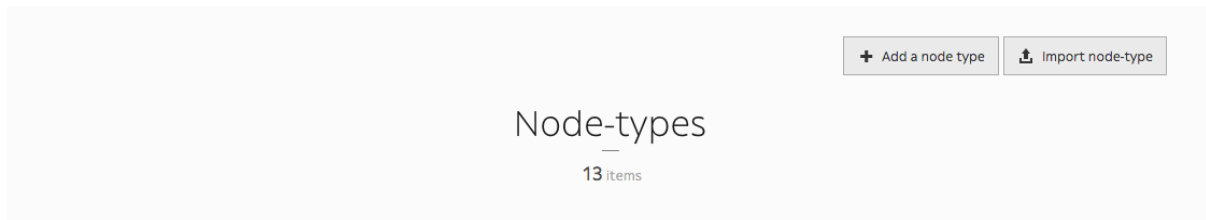
Lorsque vous travaillez dans le back-office de Roadiz, vous pouvez facilement gérer les types de nœuds via l'icône *Construction* de la barre d'outils.



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.

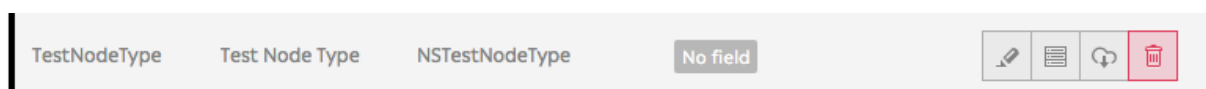


Une fois les deux paramètres obligatoires remplis - *Nom* (que les développeurs utiliseront) et *Nom d'affichage* (que les utilisateurs de back-office verront) -, vous pourrez créer votre premier type de nœud.

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>).

Name	Display name	Node-type entity name	Number of fields	Status	Actions
Page	Page	NSPage	3 fields		

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 ».

Name	Display name	Node-type entity name	Number of fields	Status	Actions
Page	Page	NSPage	3 fields		

Puis cliquez sur le bouton “Ajouter un champ”

Test Node_Type fields

Name	Label	Type	Actions

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 éléments contenant ces valeurs par défaut.

Add a node-type field

— Name

— Label

— Type

Single-line text ▾

— Description

— Group name

— Visible
☒

— Indexed
☐

— Default values

— Min length

— Max length

Autres actions

From one website to another, you can export node-types as .json files.

Name	Display name	Node-type entity name	Number of fields	Status	Actions
Page	Page	NSPage	3 fields		<div>Export</div>

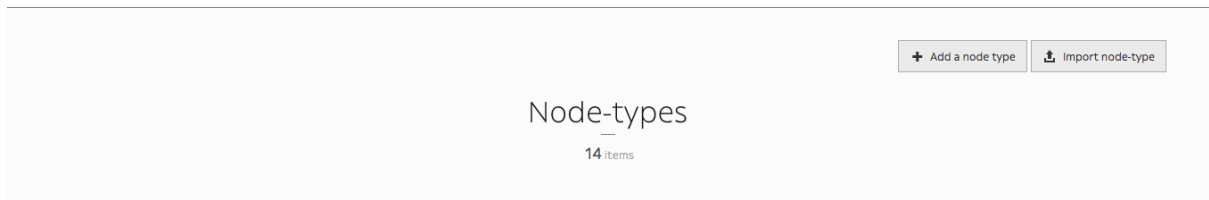
An .json file should look like this when you open it in any editor :

```
{
  "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
}
```

Notice the four fields that have been added to this *Page* node-type.

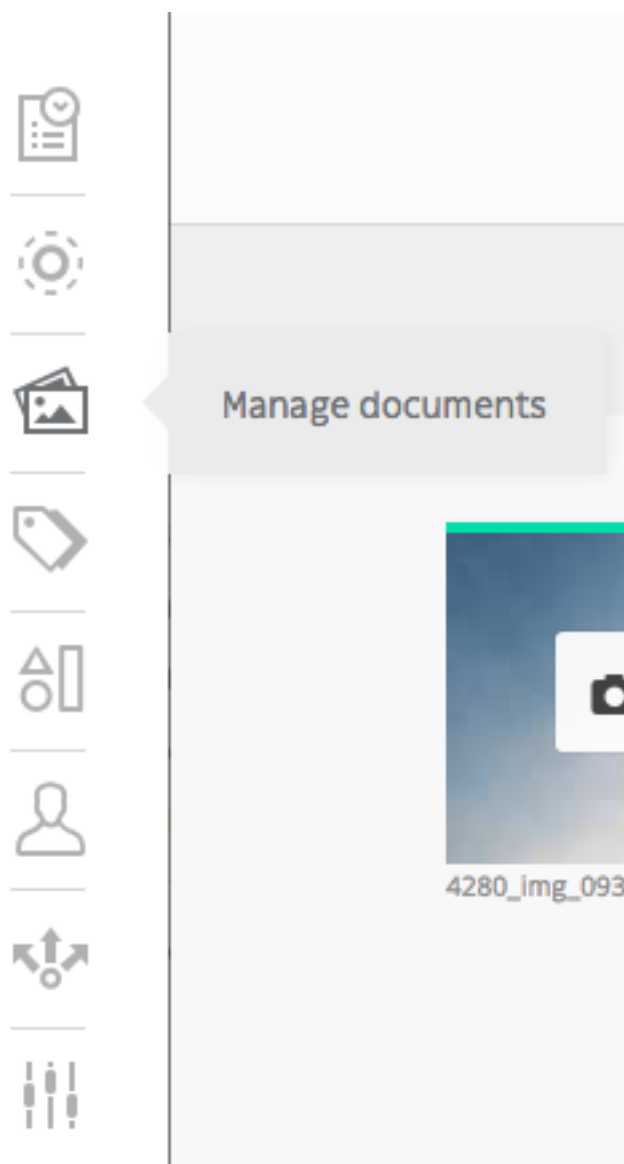
You can write an .json file yourself if you feel like it, but it is probably easier to simply export node-types from existing website, then import it on your new website. It can be easily done by clicking on *Import node-type* button, then selecting it via your explorer.



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.

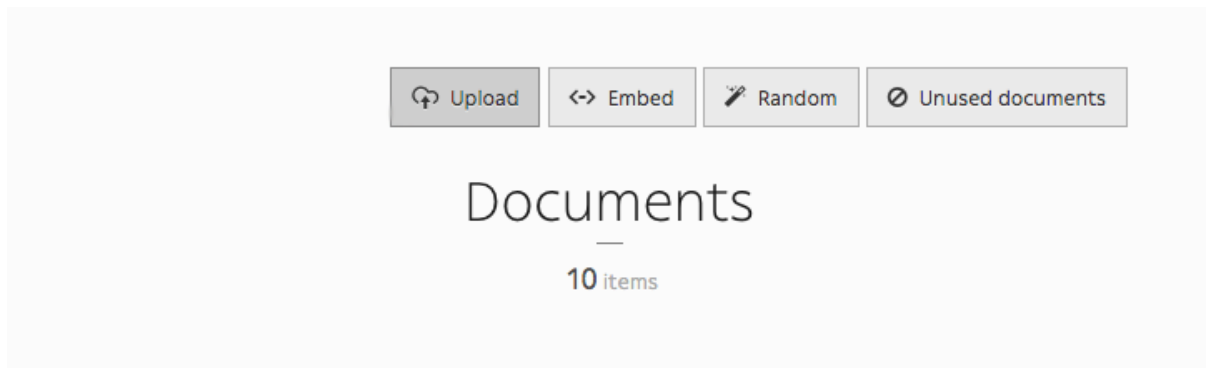
2.1.4 Gérer les documents

You can manage documents via the *Document* icon in the toolbar. You can either upload, embed, randomly downloaded documents, and even have a look at unused documents in your website.



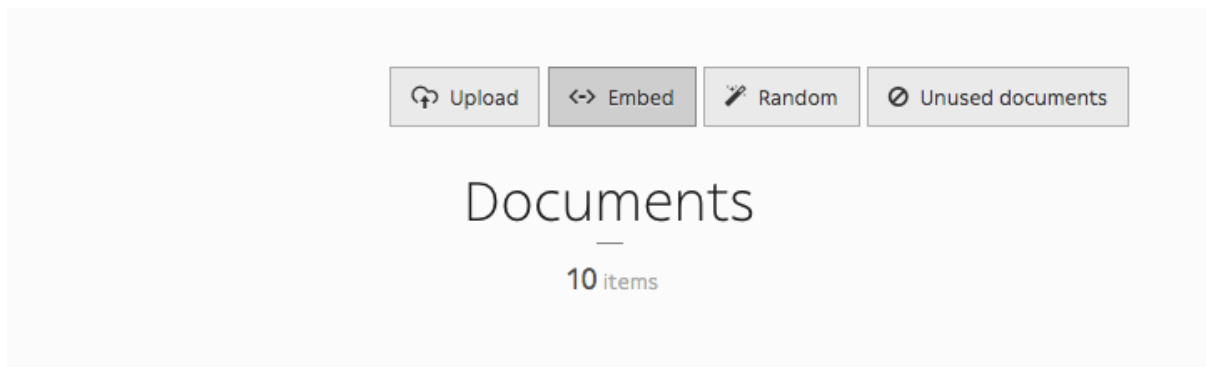
Téléverser un document

Uploading a document has never been this easy : just drag and drop the document in the designated area.



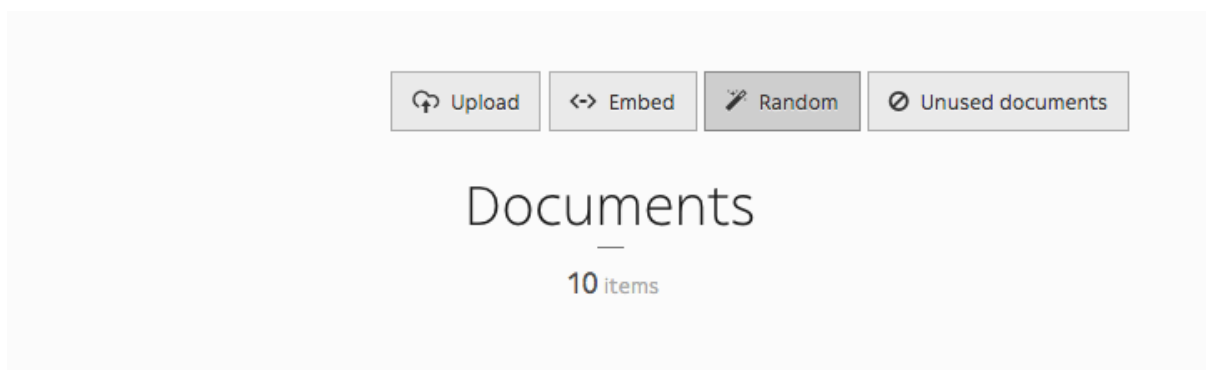
Intégrer un document

Here is the magical part of the Documents section. You can have embedded documents from *YouTube*, *SoundCloud*, *Vimeo*, *Mixcloud*, *Spotify*, *Twitch*, *TED* and *Dailymotion*, then use them as if they were images in your content blocks.



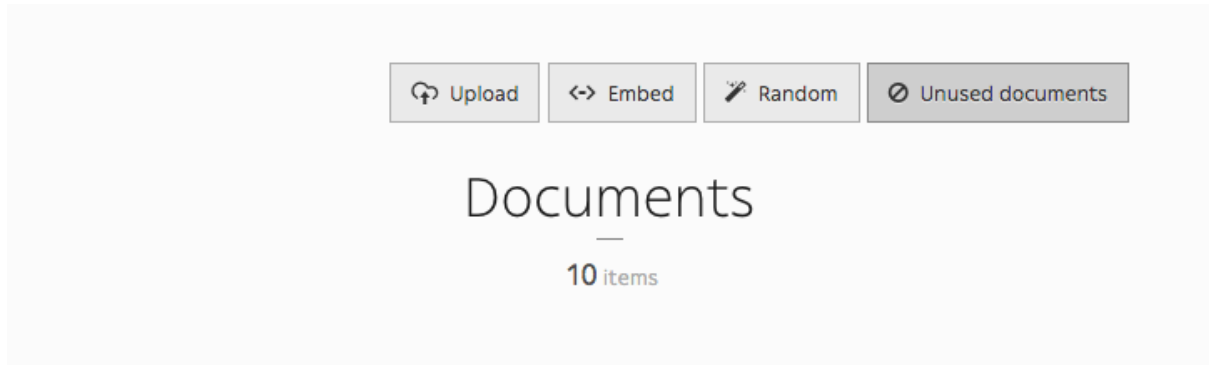
Document aléatoire

Random is a cool feature that allows to download random documents to your website from *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.

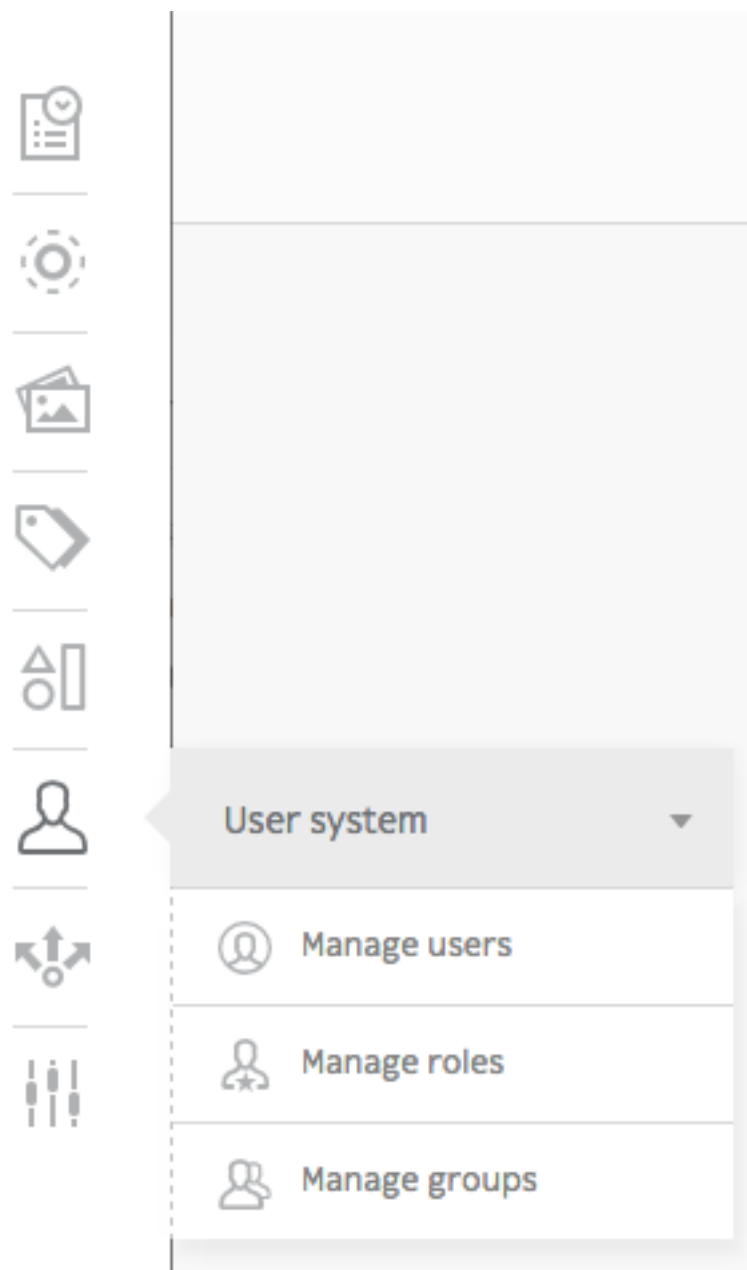


2.1.5 Gérer les utilisateurs

This is a simple guide on how to create and manage users using Roadiz CLI.

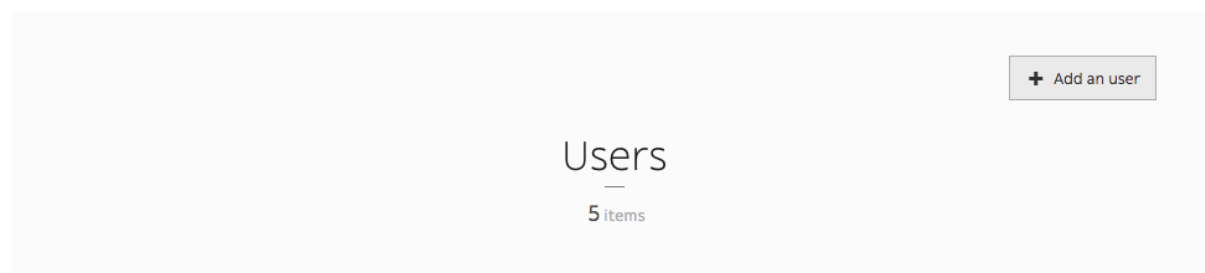
There are two ways of adding users, via the back-office and in command-line, both will be displayed in each section.

When working with Roadiz in the back-office, you can easily manage users via the *User system* icon in the toolbar.



Add user

You can add users simply by clicking *Add an user* button.

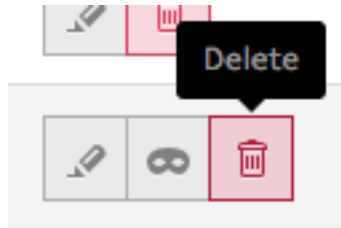


If you create an user without specifying its *password*, an email with a password reset link will sent to your new user. Make sure that you entered the right email and that your Roadiz website has a *mailer* configured. New user will be locked unless he or she reset its password first.

The command-line `bin/roadiz users:create loginname` starts a new interactive user creation session. You will create a new user with login and email, you can also choose if it's a backend user and if it's a superadmin.

Delete user

You can remove users by clicking the trashcan icon.



The command `bin/roadiz users:delete loginname` delete the user « loginname ».

Adding role

You can edit a users profile the same way you edit a node-type. You can add roles in the *Roles* tab.

If you want to add `ROLE_SUPERADMIN` role to « test » user, it would look like this in command-line :

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

Other action

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

```
bin/roadiz list users
```

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 First steps

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 7.2+ 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/MySQL/PostgreSQL or SQLite database (do not forget to install `php-xxsqli` extension according to your database driver flavor)
- 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
- `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 webrowsers
- `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 themes and system themes such as *Rozier* and *Install*
- `vendor/` : Dependencies folder managed by *Composer*

Install Roadiz Standard Edition

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

```
# 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.

Note : For *Windows* users, `bin/roadiz themes:generate --symlink --relative FooBar` command can be used without `--relative` option to create **absolute symlinks**. You can even install your theme assets as *hard copies* without `--symlink` option. Make sure that you call regularly `bin/roadiz themes:assets:install FooBar` when using *hard copy* mode to update your assets. You should use *symlinks* if possible to prevent update issues.

Dealing with Roadiz environments

Installation environment

Once you've succeeded to download Roadiz and its dependencies. You'll have to setup its database and every information needed to begin your website.

As every *Symfony* applications do, *Roadiz* works using environments. By default, there is a *production* environment which is handled by `index.php` entry point. At this moment, if you try to connect to your fresh new Roadiz website, you will get an error as we did not install its database and its essential data.

To be able to use Roadiz *install interface*, you'll need to call the *install* entry point. An `install.php` file has been generated when you executed `composer install` command. This environment will be reachable at the Url `http://mywebsite.com/install.php`.

For security reasons, we added an IP filtering in this entry point, you can add your own IP address in the following array : `array('127.0.0.1', 'fe80::1', '::1', '::ffff:127.0.0.1', '::ffff:127.0.0.1')`. This IP filtering is very important if you are working on a public server, no one except you should be able to access *install* entry point.

At the end of the install process, you will be invited to remove the `install.php` file and to connect to your website final URL.

Development environment

Roadiz *production* environment is not made for developing your own themes and extending back-office features. As the same way as *install* environment, we prepared a *dev* environment to disable resources caching and enable debug features. You'll find a `dev.php` file at your website root which was generated at `composer install`

command. As well as *install.php* entry point, you'll need to add your own IP address to filter who can access to your *dev* environment.

Preview environment

The *preview* environment is not a real one as it only adds a flag to Roadiz' Kernel to enable back-office users to see unpublished nodes. By default, it is available using *preview.php* entry point, unless you decide to remove it.

Production environment

This is the default *index.php* entry point which will be called by all your visitors. There is no restriction on it and it will wake up Roadiz application using the strongest caching policies. So it's not recommended for development usage (you would have to flush caches each time you change something in the code).

Clear cache environment

The *clear_cache* environment is only meant to empty Roadiz cache without waking up the whole application. It can be useful if you are using a op-code cache like *APC* or native PHP *OPcache*. These special caches can't be purged from command line utilities, so you need to call a PHP script from your browser or via *curl* to empty them. Like *install* and *dev* environment, *clear_cache.php* is IP-restricted not to allow everyone to flush your dear caches. You'll need to add your own IP address to filter who can access it.

Using Docker for development

Roadiz standard edition is shipped with a *docker-compose* example environment ready to use for development. *Docker* on Linux will provide awesome performances and a production-like environment without bloating your development machine. Performances won't be as good on *macOS* or *Windows* hosts, but it will prevent installing singled versioned PHP and MySQL directly on your computer.

First, copy *.env.dist* file to *.env* and configure it according to your host machine.

```
# Copy sample environment variables
# and adjust them against your needs.
# Especially APP_PORT when you're working on several projects
cp .env.dist .env;

# Build PHP image
docker-compose build;

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

Then your website will be available at `http://localhost:${APP_PORT}`.

For linux users, where *Docker* is running natively (without underlying virtualization), pay attention that *PHP* is running with *www-data* user. You must update your *.env* file to reflect your local user **UID** during image build.

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

So use the same uid in your *.env* file **before** starting and building your Docker image.

```
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 have 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 *local-host*. 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 suggestions 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.

Manual configuration

This section explains how main configuration file `app/conf/config.yml` works. It is way more more convenient than launching Install theme for each configuration update.

Your `app/conf/config.yml` file is built using YAML syntax. Each part matches a Roadiz *service* configuration.

Note : By default, every Roadiz environment read `app/conf/config.yml` configuration file. But you can specify different files for dev and test environments. Just create a `app/conf/config_dev.yml` or `app/conf/config_test.yml` file to override default parameters. You will be able to use a different database, mailer or *Solr* instance not to pollute your production environment.

Source Edition

Roadiz Source edition stores configuration files in `conf/` folder.

Doctrine

The most important configuration section deals with database connection which is handled by *Doctrine* :

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

Roadiz uses *Doctrine ORM* to store your data. It will directly pass this YAML configuration to *Doctrine* so you can use every available drivers and options from its documentation at <http://doctrine-dbal.readthedocs.org/en/latest/reference/configuration.html>

Themes

Since *Roadiz v1.0*, themes are statically registered into Roadiz configuration for better performances and delaying database usage. You have to register any front-end theme in your `app/conf/config.yml` file. Theme priority is not handled here but in each of your themes by overriding static `$priority` value;

```
themes:
  -
    classname: \Themes\DefaultTheme\DefaultThemeApp
    hostname: '*'
    routePrefix: ''
  -
    classname: \Themes\FooBarTheme\FooBarThemeApp
    hostname: 'foobar.test'
    routePrefix: ''
```

You can define hostname specific themes and add a route-prefix for your routing. Defaults values are `'*'` for the *hostname* and `''` (empty string) for the route-prefix.

Avertissement : No theme configuration will lead into a 404 error on your website home page. But you will still have access to the back-office which is now hard-registered into Roadiz configuration.

Cache drivers

When set as *null*, cache drivers will be automatically chosen by Roadiz according to your PHP setup and available extensions.

Sometimes, if a cache extension is available but you don't want to use it, you'll have to specify a cache driver type (use `array` to disable caches). This is a known case when using *OVH* shared hosting plans which provide *memcached* PHP extension but does not let you log in.

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

Available cache types are :

- *apc*
- *xcache*
- *memcache* (requires host and port configuration)
- *memcached* (requires host and port configuration)
- *redis* (requires host and port configuration)
- *array*

Monolog handlers

By default, Roadiz writes its logs to `app/logs/` folder in a file named after your running environment (eg. `roadiz_prod.log`). But you can also customize *Monolog* to use three different handlers. Pay attention that using custom log handlers will disable default Roadiz logging (except for *Doctrine* one) so it could be better to always use *default* handler along a custom one.

Available handler types :

- `default` : Reproduce the Roadiz default handler which writes to `app/logs/` folder in a file named after your running environment
- `stream` : Defines a log file stream on your local system. **Your path must be writable!**
- `syslog` : Writes to system *syslog*.
- `gelf` : Send GELF formatted messages to an external entry point defined by *url* value. Roadiz uses a fault tolerant handler which **won't trigger any error** if your path is not reachable, so make sure it's correct. It's a good idea to combine a *gelf* handler with a local logging system if your external entry point is down.
- `sentry` : Send logs to your *Sentry* instance. **Requires sentry/sentry PHP library** : `composer require sentry/sentry php-http/curl-client guzzlehttp/psr7`. It's a good idea to combine a *sentry* handler with a local logging system if your external entry point is down.

`type` and `level` values are mandatory for each handlers.

Here is an example configuration :

```
monolog:
  handlers:
    default:
      type: default
      level: INFO
    file:
      type: stream
      # Be careful path must be writable by PHP
      path: /var/log/roadiz.log
      level: INFO
    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
```

Solr endpoint

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

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

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

Reverse proxy cache invalidation

Roadiz can request cache invalidation to external and internal cache proxies such as internal *Symfony* AppCache or a *Varnish* instance. If configured, Roadiz will create a BAN request to each configured proxy **when user clears back-office caches**, and it will create a PURGE request **on each node-source update event** using first reachable node-source URL.

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

Note : Make sure you [configured your external reverse proxy](#) in order to receive and handle BAN and PURGE HTTP requests.

Entities paths

Roadiz uses *Doctrine* to map object entities to database tables. In order to make Roadiz more extensible, you can add your own paths to the entities part.

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

Configure mailer

Roadiz uses *Swift Mailer* to send emails. This awesome library is built to enable different kinds of mail transports and protocols. By default, Roadiz uses your PHP `sendmail` configuration but you can tell it to use another transport (such as an external SMTP server) in your `app/conf/config.yml` file.

You can use *SSL*, *TLS* or no encryption at all.

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

Note : Pay attention that many external SMTP services (*Mandrill*, *Mailjet*...) only accept email from validated domains. So make sure that your application uses a known `From:` email sender not to be blacklisted or blocked by these services. If you need your emails to be replied to an anonymous address, use `ReplyTo:` header instead.

Images processing

Roadiz use *Image Intervention* library to automatically create a lower quality version of your image if they are too big. You can define this threshold value in the `assetsProcessing` section. `driver` and `defaultQuality` will be also use for the on-the-fly image processing with *Intervention Request* library.

```
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
  subscribers: []
```

Additional *Intervention Request* subscribers

Any *Intervention Request* subscriber can be added to configuration with its `classname` and its constructor arguments. Here is an example with `WatermarkListener` which will print some text on all your images.

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

Use kraken.io to reduce drastically image sizes

Since you can add *Intervention Request* subscribers, we created a useful one that sends every images to kraken.io services to shrink them. Once you've configured it, do not forget to empty your caches **and** image caches to see changes.

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

Avertissement : Take note that each generated image is sent to *kraken.io* servers. It can generate some overhead time on the first time you request an image.

Console commands

Roadiz can be executed as a simple CLI tool using your SSH connection. This is useful to handle basic administration tasks with no need of backoffice administration.

```
./bin/roadiz
```

If your system is not configured to have *php* located in `/usr/bin/php` use it this way :

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

Default command with no arguments will show you the available commands list. Each command has its own parameters. You can use the argument `--help` to get more informations about each tool :

```
./bin/roadiz install --help
```

We even made *Doctrine* CLI tools directly available from Roadiz Console. Be careful, these are powerful commands which can alter your database and make you lose precious data. Especially when you will need to update your database schema after a Theme or a Core update. **Always make a database back-up before any Doctrine operation.**

Additional commands

If you are developing your own theme, you might need to create some custom CLI commands. Roadiz can handle additional commands if you add them in your `app/conf/config.yml` as you would do for any additional *entities*. Make sure that every additional commands extend `Symfony\\Component\\Console\\Command\\Command` class.

```
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 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 edition*, make sure that your Roadiz *version constraint* is set in your project `composer.json` file, then :

```
composer update -n --no-dev;
```

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

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

Then run database schema update, first review migration details to see if no data will be removed :

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

Then, if migration summary is OK (no data loss), perform the following changes :

```
bin/roadiz orm:schema-tool:update --force;
# 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 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. 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 you 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](#).

Install Standard Edition on shared hosting

... without SSH commands and FTP .

Modern CMS built on *Composer* and *CLI* commands can't be deployed easily on shared hosting environments on which only FTP is available. Here are some handy tools to deploy a *Roadiz* with FTP.

The first condition is that you'll have to setup a local environment which will be mirrored to your shared hosting FTP, `vendor/` included... yes. Grab a very long cup of coffee when you initiate the first FTP push, it will be long, very long. Next pushes will only push newer files.

The second condition is that you must create all your node-type entities on your local env first to be able to mirror all `GeneratedNodeSources*` classes as you won't be able to generate them on your production env.

Prepare your local env with Makefile

Standard-edition comes with a sample Makefile so you can write a `push-prod` recipe to automatize all process using `lftp` command. Make sure it's installed on your computer :

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

This recipe will clear your cache files, generate Apache `.htaccess` files, copy your theme assets as real files, mirror all necessary files without all exceptions (list can be improved) then copy your theme assets back to symlinks.

```
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}
```

Make sure your configuration matches your shared hosting plan, for example, adjust your cache driver to `file`, `php` to get decent performances.

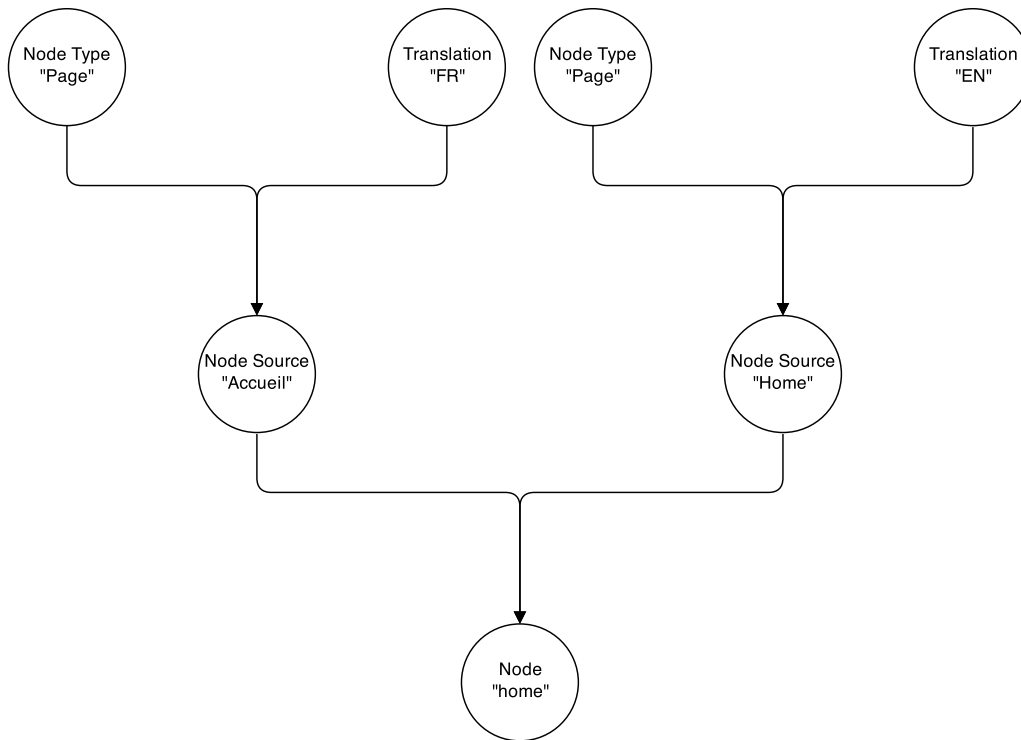
```
cacheDriver:
    type: file
    host: null
    port: null
```

3.1.2 Node system

Node-types, nodes-sources and nodes

This part is the most important part of Roadiz. Quite everything in your website will be a node.

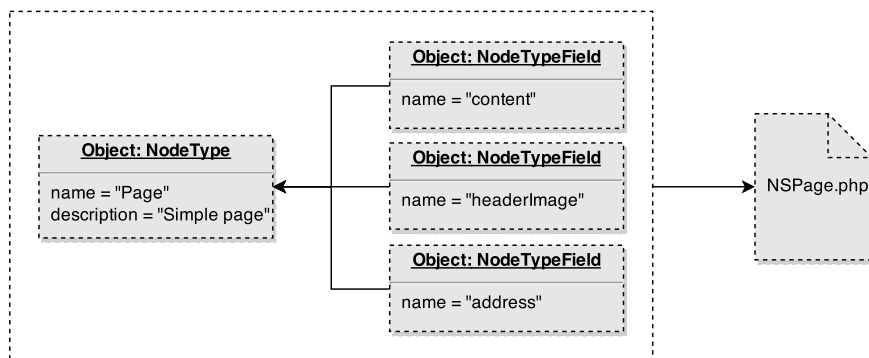
Let's check this simple node schema before explain it.



Now, it's time to explain how it's working !

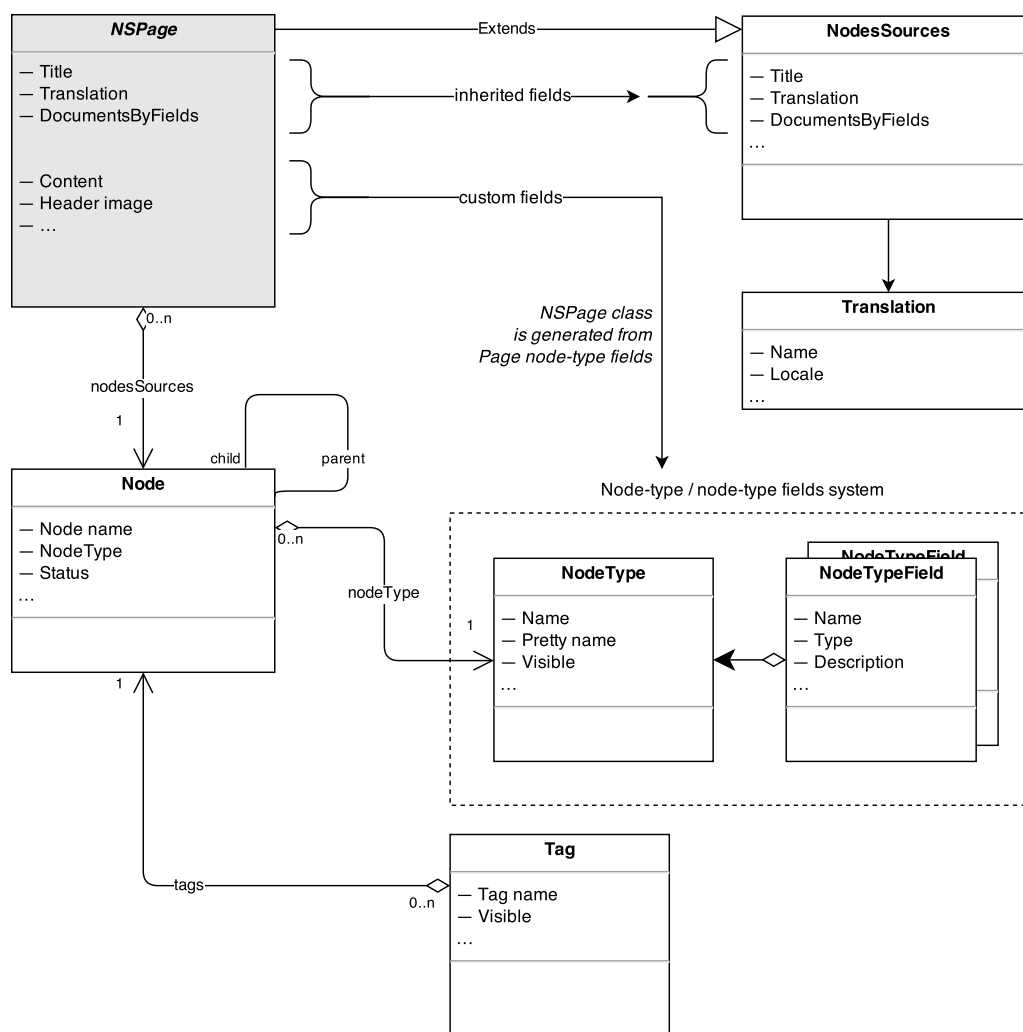
What is a Node-type

A node-type is the blueprint for your node-source. It will contain all fields that Roadiz will use to generate an extended node-source class.



For example, a node-type « Page » will contain « content » and « header image » fields. The « title » field is always available as it is hard-coded in `NodesSources` class. After saving your node-type, Roadiz generates a `NSPage` class which extends the `NodesSources` class. You will find it in the `gen-src/GeneratedNodeSources` (or `app/gen-src/GeneratedNodeSources` with *Roadiz Standard edition*). Then Roadiz calls *Doctrine* update tool to migrate your database schema. **Do not modify the generated class.** You'll have to update it by the backend interface.

Here is a schema to understand how node-types can define custom fields into node-sources :



Most of node-types management will be done in your backoffice interface. You will be able to create, update node-types objects and each of their node-type fields independently. But if you prefer, you can use CLI commands to create types and fields. With Roadiz CLI commands you get several tools to manage node-types. We really encourage you to check the commands with `--help` argument, as following :

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

Keep in mind that each node-type or node-type fields operation require a database update as Doctrine have to create a specific table per node-type. Do not forget to execute `bin/roadiz orm:schema-tool:update` tools to perform updates. It's very important to understand that *Doctrine* needs to see your node-types generated classes **before** upgrading database schema. If they don't exist, it won't able to create your custom types tables, or worst, it could delete existing data since *Doctrine* won't recognize specific tables.

Now let's have a look on node-sources.

Node-sources and translations

Once your node-type created, its definition is stored in database in `node_types` and `node_type_fields` tables. This informations will be only used to build your node-sources edition forms in backoffice and to build a custom database table.

Inheritance mapping

With Roadiz, each node-types data (called node-sources) is stored in a different table prefixed with `ns_`. When you create a *Page* node-type with 2 fields (*content* and *excerpt*), Roadiz tells Doctrine to build a `ns_page` table with 2 columns and one primary key column inherited from `nodes_sources` table. It's called *inheritance mapping* : your `ns_page` table extends `nodes_sources` table and when you are querying a *Page* from database, Doctrine mix the data coming from these 2 tables to create a complete node-source.

At the end your node-source *Page* won't contain only 2 fields but many more as `NodesSources` entity offers `title`, `metaTitle`, `metaDescription`, `metaKeywords` and others useful data-fields which can be used among all node-types.

Translations

Node-sources inheritance mapping is not only used to customize data but to make data translations available. As you saw in the first picture, each nodes can handle many node-sources, one per translation.

Node-type fields

Roadiz can handle many types of node-type fields. Here is a complete list :

Note : *Title*, *meta-title*, *meta-description* and *keywords* are always available since they are stored directly inside `NodesSources` entity. Then you will be sure to always have a *title* no matter the node-type you are using.

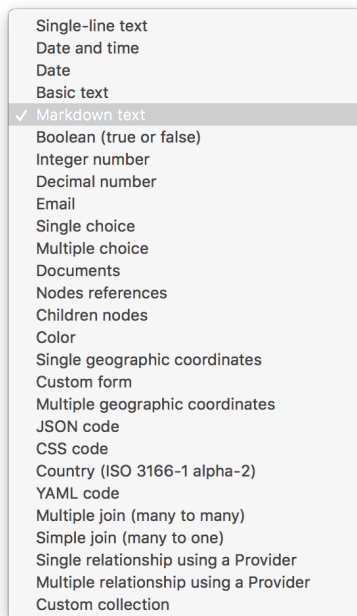
Simple data

This following fields stores simple data in your custom node-source database table.

- Single-line text
- Date
- Date and time
- Basic text
- Markdown text
- Boolean
- Integer number
- Decimal number
- Email
- Color
- Single geographic coordinates
- JSON code
- CSS code
- Country code (ISO 3166-1 alpha-2)
- YAML code
- Many to many join
- Many to one join
- Single relationship using a provider
- Multiple relationship using a provider
- Custom collection

Note : *Single geographic coordinates* field stores its data in JSON format. Make sure you don't have manually written data in its input field.

Avertissement : To use *Single geographic coordinates* you must create a *Google API Console* account with *Maps API v3* activated. Then, create a *Browser key* and paste it in “Google Client ID” parameter in Roadiz settings to enable *geographic* node-type fields. If you didn’t do it, a simple text input will be display instead of *Roadiz Map Widget*.



Virtual data

Virtual types do not really store data in node-source table. They display custom widgets in your editing page to link documents, nodes or custom-forms with your node-source.

- Documents
- Nodes references
- Custom form

Complex data

These fields types must be created with *default values* (comma separated) in order to display available default choices for “select-box” types :

- Single choice
- Multiple choices
- Children nodes

Children node field type is a special virtual field that will display a custom node-tree inside your editing page. You can add *quick-create* buttons by listing your node-types names in *default values* input, comma separated.

Universal fields

If you need a field to hold exactly the same data for all translations, you can set it as *universal*. For example for documents, numeric and boolean data that do not change from one language to another.

It will duplicate data at each save time from default translation to others. It will also hide the edit field from non-default translation to avoid confusion.

YAML field

When you use YAML field type, you get an additional method to return your code already parsed. If your field is named `data`, your methods will be generated in your `NSEntity` as `getData()` and `getDataAsObject()`.

- `getData()` method will return your YAML code as *string*.
- `getDataAsObject()` will return a mixed data,array or `stdObject` according to your code formatting. This method will throw a `\Symfony\Component\Yaml\Exception\ParseException` if your YAML code is not valid.

Many to many and Many to one joins

You can create custom relations between your node-source and whatever Doctrine entities in *Roadiz* or in your theme.

You must fill the *default values* field for these two 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
```

Single and multiple provider

The generic provider type allow you to fetch every data you want through a `Provider` class in your theme. This can be really useful if you need to fetch items from an external API and to reference them in your nodes-sources.

Imagine that you want to link your page with an *Instagram* post. You'll have to create a class that extends `Themes\Rozier\Explorer\AbstractExplorerProvider` and configure it in your field :

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

This provider will implement `getItems`, `getItemsById` and other methods from `ExplorerProviderInterface` in order to be able to display your *Instagram* posts in Roadiz explorer widget and to find your selected items back. Each *Instagram* post will be wrapped in a `Themes\Rozier\Explorer\AbstractExplorerItem` that will map your custom data to the right fields to be showed in Roadiz back-office.

You'll find an implementation example in Roadiz with `Themes\Rozier\Explorer\SettingsProvider` and `Themes\Rozier\Explorer\SettingExplorerItem`. These classes do not fetch data from an API but from your database using `EntityManager`.

Single and multiple provider types can accept additional options too. If you want to make your provider configurable at runtime you can pass options in your field configuration.

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

Then you must override your provider' `configureOptions` method to add which options are allowed.

```
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');
}
```

Custom collection

Last but not least, you can create a custom collection field to store read-only data using a dedicated *Symfony* `AbstractType`.

You must fill the *default values* field for this type :

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

You must understand that *custom collection* data will be stored as JSON array in your database. So you won't be able to query your node-source using this data.

In your `FooBarType`, you'll be able to use *Symfony* standard fields types and **Roadiz** non-virtual fields too such as `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('em')->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 hydration. So, make sure you filter your queries by one `NodeType` as much as possible with `nodeSourceApi` and `node.nodeType` criteria.

```
// Here Doctrine will only join NSPage table to NodesSources
$children = $this->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 parameters that you must take care of in your themes and your controllers, because they are not mandatory in all website cases :

- Visibility
- Publication date and time

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

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

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

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

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

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

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*.

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

        // Generate a path for current node-source
        $path = $this->generateUrl($this->nodeSource);

        // Generate an absolute URL for current node-source
        $absoluteUrl = $this->generateUrl(
            $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.

```
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();
            }
        }
    }
}

```

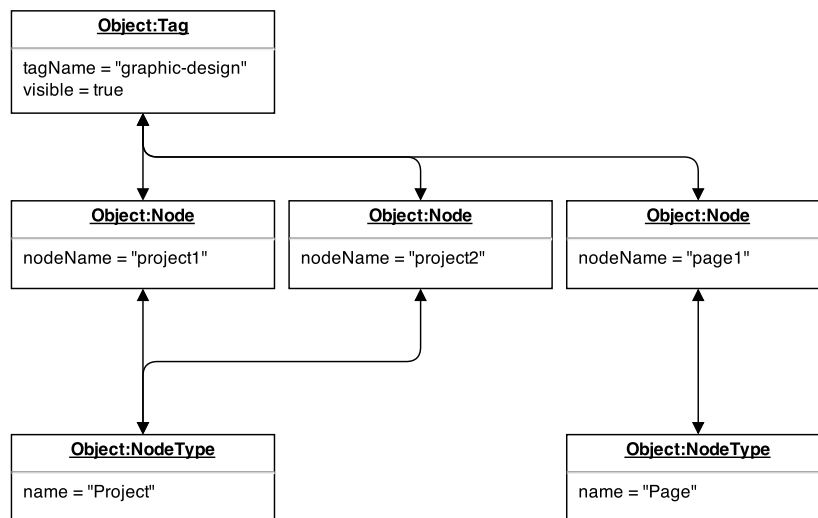
Then register your subscriber to the Roadiz event dispatcher in your theme `setupDependencyInjection` :

```
/** @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 :

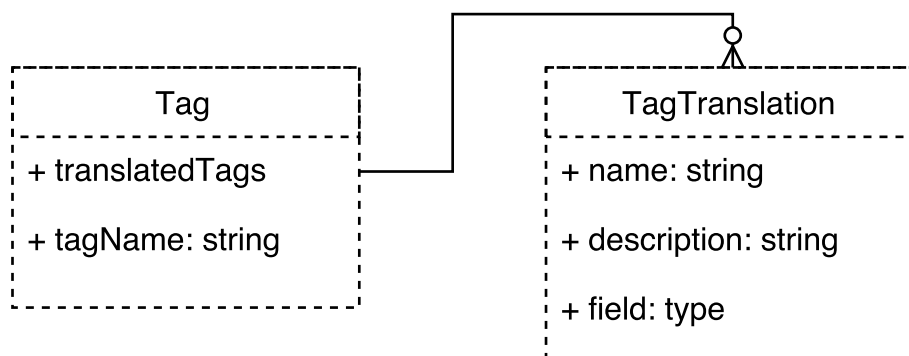


You can see that tags can gather heterogenous nodes coming from different types (pages and projects). Tags can be used to display a category-navigation on your theme or to simply tidy your backoffice node database.

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.

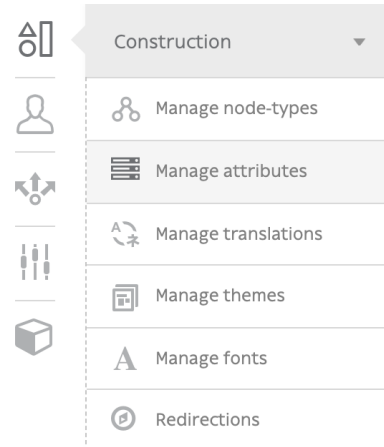
They are available with `getDocuments()` method and will be ordered by **position only**.

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 Attributes

Attributes are entities meant to qualify features on other entities, such as *nodes*. The main difference between *tags* and *attributes* is that you can set a value for each attributed *node*.



The attribute « Color » can be set to « red » for one node and set to « green » for an other one.

Okay, but now what is the difference between *attributes* and node-type fields ? Not so much because node-type fields describe your node” features too, but they are fixed and defined by the developer. Once your node-type fields are created, you have to implement your feature in your *Twig* templates, and translate it in your XLF files too.

Attributes are meant to be created and added by editors so they can use them in your website lifecycle without needing any further development.

From the developer perspective, attributes are just a collection of entities to be displayed in a loop. Then your editor can create new ones and be sure they will be displayed without any additional development.

```
<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|localizeddate('medium', 'short
        <strong>') }}
      {% elseif attributeValueTranslation is date %}
        {{ attributeValueTranslation.value|localizeddate('medium') }}
      {% elseif attributeValueTranslation is country %}
        {{ attributeValueTranslation.value|country_iso(request.locale) }}
      {% else %}
        {{ attributeValueTranslation.value }}
      {% endif %}
    </li>
  {% endfor %}
</ul>
```

If you grouped your attributes, you can use `grouped_attributes` filter instead :

```
<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|localizeddate(
↪ 'medium', 'short') }}
            {% elseif attributeValueTranslation is date %}
                {{ attributeValueTranslation.value|localizeddate(
↪ 'medium') }}
            {% elseif attributeValueTranslation is country %}
                {{ attributeValueTranslation.value|country_iso(request.
↪ locale) }}
            {% else %}
                {{ attributeValueTranslation.value }}
            {% endif %}
        </li>
    {% endfor %}
</ul>

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


Attributes types

Create an attribute

Code *

Attribute type *

Basic text string ▼

Translations

T-Shirt size

English ▼

Available options (for choice type) +

Taille du t-shirt

French ▼

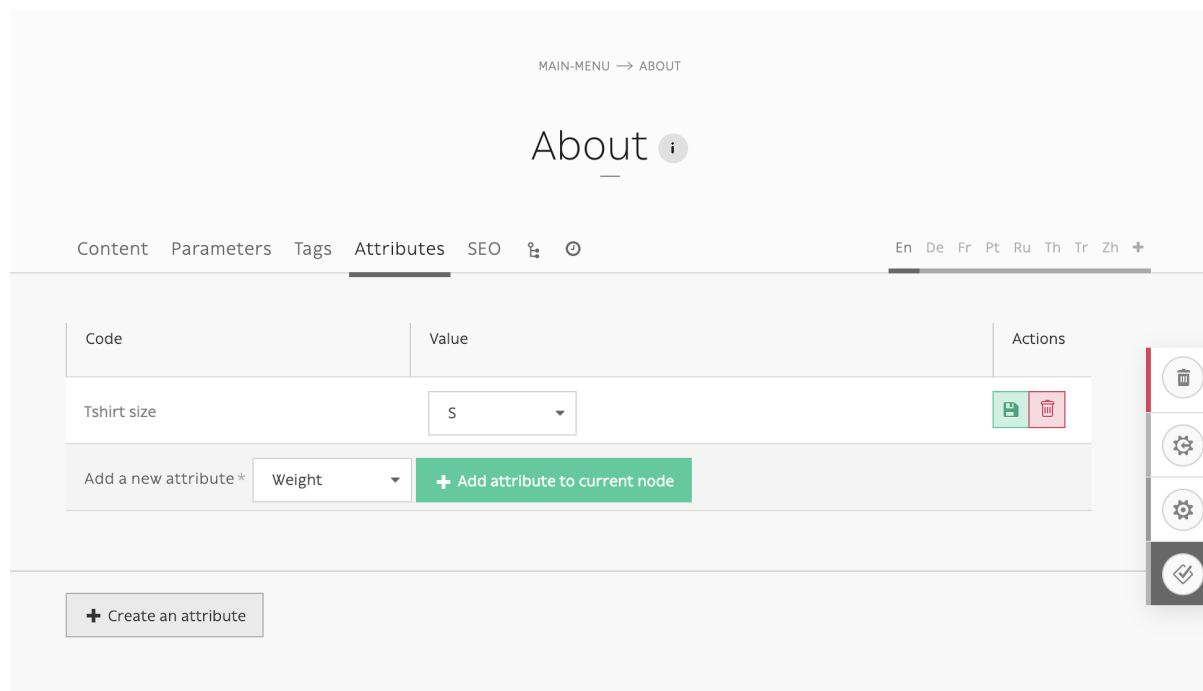
Available options (for choice type) +

⬆

- *String*
- *Date*
- *Date and time*
- *Single choice* : choice among defined options in your attribute
- *Boolean*
- *Integer*
- *Decimal*
- *Email*
- *Color*
- *Country* : ISO 2-letters country code

Add attributes to nodes

Attribute section is available for any node in any translations.



Twig extension

Several filters and tests are available to ease up templating with *attributes* :

Filters

- `attributes` : same as `node_source_attributes()` method, get all available attributes from a `NodesSources`.
- `grouped_attributes` : same as `node_source_grouped_attributes()` method, get all available attributes from a `NodesSources` and gather them into their **group**.
- `attribute_label(translation)` : get attribute translated label or code if not translated.
- `attribute_group_label(translation)` : get attribute group translated name or `canonicalName` if not translated.

Tests

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

3.1.5 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 informations (`MyAwesomeThemeApp.php`)

```
/*
 * Copyright REZO ZERO 2016
 *
 * Description
 *
 * @file MyAwesomeThemeApp.php
 * @copyright REZO ZERO 2014
 * @author Ambroise Maupate
 */
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 $themeName = 'My awesome theme';
protected static $themeAuthor = 'Ambroise Maupate';
protected static $themeCopyright = 'REZO ZERO';
protected static $themeDir = 'MyAwesomeTheme';
protected static $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->prepareThemeAssigation(null, $translation);

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

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

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

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,
    Translation $translation = null
) {
    $this->prepareThemeAssignment($node, $translation);

    return $this->render(
        'types/page.html.twig', // Twig template path
        $this->assignment        // Assignment 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 assignment array.

Note : It's very easy to create *JSON* responses for your API with Roadiz. You just have to replace `$this->render($template, $assignment);` method with `$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 adjustments.

```
/**
 * {@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 becomes 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
     */
    $this->prepareThemeAssignment($home, $translation);

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

Keep in mind that `prepareThemeAssignment` method will assign for you some useful variables no matter you choose 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 overridden in your theme if you 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 be converted to an 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* assignments 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 your 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 be 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 ascendently according to their *title*.
#}
{% set childrenBlocks = nodeSource|children({
  'node.visible': true,
  'node.nodeType': bags.nodeType.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::getTranslationMenuAssignment()`.

This filter returns some useful informations about current page available languages and their urls. See [getTranslationMenuAssignment 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.
- `truncate` and `wordwrap` which are parts of the [Text 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);
    });
}
```

```

    }
    });
    $filters->add($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/NSxxxxx` 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,

```

```
'height':200,
'crop':"1:1",
'quality':75,
'embed':true
}) }}
{% endif %}
```

HTML output options

- **absolute** (true|false), generates an *absolute* URL with protocol, domain-name and base-url. This must be used for social network images.
- **embed** (true|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), 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), use <picture> element instead of image and allow serving WebP image to compatibles browsers. **Only use if your server support WebP.**
- **inline** (true|false), **for SVG**, display SVG inline code in html instead of using an <object> tag. Default true.
- **loading** (autolazy|eager), 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)
- **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 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)
- **quality** (1-100)
- **flip** (h or v), mirror your image vertical or horizontal
- **blur** (1-100) (*can be really slow to process*)
- **sharpen** (1-100)
- **contrast** (1-100)
- **background** (hexadecimal color without #)
- **progressive** (boolean), it will interlace the image if it's a *PNG* file.
- **noProcess** (boolean) : Disable image processing

Audio / Video options

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

For *Soundcloud* embeds

- **hide_related** (boolean)
- **show_comments** (boolean)
- **show_user** (boolean)
- **show_reposts** (boolean)
- **visual** (boolean)

For *Mixcloud* embeds

- **mini** (boolean) defaults to `false`
- **light** (boolean) defaults to `true`
- **hide_cover** (boolean) defaults to `true`
- **hide_artwork** (boolean) defaults to `false`

For *Vimeo* embeds

- **displayTitle** (boolean)
- **byline** (boolean)
- **portrait** (boolean)
- **color** (boolean)
- **api** (boolean)
- **automute** (boolean)
- **autopause** (boolean)

For *Youtube* embeds

- **modestbranding** (boolean)
- **rel** (boolean)
- **showinfo** (boolean)
- **start** (integer)
- **end** (integer)
- **enablejsapi** (boolean)
- **playlist** (boolean)

You can use **multiple source files** for one video document or audio document. Just upload a file using the 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 :

```

```

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">

  
</picture>

```

More document details

You can find more details in [our API documentation](#).

- If document is an **image** : `getDocumentByArray` method will generate an `` 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 data-base. 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` twig command.

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

Extending your Twig assignment

For a simple website theme, base assignment 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 assigned 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
namespace Themes\MyTheme\Controllers;

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

/**
 * Frontend controller to handle Page node-type request.
 */
class PageController extends MyThemeApp
{
    /**
     * 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,
        Translation $translation = null
    ) {
        $this->prepareThemeAssignment($node, $translation);

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

```

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

```

{% 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 assignment

Custom assignments are great but what can I do if I have to use the same variables in several controllers? We added a special `extendAssignment` method which is called at the end of your theme preparation process (`prepareThemeAssignment` and `prepareNodeSourceAssignment`). 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.

```

/**
 * {@inheritdoc}
 */
protected function extendAssignment()
{
    parent::extendAssignment();

    $this->assignment['head']['facebookUrl'] = $this->get('settingsBag')->get(
        'facebook_url');
    $this->assignment['head']['facebookClientId'] = $this->get('settingsBag')->
        get('facebook_client_id');
    $this->assignment['head']['instagramUrl'] = $this->get('settingsBag')->get(
        'instagram_url');
    $this->assignment['head']['twitterUrl'] = $this->get('settingsBag')->get(
        'twitter_url');
    $this->assignment['head']['googleplusUrl'] = $this->get('settingsBag')->get(
        'googleplus_url');
    $this->assignment['head']['googleClientId'] = $this->get('settingsBag')->get(
        'google_client_id');
    $this->assignment['head']['maps_style'] = $this->get('settingsBag')->get(
        'maps_style');
    $this->assignment['head']['themeName'] = static::$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 moduable 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 assignments !

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

```
{% set blocks = nodeSource|children({
    node.nodeType : bags.nodeTypes.get('BasicBlock'),
}) %}
```

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 assigned blocks.

```
{% if blocks %}
<section class="page-blocks">
{% for pageBlock in blocks %}
    {% include '@MyTheme/blocks/' ~ pageBlock.node.nodeType|lower ~ '.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 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
{
    function blockAction(Request $request, NodesSources $source, $assignation)
    {
        $this->prepareNodeSourceAssignation($source, $source->getTranslation());

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

        // If you assignate session messages here, do not assignate 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 ($form->isValid()) {
            // some stuff
            throw new ForceResponseException($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, $max = 3, $_locale = 'en
    ↪')
    {
        $translation = $this->bindLocaleFromRoute($request, $_locale);
        $this->prepareThemeAssignment(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->prepareThemeAssignment(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->assignment['articles'] = $listManager->getEntities();
    $this->assignment['filters'] = $listManager->getAssignment();

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

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 informations 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 severals 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->prepareThemeAssigation(null, $translation);
    $helper = $this->get('securityAuthenticationUtils');
    $this->assigation['last_username'] = $helper->getLastUsername();
    $this->assigation['error'] = $helper->getLastAuthenticationError();

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

```

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->setAssigation([
    '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();
```

Assigation works the same way as HTML template for your website, you must assign every content and informations 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 %}
```

3.1.6 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;
3 use Symfony\Component\Form\Extension\Core\Type\FileType;
4 use Symfony\Component\Form\Extension\Core\Type\SubmitType;
5
6 ...
7 // Create contact-form manager and add 3 default fields.
8 $contactFormManager = $this->createContactFormManager()
9                       ->withDefaultFields();
10 /*
11  * (Optional) Add custom fields...
12 */
```

```

13 $formBuilder = $contactFormManager->getFormBuilder();
14 $formBuilder->add('callMeBack', CheckboxType::class, [
15     'label' => 'call.me.back',
16     'required' => false,
17 ])
18 ->add('document', FileType::class, [
19     'label' => 'document',
20     'required' => false,
21     'constraints' => [
22         new File([
23             'maxSize' => $contactFormManager->getMaxFileSize(),
24             'mimeType' => $contactFormManager->getAllowedMimeTypes(),
25         ]),
26     ]
27 )
28 ->add('send', SubmitType::class, [
29     'label' => 'send.contact.form',
30 ]);
31
32 /*
33  * This is the most important point. handle method will perform form
34  * validation and send email.
35  *
36  * Handle method should return a Response object if everything is OK.
37  */
38 if (null !== $response = $contactFormManager->handle()) {
39     return $response;
40 }
41
42 $form = $contactFormManager->getForm();
43
44 // Assignate your form view to display it in Twig.
45 $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 :

```

1  {#
2  # Display contact errors
3  #}
4  {% for label, messages in app.flashes(['warning', 'error']) %}
5      {% for message in messages %}
6          <p class="alert alert-{{ label }}">
7              {{- message -}}
8          </p>
9      {% endfor %}
10 {% endfor %}
11 {#
12 # Display contact form
13 #}
14 {% form_theme contactForm '@MyTheme/forms.html.twig' %}
15 {{ 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.


```
// ./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();

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

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

Recaptcha private key (secret)	<div></div>	<div><div></div><div></div><div></div></div>
Recaptcha site key (public)	<div></div>	<div><div></div><div></div><div></div></div>

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

```
<script src='https://www.google.com/recaptcha/api.js'></script>
```

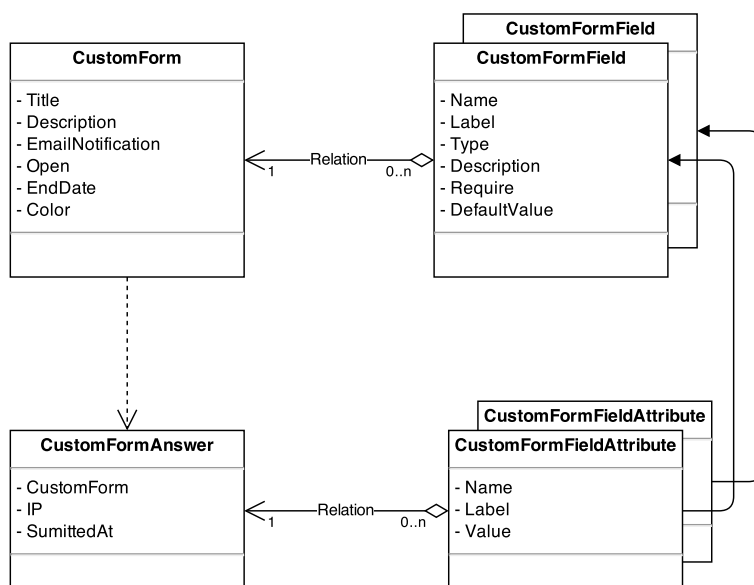
```
{# In your theme' forms.html.twig file #}
{% block recaptcha_widget -%}
    <div class="g-recaptcha" data-sitekey="{{ configs.publicKey }}"></div>
{%- endblock recaptcha_widget %}
```

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.

```

1 use RZ\Roadiz\Core\Entities\CustomForm;
2 use RZ\Roadiz\Core\Exceptions\EntityAlreadyExistsException;
3 use RZ\Roadiz\Core\Exceptions\ForceResponseException;
4 use RZ\Roadiz\Utils\CustomForm\CustomFormHelper;
5 use Symfony\Component\Form\FormError;
6 use Symfony\Component\HttpFoundation\JsonResponse;
7
8 // ...

```

```

9
10 /*
11  * Get your custom form instance from your node-source
12  * only if you added a *custom-form reference field*.
13  */
14 $customForms = $this->nodeSource->getCustomformReference();
15 if (isset($customForms[0]) && $customForms[0] instanceof CustomForm) {
16     /** @var CustomForm $customForm */
17     $customForm = $customForms[0];
18
19     /*
20      * Verify if custom form is still open
21      * for answers
22      */
23     if ($customForm->isFormStillOpen()) {
24         /*
25          * CustomFormHelper will generate Symfony form against
26          * Roadiz custom form entity.
27          * You can add a Google Recaptcha passing following options.
28          */
29         $helper = new CustomFormHelper($this->get('em'), $customForm);
30         $form = $helper->getFormFromAnswer($this->get('formFactory'), null, true,
31 ↪ [
32             'recaptcha_public_key' => $this->get('settingsBag')->get('recaptcha_
33 ↪ public_key'),
34             'recaptcha_private_key' => $this->get('settingsBag')->get('recaptcha_
35 ↪ private_key'),
36             'request' => $request,
37         ]);
38         $form->handleRequest($request);
39
40         if ($form->isSubmitted() && $form->isValid()) {
41             try {
42                 $answer = $helper->parseAnswerFormData($form, null, $request->
43 ↪ getClientIp());
44
45                 if ($request->isXmlHttpRequest()) {
46                     $response = new JsonResponse([
47                         'message' => $this->getTranslator()->trans('form_has_been_
48 ↪ successfully_sent')
49                     ]);
50                 } else {
51                     $this->publishConfirmMessage(
52                         $request,
53                         $this->getTranslator()->trans('form_has_been_successfully_
54 ↪ sent')
55                     );
56                     $response = $this->redirect($this->generateUrl($this->
57 ↪ nodeSource->getParent()));
58                 }
59                 /*
60                  * If you are in a BlockController use ForceResponseException
61                  */
62                 throw new ForceResponseException($response);
63                 /*
64                  * Or directly return redirect response.
65                  */
66                 //return $response;
67             } catch (EntityAlreadyExistsException $e) {
68                 $form->addError(new FormError($e->getMessage()));
69             }
70         }
71     }
72 }

```

```

65     $this->assignation['session']['messages'] = $this->get('session')->
↳getFlashBag()->all();
66     $this->assignation['form'] = $form->createView();
67 }
68 }

```

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

```

1  {#
2  # AwesomeTheme/Resources/views/form.html.twig
3  #}
4  {% extends "bootstrap_3_layout.html.twig" %}
5
6  {% block form_row -%}
7      <div class="form-group form-group-{{ form.vars.block_prefixes[1] }} form-
↳group-{{ form.vars.name }}">
8          {% if form.vars.block_prefixes[1] != 'separator' %}
9              {{- form_label(form) -}}
10             {% endif %}
11             {{- form_errors(form) -}}
12             {#
13              # Render field description inside your form
14              #}
15             {% if form.vars.attr['data-description'] %}
16                 <div class="form-description">
17                     {{ form.vars.attr['data-description']|markdown }}
18                 </div>
19             {% endif %}
20             {{- form_widget(form) -}}
21         </div>
22     {%- endblock form_row %}
23
24     {% block recaptcha_widget -%}
25         <div class="g-recaptcha" data-sitekey="{{ configs.publicKey }}"></div>
26     {%- endblock recaptcha_widget %}

```

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

```

1  {#
2  # AwesomeTheme/Resources/views/form-blocks/customformblock.html.twig
3  #}
4  {% if form %}
5      {% form_theme form '@AwesomeTheme/form.html.twig' %}
6      {{ form_start(form) }}
7      {{ form_widget(form) }}
8      <div class="form-group">
9          <button class="btn btn-primary" type="submit">{% trans %}send_form{% _
↳endtrans %}</button>
10      </div>
11      {{ form_end(form) }}
12  {% else %}
13      <p class="alert alert-warning">{% trans %}form_is_not_available{% endtrans %}
↳</p>
14  {% endif %}

```

3.1.7 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 implicitly 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'),
        // ...
    ],
];
```

3.1.8 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

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).

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\RedirectRouter
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\AuthenticationUtils
contextListener Symfony\Component\Security\Http\Firewall\ContextListener
accessMap Symfony\Component\Security\Http\AccessMap
userProvider RZ\Roadiz\Core\Handlers\UserProvider
userChecker Symfony\Component\Security\Core\User\UserChecker
daoAuthenticationProvider Symfony\Component\Security\Core\Authentication\Provider\DaoAuthenticationProvider
rememberMeAuthenticationProvider Symfony\Component\Security\Core\Authentication\Provider\RememberMeAuthenticationProvider
rememberMeCookieName string
rememberMeCookieLifetime integer
cookieClearingLogoutHandler Symfony\Component\Security\Http\Logout\CookieClearingLogoutHandler
tokenBasedRememberMeServices Symfony\Component\Security\Http\RememberMe\TokenBasedRememberMeServices
rememberMeListener Symfony\Component\Security\Http\Firewall\RememberMeListener
authenticationProviderList array<Symfony\Component\Security\Core\Authentication\Provider\AuthenticationProvider>
authenticationManager Symfony\Component\Security\Core\Authentication\AuthenticationManager
security.voters array
accessDecisionManager Symfony\Component\Security\Core\Authorization\AccessDecisionManager
securityAuthenticationTrustResolver Symfony\Component\Security\Core\Authentication\TrustResolverInterface
securityAuthorizationChecker Symfony\Component\Security\Core\Authorization\AuthorizationCheckerInterface
securityTokenStorage Symfony\Component\Security\Core\Authentication\Token\Storage\TokenStorageInterface

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`

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

3.1.9 Serialization

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

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-type 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.

3.1.10 Case studies

Some step-to-step guides to work with 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 caught by *Mailcatcher*. You can see them at address `http://localhost:1080`.

3.1.11 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 informations 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 codestyle 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 1 src themes/Rozier themes/Install
```

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.

3.1.12 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>

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 a 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 these 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 :

```
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 used 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-office 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 invisible.
- `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
declare(strict_types=1);

namespace Themes\MyTheme\Event;

use GeneratedNodeSources\NSGroupBlock;
use GeneratedNodeSources\NSPage;
use Pimple\Container;
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
{
    /**
     * @var Container
     */
    protected $container;

    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;
});

```


A

accessDecisionManager, [102](#)
accessDeniedHandler, [103](#)
accessMap, [102](#)
AdapterInterface, [103](#)
argumentResolver, [102](#)
assetPackages, [99](#)
attribute, [105](#)
authenticationManager, [102](#)
authenticationProviderList, [102](#)

B

backoffice.entries, [99](#)

C

CacheProvider : :class, [100](#)
color, [104](#)
config, [104](#)
config.handler, [104](#)
config.path, [104](#)
console.commands, [100](#)
contactFormManager, [100](#)
contextListener, [102](#)
cookieClearingLogoutHandler, [102](#)
csrfTokenManager, [102](#)
custom_form, [105](#)
custom_form.handler, [101](#)
custom_form_field.handler, [101](#)

D

daoAuthenticationProvider, [102](#)
debugbar, [100](#)
debugbar.renderer, [100](#)
defaultTranslation, [103](#)
dispatcher, [99](#)
doctrine.debugstack, [100](#)
doctrine.entities_paths, [100](#)
doctrine.relative_entities_paths, [100](#)
document, [105](#)
document.factory, [101](#)
document.handler, [101](#)
document.platforms, [100](#)
document.url_generator, [101](#)

document.viewer, [101](#)

E

em, [100](#)
em.config, [100](#)
em.eventSubscribers, [100](#)
emailManager, [100](#)
embed_finder.dailymotion, [100](#)
embed_finder.mixcloud, [100](#)
embed_finder.soundcloud, [100](#)
embed_finder.spotify, [100](#)
embed_finder.ted, [100](#)
embed_finder.twitch, [100](#)
embed_finder.vimeo, [100](#)
embed_finder.youtube, [100](#)

F

factory.handler, [101](#)
firewall, [103](#)
firewallMap, [103](#)
folder, [105](#)
folder.handler, [101](#)
font.handler, [101](#)
form.extensions, [101](#)
form.type.extensions, [101](#)
formFactory, [101](#)
formValidator, [101](#)

G

group.handler, [101](#)
groupVoter, [103](#)

H

httpKernel, [102](#)
httpUtils, [102](#)

I

id, [104](#)
interventionRequest, [99](#)
interventionRequestConfiguration, [99](#)
interventionRequestLogger, [99](#)
interventionRequestSubscribers, [99](#)
interventionRequestSupportsWebP, [99](#)

J

JMSSerializerSerializerBuilder, [103](#)

K

kernel, [99](#)

L

logger, [101](#)
logger.handlers, [101](#)
logger.path, [101](#)
logger.themes, [103](#)

M

mailer, [102](#)
mailer.transport, [102](#)
messagescollector, [100](#)

N

newsletter.handler, [101](#)
node, [105](#)
node.handler, [101](#)
node_type, [105](#)
node_type.handler, [101](#)
node_type_field.handler, [101](#)
nodeApi, [100](#)
NodeFactory : :class, [100](#)
nodeRouter, [102](#)
nodes_sources, [104](#)
nodes_sources.handler, [101](#)
nodes_sources_base, [104](#)
nodes_sources_default, [104](#)
nodes_sources_documents, [104](#)
nodes_sources_“group“, [105](#)
nodeSourceApi, [100](#)
nodesSourcesUrlCacheProvider, [100](#)
nodeTypeApi, [100](#)
nodeTypesBag, [99](#)

P

passwordEncoder, [103](#)
position, [104](#)

R

redirectionRouter, [102](#)
rememberMeAuthenticationProvider, [102](#)
rememberMeCookieLifetime, [102](#)
rememberMeCookieName, [102](#)
rememberMeListener, [102](#)
requestContext, [102](#)
requestStack, [102](#)
resolver, [102](#)
roleHierarchy, [103](#)
roleHierarchyVoter, [103](#)
rolesBag, [99](#)

Rollerworks\Component\PasswordStrength\Blacklist\BlacklistProviderInterface, [101](#)

Rollerworks\Component\PasswordStrength\Validator\Constraints\Blacklist, [101](#)

routeCollection, [102](#)

routeListener, [102](#)

router, [102](#)

RZ\Roadiz\CMS\Importers\ChainImporter, [101](#)

RZ\Roadiz\CMS\Importers\GroupsImporter, [101](#)

RZ\Roadiz\CMS\Importers\NodesImporter, [101](#)

RZ\Roadiz\CMS\Importers\NodeTypesImporter, [101](#)

RZ\Roadiz\CMS\Importers\RolesImporter, [101](#)

RZ\Roadiz\CMS\Importers\SettingsImporter, [101](#)

RZ\Roadiz\CMS\Importers\TagsImporter, [101](#)

S

security.voters, [102](#)
securityAccessListener, [103](#)
securityAuthenticationTrustResolver, [102](#)
securityAuthenticationUtils, [102](#)
securityAuthorizationChecker, [102](#)
securityTokenStorage, [102](#)
serializer, [103](#)
serializer.subscribers, [103](#)
session, [102](#)
session.pdo, [102](#)
session.storage, [102](#)
sessionTokenStorage, [102](#)
setting, [105](#)
setting_group, [105](#)
settingsBag, [99](#)
SolariumFactoryInterface, [103](#)
solr, [103](#)
solr.ready, [103](#)
solr.search.document, [103](#)
solr.search.nodeSource, [103](#)
staticRouter, [102](#)
stopwatch, [99](#)
switchUser, [103](#)

T

tag, [105](#)
tag.handler, [101](#)
tag_base, [105](#)
tagApi, [100](#)
TagFactory : :class, [101](#)
ThemeGenerator, [103](#)
themeResolver, [103](#)
timestamps, [104](#)
tokenBasedRememberMeServices, [102](#)
translation, [105](#)
translation.handler, [101](#)
translation.viewer, [101](#)
translator, [103](#)
translator.locale, [103](#)
twig.cacheExtension, [104](#)
twig.cacheFolder, [103](#)
twig.CentralFunctionExtension, [104](#)
twig.environment, [103](#)
twig.environment_class, [103](#)

twig.extensions, [104](#)
twig.filters, [104](#)
twig.formRenderer, [103](#)
twig.fragmentHandler, [104](#)
twig.loaderFileSystem, [103](#)
twig.profile, [104](#)
twig.routingExtension, [104](#)

U

urlGenerator, [102](#)
user.viewer, [101](#)
userChecker, [102](#)
userEncoderFactory, [103](#)
userImplementations, [103](#)
userProvider, [102](#)
utils.nodeNameChecker, [104](#)
utils.uniqueNodeGenerator, [104](#)
utils.universalDataDuplicator, [104](#)

V

versionStrategy, [99](#)

W

workflow.node_workflow, [104](#)
workflow.registry, [104](#)