

# Licence Professionnelle

## Systèmes informatiques et Logiciels

### METINET

*Du 3 octobre au 15 septembre*

## RAPPORT D'ALTERNANCE

*Hugo Alliaume*

*Promotion 2016/2017*

*NinaCMS*

*SAILING Communication & Technologies*  
*58, rue des Clercs 38200 Vienne*

*Maître d'Alternance*  
M. Philippe BOIREAUD

*Tuteur d'Alternance*  
M. Fabrice JAILLET

Université Claude Bernard – Lyon 1



Institut Universitaire de Technologie

**Institut Universitaire de Technologie**

**Département Informatique**  
Site de Bourg-en-Bresse

Département Informatique – IUT Lyon 1

71 rue Peter Fink – 01000 BOURG-EN-BRESSE

Tél. : 04 74 45 50 59 – Fax : 04 74 45 50 51

Mail : [iut.lp.metinet@univ-lyon1.fr](mailto:iut.lp.metinet@univ-lyon1.fr)

Site web : <http://iut.univ-lyon1.fr>



## Remerciements

*Je remercie la société Sailing Communication & Technologies, pour m'avoir accepté en alternance et considéré tel un véritable alternant (notamment pour le respect des périodes de travail, rien ne m'était demandé en dehors des heures de travail), où j'ai pu me rendre utile pendant la période de développement du projet NinaCMS.*

*Je remercie M. Philippe BOIREAUD, mon maître d'apprentissage, qui a su me guider et me ré-orienter pendant la période de développement du projet NinaCMS, avec lequel j'ai pu beaucoup apprendre sur le monde professionnel.*

*Je remercie Mme. Céline RUELLAN, Directrice des Ressources Humaines, pour avoir répondu à diverses questions par mail ainsi que pour m'avoir envoyé mes bulletins de paie au format PDF pendant toute cette année.*

*Je remercie M. Fabrice JAILLET, mon tuteur d'alternance, pour m'avoir suivi et encadré durant la période de l'alternance, ainsi que pour ses visites en entreprise pour voir si tout se passait bien et observer l'avancement du projet.*

*Enfin, je remercie les enseignants et surtout les intervenants (plus particulièrement M. Boris GUÉRY) qui nous ont accompagnés pendant cette année d'alternance à l'IUT de Bourg-en-Bresse, afin de nous préparer pour le monde professionnel de l'entreprise notamment grâce à leurs multiples années d'expérience.*



## Table des matières

Remerciements.....	3
1. Présentation de l'entreprise.....	7
1.1. Activités principales.....	7
1.2. Présentation de l'effectif.....	7
1.3. Partenaires.....	8
2. Présentation de l'environnement de travail.....	9
2.1. Environnement humain.....	9
2.2. Matériels et logiciels utilisés.....	9
3. Travail effectué.....	11
3.1. Présentation d'un composant.....	11
3.2. Présentation des Field.....	15
3.3. Présentation des Fragment.....	17
3.4. Présentation des Property.....	18
3.5. Partie publique.....	20
3.6. Partie administration.....	20
3.1. Présentation de l'architecture du projet.....	29
4. Conclusion.....	32
4.1. Aspects techniques et expérience acquise.....	32
4.2. Perception et intégration dans l'entreprise.....	32
4.3. Formation de l'IUT.....	33
5. Glossaire.....	35
6. Bibliographie.....	37
7. Annexes.....	39
7.1. FieldFile.....	39
7.2. FieldImage.....	40
7.3. FieldLinks.....	41
7.4. Partie publique.....	42
7.5. Partie administration.....	43



# 1. Présentation de l'entreprise

SAILING est une agence de communication Digitale & Print. Actuellement composée de 6 collaborateurs, elle a pu dégager un chiffre d'affaire de 450.000€ pendant l'année 2016.

Le siège de la société se trouve à Saint-Maur-des-Fossées (94) dans la région parisienne, et le siège secondaire se situe à Vienne (38) au sud de Lyon. C'est dans ce dernier que j'ai pu effectuer ma formation en alternance pendant toute cette année.

L'agence a vu le jour en 1996 et s'est d'abord concentré autour des métiers de l'édition et de la conception graphique de supports imprimés. Plus tard, la démocratisation du Web et l'émergence de nouveaux supports de communication ont naturellement conduit l'équipe à se positionner sur de nouveaux métiers.

Après 10 années d'existence, la structure est devenue « **SAILING Communication & Technologies** », et s'est définitivement organisée en trois pôles :

- Conseil en communication & Marketing,
- Création & Édition,
- Développement web & Intégration technique / R&D.

## 1.1. Activités principales

Les activités principales de SAILING sont les suivantes :

- C'est un studio de création graphique (création de maquette de site web, de logo, ...),
- C'est un prestataire internet (fournit une connexion Internet à des entreprises et/ou particuliers),
- Elle s'occupe du développement et de l'intégration de site web,
- Exerce de la R&D, et auteur du moteur d'applications web WysiUp.

## 1.2. Présentation de l'effectif

Voici une liste non exhaustive des personnes travaillant actuellement chez SAILING :

- Philippe Boireaud : À la tête du pôle de compétence de la **direction technique et R&D**,
- Yann Ruellan : À la tête du pôle de compétence des **conseils en communication et le pilotage de projet**,
- Éric Ruellan : À la tête du pôle de compétence de la **direction artistique**,

- Céline Ruellan : Directrice des Ressources Humaines,
- Alexis Lachaux : Commercial.

### **1.3. Partenaires**

SAILING est partenaire de FEEL EUROPE Groupe, une SSII conseil en systèmes d'information, en mesure de soutenir des projets nécessitant de faire appel à des compétences techniques additionnelles.



## 2. Présentation de l'environnement de travail





### 2.1. Environnement humain

Mon année d'alternance s'est déroulée dans l'agence de Vienne. Uniquement composée de développeurs, j'ai donc travaillé dans le service de développement et réalisation de projets webs avec Philippe Boireaud et Thibault Laperrière, lui aussi alternant (Philippe étant situé dans son bureau, et Thibault et moi dans l'open-space).







Pendant cette année d'alternance, j'ai eu pour tâche de travailler uniquement sur le projet NinaCMS, le tout attentivement surveillé et orchestré par Philippe. J'ai aussi eu l'occasion d'aider à plusieurs reprises lorsqu'il avait besoin d'aide pendant son travail.

### 2.2. Matériels et logiciels utilisés


L'ordinateur de travail était directement fourni par l'entreprise, le système d'exploitation Ubuntu étant déjà pré-installé, j'ai juste eu à installer mon environnement de développement.

	Système d'exploitation : <b>Ubuntu 16.04</b>
	Environnement de Développement Intégré (EDI) : <b>PhpStorm</b> (2017.2)
	Navigateurs Web : <b>Google Chrome</b> (XX), <b>Mozilla Firefox</b> (XX), et <b>Opera</b> (XX)
	Traitement de texte : <b>LibreOffice Writer</b> (5.2)

J'ai utilisé les langages de programmation, outils de développement, et les frameworks suivants :

	<b>PHP</b> (5.6)
	Des composants de <b>Laravel</b> (routage, DI, ORM, validation, ...)
	<b>Node.js</b> (6.11)
	<b>VueJS</b> (2.4)
	<b>Webpack</b> (3.5)
	<b>HTML</b> , <b>JavaScript</b> (ES6) et <b>SCSS</b> .

J'ai aussi utilisé divers outils externes pour le bon déroulement du projet :

	<b>Git</b> , <b>GitKraken</b> et <b>Gogs</b>
	<b>JSDoc</b> et un plugin « jsdoc-vuejs » créé à l'occasion pour documenter des composants <b>VueJS</b>
	<b>Sami</b>



### 3. Travail effectué

J'ai donc travaillé en autonomie sur NinaCMS tout le long de l'alternance, en étant encadré par mon Maître d'Alternance, Philippe Boireaud. Je me suis basé sur son prototype afin de continuer le projet. NinaCMS doit pouvoir fonctionner de **manière autonome**, ou à l'**intérieur d'un CMS**, comme **WordPress**.

Le projet se base sur le principe de composants. Dans NinaCMS, **tout** est un composant. Cela peut être un composant « **Article** », un composant « **Conteneur de colonnes** », ou alors un média « **Image** », « **Video** », etc. Un composant peut être inclus dans un autre, on arrive au final à un arbre de composants.

NinaCMS étant un éditeur visuel de site web, nous avons donc une **partie publique** qui sera affichée aux visiteurs du site, et nous avons une partie administration – basée sur la partie publique - qui sera réservée à l'administrateur.

La partie administration s'accroche à la partie publique en y ajoutant plusieurs menus pour modifier, déplacer, copier, coller des composants, ainsi qu'un explorateur de composants, un explorateur de fichiers, et divers outils.

#### 3.1. Présentation d'un composant

Un composant doit être au minimum composé d'une **classe PHP** et d'un **template (modèle) Twig**. Il peut aussi être composé de styles CSS, de scripts JS, et d'images.

Il est situé dans le dossier « **app/components** », dans un sous-dossier portant le nom du composant (ex : le composant « **Article** » se situe dans le dossier « **app/components/Article** », et le nom de sa classe PHP est « **ArticleComponent** », déclarée dans le fichier « **ArticleComponent.php** »).

La **classe PHP** du composant hérite d'une classe nommée « **BaseComponent** », cette dernière permet de rajouter plusieurs variables de configuration ainsi que plusieurs fonctions pour gérer ce composant.

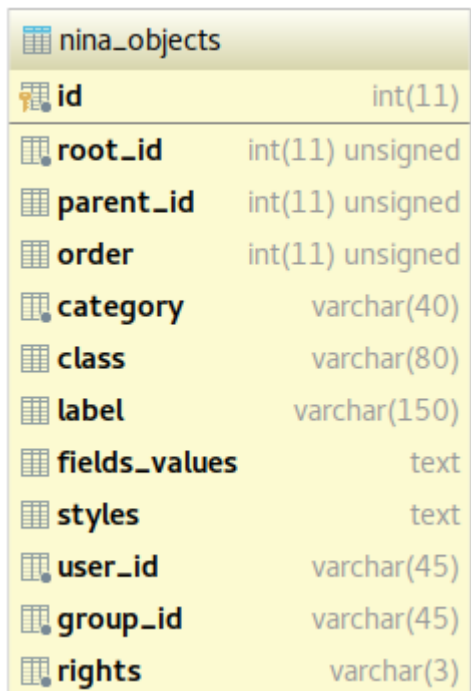
Un composant peut ou non être composé de champs, appelés **Field**. Je présenterai les **Field** plus tard, mais concrètement, un **Field** a un certain type (texte, image, adresse URL, etc ...), et il est utilisé dans la partie administration pour pouvoir customiser certaines valeurs du composant. Par exemple, un composant « **Article** » pourra avoir des **Field** qui permettront de modifier le **titre**, le **chapeau**, le **contenu**, l'**image d'illustration**, etc.

En revanche, le **Field** est incapable de faire le rendu de ses propres valeurs. En effet, si dans un composant nous avons deux **Field** de type **texte**, il se peut que nous voulions afficher la valeur du 1<sup>er</sup> **Field** au format **HTML**, et la valeur du 2<sup>ème</sup> **Field** au format **texte**.

Pour avoir le rendu voulu sans tricher, nous utilisons des **Fragment** (j'en parlerai également plus tard). Ils sont utilisés pour le rendu des valeurs des **Field**, et ce sans s'occuper du type de **Field**. On aura un **Fragment** pour afficher du **contenu HTML**, un autre pour afficher du **contenu textuel**.

Il existe aussi les **Property**, qui permettent de configurer le tag HTML utilisé pour le rendu, la visibilité sur ordinateur, tablette ou smartphone, ainsi que des classes CSS supplémentaires.

### 3.1.1. Représentation d'un composant en base de données



nina_objects	
<b>id</b>	int(11)
<b>root_id</b>	int(11) unsigned
<b>parent_id</b>	int(11) unsigned
<b>order</b>	int(11) unsigned
<b>category</b>	varchar(40)
<b>class</b>	varchar(80)
<b>label</b>	varchar(150)
<b>fields_values</b>	text
<b>styles</b>	text
<b>user_id</b>	varchar(45)
<b>group_id</b>	varchar(45)
<b>rights</b>	varchar(3)

*Représentation d'un composant en  
base de données*























Pour en revenir au composant, voici comment il est stocké en base de données :

- **id** : l'identifiant du composant, typiquement un nombre unique  $\geq 1$ ,
- **root\_id** : l'identifiant racine utilisé pour l'arbre de composants, correspond à un composant ayant pour **id** ce **root\_id**,
- **parent\_id** : l'identifiant du composant parent (cela implique que ce composant soit son enfant), ayant pour **id** ce **parent\_id**,
- **order** : la position du composant avec selon ses frères, un nombre  $\geq 0$
- **category** : la catégorie, peut être un composant, un fichier, ...
- **class** : sa classe, si la catégorie est un composant, alors sa valeur doit être l'un des composants qui se situent dans le dossier « **app/components** »,
- **label** : le nom à afficher dans l'explorateur de composants ou dans l'explorateur de fichier, selon la catégorie,
- **fields\_values** : toutes les valeurs des **Field** liés au composants,
- **styles** : des styles CSS s'appliquant au composant,
- **user\_id**, **group\_id**, et **rights** : permettent la gestion des permissions, lire [Permissions UNIX](#).

### 3.1.2. Représentation d'un composant dans le code PHP

Comme dit plus tôt, un composant hérite du composant de base « **BaseComponent** ». Voici une présentation des propriétés d'un composant :

#### BaseComponent

  fields:Nina\Field\BaseField[]  
  fieldsRules:array  
  fieldsFileRules:array  
  contains:array  
  icon:string  
  template:null|string  
  styles:array|null  
  scripts:array|null  
  adminStyles:array|null  
  adminScripts:array|null  
  group:string

*Propriétés d'un composant dans le  
code PHP*

- **fields** : un tableau de **Field**,
- **fieldsRules** : un tableau contenant des règles de validation de données pour les **fields**. (Voir <https://git.io/v50iA>),
- **fieldsFileRules** : un tableau contenant des règles de validations de données pour les **fields** de « **Fichier** »,
- **contains** : un tableau qui indique quels composants peut contenir le composant (ex : le composant « **Colonne** » peut contenir le composant « **Article** »),
- **icon** : chemin relatif vers l'icône censée représenter le composant dans l'explorateur de composants,
- **group** : utilisé pour créer une arborescence dans l'explorateur de composants,
- **template** : chemin relatif vers le fichier de template Twig à utiliser,
- **styles** et **scripts** : tableaux de chemins relatifs vers des styles CSS et scripts JS, ils seront exécutés dans la **partie publique** mais aussi dans la **partie administration**,
- **adminStyles** et **adminScripts** : tableaux de chemins relatifs des styles CSS et scripts JS, ils seront **uniquement exécutés dans la partie administration**.

Toutes ces propriétés ont été définies comme « **protégées** ». Dans le paradigme de la **Programmation Orienté Objet**, des propriétés protégées restent directement accessibles dans la classe dans lesquelles ces propriétés sont déclarées, mais aussi dans les classes héritant de celle-ci.

Par contre, elles restent inaccessibles en dehors des composants, on appelle ça l'**encapsulation des données**. Si on souhaite tout de même accéder à ces données, il faut pour cela créer des **accesseurs**. Ce sont des méthodes d'une classe qui en général, commence par « **get** ».

Et voici les différentes méthodes d'un composant :

```
m ↗ afterConstruct():void
m ↗ __clone():void
m ↗ getModel():Nina\Contracts\Entities\NinaObject
m ↗ canContains(componentShortClass):bool
m ↗ getAbsolutePath():string
m ↗ getRelativePath():mixed
m ↗ getGroup():string
m ↗ getClass()
m ↗ getShortClass()
m ↗ getMedias():array
m ↗ getFields():Nina\Field\BaseField[]
m ↗ getFieldsRules():array
m ↗ getFieldsFileRules():array
m ↗ getIcon():string
m ↗ getContains():array
m ↗ getMetadata([assets : bool = true]):array
m ↗ getTemplate():string
m ↗ getStyles():array|string
m ↗ getScripts():array|string
```

- ***afterConstruct*** : Appelée après l'initialisation du composant,
- ***\_\_clone*** : méthode spéciale PHP appelée lorsqu'on est en train de cloner un composant (typiquement un copier/coller),
- ***canContains*** : détermine si notre composant peut contenir le nom du composant passé en paramètre,
- ***getModel*** : retourne le composant stocké en base de données, on pourra récupérer son ***id***, ses ***fields\_values***, etc. Voir « **Représentation d'un composant en base de données** »,
- ***getAbsolutePath/getRelativePath*** : retournent le chemin absolu et relatif du composant,
- ***getMetadata*** : retourne des meta-données sur le composant pour être utilisées dans la partie administration,
- ***getClass/getShortClass*** : retourne la classe du composant, ex : « ArticleComponent » ou « Article »,
- Le reste des méthodes ne sont que des accesseurs vers les propriétés protégées décrites plus haut.

## 3.2. Présentation des Field

Les **Field** sont un ensemble de classes PHP et de templates Twig. La classe PHP permet de configurer le **Field**, par exemple choisir son **identifiant**, son **label**, et diverses options. Le template Twig s'occupe de faire le rendu du **Field**.

```
public function afterConstruct()
{
    $this->fields = [
        'text' => [
            'name' => 'Champs textuels',
            'fields' => [
                new FieldText('text', 'Texte :'),
                new FieldRichText('rich_text', 'Texte riche :'),
                new FieldRichTextTinyMce('rich_text_2', 'Texte riche (2) :')
            ]
        ]
    ];
}
```

*Exemple de définitions de Field à un composant*

Edition du composant « Nina\_Test\_629 »

Champs textuels

Texte :

Texte riche :

Texte riche (2) :

↶ ↷ Formats **B** *I* U ☰ ☷ ☹ 🔗

Annuler et fermer

Modifier

*Rendu des Field définis plus haut, pendant l'édition du composant*

La plupart des Field sont basiques, d'autres plus complexes ont été créés avec VueJS afin d'apporter de l'interactivité et des fonctionnalités supplémentaires.

Les Field actuellement disponibles sont :

- **FieldText** : le plus basique, il affiche une petite zone de texte (`<input type="text">`),
- **FieldRichText** : affiche une plus grande zone de texte (`<textarea></textarea>`)
- **FieldRichTextTinyMce** : affiche l'éditeur de texte [TinyMCE](#),
- **FieldDate** : affiche un champ permettant la sélection d'une date, dans une plage donnée ou non (`<input type="date">`),
- **FieldNum** : affiche un champ permettant la sélection d'un chiffre dans une plage donnée ou non (`<input type="number">`),
- **FieldList** : affiche une liste déroulante proposant divers choix (`<select></select>`),
- **FieldCheckbox** : affiche une case à cocher (`<input type="checkbox">`),
- **FieldCheckboxes** : affiche une liste de case à cocher (utilise plusieurs FieldCheckbox en interne),
- **FieldFile (VueJS)** : affiche une interface de sélection de fichiers (depuis l'ordinateur ou la médiathèque de fichiers). Il est possible de spécifier une contraintes sur les types de fichiers acceptés, ainsi que sur la taille. Étant trop complexe, des captures d'écran se trouve en annexe, voir « **7.1. FieldFile** »,
- **FieldImage (VueJS)** : affiche une interface de sélection d'image (depuis l'ordinateur ou la médiathèque de fichiers). Un éditeur d'image simple s'affiche aussi, il permet de rogner, tourner, inverser, ... sur l'image. Il est possible de choisir précisément les dimensions de rognage, ainsi que d'affichage sur le site. Voir « **7.2. FieldImage** »,
- **FieldLinks (VueJS)** : affiche une interface permettant de gérer des liens, il est possible de configurer l'adresse URL, le texte à afficher, et si le lien doit s'ouvrir dans une nouvelle fenêtre. Voir « **7.3. FieldLinks** ».



### 3.3. Présentation des Fragment

Les Fragment sont des petits templates Twig ré-utilisables. Ils prennent la valeur d'un **Field**, et en font le rendu. Ils héritent tous du template « **BaseFragment** ».

#### ▼ fragments

- 📄 BaseFragment.html.twig
- 📄 FragmentCreatedAt.html.twig
- 📄 FragmentDate.html.twig
- 📄 FragmentFile.html.twig
- 📄 FragmentFiles.html.twig
- 📄 FragmentHtml.html.twig
- 📄 FragmentImage.html.twig
- 📄 FragmentImages.html.twig
- 📄 FragmentLink.html.twig
- 📄 FragmentLinks.html.twig
- 📄 FragmentText.html.twig

*Tous les Fragment disponibles*

- **BaseFragment** : le Fragment de base dont tous les autres Fragment doivent ériter,
- **FragmentDate** : affiche une date formatée à la langue définie dans Nina (français ou anglais),
- **FragmentCreatedAt** : utilise le **FragmentDate** pour afficher « Créé le <date> »,
- **FragmentFile** : crée un lien hypertexte vers le fichier,
- **FragmentFiles** : utilise le **FragmentFile** pour chaque fichiers liés au **Field**,
- **FragmentHtml** : fait le rendu en HTML,
- **FragmentImage** : affiche une image,
- **FragmentImage** : utilise le **FragmentImage** pour chaque images liées au **Field**,
- **FragmentLink** : fait le rendu d'un lien hypertexte,
- **FragmentLinks** : utilise le **FragmentLink** pour faire le rendu de chaque liens,
- **FragmentText** : affiche la valeur en dure

Voici comment faire le rendu d'un **Field** dans un composant, en spécifiant le type de **Fragment** :

```
{{ renderField('title', 'Text') }}
```

On appelle la fonction « **renderField** », en indiquant comme 1<sup>er</sup> paramètre le nom du **Field** à rendre, et en indiquant le **nom de Fragment** à utiliser dans le 2ème paramètre.

### 3.4. Présentation des Property

Une **Property** se lie avec un **Field**, et permet de configurer le **Fragment** qui aura pour rôle de faire le rendu de ce **Field**. Il en existe 4 pour l'instant :

1. **Tag** : utilise un **FieldList** pour pouvoir sélectionner le tag HTML à utiliser (section, div, article, span, ...). Ne pas utiliser en même temps que la Property « **Tag du titre** »,
2. **Tag du titre** : utilise un **FieldList** pour pouvoir sélectionner le tag HTML pour un titre (h1, h2, ... h6). Ne pas utiliser en même temps que la Property « **Tag** »,
3. **Classes CSS** : utilise un **FieldText** pour permettre à l'utilisateur de rentrer des classes CSS personnalisées,
4. **Visibilité** : utilise un **FieldCheckboxes** pour choisir si le Fragment doit s'afficher sur ordinateur, sur tablette, ou alors sur smartphone.

Onglet principal

⚙ Titre :

⚙ Contenu:

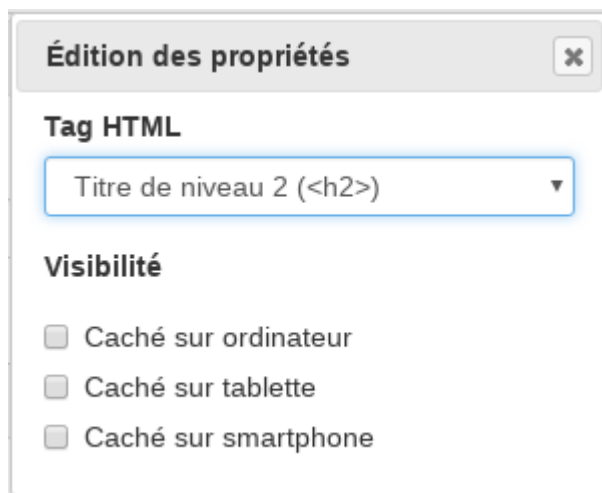
↶ ↷ Formats **B** *I* U ☰ ☷ ☰

La configuration des Property se fait en cliquant sur ⚙

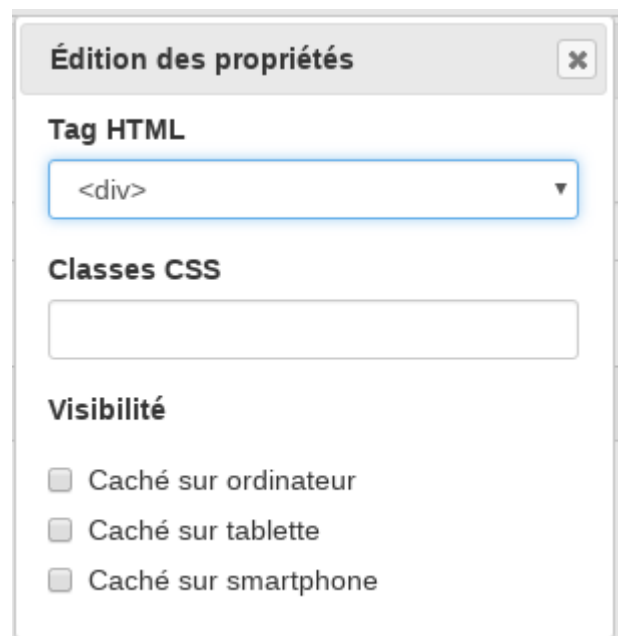
Voici comment se définissent les Property sur un Field. La méthode « **addCommonProperties()** » permet de rajouter les « Property Tag », « Classes CSS » et « Visibilité ».

```
$this->fields = [  
    'primary' => [  
        'name' => 'Onglet principal',  
        'fields' => [  
            (new FieldText('title', 'Titre :'))->addTitleTagProperty()->addVisibilityProperties(),  
            (new FieldRichTextTinyMce('content', 'Contenu: '))->addCommonProperties()  
        ],  
    ],  
];
```

Voici ce qu'il s'affiche lorsqu'on essaye de modifier les Property d'un Field :



*Pour le Field « Titre »*



*Pour le Field « Contenu »*

### 3.5. Partie publique

La partie publique ne fait que récupérer et afficher un arbre de composants en fonction d'un paramètre passé dans l'adresse URL, ex : « [https://mon-site.fr/?root\\_id=1](https://mon-site.fr/?root_id=1) ». Cela signifie qu'on affichera l'arbre de composants ayant pour **root\_id** une valeur à « **1** ».

Cependant, cette manière de faire n'est probablement pas la meilleure à utiliser. En effet, cela veut dire qu'un utilisateur pourra rentrer n'importe quel chiffre compris entre 1 et  $\infty$ , ce qui n'est pas forcément souhaitable. Il faudrait utiliser un système de ce genre : « **?page=home** », où la page « **home** » serait un alias vers les composants ayant un **root\_id** à « **1** ».

Voir un aperçu de la partie publique en annexe : « **7.4. Partie publique** ».

### 3.6. Partie administration

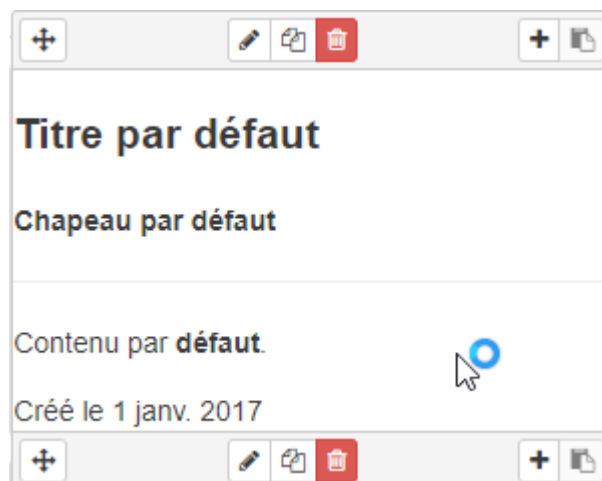
La partie administration a été réalisée entièrement avec VueJS, un framework que j'apprécie très particulièrement pour sa facilité d'apprentissage et d'utilisation, mais qui reste tout de même puissant.

La partie administration ré-utilise le rendu de la partie publique. La partie administration ne fait que se greffer au dessus d'elle.

Voici un aperçu de la partie administration en annexe : « **7.5. Partie administration** »

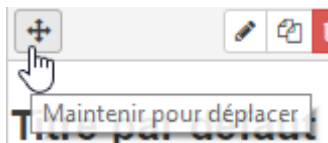
#### 3.6.1. Présentation du manager de composant (ObjectManager)

Le manager de composant – l'ObjectManager – permet de gérer un composant. Il s'affiche au survol du curseur de la souris.



*L'ObjectManager, composé de plusieurs boutons*

## Déplacement d'un composant :



Pour déplacer un composant, il suffit de cliquer sur ce bouton et de maintenir le clic.

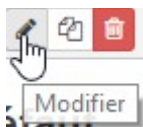


Le composant est maintenant déplaçable dans tous les composants qui peuvent contenir le composant déplacé. (ex : le composant « Colonne » peut contenir le composant « Article »)

Le composant a bien été déplacé !

Une fois le déplacement terminé, un message de succès s'affiche

## Modification d'un composant :



Clic sur le bouton de modification du composant.

Une fenêtre s'affiche avec formulaire d'édition – composé de Field – permettant de modifier notre composant..

Le composant a bien été modifié !

Une fois les modifications correctement sauvegardées, ce message s'affiche.

### Copie d'un composant :



Clic sur le bouton pour copier.

Le composant a bien été copié en mémoire.

Le composant (son **id** et sa **class**) est copié en mémoire, les boutons pour coller avant/après un autre composant sont maintenant activés.

### Collage d'un composant :

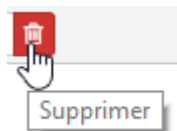


Le composant a bien été collé !

Les boutons coller avant/après sont donc activés. Cliquer sur ce bouton duplique le composant à copier, et le colle au dessus ou en dessous,

Une fois le composant a bien été collé, un message de succès s'affiche.

### Suppression d'un composant :



Le composant a bien été supprimé !

Une fois le bouton de suppression cliqué, un message de confirmation de suppression s'affiche. Si l'utilisateur confirme, le composant et ses enfants sont supprimés.

Une fois le composant supprimé, un message de succès s'affiche.

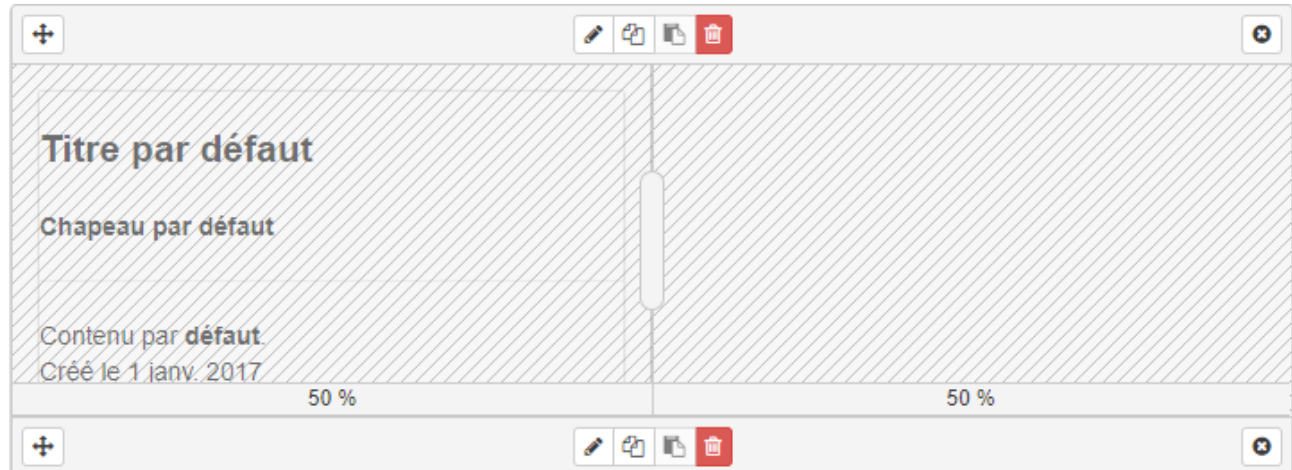
### 3.6.2. Présentation du manager des composants colonnes (BlocManager)

Il existe une autre manager de composants, mais uniquement réservé pour les colonnes, le BlocManager.

Il possède les mêmes fonctionnalités que l'ObjectManager, mais permet en plus de modifier la taille des colonnes, d'ajouter une colonne avant ou après une autre, et de déplacer une colonne avant ou après une autre.



*Survol du curseur de la souris, un bouton apparaît*



*En cliquant sur ce bouton, le BlocManager – ressemblant à l'ObjectManager – apparaît*

Je ne parlerai pas des boutons « maintenir pour déplacer », « modifier », « copier », « coller », et « supprimer » les colonnes, car ils sont très similaires aux boutons de l'ObjectManager. Je ne parlerai donc que des différences entre les deux managers.

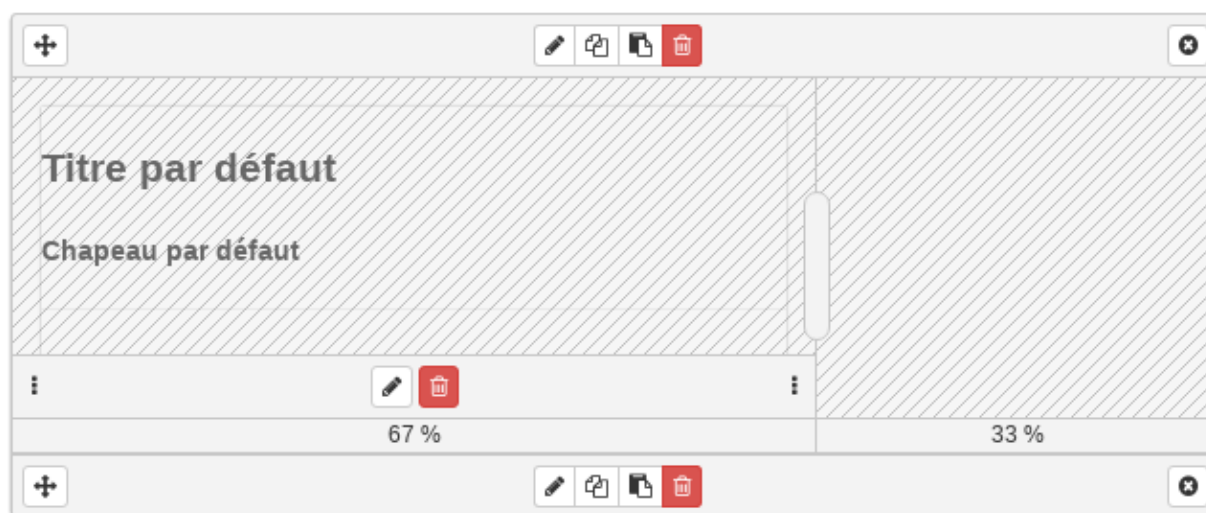
### Re-dimensionnement des colonnes :



Le redimensionnement de colonne se fait grâce à ce bouton.

Il suffit de cliquer, de maintenir le clic, et de déplacer le curseur de gauche à droite pour modifier la largeur des colonnes à côté.

Ici, j'ai redimensionné les colonnes pour qu'elles fassent une largeur de 67 % et de 33 % (valeur relative au conteneur).

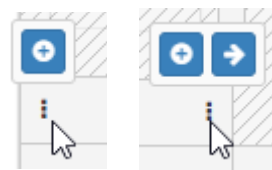


La largeur des colonnes a bien été sauvegardée !

Une fois le redimensionnement terminé, un message de succès s'affiche.

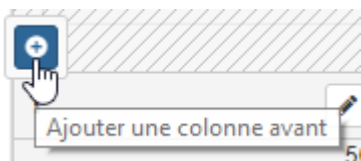
Sur la dernier capture d'écran du gestionnaire de colonnes, on voit qu'un menu pour la colonne apparaît lorsqu'on la survole avec le curseur de la souris. Ce menu est composé d'un bouton « modifier » et « supprimer », qui auront pour rôle de ... modifier et de supprimer la colonne.

On distingue aussi deux icônes aux extrémités du menu, qui au survol afficheront un sous-menu, composé des boutons « **déplacer à gauche** », « **ajouter une colonne à gauche** », ou alors « **déplacer à droite** », « **ajouter une colonne à droite** » dans la mesure du possible..





### Rajouter une colonne :



Ce bouton est affiché uniquement si une colonne peut-être rajoutée sur la gauche ou la droite de la colonne actuelle.

En effet, pour rajouter une colonne, nous avons besoin d'emprunter la largeur de la (des) colonne(s) voisine(s).



Par exemple, ici nous avons à la base une colonne ayant une largeur de 50 %.

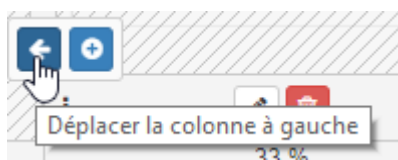
En ajoutant une colonne sur la gauche, nous avons emprunté 8 % de largeur (taille minimale) à la colonne voisine.

On se retrouve maintenant avec deux colonnes, l'une ayant une largeur de 8 %, et l'autre de 42 %.

Une colonne a bien été rajoutée !

Ensuite, un message de succès apparaît.

### Déplacement d'une colonne :



Ce bouton permet de déplacer la colonne actuelle soit vers la gauche, soit vers la droite.



Avant le déplacement



Après le déplacement

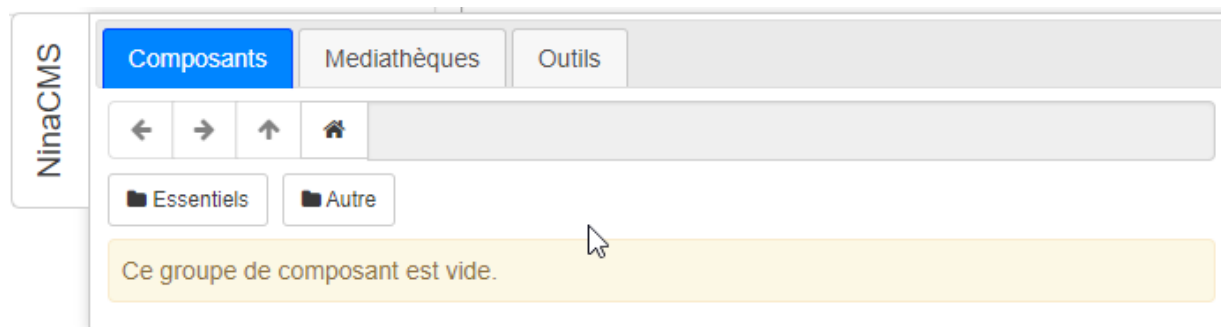
### 3.6.3. Présentation de l'explorateur de composants



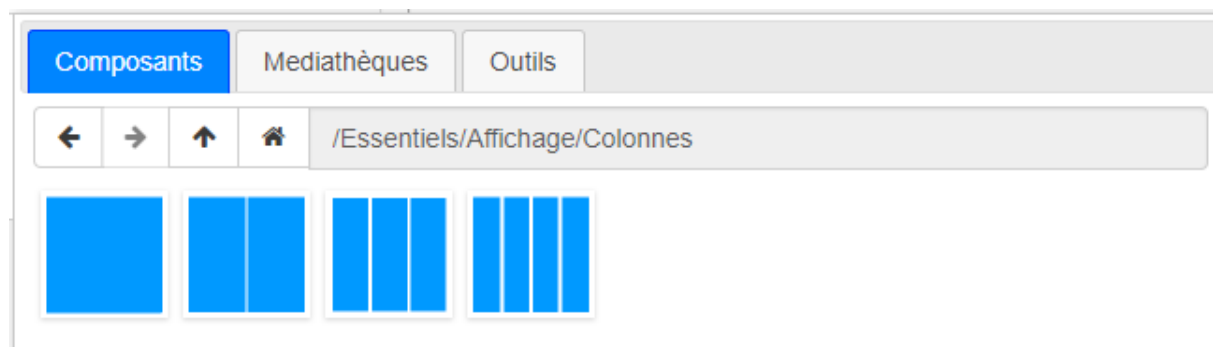
L'explorateur s'affiche en cliquant sur le bouton « **NinaCMS** » sur le côté droit de la page. C'est ce bouton qui apporte un tas d'outils supplémentaires à l'administration.

L'explorateur permet à l'utilisateur de **visualiser tous les composants** existants dans son application, et d'y naviguer à la manière d'un **explorateur** de fichiers. Il offre aussi à l'utilisateur la possibilité de les **glisser** et de les **déposer** dans la page.

En cliquant sur le bouton « NinaCMS », on arrive directement sur l'explorateur de composants, à la racine de l'arborescence. Pour l'instant, il n'y a que 2 groupes principaux de composants, les composants dit **essentiels**, et les **autres**. On observe aussi 4 boutons dans la barre de navigation : « **précédent** », « **suivant** », « **parent** », et « **racine** ».



En cliquant sur les groupes, on arrive enfin à des composants. Ici, j'ai navigué de la racine, en passant par les composants essentiels, les composants d'affichage, et les composants colonnes.



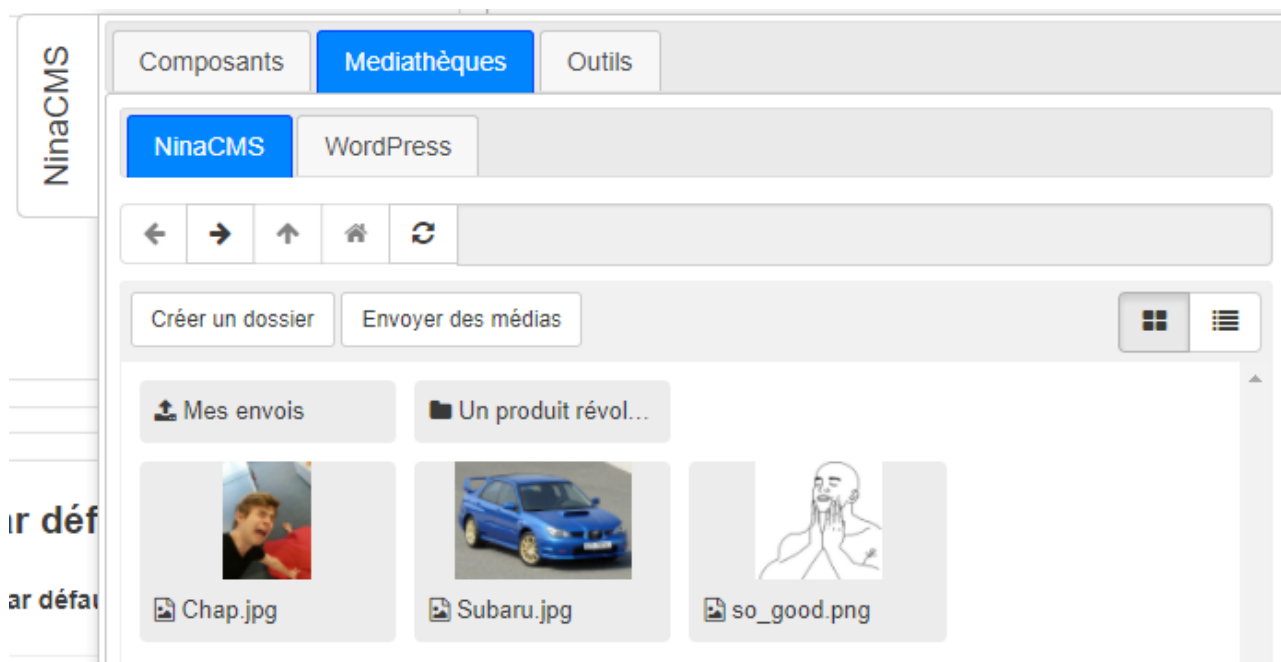
Le curseur indique que les composants sont **glissables** et **déposables**, on appelle ça le « **drag & drop** ». C'est avec cette fonctionnalité que nous pouvons glisser un composant sur la page.

Bien évidemment, les composants ne sont déposables uniquement dans les composants pouvant les contenir.

### 3.6.4. Présentation de l'explorateur de fichiers

L'explorateur de fichiers permet de naviguer à travers l'arborescence des fichiers et dossiers liés à NinaCMS ou aux autres CMS (ex : WordPress). Il se trouve à côté de l'explorateur de composants.

Pour fonctionner, l'explorateur de fichiers récupère une arborescence de dossiers de fichiers dans la base de données. Par contre, les données brutes d'un fichier ne sont pas directement stockées en base de données mais sur le disque dur de manière classique. Dans la base de données, on ne stocke donc que son chemin d'accès.



L'explorateur de fichiers est séparé en 3 parties :

1. Une barre de navigation, avec les boutons « **Précédent** », « **Suivant** », « **Parent** », « **Racine** », et « **Rafraîchir** », ainsi qu'une barre d'adresse, affichant où l'utilisateur se trouve dans l'arborescence,
2. Une barre rajoutant quelques boutons supplémentaires, comme la **création de dossier**, l'**envoi de fichiers** (l'interface est la même que celle du **FieldFile**), ou alors changer la disposition des fichiers, soit disposés en grille, soit disposés en ligne,
3. Et enfin la partie qui affiche les dossiers puis les fichiers.

Il est possible d'effectuer plusieurs actions sur les dossiers et les fichiers, comme la copie, le déplacement, le renommage, la suppression, etc. via des menus contextuels (clic droit).

**Création d'un dossier :**

// TODO : compléter cette partie

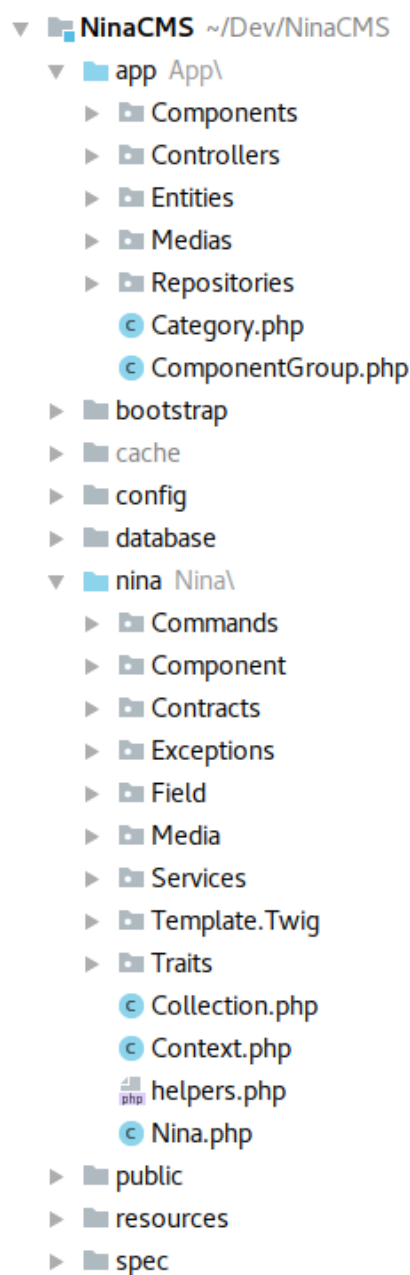
// + parler des actions du menu contextuel sur les dossiers

// + parler des actions du menu contextuel sur les fichiers

**3.6.5. Présentation des divers outils**

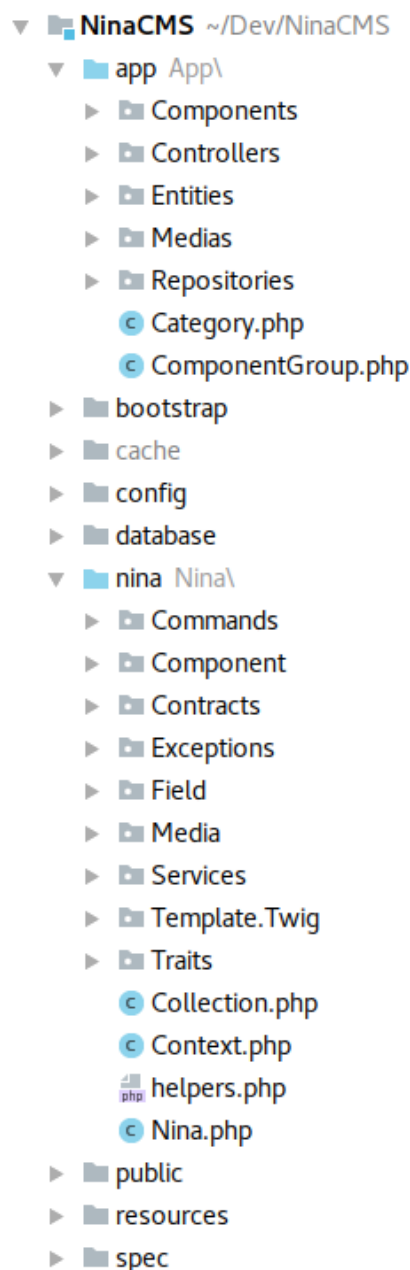
// TODO : compléter cette partie

### 3.1. Présentation de l'architecture du projet



L'architecture de NinaCMS reprend celle de certains frameworks PHP avec lesquels j'ai pu travailler, notamment Laravel et Symfony :

- Le dossier « **app** » est le dossier spécifique au client, on y trouve les **composants Nina** (ex : « **ArticleComponent** »), les **controllers** (retournent une réponse en fonction d'une requête HTTP, via un système de routes), les **entities** (représentent les tables en base de données), les **repositories** (séparent la couche d'accès au données de la couche métier), et deux classes contenant uniquement des constantes,
- Le dossier « **bootstrap** » contient les fichiers qui seront exécutés au démarrage de l'application, le dossier « **config** » contient des fichiers de configuration, le dossier « **database** » contient les différentes **migrations** (une sorte de contrôle de version de la base de données, permet de créer/supprimer des tables, ajouter/enlever des colonnes, ...) et les **seeders** (alimentent la base de données avec des données),
- Le dossier « **nina** », le cœur de NinaCMS, contient les bases fondamentales du projet :
  - Le dossier « **Commands** » contient des commandes pouvant être exécutées dans un invité de commandes,
  - Les dossiers « **Component** », « **Field** » et « **Media** » contiennent des classes de bases (« **BaseComponent** », « **BaseField** » et « **BaseMedia** ») qui sont héritées par les composants, champs, et médias situés dans le dossier « **app** »,
  - Le dossier « **Contracts** » contient des contrats, des interfaces PHP afin de garantir qu'une classe « remplisse son contrat »,
  - Le dossier « **Exceptions** » contient une grande liste des exceptions (classe PHP pour lancer une erreur) spécifiques à Nina,



- Le dossier « **Services** » contient des services, c'est-à-dire des **morceaux de code** qui ont pour but d'être **ré-utilisables** partout dans le projet, ou alors pour séparer ce code des **controllers** par soucis d'organisation et de clarté,
- Le dossier « **Template** » contient des abstractions pour les systèmes de **template** (modèle), uniquement **Twig** est supporté pour l'instant,
- Le dossier « **Traits** » contient différents **traits** PHP, ils sont utiles pour la composition de **classes PHP**, c'est-à-dire de ré-utiliser des fonctions et variables dans une classe sans pour autant faire d'héritage.
- Le dossier « **public** » contient les ressources publiques (les **assets**, c'est-à-dire des **styles CSS**, des **scripts JS**, et des **images minifiés et compressés**, le code de la **partie Administration**, et les fichiers de la Médiathèque). Ce dossier jouera le rôle de racine pour le serveur web **Apache** ou **nginx**, cela permet d'isoler le reste du projet, notamment des informations sensibles (comme le dossier « **config** »),
- Le dossier « **ressources** » contient lui aussi des ressources, mais privées. C'est dans ce dossier que se trouvent les fichiers **CSS**, **JS** et **images originaux** (sans avoir été minifiés et compressés). Le code de la **partie Administration** s'y trouve, quasiment entièrement composée de **composants VueJS** ! On y trouve aussi les différents **templates** Twig (ceux pour les **Field** et **Fragment**), ainsi que des fichiers de **traductions** (français et anglais) afin de traduire l'administration et les exceptions.
- Le dossier « **spec** » contient des tests unitaires, c'est-à-dire des tests permettant de tester .



## 4. Conclusion

### 4.1. Aspects techniques et expérience acquise

Je pense que la partie la plus complexe et la plus difficile était celle au tout début de l'alternance, lorsque j'ai du reprendre le **prototype** de NinaCMS.

En effet, ayant l'habitude de travailler avec des frameworks PHP objets, j'ai pour principe d'organiser et de documenter correctement mon code, de faire une architecture correcte (au niveau des dossiers du projets, de faire des tables en base de données correctes, de respecter un standard de code, etc.

Ici ce n'était pas le cas, ce qui est plutôt normal pour un **prototype**. Je pense que j'aurai du repartir de zéro tout en reprenant quelques aspects du prototype. Au final, tout au long de l'année j'ai pris de mon temps libre pour travailler sur ça. J'ai dépensé pas mal de temps et d'énergie à ré-organiser complètement le projet, mais j'en ai économisé des énormes quantités grâce à ça. Si je n'avais pas fais tout ce travail de ré-organisation, je ne sais honnêtement pas où est-ce que j'en serai, peut-être à 40 %, 30 % de l'avancement actuel ?

—

N'ayant jamais travaillé sur un projet aussi complexe, j'ai pu grandement améliorer mes compétences en PHP objet ainsi qu'en VueJS. Pour le PHP objet, j'ai notamment plus appliquer ce que j'avais appris pendant les cours de « PHP Objet » à l'IUT, en implémentant certains **patrons de conception** (« design pattern ») comme les **Repository**, **Factory/Builder**, **Provider**, l'**Injection de dépendances**, etc, dans NinaCMS.

Pour VueJS, j'ai pu l'utiliser d'une manière vraiment pas conventionnelle mais qui était requise pour la partie administration (vu qu'elle devait littéralement s'accrocher à la partie publique). Ici je devais initialiser moi-même les managers de composants et de colonnes et les injecter autour des composants et colonnes, chose qui n'est que très rarement (jamais) utilisé dans VueJS, mais ça montre que ça reste un framework très puissant malgré sa simplicité d'utilisation.

### 4.2. Perception et intégration dans l'entreprise

// ...



### 4.3. Formation de l'IUT

Je suis très reconnaissant des cours de « **PHP Objet** » donné par M. Boris Guery. Moi qui pensant déjà m'y connaître dans la programmation orientée objet (PHP ou en général), il a pu me montrer que non, et j'ai pu beaucoup apprendre grâce à lui.

Les cours « **Entreprise** » de M. Olivier Broaly étaient particulièrement intéressants, j'ai pu apprendre beaucoup de chose sur le fonctionnement des entreprises, de notre statut d'alternant ou d'employé, ... en peu de temps.

Enfin, il y a aussi eu les quelques séances « **Méthodes de travail** » dirigées par M. Clément Bouiller, expliquant les bonnes méthodes de travail à suivre pour un projet en entreprise, le backlog, etc, et c'est là que je me suis dit que je pouvais utiliser Trello (outil de gestion de projet) pour NinaCMS, même si je travaillai seul dessus. Un outil comme Trello apporte un système de colonne et de carte, et je peux savoir très facilement ce que j'ai déjà fais, ce que je fais, et ce que j'ai à faire.

Pour terminer, les autres cours ne m'ont pas servi (surtout J2E et ASP.NET qui ne devraient plus exister et donc être enseignés à notre époque, où ces technologies sont totalement dépréciées).



## 5. Glossaire

- **CMS** : Content Management System, un Système de Gestion de Contenu.
- **CSS** : Cascading StyleSheet, permet de définir le style visuel d'une page HTML (placement et taille des éléments, marges, couleurs, ...).
- **Framework** : Structure logicielle regroupant plusieurs composants afin de fournir des bases pour le développement d'un projet informatique – <https://fr.wikipedia.org/wiki/Framework>.
- **Git** : Logiciel de gestion de version – <https://git-scm.com>.
- **GitKraken** : Interface graphique dédiée à la gestion de projets versionnés grâce au programme git – <https://www.gitkraken.com>.
- **Gogs** : Service web auto-hébergé et de gestion de développement de logiciels, utilisant le logiciel de gestion de version Git – <https://gogs.io>.
- **HTML** : HyperText Markup Language, un format de données utilisant des balises pour décrire le contenu des pages web – <https://fr.wikipedia.org/wiki/HTML>.
- **JavaScript** : Langage de programmation de scripts initialement utilisé côté client sur des pages web dynamiques – <https://fr.wikipedia.org/wiki/JavaScript>.
- **JSDoc** : Programme permettant de générer une documentation en récupérant et en interprétant des commentaires dans du code JavaScript – <http://usejsdoc.org>.
- **JSON** : Format d'échange de données s'inspirant de JavaScript, notamment utilisé pour des communications réseau dû à sa syntaxe légère et épurée – <https://fr.wikipedia.org/wiki/JSON>.

- **Laravel** : Framework PHP suivant le principe du Modèle/View/Contrôleur, et permet au développeur de développer très rapidement un projet – <https://laravel.com>.
- **Node.js** : Environnement permettant d'exécuter du JavaScript côté serveur – <https://nodejs.org>.
- **PHP** : **PHP** **H**yperText **P**reprocessor, langage utilisé pour générer de l'HTML – <http://php.net>.
- **Sami** : Un générateur de documentation de code PHP – <https://github.com/FriendsOfPHP/Sami>.
- **SCSS (Sass)** : Pré-processeur CSS qui permet une écriture plus simple et plus organisé de code CSS, et qui sera ensuite compilé en CSS – <http://sass-lang.com>.
- **VueJS** : Framework JavaScript permettant de créer des interfaces web interactives en utilisant le principe de composants webs – <https://vuejs.org>.
- **Webpack** : Empaqueur de code JavaScript reposant sur le principe des modules JavaScript – <https://webpack.js.org>.

## 6. Bibliographie

- <http://www.sailing-up.com>
- <https://developer.mozilla.org/en-US>
- <https://vuejs.org>
- <http://www.php.net>
- <https://laravel.com>
- <https://github.com>



## 7. Annexes

### 7.1. FieldFile

Avec le code PHP suivant, nous avons créé une interface de sélections de fichiers **HTML** et de fichiers **images**, ayant une taille inférieure à **4 Mo (4096 Ko)**.

```
protected $fieldsRules = [
    'list_of_files' => 'array|nullable',
];

protected $fieldsFileRules = [
    'list_of_files' => 'file|mime:image,html|max:4096',
];

public function afterConstruct()
{
    $this->fields = [
        'foo_bar' => [
            'name' => 'FieldFile',
            'fields' => [
                new FieldFile( id: 'list_of_files', label: 'Envoi de fichiers', ['image', 'html'], maxSize: 4096),
            ]
        ]
    ];
}
```

*Code PHP*

Edition du composant « Nina\_Test\_629 »

FieldFile

Envoi de fichiers (Taille maximale : 4.00 Mo) (Fichiers acceptés : .jpg, .jpeg, .png, .gif, .html)

Ajouter un fichier Envoyer tous les fichiers Annuler tous les envois Supprimer tous les fichiers

File d'attente

La file d'attente est vide.

Fichiers liés au composant

Aucun fichier n'est lié avec ce champ.

Annuler et fermer Modifier

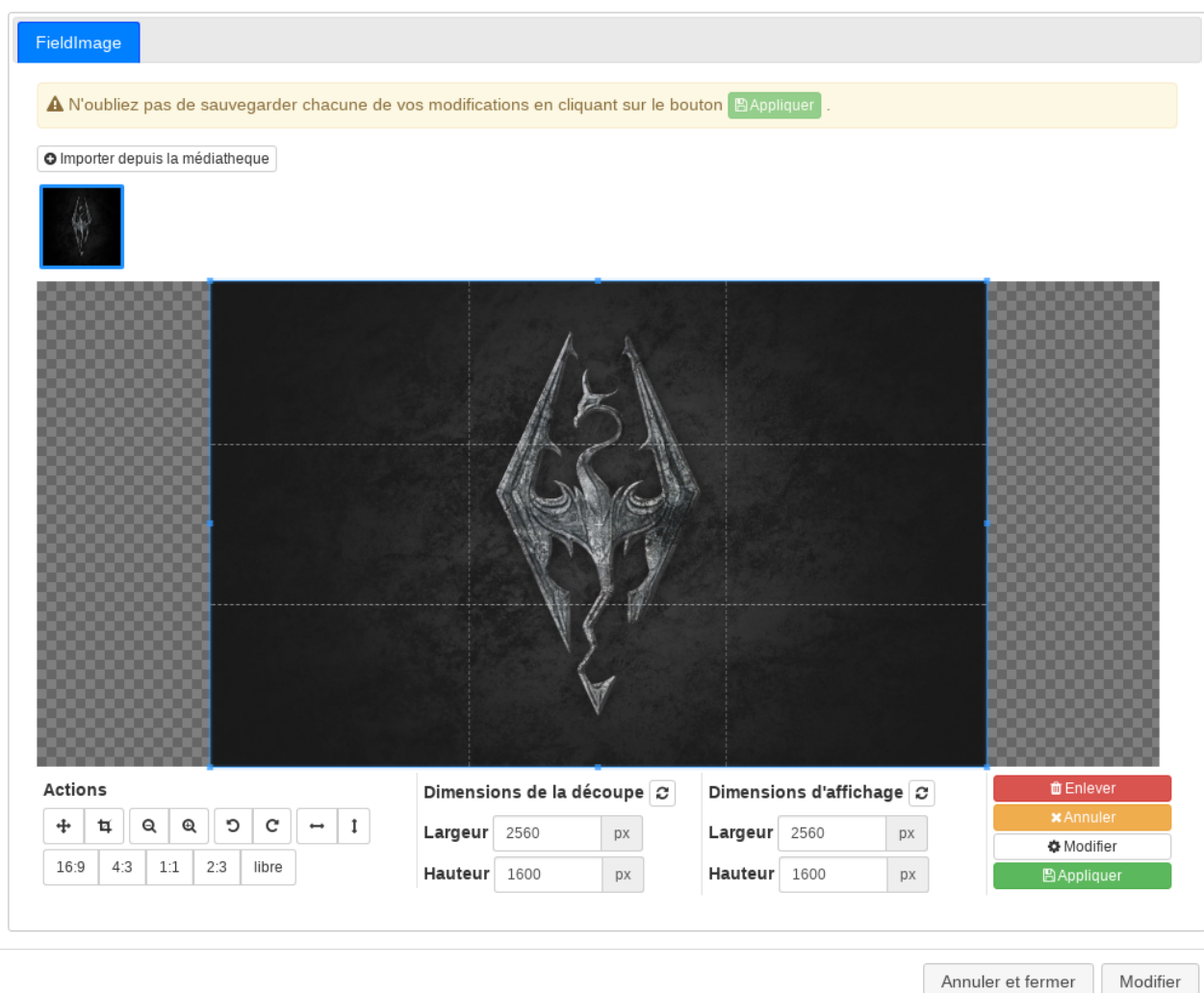
*Rendu dans l'éditeur de composant*

## 7.2. FieldImage

Avec le code PHP suivant, nous avons créé une interface de sélection et de modification d'images. En ouvrant l'Explorateur de Fichiers, **seules les images** s'afficheront afin de faciliter la sélection. Il est possible de limiter la sélection de fichiers via des options.

```
public function afterConstruct()
{
    $this->fields = [
        'foo_bar' => [
            'name' => 'FieldImage',
            'fields' => [
                new FieldImage( id: 'image')
            ]
        ],
    ];
}
```

*Code PHP*



*Rendu dans l'éditeur de composant*



## 7.3. FieldLinks

Interface pour gérer une liste de liens. Il est possible de limiter le nombre de liens via les options.

```
public function afterConstruct()
{
    $this->fields = [
        'foo_bar' => [
            'name' => 'FieldLinks',
            'fields' => [
                new FieldLinks(id: 'links', label: 'Une liste de liens')
            ]
        ],
    ];
}
```

*Code PHP*

**FieldLinks**

**Une liste de liens**

Adresse URL	Label	Options
<input type="text" value="https://"/>	<input type="text" value="Lien vers ..."/>	<input checked="" type="checkbox"/> Ouvrir dans une nouvelle fenêtre
<input type="text" value="https://"/>	<input type="text" value="Lien vers ..."/>	<input checked="" type="checkbox"/> Ouvrir dans une nouvelle fenêtre
<input type="text" value="https://"/>	<input type="text" value="Lien vers ..."/>	<input checked="" type="checkbox"/> Ouvrir dans une nouvelle fenêtre

Ajouter un lien

*Rendu dans l'éditeur de composant*

## 7.4. Partie publique

### Il se forge une réputation de génie après avoir réussi un tableau croisé dynamique

Titre par défaut

Chapeau par défaut



L'entreprise Sofinco est encore sous le choc. Hier matin, Malo, le nouveau contrôleur de gestion de la petite entreprise du Jura, a réussi un exploit rare en réalisant un tableau croisé dynamique devant tous les employés de l'open space réunis. Une prouesse historique qui en a immédiatement fait la coqueluche de l'entreprise. Récit.

C'est une histoire de bizutage qui aurait pu mal tourner. Ce mardi, alors qu'il s'apprête à partir déjeuner, Malo est hélé par Jean-Patrice et Benjamin, deux collègues de la comptabilité qui décident de « jouer un tour au nouveau » comme le veut la coutume de cette vieille PME jurassienne. « On voulait juste lui faire un peu peur », raconte Jean-Patrice, « Lui montrer qui faisait la loi ici ». « Alors, comme d'habitude, on lui a demandé l'impossible », ajoute Benjamin. « Un tableau croisé dynamique ».

« Venez voir, le nouveau va nous faire un TCD ! »

Lire la suite sur [Le Gorafi](#).

## 7.5. Partie administration

### Il se forge une réputation de génie après avoir réussi un tableau croisé dynamique



L'entreprise Sofinco est encore sous le choc. Hier matin, Malo, le nouveau contrôleur de gestion de la petite entreprise du Jura, a réussi un exploit rare en réalisant un tableau croisé dynamique devant tous les employés de l'open space réunis. Une prouesse historique qui en a immédiatement fait la coqueluche de l'entreprise. Récit.

C'est une histoire de bizutage qui aurait pu mal tourner. Ce mardi, alors qu'il s'apprête à partir déjeuner, Malo est hélé par Jean-Patrice et Benjamin, deux collègues de la comptabilité qui décident de « *jouer un tour au nouveau* » comme le veut la coutume de cette vieille PME jurassienne. « *On voulait juste lui faire un peu peur* », raconte Jean-Patrice, « *Lui montrer qui faisait la loi ici* ». « *Alors, comme d'habitude, on lui a demandé l'impossible* », ajoute Benjamin. « *Un tableau croisé dynamique* ».

« Venez voir, le nouveau va nous faire un TCD ! »

Lire la suite sur [Le Gorafi](#).

Titre par défaut

Chapeau par défaut

Contenu par défaut.

Créé le 1 janv. 2017

NinaCMS

Résumé en français :

Mots clés :

Éditeur visuel de site web, WYSIWYG, interactivité, CMS, Composants, drag & drop, explorateur de fichiers, AJAX, JSON, PHP, Laravel, Twig, Node.js, Vue, Vuex, Webpack, Gulp, ESLint, git.

Matériel / logiciels / méthodes utilisé(e)s :

- Ordinateur de l'entreprise configuré sous Ubuntu, deux écrans, une souris et un clavier,
- PhpStorm, Apache, PHP, MySQL, Node.js, git, GitKraken, Google Chrome, Mozilla Firefox, Opera,
- Méthode agile avec pour client mon maître d'apprentissage, car je développais un produit pour le compte de l'entreprise.