

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

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.....	15
3.4. Partie publique.....	15
3.5. Partie administration.....	15
3.1. Présentation de l'architecture du projet.....	17
4. Conclusion.....	20
4.1. Expérience acquise.....	20
4.2. Avenir du projet.....	20
5. Glossaire.....	22
6. Bibliographie.....	24

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 : Ubuntu 16.04
	Environnement de Développement Intégré (EDI) : PhpStorm (2017.2)
	Navigateurs Web : Google Chrome (XX), Mozilla Firefox (XX), et Opera (XX)
	Traitement de texte : LibreOffice Writer (5.2)

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

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

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

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

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.

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.

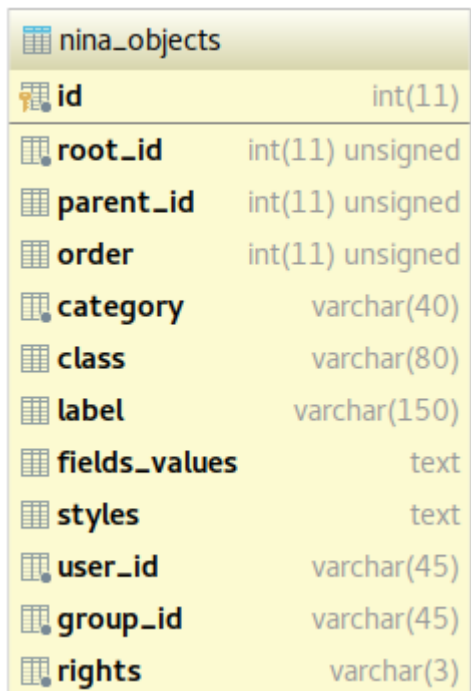
Un composant peut ou non être composé de champs, appelés **Field**. Je présenterai les **Field** plus tard, mais concrètement, un **Field** a un certain type (texte, image, adresse URL, etc ...), et il est utilisé dans la partie administration pour pouvoir customiser certaines valeurs du composant. Par 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^{er} **Field** au format **HTML**, et la valeur du 2^{ème} **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	
id	int(11)
root_id	int(11) unsigned
parent_id	int(11) unsigned
order	int(11) unsigned
category	varchar(40)
class	varchar(80)
label	varchar(150)
fields_values	text
styles	text
user_id	varchar(45)
group_id	varchar(45)
rights	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 ≥ 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 ≥ 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

- f ? fields:Nina\Field\BaseField[]
- f ? fieldsRules:array
- f ? fieldsFileRules:array
- f ? contains:array
- f ? icon:string
- f ? template:null|string
- f ? styles:array|null
- f ? scripts:array|null
- f ? adminStyles:array|null
- f ? adminScripts:array|null
- f ? 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 ☰ ☷ ☹ ☶ 🔗

p

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^{er} 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:

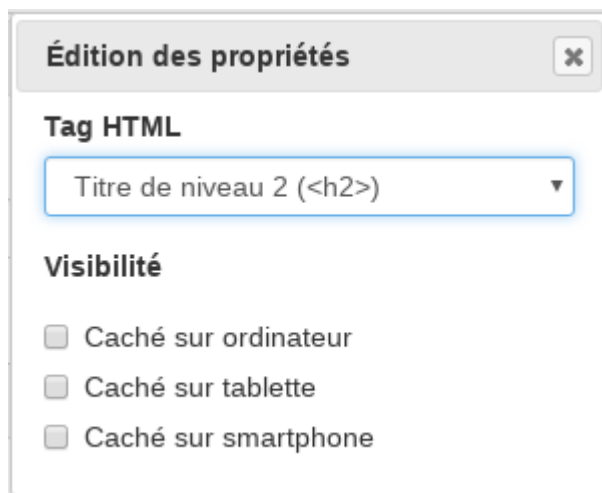
↶ ↷ Formats ▾ **B** *I* U [Liste à puces] [Liste à puces] [Liste à puces]

La configuration des Property se fait en cliquant sur ⚙

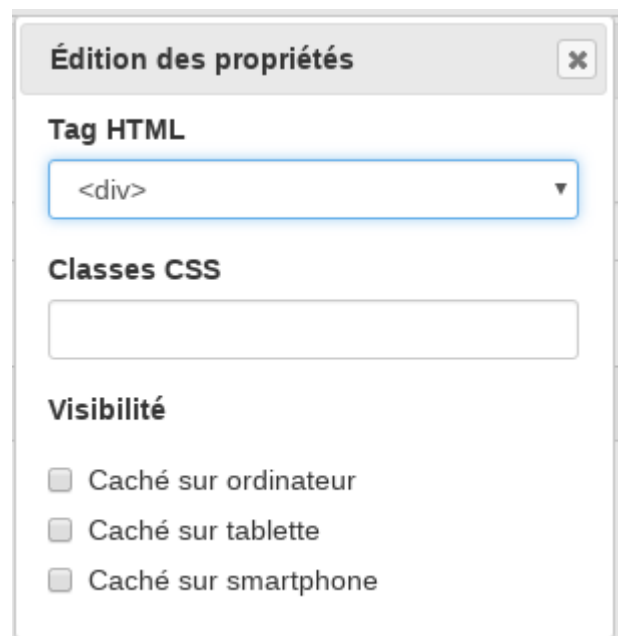
Voici comment se définissent les Property sur un Field. La méthode « **addCommonProperties()** » permet de rajouter les « Property Tag », « Classes CSS » et « Visibilité ».

```
$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 ». 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 ∞ , 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

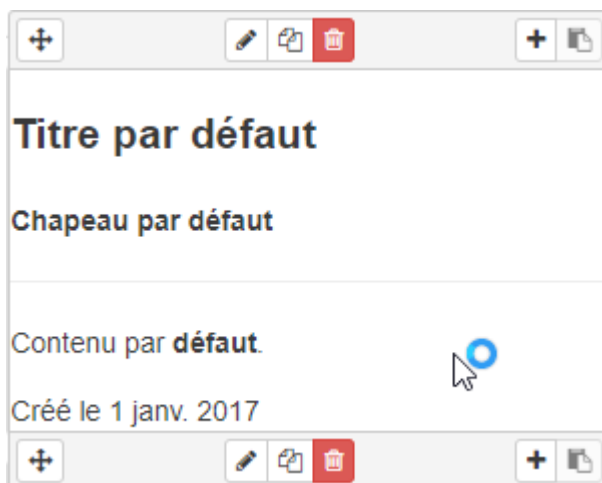
La partie administration a été réalisée entièrement avec VueJS, un framework que j'apprécie très particulièrement pour sa facilité d'apprentissage et d'utilisation, mais qui reste tout de même puissant.

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** »

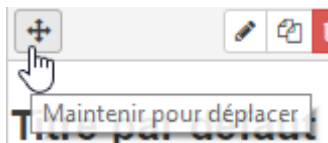
3.6.1. Présentation du manager de composant (ObjectManager)

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.

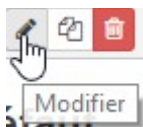


Le composant est maintenant déplaçable dans tous les composants qui peuvent contenir le composant déplacé. (ex : le composant « Colonne » peut contenir le composant « Article »)

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 :

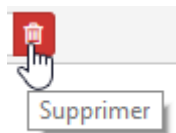


Les boutons coller avant/après sont donc activés. Cliquer sur ce bouton duplique le composant à copier, et le colle au dessus ou en dessous,

Le composant a bien été collé !

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

Suppression d'un composant :



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.

Le composant a bien été supprimé !

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

























3.6.2. Présentation du manager des composants colonnes (BlocManager)

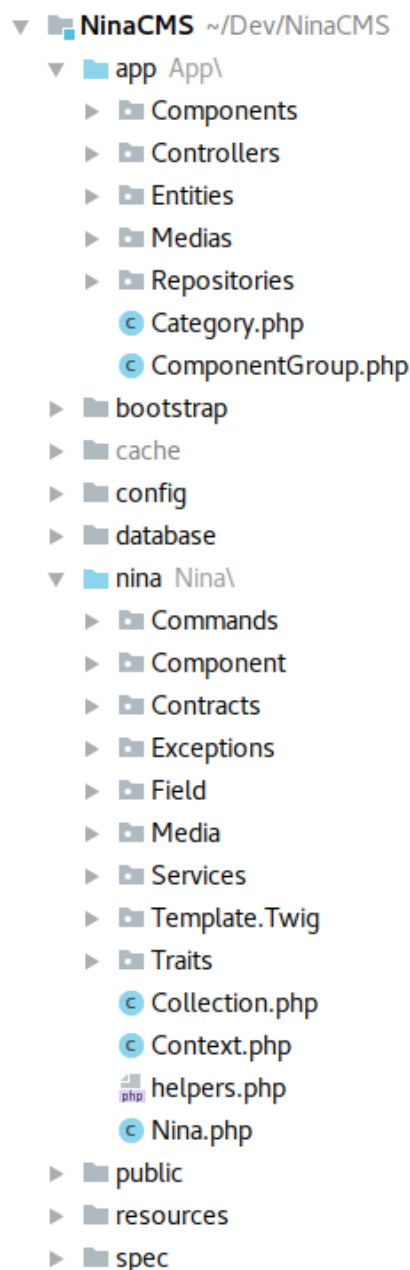
3.6.3. Présentation de l'explorateur de composants

3.6.4. Présentation de l'explorateur de fichiers

3.6.5. Présentation des divers outils

3.1. Présentation de l'architecture du projet

- ▼  **NinaCMS** ~/Dev/NinaCMS L'architecture de NinaCMS reprend celle de certains frameworks PHP avec lesquels j'ai pu travailler, notamment Laravel et Symfony :
- ▼  **app** App\
 - ▶  Components
 - ▶  Controllers
 - ▶  Entities
 - ▶  Medias
 - ▶  Repositories
 - Category.php
 - ComponentGroup.php
 - ▶  bootstrap
 - ▶  cache
 - ▶  config
 - ▶  database
 - ▼  **nina** Nina\
 - ▶  Commands
 - ▶  Component
 - ▶  Contracts
 - ▶  Exceptions
 - ▶  Field
 - ▶  Media
 - ▶  Services
 - ▶  Template.Twig
 - ▶  Traits
 - Collection.php
 - Context.php
 - helpers.php
 - Nina.php
 - ▶  public
 - ▶  resources
 - ▶  spec
 - 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. Expérience acquise

4.2. Avenir du projet

5. Glossaire

→ **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/Vue/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 organisée 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

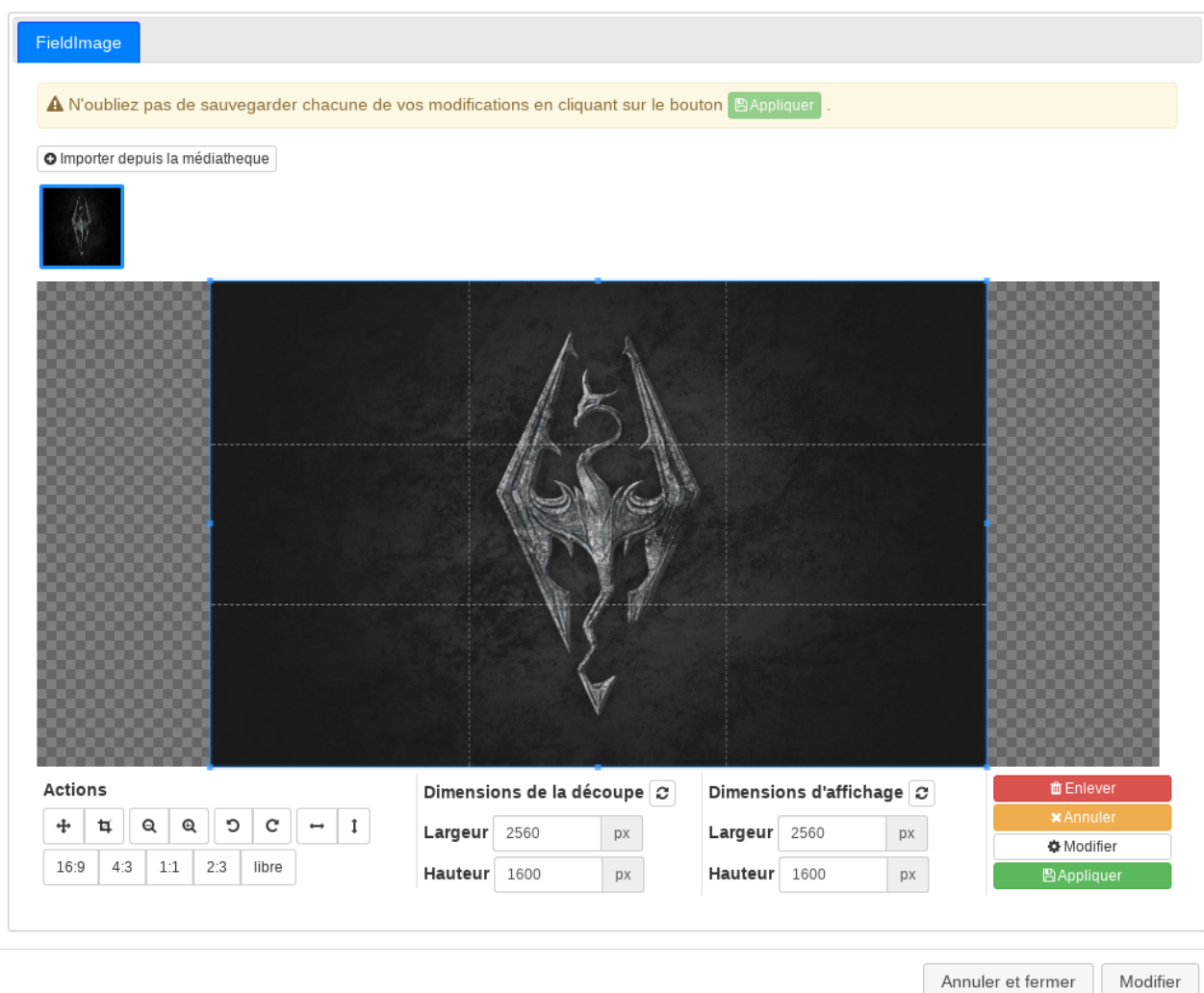
Rendu dans l'éditeur de composant

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.